ДОДАТОК А

Графічна частина кваліфікаційної роботи


ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ
Кафедра Електронних обчислювальних машин
МЕТОД КЕРУВАННЯ СИСТЕМОЮ ГЕНЕРАЦІЇ ЕЛЕКТРОЕНЕРГІЇ З ВИКОРИСТАННЯМ БЕЗДРОТОВИХ ТЕХНОЛОГІЙ
Корнієнко Є.Д.
ст. гр. СПм 21-2
Керівник:
проф. Торба А.А.


ЗМІСТ
ВСТУП
Постановка задач
Існуючи методи
Структурні схеми методу
Дослідження та розробка методу на основі мікроконтролеру
Розробка застосунку
ВИСНОВКИ
2

ВСТУП

Метою кваліфікаційної роботи є розробка та дослідження методу керування системою генерації електроенергії з використанням бездротових технологій. У процесі дослідження має бути розроблено систему сталого енергоживлення споживача, завдяки генеруванню електроенергії. Контроль системи має здійснюватися ресурсами мікроконтролеру. Моніторинг поточних параметрів генерації електроенергії та керування режимами роботи системи виконується завдяки мобільному застосунку для операційної системи Android з використанням бездротових технологій.

Предметом дослідження є метод керування та контролю системи генерації електроенергії.

Об'єктом дослідження є процес керування та контролю системи генерації електроенергії.

3

ПОСТАНОВКА ЗАДАЧ

ПРОАНАЛІЗУВАТИ ТЕОРЕТИЧНУ БАЗУ МЕТОДІВ КЕРУВАННЯ СИСТЕМ ГЕНЕРАЦІЇ ЕЛЕКТРОЕНЕРГІЇ

РОЗРОБИТИ МЕТОД КЕРУВАННЯ СИСТЕМОЮ ГЕНЕРАЦІЇЇ ЕЛЕКТРОЕНЕРГІЇ

РЕАЛІЗУВАТИ СИСТЕМУ ГЕНЕРАЦІЇ ЕЛЕКТРОЕНЕРГІЇ НА МІКРОКОНТРОЛЕРІ ЗГІДНО РОЗРОБЛЕННОГО МЕТОДУ

РОЗРОБИТИ ЗАСТОСУНОК ДЛЯ КЕРУВАННЯ ТА МОНІТОРИНГУ СИСТЕМИ ГЕНЕРАЦІЇ ЕЛЕКТРОЕНЕРГІЇ З ВИКОРИСТАННЯМ БЕЗДРОТОВИХ ТЕХНОЛОГІЙ

ПРОТЕСТУВАТИ МЕТОД КЕРУВАННЯ

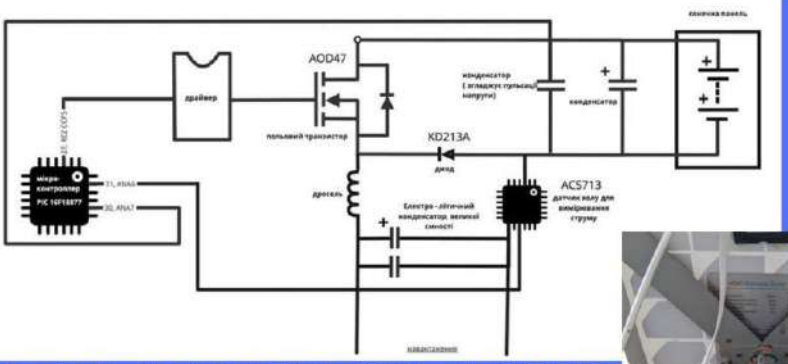
4

ІСНУЮЧІ МЕТОДИ

- PWM (Pulse Width Modulation) - метод, згідно якого береться напруга в точці максимальної потужності для цієї сонячної панелі та система підтримує цю напругу протягом роботи генерації.
- MPPT (Maximum Power Point Tracking) - метод отримання максимально можливої потужності на виході фото-модулів, принцип якого полягає в постійному скануванні потужності в точці біля точки максимальної потужності.

5

ДОСЛІДЖЕННЯ ТА АНАЛІЗ МЕТОДІВ



В РАМКАХ ДОСЛІДЖЕННЯ БУЛО ПРОВЕДЕНО ЗАГАЛЬНИЙ ЕКСПЕРИМЕНТ, ЯКИЙ МАВ НА МЕТІ ЗНАЙТИ ТОЧКУ МАКСИМАЛЬНОЇ ПОТУЖНОСТІ ЗА РІЗНИХ УМОВ ЗА ОПИСАНИМИ MPPT ТА PWM МЕТОДОМАМИ. ВИСНОВКИ ЕКСПЕРИМЕНТУ МАЛИ НА МЕТІ ЗРОЗРОБИТИ ВЛАСНИЙ МЕТОД

6

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

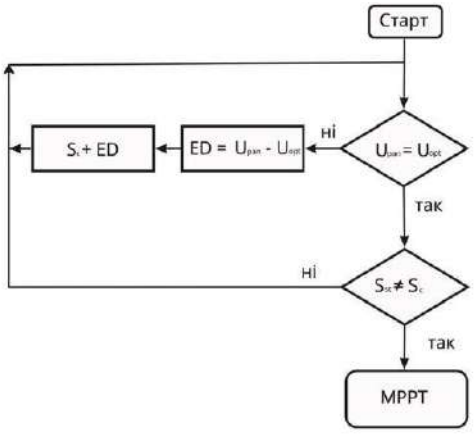
ГРАФІК ЗАЛЕЖНОСТІ ПОТУЖНОСТІ ВІД СТРУМУ *

ГРАФІК ЗАЛЕЖНОСТІ ПОТУЖНОСТІ ВІД НАПРУГИ *

ГРАФІК ЗАЛЕЖНОСТІ ПОТУЖНОСТІ ВІД ЧАСУ *

*ПІД ЧАС ПОШУКУ ТОЧКИ МАКСИМАЛЬНОЇ ПОТУЖНОСТІ

АЛГОРИТМІЧНІ СХЕМИ ДІЇ ВЛАСНОГО МЕТОДУ



Скорочення:

$U_{вих}$ - напруга завдання

$U_{рзн}$ - напруга сонячної панелі

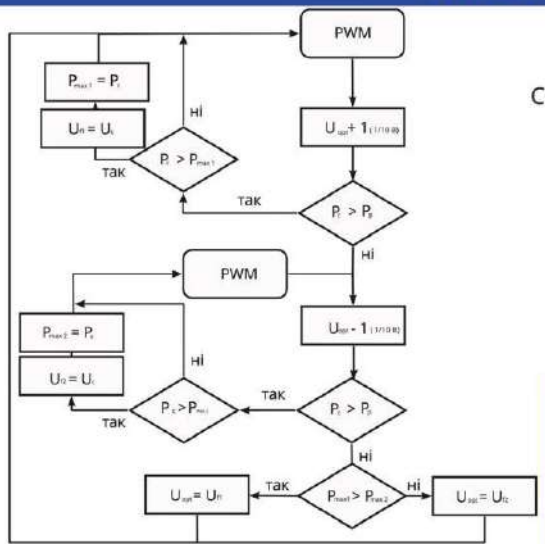
ED - дельта помилки

S_c - Поточна шпаруватість

$S_{оп}$ - Стабілізаційна шпаруватість

І ЧАСТИНА МЕТОДУ PWM БЛОК

АЛГОРИТМІЧНІ СХЕМИ ДІЇ ВЛАСНОГО МЕТОДУ

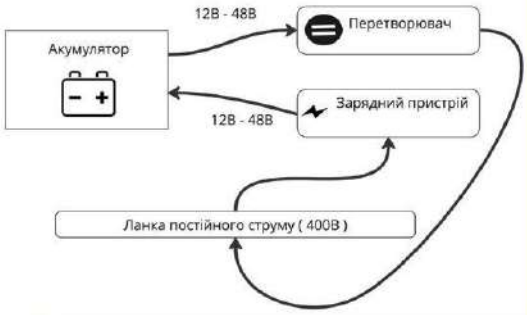


Скорочення:

- $U_{опт}$ - напруга завдання
- P_t - поточна потужність
- P_{t0} - потужність при минулій ітерації
- P_{max1} - максимальна потужність першого блоку
- P_{max2} - максимальна потужність другого блоку
- U_{n1} - фіксована напруга при максимальній потужності на першому блоці
- U_{n2} - фіксована напруга при максимальній потужності на другому блоці

II ЧАСТИНА МЕТОДУ MPPT БЛОК

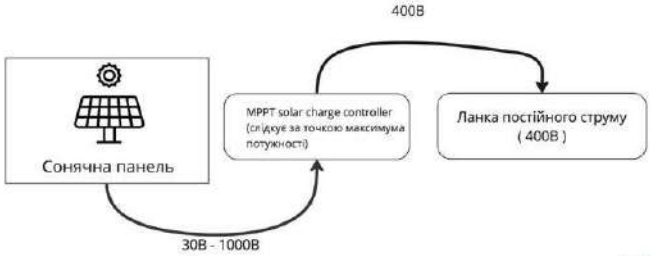
9



РОЗРОБКА СТРУКТУРНИХ СХЕМ МЕТОДУ КЕРУВАННЯ

СХЕМА З'ЄДНАННЯ АКУМУЛЯТОРУ ТА ЛАНКИ ПОСТІЙНОГО СТРУМУ

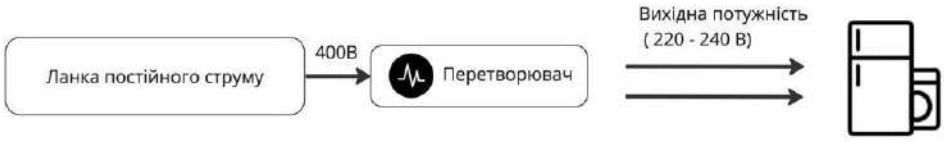
СХЕМА З'ЄДНАННЯ СОНЯЧНОЮ ПАНЕЛІ ТА ЛАНКИ ПОСТІЙНОГО СТРУМУ



10

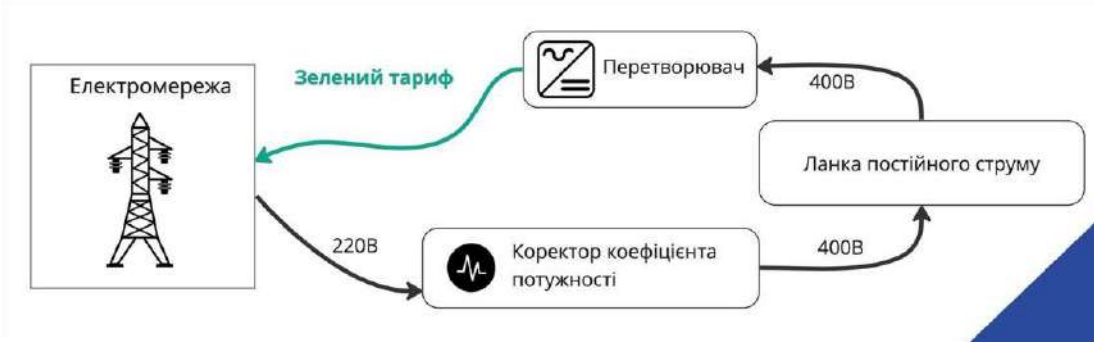
СХЕМА З'ЄДНАННЯ АКУМУЛЯТОРУ ТА ЛАНКИ ПОСТІЙНОГО СТРУМУ

СХЕМА З'ЄДНАННЯ ЛАНКИ ПОСТІЙНОГО СТРУМУ ТА СПОЖИВАЧА



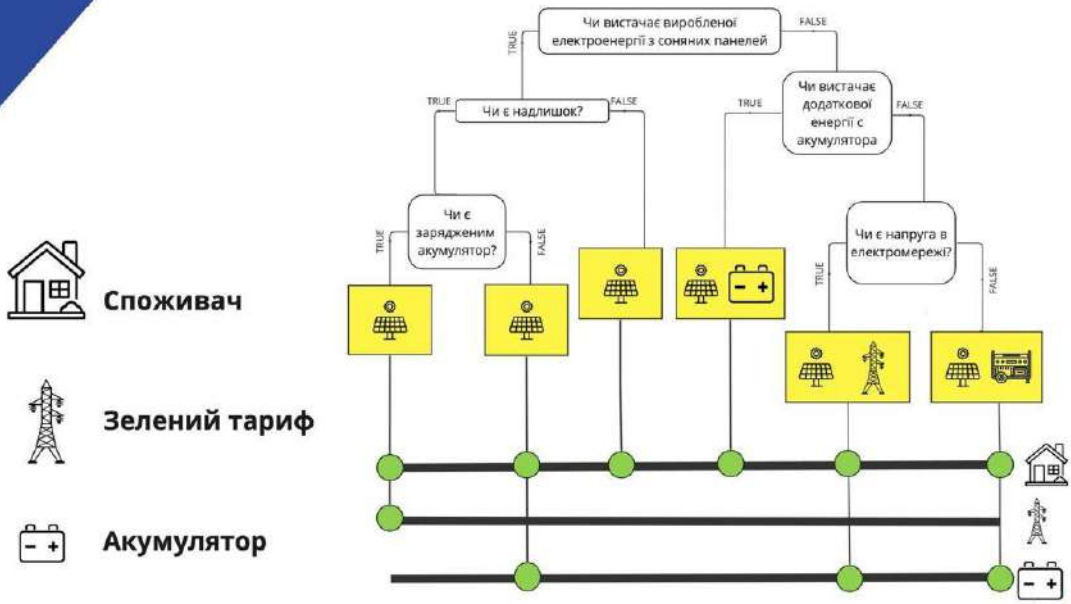
11

СХЕМА З'ЄДНАННЯ МЕРЕЖІ ТА ЛАНКИ ПОСТІЙНОГО СТРУМУ



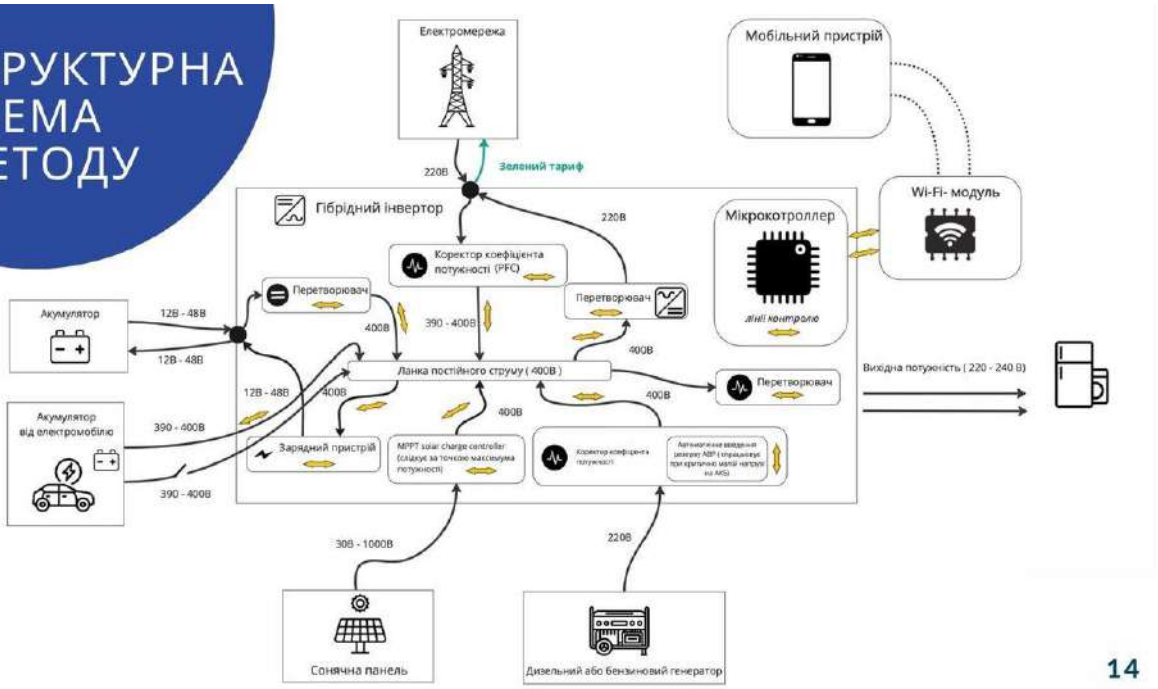
12

РЕЖИМИ РОБОТИ СИСТЕМИ



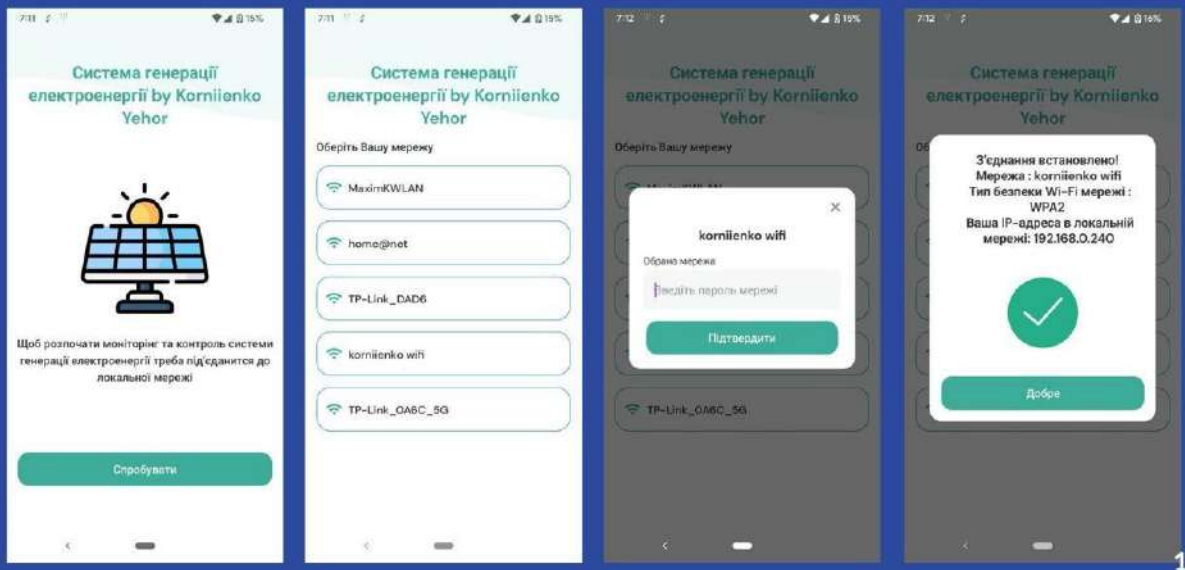
13

СТРУКТУРНА СХЕМА МЕТОДУ



14

ВІТАЛЬНИЙ ЕКРАН ТА ПОШУК МЕРЕЖІ

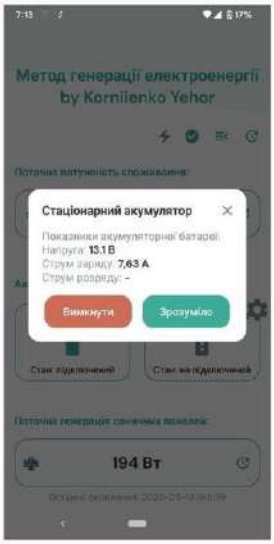


15

ГОЛОВНИЙ ЕКРАН ЗАСТОСУНКУ

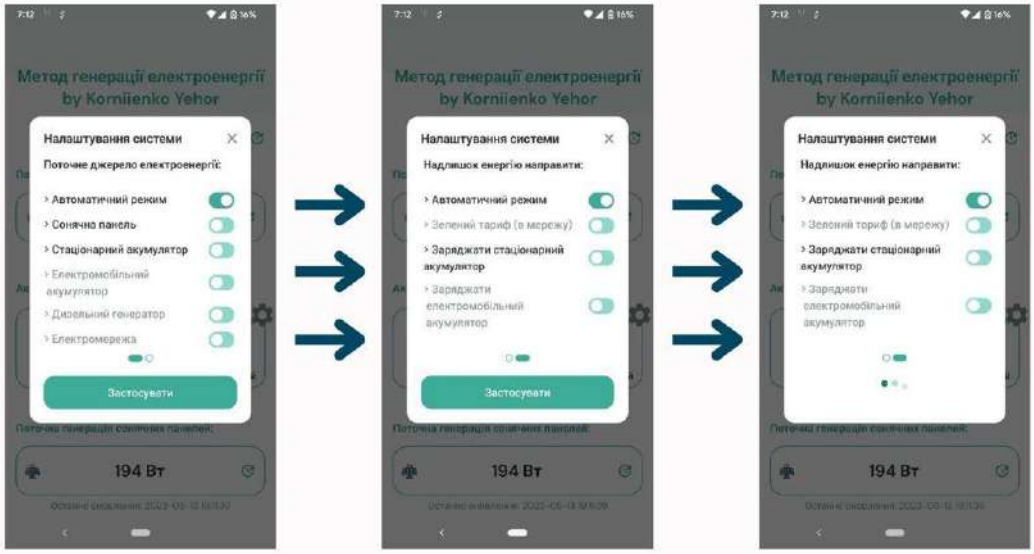
16

МОНІТОРИНГ ТА КЕРУВАННЯ АКУМУЛЯТОРНИМИ БАТАРЕЯМИ



17

НАЛАШТУВАННЯ РЕЖИМІВ РОБОТИ СИСТЕМИ ГЕНЕРАЦІЇ ЕЛЕКТРОЕНЕРГІЇ



18

ВИСНОВКИ

В ХОДІ КВАЛІФІКАЦІЙНОЇ РОБОТИ БУВ РОЗРОБЛЕНИЙ МЕТОД КЕРУВАННЯ СИСТЕМОЮ ГЕНЕРАЦІЇ ЕЛЕКТРОЕНЕРГІЇ, ЯКИЙ ПЕРЕДБАЧАЄ ВИРІШЕННЯ ПРОБЛЕМ ВЖЕ ІСНУЮЧИХ СИСТЕМ, ТАКИХ ЯК: ЗНЕСТРУМЛЕННЯ МЕРЕЖІ ВНОЧІ ЗА УМОВИ РОЗРЯДЖЕНИХ АКУМУЛЯТОРІВ; ПОВНИЙ МОНІТОРИНГ ПРОЦЕСІВ, ЗАВДЯКИ РОЗРОБЛЕНОМУ МЕХАНІЗМУ З'ЄДНАННЯ МІКРОКОНТРОЛЕРУ З КОЖНИМ ЕЛЕМЕНТОМ СИСТЕМИ; ЗАРЯДЖАННЯ АКУМУЛЯТОРІВ ПРИ ПОТРЕБІ, НЕ ТІЛЬКИ ВІД СОНЯЧНИХ ПАНЕЛЕЙ. ПРОВЕДЕНА РОЗРОБКА ТЕХНОЛОГІЇ ПОШУКУ МАКСИМАЛЬНОЇ ПОТУЖНОСТІ НА СОНЯЧНІЙ ПАНЕЛІ НА ОСНОВІ ВЖЕ ІСНУЮЧИХ РІШЕНЬ, ШЛЯХОМ КОМБІНУВАННЯ НАЙПОШИРЕНІШИХ ІЗ НИХ, ТАКИХ ЯК PWM ТА MPPT, ТА МІНІМІЗУВАННЯ НЕДОЛІКІВ КОЖНОГО З НИХ.

19

ДОДАТОК Б

Код мікроконтролеру з позиціонування сонячної панелі

```
#define _XTAL_FREQ 32000000
#include <xc.h>

#include "UprDvig.h"
#include "OprosDX.h"                    // підключення процесору
#include "Poz18877.h"
#include "obslADC.h"
#include "Statistika.h"

#pragma jis                     // допускає конфігурування не латинських символів

void Pozicioner(void){
  if(Komanda==kKomPusk){
    Komanda=0;
    Flag.BlPoz=0;           // знімаємо блокіровку позиціонування
    rVrVrGD=0;              //
    rVrVrVD=0;              // регістр часу обертання вертикального двигуна після прибуття команды с
пульта
    }
  if(Komanda==kKomStop){
    Komanda=0;
    Flag.BlPoz=1;           //блокуємо позиціонування
    }
  if(Komanda==kDvUp){
    Komanda=0;
    rVrVrVD=kVrVrD;
    Flag.BlPoz=1;           // блокуємо позиціонування
    Flag.DvUp=1;            //
    }
  if(Komanda==kDvDo){
    Komanda=0;
    rVrVrVD=kVrVrD;
    Flag.BlPoz=1;           // блокуємо позиціонування
    Flag.DvDoun=1;              //двигатель донизу
    }
  if(Komanda==kDvR){
    Komanda=0;
    rVrVrGD=kVrVrD;
    Flag.BlPoz=1;           // блокуємо позиціонування
    Flag.DvR=1;             //
    }
  if(Komanda==kDvL){
    Komanda=0;
    rVrVrGD=kVrVrD;
    Flag.BlPoz=1;           // блокуємо позиціонування
    Flag.DvL=1;             //
    }

  if(Flag.BlPoz==0){            //якщо позиціонування не заблоковано      (FlagDO.Soln==1)
    while(FlagDO.Izm==1){       //
```

```
        FlagDO.Izm=0;
        if(FlagDX.DX_Up==1||FlagDOres.Oprok==1){
          if(FlagDOres.Oprok==0){
            FlagDOres.Oprok=1;
            FlagDOres.XodDo=1;
            }else{
            if(FlagDX.DX_Do==1){
              FlagDOres.Oprok=0;
              FlagDOres.XodDo=0;
              }else{
              FlagDOres.XodDo=1;
              }
            }
          FlagDOres.Izm=1;
          break;
          }//было опрокидывание

//        if(FlagDOres.Dtil==0){     // якщо не потрібно практикувати обертання панелі після заходу
сонця
//          FlagDOres.VceF=0;      //зачищаемо усі прапори
//          }else{

//          }
        FlagDOres.XodUp=0;FlagDOres.XodDo=0;
        FlagDOres.XodL=0;FlagDOres.XodR=0;

        FlagDOres.Izm=1;

        if(FlagDO.DOtil==1){              // повинні бути повернуті до кінцевих датчиків
          FlagDOres.Dtil=1;
          }//если засвечен тыл

        if(FlagDO.DOgorL==1||FlagDO.DOgorR==1){
          FlagDOres.Dtil=0;
//          if(FlagDO.DOup==0){              може не запалюватися з-за кута
//            FlagDOres.XodDo=1;
//            }
          if(FlagDO.DOgorL==1&&FlagDO.DOgorR==1){    // завітлен влівий та правий
            FlagDOres.XodL=0;FlagDOres.XodR=0;
            if(FlagDO.DOup==0){
              FlagDOres.XodDo=1;
              }
            }else{                        //засвітлений один з лівих або правих
            if(FlagDO.DOgorL==1){
              if(FlagDO.DOgorR==0){
                FlagDOres.XodL=1;
                }
              }else{                    //одже засвілений правий
              FlagDOres.XodR=1;
              }
            }// засвілений один із лівих або правих
          }else{                       //якщо жоден не засвічений
          if(FlagDO.DOup==1){
            FlagDOres.Dtil=0;
```

```
            FlagDOres.XodUp=1;
//            break;
            }
          if(FlagDO.DOdoun==1){
            FlagDOres.Dtil=0;
            FlagDOres.XodDo=1;
//            break;
            }
          }//якщо не лівий не правий не засвічен
        if(FlagDOres.Dtil==1){          //якщо флаг залишився то не засвічен жоден
          switch(SchTil) {                  // MPPT
          case 0: //
            FlagDOres.XodL=1;
            if(FlagDX.DX_L==0){
              break;
              }
            ++SchTil;FlagDOres.XodL=0;
          case 1: //
            FlagDOres.XodR=1;
            if(FlagDX.DX_R==0){
              break;
              }
            FlagDOres.XodR=0;
          default:
            SchTil=0;
            FlagDOres.Dtil=0;
            break;
            }//switch(SchTil) {
          }else{//if(FlagDOres.Dtil==1)//залишився флаг – не засвічен жоден з правих та лівих
          SchTil=0;
          }
        break;
        } //while(FlagDO.Izm==1){
      }else{                        //позіціонування заблоковано ( управління з пульта)
      if(FlagVr.VrVrGD==1){          //якщо сплинув час обернення горизонтального двигуна
        FlagVr.VrVrGD=0;
        FlagGD.DvigON=0;
        Flag.DvR=0;
        Flag.DvL=0;
        }
      if(FlagVr.VrVrVD==1){          //якщо сплинув час оберту вертикального двигуна
        FlagVr.VrVrVD=0;
        FlagVD.DvigON=0;
        Flag.DvUp=0;
        Flag.DvDoun=0;
        }
      if(Flag.DvUp==1){
        FlagVD.DvigON=1;
        FlagVD.VrRevers=0;
        }
      if(Flag.DvDoun==1){
        FlagVD.DvigON=1;
        FlagVD.VrRevers=1;
        }
```

```
      if(Flag.DvR==1){
        FlagGD.DvigON=1;
        FlagGD.VrRevers=0;
        }
      if(Flag.DvL==1){
        FlagGD.DvigON=1;
        FlagGD.VrRevers=1;
        }
      }//позіоцінування заблоковано ( управління з пульта)
   }

void ObslVD(void){          //CWG2  PWM7
unsigned char tekUch;
if(FlagDOres.Izm==0)return;
if(FlagDOres.XodUp==0&&FlagDOres.XodDo==0){       //зупинка
   if(PWM7DCH==122){                    //якщо вже зупинились
      FlRegMPPT.VDon=0;
      return;
      }

   if(PWM7DCH>122){                     //якщо рухалися догори
      tekUch=(unsigned)(PWM7DCH-1);
      if(tekUch<=PWMvdUoff){
        tekUch=122;
        }
      }else{                            //якшо рухалися вниз
      tekUch=(unsigned)(PWM7DCH+1);
      if(tekUch>=PWMvdDoff){
        tekUch=122;
        }
      }
   PWM7DCH=tekUch;
   return;
   }//if(FlagDOres.XodUp==0&&FlagDOres.XodDo==0){       //зупиняємося

while(FlagDOres.XodUp==1){
   FlagDOres.XodUp=0;
   if(FlagDX.DX_Up==1){                 //датчик хола
      PWM7DCH=122;
      break;
      }
   if(PWM7DCH==122){                    //якщо стоїм
      PWM7DCH=PWMvdUmin;
      break;
      }
   if(PWM7DCH<122){                     //якщо рухаємося вниз
      tekUch=(unsigned)(PWM7DCH+5);
      if(tekUch>PWMvdDoff){
        tekUch=122;
        }
      PWM7DCH=tekUch;
      break;
      }
   tekUch=(unsigned)(PWM7DCH+3);
```

```
   if(tekUch>=PWMvdUmax){
      tekUch=PWMvdUmax;
      }
   PWM7DCH=tekUch;
   break;
   }//while(FlagDOres.XodR==1){
while(FlagDOres.XodDo==1){
   FlagDOres.XodDo=0;
   if(FlagDX.DX_Do==1){              //датчик холла
      PWM7DCH=122;
      break;
      }
   if(PWM7DCH==122){                 //якщо стоїм
      PWM7DCH=PWMvdDmin;
      break;
      }
   if(PWM7DCH>122){                  //обертались догори треба реверсувати
      tekUch=(unsigned)(PWM7DCH-5);
      if(tekUch<=PWMvdUoff){
         tekUch=122;
         }
      PWM7DCH=tekUch;
      break;
      }
   tekUch=(unsigned)(PWM7DCH-3);
   if(tekUch<=PWMvdDmax){
      tekUch=PWMvdDmax;
      }
   PWM7DCH=tekUch;
   break;
   }//while(FlagDOres.XodL==1){
if(PWM7DCH==122){
   FlRegMPPT.VDon=0;                 //команда на поворот є, але уперлись в кінець
   }else{
   FlRegMPPT.VDon=1;
   }

}//void ObslVD(void);


void ObslGD(void){              //CWG1  PWM6
unsigned char tekUch;
if(FlagDOres.Izm==0)return;
FlagDOres.Izm=0;
if(FlagDOres.XodR==0&&FlagDOres.XodL==0){       //зупиняємося або шукаємо сонце
   if(PWM6DCH==122){
      FlRegMPPT.GDon=0;
      }
   if(FlagDOres.Dtil==1){                //якщо шукаємо сонце
      if(FlagDOres.KoncL==0){             //ще не відпрацювали вліво
         if(FlagDX.DX_L==1){
            FlagDOres.KoncL=1;
            PWM6DCH=122;
            return;
```

```
        }
      if(PWM6DCH==122){
        PWM6DCH=PWMgdLmin;
        return;
        }
      if(PWM6DCH<122){              //обертання ліворуч
        tekUch=(unsigned)(PWM6DCH-3);
        if(tekUch<PWMgdLmax){
          tekUch=PWMgdLmax;
          }
        PWM6DCH=tekUch;
        return;
        }
      }//if(FlagDOres.KoncL==0){ //ще не поверулись ліворуч
    if(FlagDOres.KoncL==1){                 //відпрацбвали вліво
      if(FlagDX.DX_R==1){
        PWM6DCH=122;
        FlagDOres.Dtil=0;                 //знімаємо прапор пошуку сонця
        FlagDOres.KoncL=0;
        return;
        }
      if(PWM6DCH==122){
        PWM6DCH=PWMgdRmin;
        return;
        }

      if(PWM6DCH>122){
        tekUch=(unsigned)(PWM6DCH+3);
        if(tekUch>=PWMgdRmax){
          tekUch=PWMgdRmax;
          PWM6DCH=PWMgdRmin;
          return;
          }
        }
      }//if(FlagDOres.KoncR==0){            //якщо не відпрацювали вправо
    }//if(FlagDOres.Dtil==1){               //якщо шукаємо сонце
  if(PWM6DCH==122)return;                   //якщо вже зупинились
  if(PWM6DCH>122){                  //якщо рухалися вліво
    tekUch=(unsigned)(PWM6DCH-1);
    if(tekUch<=PWMgdRoff){
      tekUch=122;
      }
    }else{ //if(PWM6DCH>122){       //якщо рухалися вправо
    tekUch=(unsigned)(PWM6DCH+1);
    if(tekUch>=PWMgdLoff){
      tekUch=122;
      }
    }
  PWM6DCH=tekUch;
  return;
  }//if(FlagDOres.XodR==0&FlagDOres.XodL==0){ //зупиняємося або шукаємо сонце
while(FlagDOres.XodR==1){
  FlagDOres.XodR=0;
  if(FlagDX.DX_R==1){               //датчик холла
```

```
      PWM6DCH=122;
      break;
      }
   if(PWM6DCH==122){                  //якщо стоїм
      PWM6DCH=PWMgdRmin;
      break;
      }
   if(PWM6DCH<122){                   //якщо рухалися вліво треба реверсувати
      tekUch=(unsigned)(PWM6DCH+5);
      if(tekUch>PWMgdLoff){
        tekUch=122;
//        ZadRevGd=K_ZadRev;
        }
      PWM6DCH=tekUch;
      break;
      }
   tekUch=(unsigned)(PWM6DCH+3);
   if(tekUch>=PWMgdRmax){
      tekUch=PWMgdRmax;
      }
   PWM6DCH=tekUch;
   break;
   }//while(FlagDOres.XodR==1){

while(FlagDOres.XodL==1){
   FlagDOres.XodL=0;
   if(FlagDX.DX_L==1){                //датчик холла
      PWM6DCH=122;
      break;
      }
   if(PWM6DCH==122){                  //якщо стоїм
      PWM6DCH=PWMgdLmin;
      break;
      }
   if(PWM6DCH>122){                   //рухалися вправо треба реверсувати
      tekUch=(unsigned)(PWM6DCH-5);
      if(tekUch<=PWMgdRoff){
        tekUch=122;
//        ZadRevGd=K_ZadRev;
        }
      PWM6DCH=tekUch;
      break;  }
   tekUch=(unsigned)(PWM6DCH-3);
   if(tekUch<=PWMgdLmax){
      tekUch=PWMgdLmax;
      }
   PWM6DCH=tekUch;
   break;
   }//while(FlagDOres.XodL==1){
if(PWM6DCH==122){
   FlRegMPPT.GDon=0;
   }else{FlRegMPPT.GDon=1;  }
}//void ObslGD(void){     //CWG1  PWM6
```

ДОДАТОК В

Код мікроконтролеру з пошуку точки максимальної потужності

```
#define _XTAL_FREQ 32000000
#include <xc.h>
#include "Statistika.h"
#include "obslADC.h"


void PWM_MPPT_AKB(void){
uint24_t    tekUint24;
uint16_t    tekUint16;
//                          FlagADC.Upaneli=1;
if(FlagADC.Upaneli==1&&FlRegPWM.Vr==1){
   IzmADC=0;FlRegPWM.Vr=0;FlagADC.Upaneli=0;

while(1){

if(Upaneli<Uakb){
   PWMpan=0;
   FlRegPWM.UvPWM=1;              //прапор збільшення кута
   FlRegMPPT.MPPT=1;              //переходимо на PWM
   FlRegPWM.Uopt=0;
   koRevUopt=0;
   UpaneliOpt=kUpaneliOpt;
   break;
   }

tekUint24=Upaneli;
tekUint24*=TokBatZar;
Wpaneli=tekUint24/100;          //потужність панелі в сотнях міліват
tekUint24=TokBatZar;
tekUint24*=Uakb;
Wvix=tekUint24/100;             //потужність на АКБ в сотнях міліват
PWMpan=CCPR5;

if(FlRegMPPT.BlOffPozic==0){
   if(FlRegMPPT.GDon==1||FlRegMPPT.VDon==1)break;
   }

          //тестовый
          //if(Upaneli<UpaneliOpt){
          //   dPWMpan=UpaneliOpt-Upaneli;
          //   }else{
          //   dPWMpan=Upaneli-UpaneliOpt;
          //   }
          //if(dPWMpan>5){
          //   ++dPWMpan;
          //   }
          //break;

if(TokBatZar<=KoTokBatZarMin){        //якщо освітленість низька
   if(FlRegPWM.Uopt==1){           //якщо Uopt застабілізована
```

```
      FlRegMPPT.MPPT=1;
      if(UpaneliOpt>(UakbMax+5)){        //и знажаємо UpaneliOpt
        UpaneliOpt-=3;
        FlRegPWM.Uopt=0;
        FlRegPWM.PWMstab=0;
        }else{
        if(UpaneliOpt!=(UakbMax+5)){
          FlRegPWM.Uopt=0;
          FlRegPWM.PWMstab=0;
          }
        UpaneliOpt=(UakbMax+5);
        }
      }
    }

if(Upaneli<UpaneliOpt){
    if(FlRegPWM.UvPWM==1){
      FlRegPWM.UvPWM=0;
      if(koRevUopt<KkoRevUopt){
        ++koRevUopt;
        }
      }
    dPWMpan=(unsigned)(UpaneliOpt-Upaneli);
    if(FlRegPWM.Start==0){
      dPWMpan>>=1;
      }
    if(dPWMpan==0){dPWMpan=1;}
    if(PWMpan>=dPWMpan){
      PWMpan-=dPWMpan;
      }else{
      PWMpan=0;
      }
    }else{//if(Upaneli<UpaneliOpt)
    dPWMpan=(unsigned)(Upaneli-UpaneliOpt); //тут якщо Upaneli>=UpaneliOpt

    if(FlRegPWM.UvPWM==0){
      FlRegPWM.UvPWM=1;
      if(koRevUopt<KkoRevUopt){
        ++koRevUopt;
        }
      }else{
      if(Upaneli==UpaneliP&&dPWMpan==0){      //якщо другий вимір стоїм на UpaneliOpt
        if(koRevUopt<KkoRevUopt){
          ++koRevUopt;
          }
        }
      }

    if(FlRegPWM.Start==0){
      if(dPWMpan!=0){
        dPWMpan>>=1;
        if(dPWMpan==0){
          dPWMpan=1;
          }
```

```
        }
      }else{                    //відпрацьовуємо старт
      if(dPWMpan==0){
        dPWMpan=1;
        }
      }
   PWMpan+=dPWMpan;
   if(PWMpan>930){
      PWMpan=930;
      FlRegPWM.Uopt=1;
      koRevUopt=0;
      FlRegPWM.PWMstab=0;

      }
   }//if(Upaneli<UpaneliOpt)


   if(FlRegPWM.PWMstab==1){          //PWMstab вже було зафіксовано                    //
      if(PWMstab>PWMpan){
        tekUint16=PWMstab-PWMpan;
        }else{
        tekUint16=PWMpan-PWMstab;
        }
      if(tekUint16>6){              //умови для панелі змінились суттєво
        FlRegMPPT.MPPT=1;
        }
      }

if(TokBatZar<KoTokBatZarMin){break; }                  //якщо низька освітленість

if(FlRegMPPT.MPPT==0){koProxMPPt=0;break;}              //виходимо, якщо не пора в режимі  MPPT

if(FlRegMPPT.MPPToff==1)break;              // якщо  MPPT заблокован

if(FlRegPWM.Sum4IzmW==0)break;              //якщо не має усередненої потужності

FlRegPWM.PWMstab=0;

switch(koProxMPPt) {                    // MPPT
   case 0: //-UpaneliOpt
   ++koProxMPPt;                        //NTest=0;
   UpaneliTest1=UpaneliOpt;
   TokBatZarTest1=TokBatZarADC;
   UpaneliTest2=0;
   TokBatZarTest2=0;
   --UpaneliOpt;
   PUbT=0;
//    if(PWMstab>PWMpan){  //TokBatZarP<=TokBatZar мощность падает
//       UpaneliOpt-=5;
//       }else{
//       UpaneliOpt+=5;
//       }
//    UpaneliOpt+=2;
   break;
```

```
   case 1:
//++NTest;
if(TokBatZarTest1<TokBatZarADC){
   TokBatZarTest1=TokBatZarADC;
   UpaneliTest1=UpaneliOpt;
   PUbT=0;
   }else{
   ++PUbT;
   if(PUbT>=KPUbTmax){
      PUbT=0;
      UpaneliTest2=UpaneliOpt;
      TokBatZarTest2=TokBatZarADC;
      UpaneliOpt+=2;
      ++koProxMPPt;
      break;
      }
   }
   if(UpaneliOpt<=(UakbMax+5)){
      ++koProxMPPt;
      UpaneliTest2=UpaneliOpt;
      TokBatZarTest2=TokBatZarADC;
      UpaneliOpt+=2;
      PUbT=0;
      break;
      }
   --UpaneliOpt;
   break;
   case 2:
   if(TokBatZarTest2<TokBatZarADC){
      TokBatZarTest2=TokBatZarADC;
      UpaneliTest2=UpaneliOpt;
      PUbT=0;
      }else{
      ++PUbT;
      if(PUbT>=KPUbTmax){
         FlRegMPPT.MPPT=0;
         break;
         }
      }//ток убывает

   if(UpaneliOpt>kUpaneliOpt||TokBatZar<=(KoTokBatZarMin+1)){
      FlRegMPPT.MPPT=0;
      break;
      }
   ++UpaneliOpt;
   break;
   default:
   koProxMPPt=0;
   break;
   }//switch(FlagMPPT.BilR)

if(FlRegMPPT.MPPT==0){
   if(TokBatZarTest1>TokBatZarTest2){
      UpaneliOpt=UpaneliTest1;
```

```
      }else{
      UpaneliOpt=UpaneliTest2;
      }
      koProxMPPt=0;
   }


Sum4IzmWP=Sum4IzmWr;
FlRegPWM.Uopt=0;koRevUopt=0;
break;
}//while(1)вихід з регуляторів

if(FlRegPWM.BlUgPWM==0){
   if(Uakb>UakbMax){
      FlRegMPPT.MPPT=0;
      koProxMPPt=0;
      if(CCPR5>0){
         --CCPR5;
         }
      }else{
      CCPR5=PWMpan;
      }
   }

UpaneliP=Upaneli;
TokBatZarP=TokBatZar;

if(FlRegPWM.Uopt==0){
   FlRegPWM.PWMstab=0;
   if(koRevUopt>=KkoRevUopt){
      koRevUopt=0;
      FlRegPWM.Uopt=1;
      Sum4IzmW=Wpaneli;
      Sum4IzmPWM=CCPR5;
      IndSum4I=0;
      }else{
      FlRegPWM.Sum4IzmW=0;
      }
   }else{  //if(FlRegPWM.Uopt==1)
   koRevUopt=0;
   if(FlRegPWM.Sum4IzmW==1){         //якщо вже було
      FlRegPWM.Sum4IzmW=0 ;         //починаємо новий вимір
      IndSum4I=0;
      Sum4IzmW=Wpaneli;
      Sum4IzmPWM=CCPR5;
      }else{
      Sum4IzmW+=Wpaneli;
      Sum4IzmPWM+=CCPR5;
      ++IndSum4I;
      if(IndSum4I>=3){
         Sum4IzmWr=Sum4IzmW>>2;
         Sum4IzmPWMr=Sum4IzmPWM>>2;
         FlRegPWM.Sum4IzmW=1;
         if(FlRegPWM.PWMstab==0){
```

```
            FlRegPWM.PWMstab=1;
            PWMstab=Sum4IzmPWMr;
            }
          }
        }
    }//if(FlRegPWM.Uopt==1){



if(IndbufTest>=kGlbufTest){
   IndbufTest=0;
   }
bufTest[IndbufTest]=Upaneli;
++IndbufTest;
bufTest[IndbufTest]=PWMpan;
++IndbufTest;
bufTest[IndbufTest]=Wpaneli;
++IndbufTest;

}//if(FlagADC.Upaneli==1&FlagMPPT.Vr==1)
}//PWM_MPPT_AKB1()


void Statistika1cek(void){                    //резуьтати за 1сек
   Wnak1Sek+=Wvix;                            //потужність на АКБ в сотнях міліват
   Wnak1Sek+=WAtest;
   Anak1Sek+=TokBatZar;
   Anak1Sek+=WAtest;
   ++indBufSt1cek;
   if(indBufSt1cek>=glBufSt1cek){
      indBufSt1cek=0;
      W1Sek=Wnak1Sek/glBufSt1cek;              //для дісплею
      Wnak1Min+=W1Sek;
      Wnak1Sek=0;
      A1Sek=Anak1Sek/glBufSt1cek ;             //
      Anak1Min+=A1Sek;
      Anak1Sek=0;
      ++indBufSt1min;
//       if(indBufSt1min==glBufSt1min/2){
//          Anak1Sek=0;
//          }
      if(indBufSt1min>=glBufSt1min){           //резуьтати за 1min
         indBufSt1min=0;
         W1Min=Wnak1Min/glBufSt1min;
         Wnak1Min=0;
         Wnak1Chas+=W1Min;
         A1Min=Anak1Min/glBufSt1min;
         Anak1Min=0;
         Anak1Chas+=A1Min;
         ++indBufSt1chas;
         if(indBufSt1chas>=glBufSt1chas){        //результати за 1 годину
            indBufSt1chas=0;
            W1Chas=Wnak1Chas/glBufSt1chas;
            Wnak1Chas=0;
```

```
        WvirObsh+=W1Chas;
        W1Chas=0;
        A1Chas=Anak1Chas/glBufSt1chas;
        Anak1Chas=0;
        AvirObsh+=A1Chas;
        }//if(indBufSt1chas>=glBufSt1chas)
      }//if(indBufSt1min>=glBufSt1min)
   }


WvirObshCh = WvirObsh;                    //загальна вироблена потужність в міліват
AvirObshCh = AvirObsh;
if(indBufSt1chas!=0){
   TekUint=Wnak1Chas/glBufSt1chas;
   WvirObshCh+=TekUint;
   TekUint=Anak1Chas/glBufSt1chas;
   AvirObshCh+=TekUint;
   }
if(indBufSt1min!=0){
   TekUint=Wnak1Min/(glBufSt1min*glBufSt1min);                    //glBufSt1min*2
   WvirObshCh+=TekUint;
   TekUint=Anak1Min/(glBufSt1min*glBufSt1min);
   AvirObshCh+=TekUint;
   }
if(indBufSt1cek!=0){
   TekUint=Wnak1Sek/(unsigned)(glBufSt1min*glBufSt1min*indBufSt1cek);
   WvirObshCh+=TekUint;
   TekUint=Anak1Sek/(unsigned)(glBufSt1min*glBufSt1min*indBufSt1cek);
   AvirObshCh+=TekUint;
   }

//TekUint=glBufSt1min+1;
//TekUint/=60;
//WvirObshACh+=TekUint;
//TekUint=Wnak1Sek/(glBufSt1cek-indBufSt1cek);
//TekUint/=3600;
//WvirObshACh+=TekUint;
}
```

ДОДАТОК Г

Код застосунку Wi-Fi – менеджеру

```
package com.example.gffcompose.data.managers

import android.Manifest
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import android.content.pm.PackageManager
import android.net.wifi.ScanResult
import android.net.wifi.SupplicantState
import android.net.wifi.WifiConfiguration
import androidx.core.app.ActivityCompat
import kotlinx.coroutines.channels.awaitClose
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow
import kotlinx.coroutines.flow.flow

@Suppress("DEPRECATION")
class WifiManager(private val applicationContext: Context) {

    private val wifiManager =
        applicationContext.applicationContext.getSystemService(Context.WIFI_SERVICE) as WifiManager

    fun startScan() {
        if (!wifiManager.isWifiEnabled) {
            wifiManager.isWifiEnabled = true
        }
        wifiManager.startScan()
    }

    fun observeScanResults(context: Context): Flow<List<ScanResult>> = callbackFlow {
        val wifiScanReceiver = object : BroadcastReceiver() {
            override fun onReceive(context: Context, intent: Intent) {
                val scanResults =
                    if (ActivityCompat.checkSelfPermission(context,
                            Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
                    ) {
                        return
                    } else wifiManager.scanResults
                if (scanResults != null) trySend(scanResults)
            }
        }

        val intentFilter = IntentFilter()
        intentFilter.addAction(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION)
        context.registerReceiver(wifiScanReceiver, intentFilter)
        awaitClose { context.unregisterReceiver(wifiScanReceiver) }
    }

fun connectToNetwork(targetNetwork: ScanResult, networkPassword: String, context: Context) {
```

```
    val wifiConfig = WifiConfiguration()
    wifiConfig.SSID = targetNetwork.SSID
    wifiConfig.status = WifiConfiguration.Status.DISABLED
    wifiConfig.priority = 40
    val networkSecurity = getWifiSecurity(targetNetwork)
    when (networkSecurity) {
      "Open network" -> {
        wifiConfig.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
        wifiConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);
        wifiConfig.allowedProtocols.set(WifiConfiguration.Protocol.WPA);
        wifiConfig.allowedAuthAlgorithms.clear();
        wifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
        wifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP40);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP104);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
      }
      "WPA", "WPA2" -> {
        wifiConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);
        wifiConfig.allowedProtocols.set(WifiConfiguration.Protocol.WPA);
        wifiConfig.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA_PSK);
        wifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
        wifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP40);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP104);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
        wifiConfig.preSharedKey = "\"" + networkPassword + "\""
      }
      "WEP" -> {
        wifiConfig.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.NONE);
        wifiConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);
        wifiConfig.allowedProtocols.set(WifiConfiguration.Protocol.WPA);
        wifiConfig.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.OPEN);
        wifiConfig.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.SHARED);
        wifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
        wifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP40);
        wifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.WEP104);

        if (getHexKey(networkPassword)) wifiConfig.wepKeys[0] = networkPassword;
        else wifiConfig.wepKeys[0] = "\"" + networkPassword + "\""
        wifiConfig.wepTxKeyIndex = 0;
      }
    }
    var netId = wifiManager.addNetwork(wifiConfig)


    wifiManager.disconnect()
    wifiManager.enableNetwork(netId, true)
    wifiManager.reconnect()
}
```

```
    fun getIPAddress() = wifiManager.connectionInfo.ipAddress

    private fun getHexKey(s: String?): Boolean {
        if (s == null) {
            return false
        }
        val len = s.length
        if (len != 10 && len != 26 && len != 58) {
            return false
        }
        for (i in 0 until len) {
            val c = s[i]
            if (c >= '0' && c <= '9' || c >= 'a' && c <= 'f' || c >= 'A' && c <= 'F') {
                continue
            }
            return false
        }
        return true
    }

    fun getWifiSecurity(scanResult: ScanResult): String {
        val capabilities = scanResult.capabilities

return when {
            capabilities.contains("WPA3-SAE") -> "WPA3"
            capabilities.contains("WPA3-") -> "WPA3"
            capabilities.contains("WPA2-") -> "WPA2"
            capabilities.contains("WPA-") -> "WPA"
            capabilities.contains("WEP") -> "WEP"
            else -> "OPEN"
        }
    }

    fun wifiConnectionStateFlow(context: Context): Flow<Boolean> = flow {
        val wifiManager =
            context.applicationContext.getSystemService(Context.WIFI_SERVICE) as WifiManager
        val wifiInfo = wifiManager.connectionInfo
        val isConnected =
            wifiInfo != null && wifiInfo.networkId != -1 && wifiInfo.supplicantState ==
SupplicantState.COMPLETED
        emit(isConnected)
    }
}
```