4. Разработка ПО контроллера STM32F303CBT6 для расширения функционала устройства.

4.1 Установка даты и времени.

Для хранения даты и времени в контроллере STM32F303CBT6 предусмотрен аппаратный блок RTC (Real-Time Clock, часы реального времени).

Функция инициализации RTC:

static RTC\_HandleTypeDef handleRTC =

{

RTC,

**{**

RTC\_HOURFORMAT\_24,

127,

255,

RTC\_OUTPUT\_DISABLE,

RTC\_OUTPUT\_POLARITY\_HIGH,

RTC\_OUTPUT\_TYPE\_OPENDRAIN

**}**

};

void HAL\_RTC::Init()

{

// Включаем тактирование блока RTC

\_\_HAL\_RCC\_RTC\_ENABLE();

// Вызываем библиотечную функцию инициализации блока RTC

HAL\_RTC\_Init(&handleRTC);

#define VALUE\_FOR\_RTC 0x644

// Если сохраннённое в энергонезависимой памяти не совпадает с тем, что там должно быть, значит, прошивка запускается впервые

if (HAL\_RTCEx\_BKUPRead((RTC\_HandleTypeDef \*)&handleRTC, RTC\_BKP\_DR0) != VALUE\_FOR\_RTC)

{

// И в этом случае мы устанавливаем значение времени «по умолчанию»

SetTime(PackedTime(11, 11, 11, 11, 11, 11));

// И сохраняем признак того, что время установлено

HAL\_RTCEx\_BKUPWrite((RTC\_HandleTypeDef \*)&handleRTC, RTC\_BKP\_DR0, VALUE\_FOR\_RTC);

}

}

Функция получения текущих даты и времени:

// Структура для хранения даты и времени

struct PackedTime

{

uint hours : 5;

uint minutes : 6;

uint seconds : 6;

uint year : 6; // Год отсчитывается от 2000, т.е. 2023 будет 23

uint month : 4;

uint day : 5;

PackedTime(uint h = 11, uint m = 11, uint s = 11, uint d = 11, uint mo = 11, uint y = 11) :

hours(h), minutes(m), seconds(s), year(y), month(mo), day(d) {}

String<> ToString() const;

uint ToSecs() const;

};

PackedTime HAL\_RTC::GetTime()

**{**

RTC\_TimeTypeDef time;

RTC\_DateTypeDef date;

// Вызываем библиотечную функцию получения текущего времени

HAL\_RTC\_GetTime(&handleRTC, &time, RTC\_HOURFORMAT\_24);

// Вызываем библиотечную функцию получения текущей даты

HAL\_RTC\_GetDate(&handleRTC, &date, RTC\_HOURFORMAT\_24);

// Возвращаем время

return PackedTime(time.Hours, time.Minutes, time.Seconds, date.Date, date.Month, date.Year);

**}**

Функция установки текущих даты и времени:

void HAL\_RTC::SetTime(const PackedTime &pack\_time)

{

// Заполняем библиотечную структуру времени

RTC\_TimeTypeDef time =

{

(uint8)pack\_time.hours,

(uint8)pack\_time.minutes,

(uint8)pack\_time.seconds

};

// Заполняем библиотечную структуру даты

RTC\_DateTypeDef date =

{

0,

(uint8)pack\_time.month,

(uint8)pack\_time.day,

(uint8)pack\_time.year

};

// Вызываем библиотечную структуру установки времени

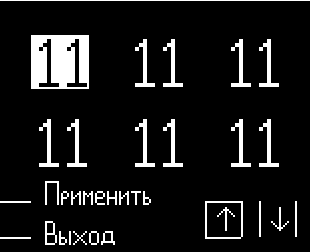
HAL\_RTC\_SetTime(&handleRTC, &time, RTC\_HOURFORMAT\_24);

// Вызываем библиотечную структуру установки даты

HAL\_RTC\_SetDate(&handleRTC, &date, RTC\_HOURFORMAT\_24);

}

В меню предусмотрена страница для установки текущего времени.



Она состоит из следующих элементов:

- 6 полей для ввода двухзначных цифр для установки года, месяца, числа, часа, минут и секунд;

- кнопка «Применит» для сохранения времени в устройстве и выхода из меню;

- кнопка «Выход» для выхода из меню без сохранения изменений;

- стрелка «Вверх» для увеличения текущего значения;

- стрелка «Вниз» для уменьшения текущего значения.

Программно страница установки времени реализована следующим образом:

struct TimeItem : public Item

{

// Функция отрисовки страницы в закрытом состоянии

void DrawClosed(int x, int y, bool active) const;

// Функция отрисовки страницы в открытом состоянии

void DrawOpened(int x, int y, bool active) const;

// Обработка нажатия кнопки

void ShortPressure(Key::E) const;

// Обработка нажатия кнопки с удержанием

void LongPressure(Key::E) const;

// Обработка двойного нажатия кнопки

void DoubleClick() const;

const DTimeItem \*ToDTimeItem() const { return (DTimeItem \*)this; }

private:

// Изменение значения в текущем поле

void ChangeCurrentField(Key::E) const;

};

void TimeItem::DrawClosed(int x, int y, bool) const

{

PackedTime time = HAL\_RTC::GetTime();

y += 4;

x += 15;

String<>("%02d:%02d:%02d", time.hours, time.minutes, time.seconds).Draw(x, y, Color::WHITE);

String<>("%02d:%02d:%04d", time.day, time.month, time.year + 2000).Draw(x + 70, y);

}

void TimeItem::LongPressure(Key::E) const

{

if (IsOpened())

{

\*ToDTimeItem()->state = 0;

}

else

{

Open();

}

}

void TimeItem::ChangeCurrentField(Key::E key) const

{

const DTimeItem \*data = ToDTimeItem();

if (\*data->cur\_field < 6)

{

static const int max[6] = { 23, 59, 59, 31, 12, 99 };

PackedTime &time = \*data->time;

uint values[6] = { time.hours, time.minutes, time.seconds,

time.day, time.month, time.year };

int value = (int)values[\*data->cur\_field];

if (key == Key::\_1)

{

Math::CircleIncrease(&value, 0, max[\*data->cur\_field]);

}

else if (key == Key::\_2)

{

Math::CircleDecrease(&value, 0, max[\*data->cur\_field]);

}

values[\*data->cur\_field] = (uint)value;

}

}

void TimeItem::ShortPressure(Key::E key) const

{

if (IsOpened())

{

const DTimeItem \*data = ToDTimeItem();

if (\*data->state == 0)

{

if (key == Key::\_1)

{

Math::CircleIncrease<int>(data->cur\_field, 0, 7);

}

else if (key == Key::\_2)

{

if (\*data->cur\_field == 6)

{

HAL\_RTC::SetTime(\*data->time);

Close();

}

else if (\*data->cur\_field == 7)

{

Close();

}

else

{

\*data->state = 1;

}

}

}

else if (\*data->state == 1)

{

ChangeCurrentField(key);

}

}

}

void TimeItem::DrawOpened(int, int, bool) const

{

const DTimeItem \*data = ToDTimeItem();

Display::BeginScene(Color::BLACK);

Rectangle(Display::WIDTH - 1, Display::HEIGHT - 1).Draw(0, 0, Color::WHITE);

int x0 = 20;

int y0 = 20;

int dX = 48;

int dY = 40;

uint values[6] = { data->*time*->hours, data->*time*->minutes, data->*time*->seconds,

data->*time*->day, data->*time*->month, data->*time*->year };

for (int i = 0; i < 6; i++)

{

Color::E color = Color::WHITE;

int x = x0 + dX \* (i % 3);

int y = y0 + (i < 3 ? 0 : dY);

if (i == \*data->cur\_field)

{

Rectangle(29, 27).Fill(x - 2, y - 2, Color::WHITE);

color = Color::BLACK;

}

Font::Text::DrawBig(x, y, 2, String<>("%02d", values[i]).c\_str(), color);

}

int x = 105;

int y = 101;

int size = 18;

int dT = 3;

Rectangle rect(size, size);

rect.Draw(x, y, Color::WHITE);

String<>(\*data->state == 0 ? "П" : "\x80").Draw(x + dT + 3, y + dT);

x += 27;

rect.Draw(x, y);

String<>(\*data->state == 0 ? "В" : "\x81").Draw(x + dT + 3, y + dT);

Color::E color = Color::WHITE;

if (\*data->cur\_field == 6)

{

Rectangle(56, 15).Fill(23, y + dT - 15, Color::WHITE);

color = Color::BLACK;

}

String<>("Применить").Draw(25, y + dT - 13, color);

color = Color::WHITE;

if (\*data->cur\_field == 7)

{

Rectangle(56, 15).Fill(23, y + dT + 5, Color::WHITE);

color = Color::BLACK;

}

String<>("Выход").Draw(25, y + dT + 7, color);

}

4.2 Измерение напряжения батареи.

Для измерения напряжения батареи используется встроенный в контроллер блок ADC (Analog-to-Digital Converter, аналогово-цифровой преобразователь).

Функция инициализации блока ADC:

void HAL\_ADC::Init()

{

pinADC.Init();

pinHumidity.Init();

handleADC.Instance = ADC1;

handleADC.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV1;

handleADC.Init.Resolution = ADC\_RESOLUTION\_12B;

handleADC.Init.ScanConvMode = ADC\_SCAN\_DISABLE;

handleADC.Init.ContinuousConvMode = DISABLE;

handleADC.Init.DiscontinuousConvMode = DISABLE;

handleADC.Init.NbrOfDiscConversion = 0;

handleADC.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

handleADC.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

handleADC.Init.NbrOfConversion = 1;

HAL\_ADC\_Init(&handleADC);

HAL\_NVIC\_SetPriority(ADC1\_IRQn, 1, 1);

}

Функция чтения значения ADC:

uint HAL\_ADC::ReadChannel(uint channel)

**{**

ADC\_ChannelConfTypeDef config = { 0 };

config.Channel = channel;

config.Rank = ADC\_REGULAR\_RANK\_1;

config.SamplingTime = ADC\_SAMPLETIME\_601CYCLES\_5;

uint value = 0;

if (HAL\_ADC\_ConfigChannel(&handleADC, &config) == HAL\_OK)

{

HAL\_NVIC\_EnableIRQ(ADC1\_IRQn);

flag\_ready = false;

if (HAL\_ADC\_Start\_IT(&handleADC) == HAL\_OK)

{

while (!flag\_ready)

{

}

value = HAL\_ADC\_GetValue(&handleADC);

}

}

return value;

**}**

Функция измерения напряжения:

float HAL\_ADC::GetVoltage()

{

static TimeMeterMS meter;

static float voltage = 0.0f;

if (meter.IsFinished())

{

voltage = (float)ReadChannel(ADC\_CHANNEL\_4) \* 3.3f \* 1.25f / (float)(1 << 12);

meter.FinishAfter(1000);

}

return voltage;

}