УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs
ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

PROJEKAT IZ PREDMETA
M2M ELEKTRONSKI SISTEMI

NAZIV PROJEKTA:

Igra memorije i rad sa GSM Click 3 modulom

TEKST ZADATKA:

Prvi deo zadatka obuhvata implementaciju igre memorije na TFT ekranu, uključujući praćenje proteklog vremena i rangiranje pet najuspešnijih rezultata. Drugi deo zadatka se bavi radom sa GSM Click 3 modulom. Kada modul detektuje poruku sa sadržajem "Pozovi me!", program automatski poziva broj sa kojeg je poruka primljena.

MENTORI PROJEKTA:

Vanr. Prof. dr Vladimir Rajs
dr Branislav Batinić

PROJEKAT IZRADILI:

Vanja Lazarević E1 77/2023
Marko Draškić E1 103/2023
Stefan Stanišić E1 50/2023

DATUM ODBRANE:

10.07.2024

Sadržaj

1. Uvod

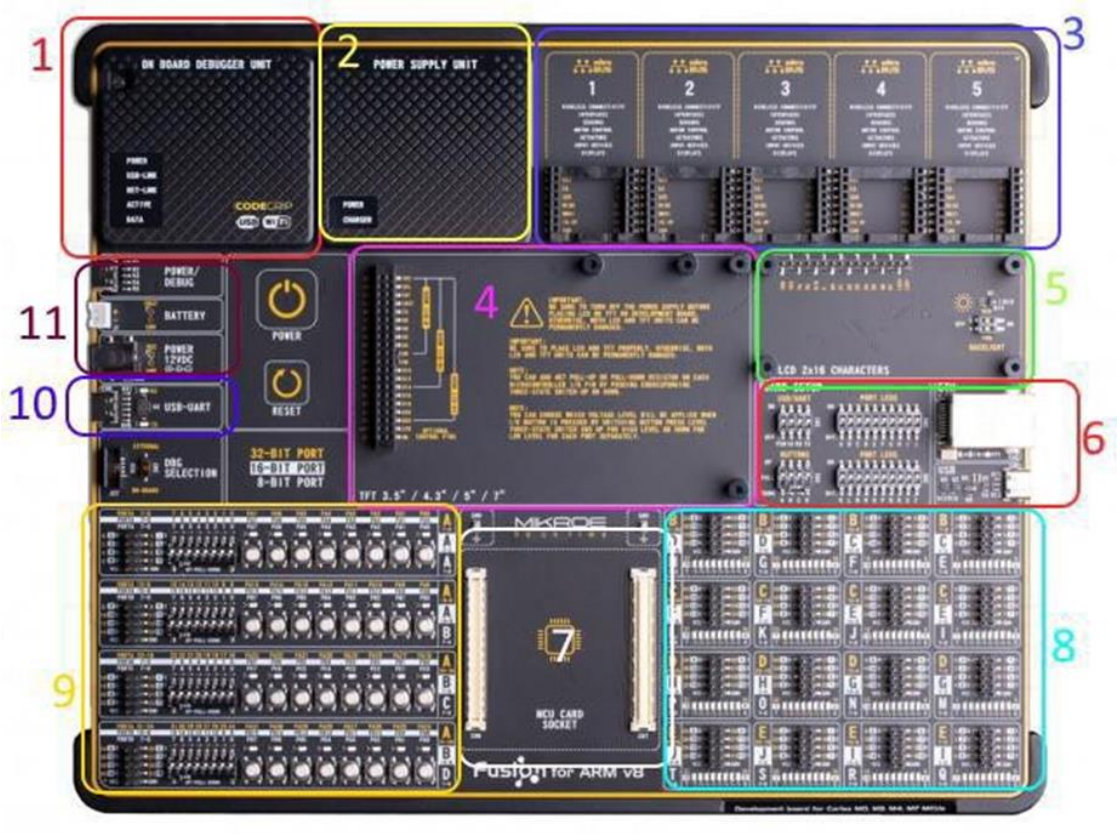
Projektni zadatak je podeljen u dve odvojene zasebne celine, tako da prva celina predstavlja rad sa *TFT* ekranom i realizaciju društvene igre pod nazivom “Igra memorije”, dok druga celina podrazumeva testiranje *GSM Click* 3 modula. Da bi oba dela bila uspešno realizovana, korišćen je razvojni sistem *Fusion Board for ARM v8* kompanije “Mikroelektronika”. Programska implementacija je izvedena u okruženju mikroC pro for *ARM*, dok je za programiranje kontrolera korišćen *CODEGRIP Suite*, takođe razvijen od strane iste spomenute kompanije.

Prvi deo zadatka predstavlja realizaciju igre pod nazivom “Igra memorije”. Na početnom ekranu nalaze se 16 tastera ispod kojih se nalaze različite kartice(slike), zatim taster za pokretanje igre, polja za upisivanje proteklog vremena, kao i taster za prikazivanje tih rezultata. Pritiskom na taster NOVA IGRA, korisnik započinje sa svojom igrom i uključuje se štoperica u pozadini. Igra se završava tek kada se pronađu sve identične karterice, kojih ima osam. Ukoliko kartice nisu iste, one se vraćaju u početno stanje, odnosno na poleđinu. Pomoću tastera REZULTATI, stvara se prikaz pet sortiranih najuspešnijih vremena. Broj odigranih partija je ograničen na deset.

Drugi deo zadatka predstavlja rad sa *GSM Click* 3 modulom i njegovo testiranje. Glavna funkcija ovakvog modula jeste čekanje poruka sa drugog mobilnog uređaja. Kada pristigne poruka sa tekstom "Pozovi me!", modul automatski poziva broj sa kojeg je poruka poslata. Iz tog razloga potrebno je da modul ima ubačenu aktivnu SIM karticu. Pored *GSM* modula, koristi se i *OLED* ekran, za lakše praćenje stanja i funkcionalnosti ovog modula.

2. Razvojni sistem Fusion Board for ARM v8

Fusion Board for ARM v8 je razvojna ploča namenjena za rad sa *ARM* mikrokontrolerima. Ovaj sistem pruža širok spektar I/O pinova, omogućavajući jednostavno povezivanje sa raznovrsnim *Click* modulima. Napajanje ploče može se vršiti putem *USB-C* kabla, Li-Ion baterije ili eksternog izvora napajanja sa maksimalnim naponom do 12V DC. Ova fleksibilnost čini ploču idealnom za različite projektne zahteve i aplikacije, pružajući stabilnu i pouzdanu platformu za korisnike. Na slici 2.1 nalazi se prikaz korišćenog razvojnog sistema.



Slika 2.1 Prikaz razvojnog sistema

Pomenuti razvojni sistemi, sastoji iz nekoliko delova obeleženih na slici:

1. *CodeGrip* Programator
2. Deo za napajanje razvojnog sistema
3. *MikroBus* kućišta za smeštanje click pločica
4. Kućište za smeštanje kapacitivnog *TFT* ekrana
5. Kućište za smeštanje 2x16 *LCD* ekrana
6. *Ethernet* i *USB* komunikacija
7. Mesto za povezivanje mikrokontrolera
8. Izvučeni pinovi mikrokontrolera

9. Tasteri i prekidači
10. Priključak za serijsku komunikaciju
11. Konektori za različite vrste napajanja

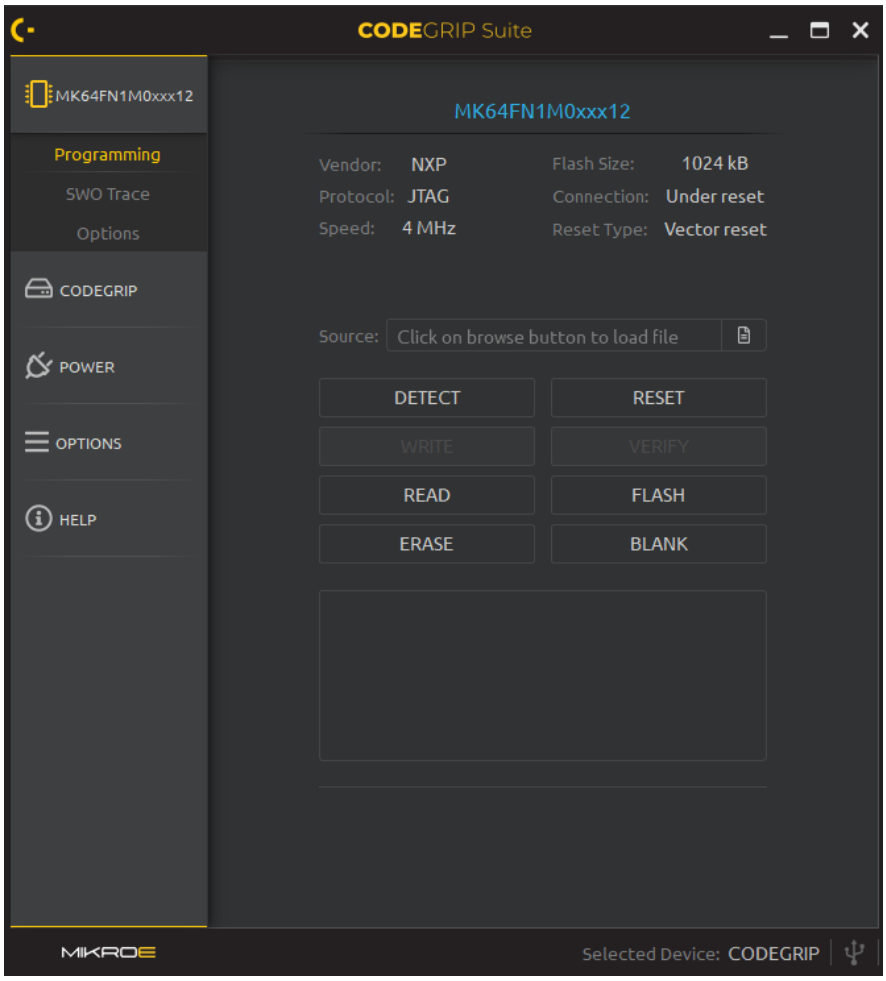
2.1 CodeGrip Programator

CodeGrip programator se koristi za programiranje mikrokontrolera i učitavanje programskog koda unutar njegove memorije. Putem *WiFi* mreže ili *USB* kabla omogućeno je programiranje mikrokontrolera. Na programatoru nalaze se *LED* indikatori koji pružaju različite informacije, poput stanja ploče, statusa programatora i drugih.

- POWER *dioda*, označava da li je ploča povezana na napajanje.
- *USB-Link dioda*, označava da je uspostavljena komunikacija *USB* kabla.
- *NET-Link dioda*, označava da je uspostavljena komunikacija putem *WiFi* mreže.
- *ACTIVE dioda*, daje informaciju o uključenom ili isključenom stanju programatora
- *DATA dioda*, u trenutku spuštanja koda, ovaj tip *LED* indikatora treperi.

Slika 2.1.1 Prikaz programatora

Uz dati programator, za spuštanje programa na razvojni sistem, koristi se *CodeGrip Suite* softverski paket, takođe razvijen od strane Mikroelektronike. Ovaj napredni alat, pruža potpunu kontrolu nad razvojnom pločom, a specijalizovan je za efikasno upravljanje debagovanjem i programiranjem mikrokontrolera. Na slici 2.1.2, prikazana je glavna strana alata.



Slika 2.1.2 Prikaz glavne strane alata

Kao što se vidi sa predhodne slike, funkcionalnosti koje omogućavaju kontrolu nad razvojnim sistemom jesu :

- *DETECT*
- *RESET*
- *WRITE*

Predhodno, povezivanje se započinje u sekciji *CODEGRIP*, gde se pritiskom na *SCAN DEVICES* detektuje razvojni sistem. Nakon odabira pomenutog mikrokontrolera i učitavanja fajla sa ekstenzijom .hex, koriste se opcije *DETECT* za detekciju ploče i *WRITE* za programiranje.

2.2 MK64FN1M0VDC12 *Kinetis mikrokontroler*

Kinetis MK64FN1M0VDC12 mikrokontroler baziran na *ARM-u Cortex-M4*, je tip mikrokontrolera koji je korišćen za potrebne i realizaciju zadatka. Za njegovo povezivanje sa razvojnim sistemom, koristi se standard MCU kartica, koji je razvila već spomenuta kompanija. Standard ima 168 konektora, što omogućava povezivanje sa velikim brojem pinova.Ovaj dizajn omogućava kompatibilnost različitih mikrokontrolera, poput Microchip PIC ili STMicroelectronics STM32, sa mnogim razvojnim sistemima. Pored tog, ovim dizajnom je olakšan rad sa mikrokontrolerima.

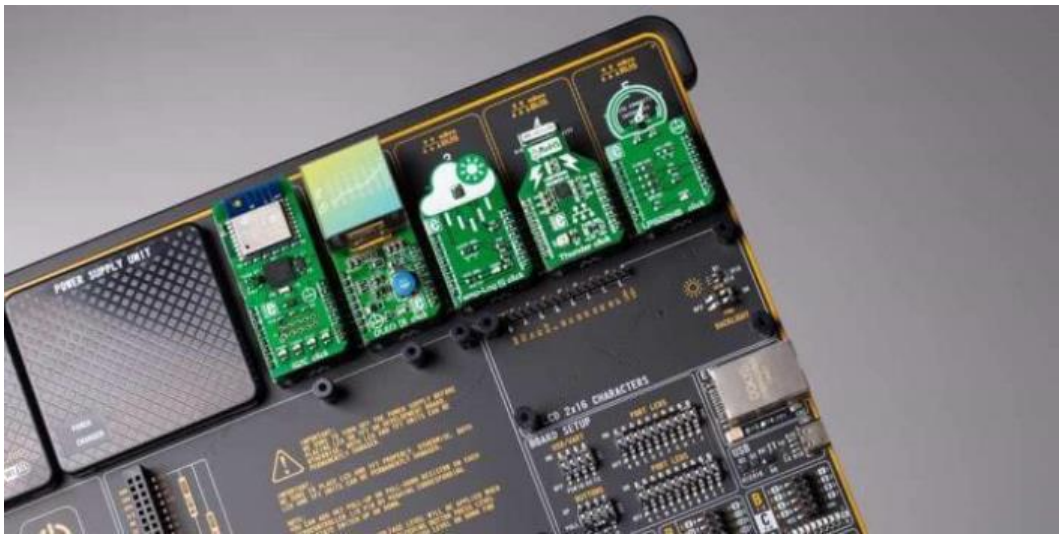
Nekoliko bitnih karakteristika pomenutog mikrokontrolera :

- Kao što je spomenuto, poseduje *ARM Cortex-M4* procesor
- 1MB *Flash* memorije
- 256kB *RAM* memorije
- Radni napon od 3.3V
- Poseduje 121 priključaka
- 6 x UART modula
- 3 x SPI modula
- 3 x I2C modula

Slika 2.2.1 Izgled mikrokontrolera

2.3 Kućišta za povezivanje Click pločica

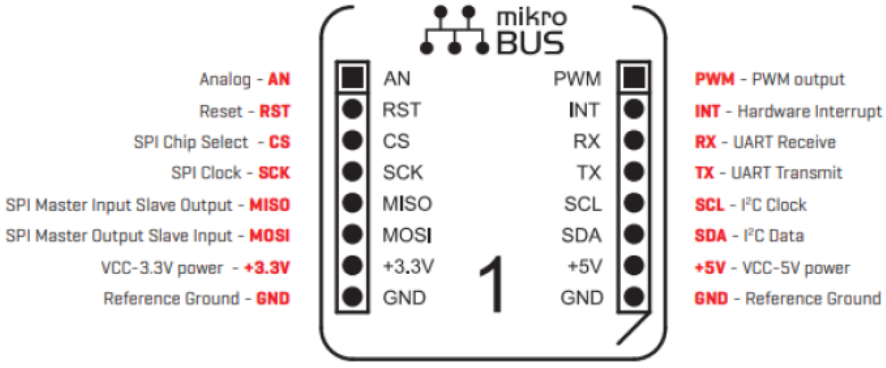
Pomenuti razvojni sistem sadrži mikroBUS kućišta za smeštanje *Click* pločica koje su razvije od strane iste kompanije.



Slika 2.3.1 Prikaz mikroBUS kućišta

Na slici 2.3.1 nalazi se prikaz mikroBUS kućišta, čiji su priključci raspoređeni u strukturu 2x8. Razvojni sistem uključuje pet ovakvih kućišta, označenih brojevima. Iz tog razloga, vrlo je bitno definisati unutar koda, koje se kućište koristi. Kućišta imaju priključke za napajanje od 3.3V i 5V.

Click pločice koje se povezuju na kućišta, podržavaju različite vrste komunikacionih protokola, kao što su *UART, SPI, I2C.* Pored priključaka za napanje, sadrže i priključak za restartovanje click pločice kao i *PWM* priključak.



Slika 3.2.2 mikroBUS standard

3. Prvi deo zadatka – Igra memorije

Kao što je već rečeno u uvodnom delu, prvi deo zadatka se odnosi na realizaciju igre memorije na *TFT* ekranu. Igra se sastoji od 16 karata raspoređenih na ekranu, gde je potrebno da korisnik klikom otkrije kartu i pokuša da pronađe njen identičan par. Igra se završava kada su svi parovi pronađeni, prikazujući proteklo vreme tokom igre i listu pet najuspešnijih rezultata. Cilj ovog dela zadatka je kreiranje interaktivne igre memorije koja će omogućiti korisnicima da testiraju svoje sposobnosti pamćenja.

3.1 *TFT 5* kapacitivni ekran

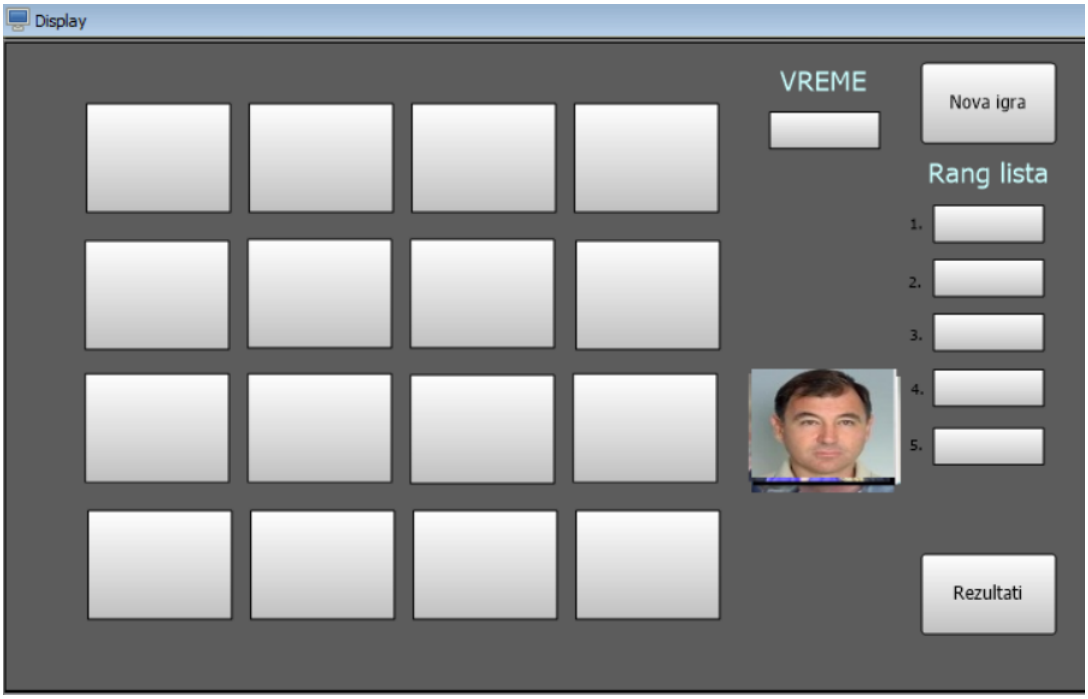
TFT 5 kapacitivni ekran, koristi se u prvom delu zadatka, kako bi imali omogućen grafički prikaz kompletne igre. Rad sa ovim tipom ekrana zahteva dodatnu aplikaciju *Visual TFT,* koja se pokreće iz programskog okruženja *mikroC for ARM.*

TFT ekran osetljiv na dodir, odnosno ima opciju detektovanja pritiska čime omogućava laku interakciju sa korisnicima, kao i da kompletna igra bude realizovana detekcijom dodira. Ekran se povezuje sa razvojnim sistemom putem 2x20-pinskog konektora, za koji već postoji odgovarajući priključak na razvojnoj ploči. Ekran koristi SSD1963 drajver i komunicira sa mikrokontrolerom putem *I2C* protokola. Eksterno napajanje nije potrebno, koristi napon od 3.3V koji je dostupan na razvojnom sistemu.

Slika 3.1.1 TFT 5 ekran

3.2 Grafički intervejs - GUI

MikroC PRO for ARM je razvojno okruženje koje unutar njega pruža mogućnost kreiranja grafičkih interfejsa na jednostavan način. Koristi širok spektar elementa poput tastera, labela, slika i mnogih drugih.



Slika 3.2.1 Prikaz grafičkog interfejsa

Za prikaz tekstualnih oznaka, poput naziva za štopericu, naziva za sortirana vremena i drugih, korišćena je komponenta *Label*. Labele su korisne i kada je potrebno često ažurirati podatke, kao što su vreme, odnoso štoperica tokom igre ili sortirani rezultati.

Prikaz kartica u stukturi 4x4, omogućen je korišćenjem komponente *Button*, sa indeksima od 1 do 16. Komponenta *Button* omogućava interaktivnost i izvršavanje određenih akcija kada se na njih pritisne i iz toga je upravo ona korišćena. Ove funkcije se tipično nazivaju *ButtonOnClick().* Zajedno sa ovom komponentom, korišćena je i komponenta *Image*, za ubacivanje određenih slika.

Komponenta *ButtonRound* je korišćena za taster, čija je uloga pokretanje nove igre, kao i za taster za prikazivanje rezultata u dnu ekrana.

3.3 Softverska realizacija prvog dela zadatka

U nastavku biće objašnjena softverska implementacija prvog dela zadatka. Jedan od najbitnijih fajlova tokom realizacije prvog dela i unutar kojeg se nalaze sve neophodne funkcije za rad programa, jeste *MyProject_events_code.c.* Na samom početku, neophodno je uključiti potrebne biblioteke. (Slika 3.3.1).

```
#include "MyProject_objects.h"
#include "MyProject_resources.h"
#include "MyProject_events_code.h"
#include "Timer2.h"
```

Slika 3.3.1 Uključene bibioteke unutar *MyProject_events_code.c*

Prva uključena biblioteka *MyProject_objects.h*, koristi se za smeštanje svih elemenata poput, tastera, slika i labela, nakon kreiranja grafičkog interfejsa. Sledeća biblioteka *MyProject_resources.h*, korisiti se za smeštanje, odnosno inicijalizaciju slika koje su opcione, ne moraju biti iste kao u zadatku. (Slika 3.3.2).

```
const code char Tahoma12x16_Regular[];
const code char Tahoma20x24_Regular[];
const code char Tahoma11x13_Regular[];
const code char bajic_bmp[16806];
const code char rajss_bmp[16806];
const code char bodic_bmp[16806];
const code char kalman_bmp[16806];
const code char brkicc_bmp[16806];
const code char miraa_bmp[16806];
const code char kisic_bmp[16806];
const code char tomic_bmp[16806];
```

Slika 3.3.2 Slike unutar *fajla MyProject_resources.h*

Možemo identifikovati 8 različitih slika, pri čemu svaka ima jedinstveni niz u heksadecimalnom formatu. Ovo je uobičajen način prikazivanja slika. Dodatno, postupak dodavanja slika je olakšan zahvaljujući direktnoj integraciji slika unutar programa, bez potrebe za dodatnim *.jpg* fajlovima. Nakon toga, potrebno je još uključiti i *Timer2.h.* Unutar iste, nalazi se definisana funkcija za inicijalizaciju tajmera, čija se implementacija nalazi u *Timer2.c* fajlu. Ovi fajlovi su nam veoma bitni, jer nam omogućavaju da pratimo proteklo vreme tokom izvršavanje igre.

Unutar fajla *Timer2.c* uključene su dodatne neophodne biblioteke, koje pružaju mogućnost rada sa standardnim funkcijama u C/C++ programskom jeziku. (Slika 3.3.3).

```
#include "Timer2.h"
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
```

Slika 3.3.3 Biblioteke unutar fajla Timer2.c

Unutar istog, nakon uključivanja neophodnih biblioteka, nalazi se još i implementacija funckije za inicijalizaciju tajmera, prikazana na slici 3.3.4.

```
void Timer2_Init(double period) {

  //proracun za podesavanje perioda: 120MHz /2 * period[s]
  uint32_t Y=0;
  double X=0.0;
  X = 60000000.0 * period;
  Y =(uint32_t) X;

  SIM_SCGC6 |= (1ul << PIT);     // Enable clock gating for Periodic interrupt timers
  PIT_MCR = 0x00;

  PIT_LDVAL2 = Y;      // Set timer prescaler. (120MHz, 500ms),  PIT_LDVAL2 = 120 [MHz] / 2 * interval [sec] ,
pri cemu je ovde interval 0.5 sekundi

  NVIC_IntEnable(IVT_INT_PIT2);  // Enable timer interrupts
  PIT_TCTRL2 = (1 << TIE);           // enable Timer 2 interrupts
  PIT_TCTRL2 |= (1 << TEN);           // start Timer 2

}
```

Slika 3.3.4 Inicijalizacija tajmera

Inicijalizacija tajmera počinje izračunavanjem Y vrednosti na osnovu zadatog perioda u sekundama, što uključuje konverziju sekundi u broj taktova tajmera i smeštanje te vrednosti u *PIT_LDVAL2* registar. Modul *SIM_SCGC6* omogućava uključivanje kloka za periodične prekide tajmera, dok se registar za kontrolu modula *PIT* postavlja na nulu kako bi se osiguralo da su tajmeri aktivirani. Na kraju, prekidna rutina se pokreće i tajmer započinje rad.

Dalje, vraćamo se na predhodni fajl *MyProject_events_code.c.* Nakon uključenih fajlova, potrebno je deklasirati neophodne promenljive, kao na slici 3.3.5. One su defisane unutar bloka *User code – End of User code*. U komentarima pored svake promenljive, naglašeno je šta ista predstavlja unutar trenutnog fajla.

```
//---------------------- User code ----------------------//
int pictureNum; // broj slike
int oldButtonNum; // predhodni taster, čuva prvi pritisnut taster
int guessedCounter; // brojač ukoliko se desio pogodak
int buttonNum; // broj pritisnutog tastera
int oldCard; // predhodna slika
int newGame;
int counter = 0;
char timeArraySeconds[3];
int gameCounter = 0;
int minutes = 0;
int seconds = 0;
int win;
int results[9]; // za smeštanje sekundi, odnosno prikaz vremena
char first[10]=""; // prikaz prvog rezultata
char second[10]="";
char third[10]="";
char fourth[10]="";
char fifth[10]="";
//----------------- End of User code ------------------//
```

Slika 3.3.5 Definasanje neophodnih promenljivih

Dalje, nakon toga sledi implementacija jednostavne funkcije, odnosno prekidna rutina tajmera. Prikazana je na slici ispod 3.3.6.

```
void Timer2_interrupt() iv IVT_INT_PIT2 {

    seconds ++;

    PIT_TFLG2.TIF = 1;

}
```

Slika 3.3.6 Timer2_Interrupt

Sa slike 3.3.6, vidi se da prekidna rutina sadrži samo dve linije koda, koje su dovoljne za potrebe realizacije prvog dela zadatka. Prvo ima povećanje promenljive *seconds*, gde nakon toga setovanjem vrednosti *PIT_TFLG2.TIF,* čisti se fleg za prekidnu rutinu tajmera, kako bi se omogućili dalji prekidi. Prekidna rutina se poziva unutar funckije *startGame()*, čija implementacija sledi nakon nje.

Pritiskom na taster *Nova igra*, poziva se funckija *startGame()*. Unutar iste, prvo se nalazi funckija *DrawScreen(&Screen1)*, koja iscrtava početno stanje ekrana, tako da su svi tasteri okrenuti na poleđinu i vreme odbroja ispočetka.

Promenljiva *newGame* se menja sa nulte vrednosti na vrednost 1, čime otpočinje nova igra. Pored toga, vrlo je bitno promenljive *seconds* i *minutes* setovati na nultu vrednost, kako se proteklo vreme ne bi dodalo dalje. Na kraju funkcije, poziva se inicijalizacija tajmera 2 koja je prethodno bila objašnjena. Kompletna funckija prikazana je na slici 3.3.7

```
void startGame(){

    DrawScreen(&Screen1);
    newGame = 1;
    seconds = 0;
    minutes = 0;
    Timer2_Init(1);
}
```

Slika 3.3.7 Funkcija startGame()

Nakon pokretanja nove igre, sledi niz od 16 funkcija sa identičnom implementacijom, pri čemu se naziv svake razlikuje samo po indeksu od 1 do 16 – showImage(). Funkcije služe za prikaz slika prilikom pritiska na odredjeni taster. Započinju sa proverom uslova da li je promenljiva *newGame* jednaka jedinici, odnosno da li je igra započeta. Ukoliko je uslov zadovoljen i igra započetka, postavljaju se promenljive *buttonNum* i *pictureNum* na odredjenu vrednost. Promenljiva *buttonNum* može imati vrednost od 1 do 16, predstavlja broj tastera na ekranu, dok *pictureNum*, može imati vrednost od 1 do 8 i predstavlja broj slike. Nakon toga, vrednost brojača *counter* se uvećava za jedan. Zatim, sledeća funkcija *TFT_image(60,45,bajic_bmp,1)* služi za iscrtavanje slike sa oznakom *bajic_bmp,* na poziciji *(x, y) = (60, 45)*. Na kraju funkcije, poziva se *imageFound()* za proveru da li se desio pogodak. Na slici 3.3.8 prikaza ja kompletna funckija *showImage*().

```
void showImage1(){

  if (newGame ==1){
    buttonNum =1;
    pictureNum =1;
    counter ++;
    TFT_image(60,45,bajic_bmp,1);
    imageFound();
  }

}
```

Slika 3.3.8 Funkcija showImage()

Dalje, na slici 3.3.9 prikazana je funkcija *imageFound*(), koja obrađuje stanje nakon pokušaja pronalaženja parova, odnosno da li su dve slike identične ili nisu. Funkcija je implementirana pomoću *switch-case* strukture i koristi promenljivu *counter* za upravljanje stanjem igre. U prvom slučaju, beleži se trenutni broj slike i tastera, dok se u drugom slučaju, proverava da li je trenutna slika identična kao prethodna. Ako su identične, prvo se promenljiva *counter* setuje na nulu, zatim se povećava brojač pogodaka *guessedCounter*. Ako su pronađeni svi parovi, promenljiva *win* se uvećava, *guessedCounter* se setuje na nulu i poziva se funkcija endGame(). Ako slike nisu identične, promenljiva *counter* se resetuje i poziva se funkcija *flip*(), koja ponovo okreće sliku kao što je bila u početnom stanju.

```
void imageFound() {
   switch(counter) {
      case 1:
         oldCard = pictureNum;
         oldButtonNum = buttonNum;
         break;
      case 2:
         if(oldCard == pictureNum) {
               counter = 0;
               guessedCounter++;
               Delay_ms(50);

            if(guessedCounter == 8) {
               win++;
               Delay_ms(50);
               endGame();
               guessedCounter = 0;
            }

         } else {
            counter = 0;
            Delay_ms(300);
            flip();
         }
         break;
      default:
         break;
   }
}
```

Slika 3.3.9 Funkcija imageFound()

Unutar funkcije flip(), za ispravno iscrtavanja početnog stanja tastera, promenljive *buttonNum* i *oldButtonNum* se smanjuju, nakon čega se poziva *DrawButton(&Button).* Funkcija je prikazana na slici 3.3.10.

```
void flip() {

   buttonNum--;
   oldButtonNum--;
   DrawButton(&Button1 + buttonNum);
   DrawButton(&Button1 + oldButtonNum);

}
```

Slika 3.3.10 Funkcija flip()

Kao što je rečeno, unutar funkcije *imageFound,* poziva se i funckija *endGame*(), ukoliko je izvršen određen broj pogođenih slika. Funkcija *endGame*() upravlja rezultatima, na osnovu vrednosti promenljive *win* kao i pomoću *switch-case* strukture. Ukupan broj slučajeva unutar iste je deset, što označava da je moguće ukupno odigrati deset partija. Ukoliko broj partija dostigne određenu granicu, odnosno promenljiva *win* bude veća od 10, ispisuje se poruka "Pokrenite igricu ponovo.". Kada se igra završi, trenutni broj sekundi se čuva u dati niz *results*, nakon čega se vrši konvertovanje u string funkcijom *IntToStr(results[0],timeArraySeconds)* i prikaz u labeli. (Slika 3.3.11).

```
void endGame(){

    switch (win){
    case 1:
    results[0] = seconds;
    IntToStr(results[0],timeArraySeconds);
    Label3.Caption = timeArraySeconds;
    DrawBox(&Box1);
    DrawLabel(&Label3);


    break;
    case 2:
    results[1] = seconds;
    IntToStr(results[1],timeArraySeconds);
    Label3.Caption = timeArraySeconds;
    DrawBox(&Box1);
    DrawLabel(&Label3);


    break;
    case 3:
    results[2] = seconds;
    IntToStr(results[2],timeArraySeconds);
    Label3.Caption = timeArraySeconds;
    DrawBox(&Box1);
    DrawLabel(&Label3);


    break;
    case 4:
    results[3] = seconds;
    IntToStr(results[3],timeArraySeconds);
    Label3.Caption = timeArraySeconds;
    DrawBox(&Box1);
    DrawLabel(&Label3);
```

Slika 3.3.11 Funkcija endGame()

Pritiskom na taster sa oznakom *Rezultati*, poziva se funkcija *showResults*(), koja prikazuje sortiranu listu pet najbržih vremena. Na početku, prvo se poziva funkcija *bubbleSort(results, win)* koja sortira niz *results* do pozicije promeljive *win,* koja označava broj elemenata. U svakom prolazu, funkcija poredi susedne elemente i zamenjuje ih, ako su u pogrešnom redosledu. Nakon toga, za svaki rezultat koji nije nula, vrednost se konvertuje u string funkcijom *IntToStr(results[0], first)* i postavlja se kao tekst u odgovarajuće labele, koje se koriste za prikaz rezultata. (Slika 3.3.12).

```
void showResults(){

  bubbleSort(results, win);

  if (results[0] != 0){
    IntToStr(results[0], first);
    Label4.Caption = first;
  }
  if (results[1] != 0){
     IntToStr(results[1], second);
     Label5.Caption = second;
   }

  if (results[2] != 0) {
    IntToStr(results[2], third);
    Label6.Caption = third;
  }

  if (results[3] != 0){
     IntToStr(results[3], fourth);
     Label7.Caption = fourth;
  }

  if (results[4] != 0) {
     IntToStr(results[4], fifth);
     Label8.Caption = fifth;
  }

}
```

Slika 3.3.12 Funkcija showResults()

4. Drugi deo projekta

Za realizaciju drugog dela projekta iskorišćen je *GSM* modul. Ovaj dio projekta podrazumeva korišćenje *GSM Click 3* pločice kompanije Mikroelektronika gde je bilo potrebno napisati program koji aktivira modem koji čeka dolaznu *SMS* poruku, a nakon prijema poruke, proverava njenu sadržinu, i ukoliko se unutar te poruke nalazi string: "Pozovi me!", program poziva broj sa koga je pristigla poruka .

Da bi se projekat realizovao bilo je potrebno sledeće:

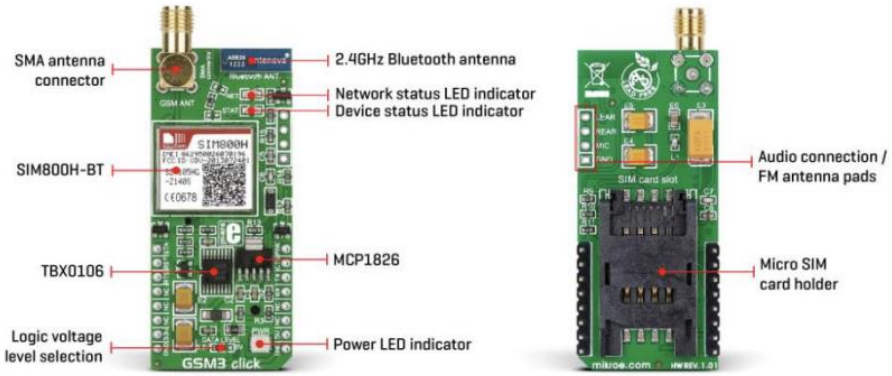
- *GSM Click 3*
- Aktivna SIM kartica koju koristi *GSM* modul
- Mobilni telefon sa aktivnom *SIM* karticom
- *Oled 0.96“* ekran

Nastavak dokumentacije objašnjava svaku od korišćenih komponenti kao i postupak realizacije projekta uz prikaz i objašnjenje algoritma rada i korišćenog koda.

4.1 GSM Click 3

GSM Click 3 je integrisano rešenje za komunikaciju putem GSM mobilne mreže. Srce ove pločice čini SIM800H-BT, četvorofrekventni 2G GSM/GPRS modul, koji podržava glavne GSM frekvencije - 850MHz, 900MHz, 1800MHz i 1900MHz..

Ovaj modul nudi indikaciju statusa mreže, detekciju smetnji i integrisane internet protokole kao što su TCP/IP, UDP, FTP, PPP, HTTP, e-mail, MMS i mnoge druge. Takođe sadrži napredne glasovne i audio funkcije, uključujući *FM* radio interfejs. Brzina prenosa podataka iznosi prosečno 85.6 kbps. Posebno značajna karakteristika je podrška za *Bluetooth 3.0+ EDR protokol*, omogućavajući prenos veće količine podataka između uređaja. Na slici 4.1.1 prikazan je izgled opisanog modula.



Slika 4.1.1. Izgled *GSM 3 Click* pločice – prednja i zadnja strana

Kao *GSM*/*GPRS* modul, može se koristiti širom sveta. Modul se pokreće putem standardnih *AT* komandi (*AT* potiče od engleske reči *Attention*), koje čine ovaj modul izuzetno korisnim za razne M2M (eng. *Machine to Machine*) aplikacije, uključujući mobilne internet terminale, automatsko očitavanje brojila, nadzor i bezbednosne aplikacije, cene puteva, praćenje imovine, kao i mnoge druge aplikacije koje koriste mobilnu mrežu.

Ključne karakteristike i funkcionalnosti:

- Napajanje: Modul se napaja sa 4V, dok se digitalni blokovi napajaju sa 2.8V. Napon od 5V sa mikroBus utičnice se smanjuje pomoću MCP1826 naponskog regulatora.
- Komunikacija: Ostvarena putem *UART2* serijskog protokola, sa brzinom prenosa između 1200bps i 115.2 kbps. *RX* linija se koristi za prijem, a *TX* linija za slanje podataka.
- *SIM* Kartica: Potrebna je *SIM* kartica sa aktivnim mobilnim brojem za registraciju na mrežu. Koristi se *SIM* kartica najvećih dimenzija, a za mikro ili nano kartice potreban je adapter. *SIM* kartica se postavlja na zadnju stranu pločice i osigurava metalnim delom.
- Antena: Za poboljšanje signala i pouzdanosti veze, neophodna je *GSM* antena.

Zbog neprilagođenosti datog mikrokontrolera razvojnom sistemu za povezivanje *GSM* modula sa mikrokontrolerom, pinovi se povezuju na sledeći način:

- *TX* pin modula na *RX* pin razvojne ploče (PD2).
- *RX* pin modula na *TX* pin razvojne ploče (PD3).
- PWK pin za pokretanje modula na PD5 pin.
- GND na GND.
- 3.3V pin na 3.3V.
- 5V pin za radni napon modula.

Statusne diode modula:

Crvena LED dioda (označena sa NET - indikacija statusa mreže):

- Konstantno upaljena: uređaj se ne napaja.
- Uključena 64ms, isključena 800ms: uređaj nije registrovan na mrežu.
- Uključena 64ms, isključena 3ms: uređaj registrovan na mrežu.
- Uključena 64ms, isključena 300ms: komunikacija uspostavljena.

Žuta LED dioda (STAT): Svetli kada je uređaj u funkciji ili spreman za rad.

Upravljanje modulom:

- *PWRKEY* pin: Za pravilno uključivanje i isključivanje modula, spaja se na *PWM* pin mikroBus utičnice. Snižavanjem nivoa pina na nisku vrednost na najmanje jednu sekundu menja se stanje napajanja modula, čime se sprečava gubitak trenutnih podešavanja.
- *RESET* pin: Alternativa za PWRKEY pin.

Dodatne funkcionalnosti:

- *Bluetooth*: Podrška za *Bluetooth 3.0+ EDR* protokol za prenos veće količine podataka između uređaja.
- Napredne glasovne i audio funkcije: Uključujući *FM* radio interfejs.

4.2 0.96” *OLED* ekran

0.96” OLED ekran rezolucije 128 x 64 piksela prikazan je na slici 4.2.1. Koristi se za praćenje stanja *GSM* modula u projektu. Na ekranu se ispisuju odgovarajuće poruke za svako stanje modula, olakšavajući praćenje i dijagnostiku.

Slika 4.2.1. *OLED 0.96’’* ekran

Ključne karakteristike i funkcionalnosti:

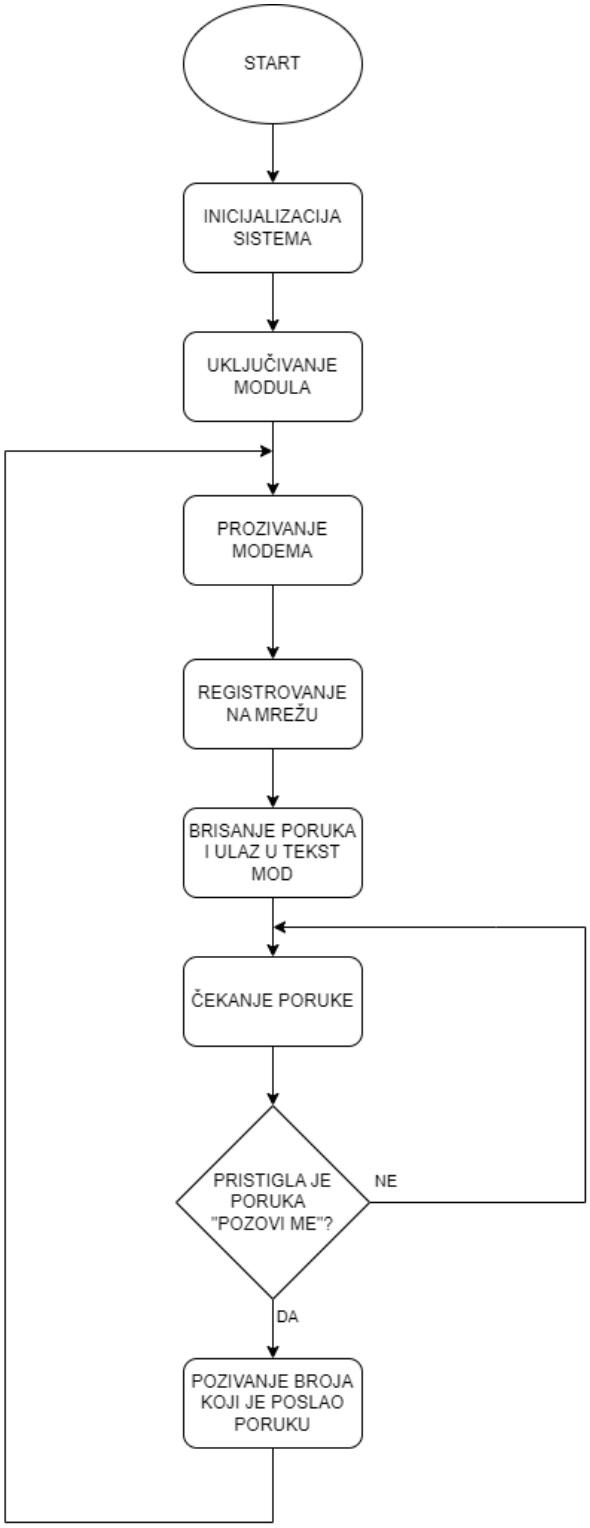
- Rezolucija: 128 x 64 piksela.
- Komunikacioni protokol: *I2C*.

Povezivanje *OLED* ekrana:

- *SDA*: Povezuje se na pin PE0 za prenos podataka.
- *SCL*: Povezuje se na pin PE1 za prenos generisanog takta.
- *GND*: Povezuje se na GND.
- *VCC*: Povezuje se na 3.3V napajanje na razvojnom sistemu.

Povezivanjem displeja i porta E razvojnog sistema putem kratkospojnika, uspostavlja se *I2C* komunikacija koja omogućava prenos informacija o trenutnom stanju modula na displej. *OLED 0.96”* ekran, sa svojim pinovima i povezivanjem, je ključni element za lako praćenje i dijagnostiku rada *GSM* modula.

4.3 Algoritam rada



Slika 4.3.1 Algoritam rada drugog zadatka

Na slici 4.3.1 prikazan je algoritam rada zadatka vezanog za rad sa *GSM* modemom. Nakon uključenja, najprije se vrši inicijalizacija sistema, gde se vrši inicijalizacija *UART* i *I2C* komunikacije potrebnih za rad sa *GSM* modemom i *OLED* ekranom. Pored toga, izvršena je inicijalizacija i *OLED* ekrana.

Nakon inicijalizacije, vrši se uključivanje *GSM* modema. Važno je napomenuti da se svako stanje modema ispisuje na *OLED* ekranu radi lakšeg praćenja njegovog rada. Po uključivanju modema, vrši se njegovo pozivanje i registrovanje na mrežu. Pre prijema bilo kakve poruke, potrebno je izbrisati prethodno primljene poruke zbog ograničene memorije za upis i prijem poruka kako bi se oslobodio prostor za narednu pristiglu poruku. Nakon brisanja poruka, dolazi se u stanje čekanja poruka u kojem modem čeka na prijem poruke od korisnika. Ukoliko je primljena poruka jednaka „Pozovi me!“, modem poziva korisnika i vraća se u početno stanje i proces se ponavlja. Ukoliko se poruka razlikuje od „Pozovi me!“, modem ostaje u tom stanju sve dok pristigla poruka ne bude jednaka zadatoj.

4.4 Softverska implementacija drugog zadatka

U nastavku projekta biće objašnjena softverska implementacija drugog dela zadatka. Bibloteka koje su korišćene za rad prikazane su na slici 4.4.1.

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "uart2.h"
#include "OLED_096.h"
```

Slika 4.4.1 Korišćene biblioteke pri radu sa *GSM* modemom

Pored standardnih bibloteka u projekat je bilo potrebno i uvrstiti i biblioteke za rad sa *UAR2* komunikacijom i *OLED* ekranom.Uz pomoć *UART2* komunikacije vršena je komunikacija razvojnog sistema i mikrokontrolera sa *GSM* modemom a korišćenjem *OLED* ekrana vršeno je praćenje stanja *GSM* modema.

Potom je izvršeno definisanje makroa, prikazanih na slici 4.4.2. Ovo je urađeno za lakši rad i razumevanje koda. Makroi *STIGAO* i *PONISTI* se koriste kako bi se signaliziralo da li je stigla neka poruka sa *UART2* modula razvojne ploče. Makro *BUFFER_SIZE* označava veličinu niza za prijem poruke poslate *GSM* modulu, dok makro *PHONE_NUMBER_SIZE* označava broj cifara broja telefona.

```
#define STIGAO 1
#define PONISTI 0
#define BUFFER_SIZE 200
#define PHONE_NUMBER_SIZE 15
```

Slika 4.4.2 Makroi za rad sa *GSM* modulom

Na slici 4.4.3. su date promenljive koje se koriste u main.c fajlu.

```
uint8_t stringUART=0, i=0,a=0;
uint8_t rx_buffer[BUFFER_SIZE];  //velicina rx buffera
uint8_t receive;  //promenjiva za smestanje karaktera UART5
uint8_t state=0;    //promenjiva za stanja modema
int p;
char phoneNumber[PHONE_NUMBER_SIZE];          //promenjiva za broj koji iscitavamo iz poruke
char *br;
char callPhoneNumber[19] = "ATD+";           //broj koji pozivamo
void GSM_PowerON();
void clear_rx_buffer();
```

Slika 4.4.3 Promenljive u okviru *main.c* fajla

Prekidna rutina za *UART2* prikaza je na slici 4.4.4. Postupak je takav da se u promenljivu receive smešta svaki pristigli karakter, pa se zatim niz *rx_buffer* puni smeštajući vrednost promenljive receive u dati niz. Ukoliko nije završeno slanje poruke, to jest ako receive ne poprimi vrednost *0x0D* što je u ovom slučaju indikator za kraj poruke (eng. *Carriage Return, CR*), u tom slučaju se nastavlja popunjavanje niza *rx_buffer*. U suprotnom, receive se postavlja na nultu vrednost a podiže se flag da je poruka stigla dodeljivanjem makroa *STIGAO* promenljivoj *stringUART*.

```
void INT_UART2_RX() iv IVT_INT_UART2_RX_TX ics ICS_AUTO{

 while ( ( UART2_S1 & UART_S1_RDRF_MASK) == 0 );
    (void) UART2_S1;
    receive = UART2_D;
    if(receive!=0x0D){
        rx_buffer[i]=receive;
        i++;
    }
    else{
        receive=0;
        i=0;
        stringUART=STIGAO;

    }
}
```

Slika 4.4.4 Prekidna rutina za *UART2* komunikaciju

Pored prekidne rutine, dodata je i funkcija koja niz *rx_buffer* prazni i postavlja u inicijalno stanje. Funkcija je prikazana na slici 4.4.5. Napravljena je tako da se svaki element niza *rx_buffer* postavlja na *null*, što može da se interpretira i kao kraj stringa. Na taj način se osigurava da je niz *rx_buffer* postavljen u inicijalno stanje.

```
void clear_rx_buffer()
{
     for(a = 0; a < 200; a++)
       rx_buffer[a] = '\0';
     i=0;
}
```

Slika 4.4.5 Funkcija za pražnjenje niza *rx_buffer*

Funkcija za uključivanje *GSM* modula prikazana je na Slici 4.4.6. Njome se vrši uključivanje *GSM* modula tako što se postavlja Pin 5 na portu D na visoki nivo (1), zatim se čeka 200 milisekundi. Nakon toga se taj pin postavlja na nizak nivo (0) i čeka 800 milisekundi. Zatim se ponovo pin na portu D postavlja na visoki nivo (1) i čeka 200 milisekundi.

```
void GSM_PowerON()
{
        GPIOD_PSOR |= (1 << 5);
        Delay_ms(200);
        GPIOD_PCOR |= (1 << 5);
        Delay_ms(800);
        GPIOD_PSOR |= (1 << 5);
        Delay_ms(200);
}
```

Slika 4.4.6 Funkcija za pražnjenje niza rx_buffer

Funkcija za inicijalizaciju sistema prikazana je na slici 4.4.7. Unutar ove funkcije izvršen je poziv funkcije za inicijalizaciju *UART2* komunikacije i uključenje prekidne rutine za *UART2*. Nakon toga podešeni su pinovi za *I2C* komunikaciju kao i njena inicijalizacija. Zatim je izvršena inicijalizacija *OLED* ekrana i čišćenje prethodnog ispisa na ekranu. Na kraju je izvršeno uključivanje samog *GSM* modula.

```
void init_system() {

    UART2_Inicijalizacija();
    delay_ms(100);
    NVIC_IntEnable(IVT_INT_UART2_RX_TX);
    GPIO_Digital_Output(&PTE_PDOR, _GPIO_PINMASK_0); // i2c1 SDA
    GPIO_Digital_Output(&PTE_PDOR, _GPIO_PINMASK_1); // i2c1 SCK
    I2C1_Init_Advanced(4000000, &_GPIO_Module_I2C1_PE1_0);
    Init_OLED();
    delay_ms(100);
    oledClear();
    GPIO_Digital_Output(&PTD_PDOR, _GPIO_PINMASK_5); // reset pin modema
    GSM_PowerON();
}
```

Slika 4.4.6 Funkcija za inicijalizaciju sistema

Na slici 4.4.7 prikazan je izgled same *main()* funkcije programa. U okviru main funkcije najprije se vrši inicijalizacija čitavog sistema , zatim se u okviru *while* petlje vrši pozivanje funkcije process_state koja će biti objašnjena u nastavku. Na kraju se čeka 100milisekundi kako bi se dalo dovoljno vremena sistemu za sledeći start.

```
void main() {
    init_system();

    while (1) {
        process_state();
        Delay_ms(100); // Sacekaj za sledeci start
    }
}
```

Slika 4.4.7 Prikaz *main* funkcije

Proces state funkcija prikazana je u nastavku. Nulto stanje ove funkcije prikazano je na slici 4.4.8.

Na početku stanja se na *OLED* ekranu ispisuje poruka „Proziva modem“. Prilikom definisanja promenljive *state*, postavljeno je da ona ima inicijalnu vrednost 0. Na početku kako bi se *GSM* modul aktivirao, mikrokontroler putem *UART2* modula šalje poruku GSM modulu u vidu stringa „AT“, te ako je za početak stigla poruka sa modula i ako je ta poruka sadržaja „OK“, može se preći na naredno stanje modema, a to je aktivno stanje. Poruka „Stanje modema aktivan“ se ispisuje u nultom stanju ako su uslovi zadovoljeni. Zatim se promenljivoj *stringUART* dodeljuje vrednost PONISTI i niz *rx_buffer* se prazni i prelazi se na naredno, prvo stanje.

```
void process_state() {
          switch (state){
            case 0:      //state 0:Proziva modem
                 oledClear();
                 oledGotoYX(0,0);
                 oledPrint("Proziva modem");
                 Delay_ms(500);
                 Uart2_WriteString("AT\r");
                 Delay_ms(500);
                 if(stringUART==STIGAO){
                    if(strstr(rx_buffer, "OK")){
                     oledClear();
                     oledGotoYX(0,0);
                     oledPrint("Stanje modema:");
                     oledGotoYX(1,0);
                     oledPrint("aktivan");
                     Delay_ms(500);
                     clear_rx_buffer();
                     state=1;
                      stringUART=PONISTI;
                    }
                    clear_rx_buffer();
                 }
                 break;
```

Slika 4.4.8 Nulto stanje *process state* funkcije

Prvo stanje *process_state()* funkcije prikazano je na slici 4.4.9. Najprije se na *OLED* ekranu ispisuje poruka „Registruje se na mrežu“. Ovo stanje predstavlja aktivno stanje modema i u tom slučaju promenljiva state ima vrednost 1. Iako je *GSM* modul aktivan, to ne znači i da je povezan na mrežu. Upravo ovo stanje opisuje registrovanje uređaja na mrežu. Registrovanje se vrši komandom *„AT+CREG=1“* i ako je odgovor modula potvrdan (*„OK“*), ispisuje se poruka: *„Registrovan“* i prelazi se u naredno, drugo stanje.

```
case 1:      //state 1: Registruje se na mrezu
      oledClear();
      oledGotoYX(0,0);
      oledPrint("Registruje se");
      oledGotoYX(1,0);
      oledPrint("na mrezu");
      Uart2_WriteString("AT+CREG=1\r");
      Delay_ms(500);
      if(strstr(rx_buffer, "OK")){
          oledClear();
          oledGotoYX(0,0);
          oledPrint("Registrovan");
          Delay_ms(500);
          state=2;
          }
      clear_rx_buffer();
      break;
```

Slika 4.4.9 Prvo stanje process state funkcije

Da bi modul uopšte mogao da čita poruke, mora mu se proslediti odgovarajuća komanda, što je u ovom slučaju *„AT+CMGF=1“* U ovom slučaju je modem u modu za poruke i može njima i da manipuliše. Da se ne bi desilo da modul pokuša da zove nešto što je zaostalo u njegovoj memoriji, prvo se briše sve iz njegove memorije pomoću komande *„AT+CMGD=X“*, gde X predstavlja broj između 0 i 4 i ispisivanje komandi tako da na kraju stoje svi brojevi između 0 i 4 označava brisanje svih poruka iz sistemske memorije *GSM* modula. Pošto su sve poruke obrisane, može se preći u naredno, treće stanje.

Drugo stanje process state funkcije prikazano je na slici 4.4.10.

```
case 2: //state2: Brise poruke i ulazi u text mod
    oledClear();
    oledGotoYX(0,0);
    oledPrint("Ulazi u text");
    oledGotoYX(1,0);
    oledPrint("mod i brisanje");
    oledGotoYX(2,0);
    oledPrint("starih poruka");
    Uart2_WriteString("AT+CMGF=1\r");  //Text mod
    Delay_ms(300);

    Uart2_WriteString("AT+CMGD=0\r");  //Brisanje poruka
    Delay_ms(300);
    Uart2_WriteString("AT+CMGD=1\r");
    Delay_ms(300);
    Uart2_WriteString("AT+CMGD=2\r");
    Delay_ms(300);
    Uart2_WriteString("AT+CMGD=3\r");
    Delay_ms(300);
    Uart2_WriteString("AT+CMGD=4\r");
    Delay_ms(300);

    if (strstr(rx_buffer, "OK")){
       oledClear();
       oledGotoYX(0,0);
       oledPrint("Poruke obrisane");
       clear_rx_buffer();
       state=3;
    }
    break;
```

Slika 4.4.10 Drugo stanje *process state* funkcije

Treće stanje *process state* funkcije služi za detekciju pristigle poruke korisnika. Najprije se na *OLED* ekranu ispisuje „*Cekam poruku*“. Sistem čeka sve dok primljena poruka nije jednaka "*Pozovi me!*". U trenutku prijema ispravne poruke na OLED ekranu se ispisuje „Stigla ispravna poruka“ i zatim se traži broj sa kojeg je poslata poruka i on se smešta u niz dodavanjem pozivnog broja „381“. Nakon toga se broj ispisuje na *OLED* ekranu .

Treće stanje *process state* funkcije prikazano je na slici 4.4.11.

```
case 3:     //state3: provera pristigle poruke
   Uart2_WriteString("ATE0\r");   //iskljucivanje eha
   Delay_ms(500);
   clear_rx_buffer();
   Delay_ms(500);
   oledClear();
   oledGotoYX(0,0);
   oledPrint("Cekam poruku");
   Delay_ms(200);
   while(!(strstr(rx_buffer,"Pozovi me!"))) {
      clear_rx_buffer();
      Uart2_WriteString("AT+CNMI=1,2,0,0,0\r");
      Delay_ms(500);
      oledClear();
      oledGotoYX(0,0);
      oledPrint("Cekam poruku");
      oledGotoYX(1,0);
      oledPrint(rx_buffer);
      Delay_ms(500);
      }
   oledClear();
   oledGotoYX(0,0);
   oledPrint("Stigla ispravna");
   oledGotoYX(1,0);
   oledPrint("poruka");
   Delay_ms(1000);
   br=strstr(rx_buffer, "381");  //Trazi broj telefona
   for(i=0; i<12; i++){
       phoneNumber[i]=*br;
       br++;
   }
   clear_rx_buffer();
   oledClear();
   oledGotoYX(1,0);
   oledPrint("Broj ");
   oledGotoYX(2,0);
   oledPrint(phoneNumber);
   Delay_ms(1000);
   state=4;
   oledClear();
   break;
```

Slika 4.4.11 Trece stanje *process state* funkcije

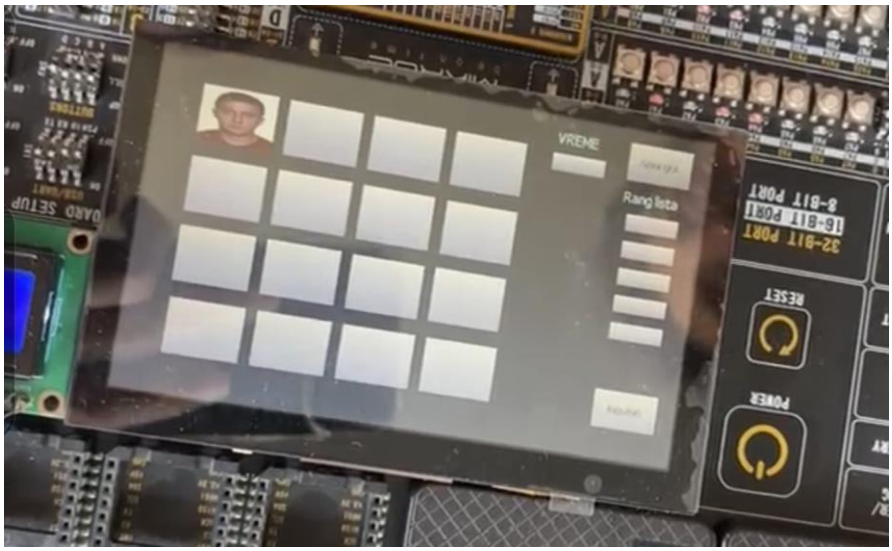
Četvrto i finalno stanje prikazano je na slici 4.4.12. U okviru njega vrši se dodavanje registrovanog broja sa kog je poslata poruka na komandu *ATD+* koja služi za poziv broja i na *OLED* ekranu se ispisuje „*Pozivam*“ nakon čega sledi poziv željenog broja.

```
case 4:
 clear_rx_buffer();
 strcat(callPhoneNumber, phoneNumber);     //  na ATD+ dodaje broj telefona
 strcat(callPhoneNumber,";\r");      //  dodaje ; na kraju
 Delay_ms(500);
 Uart2_WriteString(callPhoneNumber);
 //Uart2_WriteString("ATD+381649027709;\r"); //test broj
 Delay_ms(10000);                    //povecan Delay da bi modem stigao da izvrsi poziv!!!
 oledClear();
 oledGotoYX(0,0);
 oledPrint("Pozivam");
 for(i=0; i<15; i++){
     phoneNumber[i]='\0';
     br--;
 }
 for(i=4; i<17; i++){
     callPhoneNumber[i]='\0';
 }
 clear_rx_buffer();
 Delay_ms(5000);
 state=0;
 break;
}
```

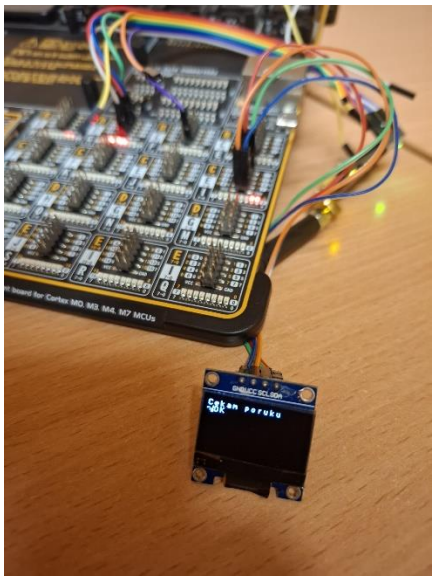
Slika 4.4.12 Trece stanje process state funkcije

5. Fizički prikaz prvog i drugog dela projekta

Na slici 5.1. i 5.2. nalazi se prikaz izgleda realizovanog prvog i drugog dela projekta. Na slici 5.1 prikazan je izgled ekrana prilikom pokretanja Igre memorije, dok se na slici 5.2 nalazi prikaz poruke na *OLED 0.96''* ekranu za jedno od stanja u kom se *GSM* modul može naći



Slika 5.1. Prikaz *TFT 5''* kapacitivnog ekrana sa kalkulatorom



Slika 5.2. Prikaz stanja *GSM* modula na *0.96'' OLED* ekranu

6. Zaključak

Realizovani projekat u potpunosti ispunjava postavljene ciljeve i nudi zadovoljavajuću funkcionalnost. Projekat je detaljno opisan, omogućavajući korisniku da prilagodi sistem prema svojim potrebama.

U okviru prvog dela projekta realizovana je igra memorije. Implementacijom ovog projekta omogućeno nam je bolje razumevanje rada sa korisničkim interfejsom i *TFT* ekranom. Takođe, unapređeno je naše prethodno znanje i korišćenje C programskog jezika. Jedan od problema s kojim smo se suočili tokom rada je pravilno postavljanje fotografija na željena mesta, kao i pravilno brojanje i računanje vremena trajanja svake runde, te njihovo ispravno rangiranje. Otežavajuća okolnost pri radu sa *GUI* u okviru ovog projekta predstavlja samo korišćenje i generisanje njegove biblioteke, za šta je bilo potrebno nakon svake izmene generisati projekat ispočetka.

Važno je napomenuti da se slike za igru memorije generišu uvek na iste pozicije, što omogućava igračima da nakon više ponovljenih igara upamte raspored fotografija, čineći igru znatno lakšom. Za buduće unapređenje projekta to bi bila jedna od osnovnih ideja, kao i istovremeno brojanje i prikazivanje vremena trajanja jedne runde, što u ovoj realizaciji nije slučaj jer se vreme prikazuje na samom kraju svake runde.

Drugi deo projekta uspešno implementira funkcionalnost koja koristi *GSM* modul za automatsko pozivanje korisnika koji pošalje poruku "Pozovi me!". Za olakšan rad i praćenje stanja *GSM* modula iskorišćen je *OLED* ekran. Ovo predstavlja jedan pojednostavljen primer nekog mnogo složenijeg sistema koji se može koristiti u informativne ili sigurnosne svrhe, kao i za samu komunikaciju između korisnika.

Kroz ovaj projekat unapređeno je znanje vezano za rad sa ovakvim vidom modula i komunikacijom između različitih komponenti. Ograničavajuća funkcionalnost ovog dela projekta predstavlja stanje u kojem modul ne registruje mrežu dovoljno dobro te nije u mogućnosti izvršiti poziv korisniku koji je poslao poruku, te je potrebno izvršiti slanje poruke iz više pokušaja.

Literatura

[1] Razvojni sistem *Fusion for ARM v8*, https://www.mikroe.com/fusion-for-arm, jul 2024.

[2]-,*Datasheet* MK64FN1M0VDC12, https://www.mikroe.com/mcu-card-for-kinetis-mk64fn1m0vdc12, jul 2024.

[3] *TFT Board 5*, https://www.mikroe.com/tft-board-5-capacitive, jul 2024.

[4] *CodeGrip Suite*, https://www.mikroe.com/codegrip, jul 2024.

[5] *GSM Click 3*, https://www.mikroe.com/gsm-3-click, jul 2024.

[6] *0.96" OLED Display*, https://www.elektor.com/products/0-96-oled-display-blue-i2c-4-pin, jul 2024.

[7] *MikroBUS standard*, https://download.mikroe.com/documents/standards/mikrobus/mikrobus-standard-specification-v200.pdf, jul 2024.