**GyverTwink_v1.0.ino**

```
/*
  Скетч к проекту "GyverTwink"
  - Страница проекта (схемы, описания): https://alexgyver.ru/gyvertwink/
  - Исходники на GitHub: https://github.com/AlexGyver/GyverTwink
  Проблемы с загрузкой? Читай гайд для новичков: https://alexgyver.ru/arduino-first/
  AlexGyver, AlexGyver Technologies, 2021
*/

/*
  Мигает синим - открыт портал
  Мигает жёлтым - подключаемся к точке
  Мигнул зелёным - подключился к точке
  Мигнул красным - ошибка подключения к точке
  Мигнул розовым - создал точку
*/

// ================ НАСТРОЙКИ ================
#define BTN_PIN D3      // пин кнопки
#define LED_PIN D1      // пин ленты
#define LED_TYPE WS2812 // чип ленты
#define LED_ORDER GRB   // порядок цветов ленты
#define LED_MAX 500     // макс. светодиодов

// имя точки в режиме AP
#define GT_AP_SSID "GyverTwink"
#define GT_AP_PASS "12345678"
//#define DEBUG_SERIAL_GT   // раскомментируй, чтобы включить отладку
```

```
// ================== LIBS ==================
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <SimplePortal.h>
#include <FastLED.h>
#include <EEManager.h>
#include <EncButton.h>
#include "palettes.h"
#include "Timer.h"

// ================== OBJECTS ==================
WiFiServer server(80);
WiFiUDP udp;
EEManager EEwifi(portalCfg);
CRGB leds[LED_MAX];
CLEDController *strip;
EncButton<EB_TICK, BTN_PIN> btn;
IPAddress myIP;

// ================== EEPROM BLOCKS ==================
struct Cfg {
  uint16_t ledAm = 50;
  bool power = 1;
  byte bright = 100;
  bool autoCh = 0;
  bool rndCh = 0;
  byte prdCh = 1;
```

```
  bool turnOff = 0;
  byte offTmr = 60;
};
Cfg cfg;
EEManager EEcfg(cfg);

byte xy[LED_MAX][2];
EEManager EExy(xy);

struct MM {
  byte minY = 0;
  byte maxY = 255;
  byte minX = 0;
  byte maxX = 255;
  byte w = 255;
  byte h = 255;
};
MM mm;
EEManager EEmm(mm);

#define ACTIVE_PALETTES 11
struct Effects {
  bool fav = true;
  byte scale = 50;
  byte speed = 150;
};
Effects effs[ACTIVE_PALETTES * 2];
EEManager EEeff(effs);
```

```
// ================== MISC DATA ==================
Timer forceTmr(30000, false);
Timer switchTmr(0, false);
Timer offTmr(60000, false);
bool calibF = false;
byte curEff = 0;
byte forceEff = 0;

#ifdef DEBUG_SERIAL_GT
#define DEBUGLN(x) Serial.println(x)
#define DEBUG(x) Serial.print(x)
#else
#define DEBUGLN(x)
#define DEBUG(x)
#endif

// ================== SETUP ==================
void setup() {
#ifdef DEBUG_SERIAL_GT
  Serial.begin(115200);
  DEBUGLN();
#endif
  startStrip();
  EEPROM.begin(2048); // с запасом!

  // если это первый запуск или щелчок по кнопке, открываем портал
  if (EEwifi.begin(0, 'a') || checkButton()) portalRoutine();
```

```
  // создаём точку или подключаемся к AP
  if (portalCfg.mode == WIFI_AP || (portalCfg.mode == WIFI_STA &&
portalCfg.SSID[0] == '\0')) setupAP();
  else setupSTA();
  DEBUGLN(myIP);

  EEcfg.begin(EEwifi.nextAddr(), 'a');
  EEeff.begin(EEcfg.nextAddr(), 'a');
  EEmm.begin(EEeff.nextAddr(), (uint8_t)LED_MAX);
  EExy.begin(EEmm.nextAddr(), (uint8_t)LED_MAX);

  switchTmr.setPrd(cfg.prdCh * 60000ul);
  if (cfg.autoCh) switchTmr.restart();
  switchEff();
  cfg.turnOff = false;
  strip->setLeds(leds, cfg.ledAm);
  udp.begin(8888);
}

// ================== LOOP ==================
void loop() {
  button();   // опрос кнопки

  // менеджер епром
  EEcfg.tick();
  EEeff.tick();

  parsing();  // парсим udp
```

```
  // таймер принудительного показа эффектов
  if (forceTmr.ready()) {

    forceTmr.stop();
    switchEff();
  }

  // форс выключен и настало время менять эффект
  if (!forceTmr.state() && switchTmr.ready()) switchEff();

  // таймер выключения
  if (offTmr.ready()) {
    offTmr.stop();
    cfg.turnOff = false;
    cfg.power = false;
    strip->showLeds(0);
    EEcfg.update();
    DEBUGLN("Off tmr");
  }

  // показываем эффект, если включены
  if (!calibF && cfg.power) effects();
}
```

**Timer.h**

```
// класс таймера
#pragma once
```

```
struct Timer {
  Timer (uint32_t prd, boolean state = true) {
    setPrd(prd);
    _state = state;
  }

  void setPrd(uint32_t prd) {
    if (prd > 0) _prd = prd;
    else stop();
  }

  bool ready() {
    if (_state && millis() - _tmr >= _prd) {
      restart();
      return true;
    }
    return false;
  }

  void restart() {
    _tmr = millis();
    _state = true;
  }

  void stop() {
    _state = false;
  }
```

```
  bool state() {
    return _state;
  }

  uint32_t _tmr, _prd;
  bool _state = true;
};
```

**button.ino**

```
void button() {
  static bool dir = 1;

  btn.tick();
  if (btn.hasClicks(1)) {
    DEBUGLN("1x click - power");
    cfg.power = !cfg.power;
    if (!cfg.power) strip->showLeds(0);
    EEcfg.update();
  }

  if (btn.hasClicks(2)) {
    DEBUGLN("2x click - next");
    switchEff();
  }

  if (btn.step()) {
    DEBUGLN("change bright");
    if (cfg.power) {
```

```
      // меняем безопасно
      int bri = cfg.bright;
      bri += dir ? 30 : -30;
      bri = constrain(bri, 0, 255);
      cfg.bright = bri;
      if (bri == 255) {
        strip->showLeds(0);
        delay(100);
        strip->showLeds(bri);
      }
      EEcfg.update();
    }
  }

  if (btn.releaseStep()) {
    DEBUGLN("change dir");
    dir = !dir;
  }
}
```

**effects.ino**

```
void effects() {
  static Timer effTmr(30);
  static uint16_t countP = 0;
  static byte countSkip = 0;
  static byte prevEff = 255;
  static byte fadeCount = 0;
```

```
  if (effTmr.ready()) {
    byte thisEffect;

    if (forceTmr.state()) thisEffect = forceEff;
    else thisEffect = curEff;

    // эффект сменился
    if (prevEff != curEff) {
      prevEff = curEff;
      fadeCount = 25;
    }

    byte scale = effs[thisEffect].scale;
    byte speed = effs[thisEffect].speed;
    byte curPal = thisEffect;
    if (curPal >= ACTIVE_PALETTES) curPal -= ACTIVE_PALETTES;

    for (int i = 0; i < cfg.ledAm; i++) {
      byte idx;

      if (thisEffect < ACTIVE_PALETTES) {
        // первые ACTIVE_PALETTES эффектов - градиент
        // idx = map(xy[i][1], mm.minY, mm.maxY, 0, 255) + counter;   // прямой
градиент
        idx = countP + ((mm.w * xy[i][0] / mm.h) + xy[i][1]) * scale / 100;   //
диагональный градиент
      } else {
        // следующие - перлин нойс
        idx = inoise8(xy[i][0] * scale / 10, xy[i][1] * scale / 10, countP);
```

```
      }
      CRGB color = ColorFromPalette(paletteArr[curPal], idx, 255, LINEARBLEND);

      // плавная смена эффекта
      // меняется за 25 фреймов
      if (fadeCount) leds[i] = blend(leds[i], color, 40);
      else leds[i] = color;
    }
    if (fadeCount) fadeCount--;

    countP += (speed - 128) / 10;
    strip->showLeds(cfg.bright);
  }
}
```

**faders.ino**

```
void fader(CRGB color) {
  static uint32_t tmr;
  static int val = 0;
  static bool dir = true;
  if (millis() - tmr >= 20) {
    tmr = millis();
    val += dir ? 3 : -3;
    val = constrain(val, 5, 120);
    if (val >= 120 || val <= 5) dir = !dir;
    strip->showColor(color, val);
  }
}
```

```
void fadeBlink(CRGB color) {
  for (int i = 0; i < 200; i += 20) {
    strip->showColor(color, i);
    delay(20);
  }
  for (int i = 200; i > 0; i -= 20) {
    strip->showColor(color, i);
    delay(20);
  }
  strip->clearLedData();
  strip->showLeds(0);
}
```

**palettes.h**

```
// в общем это сборник палитр из GyverLamp2
// я оставил самые подходящие (листай в самый низ)

#include <FastLED.h>      // лента
// http://soliton.vm.bytemark.co.uk/pub/cpt-city/

CRGBPalette16 customPal;

DEFINE_GRADIENT_PALETTE( Fire_gp ) {
  0,     0,  0,  0,
  128,   255,  0,  0,
  224,   255, 255,  0,
  255,   255, 255, 255
```

```
};

DEFINE_GRADIENT_PALETTE( Sunset_Real_gp ) {
  0, 120,  0,  0,
  22, 179, 22,  0,
  51, 255, 104,  0,
  85, 167, 22, 18,
  135, 100,  0, 103,
  198,  16,  0, 130,
  255,   0,  0, 160
};

DEFINE_GRADIENT_PALETTE( dkbluered_gp ) {
  0,   1,  0,  4,
  8,   1,  0, 13,
  17,   1,  0, 29,
  25,   1,  0, 52,
  33,   1,  0, 83,
  42,   1,  0, 123,
  51,   1,  0, 174,
  59,   1,  0, 235,
  68,   1,  2, 255,
  76,   4, 17, 255,
  84,  16, 45, 255,
  93,  37, 82, 255,
  102,  69, 127, 255,
  110, 120, 168, 255,
  119, 182, 217, 255,
```

```
  127, 255, 255, 255,
  135, 255, 217, 184,
  144, 255, 168, 123,
  153, 255, 127, 73,
  161, 255, 82, 40,
  170, 255, 45, 18,
  178, 255, 17,  5,
  186, 255,  2,  1,
  195, 234,  0,  1,
  204, 171,  0,  1,
  212, 120,  0,  1,
  221,  79,  0,  1,
  229,  48,  0,  1,
  237,  26,  0,  1,
  246,  12,  0,  1,
  255,   4,  0,  1
};
DEFINE_GRADIENT_PALETTE( Optimus_Prime_gp ) {
  0,   5, 16, 18,
  25,   5, 16, 18,
  51,   7, 25, 39,
  76,   8, 38, 71,
  102,  64, 99, 106,
  127, 194, 189, 151,
  153, 182, 63, 42,
  178, 167,  6,  2,
  204, 100,  3,  1,
  229,  53,  1,  1,
```

```
  255,  53,  1,  1
};

DEFINE_GRADIENT_PALETTE( warmGrad_gp ) {
  0, 252, 252, 172,
  25, 239, 255, 61,
  53, 247, 45, 17,
  76, 197, 82, 19,
  96, 239, 255, 61,
  124,  83,  4,  1,
  153, 247, 45, 17,
  214,  23, 15, 17,
  255,   1,  1,  1
};

DEFINE_GRADIENT_PALETTE( coldGrad_gp ) {
  0,  66, 186, 192,
  43,   1, 22, 71,
  79,   2, 104, 142,
  117,  66, 186, 192,
  147,   2, 104, 142,
  186,   1, 22, 71,
  224,   2, 104, 142,
  255,   4, 27, 28
};

DEFINE_GRADIENT_PALETTE( hotGrad_gp ) {
  0, 157, 21,  2,
```

```
  35, 229, 244, 16,
  73, 255, 44,  7,
  107, 142,  7,  1,
  153, 229, 244, 16,
  206, 142,  7,  1,
  255, 135, 36,  0
};


DEFINE_GRADIENT_PALETTE( pinkGrad_gp ) {
  0, 249, 32, 145,
  28, 208,  1,  7,
  43, 249,  1, 19,
  56, 126, 152, 10,
  73, 234, 23, 84,
  89, 224, 45, 119,
  107, 232, 127, 158,
  127, 244, 13, 89,
  150, 188,  6, 52,
  175, 177, 70, 14,
  221, 194,  1,  8,
  255, 112,  0,  1
};

DEFINE_GRADIENT_PALETTE( comfy_gp ) {
  0, 255, 255, 45,
  43, 208, 93,  1,
  137, 224,  1, 242,
```

```
  181, 159,  1, 29,
  255,  63,  4, 68
};

DEFINE_GRADIENT_PALETTE( cyperpunk_gp ) {
  0,   3,  6, 72,
  38,  12, 50, 188,
  109, 217, 35,  1,
  135, 242, 175, 12,
  178, 161, 32, 87,
  255,  24,  6, 108
};

DEFINE_GRADIENT_PALETTE( girl_gp ) {
  0, 103,  1, 10,
  33, 109,  1, 12,
  76, 159,  5, 48,
  119, 175, 55, 103,
  127, 175, 55, 103,
  178, 159,  5, 48,
  221, 109,  1, 12,
  255, 103,  1, 10
};

DEFINE_GRADIENT_PALETTE( xmas_gp ) {
  0,   0, 12,  0,
  40,   0, 55,  0,
  66,   1, 117,  2,
```

```
  77,   1, 84,  1,
  81,   0, 55,  0,
  119,   0, 12,  0,
  153,  42,  0,  0,
  181, 121,  0,  0,
  204, 255, 12,  8,
  224, 121,  0,  0,
  244,  42,  0,  0,
  255,  42,  0,  0
};

DEFINE_GRADIENT_PALETTE( acid_gp ) {
  0,   0, 12,  0,
  61, 153, 239, 112,
  127,   0, 12,  0,
  165, 106, 239,  2,
  196, 167, 229, 71,
  229, 106, 239,  2,
  255,   0, 12,  0
};


DEFINE_GRADIENT_PALETTE( blueSmoke_gp ) {
  0,   0,  0,  0,
  12,   1,  1,  3,
  53,   8,  1, 22,
  80,   4,  6, 89,
  119,   2, 25, 216,
```

```
  145,   7, 10, 99,
  186,  15,  2, 31,
  233,   2,  1,  5,
  255,   0,  0,  0
};

DEFINE_GRADIENT_PALETTE( gummy_gp ) {
  0,   8, 47,  5,
  31,  77, 122,  6,
  63, 249, 237,  7,
  95, 232, 51,  1,
  127, 215,  0,  1,
  159,  47,  1,  3,
  191,   1,  7, 16,
  223,  52, 22,  6,
  255, 239, 45,  1,
};

DEFINE_GRADIENT_PALETTE( leo_gp ) {
  0, 0, 0, 0,
  16, 0, 0, 0,
  32, 0, 0, 0,
  18, 0, 0, 0,
  64, 16, 8, 0,
  80, 80, 40, 0,
  96, 16, 8, 0,
  112, 0, 0, 0,
  128, 0, 0, 0,
```

```
  144, 0, 0, 0,
  160, 0, 0, 0,
  176, 0, 0, 0,
  192, 0, 0, 0,
  208, 0, 0, 0,
  224, 0, 0, 0,
  240, 0, 0, 0,
  255, 0, 0, 0,
};

DEFINE_GRADIENT_PALETTE ( aurora_gp ) {
  0,  17, 177,  13,    //Greenish
  64, 121, 242,   5,    //Greenish
  128,  25, 173, 121,    //Turquoise
  192, 250,  77, 127,    //Pink
  255, 171, 101, 221     //Purple
};

DEFINE_GRADIENT_PALETTE ( redwhite_gp ) {
  0, 255, 0, 0,
  25, 255, 255, 255,
  51, 255, 0, 0,
  76, 255, 255, 255,
  102, 255, 0, 0,
  127, 255, 255, 255,
  153, 255, 0, 0,
  178, 255, 255, 255,
  204, 255, 0, 0,
```

```
  229, 255, 255, 255,

  255, 255, 0, 0,

};


const TProgmemRGBPalette16 WoodFireColors_p PROGMEM = {CRGB::Black,
0x330e00, 0x661c00, 0x992900, 0xcc3700, CRGB::OrangeRed, 0xff5800, 0xff6b00,
0xff7f00, 0xff9200, CRGB::Orange, 0xffaf00, 0xffb900, 0xffc300, 0xffcd00,
CRGB::Gold};              //* рыжий

const TProgmemRGBPalette16 NormalFire_p PROGMEM = {CRGB::Black,
0x330000, 0x660000, 0x990000, 0xcc0000, CRGB::Red, 0xff0c00, 0xff1800,
0xff2400, 0xff3000, 0xff3c00, 0xff4800, 0xff5400, 0xff6000, 0xff6c00, 0xff7800};
// красный

const TProgmemRGBPalette16 LithiumFireColors_p PROGMEM = {CRGB::Black,
0x240707, 0x470e0e, 0x6b1414, 0x8e1b1b, CRGB::FireBrick, 0xc14244, 0xd16166,
0xe08187, 0xf0a0a9, CRGB::Pink, 0xff9ec0, 0xff7bb5, 0xff59a9, 0xff369e,
CRGB::DeepPink};        //* пастель

const TProgmemRGBPalette16 SodiumFireColors_p PROGMEM = {CRGB::Black,
0x332100, 0x664200, 0x996300, 0xcc8400, CRGB::Orange, 0xffaf00, 0xffb900,
0xffc300, 0xffcd00, CRGB::Gold, 0xf8cd06, 0xf0c30d, 0xe9b913, 0xe1af1a,
CRGB::Goldenrod};           //* Yellow

const TProgmemRGBPalette16 CopperFireColors_p PROGMEM = {CRGB::Black,
0x001a00, 0x003300, 0x004d00, 0x006600, CRGB::Green, 0x239909, 0x45b313,
0x68cc1c, 0x8ae626, CRGB::GreenYellow, 0x94f530, 0x7ceb30, 0x63e131,
0x4bd731, CRGB::LimeGreen};     //* Green

const TProgmemRGBPalette16 AlcoholFireColors_p PROGMEM = {CRGB::Black,
0x000033, 0x000066, 0x000099, 0x0000cc, CRGB::Blue, 0x0026ff, 0x004cff,
0x0073ff, 0x0099ff, CRGB::DeepSkyBlue, 0x1bc2fe, 0x36c5fd, 0x51c8fc, 0x6ccbfb,
CRGB::LightSkyBlue};  //* Blue


CRGBPalette16 paletteArr[] = {

  /*HeatColors_p,

  Fire_gp,

  WoodFireColors_p,

  NormalFire_p,

  LithiumFireColors_p,

  SodiumFireColors_p,
```

CopperFireColors_p,

AlcoholFireColors_p,

LavaColors_p,

PartyColors_p,

RainbowColors_p,

RainbowStripeColors_p,

CloudColors_p,

OceanColors_p,

ForestColors_p,

Sunset_Real_gp,

dkbluered_gp,

Optimus_Prime_gp,

warmGrad_gp,

coldGrad_gp,

hotGrad_gp,

pinkGrad_gp,

comfy_gp,

cyperpunk_gp,

girl_gp,

xmas_gp,

acid_gp,

blueSmoke_gp,

gummy_gp,

leo_gp,

aurora_gp,

redwhite_gp,*/

PartyColors_p,

RainbowColors_p,

```
  RainbowStripeColors_p,
  Sunset_Real_gp,
  dkbluered_gp,
  warmGrad_gp,
  coldGrad_gp,
  hotGrad_gp,
  pinkGrad_gp,
  cyperpunk_gp,
  redwhite_gp,
};
```

**parsing.ino**

```
#define MAX_UDP_PACKET 30
byte ubuf[MAX_UDP_PACKET];

void reply(byte* data, byte size) {
  udp.beginPacket(udp.remoteIP(), udp.remotePort());
  udp.write("GT");
  udp.write(data, size);
  udp.endPacket();
}

void parsing() {
  static uint32_t tmr;
  if (udp.parsePacket()) {
    if (millis() - tmr < 50) {  // "антидребезг" приёма
      udp.read(ubuf, MAX_UDP_PACKET);
      return;
```

```
}
tmr = millis();
int n = udp.read(ubuf, MAX_UDP_PACKET);
ubuf[n] = 0;
if (ubuf[0] != 'G' || ubuf[1] != 'T') return;
/*for (int i = 2; i < n; i++) {
  DEBUG(ubuf[i]);
  DEBUG(',');
  }
  DEBUGLN();*/
byte answ[10];

switch (ubuf[2]) {
  case 0:   // запрос IP
    delay(myIP[3] * 2);
    answ[0] = 0;
    answ[1] = myIP[3];
    reply(answ, 2);
    break;

  case 1:   // запрос настроек
    answ[0] = 1;
    answ[1] = cfg.ledAm / 100;
    answ[2] = cfg.ledAm % 100;
    answ[3] = cfg.power;
    answ[4] = cfg.bright;
    answ[5] = cfg.autoCh;
    answ[6] = cfg.rndCh;
```

```
  answ[7] = cfg.prdCh;
  answ[8] = cfg.turnOff;
  answ[9] = cfg.offTmr;
  reply(answ, 10);
  break;

case 2:   // приём настроек
  forceTmr.stop();
  switch (ubuf[3]) {
    case 0: cfg.ledAm = ubuf[4] * 100 + ubuf[5];
      strip->setLeds(leds, cfg.ledAm);
      break;
    case 1: cfg.power = ubuf[4];
      break;
    case 2: cfg.bright = ubuf[4];
      break;
    case 3: cfg.autoCh = ubuf[4];
      if (cfg.autoCh) switchTmr.restart();
      else switchTmr.stop();
      break;
    case 4: cfg.rndCh = ubuf[4];
      break;
    case 5: cfg.prdCh = ubuf[4];
      switchTmr.setPrd(cfg.prdCh * 60000ul);
      if (cfg.autoCh) switchTmr.restart();
      break;
    case 6:   // нехт эффект
      switchEff();
```

```
      if (cfg.autoCh) switchTmr.restart();
      return;
      break;
    case 7:
      cfg.turnOff = ubuf[4];
      if (cfg.turnOff) offTmr.restart();
      else offTmr.stop();
      break;
    case 8:
      cfg.offTmr = ubuf[4];
      offTmr.setPrd(cfg.offTmr * 60000ul);
      if (cfg.turnOff) offTmr.restart();
      break;
  }
  if (!cfg.power) strip->showLeds(0);
  EEcfg.update();
  break;

case 3:
  switch (ubuf[3]) {
    case 0:   // запуск калибровки
      DEBUGLN("Calibration start");
      calibF = true;
      strip->clearLedData();
      strip->showLeds(0);
      break;

    case 1:   // следующий лед
```

```
  if (ubuf[4] > 0) {
    xy[ubuf[4] - 1][0] = ubuf[5];
    xy[ubuf[4] - 1][1] = ubuf[6];
  }
  strip->clearLedData();
  leds[ubuf[4]] = CRGB::White;
  strip->showLeds(200);
  break;

case 2:   // калибровка окончена
  DEBUGLN("Finished");
  calibF = false;
  strip->clearLedData();
  strip->showLeds(0);
  EExy.updateNow();

  mm.minY = 255;
  mm.maxY = 0;
  mm.minX = 255;
  mm.maxX = 0;
  for (int i = 0; i < cfg.ledAm; i++) {
    mm.minX = min(mm.minX, xy[i][0]);
    mm.maxX = max(mm.maxX, xy[i][0]);
    mm.minY = min(mm.minY, xy[i][1]);
    mm.maxY = max(mm.maxY, xy[i][1]);
  }
  mm.w = mm.maxX - mm.minX;
  mm.h = mm.maxY - mm.minY;
```

```
      EEmm.updateNow();
      break;
    }
    break;

  case 4:   // управление эффектами
    forceTmr.restart();
    EEeff.update();
    switch (ubuf[3]) {
      case 0:   // выбор эффекта в дропе
        forceEff = ubuf[4];
        answ[0] = 4;
        answ[1] = effs[forceEff].fav;
        answ[2] = effs[forceEff].scale;
        answ[3] = effs[forceEff].speed;
        reply(answ, 4);
        break;
      case 1:   // флажок избранное
        effs[forceEff].fav = ubuf[4];
        break;
      case 2:   // масштаб
        effs[forceEff].scale = ubuf[4];
        break;
      case 3:   // скорость
        effs[forceEff].speed = ubuf[4];
        break;
    }
    break;
```

```
    }
  }
}
```

**parsing.ino**

```
#define MAX_UDP_PACKET 30
byte ubuf[MAX_UDP_PACKET];

void reply(byte* data, byte size) {
  udp.beginPacket(udp.remoteIP(), udp.remotePort());
  udp.write("GT");
  udp.write(data, size);
  udp.endPacket();
}

void parsing() {
  static uint32_t tmr;
  if (udp.parsePacket()) {
    if (millis() - tmr < 50) {  // "антидребезг" приёма
      udp.read(ubuf, MAX_UDP_PACKET);
      return;
    }
    tmr = millis();
    int n = udp.read(ubuf, MAX_UDP_PACKET);
    ubuf[n] = 0;
    if (ubuf[0] != 'G' || ubuf[1] != 'T') return;
    /*for (int i = 2; i < n; i++) {
      DEBUG(ubuf[i]);
```

```
  DEBUG(',');
  }
  DEBUGLN();*/
byte answ[10];

switch (ubuf[2]) {
  case 0:   // запрос IP
    delay(myIP[3] * 2);
    answ[0] = 0;
    answ[1] = myIP[3];
    reply(answ, 2);
    break;

  case 1:   // запрос настроек
    answ[0] = 1;
    answ[1] = cfg.ledAm / 100;
    answ[2] = cfg.ledAm % 100;
    answ[3] = cfg.power;
    answ[4] = cfg.bright;
    answ[5] = cfg.autoCh;
    answ[6] = cfg.rndCh;
    answ[7] = cfg.prdCh;
    answ[8] = cfg.turnOff;
    answ[9] = cfg.offTmr;
    reply(answ, 10);
    break;

  case 2:   // приём настроек
```

```
forceTmr.stop();
switch (ubuf[3]) {
  case 0: cfg.ledAm = ubuf[4] * 100 + ubuf[5];
    strip->setLeds(leds, cfg.ledAm);
    break;
  case 1: cfg.power = ubuf[4];
    break;
  case 2: cfg.bright = ubuf[4];
    break;
  case 3: cfg.autoCh = ubuf[4];
    if (cfg.autoCh) switchTmr.restart();
    else switchTmr.stop();
    break;
  case 4: cfg.rndCh = ubuf[4];
    break;
  case 5: cfg.prdCh = ubuf[4];
    switchTmr.setPrd(cfg.prdCh * 60000ul);
    if (cfg.autoCh) switchTmr.restart();
    break;
  case 6:   // нехт эффект
    switchEff();
    if (cfg.autoCh) switchTmr.restart();
    return;
    break;
  case 7:
    cfg.turnOff = ubuf[4];
    if (cfg.turnOff) offTmr.restart();
    else offTmr.stop();
```

```
      break;
    case 8:
      cfg.offTmr = ubuf[4];
      offTmr.setPrd(cfg.offTmr * 60000ul);
      if (cfg.turnOff) offTmr.restart();
      break;
  }
  if (!cfg.power) strip->showLeds(0);
  EEcfg.update();
  break;

case 3:
  switch (ubuf[3]) {
    case 0:   // запуск калибровки
      DEBUGLN("Calibration start");
      calibF = true;
      strip->clearLedData();
      strip->showLeds(0);
      break;

    case 1:   // следующий лед
      if (ubuf[4] > 0) {
        xy[ubuf[4] - 1][0] = ubuf[5];
        xy[ubuf[4] - 1][1] = ubuf[6];
      }
      strip->clearLedData();
      leds[ubuf[4]] = CRGB::White;
      strip->showLeds(200);
```

```
      break;

    case 2:   // калибровка окончена
      DEBUGLN("Finished");
      calibF = false;
      strip->clearLedData();
      strip->showLeds(0);
      EExy.updateNow();

      mm.minY = 255;
      mm.maxY = 0;
      mm.minX = 255;
      mm.maxX = 0;
      for (int i = 0; i < cfg.ledAm; i++) {
        mm.minX = min(mm.minX, xy[i][0]);
        mm.maxX = max(mm.maxX, xy[i][0]);
        mm.minY = min(mm.minY, xy[i][1]);
        mm.maxY = max(mm.maxY, xy[i][1]);
      }
      mm.w = mm.maxX - mm.minX;
      mm.h = mm.maxY - mm.minY;
      EEmm.updateNow();
      break;
  }
  break;

case 4:   // управление эффектами
  forceTmr.restart();
```

```
        EEeff.update();
        switch (ubuf[3]) {
          case 0:   // выбор эффекта в дропе
            forceEff = ubuf[4];
            answ[0] = 4;
            answ[1] = effs[forceEff].fav;
            answ[2] = effs[forceEff].scale;
            answ[3] = effs[forceEff].speed;
            reply(answ, 4);
            break;
          case 1:   // флажок избранное
            effs[forceEff].fav = ubuf[4];
            break;
          case 2:   // масштаб
            effs[forceEff].scale = ubuf[4];
            break;
          case 3:   // скорость
            effs[forceEff].speed = ubuf[4];
            break;
        }
        break;
    }
  }
}
```

**startup.ino**

```
void portalRoutine() {
  // запускаем portal
```

```
  portalStart();
  DEBUGLN("Portal start");

  // ждём действий пользователя, мигаем
  while (!portalTick()) fader(CRGB::Blue);

  // если это 1 connect, 2 ap, 3 local, обновляем данные в епр
  if (portalStatus() <= 3) EEwifi.updateNow();

  DEBUG("Portal status: ");
  DEBUGLN(portalStatus());
}

void startStrip() {
  strip = &FastLED.addLeds<LED_TYPE, LED_PIN, LED_ORDER>(leds,
LED_MAX).setCorrection(TypicalLEDStrip);
  strip->setLeds(leds, LED_MAX);
  strip->clearLedData();
  // выводим ргб
  leds[0] = CRGB::Red;
  leds[1] = CRGB::Green;
  leds[2] = CRGB::Blue;
  strip->showLeds(50);
}

bool checkButton() {
  uint32_t tmr = millis();
  while (millis() - tmr < 2000) {
    if (!digitalRead(BTN_PIN)) return true;
```

```
  }
  return false;
}

void setupAP() {
  DEBUG("AP Mode");
  WiFi.disconnect();
  WiFi.mode(WIFI_AP);
  WiFi.softAP(GT_AP_SSID, GT_AP_PASS);
  myIP = WiFi.softAPIP();
  server.begin();
  fadeBlink(CRGB::Magenta);
}

void setupSTA() {
  DEBUG("Connecting to AP... ");
  WiFi.softAPdisconnect();
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);
  WiFi.begin(portalCfg.SSID, portalCfg.pass);
  uint32_t tmr = millis();
  bool state = false;
  while (millis() - tmr < 15000) {
    if (WiFi.status() == WL_CONNECTED) {
      fadeBlink(CRGB::Green);
      DEBUGLN("ok");
      myIP = WiFi.localIP();
      return;
```

```
    }
    fader(CRGB::Yellow);
    yield();
  }
  fadeBlink(CRGB::Red);
  DEBUGLN("fail");
  setupAP();
}
```

**switcher.ino**

```
void switchEff() {
  // поиск favorite эффектов
  while (true) {
    if (cfg.rndCh) curEff = random(0, ACTIVE_PALETTES * 2);
    else {
      if (++curEff >= ACTIVE_PALETTES * 2) curEff = 0;
    }
    if (effs[curEff].fav) break;
  }
  DEBUG("switch to: ");
  DEBUGLN(curEff);
}
```

EEManager.h

```
/*
    Менеджер EEPROM - библиотека для уменьшения износа памяти
    Документация:
```

```
    GitHub: https://github.com/GyverLibs/EEManager
    Возможности:
    - Отложенная запись (обновление) по таймеру
    - Работает на базе стандартной EEPROM.h
    - Встроенный механизм ключа первой записи
    AlexGyver, alex@alexgyver.ru
    https://alexgyver.ru/
    MIT License
    Версии:
    v1.0 - релиз
    v1.1 - изменены коды возврата begin
    v1.2 - добавлена nextAddr()
*/

#ifndef _EEManager_h
#define _EEManager_h
#include <Arduino.h>
#include <EEPROM.h>

class EEManager {
public:
    // передать данные любого типа, опционально таймаут обновления в мс
    template <typename T> EEManager(T &data, uint16_t tout = 5000) {
        _data = (uint8_t*) &data;
        _size = sizeof(T);
        _tout = tout;
    }
```

```
    // сменить таймаут
    void setTimeout(uint16_t tout = 5000) {
        _tout = tout;
    }

    // начать работу, прочитать данные в переменную. Принимает адрес начала хранения даты и ключ
    uint8_t begin(uint8_t addr, uint8_t key) {
        _addr = addr;
        if (_addr + _size + 1 > EEPROM.length()) return 2;  // не хватит места
        _ready = 1;
        if (EEPROM.read(_addr + _size) != key) {            // ключ не совпал
            EEPROM.write(_addr + _size, key);               // пишем ключ
            updateNow();                                    // пишем стандартные значения
            return 1;
        }
        for (uint16_t i = 0; i < _size; i++) _data[i] = EEPROM.read(_addr + i);
        return 0;
    }

    // обновить данные в еепром сейчас
    void updateNow() {
        if (_ready) {
#ifdef ESP8266
            for (uint16_t i = 0; i < _size; i++) EEPROM.write(_addr + i, _data[i]);
            EEPROM.commit();
#else
            for (uint16_t i = 0; i < _size; i++) EEPROM.update(_addr + i, _data[i]);
#endif
```

```
        }
    }

    // отложить обновление и сбросить таймер
    void update() {
        _tmr = millis();
        _update = 1;
    }

    // тикер обновления
    bool tick() {
        if (_update && millis() - _tmr >= _tout) {
            updateNow();
            _update = 0;
            return 1;
        } return 0;
    }

    // сбросить ключ запуска. При перезагрузке (или вызове begin) запишутся стандартные данные
    void reset() {
        EEPROM.write(_addr + _size, EEPROM.read(_addr + _size) + 1);   // меняем ключ на +1, при перезапуске будет дефолт
    }

    // получить размер данных
    uint16_t dataSize() {
        return _size;
    }
```

```
    // получить размер всего блока (данные + ключ)
    uint16_t blockSize() {
        return _size + 1;
    }

    // получить адрес первого байта в блоке
    uint16_t startAddr() {
        return _addr;
    }

    // получить адрес последнего байта в блоке (включая ключ)
    uint16_t endAddr() {
        return _addr + _size;
    }

    // получить первый свободный адрес для следующего блока
    uint16_t nextAddr() {
        return _addr + _size + 1;
    }

private:
    uint8_t* _data;
    uint16_t _size, _addr;
    bool _ready = 0, _update = 0;
    uint32_t _tmr = 0;
    uint16_t _tout;
};
```

```
#endif
```

EncButton.h

```
#ifndef _EncButton_h
#define _EncButton_h

// ========= НАСТРОЙКИ (можно передефайнить из скетча) ==========
#define _EB_FAST 30         // таймаут быстрого поворота
#define _EB_DEB 50          // дебаунс кнопки
#define _EB_HOLD 1000       // таймаут удержания кнопки
#define _EB_STEP 500        // период срабатывания степ
#define _EB_CLICK 400           // таймаут накликивания
//#define EB_BETTER_ENC     // точный алгоритм отработки энкодера (можно задефайнить в скетче)

// ========== НЕ ТРОГАЙ ============
#include <Arduino.h>

#ifndef nullptr
#define nullptr NULL
#endif

// флаг макро
#define _EB_setFlag(x) (flags |= 1 << x)
#define _EB_clrFlag(x) (flags &= ~(1 << x))
#define _EB_readFlag(x) ((flags >> x) & 1)
```

```
#ifndef EB_FAST
#define EB_FAST _EB_FAST
#endif
#ifndef EB_DEB
#define EB_DEB _EB_DEB
#endif
#ifndef EB_HOLD
#define EB_HOLD _EB_HOLD
#endif
#ifndef EB_STEP
#define EB_STEP _EB_STEP
#endif
#ifndef EB_CLICK
#define EB_CLICK _EB_CLICK
#endif

enum eb_callback {
    TURN_HANDLER,       // 0
    LEFT_HANDLER,       // 1
    RIGHT_HANDLER,      // 2
    LEFT_H_HANDLER,     // 3
    RIGHT_H_HANDLER,    // 4
    CLICK_HANDLER,      // 5
    HOLDED_HANDLER,     // 6
    STEP_HANDLER,       // 7
    PRESS_HANDLER,      // 8
    CLICKS_HANDLER,     // 9
    RELEASE_HANDLER,    // 10
```

```
    HOLD_HANDLER,       // 11
    TURN_H_HANDLER,     // 12
    // clicks amount 13
};

// константы
#define EB_TICK 0
#define EB_CALLBACK 1

#define EB_NO_PIN 255

#define VIRT_ENC 254
#define VIRT_ENCBTN 253
#define VIRT_BTN 252

#ifdef EB_BETTER_ENC
static const int8_t _EB_DIR[] = {
  0, -1,  1,  0,
  1,  0,  0, -1,
  -1,  0,  0,  1,
  0,  1, -1,  0
};
#endif
template < uint8_t _EB_MODE, uint8_t _S1 = EB_NO_PIN, uint8_t _S2 =
EB_NO_PIN, uint8_t _KEY = EB_NO_PIN >
class EncButton {
public:
    // можно указать режим работы пина
    EncButton(const uint8_t mode = INPUT_PULLUP) {
```

```
    if (_S1 < 252 && mode == INPUT_PULLUP) pullUp();
    setButtonLevel(LOW);
}

// подтянуть пины внутренней подтяжкой
void pullUp() {
    if (_S1 < 252) {                    // реальное устройство
        if (_S2 == EB_NO_PIN) {             // обычная кнопка
            pinMode(_S1, INPUT_PULLUP);
        } else if (_KEY == EB_NO_PIN) {     // энк без кнопки
            pinMode(_S1, INPUT_PULLUP);
            pinMode(_S2, INPUT_PULLUP);
        } else {                            // энк с кнопкой
            pinMode(_S1, INPUT_PULLUP);
            pinMode(_S2, INPUT_PULLUP);
            pinMode(_KEY, INPUT_PULLUP);
        }
    }
}

// установить таймаут удержания кнопки для isHold(), мс (до 30 000)
void setHoldTimeout(int tout) {
    _holdT = tout >> 7;
}

// виртуально зажать кнопку энкодера
void holdEncButton(bool state) {
    if (state) _EB_setFlag(8);
```

```
        else _EB_clrFlag(8);
    }

    // уровень кнопки: LOW - кнопка подключает GND (умолч.), HIGH - кнопка подключает VCC
    void setButtonLevel(bool level) {
        if (level) _EB_clrFlag(11);
        else _EB_setFlag(11);
    }

    // ====================================== TICK ======================================
    // тикер, вызывать как можно чаще
    // вернёт отличное от нуля значение, если произошло какое то событие
    uint8_t tick(uint8_t s1 = 0, uint8_t s2 = 0, uint8_t key = 0) {
        tickISR(s1, s2, key);
        checkCallback();
        return EBState;
    }

    // тикер специально для прерывания, не проверяет коллбэки
    uint8_t tickISR(uint8_t s1 = 0, uint8_t s2 = 0, uint8_t key = 0) {
        if (!_isrFlag) {
            _isrFlag = 1;

            // обработка энка (компилятор вырежет блок если не используется)
            // если объявлены два пина или выбран вирт. энкодер или энкодер с кнопкой
            if ((_S1 < 252 && _S2 < 252) || _S1 == VIRT_ENC || _S1 == VIRT_ENCBTN) {
```

```
            uint8_t state;
            if (_S1 >= 252) state = s1 | (s2 << 1);             // получаем код
            else state = fastRead(_S1) | (fastRead(_S2) << 1);  // получаем код
            poolEnc(state);
          }

          // обработка кнопки (компилятор вырежет блок если не используется)
          // если S2 не указан (кнопка) или указан KEY или выбран вирт. энкодер с кнопкой или кнопка
          if ((_S1 < 252 && _S2 == EB_NO_PIN) || _KEY != EB_NO_PIN || _S1 == VIRT_BTN || _S1 == VIRT_ENCBTN) {
            if (_S1 < 252 && _S2 == EB_NO_PIN) _btnState = fastRead(_S1);   // обычная кнопка
            if (_KEY != EB_NO_PIN) _btnState = fastRead(_KEY);              // энк с кнопкой
            if (_S1 == VIRT_BTN) _btnState = s1;                            // вирт кнопка
            if (_S1 == VIRT_ENCBTN) _btnState = key;                        // вирт энк с кнопкой
            _btnState ^= _EB_readFlag(11);                                  // инверсия кнопки
            poolBtn();
          }
        }
        _isrFlag = 0;
        return EBState;
    }

    // ======================================= CALLBACK ====================================

    // проверить callback, чтобы не дёргать в прерывании
    void checkCallback() {
        if (_EB_MODE) {
```

```
        if (turn()) exec(0);
        if (turnH()) exec(12);
        if (EBState > 0 && EBState <= 8) exec(EBState);
        if (release()) exec(10);
        if (hold()) exec(11);
        if (checkFlag(6)) {
            exec(9);
            if (clicks == _amount) exec(13);
        }
        EBState = 0;
    }
}

// подключить обработчик
void attach(eb_callback type, void (*handler)()) {
    _callback[type] = *handler;
}

// отключить обработчик
void detach(eb_callback type) {
    _callback[type] = nullptr;
}

// подключить обработчик на количество кликов (может быть только один!)
void attachClicks(uint8_t amount, void (*handler)()) {
    _amount = amount;
    _callback[13] = *handler;
}
```

```
    // отключить обработчик на количество кликов
    void detachClicks() {
        _callback[13] = nullptr;
    }

    // ====================================== STATUS
=====================================
    uint8_t getState() { return EBState; }      // получить статус
    void resetState() { EBState = 0; }          // сбросить статус

    // ======================================== ENC
=======================================
    bool left() { return checkState(1); }       // поворот влево
    bool right() { return checkState(2); }      // поворот вправо
    bool leftH() { return checkState(3); }      // поворот влево нажатый
    bool rightH() { return checkState(4); }     // поворот вправо нажатый

    bool fast() { return _EB_readFlag(1); }     // быстрый поворот
    bool turn() { return checkFlag(0); }        // энкодер повёрнут
    bool turnH() { return checkFlag(9); }       // энкодер повёрнут нажато
    int8_t getDir() { return _dir; }            // направление последнего поворота, 1 или -
1
    int16_t counter = 0;                        // счётчик энкодера

    // ======================================== BTN
=======================================
    bool press() { return checkState(8); }      // кнопка нажата
    bool release() { return checkFlag(10); }    // кнопка отпущена
    bool click() { return checkState(5); }      // клик по кнопке
```

```
    bool held() { return checkState(6); }       // кнопка удержана

    bool hold() { return _EB_readFlag(4); }     // кнопка удерживается

    bool step() { return checkState(7); }       // режим импульсного удержания

    bool state() { return _btnState; }          // статус кнопки

    bool releaseStep() {return checkFlag(12);}  // кнопка отпущена после импульсного удержания


    uint8_t clicks = 0;                         // счётчик кликов

    bool hasClicks(uint8_t num) { return (clicks == num && checkFlag(7)) ? 1 : 0; } // имеются клики

    uint8_t hasClicks() { return checkFlag(6) ? clicks : 0; }                       // имеются клики


    //
================================================================================

    // ================================== DEPRECATED ==================================

    bool isStep() { return step(); }

    bool isHold() { return hold(); }

    bool isHolded() { return held(); }

    bool isHeld() { return held(); }

    bool isClick() { return click(); }

    bool isRelease() { return release(); }

    bool isPress() { return press(); }

    bool isTurnH() { return turnH(); }

    bool isTurn() { return turn(); }

    bool isFast() { return fast(); }

    bool isLeftH() { return leftH(); }

    bool isRightH() { return rightH(); }

    bool isLeft() { return left(); }
```

```
    bool isRight() { return right(); }


    // ====================================== PRIVATE ======================================

private:
    bool fastRead(const uint8_t pin) {
#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
        if (pin < 8) return bitRead(PIND, pin);
        else if (pin < 14) return bitRead(PINB, pin - 8);
        else if (pin < 20) return bitRead(PINC, pin - 14);
#elif defined(__AVR_ATtiny85__) || defined(__AVR_ATtiny13__)
        return bitRead(PINB, pin);
#else
        return digitalRead(pin);
#endif
        return 0;
    }


    // ===================================== POOL ENC =====================================

    void poolEnc(uint8_t state) {
    #ifdef EB_BETTER_ENC
        if (_prev != state) {
            _ecount += _EB_DIR[state | (_prev << 2)];                // сдвиг внутреннего счётчика
            _prev = state;
            if (state == 0x3 && _ecount != 0) {                      // защёлкнули позицию
                EBState = (_ecount < 0) ? 1 : 2;
                _ecount = 0;
```

```
            if (_S2 == EB_NO_PIN || _KEY != EB_NO_PIN) {          // энкодер с кнопкой
                if (!_EB_readFlag(4) && (_btnState || _EB_readFlag(8))) EBState += 2; // если кнопка не "удерживается"
            }
            _dir = (EBState & 1) ? -1 : 1;                        // направление
            counter += _dir;                                      // счётчик
            if (EBState <= 2) _EB_setFlag(0);                     // флаг поворота для юзера
            else if (EBState <= 4) _EB_setFlag(9);                // флаг нажатого поворота для юзера
            if (millis() - _debTimer < EB_FAST) _EB_setFlag(1);   // быстрый поворот
            else _EB_clrFlag(1);                                  // обычный поворот
            _debTimer = millis();
        }
    }
  #else
    if (_encRST && state == 0b11) {                               // ресет и энк защёлкнул позицию
        if (_S2 == EB_NO_PIN || _KEY != EB_NO_PIN) {              // энкодер с кнопкой
            if ((_prev == 1 || _prev == 2) && !_EB_readFlag(4)) {  // если кнопка не "удерживается" и энкодер в позиции 1 или 2
                EBState = _prev;
                if (_btnState || _EB_readFlag(8)) EBState += 2;
            }
        } else {                                                  // просто энкодер
            if (_prev == 1 || _prev == 2) EBState = _prev;
        }
```

```
        if (EBState > 0) {                                  // был поворот
          _dir = (EBState & 1) ? -1 : 1;                    // направление
          counter += _dir;                                  // счётчик
          if (EBState <= 2) _EB_setFlag(0);                                 // флаг поворота для юзера
          else if (EBState <= 4) _EB_setFlag(9);                            // флаг нажатого поворота для юзера
          if (millis() - _debTimer < EB_FAST) _EB_setFlag(1);     // быстрый поворот
          else _EB_clrFlag(1);                                              // обычный поворот
        }

        _encRST = 0;
        _debTimer = millis();
      }
      if (state == 0b00) _encRST = 1;
      _prev = state;
  #endif
  }

  // ====================================== POOL BTN ======================================
  void poolBtn() {
    uint32_t thisMls = millis();
    uint32_t debounce = thisMls - _debTimer;
    if (_btnState) {                                    // кнопка нажата
      if (!_EB_readFlag(3)) {                           // и не была нажата ранее
        if (debounce > EB_DEB) {                        // и прошел дебаунс
          _EB_setFlag(3);                               // флаг кнопка была нажата
```

```
            _debTimer = thisMls;                            // сброс таймаутов
            EBState = 8;                                    // кнопка нажата
        }
        if (debounce > EB_CLICK) {
                    // кнопка нажата после EB_CLICK
            clicks = 0;
                            // сбросить счётчик и флаг кликов
            flags &= ~0b0011000011100000;                   // clear 5 6 7 12 13
(клики)
        }
    } else {                                                // кнопка уже была нажата
        if (!_EB_readFlag(4)) {                             // и удержание ещё не
зафиксировано
            if (debounce < (_holdT << 7)) {                 // прошло меньше
удержания
                if (EBState != 0 && EBState != 8) _EB_setFlag(2);   // но энкодер
повёрнут! Запомнили
            } else {                                        // прошло больше времени
удержания
                if (!_EB_readFlag(2)) {                     // и энкодер не повёрнут
                    EBState = 6;                            // значит это удержание
(сигнал)
                    _EB_setFlag(4);                         // запомнили что
удерживается
                    _debTimer = thisMls;                    // сброс таймаута
                }
            }
        } else {                                            // удержание зафиксировано
            if (debounce > EB_STEP) {                       // таймер степа
                EBState = 7;                                // сигналим
                _EB_setFlag(13);                            // зафиксирован режим step
                _debTimer = thisMls;                        // сброс таймаута
```

```
                }
            }
        }
    } else {                                        // кнопка не нажата
        if (_EB_readFlag(3)) {                      // но была нажата
            if (debounce > EB_DEB) {
                if (!_EB_readFlag(4) && !_EB_readFlag(2)) {         // энкодер не трогали и не удерживали - это клик
                    EBState = 5;
                    clicks++;
                }
                flags &= ~0b00011100;                   // clear 2 3 4
                _debTimer = thisMls;                    // сброс таймаута
                _EB_setFlag(10);                        // кнопка отпущена
                if (checkFlag(13)) _EB_setFlag(12);             // кнопка отпущена после step
            }
        } else if (clicks > 0 && debounce > EB_CLICK && !_EB_readFlag(5)) flags |= 0b11100000;   // set 5 6 7 (клики)
    }
}


// ================================== MISC ==================================
bool checkState(uint8_t val) {
    if (EBState == val) {
        EBState = 0;
        return 1;
    } return 0;
}
```

```
bool checkFlag(uint8_t val) {
    if (_EB_readFlag(val)) {
        _EB_clrFlag(val);
        return 1;
    } return 0;
}
void exec(uint8_t num) {
    if (*_callback[num]) _callback[num]();
}

uint8_t _prev : 2;
uint8_t EBState : 4;
bool _btnState : 1;
bool _encRST : 1;
bool _isrFlag : 1;
uint16_t flags = 0;

#ifdef EB_BETTER_ENC
int8_t _ecount = 0;
#endif

uint32_t _debTimer = 0;
uint8_t _holdT = (EB_HOLD >> 7);
int8_t _dir = 0;
void (*_callback[_EB_MODE ? 14 : 0])() = {};
uint8_t _amount = 0;

// flags
```

```
    // 0 - enc turn
    // 1 - enc fast
    // 2 - enc был поворот
    // 3 - флаг кнопки
    // 4 - hold
    // 5 - clicks flag
    // 6 - clicks get
    // 7 - clicks get num
    // 8 - enc button hold
    // 9 - enc turn holded
    // 10 - btn released
    // 11 - btn level
    // 12 - btn released after step
    // 13 - step flag

    // EBState
    // 0 - idle
    // 1 - left
    // 2 - right
    // 3 - leftH
    // 4 - rightH
    // 5 - click
    // 6 - held
    // 7 - step
    // 8 - press
};

#endif
```