#include "defines.h"

#include "Device.h"

#include "Modules/BME280/BME280.h"

#include "Modules/BH1750/BH1750.h"

#include "Modules/ST7735/ST7735.h"

#include "Modules/HI50/HI50.h"

#include "Hardware/Timer.h"

#include "Hardware/InterCom.h"

#include "Hardware/Keyboard.h"

#include "Hardware/Beeper.h"

#include "Hardware/EnergySwitch.h"

#include "Menu/Menu.h"

#include "SCPI/SCPI.h"

#include "Modules/L00256L/L00256L.h"

namespace Device

{

static void ProcessMeasure(const Measure &, uint time);

}

void Device::Init()

{

HAL::Init();

ST7735::Init();

EnergySwitch::Init();

BME280::Init(); // Пытаемся инициализировать датчик давления

if (!BH1750::Init()) // Если нет - то освещённость

{

\_\_HAL\_RCC\_I2C1\_CLK\_DISABLE();

if (!HI50::Init()) // Если нет - то датчик дальности

{

L00256L::Init();

}

}

if (!HI50::IsExist()) // Если обнаружен дальномер, то не включаем HC12 на передачу - HI50 сам будет его включать,

{ // когда понадобится

HAL\_USART1::SetModeHC12();

}

Keyboard::Init();

Beeper::Init();

InterCom::SetDirection((Direction::E)(Direction::HC12 | Direction::Display));

}

void Device::Update()

{

Measure temp;

Measure pressure;

Measure humidity;

Measure dew\_point;

Measure illuminate;

Measure distance;

uint time = TIME\_MS;

if (BME280::GetMeasures(&temp, &pressure, &humidity, &dew\_point))

{

ProcessMeasure(temp, time);

ProcessMeasure(pressure, time);

ProcessMeasure(humidity, time);

ProcessMeasure(dew\_point, time);

}

if (BH1750::GetMeasure(&illuminate))

{

ProcessMeasure(illuminate, time);

}

if (HI50::GetMeasure(&distance))

{

InterCom::SetDirection(Direction::Display);

ProcessMeasure(distance, time);

InterCom::SetDirection((Direction::E)(Direction::HC12 | Direction::Display));

}

if (!Menu::IsOpened())

{

Beeper::Update();

}

Keyboard::Update();

Display::Update(TIME\_MS);

HAL\_ADC::GetVoltage();

EnergySwitch::Update();

HAL\_USART1::Update();

HI50::Update();

SCPI::Update();

L00256L::Update();

}

void Device::ProcessMeasure(const Measure &measure, uint time)

{

if (measure.correct)

{

InterCom::Send(measure, time);

}

}

#pragma once

namespace L00256L

{

void Init();

void Update();

void CallbackOnInterrupt();

bool IsEnabled();

float GetAngleFull();

float GetAngleRelative();

float GetVelocity();

}

#include "defines.h"

#include "Modules/L00256L/L00256L.h"

#include "Hardware/Timer.h"

#include <stm32f3xx\_hal.h>

namespace L00256L

{

TimeMeterMS meter;

bool pressed = false; // Если true, клавиша нажата

bool taboo\_long = false; // Если true, запрещено длинное срабатывание

#define STATE\_A (HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_0) == GPIO\_PIN\_SET)

#define STATE\_B (HAL\_GPIO\_ReadPin(GPIOB, GPIO\_PIN\_1) == GPIO\_PIN\_SET)

static bool prev\_a = false;

static const float step\_angle = 360.0f / 256.0f;

static float angle\_full = 0;

static bool is\_enabled = false;

namespace Velocity

{

static uint time\_start = 0;

static float angle = 0.0f;

static float speed = 0.0f;

static void CallbackOnInterrupt(float delta)

{

angle += delta;

}

static void Update()

{

static uint delta\_time = 256;

if (time\_start == 0)

{

time\_start = TIME\_MS;

}

if (TIME\_MS - time\_start >= delta\_time)

{

speed = angle / (float)(TIME\_MS - time\_start) \* 1e3f;

angle = 0;

time\_start = TIME\_MS;

}

}

}

}

void L00256L::Init()

{

GPIO\_InitTypeDef is =

{

GPIO\_PIN\_0,

GPIO\_MODE\_IT\_RISING\_FALLING,

GPIO\_NOPULL,

GPIO\_SPEED\_FREQ\_HIGH,

0

};

HAL\_GPIO\_Init(GPIOA, &is);

is.Pin = GPIO\_PIN\_1;

is.Mode = GPIO\_MODE\_INPUT;

HAL\_GPIO\_Init(GPIOB, &is);

HAL\_NVIC\_SetPriority(EXTI0\_IRQn, 2, 0);

HAL\_NVIC\_EnableIRQ(EXTI0\_IRQn);

is\_enabled = true;

}

bool L00256L::IsEnabled()

{

return is\_enabled;

}

void L00256L::CallbackOnInterrupt()

{

static bool first = true;

float before = angle\_full;

if (first)

{

first = false;

prev\_a = STATE\_A;

return;

}

bool state\_a = STATE\_A;

if (state\_a && !prev\_a)

{

if (STATE\_B)

{

angle\_full -= step\_angle;

}

else

{

angle\_full += step\_angle;

}

}

prev\_a = state\_a;

Velocity::CallbackOnInterrupt(angle\_full - before);

}

void L00256L::Update()

{

Velocity::Update();

}

float L00256L::GetAngleFull()

{

return (float)angle\_full;

}

float L00256L::GetAngleRelative()

{

float angle = angle\_full;

while (angle < 0.0f)

{

angle += 360.0f;

}

while (angle >= 360.0f)

{

angle -= 360.0f;

}

return angle;

}

float L00256L::GetVelocity()

{

return Velocity::speed;

}

#pragma once

#include <stm32f3xx\_hal.h>

struct Pin

{

Pin(GPIO\_TypeDef \*\_gpio, uint16 \_pin, uint \_mode, uint \_pull, uint \_alternate = 0U) :

gpio(\_gpio), pin(\_pin), mode(\_mode), pull(\_pull), alternate(\_alternate) { }

void Init();

void ToLow();

void ToHi();

void Set(bool);

bool IsHi() const;

bool IsLow() const;

private:

GPIO\_TypeDef \*gpio;

uint16 pin;

uint mode;

uint pull;

uint alternate;

};

struct PinInput : public Pin

{

PinInput(GPIO\_TypeDef \*\_gpio, uint16 \_pin, uint pull) : Pin(\_gpio, \_pin, GPIO\_MODE\_INPUT, pull) { }

};

struct PinOutputPP : public Pin

{

PinOutputPP(GPIO\_TypeDef \*\_gpio, uint16 \_pin, uint pull) : Pin(\_gpio, \_pin, GPIO\_MODE\_OUTPUT\_PP, pull) { }

};

struct PinAF\_OD : public Pin

{

PinAF\_OD(GPIO\_TypeDef \*\_gpio, uint16 \_pin) : Pin(\_gpio, \_pin, GPIO\_MODE\_AF\_OD, GPIO\_PULLUP) { }

};

struct PinAF\_PP : public Pin

{

PinAF\_PP(GPIO\_TypeDef \*\_gpio, uint16 \_pin, uint pull) : Pin(\_gpio, \_pin, GPIO\_MODE\_AF\_PP, pull) { }

};

struct PinAnalog : public Pin

{

PinAnalog(GPIO\_TypeDef \*\_gpio, uint16 \_pin) : Pin(\_gpio, \_pin, GPIO\_MODE\_ANALOG, GPIO\_NOPULL) { }

};

extern Pin pinWP;

extern PinOutputPP pinBEEP;

extern PinAnalog pinADC;

extern PinAnalog pinHumidity;

extern Pin pinSCK\_SPI1; // SPI1

extern Pin pinMOSI\_SPI1; // SPI1

extern Pin pinMISO\_SPI1; // SPI1

extern PinOutputPP pinNSS\_SPI1; // SPI1

extern Pin pinSCL\_SPI2; // SPI2 ST7735

extern Pin pinMOSI\_SPI2; // SPI2 ST7735

extern PinOutputPP pinRESET\_ST; // ST7735

extern PinOutputPP pinDC\_ST; // ST7735

extern PinOutputPP pinCS\_ST; // ST7735

extern Pin pinSCL; // I2C

extern Pin pinSDA; // I2C

extern Pin pinTX\_HC12;

extern Pin pinRX\_HC12;

extern PinOutputPP pinCS\_HC12;

extern Pin pinTX\_NEO\_8M;

extern Pin pinRX\_NEO\_8M;

extern Pin pinKey1; // Keyboard

extern Pin pinKey2; // Keyboard

extern PinInput pinPowerIn; // Если здесь появляется 1, нужно выключать

extern PinOutputPP pinPowerOut; // Подачей сюда 0 производится выключение

extern PinOutputPP pinOUT;

#include "defines.h"

#include "Hardware/HAL/HAL.h"

#include <stm32f3xx\_hal.h>

PinInput pinPowerIn(GPIOA, GPIO\_PIN\_8, GPIO\_PULLDOWN);

PinOutputPP pinPowerOut(GPIOB, GPIO\_PIN\_10, GPIO\_PULLDOWN);

PinOutputPP pinOUT(GPIOB, GPIO\_PIN\_5, GPIO\_PULLUP);

Pin pinWP(GPIOB, GPIO\_PIN\_0, GPIO\_MODE\_OUTPUT\_PP, GPIO\_PULLUP);

PinOutputPP pinBEEP(GPIOB, GPIO\_PIN\_4, GPIO\_PULLUP);

PinAnalog pinADC(GPIOA, GPIO\_PIN\_3);

PinAnalog pinHumidity(GPIOA, GPIO\_PIN\_0);

Pin pinSCL(GPIOB, GPIO\_PIN\_6, GPIO\_MODE\_AF\_OD, GPIO\_NOPULL, GPIO\_AF4\_I2C1);

Pin pinSDA(GPIOB, GPIO\_PIN\_7, GPIO\_MODE\_AF\_OD, GPIO\_NOPULL, GPIO\_AF4\_I2C1);

Pin pinSCK\_SPI1(GPIOA, GPIO\_PIN\_5, GPIO\_MODE\_AF\_PP, GPIO\_NOPULL, GPIO\_AF5\_SPI1);

Pin pinMOSI\_SPI1(GPIOA, GPIO\_PIN\_7, GPIO\_MODE\_AF\_PP, GPIO\_NOPULL, GPIO\_AF5\_SPI1);

Pin pinMISO\_SPI1(GPIOA, GPIO\_PIN\_6, GPIO\_MODE\_AF\_PP, GPIO\_NOPULL, GPIO\_AF5\_SPI1);

PinOutputPP pinNSS\_SPI1(GPIOA, GPIO\_PIN\_4, GPIO\_NOPULL);

Pin pinTX\_HC12(GPIOA, GPIO\_PIN\_9, GPIO\_MODE\_AF\_PP, GPIO\_NOPULL, GPIO\_AF7\_USART1);

Pin pinRX\_HC12(GPIOA, GPIO\_PIN\_10, GPIO\_MODE\_AF\_PP, GPIO\_NOPULL, GPIO\_AF7\_USART1);

PinOutputPP pinCS\_HC12(GPIOB, GPIO\_PIN\_2, GPIO\_NOPULL);

Pin pinTX\_NEO\_8M(GPIOA, GPIO\_PIN\_2, GPIO\_MODE\_AF\_PP, GPIO\_PULLUP, GPIO\_AF7\_USART2);

Pin pinRX\_NEO\_8M(GPIOA, GPIO\_PIN\_3, GPIO\_MODE\_AF\_PP, GPIO\_NOPULL, GPIO\_AF7\_USART2);

Pin pinKey1(GPIOB, GPIO\_PIN\_8, GPIO\_MODE\_IT\_RISING\_FALLING, GPIO\_PULLUP);

Pin pinKey2(GPIOB, GPIO\_PIN\_9, GPIO\_MODE\_IT\_RISING\_FALLING, GPIO\_PULLUP);

PinOutputPP pinRESET\_ST(GPIOB, GPIO\_PIN\_11, GPIO\_NOPULL);

PinOutputPP pinCS\_ST(GPIOB, GPIO\_PIN\_12, GPIO\_PULLDOWN);

Pin pinSCL\_SPI2(GPIOB, GPIO\_PIN\_13, GPIO\_MODE\_AF\_PP, GPIO\_PULLDOWN, GPIO\_AF5\_SPI2);

PinOutputPP pinDC\_ST(GPIOB, GPIO\_PIN\_14, GPIO\_NOPULL);

Pin pinMOSI\_SPI2(GPIOB, GPIO\_PIN\_15, GPIO\_MODE\_AF\_PP, GPIO\_PULLDOWN, GPIO\_AF5\_SPI2);

void Pin::Init()

{

GPIO\_InitTypeDef is =

{

pin,

mode,

pull,

GPIO\_SPEED\_FREQ\_HIGH,

alternate

};

HAL\_GPIO\_Init(gpio, &is);

}

void Pin::ToLow()

{

HAL\_GPIO\_WritePin(gpio, pin, GPIO\_PIN\_RESET);

}

void Pin::ToHi()

{

HAL\_GPIO\_WritePin(gpio, pin, GPIO\_PIN\_SET);

}

void Pin::Set(bool hi)

{

HAL\_GPIO\_WritePin(gpio, pin, hi ? GPIO\_PIN\_SET : GPIO\_PIN\_RESET);

}

bool Pin::IsHi() const

{

return HAL\_GPIO\_ReadPin(gpio, pin) != GPIO\_PIN\_RESET;

}

bool Pin::IsLow() const

{

return HAL\_GPIO\_ReadPin(gpio, pin) == GPIO\_PIN\_RESET;

}