using namespace Primitives;

// Нарисовать подсказку

static void DrawHint(int x, int y);

static void DrawInfo();

static void SetTopRow(int i);

static bool needRedraw = true; // Если true, требуется перерисовка дисплея

static uint timeAutoHint = 0;

static int second = 0;

static int topRow = 0;

static uint timeStart = 0;

static uint timeFrame = 0;

static uint fps = 0; // Столько кадров отрисовано за последнюю секунду

static uint beginSecond = 0; // В это время началась последняя секунда

static uint timePaint = 0; // Время отрисовки за секунду

static int topDraw = 0; // Верхний у отрисовываемой части экрана

static int bottomhDraw = 0; // Нижний у отрисовываемой части экрана

static int width = Display::PHYSICAL\_WIDTH;

static int height = Display::PHYSICAL\_HEIGHT;

bool Display::sendToSCPI = false;

bool Display::drawingScene = false;

struct Coord

{

int x;

int y;

};

static int yString = 110;

static Coord coordMemory = { 15, yString };

static Coord coordTest = { 40, yString };

static Coord coordExtGenerator = { 95, yString };

static Coord coordLaunch = { 130, yString };

static const int MAX\_OBJECTS = 10;

static RedrawingZone \*objects[MAX\_OBJECTS];

static int numObjects = 0;

static DataZone sDataZone;

DataZone \*Display::zoneData = &sDataZone;

static ProgressBarTimeMeasureZone sProgressBarTimeMeasureZone;

ProgressBarTimeMeasureZone \*Display::zoneProgressBarTimeMeasure = &sProgressBarTimeMeasureZone;

static SynchroZone sSynchroZone;

SynchroZone \*zoneSynchro = &sSynchroZone;

static void AddObject(RedrawingZone \*object)

{

objects[numObjects++] = object;

}

void Display::Init()

{

InitHardware();

Font::Set(TypeFont::GOSTAU16BOLD);

Font::SetSpacing(2);

AddObject(zoneData);

AddObject(zoneProgressBarTimeMeasure);

AddObject(zoneSynchro);

}

static void DrawValue(pString string, int x, int y)

{

Rectangle(Font::GetLengthText(string) + 5, 20).FillRounded(x - 3, y - 3, 2, Color::BACK, Color::FILL);

Text(string).Write(x, y, Color::FILL);

}

void DrawRectangle1(int x, int y)

{

static uint time\_start = TIME\_MS;

const int w = Display::PHYSICAL\_WIDTH - x \* 2;

const int h = Display::PHYSICAL\_HEIGHT - y \* 2;

const int x\_start = x;

const int x\_end = x + w;

const int y\_start = y;

const int y\_end = y + h;

int cell = 10;

Rectangle(w, h).Draw(x, y);

Rectangle(w - cell \* 2, h - cell \* 2).Draw(x + cell, y + cell);

VLine vline(cell);

HLine hline(cell);

int delta = (int)((float)(TIME\_MS - time\_start) / 5.0f);

while (delta > cell)

{

delta -= cell;

}

for (int i = x\_start + delta; i < x\_end; i += cell)

{

vline.Draw(i, y);

}

for (int i = y\_start + delta; i < y\_end; i += cell)

{

hline.Draw(x\_end - cell, i);

}

for (int i = x\_end - delta; i >= x\_start; i -= cell)

{

vline.Draw(i, y\_end - cell);

}

for (int i = y\_end - delta; i >= y\_start; i -= cell)

{

hline.Draw(x, i);

}

}

void DrawRectangle2(int x, int y)

{

int cell = 15;

while (x >= 0)

{

int w = Display::PHYSICAL\_WIDTH - x \* 2;

int h = Display::PHYSICAL\_HEIGHT - y \* 2;

Rectangle(w, h).Draw(x, y);

x -= cell;

y -= cell;

}

Rectangle(Display::PHYSICAL\_WIDTH, Display::PHYSICAL\_HEIGHT).Draw(0, 0);

}

void DrawRectangle3(int, int)

{

Rectangle(Display::PHYSICAL\_WIDTH, Display::PHYSICAL\_HEIGHT).Draw(0, 0);

static uint time\_start = TIME\_MS;

int cell = 20;

Rectangle(Display::PHYSICAL\_WIDTH - cell \* 2, Display::PHYSICAL\_HEIGHT - cell \* 2).Draw(cell, cell);

int delta = ((int)(TIME\_MS - time\_start) / 20) % cell;

VLine vline(cell);

HLine hline(cell);

int x\_start = 0;

int x\_end = Display::PHYSICAL\_WIDTH;

int y = 0;

int y\_start = 0;

int y\_end = Display::PHYSICAL\_HEIGHT;

int x = 0;

for (int i = x\_start + delta; i < x\_end; i += cell)

{

vline.Draw(i, y);

}

for (int i = y\_start + delta; i < y\_end; i += cell)

{

hline.Draw(x\_end - cell, i);

}

for (int i = x\_end - delta; i >= x\_start; i -= cell)

{

vline.Draw(i, y\_end - cell);

}

for (int i = y\_end - delta; i >= y\_start; i -= cell)

{

hline.Draw(x, i);

}

}

void DrawRectangle(int, int)

{

Rectangle(Display::PHYSICAL\_WIDTH, Display::PHYSICAL\_HEIGHT).Draw(0, 0);

static uint time\_start = TIME\_MS;

int cell = 18;

Rectangle(Display::PHYSICAL\_WIDTH - cell \* 2, Display::PHYSICAL\_HEIGHT - cell \* 2).Draw(cell, cell);

const int delta = ((int)(TIME\_MS - time\_start) / 40) % cell;

VLine vline(cell);

HLine hline(cell);

const int x\_start = 0;

const int x\_end = Display::PHYSICAL\_WIDTH;

const int y\_start = 0;

const int y\_end = Display::PHYSICAL\_HEIGHT;

for (int i = x\_start + delta; i < x\_end; i += cell)

{

DLine(cell, 1, 1).Draw(i, y\_start);

}

for (int i = y\_end - delta - 8; i >= y\_start; i -= cell)

{

DLine(cell, 1, 1).Draw(x\_end - cell, i);

}

for (int i = x\_start + delta + 1; i < x\_end + cell; i += cell)

{

DLine(cell, -1, -1).Draw(i, y\_end - 1);

}

for (int i = y\_end - delta - 2; i >= y\_start; i -= cell)

{

Line().Draw(x\_start, i, x\_start + cell, i + cell);

}

}

void Display::DrawWelcomeScreen()

{

uint startTime = TIME\_MS;

int counter = 0;

while (TIME\_MS - startTime < 3000)

{

for (int i = 0; i < NUM\_PARTS; i++)

{

SetTopRow(i);

BeginScene();

Color::FILL.SetAsCurrent();

// Text(String(LANG\_IS\_RU ? "OAO МНИПИ" : "OAO MNIPI")).WriteScaled(160, 70, 2);

// Text(String(STR\_NUM\_VERSION)).WriteScaled(140, 152, 2);

Text(String(LANG\_IS\_RU ? "OAO МНИПИ" : "OAO MNIPI")).WriteScaled(110, 60, 3);

Text(String(STR\_NUM\_VERSION)).WriteScaled(140, 160, 2);

EndScene();

}

counter++;

}

}

void Display::DrawKeyboardFailScreen()

{

uint startTime = TIME\_MS;

while (TIME\_MS - startTime < 3000)

{

for (int i = 0; i < NUM\_PARTS; i++)

{

SetTopRow(i);

BeginScene();

Text(LANG\_IS\_RU ? "ОШИБКА КЛАВИАТУРЫ" : "ERROR KEYBOARD").Write(100, 110, Color::FILL);

EndScene();

}

}

}

static void SetTopRow(int i)

{

topRow = i \* (Display::PHYSICAL\_HEIGHT / Display::NUM\_PARTS);

topDraw = topRow;

bottomhDraw = topDraw + Display::PHYSICAL\_HEIGHT / Display::NUM\_PARTS;

}

void Display::Refresh()

{

needRedraw = true;

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

{

objects[i]->Refresh();

}

}

void Display::Update()

{

zoneSynchro->Refresh();

#ifdef GUI

BeginScene();

DrawScreen();

Console::Draw();

EndScene();

#else

static uint currentFramesInSec = 0; // Столько кадров отрисовано за текущую секунду

static uint currentTimePaintInSec = 0; // Столько времени потрачено на отрисовку за текущую секунду

static char prevHint = 0;

zoneProgressBarTimeMeasure->Refresh(); // Прогресс-бар будем перерисовывать каждый кадр

if (Hint::Text()[0] != prevHint) // Если изменилась подсказка - перерисовываем экран

{

prevHint = Hint::Text()[0];

Display::Refresh();

}

if (timeAutoHint != 0U && (TIME\_MS - timeAutoHint) > 10000)

{

timeAutoHint = 0U;

Display::Refresh();

}

if (needRedraw)

{

timeStart = TIME\_MS;

for (int i = 0; i < NUM\_PARTS; i++)

{

DrawPartScreen(i, true);

}

timeFrame = TIME\_MS - timeStart;

currentFramesInSec++;

currentTimePaintInSec += timeFrame;

}

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

{

objects[i]->Update(RedrawingZone::ModeDraw::ToHardware);

}

needRedraw = false;

if (TIME\_MS >= beginSecond + 1000)

{

fps = currentFramesInSec;

currentFramesInSec = 0;

beginSecond = TIME\_MS;

timePaint = currentTimePaintInSec;

currentTimePaintInSec = 0;

}

#endif

}

void Display::DrawPartScreen(int num, bool debugInfo)

{

SetTopRow(num);

if (num == 0)

{

timeStart = TIME\_MS;

}

Display::BeginScene();

DrawScreen();

if (num == 0)

{

Console::Draw();

}

if (num == 0 && debugInfo && glob\_set.showStatistics)

{

Text(String("%d", timeFrame)).Write(440, 0, Color::EMPTY);

Text(String("%d", fps)).Write(440, 15);

Text(String("%d", timePaint)).Write(440, 30);

}

Display::EndScene();

if (num == Display::NUM\_PARTS)

{

timeFrame = TIME\_MS - timeStart;

}

}

void Display::DrawScreen()

{

if (PageIndication::calibrationMode.IsEnabled())

{

Text(LANG\_IS\_RU ? "---Режим Калибровка---" : "---Calibration mode---").Write(140, 10, Color::FILL);

Text(LANG\_IS\_RU ? "Нажмите ЭНК. для сохранения" : "Click ENC. to preserve").Write(125, 40);

Text(LANG\_IS\_RU ? "Нажмите любую кнопку для выхода" : "Press any button to exit").Write(105, 70);

Text(SU::Int2String(FPGA::GovernorData::ValueCalibrator()).c\_str()).Write(210, 100);

}

else

{

Channel::Current()->DrawSettings(10, 15);

Channel::Current()->DrawMode(10, 57);

DrawHint(10, Display::PHYSICAL\_HEIGHT - Item::HEIGHT - 30);

DrawInfo();

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

{

objects[i]->Update(RedrawingZone::ModeDraw::ToBuffer);

}

Menu::Draw();

}

}

static void DrawHint(int x, int y)

{

int dX = 4;

int dY = 6;

static bool autoFlag = false;

if ((TIME\_MS < timeAutoHint + 10000) && timeAutoHint != 0 && autoFlag)

{

Rectangle(360, 30).FillRounded(x, y, 2, Color::BACK, Color::BACK);

Text(FPGA::Auto::Give().c\_str()).Write(x + dX, y + dY, Color::FILL);

FreqMeter::UnloadAuto();

}

else

{

if (FPGA::InAutoMode())

{

if (FPGA::Auto::ObtainedResult())

{

Rectangle(360, 30).FillRounded(x, y, 2, Color::BACK, Color::BACK);

Text(FPGA::Auto::Give().c\_str()).Write(x + dX, y + dY, Color::FILL);

FPGA::DisableAuto();

Keyboard::Unlock();

timeAutoHint = TIME\_MS;

autoFlag = true;

Display::Refresh();

}

else

{

Rectangle(360, 30).FillRounded(x, y, 2, Color::BACK, Color::BACK);

Text(LANG\_IS\_RU ? "Установка уровня синхронизации" : "Setting the sync level").

Write(x + dX, y + dY, Color::FILL);

timeAutoHint = 0;

}

}

else

{

if (Hint::Shown() != 0)

{

Rectangle(360, 30).FillRounded(x, y, 2, Color::BACK, Color::BACK);

Text(Hint::Text()).Write(x + dX, y + dY, Color::FILL);

}

autoFlag = false;

}

}

}

static void DrawInfo()

{

if (PageIndication::memoryMode == MemoryMode::On)

{

if (CURRENT\_CHANNEL\_IS\_A &&

Channel::A->mod.modeFrequency.IsTachometer() &&

(Channel::A->mod.typeMeasure.IsFrequency()))

{

}

else if (CURRENT\_CHANNEL\_IS\_B &&

Channel::B->mod.modeFrequency.IsTachometer() &&

Channel::B->mod.typeMeasure.IsFrequency())

{

}

else if (CURRENT\_CHANNEL\_IS\_A &&

Channel::A->mod.typeMeasure.IsCountPulse() &&

Channel::A->mod.modeCountPulse.Is\_StartStop())

{

}

else if (CURRENT\_CHANNEL\_IS\_B &&

Channel::B->mod.typeMeasure.IsCountPulse() &&

Channel::B->mod.modeCountPulse.Is\_StartStop())

{

}

else

{

DrawValue("M", coordMemory.x, coordMemory.y);

}

}

if (FreqMeter::modeTest.IsEnabled())

{

DrawValue(LANG\_IS\_RU ? "Тест" : "Test", coordTest.x, coordTest.y);

}

if (PageIndication::refGenerator == RefGenerator::External)

{

DrawValue(LANG\_IS\_RU ? "ОГ" : "Ref", coordExtGenerator.x, coordExtGenerator.y);

}

if (PageIndication::launchSource.IsExternal())

{

DrawValue(LANG\_IS\_RU ? "Зап:внешн" : "Launch:ext", coordLaunch.x, coordLaunch.y);

}

else if (PageIndication::launchSource.IsOneTime())

{

DrawValue(LANG\_IS\_RU ? "Зап:однокр" : "Launch:single", coordLaunch.x, coordLaunch.y);

}

if ((CURRENT\_CHANNEL\_IS\_A && Channel::A->mod.typeMeasure.IsCountPulse() && Channel::A->mod.modeCountPulse.Is\_StartStop()) ||

(CURRENT\_CHANNEL\_IS\_B && Channel::B->mod.typeMeasure.IsCountPulse() && Channel::B->mod.modeCountPulse.Is\_StartStop()))

{

Text(ModeStartStop::IsEnabled() ? (LANG\_IS\_RU ? "Старт" : "Start") : (LANG\_IS\_RU ? "Стоп" : "Stop")).Write(430, 60);

}

if (PageIndication::launchSource == LaunchSource::OneTime)

{

if (PageIndication::OnceLaunch() == true)

{

if (second == 0)

{

second = (int)TIME\_MS;

}

Text(LANG\_IS\_RU ? "ПУСК" : "START").Write(430, 110);

if ((second + 1000) < (int)TIME\_MS)

{

second = 0;

PageIndication::OnceLaunchSwitchFalse();

}

}

else

{

Text(" ").Write(430, 80);

}

}

}

int Display::TopRow()

{

return topRow;

}

void Display::SendToSCPI()

{

sendToSCPI = true;

Refresh();

}

static int oldTopRow = 0;

void Display::Sender::Prepare(int w, int h)

{

width = w;

height = h;

oldTopRow = topRow;

topRow = 0;

}

void Display::Sender::Restore()

{

width = PHYSICAL\_WIDTH;

height = PHYSICAL\_HEIGHT;

topRow = oldTopRow;

}

int Display::Width()

{

return width;

}

int Display::Height()

{

return height;

}

#ifdef WIN32

bool Display::InDrawingPart(int, int)

{

return true;

#else

bool Display::InDrawingPart(int y, int height)

{

if (!drawingScene)

{

return true;

}

int yBottom = y + height;

if (y >= topDraw && y <= bottomhDraw)

{

return true;

}

if (y <= topDraw && yBottom >= bottomhDraw)

{

return true;

}

if (yBottom >= topDraw && yBottom <= bottomhDraw)

{

return true;

}

return false;

#endif

}

#include "defines.h"

#include "Display/Console.h"

#include "Display/Display.h"

#include "Display/Primitives.h"

#include "Display/Text.h"

#include "Utils/String.h"

#include <cstring>

#include <cstdio>

#define S\_DBG\_NUM\_STRINGS\_IN\_CONSOLE 1

char Console::buffer[10][100];

bool Console::inProcessDraw = false;

bool Console::inProcessAddingString = false;

int Console::stringInConsole = 0;

int16 Console::prevMaxStrinsInConsole = -1;

void Console::Init()

{

inProcessDraw = false;

inProcessAddingString = false;

stringInConsole = 0;

prevMaxStrinsInConsole = -1;

}

void Console::Draw()

{

/\*

if (prevMaxStrinsInConsole == -1)

{

prevMaxStrinsInConsole = S\_DBG\_NUM\_STRINGS\_IN\_CONSOLE;

}

if (!IsShown() || stringInConsole == 0)

{

return;

}

inProcessDraw = true;

int y = 5;

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

{

String string(buffer[i]);

Primitives::Rectangle(string.Length() + 1, 15).Fill(0, y, Color::BLACK);

Text(string.c\_str()).Write(1, y, Color::\_WHITE);

y += 15;

}

inProcessDraw = false;

\*/

}

void Console::DeleteFirstString()

{

for (int16 i = 1; i < stringInConsole; i++)

{

std::strcpy(buffer[i - 1], buffer[i]);

}

stringInConsole--;

}

void Console::AddString(pchar string)

{

inProcessAddingString = true;

static int count = 0;

if (stringInConsole == S\_DBG\_NUM\_STRINGS\_IN\_CONSOLE)

{

DeleteFirstString();

}

std::sprintf(buffer[stringInConsole], "%d %s", count++, string);

stringInConsole++;

inProcessAddingString = false;

Display::Refresh();

}

bool Console::IsShown()

{

return true;

}

void Console::OnChanged\_MaxStringsInConsole()

{

// \todo Здесь, видимо, не совсем корректное поведение в случае, когда реальных строк меньше, чем максимально допустимое их количество

int delta = prevMaxStrinsInConsole - S\_DBG\_NUM\_STRINGS\_IN\_CONSOLE;

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

{

DeleteFirstString();

}

prevMaxStrinsInConsole = S\_DBG\_NUM\_STRINGS\_IN\_CONSOLE;

}

bool Console::IsBusy()

{

return inProcessDraw || inProcessAddingString;

}

#include "defines.h"

#include "Display/Colors.h"

#include "Display/Display.h"

#include "Display/Primitives.h"

#include "Display/Text.h"

#include "Hardware/VCP.h"

#include "Hardware/HAL/HAL.h"

#include "Utils/Math.h"

#include "Utils/String.h"

#include "Settings.h"

#include <cmath>

#include <cstring>

#include <cstdlib>

using namespace Primitives;

#define CS\_OPEN HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_RESET)

#define CS\_CLOSE HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_4, GPIO\_PIN\_SET)

#define SET\_DC\_COMMAND HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_5, GPIO\_PIN\_RESET)

#define SET\_DC\_DATA HAL\_GPIO\_WritePin(GPIOC, GPIO\_PIN\_5, GPIO\_PIN\_SET)

#define SET\_RES\_LOW HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, GPIO\_PIN\_RESET)

#define SET\_RES\_HI HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, GPIO\_PIN\_SET)

using namespace Primitives;

/// В этом буфере будем рисовать. Ширина равна 480 / 2 потому, что в байте хранятся 2 пикселя с 1 из 16-ти возможных градация каждая.

#define WIDTH\_BUFFER (480)

#define HEIGHT\_BUFFER (272 / Display::NUM\_PARTS)

static uint8 buffer[HEIGHT\_BUFFER][WIDTH\_BUFFER];

static const uint8 \*startBuffer = &buffer[0][0];

static const uint8 \*endBuffer = startBuffer + WIDTH\_BUFFER \* HEIGHT\_BUFFER;

uint8 lineBackground[Display::PHYSICAL\_WIDTH \* 2]; // Эта последовательность байт используется для отрисовки фона

static void SetLShiftFreq(uint freq)

{

HAL\_FSMC::WriteCommand(0xe6); // set the LSHIFT (pixel clock) frequency

HAL\_FSMC::WriteData((uint8)(freq >> 16));

HAL\_FSMC::WriteData((uint8)(freq >> 8));

HAL\_FSMC::WriteData((uint8)(freq));

}

static void SetHorizPeriod(uint16 HT, // Horizontal total period

uint16 HPS, // Non-display period between the start of the horizontal sync signal nad the first display data

uint8 HPW, // Sync pulse width

uint16 LPS, // Horizontal sync pulse start location

uint16 LPSPP // for serial TFT interfact

)

{

HAL\_FSMC::WriteCommand(0xb4);

HAL\_FSMC::WriteData((uint8)(HT >> 8)); // 0x020d 525

HAL\_FSMC::WriteData((uint8)HT);

HAL\_FSMC::WriteData((uint8)(HPS >> 8)); // 0x0014 20

HAL\_FSMC::WriteData((uint8)(HPS));

HAL\_FSMC::WriteData(HPW); // 0x05

HAL\_FSMC::WriteData((uint8)(LPS >> 8));

HAL\_FSMC::WriteData((uint8)(LPS));

HAL\_FSMC::WriteData(LPSPP);

}

static void SetModeLCD()

{

HAL\_FSMC::WriteCommand(0xb0);

HAL\_FSMC::WriteData(0x20);

HAL\_FSMC::WriteData(0x80);

HAL\_FSMC::WriteData(0x01);

HAL\_FSMC::WriteData(0xdf);

HAL\_FSMC::WriteData(0x01);

HAL\_FSMC::WriteData(0x0f);

HAL\_FSMC::WriteData(0x00);

}

void Display::InitHardware()

{

HAL\_FSMC::Reset();

HAL\_FSMC::WriteCommand(0x01); // soft reset

HAL\_TIM::DelayMS(10);

HAL\_FSMC::WriteCommand(0xe0, 0x01); // set pll

HAL\_TIM::DelayMS(10);

HAL\_FSMC::WriteCommand(0xe0, 0x03); // set pll

HAL\_TIM::DelayMS(10);

SetModeLCD();

HAL\_FSMC::WriteCommand(0xf0, 0x02); // set pixel data interface 0x03 for 16bit, 0x00 for 8bit

HAL\_FSMC::WriteCommand(0x3a, 0x50);

// Set the MN of PLL

HAL\_FSMC::WriteCommand(0xe2, 0x1d, 0x02, 0x54); // Set the PLL

HAL\_TIM::DelayMS(100);

SetLShiftFreq(0xfffff / 8);

SetHorizPeriod(525, 25, 5, 0, 0);

HAL\_FSMC::WriteCommand(0xb6); // set vert period

HAL\_FSMC::WriteData(0x01);

HAL\_FSMC::WriteData(0x24);

HAL\_FSMC::WriteData(0x00);

HAL\_FSMC::WriteData(0x0a);

HAL\_FSMC::WriteData(0x05);

HAL\_FSMC::WriteData(0x00);

HAL\_FSMC::WriteData(0x00);

HAL\_FSMC::WriteCommand(0x29); // Включить дисплей

HAL\_FSMC::WriteCommand(0x2a);

HAL\_FSMC::WriteData(0x00);

HAL\_FSMC::WriteData(0x00);

HAL\_FSMC::WriteData(0x01);

HAL\_FSMC::WriteData(0xdf); // 0..479

HAL\_FSMC::WriteCommand(0x2b);

HAL\_FSMC::WriteData(0x00);

HAL\_FSMC::WriteData(0x00);

HAL\_FSMC::WriteData(0x01);

HAL\_FSMC::WriteData(0x0f); //0..271

HAL\_FSMC::WriteCommand(0x2c);

uint8 \*pointer = lineBackground;

for (int i = 0; i < Display::PHYSICAL\_WIDTH \* 2; i += 2)

{

\*pointer++ = Color::BACK.Index();

\*pointer++ = Color::BACK.Index();

}

}