

Dokumentacja do projektu z przedmiotu:

**Technologie Bezprzewodowe w Automatyce i Robotyce**

Sterownik BlueTooth

do Lampy LED

KwadraLampa

LP1 C1, EADI-3

Bełch Jakub

Rzeszów, IV 2017

1. Cel i zakres projektu

Projekt miał na celu wykonanie sterownika do pokojowej stojącej Lampy LED, można go również użyc do innych zastosowań. Zakres obejmował wykonanie firmwaru do mikrokontorolera oraz klienta na Android i PC. Z braku czasu na tę chwilę zrezygnowałem z wykonania klienta na PC.

Podstawowe założenia:

* właczanie/wyłaczanie lamp
* regulacja jasności
* właczanie/wylaczanie po zadanym czasie – TIMER
* właczanie/wyłaczanie o zadanej godzinie – CLOCK

Dodatkowo w trybach CLOCKi TIMER mozna ustawic opcje ze zamiast naglego wylaczenia po zadanym czasie lampa sciemnia/rozjasnia sie stopniowo do 0/255 w ciagu tego czasu.

Na razie w trybach *timer i clock*  działa tylko wyłączanie, muszę pomyśleć jak zrealizować to rozróżnienie

Komunikacja bezprzewodowa realizowana jest przez dowolny modul BT w trybie SPP

Cały projekt tutaj: <https://github.com/KubaMiszcz/KwadraLampa>

1. Ogólny opis rozwiązania

Zadanie było w miarę proste jednak wymagało troche czasu zeby wszystko dzialalo wygodnie. sterowanie realizowane jest w dwoch trybach MANUAL i BLUTUTU*. O*bsluga obu lamp jest identyczna.

* 1. tryb Manual:

Wykorzystalem dostepny fizyczny interfejs tzn dwie galki, z braku potencjometrow z wylacznikiem, wylaczanie polega na skreceniu potencjometrow na 0, regulacja polega na kreceniu galkami. Lampa ma byc prosta intuicyjna w oblsudze i wygodna dla postronnych osob dlatego nie mnozylem dalszych kontrolek

* 1. tryb Blututu:

Realizowany jest na zasadzie przesylanie kilkunastu komunikatow sterujacych plytka. Dostepne opcje:

* gaszenie/zapalanie przyciskiem
* regulacja jasnosci
* właczanie/wylaczanie po zadanym czasie – timer
* właczanie/wyłaczanie o zadanej godzinie – clock

Dodatkowo w trybach clock i timer mozna ustawic opcje ze zamiast naglego wylaczenia po zadanym czasie lampa sciemnia sie stopniowo do 0 w ciagu tego czasu (nieliniowo poniewaz jasnosc diody nie maleje liniowo wraz z pradem).

Przed podlaczeniem konieczne jest sparowanie urzadzenia z telefonem

Aby moc korzystac z trybu clock konieczna jest znajomosc rzeczywistego czasu, poniewaz nie jest to jakos szczegolnie wazne zrezygnowalem z jakiegos modulu podtrzymywania/pobierania go, a rozwiazalem to w ten sposob ze po polaczeniz Klientem na ktorym mamy aktualny czas jest on przesylany i uaktualniany w sterowniku,

Czas w sterowniku przekreca sie co ok 40 dni, i kasuje po zaniku zasilana, jednak przy sterowaniu recznym i tak jest zbedny a po polaczeniu z BT jest ustawiany wiec ten problem uznalem za rozwiazany

Komunikaty sterujace:

1. Komunikaty do sterownika

Długosc komunikatu 11znakow (wazne! aczkolwiek firmware dopelnia sam do 11 znakow jesli dostał krotszy komunikat, jesli dostal dluzszy, sprawdza ostatnie 11znakow)

Liczby lepiej byłoby przesylac jako bajty (krocej) ale potem trza by konwertowac na stringi, na etapie prototypowania prosciej wysylac jako stringi/char.

Tak samo tryby i w ogole komunikaty, docelowo lepiej przerobic na typ ENUM i wtedy wysylamy jako jeden bajt caly string – ale trzeba by wtedy pamietac w terminalu te "kody" trybow

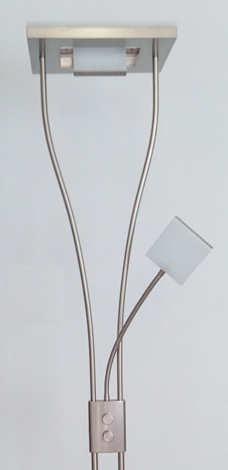
Spis komunikatów:

|  |  |
| --- | --- |
| Zapytania do sterownika: | |
| ::HM:? | Przeslanie aktualnego czasu ze sterownika do klienta |
| ::L1:? | Odsyla stan lampy do klienta |
| ::L1:DIM? | Odsyla stan opcji sciemniania do klienta |
| ::L1:BRGHT? | Odsyla aktualna jasnosc lampy |
|  | |
| Ustawienia sterownika: | |
| ::HM:***hhmm*** | Przeslanie aktualnego czasu do sterownika |
| ::L1:on  ::L1:off | Wyl/zal lampy 1 |
| ::L1:PWM***5***  ::L1:PWM***55***  ::L1:PWM***255*** | Przeslanie zadanej jasnosci |
| ::L1:DIMon  ::L1:DIMoff | Wyl/zal sciemnianie opcji sciemniania dla trybu timer i clock |
| ::L1:CL**hhmm**  ::L1:CL**hhmm** | Wyl/zal lampy o zadanej godzinie |
| ::L1:TIM***5***  ::L1:TIM***55***  ::L1:TIM***255*** | Wyl/zal lampy po zadanym czasie |
| ::L1:TIcncl | Anulowanie timera |
| ::L1:CLcncl | Anulowanie clock |
| ::L1:TGLon | Po zadanym czasie/godzine lampa sie wlacza |
| ::L1:TGLoff | Po zadanym czasie/godzine lampa sie wylacza |

Analogicznie dla lampy L2 ,Szczegolowy opis firmware – dalej

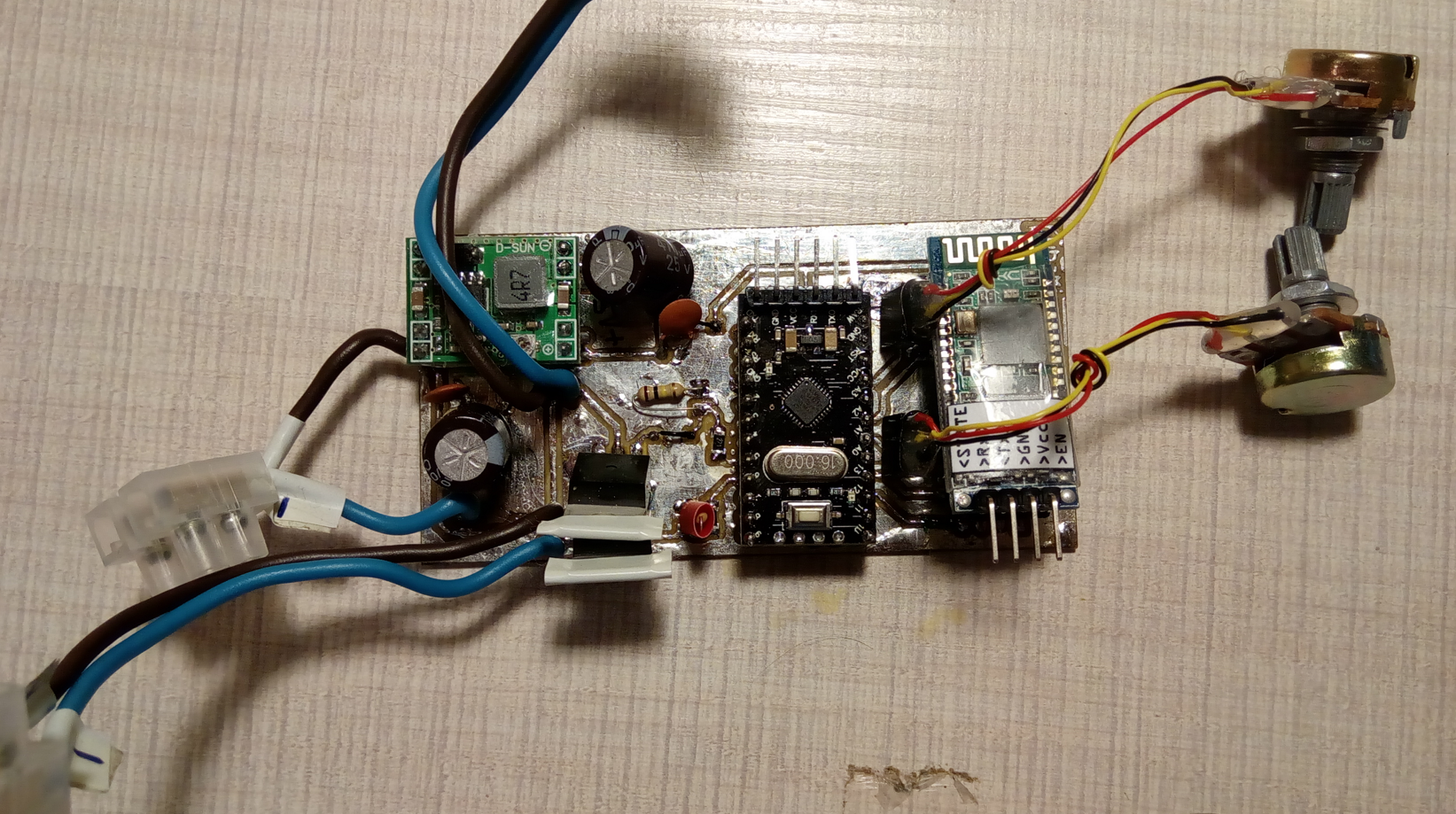
1. Sprzęt
   1. Lampa

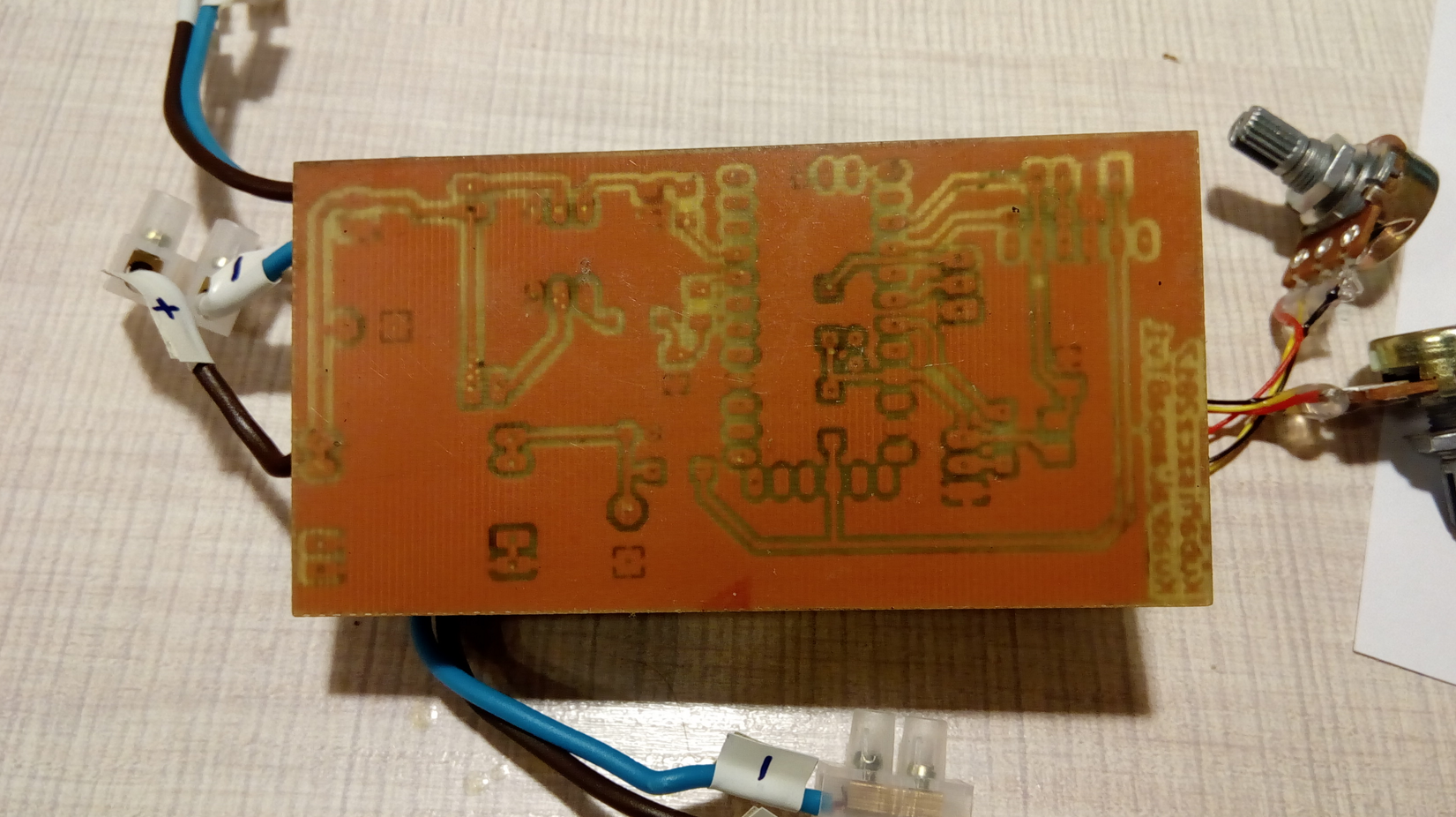
Lampa sklada sie z dwoch modułów LED 20W + 4W, sterowanie kazda lampa osobno, wylaczenie lampy realizowane jest przez skrecenie jasnosci do 0, lub ustawienie jej programowo na 0 (zatkanie tranzystora MOSFET)



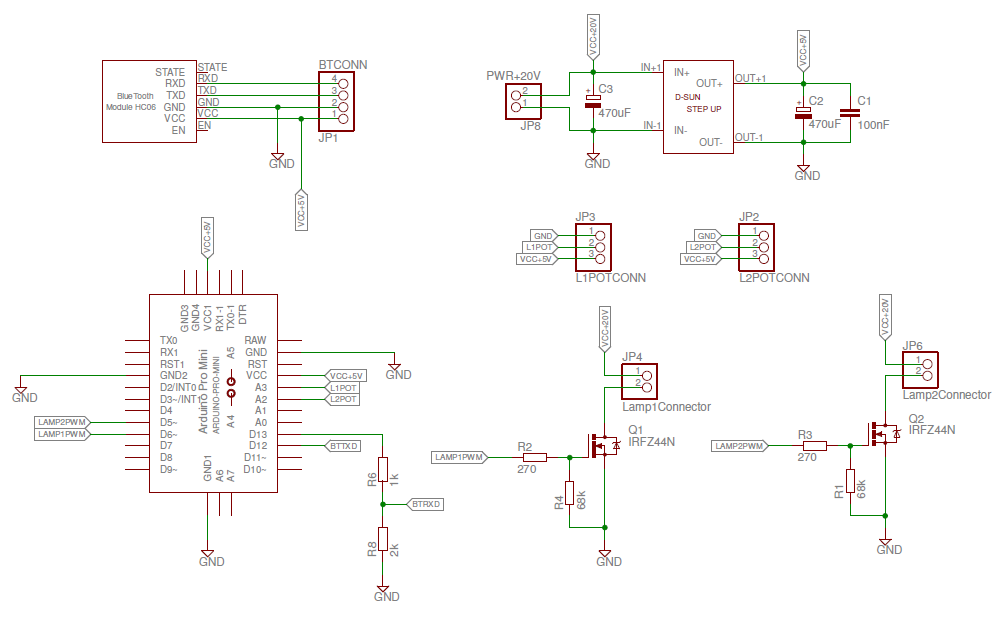
* 1. Płytka PCB

W zwiazku z zasilaniem 20V i maksymalnym prądem ok 1A konieczne bylo zastosowanie tranzystora sterowanego PWM (0-255 ok. 1kHz) jako regulatora, wybor padl na MOSFET IRFZ44N gdyz jest w miare uniwersalny, miałem go pod ręka, dla takich parametrów prawie sie nie grzeje (nie trzeba radiatora), i pobiera duzo mniej mocy niz analogiczny bipolarny.

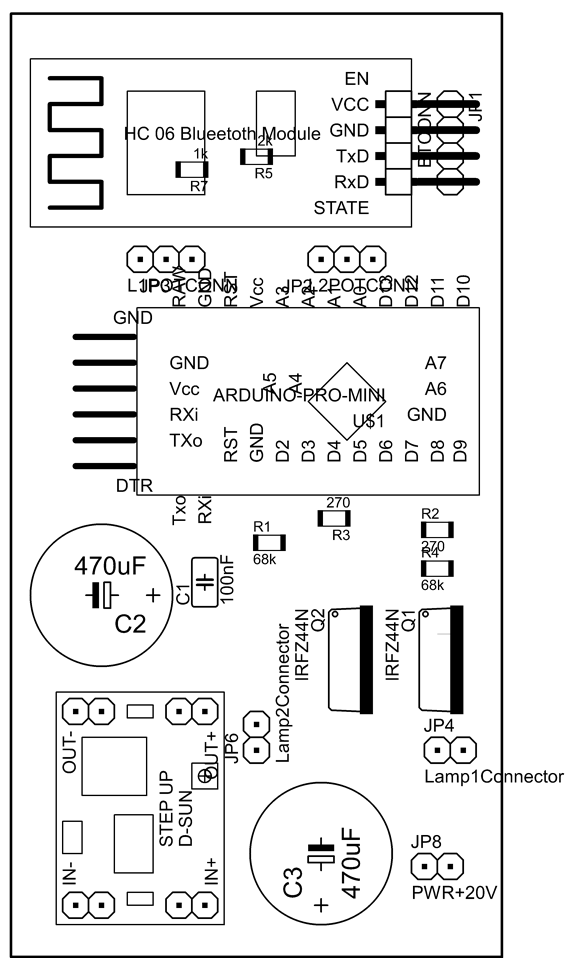
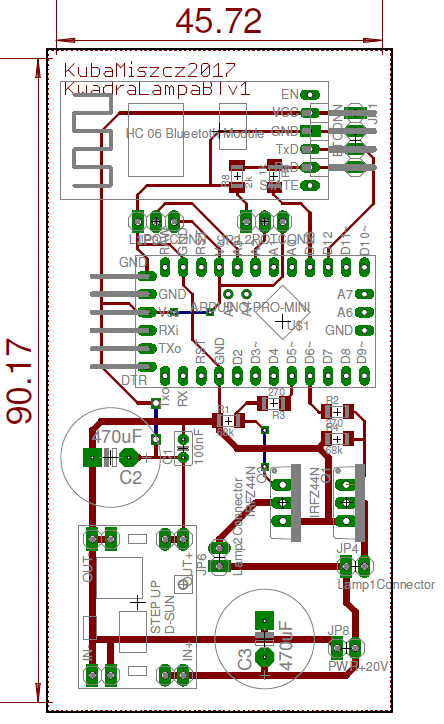
Logikę zrealizowałem za pomocą mikrokontrolera Atmega8A z bootloaderem Arduino, redukcje napiecia z Arduino na poziom 3.3V (RxD w HC-05) za pomoca dzielnika R 3x10kohm. widok zmontowanej płytki



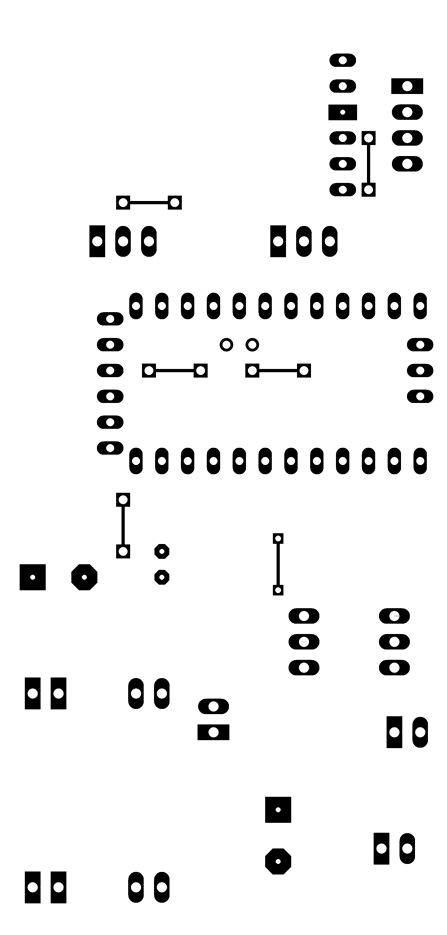
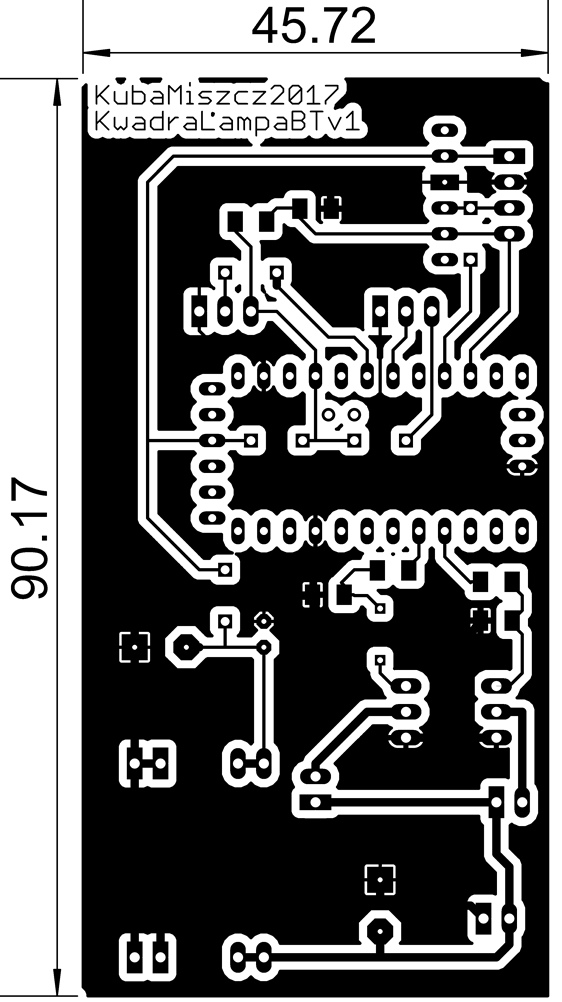
widok od spodu



schemat ideowy



widok elementów

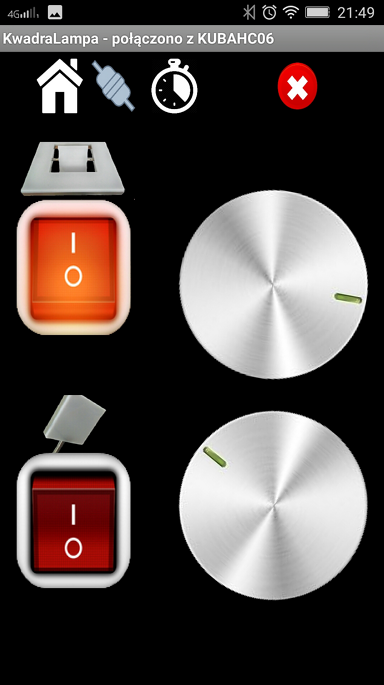
widok warstw TOP i BOTTOM

warstwa TOP nie byla drukowana, a mostki zostaly dolutowane na BOTTOM

HC-05 mozna wypinac w celu ewentulanej rekonfiguracji, podobnie mozna podpiac sie do pinow arduino w celu aktualizacji firmware

Poniewaz wiercenie jest uciazliwe elementy przewlekane rowniez zrealizowalem jako SMD, nie pozwala to na druk etykiet od strony elementow, jednak sprawe rozwiazuje dolozenia kartki z opiska elementow na plytce (w razie poprawek za dluzszy czas)

* 1. Klienty
* Android – telefon z androidem, apka w AppInventor

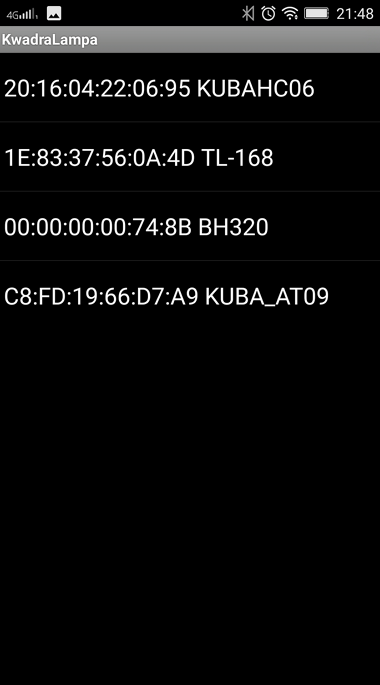
 

ekran glowny

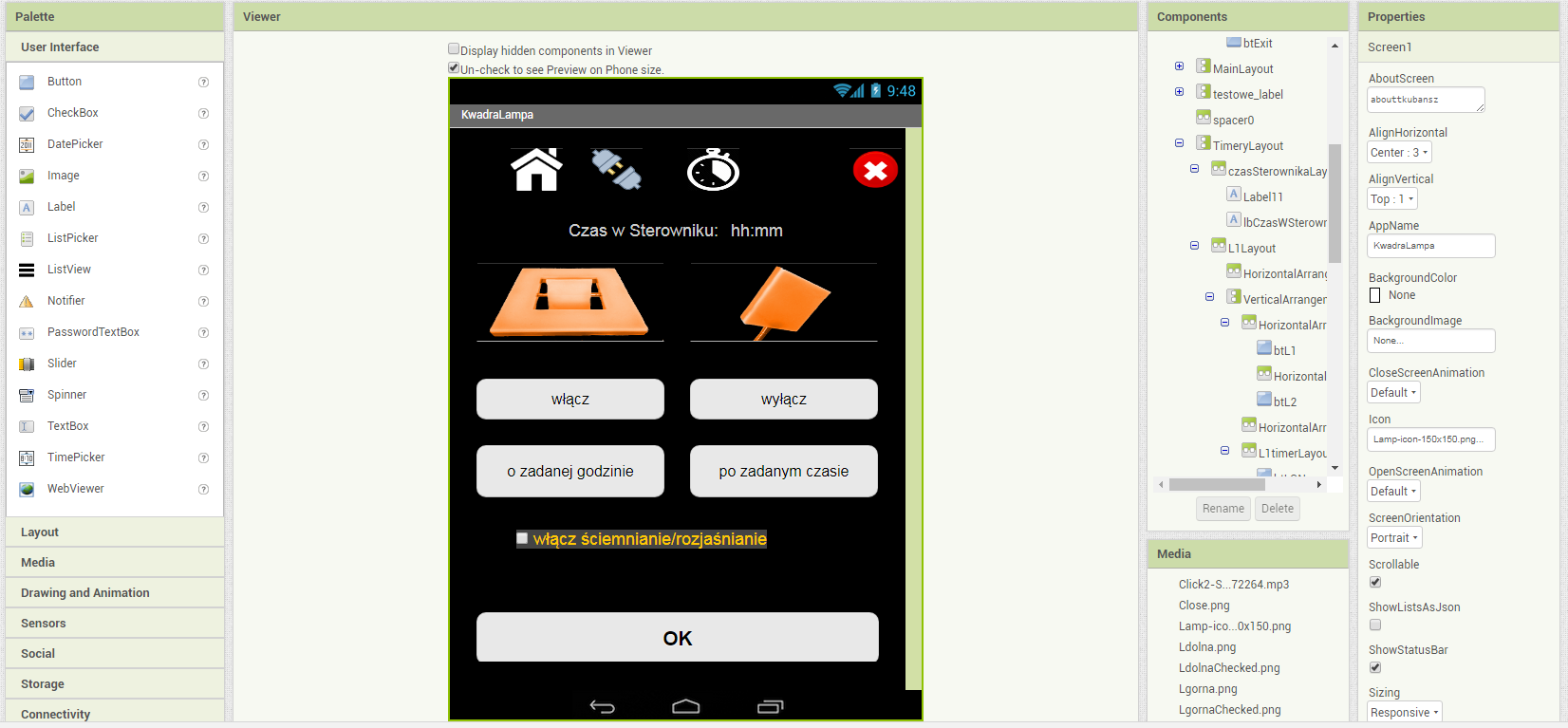
Poniewaz (najprawdopodobniej) w AppInventor jest na sztwyno zapisany interwał między wyslaniem komunikatow, konieczne bylo zrobienie sobie nibybufora, ktory przy szybkim klikaniu dopisuje je do listy stringow, i wysyla co 1sek usuwajac z listy, zapobiega to gubieniu komunikatow, lub wysylaniu dluugiej listy, ktora trzeba by dzielic w sterowniku, co nie byloby trudne do zrobienia jednak glownym powodem bylo to ze w przypadku regulacji jasnosci sposob ten bylby niedobry gdyz przy zmianie o 100jednostek trzeba byloby czekac 100sek, gdybysmy brali ostatnia wartosc gubilibysmy inne komunikaty przy szybkim klikaniu, aktualnie sterowanie jasnoscia polega na wysylaniu ostatniego komunikatu, natomiast pozostale komuniakty wysylane sa przez bufor.

Layoutu i jego parametrów nie bede opisywal, gdyz sa to proste ustawienia kolorow, czcionek, rozmieszczenia elementow w kontenerach itp, tym bardziej ze AppInventor ma bardzo ubogi i uciazliwy tryb edycji tych elementow, i w celu dalszego rozwijania klienta zdecydowalem sie przepisac go od nowa w Xamari/C#.

Podobnie obiekt BTclient istnieje tylko w jednym ekranie i trzeba przesylac dane (nie ma do neigo dostepu z innej activity/ekranu), rozwiazalem to zmieniajac atrybut Visible poszczegolnych layoutow (wszystkie sa taknaprawde na jednym ekranie)



lista wyboru nadajnikow BT i obsługa timerów



przykladowe okno edytora

* 1. Obsługa Klienta

Przed uzyciem trzeba sparowac urzadzenie w ustawieniach BT telefonu.

Po kliknieciu na ikonke z wtyczka pojawi sie lista sparowanych klientow, po kliknieciu nastapi proba polaczenia i powrot do ekranu glownego.

Po połączeniu mozna operowac kontrolkamni, regulacja jasnosci aktulazuje jasnosc w sterowniku jednak lampa wlacza sie dopiero po jej wlaczeniu wlacznikiem, podobnie efekt zmiany wystepuje tylko gdy lampa jest wlaczona

Przy przekreceniu galaka sterownik przechodzi w tryb reczny i aktualizuje jasnoc dla ustawione na lampie a nie telefonie.

w trybie timer nalezy po kolei wybrac lampe, czy ma sie wlaczyc/wylaczyc, w jakim trybie (jednoczesnie otwiera sie kontrolka wyboru czasu), oraz ewentualny tryb plynnej jasnosci, i zatwierzdzic OK.

Poniewaz jasnosc moze zmieniac sie najmniej o 1 (int), wiec aby zmieniala sie jak najplynniej ten interwal czasowy wyliczany jest na podstawie nastawionego czasu i aktualnej jasnosci, mimo to czasem zdarza sie ze sterownik przegapi kilka krokow(jak sie okazuje 16MHz to nie tak szybko), dlatego pilnuje tego i czasem w razie potrzeby zmienia jasnosc o kilka kroków (szczegoly w listingu firmware). Na koniec czasu jednak osiaga zadana wartosc.

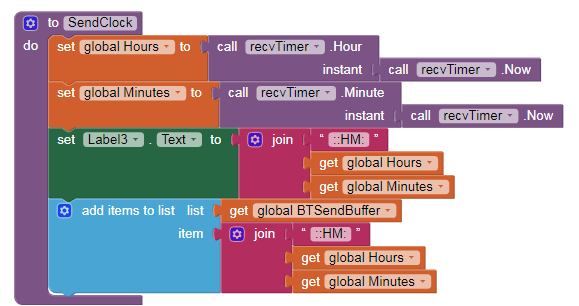
Zmiany na ekranie glownym i sterowanie ręcznie anuluja timery.

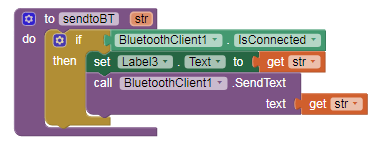
Za pomoca wbudowanych komunikatow mozna wykonac w zasadzie wszystkie operacje z lampkami dlatego w przyszlosci wymagana moze byc tylko aktualizacja klientów bez modyfikacji i demontazu plytki.

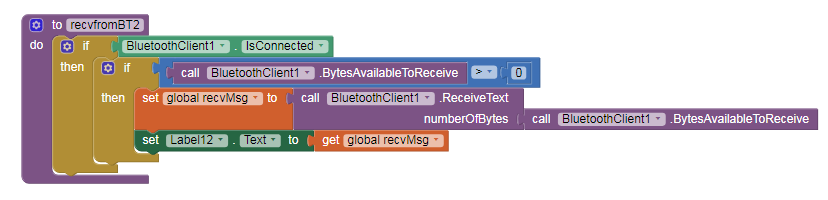
cały program

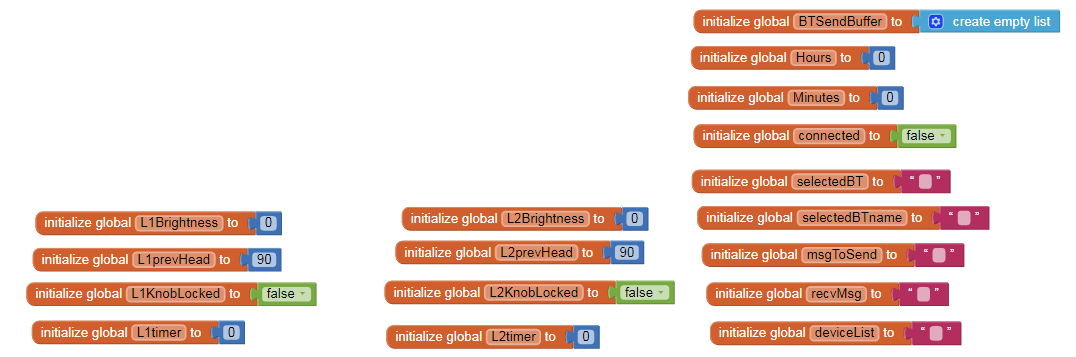


obsluga BT i bufor wysylajacy i oprozniajacy liste o wyslany element



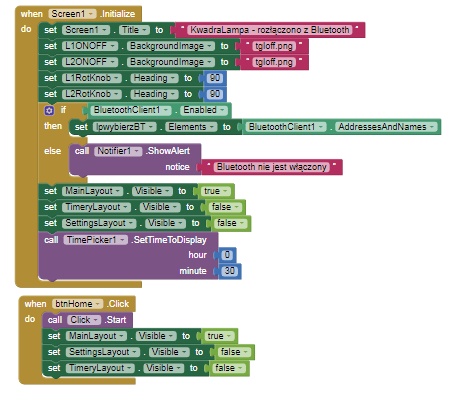
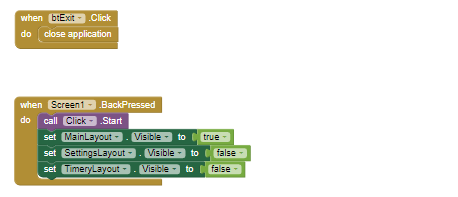


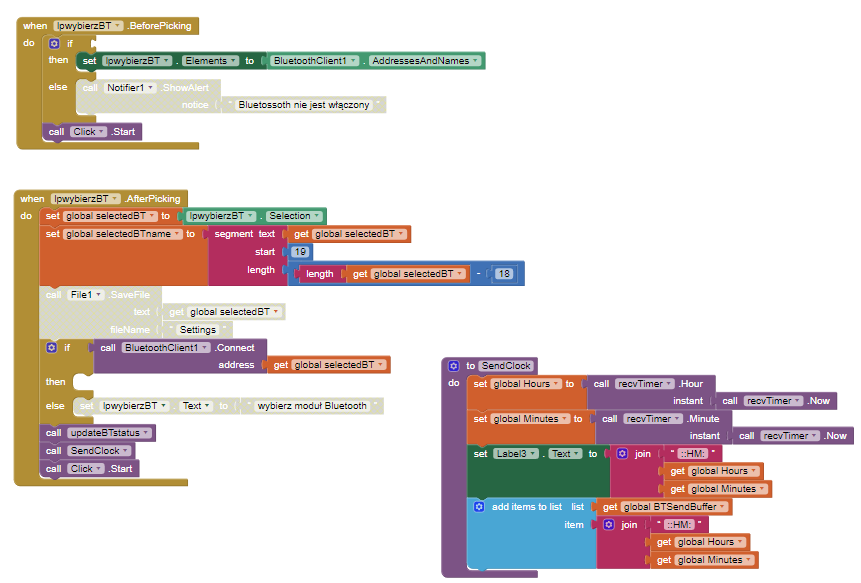
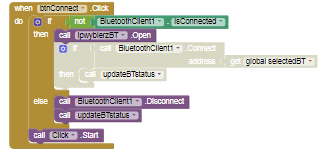




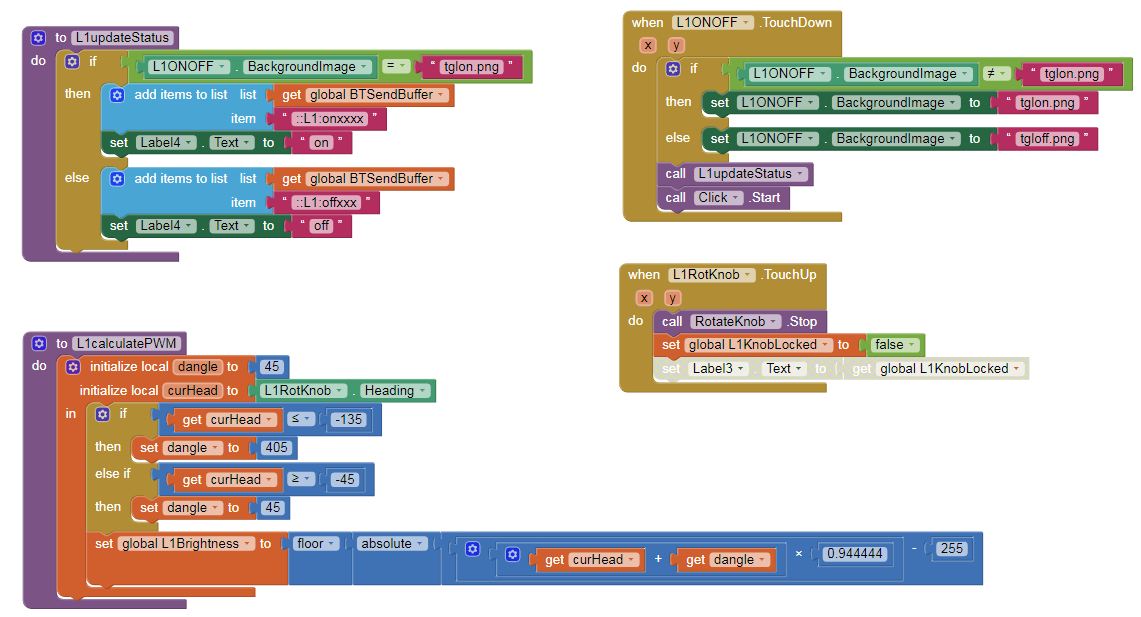
zmienne globalne

obsługa glownego okna

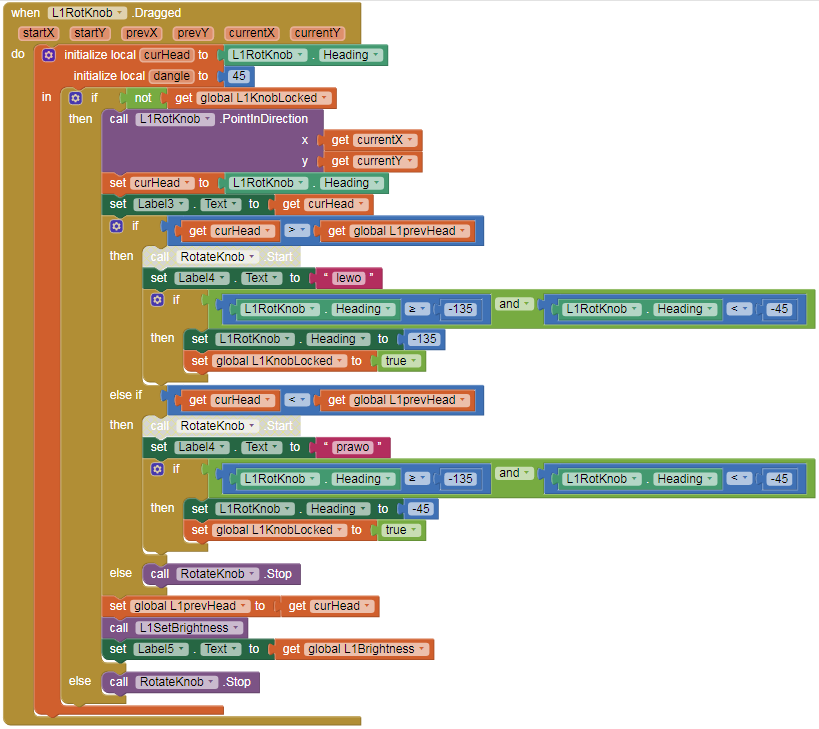


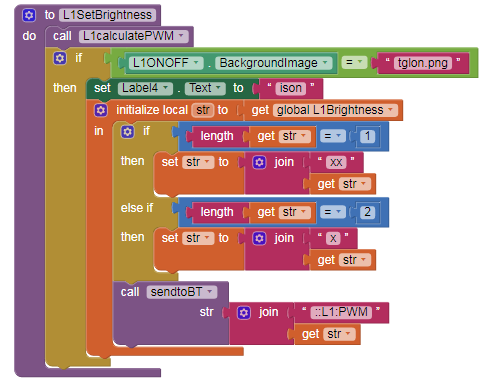


obsluga lampy

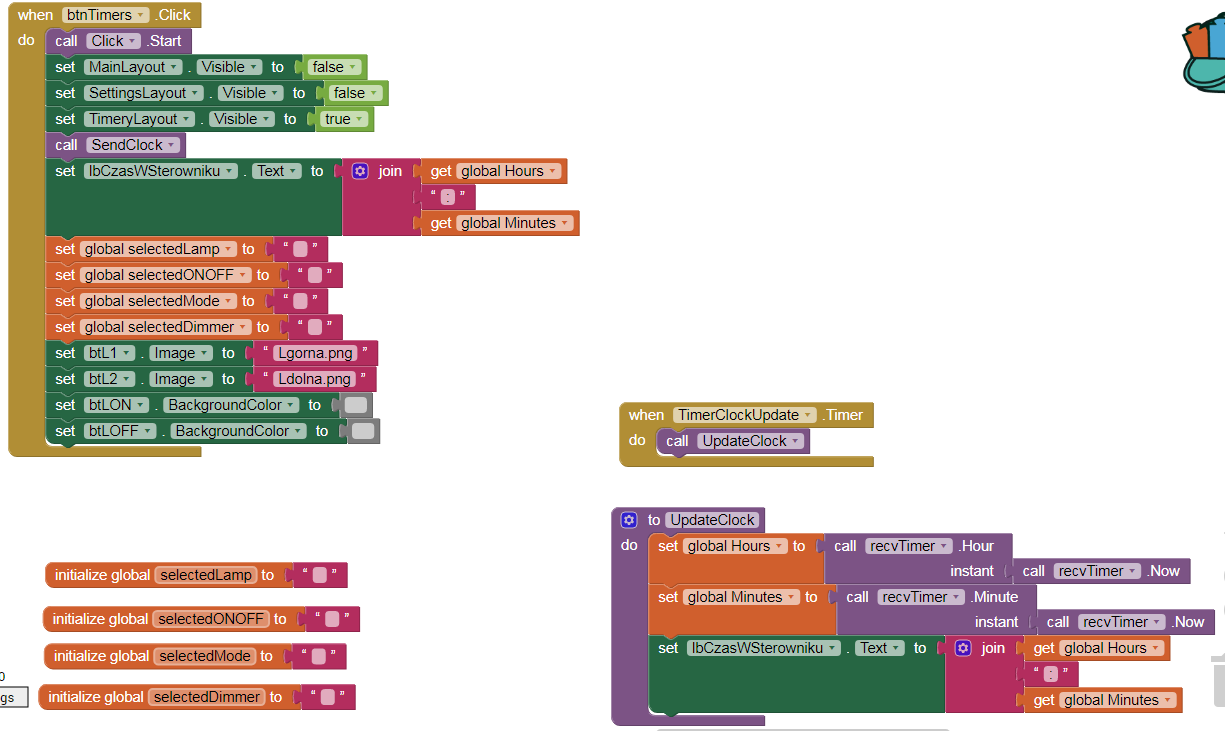


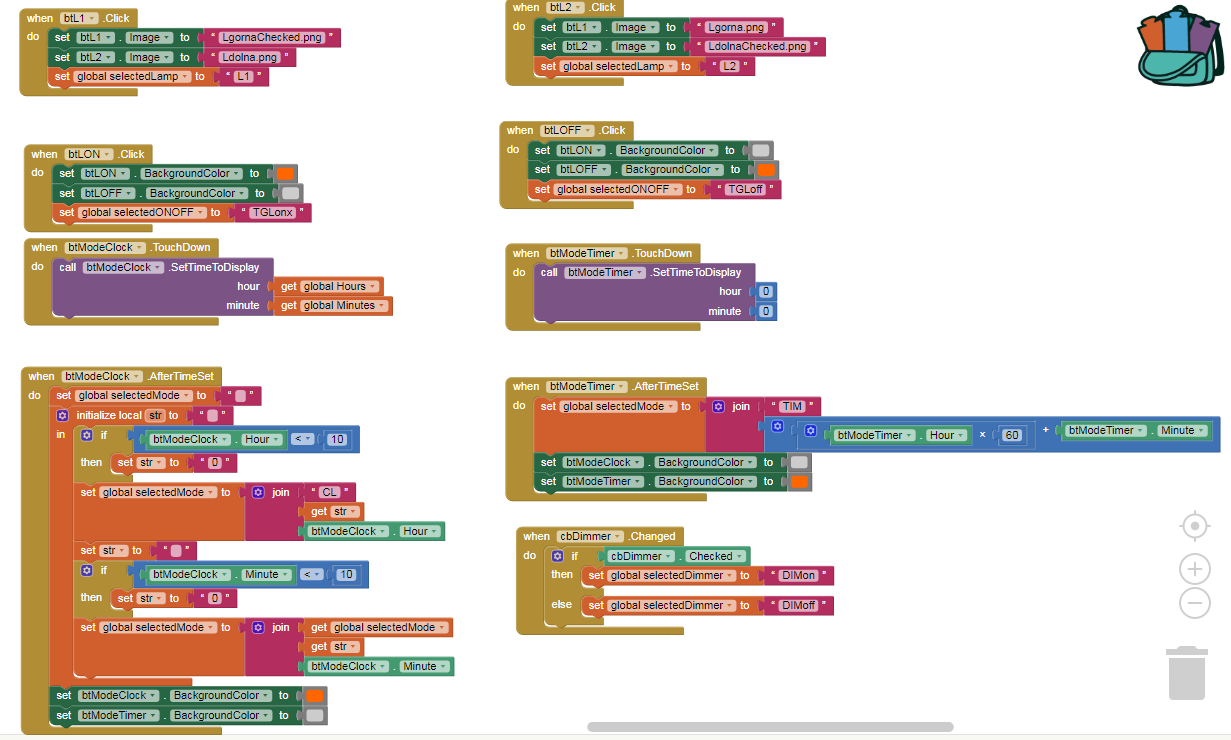
oblsuga regulacji jasnosci

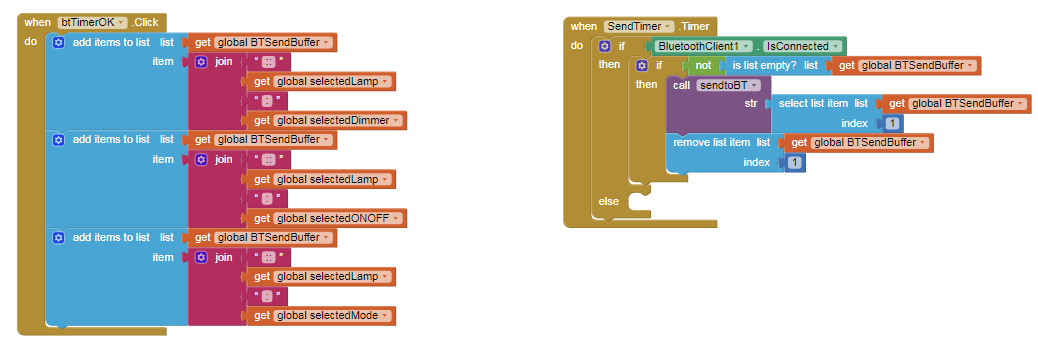




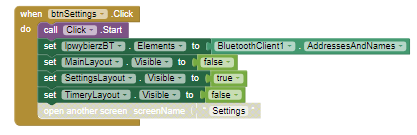
oblsuga timerow







obłsuga ustawien (w trakcie)



* PC – dowolny terminal RS, lub program np w C# + kontrolka SerialPort
  1. Firmware

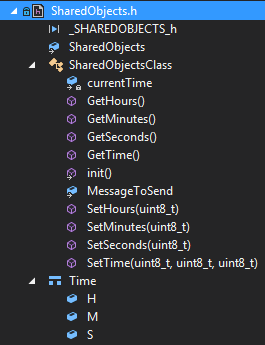
Platforma Arduino, C++, programowanie obiektowe. Wiekszosc jest wyjasniona w komentarzach w kodzie.

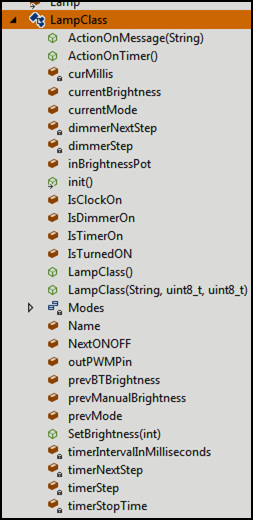
Ogolnie firmware posiada klase Lampa, i dwa obiekty tej klasy, zestaw metod do nich, plus pomocnicze statyczne metody w SharedObjectsClass, enumerator trybow, operacje na czasie itp.

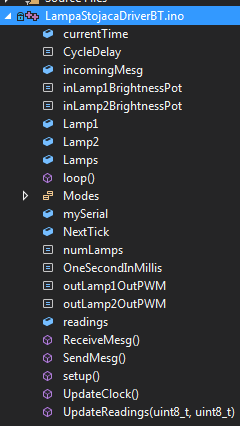
podczas dzialania sprawdza stan na pinach. po wykryciu prawidlowego komunikatu przechodzi w tryb BT, nie obowiazuja wtedy ustawienia galek, po wykryciu zmiany na galkach przestaja obowiazywac ustawienia BT (zerowane sa timery)

oblsuga trybu CLOCK jest zrealizowana za pomoca trybu timer – jest wyliczany czas miedzy zadana godizna i aktualna i wykonywana metoda dla TIMER

klasa Sharedobjects (statyczne metody i struktury pomocnicze)



główna klasa klasa LampClass



* 1. Listingi

// the setup function runs once when you press reset or power the board

#include <SoftwareSerial.h>

#include "Lamp.h"

#include "SharedObjects.h"

//#include <EEPROM.h>

//stale

const uint8\_t numLamps = 2;

enum Modes { Manual = 0, Blututu };

const unsigned long int OneSecondInMillis = 1000;

const unsigned int CycleDelay = 10;

//komunikaty IN

//const String MsgSetClock = "HM:";

//const String MsgGetClock = "HM:?";

//komunikaty OUT

//pinout

//Lampa 1 - gorna 20W

const uint8\_t inLamp1BrightnessPot = A2;

const uint8\_t outLamp1OutPWM = 6;

LampClass Lamp1("L1", inLamp1BrightnessPot, outLamp1OutPWM);

//!!!!!!!!!!

//nazwy musza byc L1 L2... bo tak jest potem w komunikatach na sztywno ustawione inna nazwa i komunikat nie zadziala

//!!!!!!!!!!!!!!

//Lampa 2 - gorna 4W

const uint8\_t inLamp2BrightnessPot = A1;

const uint8\_t outLamp2OutPWM = 5;

LampClass Lamp2("L2", inLamp2BrightnessPot, outLamp2OutPWM);

//software serial

SoftwareSerial mySerial(13, 12); // RX, TX

LampClass Lamps[] = { Lamp1,Lamp2 }; //3 zeby nie zaczynac od indeksu 1-L1, 2-L2

//zmienne

String incomingMesg = "";

int readings[] = { 0,0,0 };

unsigned long int NextTick;

//unsigned int Hours = 0, Minutes = 0, Seconds = 0;

Time currentTime;

//SharedObjectsClass SharedObjects;

void setup() {

Serial.begin(9600);

Serial.println("hello serial");

mySerial.begin(9600);

mySerial.println("hello Myserial v2");

pinMode(inLamp1BrightnessPot, INPUT);

pinMode(outLamp1OutPWM, OUTPUT);

pinMode(inLamp2BrightnessPot, INPUT);

pinMode(outLamp2OutPWM, OUTPUT);

NextTick = millis() + OneSecondInMillis;//+1 second

for (int i = 0; i < numLamps; i++)

{

UpdateReadings(i, Lamps[i].inBrightnessPot);

Serial.print("i= " + String(i));

Serial.print(" name= " + Lamps[i].Name);

Serial.print(" pin= " + String(Lamps[i].inBrightnessPot));

Serial.println(" read= " + String(readings[i]));

Lamps[i].SetBrightness(readings[i]);

Lamps[i].IsTurnedON = false;

Lamps[i].prevBTBrightness = Lamps[i].currentBrightness;

Lamps[i].prevManualBrightness = Lamps[i].currentBrightness;

}

}

//void loop2() {

// int read = analogRead(Lamp1.inBrightnessPot);//0-1023

// Serial.print(read); Serial.print(" ");

// int curManBrightness = map(read, 0, 1023, 0, 255);//0-255

// Serial.print(curManBrightness); Serial.print(" ");

// Lamp1.SetBrightness(curManBrightness);

// Serial.print(Lamp1.currentBrightness); Serial.print(" \n");

//}

// the loop function runs over and over again forever

void loop() {

delay(CycleDelay);

//########## dodaj pobieranie satusu zmiennych zeby poustwaiac kontrolki w apce przy polaczeniu

//przeyslanie timera stanow alampek jasnosci itp

String str = ReceiveMesg(); //odebranie wiadomosci z nadajnika

SendMesg();

UpdateClock();

//ReceiveMesg222();

//UpdateClockv2(&Seconds);

for (uint8\_t i = 0; i < numLamps; i++)

{

//##############################################

//zzaczynamy w trybie manual

//##############################################

#pragma region MANUAL currentMode

if (Lamps[i].currentMode == Modes(Manual)) //MANUAL

{ //przejsice w blututu

if (str.startsWith(Lamps[i].Name))

{

Lamps[i].currentMode = Modes(Blututu);

Lamps[i].prevMode = Modes(Manual);

Serial.println("curmode " + Lamps[i].Name + " =man>>BT"

+ String(Lamps[i].currentMode) + " "

+ String(Lamps[i].currentBrightness));

//zapamietanie ostatneigo brighness w MAN

UpdateReadings(i, Lamps[i].inBrightnessPot);

Lamps[i].prevManualBrightness = readings[i];

}

else { //zostajemy w manual

UpdateReadings(i, Lamps[i].inBrightnessPot);

//Serial.print("man analog read " + String(read));

//Serial.println(" man mapped analog readings[i] " + String(readings[i]));

Lamps[i].SetBrightness(readings[i]);

//if (Lamps[i].currentBrightness < 5)

//{

// //mySerial.println("::L1:PWM" + String(Lamp1.currentBrightness) + "\n");

//}

}

}

#pragma endregion

//##############################################

//a teraz jesli tryb BT //blutut

//##############################################

#pragma region Blutut Mode

if (Lamps[i].currentMode == Modes(Blututu)) //BLUTUT

{

Lamps[i].ActionOnTimer(); //tylko w btmode, w manual nie ma timera

//czy przejsc w man mode

UpdateReadings(i, Lamps[i].inBrightnessPot);

if (

(readings[i] > (Lamps[i].prevManualBrightness + 10)) //filtre bo ADC plywa moze dodoac jakies 100nF?

||

(readings[i] < (Lamps[i].prevManualBrightness - 10)))

{

Lamps[i].currentMode = Modes(Manual);

Lamps[i].prevMode = Modes(Blututu);

Lamps[i].SetBrightness(readings[i]);

Serial.println("curmode= " + Lamps[i].Name + " BT>>MAN"

+ String(Lamps[i].currentMode) + " "

+ String(Lamps[i].currentBrightness));

}

else if (str.startsWith(Lamps[i].Name)) {

Serial.println(str);

Lamps[i].ActionOnMessage(str);//a co z tym stringiem zrobic robi metoda w klasie

}

}

#pragma endregion

}

//##########KONIEC PRZELATYWANIA PO TABLICY LAMP

//#############ZAPYTANIA DO STEROWNIKA

//Przy przesylania pamietaj zeby dlugosc komunikatu byla 11 znakow

//dodaj te :: na poczatku, ale juz do rozpoznawanie jest bez tych ::

//###### ZAPYTANIA LUB USTAWIANIE PARAMETROW STEROWNIKA

if (str.startsWith("HM:")) { //HM:1234xx

if (str.startsWith("HM:?")) { //ktora godizna

mySerial.println("Clock: " + String(currentTime.H) + ":" + String(currentTime.M) + ":" + String(currentTime.S));

Serial.println("Clock send: " + String(currentTime.H) + ":" + String(currentTime.M) + ":" + String(currentTime.S));

}

else //ustawianie zegara

{

currentTime.H = str.substring(3, 5).toInt();

currentTime.M = str.substring(5, 7).toInt();//

SharedObjects.SetTime(currentTime.H, currentTime.M, currentTime.S);

Serial.println("Clock Updated: " + String(currentTime.H) + ":" + String(currentTime.M) + ":" + String(currentTime.S));

}

}

}

String ReceiveMesg() {

String str = "";

if (mySerial.available() > 0) {

str = mySerial.readString();

str.trim();

Serial.println("Received>> " + str); //echo na sterownik

if (str.startsWith("::")) {

if (str.length()>11) {

str = str.substring(str.length() - 9); //9 osttanich znakow, juz bez "::"

}

else {

str =str+"xxxxxxxxxxx"; //dopelnienie do 11 znakow

str = str.substring(2, 11);

}

}

else str = "";

}

else str = "";

return str;

}

void SendMesg() {

if (SharedObjects.MessageToSend != "") {

mySerial.println(SharedObjects.MessageToSend);

SharedObjects.MessageToSend = "";

}

}

//void ReceiveMesg222() {

// String str = "";

// if (Serial.available() > 0) {

// str = Serial.readString();

// Serial.println("Received222>> " + str);

// }

//}

void UpdateReadings(uint8\_t i, uint8\_t BrightnessPotentiometer) {

readings[i] = analogRead(BrightnessPotentiometer);//0-1023

readings[i] = map(readings[i], 0, 1023, 0, 255);//0-255

}

void UpdateClock() {

if (millis() > NextTick)

{

currentTime.S += 1;

if (currentTime.S >= 60)

{

currentTime.S = 0;

currentTime.M += 1;

Serial.println("CLK: " + String(currentTime.H) + ":" + String(currentTime.M) + ":" + String(currentTime.S));

if (currentTime.M >= 60)

{

currentTime.M = 0;

currentTime.H += 1;

if (currentTime.H >= 24) {

currentTime.H = 0;

}

}

}

SharedObjects.SetTime(currentTime.H, currentTime.M, currentTime.S);

//Serial.println("CLK: " + String(currentTime.H) + ":" + String(currentTime.M) + ":" + String(currentTime.S));

NextTick += OneSecondInMillis;

}

}

* 1. Lamp.h

// Lamp.h

#ifndef \_LAMP\_h

#define \_LAMP\_h

#if defined(ARDUINO) && ARDUINO >= 100

#include "arduino.h"

#include "SharedObjects.h"

#else

#include "WProgram.h"

#endif

class LampClass

{

public:

String Name;

uint8\_t inBrightnessPot;

uint8\_t outPWMPin;

uint8\_t currentBrightness;//0-255

uint8\_t currentMode; //enum "BT" or "Man"

uint8\_t prevMode;

bool IsTurnedON;

uint8\_t prevBTBrightness;//do sprawdzania czy zmienic mode

uint8\_t prevManualBrightness;//do sprawdzania czy zmienic mode

bool IsTimerOn;

bool IsClockOn;

bool IsDimmerOn;

bool NextONOFF; //0 off, 1 on

private:

enum Modes { Manual = 0, Blututu };

unsigned long int timerIntervalInMilliseconds;

unsigned long int timerStopTime;

unsigned long int timerStep = 1000;//3sek

unsigned long int timerNextStep;

unsigned long int dimmerStep;

unsigned long int dimmerNextStep;

unsigned long int curMillis;

public:

void init();

LampClass() {};

LampClass(String name, uint8\_t potPin, uint8\_t PWMpin) {

Name = name;

inBrightnessPot = potPin;

outPWMPin = PWMpin;

prevManualBrightness = map(analogRead(potPin), 0, 1023, 0, 255);//0-255

currentBrightness = prevManualBrightness;

prevBTBrightness = currentBrightness;

currentMode = Modes(Manual);

prevMode = currentMode;

IsTurnedON = false;

//prevBTBrightness;//do sprawdzania czy zmienic mode

//prevManBrightness;//do sprawdzania czy zmienic mode

timerIntervalInMilliseconds = 0;

IsTimerOn = false;

IsDimmerOn = false;

NextONOFF = 0;

}

//0-255

void SetBrightness(int val) {

if (val < 0) {

val = 0;

}

if (val > 255) {

val = 255;

}

currentBrightness = val;

float factor = val / 255.;

int brightnessToSet = (uint8\_t)(val\*factor\*factor\*factor); //doswiadczalnie zeby jasnosc rosla w miare liniowo

analogWrite(outPWMPin, brightnessToSet);

//Serial.println("curbright "+Name+" = "+String(currentBrightness));

}

void ActionOnMessage(String str) { //reakcja na blutut

curMillis = millis();

str = str.substring(3);//usuniecie "L1:"

//Serial.println("str-3= " + str);

//power on //::L1:onxxxx

if (str.startsWith("on")) {

IsTurnedON = true;

IsTimerOn = false;

SetBrightness(prevBTBrightness);//ostatnia brightness a nie 255

Serial.println("BT " + Name + " on prevBright=" + String(prevBTBrightness) + " on curBright=" + String(currentBrightness));

}

//power off //::L1:offxxx

if (str.startsWith("off")) {

IsTurnedON = false;

IsTimerOn = false;

prevBTBrightness = currentBrightness;

SetBrightness(0);

Serial.println("BT " + Name + " on prevBright=" + String(prevBTBrightness) + " on curBright=" + String(currentBrightness));

}

//set brightness //::L1:PWM12x

if (str.startsWith("PWM")) {

IsTurnedON = true;

str = str.substring(3);//usuniecie "PWM"

str.replace("x", "");//usuniecie xx

currentBrightness = str.toInt();//przeyslane jako 0-255 a nie 0-100

SetBrightness(currentBrightness);

Serial.println("BT " + Name + " setBright= " + String(currentBrightness));

}

//######## TIMER //::L1:TIM12x

if (str.startsWith("TIM")) {//set timer

IsTimerOn = true;

prevBTBrightness = currentBrightness;

str = str.substring(3);//usuniecie "TIM"

str.replace("x", "");//usuniecie xx

unsigned long int timerValueInMinutes = str.toInt();//0-999 minut

Serial.println("BT " + Name + " tim= " + String(timerValueInMinutes) + "min");

//#########timer w minutach tak jak trzeba do release

timerIntervalInMilliseconds = timerValueInMinutes \* 60 \* 1000;

//##############

//minuty jako skenudy - to potem usun tylko do testow

//timerIntervalInMilliseconds = timerIntervalInMilliseconds / 60;

timerStopTime = curMillis + timerIntervalInMilliseconds;

Serial.println("BT " + Name + " curmillis= " + String(curMillis));

//Serial.println("BT " + Name + " curInterval= " + String(timerIntervalInMilliseconds));

Serial.println("BT " + Name + " timStop= " + String(timerStopTime));

if (IsDimmerOn)

{

if (NextONOFF == 0) //ma wylaczyc lampe po czasie

{

dimmerStep = timerIntervalInMilliseconds / currentBrightness;

dimmerNextStep = curMillis + dimmerStep;

}

else { //ma wlaczyc lampe po czasie

dimmerStep = timerIntervalInMilliseconds / (255 - currentBrightness);

dimmerNextStep = curMillis + dimmerStep;

}

}

}

//######### cancel TIMER

if (str.startsWith("TIcncl")) {//set dimmmer

IsTimerOn = false;

Serial.println("BT " + Name + " TIMCancel");

}

//########### CLOCK //::L1:CLhhmm

if (str.startsWith("CL")) {//set Clock on/off

//CLOCK to to samo co timer tylko tutaj zamiast recznei wyliczac czas timer

//to tutaj licze podst godziny obecnej i nastawionej

IsTimerOn = true;

prevBTBrightness = currentBrightness;

uint8\_t CLKHours = str.substring(2, 4).toInt();//

uint8\_t CLKMinutes = (str.substring(4, 6)).toInt();//

Serial.println("BT " + Name + " CLKtim= " + String(CLKHours) + ":" + String(CLKMinutes));

//przeliczanie na timer

//dorob ifa gdy przepelnienie czyli godzina ustawiona wczesniej niz obecan godzina

unsigned long int curTimeInMinutes =

SharedObjects.GetTime().H \* 60

+ SharedObjects.GetTime().M;

unsigned long int setTimeInMinutes =

CLKHours \* 60

+ CLKMinutes;

//#########timer tak jak trzeba do release

if (setTimeInMinutes > curTimeInMinutes)

{//godizna nastawiona pozniej niz obecna

timerIntervalInMilliseconds = (setTimeInMinutes - curTimeInMinutes) \* 60 \* 1000;

}

else

{//godizna nastawiona wczensije niz obecna

timerIntervalInMilliseconds = ((24 \* 60) + (setTimeInMinutes - curTimeInMinutes)) \* 60 \* 1000;

}

//minuty jako skenudy - to potem usun tylko do testow

//timerIntervalInMilliseconds = timerIntervalInMilliseconds / 60;

timerStopTime = curMillis + timerIntervalInMilliseconds;

timerNextStep = curMillis + timerStep;

Serial.println("BT " + Name + " curmillis= " + String(curMillis));

//Serial.println("BT " + Name + " CLKInterv= " + String(timerIntervalInMilliseconds));

Serial.println("BT " + Name + " CLKStop= " + String(timerStopTime));

if (IsDimmerOn)

{

dimmerStep = timerIntervalInMilliseconds / currentBrightness;

dimmerNextStep = curMillis + dimmerStep;

}

}

//######### cancel CLOCK

if (str.startsWith("Clcncl")) {//set dimmmer

IsTimerOn = false;

Serial.println("BT " + Name + " CLKCancel");

}

//############# DIMMER

if (str.startsWith("DIMon")) {//set dimmmer

IsDimmerOn = true;

Serial.println("BT " + Name + " DIMon");

}

if (str.startsWith("DIMoff")) {//set dimmmer

IsDimmerOn = false;

Serial.println("BT " + Name + " DIMoff");

}

//############# czy wlaczyc/wylaczyc po dimmer

if (str.startsWith("TGLon")) {//set dimmmer

NextONOFF = 1;

Serial.println("BT " + Name + " TGLon");

}

if (str.startsWith("TGLoff")) {//set dimmmer

NextONOFF = 0;

Serial.println("BT " + Name + " TGLoff");

}

//############zapytania ::L1:DIMoff

if (str.startsWith("?")) {//stan lampy

if (currentBrightness > 0)

str = ("::" + Name + ":" + "onxxxx");

else

{

str = ("::" + Name + ":" + "offxxx");

}

SharedObjects.MessageToSend = str;

Serial.println("Status send: " + str);

}

if (str.startsWith("DIM?")) {//stan sciemniacza

if (IsDimmerOn)

str = ("::" + Name + ":" + "DIMonxxxx");

else

{

str = ("::" + Name + ":" + "DIMoffxxx");

}

SharedObjects.MessageToSend = str;

Serial.println("Status send: " + str);

}

if (str.startsWith("BRGHT?")) {//stan sciemniacza

str = ("::" + Name + ":" + "PWM=" + String(currentBrightness));

SharedObjects.MessageToSend = str;

Serial.println("Status send: " + str);

}

}

void ActionOnTimer() {

curMillis = millis();

if (IsTimerOn)

{

//do testow timera

//if (curMillis > timerNextStep) {

// timerNextStep = curMillis + timerStep;

//}

if (IsDimmerOn)

{

if (curMillis > dimmerNextStep)

{

//Serial.println("dimstep" + String(dimmerStep) + " cur-nexstep " + String(curMillis - dimmerNextStep));

uint8\_t brightnessStep = ((curMillis - dimmerNextStep) / dimmerStep) + 1;

if (NextONOFF == 0) {

SetBrightness(currentBrightness - brightnessStep);

}

else {

SetBrightness(currentBrightness + brightnessStep);

}

//dimmerNextStep = curMillis + dimmerStep;

dimmerNextStep += brightnessStep \* dimmerStep;

Serial.println("BT " + Name + " DimFires " + String(dimmerNextStep) + " Bright " + String(currentBrightness));

}

}

if (curMillis > timerStopTime)

{

IsTimerOn = false;

if (NextONOFF == 1)

{

IsTurnedON = true;

SetBrightness(255);

}

else {

IsTurnedON = false;

SetBrightness(0);

}

Serial.println("BT " + Name + " TIMFires");

}

}

}

};

extern LampClass Lamp;

#endif

* 1. SharedObjects.h

// SharedObjects.h

#ifndef \_SHAREDOBJECTS\_h

#define \_SHAREDOBJECTS\_h

#if defined(ARDUINO) && ARDUINO >= 100

#include "arduino.h"

#else

#include "WProgram.h"

#endif

struct Time {

uint8\_t H; uint8\_t M; uint8\_t S;

};

class SharedObjectsClass

{

private:

static Time currentTime;

public:

void init();

static String MessageToSend;

int GetHours() {

return currentTime.H;

}

void SetHours(uint8\_t val) {

if (val<0 || val>23)currentTime.H=0;

else currentTime.H = val;

}

int GetMinutes() {

return currentTime.M;

}

void SetMinutes(uint8\_t val) {

if (val<0 || val>59)currentTime.M = 0;

else currentTime.M = val;

}

int GetSeconds() {

return currentTime.S;

}

void SetSeconds(uint8\_t val) {

if (val<0 || val>59)currentTime.S = 0;

else currentTime.S = val;

}

void SetTime(uint8\_t h, uint8\_t m, uint8\_t s) {

currentTime.H = h;

currentTime.M = m;

currentTime.S = s;

}

Time GetTime() {

return currentTime;

}

};

extern SharedObjectsClass SharedObjects;

#endif

1. Podsumowanie i Wnioski

Podczas pisania programu spotkałem się ze zjawiskiem ze zabrakło pamieci, stała zajetośc wyniosła ok 60%, jednak doszło do przepelnienia stosu, po usunieciu stałych stringow z definicji klasy, zamianie zmiennych int na uint\_8t itp, zajetos spadla do 42% i juz problem juz się nie pojawial. Pomogly by pewno tez dyrektywy preprocesora #define

Problem z czytelnoscia komunikatow czyli zapis ich gdzies na poczatku moglby rozwiazac jeszcze jeden enumarator (jak w trybach), wtedy zamiast calego stringu mamy tylko jeden bajt. Ale w firmware w zasadzie nie ma co rozbudowywac wiec nie poprawialem juz tego.

Na przyszlosc lepiej zainteresowac się mikrokontrolerem STM32F103C9T6, podobny rozmiar nieco tanszy a oferujacy duzo wieksze mozliwosci , dzieki bibliotece CubeMX przypomina Arduino z jego prostotą i szybkościa łatwego tworzenia działajacych aplikacji

AppInventor nie spelnil oczekiwan, mimo ze na poczatku wszytsko ladnie dzialalo to dosc szybko dal sie we znaki toporny interfejs gdzie kazda kontrolke trzeba bylo do nowa ustawiac, nie ma kopiuj/wklej, layout nie odpowiada rzeczywistosci, kompilator czesto zrywa polaczenie.

Edytor z kolei ma kopiuj/wklej, ale nie mozna zaznaczac kilku obiektow co utrudnia organizacji programu, komentarze sa niewygodne.

AppInventor nadaje sie do naprawde prowizorycznych programow.

Po doswiadczeniach z Xamarinem zalowalem ze od razu nie napisalem klienta w C#, podejrzewam ze na walke z AppInventroem i bezmyslne i wtórne klikanie stracilem o wiele czasu niż zaoszczedzony na poczatku.

Ale zawsze to wiedza na przyszlosc, podobnie jak to ze kolejny projekt bede robil na STM32

1. Planowane Upgrady
   1. ogólnie

* liniowe rozjaśnianie
* przyspieszenie wysylania – plynne rozjasnianie w czasei rzeczywistym
* zmiana nazwy modulu z HC-05 na KwadraLampa
  1. Klient Android
* dzwieki on/off
* krotkei nazwy nadajnikow na liscie
* domsylny nadajnik zapisz , zapis aktualny jako domyslny,
* lacz przy starcie z domyslnym,
* ikonka help i about
* ustwiaenia BT i inne do ekranu ustwaienia
* komunikacja w obie strony