Создпние програмного обеспечения для трехмерной светодиодной матрицы с расширением 8\*8\*8 на базе платы Arduino Nano и регистров сдвига

#include <Button.h>

#define INVERT\_Y 2 // инвертировать по вертикали (если дождь идёт вверх)

#define INVERT\_X 1 // инвертировать по горизонтали (если текст не читается)

// текст для режима текста

String disp\_text = "HELLO!";

#define XAXIS 0

#define YAXIS 1

#define ZAXIS 2

#define POS\_X 0

#define NEG\_X 1

#define POS\_Z 2

#define NEG\_Z 3

#define POS\_Y 4

#define NEG\_Y 5

#define BUTTON1 18

#define BUTTON2 19

#define RED\_LED 5

#define GREEN\_LED 7

#define TOTAL\_EFFECTS 8 // количество эффектов

#define RAIN 0

#define PLANE\_BOING 1

#define WOOP\_WOOP 2

#define TEXT 3

#define LIT 4

#include <SPI.h> //это синхронный протокол последовательной передачи данных, используемый для связи микроконтроллера с одним или несколькими периферийными устройствами.

#include "Button.h"

#include "fonts.h"

GButton butt1(BUTTON1);

GButton butt2(BUTTON2);

#define RAIN\_TIME 260

#define PLANE\_BOING\_TIME 220

#define WOOP\_WOOP\_TIME 350

#define TEXT\_TIME 300

#define CLOCK\_TIME 500

#define WALKING\_TIME 100

/\*

u - unsigned - беззнаковое

int - integer - целое

8 - кол-во битов

\_t - обозначение того что эта аббревиатура не макрос и не функция и не процедура а ТиП

\*/

uint8\_t charCounter = 0;

uint8\_t charPosition = 0;

uint8\_t cubeSize = 0;

bool cubeExpanding = true;///это было перед функцией вупвуп

uint8\_t xPos;

uint8\_t yPos;

uint8\_t zPos;

uint8\_t selX = 0;

uint8\_t selY = 0;

uint8\_t selZ = 0;

uint8\_t sendDirection = 0;

bool sending = false;

uint8\_t cube[8][8];

int8\_t currentEffect;

uint16\_t timer;

uint16\_t modeTimer;

bool loading;

int8\_t pos;

int8\_t vector[3];

int16\_t coord[3];

uint8\_t len = disp\_text.length();

void setup() {

Serial.begin(9600);

loading = true;

currentEffect = RAIN;

SPI.begin();

SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI\_MODE0)); // 8 МГЦ, старший бит первым , 0 мод

pinMode(RED\_LED, OUTPUT);

pinMode(GREEN\_LED, OUTPUT);

randomSeed(analogRead(0)); //инициализирует генератор псевдо-случайных чисел, запуская его с произвольной точки последовательности случайных чисел, параметром напряжения

digitalWrite(GREEN\_LED, HIGH);

butt1.setIncrStep(5); // настройка инкремента, может быть отрицательным (по умолчанию 1)

butt1.setIncrTimeout(100); // настрйока интервала инкремента (по умолчанию 800 мс)

butt2.setIncrStep(-5); // настройка инкремента, может быть отрицательным (по умолчанию 1)

butt2.setIncrTimeout(100); // настрйока интервала инкремента (по умолчанию 800 мс)

currentEffect = TEXT;

changeMode();

}

void loop() {

butt1.tick();

butt2.tick();

if (butt1.isSingle()) {

currentEffect++;

if (currentEffect >= TOTAL\_EFFECTS) currentEffect = 0;

changeMode();

}

if (butt2.isSingle()) {

currentEffect--;

if (currentEffect < 0) currentEffect = TOTAL\_EFFECTS - 1;

changeMode();

}

if (butt1.isIncr()) { // если кнопка была удержана (это для инкремента)

modeTimer = butt1.getIncr(modeTimer); // увеличивать/уменьшать переменную value с шагом и интервалом

}

if (butt2.isIncr()) { // если кнопка была удержана (это для инкремента)

modeTimer = butt2.getIncr(modeTimer); // увеличивать/уменьшать переменную value с шагом и интервалом

}

case RAIN: rain(); break; // дождь

case PLANE\_BOING: planeBoing(); break;//прорисовка слоев и колон

case WOOP\_WOOP: woopWoop(); break;//дыщащий статичный куб от 2\*2\*2 до 888

case TEXT: text(); break;// РЕЖИМ ПРИ ПОДАЧИ ПИТАНИЯ , вывод текста

case LIT: lit(); break;// горит

case 8: sinusFill(); break;//толстая синуса

case 9: sinusThin(); break;//тонкая

case 10: walkingCube(); break;//продачий куб 2\*2\*2

default: rain(); break;

}

renderCube();

}

void changeMode() {

clearCube();

loading = true;

timer = 0;

randomSeed(millis());

digitalWrite(RED\_LED, HIGH);

digitalWrite(GREEN\_LED, LOW);

delay(500);

digitalWrite(RED\_LED, LOW);

digitalWrite(GREEN\_LED, HIGH);

switch (currentEffect) {

case RAIN: modeTimer = RAIN\_TIME; break;

case PLANE\_BOING: modeTimer = PLANE\_BOING\_TIME; break;

case WOOP\_WOOP: modeTimer = WOOP\_WOOP\_TIME; break;

case TEXT: modeTimer = TEXT\_TIME; break;

case LIT: modeTimer = CLOCK\_TIME; break;

case 8: modeTimer = RAIN\_TIME; break;

case 9: modeTimer = RAIN\_TIME; break;

case 10: modeTimer = WALKING\_TIME; break;

}

modeTimer = RAIN\_TIME;

}

void renderCube() // прорисовка куба через регистры

{

for (uint8\_t i = 0; i < 8; i++) {

digitalWrite(SS, LOW); // выключает все регистры для просчета

if (INVERT\_Y) SPI.transfer(0x01 << (7 - i));// инвертированые слои

else SPI.transfer(0x01 << i);

for (uint8\_t j = 0; j < 8; j++) {

if (INVERT\_X) SPI.transfer(cube[7 - i][j]);// инвертированые коллоны

else SPI.transfer(cube[i][j]);

}

digitalWrite(SS, HIGH); // включает все регистры

//delay(1);

}

}

void walkingCube() // пострение движущегося куба по рандомным координатам

{

if (loading) {

clearCube();

loading = false;

for (byte i = 0; i < 3; i++) {

// координата от о до 700!

coord[i] = 300; //начальная координата

vector[i] = random(3, 8) \* 15; // случайное число от 3 до 7

}

}

timer++;

if (timer > modeTimer) {

timer = 0;

clearCube();

for (byte i = 0; i < 3; i++) {

coord[i] += vector[i];

if (coord[i] < 1) {

coord[i] = 1;

vector[i] = -vector[i];

vector[i] += random(0, 6) - 3;

}

if (coord[i] > 700 - 100) {

coord[i] = 700 - 100;

vector[i] = -vector[i];

vector[i] += random(0, 6) - 3;

}

}

int8\_t thisX = coord[0] / 100;

int8\_t thisY = coord[1] / 100;

int8\_t thisZ = coord[2] / 100;

setVoxel(thisX, thisY, thisZ);

setVoxel(thisX + 1, thisY, thisZ);

setVoxel(thisX, thisY + 1, thisZ);

setVoxel(thisX, thisY, thisZ + 1);

setVoxel(thisX + 1, thisY + 1, thisZ);

setVoxel(thisX, thisY + 1, thisZ + 1);

setVoxel(thisX + 1, thisY, thisZ + 1);

setVoxel(thisX + 1, thisY + 1, thisZ + 1);

}

}

void sinusFill() // заполненая синусоида

{

if (loading) {

clearCube();

loading = false;

}

timer++;

if (timer > modeTimer) {

timer = 0;

clearCube();

if (++pos > 10) pos = 0;

for (uint8\_t i = 0; i < 8; i++) {

for (uint8\_t j = 0; j < 8; j++) {

int8\_t sinZ = 4 + ((float)sin((float)(i + pos) / 2) \* 2);

for (uint8\_t y = 0; y < sinZ; y++) {

setVoxel(i, y, j);

}

}

}

}

}

void sinusThin() // тонкая синусоида (кривая)

{

if (loading) {

clearCube();

loading = false;

}

timer++;

if (timer > modeTimer) {

timer = 0;

clearCube();

if (++pos > 10) pos = 0;

for (uint8\_t i = 0; i < 8; i++) {

for (uint8\_t j = 0; j < 8; j++) {

int8\_t sinZ = 4 + ((float)sin((float)(i + pos) / 2) \* 2);

setVoxel(i, sinZ, j);

}

}

}

}

void rain() // наверное дождь

{

if (loading) {

clearCube();

loading = false;

}

timer++;

if (timer > modeTimer) {

timer = 0;

shift(NEG\_Y);

uint8\_t numDrops = random(0, 5);

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

setVoxel(random(0, 8), 7, random(0, 8));

}

}

}

uint8\_t planePosition = 0;

uint8\_t planeDirection = 0;

bool looped = false;

void planeBoing()// непонятный релакс ???

{

if (loading) {

clearCube();

uint8\_t axis = random(0, 3);

planePosition = random(0, 2) \* 7;

setPlane(axis, planePosition);

if (axis == XAXIS) {

if (planePosition == 0) {

planeDirection = POS\_X;

} else {

planeDirection = NEG\_X;

}

} else if (axis == YAXIS) {

if (planePosition == 0) {

planeDirection = POS\_Y;

} else {

planeDirection = NEG\_Y;

}

} else if (axis == ZAXIS) {

if (planePosition == 0) {

planeDirection = POS\_Z;

} else {

planeDirection = NEG\_Z;

}

}

timer = 0;

looped = false;

loading = false;

}

timer++;

if (timer > modeTimer) {

timer = 0;

shift(planeDirection);

if (planeDirection % 2 == 0) {

planePosition++;

if (planePosition == 7) {

if (looped) {

loading = true;

} else {

planeDirection++;

looped = true;

}

}

} else {

planePosition--;

if (planePosition == 0) {

if (looped) {

loading = true;

} else {

planeDirection--;

looped = true;

}

}

}

}

}

void woopWoop() // статичный дыщащий куб

{

if (loading) {

clearCube();

cubeSize = 2;

cubeExpanding = true;

loading = false;

}

timer++;

if (timer > modeTimer) {

timer = 0;

if (cubeExpanding) {

cubeSize += 2;

if (cubeSize == 8) {

cubeExpanding = false;

}

} else {

cubeSize -= 2;

if (cubeSize == 2) {

cubeExpanding = true;

}

}

clearCube();

drawCube(4 - cubeSize / 2, 4 - cubeSize / 2, 4 - cubeSize / 2, cubeSize);

}

}

void text() {

if (loading) {

clearCube();

charPosition = -1;

charCounter = 0;

loading = false;

}

timer++;

if (timer > modeTimer) {

timer = 0;

shift(NEG\_Z);

charPosition++;

if (charPosition == 7) {

charCounter++;

if (charCounter > len - 1) {

charCounter = 0;

}

charPosition = 0;

}

if (charPosition == 0) {

for (uint8\_t i = 0; i < 8; i++) {

cube[i][0] = getFont(disp\_text[charCounter], i);

}

}

}

}

void lit() {

if (loading) {

clearCube();

for (uint8\_t i = 0; i < 8; i++) {

for (uint8\_t j = 0; j < 8; j++) {

cube[i][j] = 0xFF;

}

}

loading = false;

}

}

void setVoxel(uint8\_t x, uint8\_t y, uint8\_t z) {

cube[7 - y][7 - z] |= (0x01 << x);

}

void clearVoxel(uint8\_t x, uint8\_t y, uint8\_t z) {

cube[7 - y][7 - z] ^= (0x01 << x);

}

void setPlane(uint8\_t axis, uint8\_t i) {

for (uint8\_t j = 0; j < 8; j++) {

for (uint8\_t k = 0; k < 8; k++) {

if (axis == XAXIS) {

setVoxel(i, j, k);

} else if (axis == YAXIS) {

setVoxel(j, i, k);

} else if (axis == ZAXIS) {

setVoxel(j, k, i);

}

}

}

}

void shift(uint8\_t dir) {

if (dir == POS\_X) {

for (uint8\_t y = 0; y < 8; y++) {

for (uint8\_t z = 0; z < 8; z++) {

cube[y][z] = cube[y][z] << 1;

}

}

} else if (dir == NEG\_X) {

for (uint8\_t y = 0; y < 8; y++) {

for (uint8\_t z = 0; z < 8; z++) {

cube[y][z] = cube[y][z] >> 1;

}

}

} else if (dir == POS\_Y) {

for (uint8\_t y = 1; y < 8; y++) {

for (uint8\_t z = 0; z < 8; z++) {

cube[y - 1][z] = cube[y][z];

}

}

for (uint8\_t i = 0; i < 8; i++) {

cube[7][i] = 0;

}

} else if (dir == NEG\_Y) {

for (uint8\_t y = 7; y > 0; y--) {

for (uint8\_t z = 0; z < 8; z++) {

cube[y][z] = cube[y - 1][z];

}

}

for (uint8\_t i = 0; i < 8; i++) {

cube[0][i] = 0;

}

} else if (dir == POS\_Z) {

for (uint8\_t y = 0; y < 8; y++) {

for (uint8\_t z = 1; z < 8; z++) {

cube[y][z - 1] = cube[y][z];

}

}

for (uint8\_t i = 0; i < 8; i++) {

cube[i][7] = 0;

}

} else if (dir == NEG\_Z) {

for (uint8\_t y = 0; y < 8; y++) {

for (uint8\_t z = 7; z > 0; z--) {

cube[y][z] = cube[y][z - 1];

}

}

for (uint8\_t i = 0; i < 8; i++) {

cube[i][0] = 0;

}

}

}

void drawCube(uint8\_t x, uint8\_t y, uint8\_t z, uint8\_t s) // обозначение координат для прорисовки

{

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

setVoxel(x, y + i, z);

setVoxel(x + i, y, z);

setVoxel(x, y, z + i);

setVoxel(x + s - 1, y + i, z + s - 1);

setVoxel(x + i, y + s - 1, z + s - 1);

setVoxel(x + s - 1, y + s - 1, z + i);

setVoxel(x + s - 1, y + i, z);

setVoxel(x, y + i, z + s - 1);

setVoxel(x + i, y + s - 1, z);

setVoxel(x + i, y, z + s - 1);

setVoxel(x + s - 1, y, z + i);

setVoxel(x, y + s - 1, z + i);

}

}

void clearCube()// oчистка куба

{

for (uint8\_t i = 0; i < 8; i++) {

for (uint8\_t j = 0; j < 8; j++) {

cube[i][j] = 0;

}

}}

Файл Button.cpp;

#include "Button.h"

#include <Arduino.h>

GButton::GButton(uint8\_t BUTT) {

\_BUTT = BUTT;

\_debounce = 80;

\_timeout = 500;

\_incr\_timeout = 800;

\_incr\_step = 1;

pinMode (\_BUTT, INPUT\_PULLUP);

}

void GButton::setDebounce(uint8\_t debounce) {

\_debounce = debounce;

}

void GButton::setTimeout(uint16\_t timeout) {

\_timeout = timeout;

}

void GButton::setIncrStep(int16\_t incr\_step) {

\_incr\_step = incr\_step;

}

void GButton::setIncrTimeout(uint16\_t incr\_timeout) {

\_incr\_timeout = incr\_timeout;

}

boolean GButton::isPress() {

if (isPress\_f) {

isPress\_f = false;

return true;

} else return false;

}

boolean GButton::isRelease() {

if (isRelease\_f) {

isRelease\_f = false;

return true;

} else return false;

}

boolean GButton::isHolded() {

if (isHolded\_f) {

isHolded\_f = false;

return true;

} else return false;

}

boolean GButton::isHold() {

if (isHold\_f) return true;

else return false;

}

boolean GButton::isSingle() {

if (counter\_flag && last\_counter == 1) {

counter\_flag = false;

return true;

} else return false;

}

boolean GButton::isDouble() {

if (counter\_flag && last\_counter == 2) {

counter\_flag = false;

return true;

} else return false;

}

boolean GButton::isTriple() {

if (counter\_flag && last\_counter == 3) {

counter\_flag = false;

return true;

} else return false;

}

boolean GButton::hasClicks() {

if (counter\_flag) {

counter\_flag = false;

return true;

} else return false;

}

uint8\_t GButton::getClicks() {

return last\_counter;

}

boolean GButton::isIncr() {

if (incr\_flag) return true;

else return false;

}

int16\_t GButton::getIncr(int16\_t incr\_value) {

if (isHold\_f && (millis() - incr\_timer > \_incr\_timeout)) {

incr\_timer = millis();

incr\_flag = true;

return (incr\_value + \_incr\_step);

} else return incr\_value;

}

void GButton::tick() {

btn\_state = !digitalRead(\_BUTT);

if (btn\_state) isHold\_f = true;

else isHold\_f = false;

if (btn\_state && !btn\_flag && (millis() - btn\_timer > \_debounce)) {

btn\_flag = true;

btn\_counter++;

btn\_timer = millis();

isPress\_f = true;

}

if (!btn\_state && btn\_flag) {

btn\_flag = false;

hold\_flag = false;

isRelease\_f = true;

btn\_timer = millis();

incr\_flag = false;

}

if (btn\_flag && btn\_state && (millis() - btn\_timer > \_timeout) && !hold\_flag) {

hold\_flag = true;

btn\_counter = 0;

isHolded\_f = true;

incr\_flag = true;

incr\_timer = millis();

}

if ((millis() - btn\_timer > \_timeout) && (btn\_counter != 0)) {

last\_counter = btn\_counter;

btn\_counter = 0;

counter\_flag = true;

}

}

Файл Button.h;

#ifndef Button\_h

#define Button\_h

#include <Arduino.h>

class GButton

{

public:

GButton(uint8\_t BUTT);

void setDebounce(uint8\_t debounce);

void setTimeout(uint16\_t timeout);

void tick();

boolean isPress();

boolean isRelease();

boolean isHolded();

boolean isHold();

boolean hasClicks();

uint8\_t getClicks();

boolean isSingle();

boolean isDouble();

boolean isTriple();

boolean isIncr();

void setIncrStep(int16\_t incr\_step);

void setIncrTimeout(uint16\_t incr\_timeout);

int16\_t getIncr(int16\_t incr\_value);

private:

uint8\_t \_BUTT;

uint8\_t \_debounce;

uint16\_t \_timeout;

uint8\_t btn\_counter, last\_counter;

boolean btn\_state, btn\_flag, hold\_flag, counter\_flag;

uint32\_t btn\_timer, incr\_timer;

boolean isHold\_f, isHolded\_f, isRelease\_f, isPress\_f, incr\_flag;

int16\_t \_incr\_step;

uint16\_t \_incr\_timeout;

};

#endif

Файл keywords.txt;

#######################################

# Syntax Coloring Map For Button

#######################################

#######################################

# Datatypes (KEYWORD1)

#######################################

GButton KEYWORD1

setDebounce KEYWORD1

setTimeout KEYWORD1

#######################################

# Methods and Functions (KEYWORD2)

#######################################

tick KEYWORD2

isPress KEYWORD2

isRelease KEYWORD2

isHolded KEYWORD2

isHold KEYWORD2

hasClicks KEYWORD2

getClicks KEYWORD2

isSingle KEYWORD2

isDouble KEYWORD2

isTriple KEYWORD2

isIncr KEYWORD2

setIncrStep KEYWORD2

setIncrTimeout KEYWORD2

getIncr KEYWORD2

Файл button\_example.ino;

#define PIN 3 // пин на Ардуинке

#include "Button.h"

GButton butt1(PIN);

void setup() {

Serial.begin(9600);

butt1.setDebounce(50); // настройка антидребезга (по умолчанию 80)

butt1.setTimeout(300); // настройка таймаута (по умолчанию 500)

}

void loop() {

butt1.tick(); // обязательная функция отработки. Должна постоянно опрашиваться

if (butt1.isSingle()) Serial.println("Single"); // проверка на один клик

if (butt1.isDouble()) Serial.println("Double"); // проверка на двойной клик

if (butt1.isTriple()) Serial.println("Triple"); // проверка на тройной клик

if (butt1.hasClicks()) // проверка на наличие нажатий

Serial.println(butt1.getClicks()); // получить (и вывести) число нажатий

if (butt1.isPress()) Serial.println("Press"); // нажатие на кнопку (+ дебаунс)

if (butt1.isRelease()) Serial.println("Release"); // отпускание кнопки (+ дебаунс)

if (butt1.isHolded()) Serial.println("Holded"); // проверка на удержание

}

Файл Gbutton\_example.ino;

#define PIN 3 // пин на Ардуинке

#include "Button.h"

GButton butt1(PIN);

int value = 0;

void setup() {

Serial.begin(9600);

butt1.setDebounce(50); // настройка антидребезга (по умолчанию 80 мс)

butt1.setTimeout(300); // настройка таймаута на удержание (по умолчанию 500 мс)

butt1.setIncrStep(2); // настройка инкремента, может быть отрицательным (по умолчанию 1)

butt1.setIncrTimeout(500); // настрйока интервала инкремента (по умолчанию 800 мс)

}

void loop() {

butt1.tick(); // обязательная функция отработки. Должна постоянно опрашиваться

if (butt1.isSingle()) Serial.println("Single"); // проверка на один клик

if (butt1.isDouble()) Serial.println("Double"); // проверка на двойной клик

if (butt1.isTriple()) Serial.println("Triple"); // проверка на тройной клик

if (butt1.hasClicks()) // проверка на наличие нажатий

Serial.println(butt1.getClicks()); // получить (и вывести) число нажатий

if (butt1.isPress()) Serial.println("Press"); // нажатие на кнопку (+ дебаунс)

if (butt1.isRelease()) Serial.println("Release"); // отпускание кнопки (+ дебаунс)

if (butt1.isHolded()) Serial.println("Holded"); // проверка на удержание

if (butt1.isIncr()) { // если кнопка была удержана (это для инкремента)

value = butt1.getIncr(value); // увеличивать/уменьшать переменную value с шагом и интервалом

Serial.println(value); // для примера выведем в порт

}

}

Файл Gbutton\_timer\_interrupt.ino;

#define PIN 3 // пин на Ардуинке

#include "Button.h"

#include "TimerOne.h"

GButton butt1(PIN);

int value = 0;

void setup() {

Serial.begin(9600);

Timer1.initialize(10000); // установка таймера на каждые 10000 микросекунд (= 10 мс)

Timer1.attachInterrupt(timerIsr); // запуск таймера

butt1.setDebounce(50); // настройка антидребезга (по умолчанию 80 мс)

butt1.setTimeout(300); // настройка таймаута на удержание (по умолчанию 500 мс)

butt1.setIncrStep(2); // настройка инкремента, может быть отрицательным (по умолчанию 1)

butt1.setIncrTimeout(500); // настрйока интервала инкремента (по умолчанию 800 мс)}

void timerIsr() { // прерывание таймера

butt1.tick(); // отработка теперь находится здесь}

void loop() {

if (butt1.isSingle()) Serial.println("Single"); // проверка на один клик

if (butt1.isDouble()) Serial.println("Double"); // проверка на двойной клик

if (butt1.isTriple()) Serial.println("Triple"); // проверка на тройной клик

if (butt1.hasClicks()) // проверка на наличие нажатий

Serial.println(butt1.getClicks()); // получить (и вывести) число нажатий

if (butt1.isPress()) Serial.println("Press"); // нажатие на кнопку (+ дебаунс)

if (butt1.isRelease()) Serial.println("Release"); // отпускание кнопки (+ дебаунс)

if (butt1.isHolded()) Serial.println("Holded"); // проверка на удержание

//if (butt1.isHold()) Serial.println("Hold"); // возвращает состояние кнопки

if (butt1.isIncr()) { // если кнопка была удержана (это для инкремента)

value = butt1.getIncr(value); // увеличивать/уменьшать переменную value с шагом и интервалом

Serial.println(value); // для примера выведем в порт}

}

Файл Gbutton\_timer\_interrupt.ino;

const uint8\_t fontHEX[][8] = {

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},// 0x20 32

{0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x00, 0x02},

// ! 0x21 33

{0x05, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},

// " 0x22 34

{0x28, 0x28, 0x7E, 0x14, 0x14, 0x3F, 0x0A, 0x0A},

// # 0x23 35

{0x04, 0x1E, 0x05, 0x05, 0x0E, 0x14, 0x14, 0x0F},

// $ 0x24 36

{0x46, 0x49, 0x29, 0x26, 0x90, 0x50, 0x48, 0x88},

// % 0x25 37

{0x06, 0x09, 0x09, 0x26, 0x29, 0x11, 0x31, 0x4E},

// & 0x26 38

{0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},

// ' 0x27 39

{0x02, 0x02, 0x01, 0x01, 0x01, 0x01, 0x01, 0x02},

// ( 0x28 40

{0x02, 0x02, 0x04, 0x04, 0x04, 0x04, 0x04, 0x02},

// ) 0x29 41

{0x15, 0x0E, 0x15, 0x04, 0x00, 0x00, 0x00, 0x00},

// \* 0x2a 42

{0x00, 0x08, 0x08, 0x08, 0x7F, 0x08, 0x08, 0x08},

// + 0x2b 43

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x02},

// , 0x2c 44

{0x00, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00},

// - 0x2d 45

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x02},

// . 0x2e 46

{0x04, 0x04, 0x02, 0x02, 0x02, 0x02, 0x02, 0x01},

// / 0x2f 47

{0x0E, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x0E},

// 0 0x30 48

{0x04, 0x06, 0x04, 0x04, 0x04, 0x04, 0x04, 0x0E},

// 1 0x31 49

{0x0E, 0x11, 0x10, 0x08, 0x04, 0x02, 0x01, 0x1F},

// 2 0x32 50

{0x0E, 0x11, 0x10, 0x0C, 0x10, 0x10, 0x11, 0x0E},

// 3 0x33 51

{0x08, 0x0C, 0x0A, 0x09, 0x1F, 0x08, 0x08, 0x08},

// 4 0x34 52

{0x1F, 0x01, 0x01, 0x0F, 0x10, 0x10, 0x11, 0x0E},

// 5 0x35 53

{0x0C, 0x02, 0x01, 0x0F, 0x11, 0x11, 0x11, 0x0E},

// 6 0x36 54

{0x1F, 0x10, 0x08, 0x08, 0x04, 0x04, 0x02, 0x02},

// 7 0x37 55

{0x0E, 0x11, 0x11, 0x0E, 0x11, 0x11, 0x11, 0x0E},

// 8 0x38 56

{0x0E, 0x11, 0x11, 0x11, 0x1E, 0x10, 0x08, 0x06},

// 9 0x39 57

{0x00, 0x00, 0x02, 0x02, 0x00, 0x00, 0x02, 0x02},

// : 0x3a 58

{0x00, 0x00, 0x02, 0x02, 0x00, 0x00, 0x02, 0x02},

// ; 0x3b 59

{0x00, 0x40, 0x30, 0x0C, 0x02, 0x0C, 0x30, 0x40},

// < 0x3c 60

{0x00, 0x00, 0x00, 0x7F, 0x00, 0x7F, 0x00, 0x00},

// = 0x3d 61

{0x00, 0x02, 0x0C, 0x30, 0x40, 0x30, 0x0C, 0x02},

// > 0x3e 62

{0x07, 0x08, 0x08, 0x04, 0x02, 0x02, 0x00, 0x02},

// ? 0x3f 63

{0x7C, 0x82, 0x39, 0x25, 0x25, 0x25, 0xF9, 0x02},

// @ 0x40 64

{0x18, 0x18, 0x24, 0x24, 0x24, 0x7E, 0x42, 0x42},

// A 0x41 65

{0x1E, 0x22, 0x22, 0x1E, 0x22, 0x22, 0x22, 0x1E},

// B 0x42 66

{0x78, 0x04, 0x02, 0x02, 0x02, 0x02, 0x04, 0x78},

// C 0x43 67

{0x1E, 0x22, 0x42, 0x42, 0x42, 0x42, 0x22, 0x1E},

// D 0x44 68

{0x3E, 0x02, 0x02, 0x1E, 0x02, 0x02, 0x02, 0x3E},

// E 0x45 69

{0x3E, 0x02, 0x02, 0x3E, 0x02, 0x02, 0x02, 0x02},

// F 0x46 70

{0x78, 0x04, 0x02, 0x02, 0x72, 0x42, 0x44, 0x78},

// G 0x47 71

{0x42, 0x42, 0x42, 0x7E, 0x42, 0x42, 0x42, 0x42},

// H 0x48 72

{0x1C, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x1C},

// I 0x49 73

{0x38, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x1C},

// J 0x4a 74

{0x22, 0x12, 0x0A, 0x06, 0x06, 0x0A, 0x12, 0x22},

// K 0x4b 75

{0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x3C},

// L 0x4c 76

{0x63, 0x63, 0x55, 0x55, 0x49, 0x49, 0x41, 0x41},

// M 0x4d 77

{0x46, 0x46, 0x4A, 0x4A, 0x52, 0x52, 0x62, 0x62},

// N 0x4e 78

{0x1C, 0x22, 0x41, 0x41, 0x41, 0x41, 0x22, 0x1C},

// O 0x4f 79

{0x1E, 0x22, 0x22, 0x22, 0x1E, 0x02, 0x02, 0x02},

// P 0x50 80

{0x1C, 0x22, 0x41, 0x41, 0x41, 0x41, 0x22, 0x1C},

// Q 0x51 81

{0x1E, 0x22, 0x22, 0x22, 0x1E, 0x12, 0x22, 0x42},

// R 0x52 82

{0x3C, 0x02, 0x02, 0x1C, 0x20, 0x20, 0x20, 0x1E},

// S 0x53 83

{0x3E, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08},

// T 0x54 84

{0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x3C},

// U 0x55 85

{0x22, 0x22, 0x22, 0x14, 0x14, 0x14, 0x08, 0x08},

// V 0x56 86

{0x91, 0x91, 0x91, 0xAA, 0xAA, 0xAA, 0x44, 0x44},

// W 0x57 87

{0x22, 0x22, 0x14, 0x08, 0x08, 0x14, 0x22, 0x22},

// X 0x58 88

{0x22, 0x22, 0x14, 0x14, 0x08, 0x08, 0x08, 0x08},

// Y 0x59 89

{0x3E, 0x20, 0x10, 0x08, 0x08, 0x04, 0x02, 0x3E},

// Z 0x5a 90

{0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10},

// [ 0x5b 91

{0x01, 0x01, 0x02, 0x02, 0x02, 0x02, 0x02, 0x04},

// \ 0x5c 92

{0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04},

// ] 0x5d 93

{0x08, 0x14, 0x22, 0x41, 0x00, 0x00, 0x00, 0x00},

// ^ 0x5e 94

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},

// \_ 0x5f 95

{0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},

// ` 0x60 96

{0x00, 0x00, 0x1C, 0x20, 0x3C, 0x22, 0x22, 0x3C},

// a 0x61 97

{0x02, 0x02, 0x1E, 0x22, 0x22, 0x22, 0x22, 0x1E},

// b 0x62 98

{0x00, 0x00, 0x38, 0x04, 0x04, 0x04, 0x04, 0x38},

// c 0x63 99

{0x20, 0x20, 0x3C, 0x22, 0x22, 0x22, 0x22, 0x3C},

// d 0x64 100

{0x00, 0x00, 0x1C, 0x22, 0x3E, 0x02, 0x22, 0x1C},

// e 0x65 101

{0x30, 0x08, 0x08, 0x1C, 0x08, 0x08, 0x08, 0x08},

// f 0x66 102

{0x3C, 0x22, 0x22, 0x22, 0x3C, 0x20, 0x20, 0x1C},

// g 0x67 103

{0x02, 0x02, 0x02, 0x1E, 0x22, 0x22, 0x22, 0x22},

// h 0x68 104

{0x10, 0x00, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10},

// i 0x69 105

{0x10, 0x00, 0x18, 0x10, 0x10, 0x10, 0x10, 0x0C},

// j 0x6a 106

{0x02, 0x02, 0x12, 0x0A, 0x06, 0x0A, 0x12, 0x22},

// k 0x6b 107

{0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08},

// l 0x6c 108

{0x00, 0x00, 0x37, 0x49, 0x49, 0x49, 0x49, 0x49},

// m 0x6d 109

{0x00, 0x00, 0x1E, 0x22, 0x22, 0x22, 0x22, 0x22},

// n 0x6e 110

{0x00, 0x00, 0x1C, 0x22, 0x22, 0x22, 0x22, 0x1C},

// o 0x6f 111

{0x1E, 0x22, 0x22, 0x22, 0x1E, 0x02, 0x02, 0x02},

// p 0x70 112

{0x3C, 0x22, 0x22, 0x22, 0x3C, 0x20, 0x20, 0x20},

// q 0x71 113

{0x00, 0x00, 0x34, 0x0C, 0x04, 0x04, 0x04, 0x04},

// r 0x72 114

{0x00, 0x00, 0x38, 0x04, 0x0C, 0x30, 0x20, 0x1C},

// s 0x73 115

{0x08, 0x08, 0x3C, 0x08, 0x08, 0x08, 0x08, 0x30},

// t 0x74 116

{0x00, 0x00, 0x22, 0x22, 0x22, 0x22, 0x22, 0x3C},

// u 0x75 117

{0x00, 0x00, 0x22, 0x22, 0x14, 0x14, 0x08, 0x08},

// v 0x76 118

{0x00, 0x00, 0x49, 0x49, 0x55, 0x55, 0x22, 0x22},

// w 0x77 119

{0x00, 0x00, 0x22, 0x14, 0x08, 0x08, 0x14, 0x22},

// x 0x78 120

{0x00, 0x00, 0x12, 0x12, 0x12, 0x1C, 0x10, 0x0C},

// y 0x79 121

{0x00, 0x00, 0x3C, 0x20, 0x10, 0x08, 0x04, 0x3C},

// z 0x7a 122

{0x04, 0x04, 0x04, 0x04, 0x03, 0x04, 0x04, 0x04},//{ 0x7b 123

{0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02},//| 0x7c 124

{0x02, 0x02, 0x02, 0x02, 0x0C, 0x02, 0x02, 0x02},//} 0x7d 125

{0x00, 0x00, 0x00, 0x46, 0x49, 0x31, 0x00, 0x00},//~ 0x7e 126

// интерпретатор кода символа по ASCII в его номер в массиве fontHEX

uint8\_t getFont(uint8\_t font, uint8\_t row) {

font = font - '0' + 16; // перевод код символа из таблицы ASCII в номер согласно нумерации массива

if (font < 126) return fontHEX[font][row];

}