Vežba 4 - Organizacija projekata sa više izvornih datoteka

U svim dosadašnjim primerima, razvijane su relativno jednostavnije aplikacije koje su realizovane pisanjem izvornog koda koji je smešten u jednu .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. Izvorni kod napisan u jeziku C smešta se u dva tipa datoteka:

- **.h datoteke** sadrže deklaracije (prototipove) funkcija, makroe, 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 projekta koji sadrži ukupno 5 izvornih datoteka:

- **glavni.c** (sadrži funkciju main)
- **sabiranje.h**
- **sabiranje.c**
- **oduzimanje.h**
- **oduzimanje.c**

glavni.c:

```c
#include "sabiranje.h"
#include "oduzimanje.h"

void main(void)
{
    int a;
    a  = saberi(1, oduzmi(4, 2));

    while(1);
}
```

sabiranje.h:

```c
int saberi(int a, int b);
```

sabiranje.c:

```c
#include "sabiranje.h"

int saberi(int a, int b)
{
    return (a + b);
}
```

oduzimanje.h:

```c
int oduzmi(int a, int b);
```

oduzimanje.c:

```c
#include "oduzimanje.h"

int oduzmi(int a, int b)
{
    return (a - b);
}
```

Razvojni put projekta koji se sastoji od više biblioteka biće ilustrovan na primeru programske podrške upravljanju pinovima i tajmerom 0. Na kraju ovog dokumenta se nalazi kod programa koji implementira naizmenično treptanje diode različitim brzinama. Kod je napisan unutar jedne datoteke, gde su definisani svi makroi, promenljive i funkcije koje su upotrebljene prilikom realizacije programa. Kako se broj makroa, promenljivih, funkcija i komentara koji ih opisuju povećava, kod postaje sve nepregledniji i neorganizovaniji. 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:

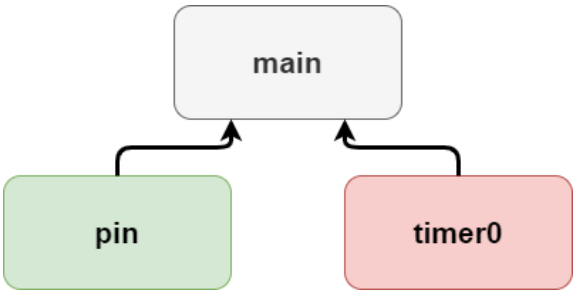
- pinPulsing
- pinPulse
- pinSetValue
- pinInit
- timer0DelayMs
- timer0InteruptInit
- ISR(TIMER0_COMPA_vect)
- calculateHalfPeriod

i makroa:

- HIGH
- LOW
- OUTPUT
- INPUT
- PORT_B
- PORT_C
- PORT_D
- DIODE_PIN
- FAST
- SLOW
- FAST_REPETITIONS
- SLOW_REPETITIONS

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 u dve datoteke, to su zaglavlje **pin.h** i izvorna datoteka **pin.c**. Za inicijalizaciju pinova, koristi se funkcija *pinInit*, dok se za osnovu manipulaciju pinom koristi funkcija *pinSetValue*. Stoga je potrebno realizovati ove dve funkcije unutar biblioteke **pin**, gde je takođe potrebno definisati odgovarajuće vrednosti upotrebom makroa. Makroi koji se koriste prilikom realizacije i upotrebe ovih funkcija su *HIGH*, *LOW*, *OUTPUT*, *INPUT*, *PORT_B*, *PORT_C* i *PORT_D*.
Prilikom implementacije biblioteke za manipulaciju tajmerom 0, **timer0**, kreiramo zaglavlje **timer0.h** i izvornu datoteku **timer0.c**. Ova biblioteka će kao svoj sadržaj posedovati funkciju za inicijalizaciju tajmera *timer0Init*, funkciju za implementaciju pauze *timer0DelayMs* i prekidnu rutinu *ISR(TIMER0_COMPA_vect)*. Kako se za implementaciju i izvršavanje ovih funkcija ne upotrebljava ni jedan makro, ova biblioteka neće posedovati ni jednu definiciju konstante pomoću makroa. Međutim, suština ove biblioteke se zasniva na promenljivoj *volatile unsigned long ms*, čija se uloga ogleda u skladištenju broja milisekundi proteklih od pokretanja programa. Prilikom definisanja ove promenljive, možemo se voditi pomoću dve različite ideje. 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

skrivenosti promenljive, gde će ona biti dostupna samo funkcijama ove biblioteke, čime se sprečava mogućnost ostatka programa da pristupi ili promeni vrednost programskog vremena. Prilikom razvoja ove biblioteke, vodićemo se drugom idejom, gde ćemo promenljivu *ms* definisati tako da bude vidljiva samo od strane biblioteke **timer0**. Ovo postižemo tako što promenljivu definišemo u izvornoj datoteci **timer0.c**. Ukoliko smo promenljivu zasnovali na prvoj ideji, ona bi bila definisana unutar zaglavlja.
Nakon kreiranja ove dve biblioteke, hijerarhija projekta se zasniva na uključivanju biblioteka **pin** i **timer0** u **main** datoteku.



Nakon kreiranja ovih biblioteka, u main datoteci preostaju sledeće funkcije i makroi.
Datoteka se sastoji od sledećih funkcija:

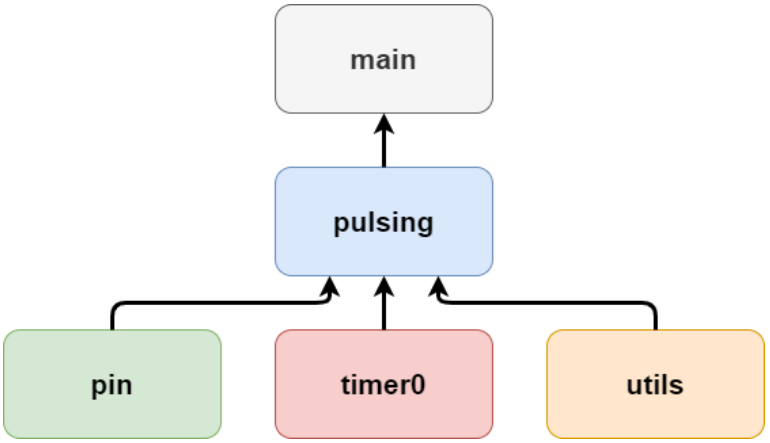
- pinPulsing
- pinPulse
- calculateHalfPeriod

i makroa:

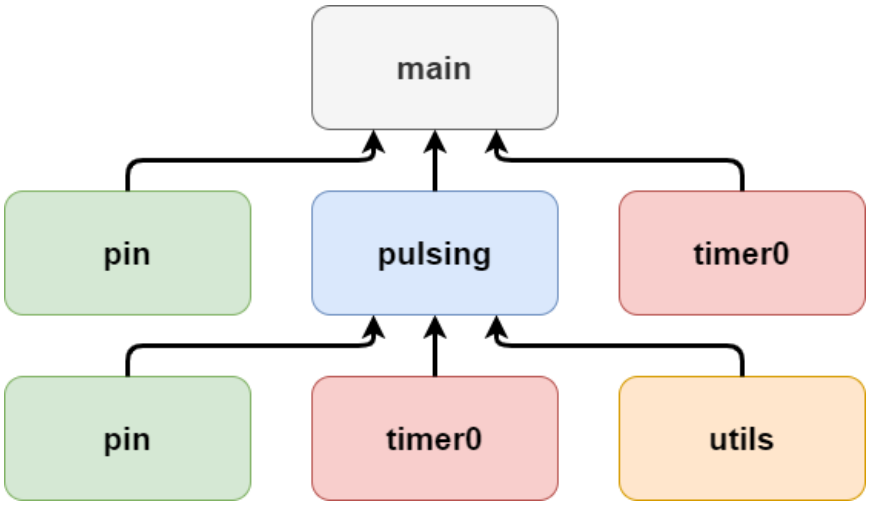
- DIODE_PIN
- FAST
- SLOW
- FAST_REPETITIONS
- SLOW_REPETITIONS

Ukoliko uporedimo preostale funkcije, možemo primetiti da funkcija *calculateHalfPeriod* ne upravlja ni 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 postoji veliki broj ovih funkcija, obično se one mogu organizovati u neke logički smislene pomoćne (eng. *utility)* biblioteke. U ovom projektu kreiraćemo jednu pomoćnu biblioteku koju ćemo nazvati **utils**. Njen sadržaj će činiti upravo ova funkcija.
Dve funkcije koje kreiraju pulsiranje pina, odnosno treptanje 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**, nego i funkcije iz biblioteka **timer0** i **utils**. 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 da ukoliko želimo da uključimo biblioteku za upravljanje pinovima, automatski uključujemo i biblioteku za upravljanje tajmerom 0 i skup pomoćnih funkcija. Iz ovog razloga, praktikuje se kreiranje nove biblioteke koja će biti uključena ukoliko postoji potreba za upotrebu ovih funkcija. Ovu biblioteku nazivamo **pulsing**, i u njoj implementiramo funkcije *pinPulse* i *pinPulsing*. Biblioteka neće sadržati makroe, niti dodatne promenljive.

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



Ukoliko na ovaj način organizujemo projekat, sve funkcije iz bibilioteke **pulsing** će biti uključene u **main**, čime možemo realizovati treptanje. Međutim, moramo 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 timer 0. Kako su ove dve funkcije implementirane u bibiliotekama **pin** i **utils** respektivno, potrebno je ove dve biblioteke takođe uključiti u **main**. Nakon ovih izmena, organizacija projekta izgleda kao na sledećoj slici.



Include zaštita

#include zaštita, ili kako se nekad naziva zaštita zaglavlja, predstavlja konstrukciju koja se koristiti kako bi se izbegao problem dvostrukog uključivanja prilikom korišćenja *include* preprocesorske direktive. Stoga je potrebno unutar zaglavlja svake biblioteke koja je implementirana postaviti *include* zaštitu na sledeći način:

```
#ifndef MY_HEADER_H
#define MY_HEADER_H

...
code
...

#endif /* MY_HEADER_H */
```

Kao što se može videti, prvo je potrebno proveriti da li je makro `MY_HEADER_H` definisan, korišćenjem preprocesorske direktive **#ifndef** (if-not-defined). Ukoliko nije, to će biti učinjeno i svaki naredni put, kada ova biblioteka bude uključena, deo koda koji obuhvata ova preprocesorska direktiva biće preskočen, budući da je ovaj makro već prethodno bio definisan, što znači da su sve deklaracije već napravljenje. Na ovaj način je izbegnuta pojava dvostrukog uključivanja.
Primer koji demonstrira pojavu dvostrukog uključivanja je dat u nastavku.

Datoteka **grandparent.h**

```
struct s{
    int member;
};
```

Datoteka **parent.h**

```
#include "grandparent.h"
```

Datoteka **child.h**

```
#include "grandparent.h"
#include "parent.h"
```

Ovo će rezultovati u dvostrukoj kopiji sadržaja datoteke **grandparent.h**, što će izazvati grešku prilikom kompilacije koda, budući da je struktura **s** definisana dva puta. Prema tome, ovo je potrebno rešiti korišćenjem *include* zaštite, na sledeći način:

```
#ifndef GRANDPARENT_H
#define GRANDPARENT_H

struct s{
    int member;
};

#endif /* GRANDPARENT_H */
```

Na ovaj način, dvostruko uključivanje biće izbegnuto.

```c
/**
 * @file main.c
 * @brief Aplikacija koja implementira ogranicen broj treptanja diode
 * @author Ime Prezime
 * @date 14-03-2021
 * @version 1.0
 */
#include <avr/io.h>
#include <avr/interrupt.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_REPETITIONS 15
/// Makro za podesavanje broja sporih treptaja
#define SLOW_REPETITIONS 3

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

/**
  * pinPulsing - Funkcija koja implementira num_of_repetitions perioda podizanja i spustanja vrednosti na pinu
 * odgovarajucom brzinom
  * @param port - ulaz tipa unsigned char - Port na kojem je potrebno implementirati funkcionalnost
  * @param pin - ulaz tipa unsigned char - Pin na kojem je potrebno implementirati funkcionalnost
```

```
 * @param period - ulaz tipa unsigned long - Perioda promene vrednosti na
pinu
 * @param num_of_repetitions - ulaz tipa unsigned char - Broj perioda koje je
potrebno implementirati
 * @return Nema povratnu vrednost
 */
void pinPulsing(unsigned char port, unsigned char pin, unsigned long period,
unsigned char num_of_repetitions);

/**
 * pinPulse - Funkcija koja implementiran podizanje i spustanje vrednosti na
pinu
 * odgovarajucom brzinom
  * @param port - ulaz tipa unsigned char - Port na kojem je potrebno
implementirati funkcionalnost
  * @param pin - ulaz tipa unsigned char - Pin na kojem je potrebno
implementirati funkcionalnost
 * @param period - ulaz tipa unsigned long - Perioda promene vrednosti na
pinu
 * @return Nema povratnu vrednost
 */
void pinPulse(unsigned char port, unsigned char pin, unsigned long period);

/**
 * pinSetValue - Funkcija koja postavlja vrednost na pinu
 * @param port - ulaz tipa unsigned char - Port na kojem je pin ciju vrednost
potrebno postaviti
  * @param pin - ulaz tipa unsigned char - Pin ciju je vrednost potrebno
postaviti
  * @param value - ulaz tipa unsigned char - Vrednost koju je potrebno
postaviti na pin
 * @return Nema povratnu vrednost
 */
void pinSetValue(unsigned char port, unsigned char pin, unsigned char value);

/**
 * pinInit - Funkcija koja implementiran inicijalizaciju pina
  * @param port - ulaz tipa unsigned char - Port na kojem je pin koji je
potrebno inicijalizovati
   * @param pin - ulaz tipa unsigned char - Pin koji je potrebno
inicijalizovati
 * @param direction - ulaz tipa unsigned char - Smer prema kojem je potrebno
inicijalizovati pin
 * @return Nema povratnu vrednost
 */
void pinInit(unsigned char port, unsigned char pin, unsigned char direction);

/**
 * timer0DelayMs - Funkcija koja implementira pauzu u broju milisekundi koji
je prosledjen
 * kao parametar
  * @param delay_length - ulaz tipa unsigned long - Duzina pauze u
milisekundama
```

```
 * @return Povratna vrednost je tipa unsigned long i ima vrednost broja milisekundi
 * proteklih od pocetka aplikacije do trenutka izlaska iz funkcije
 */
unsigned long timer0DelayMs(unsigned long delay_length);

/**
 * timer0InteruptInit - Funkcija koja inicijalizuje timer 0 tako da pravi prekide
 * svake milisekunde
 * @return Nema povratnu vrednost
 */
void timer0InteruptInit();

/**
 * calculateHalfPeriod - Funkcija koja izracunava polovinu prosledjene periode
 * @param period - input tipa unsigned long - Duzina periode
 * @return Povratna vrednost je tipa unsigned long i predstavlja polovinu periode
 */
unsigned long calculateHalfPeriod(unsigned long period) ;

/**
 * main - funkcija koja implementiran glavni deo aplikacije
 * @return Nema povratnu vrednost
 */
int main()
{
    unsigned long period = 1000;    // Period jednog treptaja
    unsigned char repetitions = 5;  // Broj treptaja

    // Inicijalizacija
    pinInit(PORT_B, DIODE_PIN, OUTPUT);
    timer0InteruptInit();

    // Glavna petlja
    while (1)
    {
        // Brzo treptanje
        pinPulsing(PORT_B, DIODE_PIN, FAST, FAST_REPETITIONS);

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

        // Kraj
        while (1)
        ;

    }

    return 0;
}
```

```
/*************************************************************************
**************/

void pinPulsing(unsigned char port, unsigned char pin, unsigned long period,
unsigned char num_of_repetitions)
{
    unsigned char i;

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

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

void pinPulse(unsigned char port, unsigned char pin, unsigned long 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(unsigned char port, unsigned char pin, unsigned char 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(unsigned char port, unsigned char pin, unsigned char 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;
    }

}

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

unsigned long timer0DelayMs(unsigned long delay_length)
{
    unsigned long t0; // Trenutak pocevsi od kog se racuna pauza

    // Implementacija pauze
    t0 = ms;
    while ((ms - t0) < delay_length)
    ;  // Pauza delay_length milisekundi

    return ms;
}
```

```
/*************************************************************************
**************/

void timer0InteruptInit()
{
    // Inicijalizacija tajmera 0 tako da perioda prekida bude 1ms
    TCCR0A = 0x02;
    TCCR0B = 0x03;
    OCR0A = 249;
    TIMSK0 = 0x02;

    // Podesavanje globalne dozvole prekida
    sei();
}

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

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

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

unsigned long calculateHalfPeriod(unsigned long period)
{
    return (period/2);
}
```