TempGatheringSystem

Artur Siepietowski, Maciej Hełmecki

9/15/2017

Sprawozdanie przygotowane na przedmiot Programowanie Mikrokontrolerów i Mikroprocesorów. Projekt nie należy do zadań podanych przez prowadzącego, ale wyróżnia się za to większą złożonością i nadkładem pracy potrzebnym do przygotowania. Przedstawiony projekt ma realne zastosowanie

Spis Treści

1. Wprowadzenie

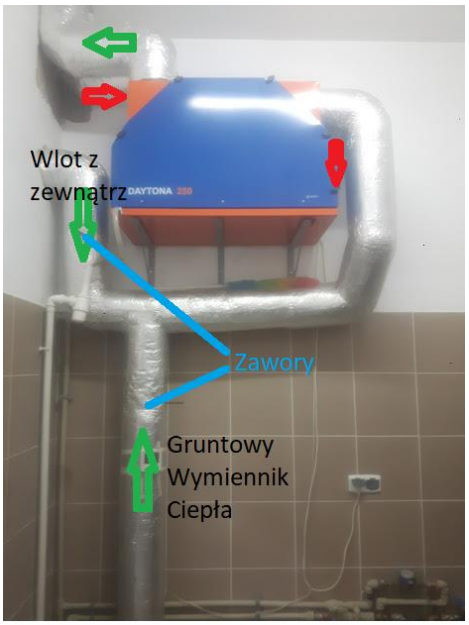
TempGatheringSystem to urządzenie zbudowane w całości przez studentów Informatyki 2 roku z wydziału WEAIiIB AGH. Zbiegiem okoliczności stało się, że projekt miał być w założeniu stworzony na Arduino podobnie jak projekty z przedmiotu Programowanie Mikrokontrolerów i Mikroprocesorów. Ma ono na celu pobieranie informacji o temperaturze układu rekuperacji w gospodarstwie domowym. W tym celu wykorzystujemy 6 czujników temperatury, 1 czujnik temperatury i wilgotności, wyświetlacz czytnik kart SD oraz inne potrzebne części elektroniczne.

2. Docelowy system rekuperacji

Docelowy system rekuperacji bazuje na urządzeniu niemieckiej firmy Berluf model Daytona 250. Jest urządzaniem dedykowanym do pomieszczeń użyteczności zarówno publicznej jak i mieszkalnej. Zadaniem centrali jest doprowadzenie powietrza świeżego z zewnątrz oraz odprowadzeniem zużytego z pomieszczeń przy jednoczesnym odzysku energii cieplnej. Centrala charakteryzuje się bardzo wysokim współczynnikiem odzysku ciepła do 95%. System wyposażony jest we 2 wloty powietrza, z możliwością ich zamknięcia:

- Wlot poprowadzony przez gruntowy wymiennik ciepła
- Wlot poprowadzony bezpośrednio z zewnątrz,

Jeden wylot zużytego powietrza oraz przewody wentylacyjne rozprowadzone w odpowiednich miejscach w budynku.



W związku z tym czujniki temperatury powinny być umieszczone w następujących miejscach

3. Części składowe systemu TGS.

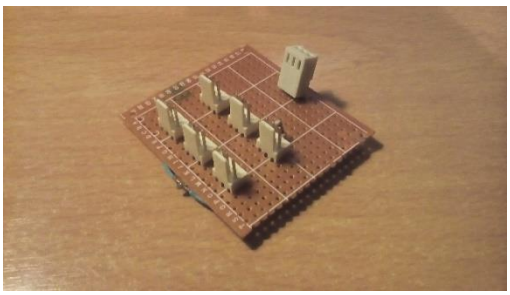
Chciałbym przedstawić elementy układu oraz moduły wchodzące w skład całego urządzenia.

a) Arduino Uno

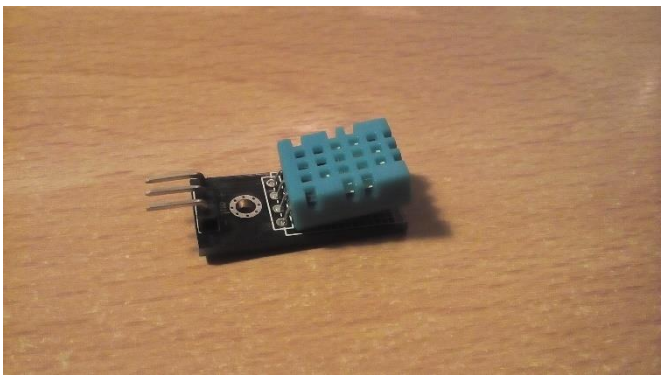
Popularny moduł z mikrokontrolerem AVR ATmega328 w wymiennej obudowie. Posiada 32 kB pamięci Flash, 2 kB RAM, 14 cyfrowych wejść/wyjść z czego 6 można wykorzystać jako kanały PWM, 6 wejść analogowych oraz popularne interfejsy komunikacyjne. W naszym projekcie wykorzystaliśmy 2 interfejsy szeregowe: I2C (wyświetlacz) oraz SPI (czytnik kart SD) poprzez złącze ICSP.

b) Czujnik temperatury DS18B20 x6

Czujnik temperatury DS18B20 to cyfrowy czujnik na magistrali one-wire. Działający w zakresie -55 °C do 125 °C. Jeden z tańszych czujników, ale jego ceną jest niska dokładność, którą producent podaje na +/- 0,5 °C w zakresie -10 °C do 85 °C. Zdecydowaliśmy się na ten czujnik, gdyż Arduino Uno (6 pinów analogowych) z którego korzystam nie miałoby dostatecznie dużo pinów by obsłużyć czujniki analogowe wraz z magistralą I2C o której wspomnę w ramach wyświetlacza LCD. Wszystkie 6 czujników nalutowane jest na 3 żyłowy kabel o długości 25 centymetrów zakończony wtyczką 3 pinową. Elementy narażone na zalanie umieszczone są w opaskach termokurczliwych. A wszystkie czujniki można podłączyć do tej oto płytki:



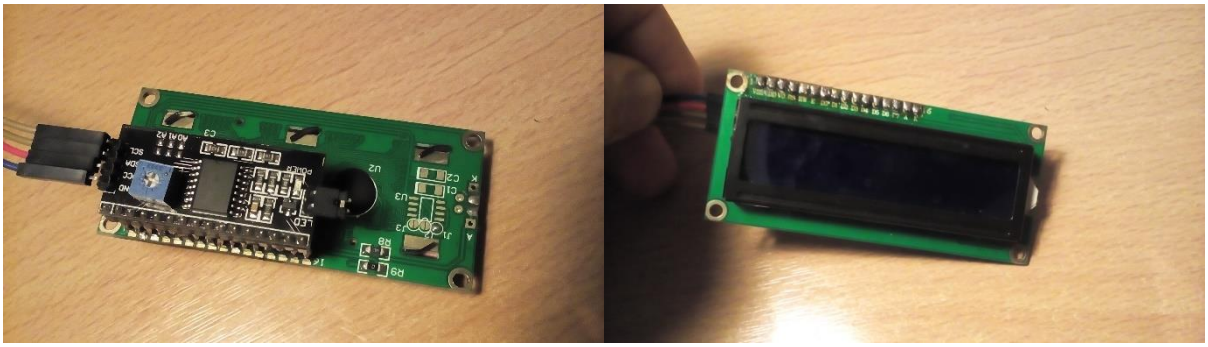
c) Czujnik temperatury i wilgotności DHT11



Popularny czujnik temperatury i wilgotności powietrza z interfejsem cyfrowym, jednoprzewodowym. Zakres pomiarowy: temperatura 0 - 50 °C, wilgotność 20-90 %RH. Nie będę się rozwodził nad tym czujnikiem. Udostępniam dokumentację. Czujnik znajduje się w obudowie co odpowiada warunkom w pomieszczeniu.

d) Wyświetlacz LCD 16x2 oraz konwerter LCM1602

Takie rozwiązanie pozwala nam zaoszczędzić 4 cyfrowe piny kosztem obsługi magistrali I2C. Jej interfejs obsługiwany jest przez konwerter LCM1602, który nie dość, że redukuje ilość pinów to wygląda estetycznie i idealnie wpasowuje się w kształt obudowy. Sam konwerter posiada potencjometr do regulacji kontrastu oraz zworkę do włączenia/wyłączenia podświetlania wyświetlacza.

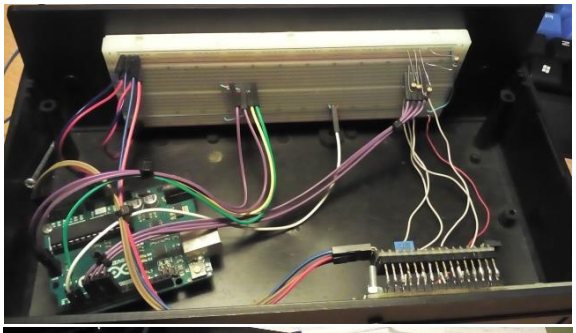


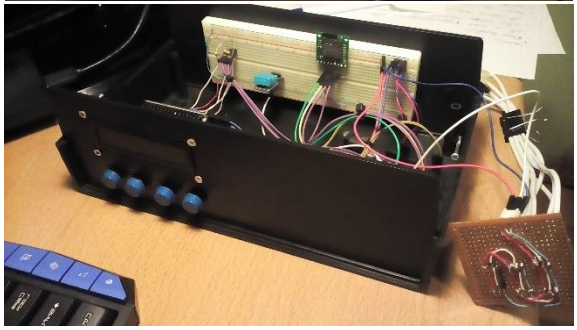
e) Moduł czytnika kart microSD z konwerterem napięć 5V oraz karta SD

Moduł czytnika kart micro SD firmy Pololu ze złączami goldpin. Komunikuje się z płytką główną poprzez interfejs SPI, posiada wbudowany regulator i konwerter, dzięki temu pracuje z napięciem 5V. Zdecydowaliśmy się na ten czytnik kart z powodu ceny oraz wbudowanego konwertera napięć. Jak wiadomo karty SD działają z napięciem 3.3V. Zależało nam na zapewnionej przez producenta bezawaryjności.

f) Obudowa i złożony projekt

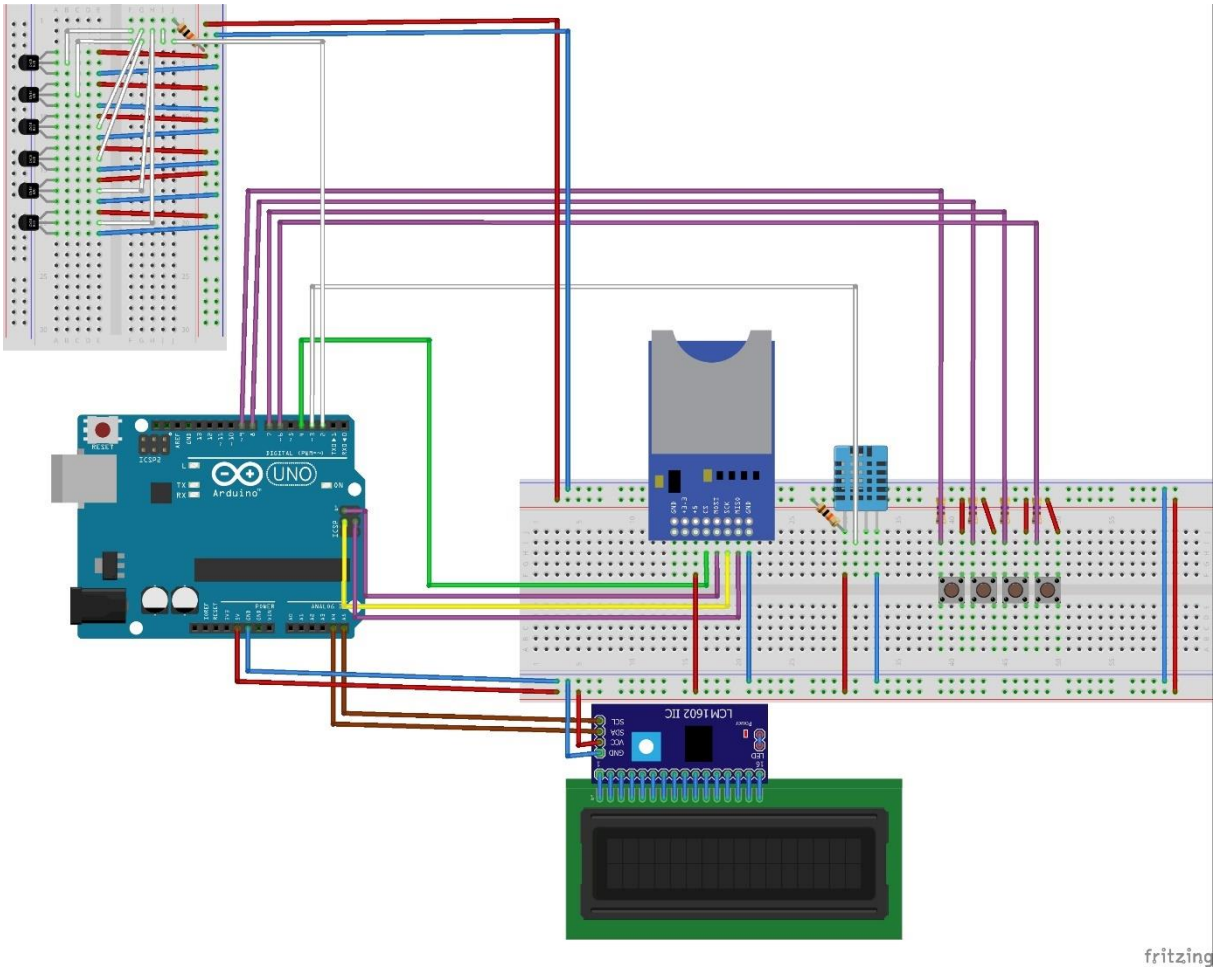
Cały układ zamknięty jest w plastikowej obudowie o grubości 1mm. Zostały nawiercone otwory na 4 przyciski wyświetlacz LCD, kabel zasilający oraz wyjście czujników temperatury DS18B20. Umieszczenie całości w kompaktowej obudowie jest wygodne, a dodatkowo bezpieczne, gdyż do tej pory cały układ zbudowany jest na płytce prototypowej oraz przewodach połączeniowych. Więcej na temat rozbudowy napiszemy w rozdziale „6. Możliwości rozbudowy”.





4. Schemat połączeń płytki i modułów

Schemat przygotowany w programie fritzing. W związku z tym można przeprowadzić w nim symulacje systemu wgrywając kod. Sam plik .fzz pozwoliliśmy sobie dodać do dodatkowego archiwum na wypadek chęci sprawdzenia autentyczności.

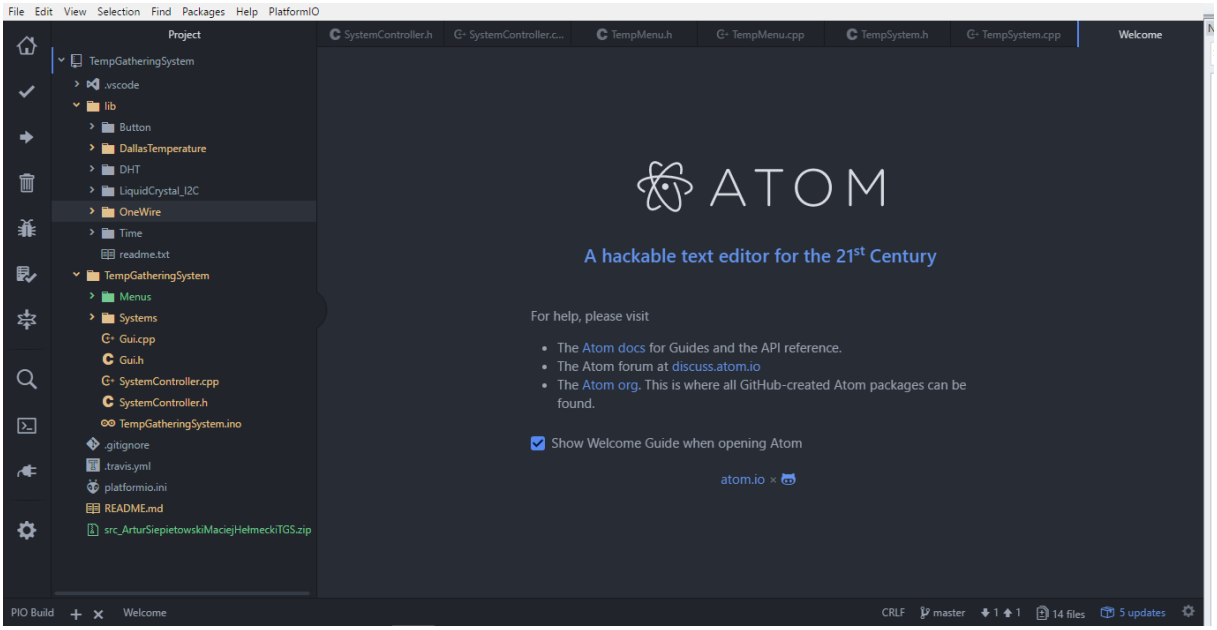


5. Przegląd kodu

Cały projekt był pisany w IDE Atom, wraz z wgranym pluginem PlatformIO. Dzięki temu mieliśmy większą kontrolę nad wgrywaniem programu. Ponad to zależało nam na przeniesieniu programu na plugin PlatformIO w celu debugowania. Wadą Arduino jest jednak brak możliwości debugowania w czasie rzeczywistym. Jedynym rozwiązaniem jest stosowanie portu szeregowego.

Cały kod napisany jest obiektowo w c++, przez co naszym zdaniem jest czytelniejszy, a prędkość programu jest wystarczająca.

Projekt został podzielony na 2 główne foldery: „lib" dla bibliotek oraz „TempGatheringSystem" dla naszego kodu. Atom daję tą wygodę, że bibliotek nie trzeba wgrywać do folderu domowego Arduino tylko umieszczone są w folderze projektu. Poniżej przedstawiamy drzewo projektu oraz samo IDE.



a) Wykorzystywane Biblioteki

Na cele projektu wykorzystaliśmy 5 bibliotek stworzonych przez inne osoby. Są to odpowiednio:

- DallasTemperature wraz z OneWire – biblioteka do obsługi czujników DS18B20,
- DHT – biblioteka do obsługi czujników DHT11 oraz DHT22. Jest napisana w sposób obiektowy przez co zapewnia bardzo wygodny interfejs,
- LiquidCrystal_I2C – biblioteka do obsługi wyświetlacza LCD wraz konwerterem LCM1602, interfejsem nie różni się od standardowej biblioteki Arduino LiquidCrystal,
- Time – biblioteka obsługująca czas. Działa ona na zasadzie obliczania czasu od uruchomienia procesora. Nie mamy możliwości pobierania tych informacji z sieci. W rozdziale „6. Możliwość Rozbudowy" wspominamy w jaki sposób można dodać kartę Wifi wprowadzając cały projekt na nowy poziom.

Nie będziemy rozpisywać się na temat wszystkich powyższych bibliotek. Są one ogólne dostępne na githubach swoich autorów. Zamiast tego przejdziemy do kodu, który stworzyliśmy sami. Ostania biblioteka „Button" jest stworzona przez nas. Wstępnie jako klasa, ale pozwoliliśmy sobie ją wrzucić do folderu z bibliotekami.

i) Button

BUTTON.h

```
#ifndef BUTTON_H
#define BUTTON_H

class Button{
        private:
                int m_pin;
                bool m_state;
        public:
                Button();
               Button(int pin);
                Button(Button& other);
                ~Button();
                void checkIfPushed();
                bool isPushed();
};

#endif
```

BUTTON.cpp

```
#include<Button.h>

#if ARDUINO >= 100
  #include "Arduino.h"
#else
extern "C" {
    #include "WConstatets.h"
}
#endif

Button::Button() {
  m_pin = -1;
  m_state = false;
  Serial.print("Konst bezparam, adres: ");
  Serial.println((int)this);
}

Button::Button(int pin) {
        pinMode(pin, INPUT);
        m_pin = pin;
        m_state = false;
  Serial.print("Konst param, adres: ");
  Serial.println((int)this);
}

Button::Button(Button& other) {
  m_pin = other.m_pin;
  m_state = other.m_state;
  pinMode(m_pin, INPUT);
  Serial.print("Konst kop, adres: ");
  Serial.println((int)this);
}

Button::~Button() {}

void Button::checkIfPushed() {
        if((m_state == false) and (digitalRead(m_pin) == HIGH)) {
                delay(30);
                m_state = true;
                while(digitalRead(m_pin) == HIGH);
```

```
                delay(30);
        }
}

bool Button::isPushed() {
        if(m_state == true) {
                m_state = false;
                return true;
        }
        else {
                return false;
        }
}
```

Cały pomysł na zbieranie informacji polega na tym, że w głównej pętli programu wywoływana jest funkcja `checkIfPushed();` dla każdego obiektu Button. W związku z tym, że taka iteracja trwa setne części sekundy, a samo naciśniecie przycisku co najmniej sekundę, funkcja `checkIfPushed();` jest w stanie ustawić flagę, a na tą flagę można zareagować sprawdzając co zwróci `isPushed();`. Reszta klasy to konstruktory.

b) Klasy

Teraz przechodzimy do kodu samego programu (folder „TempGatheringSystem") i klas na które jest podzielony. Staraliśmy się użyć wzorca architektonicznego Model-View-Controller. Mniej więcej nam sie to udało. Wszystkie Menu zawarte są w folderze „Menus", a systemy (modele) w „Systems". Ponad to mamy główny plik

`TempGatheringSystem.ino`

```
#include <Arduino.h>
#include <Wire.h>
#include "SystemController.h"

void setup() {
  Serial.begin(9600);
  if(Serial) {
    Serial.println("--- Uruchomienie Systemu (setup) ---");
  }
  SystemController systemController;
  systemController.loop();

}

//The loop is fulfilled by SystemController method
void loop() {
}
```

W tym kodzie odpalamy port szeregowy na o 9600 bitach na sekundę transmisji. Jeżeli został odpalony wypisujemy informacje. Tworzymy obiekt klasy SystemController i odpalmy metodę `loop()` tejże. Zastępuje nam ona oryginalnego loopa Arduino. Ale w takim razie czym jest SystemController?

i) SystemController

`SystemContoller.h`

```
#ifndef SYSTEMCONTROLLER_H
#define SYSTEMCONTROLLER_H
```

```
#if ARDUINO >= 100
    #include "Arduino.h"
#else
extern "C" {
    #include "WConstants.h"
}
#endif

#include <Button.h>
#include "Menus/MainMenu.h"
#include "Menus/StateMenu.h"
#include "Menus/TimeMenu.h"
#include "Menus/TempMenu.h"
#include "Systems/TimeSystem.h"
#include "Systems/TempSystem.h"

class SystemController {
  private:
    //Inferface for lcd display
    Gui* view;
    //Buttons are
    //pin 6 - accept
    //pin 7 - left
    //pin 8 - right
    //pin 9 - back
    Button buttons[4];
    //Singletons
    TimeSystem timeSystem = TimeSystem::getInstance();
    TempSystem tempSystem = TempSystem::getInstance();
    //2 ways to check if the display should be refreshed
    String time;          //After every minute
    uint16_t cycleCounter = 0;      //After 10000 cycle of the loop
    uint8_t TempCounter;
    void creteMenuFromMainMenu();
  public:
    SystemController();
    ~SystemController();
    //Main loop of the system
    void loop();
};

#endif
```

Jest to klasa, która obsługuje całość systemu. Mamy tutaj obiekt `Gui*` który jest interfejsem do obsługi wyświetlacza LCD. Obiekty przycisków. Obiekty systemów, które w tym przypadku są singletonami by mieć pewność, że istnieje tylko jeden unikatowy system. Główną metodą tej klasy jest **`void`** `loop();`. To ona sprawdza czy przyciski zostały naciśnięte, liczy cykle po których ma nastąpić odświeżenie wyświetlacza, sprawdza czy nie nastąpił, żadne alarm, oraz obsługuje interfejs Gui. Przejdźmy może do niego

ii) Gui

`Gui.h`

```
#ifndef GUI_H
#define GUI_H

#if ARDUINO >= 100
```

```
    #include "Arduino.h"
#else
extern "C" {
    #include "WConstants.h"
}
#endif

#include <LiquidCrystal_I2C.h>
#include "Systems/TimeSystem.h"
#include "Systems/TempSystem.h"

class Gui {
  private:
    LiquidCrystal_I2C lcd{0x27,16,2};
    TimeSystem* timeSystem = &TimeSystem::getInstance();
    TempSystem* tempSystem = &TempSystem::getInstance();
  public:
    Gui();
    virtual ~Gui();
    void guiprint(int8_t col, int8_t row, String text);
    void guiBlink(int8_t col, int8_t row);
    TimeSystem* getTimeSystem();
    TempSystem* getTempSystem();
    virtual int8_t getMenuID() = 0;
    virtual void increase() = 0;
    virtual void decrease() = 0;
    virtual int8_t accept() = 0;
    virtual int8_t undo() = 0;
    virtual void refreshScreen() = 0;
};

#endif
```

Gui to klasa czysto abstrakcyjna. Jako pola posiada 2 wskaźniki na systemy: `TimeSystem` i `TempSystem` oraz obiekt wyświetlacza `lcd`. Są tutaj 2 istotne metody `guiprint`, `guiBlink` to one są zapewniają funkcjonalność dla klas dziedziczących. Pierwsza z nich wypisuje stringa w odpowiednie miejsce podane jako argument. Druga sprawia, że podane miejsce mruga. Dodatkowo powiemy tylko po co są metody `increase()`, `decrease()`, `accept()`, `undo()`. To właśnie je `SystemController` wywołuje w przypadku naciśnięcia odpowiedniego przycisku. Przez to, że muszą być przeciążone, gdyż są wirtualne wywoływane są metody klasy dziedziczącej po `Gui`. Wydaje mi się, że reszta kodu powinna być jasna, a nazwy wymowne.

iii) Menus

Wszystkie klasy menus są do siebie podobne, dlatego też przybliżymy tylko jedną będzie to:

`MainMenu.h`

```
#ifndef MAINMENU_H
#define MAINMENU_H

#if ARDUINO >= 100
    #include "Arduino.h"
#else
extern "C" {
    #include "WConstants.h"
}
#endif
```

```
#include "Gui.h"


class MainMenu : public Gui {
  private:
    int8_t menuID = 0;
    int8_t numberOfMenu = 3;
    int8_t currentMenu = 0;
    void refreshTopLine();
    void refreshBottomLine();
  public:
    MainMenu();
    ~MainMenu();
    virtual int8_t getMenuID() override;
    virtual void refreshScreen() override;
    virtual void increase() override;
    virtual void decrease() override;
    virtual int8_t accept() override;
    virtual int8_t undo() override;
};

#endif
```

Tak jak poprzednio nazwy pól i metod powinny być wymowne.

MainMenu.cpp

```
#include "MainMenu.h"

MainMenu::MainMenu( ) {
  refreshScreen();
}

MainMenu::~MainMenu() {}

void MainMenu::refreshTopLine() {
  switch(currentMenu){
    case 0:
      guiprint(0,0,"[Stan]Temp Czas ");
      break;
    case 1:
      guiprint(0,0," Stan[Temp]Czas ");
      break;
    case 2:
      guiprint(0,0," Stan Temp[Czas]");
      break;
  }
  return;
}

void MainMenu::refreshBottomLine() {
  guiprint(1,1,getTimeSystem()->getWholeDate());
}

void MainMenu::refreshScreen() {
  refreshTopLine();
  refreshBottomLine();
}
```

```
int8_t MainMenu::getMenuID() {
  return menuID;
}

void MainMenu::decrease() {
  Serial.println("Left");
  currentMenu = (currentMenu + 2) % numberOfMenu;
  refreshScreen();
}

void MainMenu::increase() {
  Serial.println("Right");
  currentMenu = (currentMenu + 1) % numberOfMenu;
  refreshScreen();
}

int8_t MainMenu::accept() {
  Serial.println("Enter");
  return currentMenu;
}

int8_t MainMenu::undo() {return -1;}
```

W zależności co zwraca `accept()` takie zostanie wybrane menu: 0 – StateMenu, 1 – TempMenu, 2 – TimeMenu. Dzieje się to poprzez utworzenie dynamicznie odpowiedniej klasy i wpisanie jej adresu pod wskaźnik `gui`. Tą sprawą zajmuje się `SystemController`. Podobny cel ma metoda `undo()`.

iv) Systems

Tutaj podobnie jak z Menu przybliżymy tylko jeden System. Za czas zarządzaniem czasem odpowiedzialny jest:

`TimeSystem.h`

```
#ifndef TIMESYSTEM_H
#define TIMESYSTEM_H

#if ARDUINO >= 100
    #include "Arduino.h"
#else
extern "C" {
    #include "WConstants.h"
}
#endif

#include <Time.h>

class TimeSystem {
  private:
    TimeSystem();
  public:
    static TimeSystem & getInstance();
    String getWholeDate();
    String getTime();
    String conv(int timeUnit);
    void modify(int flag, bool increase);
};
```

```
#endif
```

TimeSystem.cpp

```
#include "TimeSystem.h"



TimeSystem::TimeSystem() {
  Serial.println("Time System Construktor");
}

TimeSystem & TimeSystem::getInstance() {
  static TimeSystem singleton;
  return singleton;
}

String TimeSystem::getWholeDate() {
  return( conv(day()) + ":" +
           conv(month()) + ":" +
           conv(year()%100) + " " +
           getTime());
}

String TimeSystem::getTime() {
  return( conv(hour()) + ":" +
          conv(minute()));
}

String TimeSystem::conv(int timeUnit) {
  if (timeUnit / 10 == 0) {
    return "0" + String(timeUnit);
  }
  return String(timeUnit);
}

void TimeSystem::modify(int flag, bool increase) {
  int temp[5] = {0,0,0,0,0};
  int unit = 0;
  if(increase){
    unit = 1;
  } else {
    unit = -1;
  }
  temp[flag] = unit;
  setTime((24+hour()+temp[0])%24,    (60+minute()+temp[1])%60,    second(),
day()+temp[2], (13+month()+temp[3])%13, year()+temp[4]);
}
```

Tak jak zaznaczałem wcześniej wszystkie systemy są singletonami, co daje nam pewność istnienia tylko jednego obiektu. Zrobiliśmy to tworząc prywatny konstruktor i dodając metodę, która zwraca nam statyczny obiekt systemu. Metoda `conv(int)` ma zastosowanie w przypadku, gdy daty jest przykładowo jedno cyfrowa, powiedzmy 1.1.17. Po użyciu metody na dniu i miesiącu zostanie wyświetlona data 01.01.17. Metoda `modify(int, bool)` ma zastosowanie podczas ręcznej zmiany daty przy użyciu przycisków.

6. Możliwości rozbudowy

Kończąc sprawozdanie chcielibyśmy przedstawić przykładowe możliwości rozbudowy i elementy, które można byłoby poprawić:

- Układ wytrawiany/Shield Arduino – duża ilość kabli umieszczonych na płytce prototypowej to proszenie się o problemy. Cały układ (czujników, czytników, rezystorów z łatwością można byłoby umieścić na zlutowanej płytce lub układzie wytrawianym. Jest również możliwość kupienia shielda na arduino własnego projektu.
- Wifi – Zamiast bawić się w zapisywanie danych na karcie SD, lepszym rozwiązaniem jest przesyłanie informacji o stanie urządzenia do sieci. Dostarcza to również możliwość automatycznego odświeżania czasu. Do tego celu można zastosować inną wersję płytki Arduino np. Arduio YUN. Inną możliwością jest wyposażenie układu w szybszy procesor lub płytkę ARDUINO MEGA 2560 i zakupienie modułu wifi.
- Większa ilość czujników o lepszych parametrach – zamiast DTH11 można użyć DTH22 o znacznie lepszych parametrach
- Backup zasilania – w przypadku odcięcia zasilania sieciowego, przydatny byłby backup zasilania w postaci baterii.
- Bardziej intuicyjny interfejs – Wyświetlacz LCD 16x2 ma swoje ograniczenia. Nie można na nim wyświetlić wykresów, a poruszanie po menu może wydawać się siermiężne. Można doposażyć się w wyświetlacz o większych rozmiarach np. 4x20. Najlepszym jednak rozwiązaniem byłoby użycie wyświetlacza o większej ilości pikseli co najmniej 128x160px.

7. Zakończenie

Na skład sprawozdania będzie się składać:

- Ten oto plik PDF „doc_ ArturSiepietowskiMaciejHełmeckiTGS.pdf”
- Schemat Układu „Schemat_ArturSiepietowskiMaciejHełmeckiTGS.jpg” oraz „Schemat_ArturSiepietowskiMaciejHełmeckiTGS.fzz”
- Kod źródłowy wraz z wykorzystanymi bibliotekami „src_ArturSiepietowskiMaciejHełmeckiTGS.zip”

Z góry dziękujemy za przyjęcie naszego sprawozdania, mimo późnej pory.