#include "defines.h"

#include "Device.h"

int main(void)

{

Device::Init();

while (true)

{

Device::Update();

}

}

#pragma once

#include "Storage/Measures.h"

namespace HI50

{

bool Init();

bool IsExist();

void Update();

bool GetMeasure(Measure \*);

void CallbackOnReceive(pchar);

}

#include "defines.h"

#include "Modules/HI50/HI50.h"

#include "Hardware/HAL/HAL.h"

#include "Hardware/Timer.h"

#include "Hardware/HAL/HAL.h"

#include "Hardware/InterCom.h"

#include <cstdlib>

namespace HI50

{

static const uint8 TURN\_ON = 0x4f;

// static const uint8 MEAS\_AUTO = 0x44;

static const uint8 MEAS\_HI = 0x4D;

struct State

{

enum E

{

IDLE,

WAIT\_TURN\_ON,

WAIT\_MEASURE

};

};

static State::E state = State::IDLE;

static Measure distance;

static bool is\_exist = false; // true, если модуль измерения дальности подключён

}

bool HI50::IsExist()

{

return is\_exist;

}

bool HI50::Init()

{

HAL\_USART1::SetModeHI50();

state = State::WAIT\_TURN\_ON;

HAL\_USART1::Send(TURN\_ON);

TimeMeterMS meter;

while (meter.ElapsedTime() < 500)

{

HAL\_USART1::Update();

if (state == State::WAIT\_MEASURE)

{

is\_exist = true;

break;

}

}

return is\_exist;

}

void HI50::Update()

{

if (!is\_exist)

{

return;

}

switch (state)

{

case State::IDLE:

HAL\_USART1::Send(TURN\_ON);

state = State::WAIT\_TURN\_ON;

break;

case State::WAIT\_TURN\_ON:

break;

case State::WAIT\_MEASURE:

break;

}

}

void HI50::CallbackOnReceive(pchar message)

{

switch (state)

{

case State::IDLE:

break;

case State::WAIT\_TURN\_ON:

HAL\_USART1::Send(MEAS\_HI);

state = State::WAIT\_MEASURE;

break;

case State::WAIT\_MEASURE:

// Сюда попадает полное сообщение от измерителя

static const int SIZE\_BUFER = 128;

char buffer\_digits[SIZE\_BUFER] = { '\0' };

for (int i = 0; i < SIZE\_BUFER; i++)

{

char buf[2] = { message[i], '\0' };

std::strcat(buffer\_digits, buf);

if (buf[0] == 0)

{

if (buffer\_digits[0] == 'M')

{

for (int index = 3; index < SIZE\_BUFER; index++)

{

if (buffer\_digits[index] == 'm')

{

buffer\_digits[index] = '\0';

std::memmove(buffer\_digits, buffer\_digits + 3, (uint)(index - 3));

float value = (float)std::atof(buffer\_digits);

distance.Set(Measure::Distance, value);

Measure measure;

measure.Set(Measure::Distance, value);

HAL\_USART1::SetModeHC12();

InterCom::Send(measure, TIME\_MS);

HAL\_USART1::SetModeHI50();

HAL\_USART1::Send(MEAS\_HI);

break;

}

}

}

else

{

HAL\_USART1::Send(MEAS\_HI);

break;

}

}

}

break;

}

}

bool HI50::GetMeasure(Measure \*measure)

{

if (distance.correct)

{

\*measure = distance;

return true;

}

return false;

}

#include "defines.h"

#include "Modules/ST7735/ST7735.h"

#include "Hardware/Timer.h"

#include "Display/Font/Font.h"

#include "Utils/Text/String.h"

#include "Display/Display.h"

#include "Hardware/HAL/HAL.h"

#include "Hardware/Timer.h"

#include <stm32f3xx\_hal.h>

#include <cstring>

// !!!!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

// \warning В настройках файла должна быть отключена оптимизация -O0, иначе не работает

// !!!!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

/\*

B11 - RST Reset

B14 - RS D/C

B12 - CS

B13 - SCK SCL

B15 - SDA MOSI

\*/

const uint16 Color::colors[Color::Count] =

{

0xffff, // WHITE

0x0000, // BLACK

MAKE\_COLOR(31, 0, 0), // RED

MAKE\_COLOR(0, 63, 0), // GREEN

MAKE\_COLOR(0, 38, 5), // \_1

MAKE\_COLOR(0, 31, 0), // GREEN\_50

MAKE\_COLOR(0, 0, 31), // BLUE

MAKE\_COLOR(15, 31, 15),

MAKE\_COLOR(7, 15, 7),

MAKE\_COLOR(3, 6, 3)

};

#define SPI2\_DR\_8bit \*(\_\_IO uint8\_t\*)&(SPI2->DR)

namespace ST7735

{

#define SET\_DC pinDC\_ST.ToHi()

#define RESET\_DC pinDC\_ST.ToLow()

#define SET\_CS pinCS\_ST.ToHi()

#define RESET\_CS pinCS\_ST.ToLow()

static SPI\_HandleTypeDef handle;

static void SendCommand(uint8);

static void SendData8(uint8);

static void SetWindow(int startX, int startY, int stopX, int stopY);

}

void ST7735::Init()

{

\_\_HAL\_RCC\_SPI2\_CLK\_ENABLE();

pinSCL\_SPI2.Init();

pinMOSI\_SPI2.Init();

handle.Instance = SPI2;

handle.Init.Mode = SPI\_MODE\_MASTER;

handle.Init.Direction = SPI\_DIRECTION\_2LINES;

handle.Init.DataSize = SPI\_DATASIZE\_8BIT;

handle.Init.CLKPolarity = SPI\_POLARITY\_LOW;

handle.Init.CLKPhase = SPI\_PHASE\_1EDGE;

handle.Init.NSS = SPI\_NSS\_SOFT;

handle.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_2;

handle.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

handle.Init.TIMode = SPI\_TIMODE\_DISABLE;

handle.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

handle.Init.CRCPolynomial = 7;

HAL\_SPI\_Init(&handle);

pinRESET\_ST.ToLow();

pinDC\_ST.ToLow();

pinCS\_ST.ToHi();

pinRESET\_ST.Init();

pinDC\_ST.Init();

pinCS\_ST.Init();

SPI2->CR1 |= SPI\_CR1\_SPE;

pinCS\_ST.ToLow();

pinRESET\_ST.ToHi(); //-V525

HAL\_Delay(20);

pinRESET\_ST.ToLow();

HAL\_Delay(20);

pinRESET\_ST.ToHi();

HAL\_Delay(20);

SendCommand(0x01); // SWRESET Software reset

HAL\_Delay(12);

SendCommand(0x11); // SLPOUT Sleep out

HAL\_Delay(12);

SendCommand(0x3A); // COLMOD Interface pixel format

SendData8(0x05); // 16 bit / pixel

SendCommand(0x36); // MADCTL Memory Data Access Control

SendData8(BINARY\_U8(01100000));

Display::BeginScene(Color::BLACK);

Display::EndScene();

SendCommand(0x29); // DISPON Display on

}

#ifndef GUI

#define WRITE\_NIBBLE(nibble) \

asm("nop"); \

asm("nop"); \

SPI2->DR = Color::colors[value & 0x0f]; \

value >>= 4

void ST7735::WriteBuffer(int x0, int y0, int width, int height)

{

TimeMeterMS meter;

SetWindow(x0, y0, width, height);

SendCommand(0x2C);

SPI2->CR2 &= ~SPI\_CR2\_DS\_Msk;

SPI2->CR2 |= SPI\_DATASIZE\_16BIT;

SET\_DC;

RESET\_CS;

if ((x0 % 8) == 0 && ((width % 8) == 0))

{

for (int y = y0; y < y0 + height; y++)

{

uint\* points = (uint \*)Display::Buffer::GetLine(x0, y);

uint value = \*points;

for (int i = 0; i < width; i += 8)

{

WRITE\_NIBBLE(0);

WRITE\_NIBBLE(1);

WRITE\_NIBBLE(2);

WRITE\_NIBBLE(3);

WRITE\_NIBBLE(4);

WRITE\_NIBBLE(5);

WRITE\_NIBBLE(6);

WRITE\_NIBBLE(7);

value = \*(++points); //-V519

}

}

}

else

{

for (int y = y0; y < y0 + height; y++)

{

uint8\* points = Display::Buffer::GetLine(x0, y);

uint8 value = \*points;

for (int i = 0; i < width; i += 2)

{

WRITE\_NIBBLE(0); //-V760

WRITE\_NIBBLE(1);

value = \*(++points); //-V519

}

}

}

SET\_CS;

SPI2->CR2 &= ~SPI\_CR2\_DS\_Msk;

SPI2->CR2 |= SPI\_DATASIZE\_8BIT;

}

void ST7735::WritePoint(int x, int y, uint16 color)

{

SetWindow(x, y, 2, 2);

SendCommand(0x2C);

SPI2->CR2 &= ~SPI\_CR2\_DS\_Msk;

SPI2->CR2 |= SPI\_DATASIZE\_16BIT;

SET\_DC;

RESET\_CS;

SET\_CS;

SPI2->CR2 &= ~SPI\_CR2\_DS\_Msk;

SPI2->CR2 |= SPI\_DATASIZE\_8BIT;

}

void ST7735::SetWindow(int x, int y, int width, int height)

{

SendCommand(0x2A); // CASET

SendData8(0x00);

SendData8((uint8)x);

SendData8(0x00);

SendData8((uint8)(x + width - 1));

SendCommand(0x2B); // RASET

SendData8(0x00);

SendData8((uint8)y);

SendData8(0x00);

SendData8((uint8)(y + height));

}

#endif

void ST7735::SendData8(uint8 data)

{

\_\_HAL\_SPI\_ENABLE(&handle);

TimeMeterMS meter;

SET\_DC;

RESET\_CS;

while ((SPI2->SR & SPI\_SR\_BSY))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

while (!(SPI2->SR & SPI\_SR\_TXE))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

SPI2\_DR\_8bit = data;

while (!(SPI2->SR & SPI\_SR\_TXE))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

while ((SPI2->SR & SPI\_SR\_BSY))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

SET\_CS;

}

void ST7735::SendCommand(uint8 data)

{

\_\_HAL\_SPI\_ENABLE(&handle);

TimeMeterMS meter;

RESET\_DC;

RESET\_CS;

while ((SPI2->SR & SPI\_SR\_BSY))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

while (!(SPI2->SR & SPI\_SR\_TXE))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

SPI2\_DR\_8bit = data;

while (!(SPI2->SR & SPI\_SR\_TXE))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

while ((SPI2->SR & SPI\_SR\_BSY))

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

SET\_CS;

}