2. Разработка ПО контроллера STM32F303CBT6 для работы по интерфейсу SPI с микросхемой памяти W25Q80DVSNIG.

Работа с микросхемой памяти осуществляется при помощи следующего интерфейса.

namespace W25Q80DV

{

static const uint BEGIN = 0;

static const uint END = 1024 \* 1024;

static const int SIZE\_PAGE = 256;

static const int SIZE\_SECTOR = SIZE\_PAGE \* 16;

// Запись буфера байт в микросхему

void WriteLess256bytes(uint address, const void \*buffer, int size);

// Чтение буфера байта из микросхемы

void ReadLess1024bytes(uint address, void \*buffer, int size);

// Чтение идентификатора микросхемы

void ReadID(uint8 id[2]);

// Стереть сектор, которому принадлежит address

void EraseSectorForAddress(uint address);

// Стереть сектор с номером num\_sector

void EraseSector(int num\_sector);

// Записть целочисленного значения без знака

void WriteUInt(uint address, uint value);

// Запись байта без знака

void WriteUInt8(uint address, uint8 byte);

// Запись проивольного буфера данных

void WriteData(uint, const void \*, int);

// Чтение байта без знака

uint8 ReadUInt8(uint address);

namespace Test

{

// Тест работатоспособности микросхемы памяти

bool Run();

// Возвращает результат выполнения теста

bool Result();

}

}

Реализация функций приведена ниже.

// 2023/08/07 09:57:09 (c) Aleksandr Shevchenko e-mail : Sasha7b9@tut.by

#include "defines.h"

#include "Modules/W25Q80DV/W25Q80DV.h"

#include "Hardware/HAL/HAL.h"

#include "Utils/Buffer.h"

#include "Hardware/Timer.h"

#include <cstdlib>

/\*

\* Block 0

\* Sector 0

\* Адреса 000000h - 0000FFh

\*/

// page

#define PROGRAM\_PAGE 0x02 /\* 34 \*/

#define READ\_DATA 0x03 /\* 26 \*/

#define WRITE\_DISABLE 0x04 /\* 23 \*/

#define READ\_STATUS\_1 0x05 /\* 24 \*/

#define WRITE\_ENABLE 0x06 /\* 22 \*/

#define SECTOR\_ERASE 0x20 /\* 36 \*/

namespace W25Q80DV

{

// Записывает uin8, а затем младшие 3 байта из второго значения

static void Write32bit(uint8, uint);

static bool IsBusy();

static void WaitRelease();

// Возвращает номер сектора, в котором находится данный адрес

static uint NumSectorForAddress(uint address);

// Возвращает номер страницы, в которой находится данный адрес

static uint NumPageForAddress(uint address);

namespace Test

{

static bool result = false;

}

}

void W25Q80DV::WriteLess256bytes(uint address, const void \*\_buffer, int size)

{

if (size > 256)

{

LOG\_ERROR("Too much data");

return;

}

if (NumPageForAddress(address) != NumPageForAddress(address + size - 1))

{

LOG\_ERROR("The data is located on different pages");

return;

}

const uint8 \*buffer = (const uint8 \*)\_buffer;

pinWP.ToHi();

WaitRelease();

HAL\_SPI1::Write(WRITE\_ENABLE); // Write enable

Buffer<uint8, 1024> data;

data[0] = PROGRAM\_PAGE; //-V525

data[1] = (uint8)(address >> 16); // \.

data[2] = (uint8)(address >> 8); // | Адрес

data[3] = (uint8)(address); // /

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

{

data[4 + i] = buffer[i];

}

// команда адрес

HAL\_SPI1::Write(data.Data(), size + 1 + 3); // Page program

HAL\_SPI1::Write(WRITE\_DISABLE); // Write disable

pinWP.ToLow();

}

void W25Q80DV::WriteUInt(uint address, uint value)

{

WriteLess256bytes(address, &value, (int)sizeof(value));

}

void W25Q80DV::WriteUInt8(uint address, uint8 byte)

{

pinWP.ToHi();

WriteLess256bytes(address, &byte, 1);

pinWP.ToLow();

}

void W25Q80DV::WriteData(uint, const void \*, int)

{

LOG\_ERROR("Function not realized");

}

void W25Q80DV::EraseSectorForAddress(uint address)

{

pinWP.ToHi();

address = NumSectorForAddress(address) \* SIZE\_SECTOR; // Рассчитываем адрес первого байта стираемого сектора

WaitRelease();

HAL\_SPI1::Write(WRITE\_ENABLE);

Write32bit(SECTOR\_ERASE, address);

HAL\_SPI1::Write(WRITE\_DISABLE);

pinWP.ToLow();

}

uint W25Q80DV::NumSectorForAddress(uint address)

{

return address / SIZE\_SECTOR;

}

uint W25Q80DV::NumPageForAddress(uint address)

{

return address / SIZE\_PAGE;

}

void W25Q80DV::EraseSector(int num\_sector)

{

EraseSectorForAddress((uint)num\_sector \* SIZE\_SECTOR);

}

void W25Q80DV::ReadLess1024bytes(uint address, void \*\_buffer, int size)

{

uint8 \*buffer = (uint8 \*)\_buffer;

WaitRelease();

Buffer<uint8, 1024> out;

out[0] = READ\_DATA; //-V525

out[1] = (uint8)(address >> 16);

out[2] = (uint8)(address >> 8);

out[3] = (uint8)(address);

Buffer<uint8, 1024> in;

HAL\_SPI1::WriteRead(out.Data(), in.Data(), size + 1 + 3);

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

{

buffer[i] = in[4 + i];

}

}

uint8 W25Q80DV::ReadUInt8(uint address)

{

uint8 result = 0;

ReadLess1024bytes(address, &result, 1);

return result;

}

bool W25Q80DV::Test::Run()

{

EraseSectorForAddress(0);

result = false;

for (uint i = 0; i < 1024; i++)

{

uint8 byte = (uint8)*std*::*rand*();

WriteUInt8(i, byte);

if (byte != ReadUInt8(i))

{

EraseSectorForAddress(0);

result = false;

return result;

}

}

EraseSectorForAddress(0);

result = true;

return result;

}

bool W25Q80DV::Test::Result()

{

return result;

}

void W25Q80DV::Write32bit(uint8 command, uint bits24)

{

uint8 data[4];

data[0] = command;

data[1] = (uint8)(bits24 >> 16);

data[2] = (uint8)(bits24 >> 8);

data[3] = (uint8)(bits24);

HAL\_SPI1::Write(data, 4);

}

void W25Q80DV::WaitRelease()

{

TimeMeterMS meter;

while (IsBusy())

{

if (meter.ElapsedTime() > 100)

{

break;

}

}

}

bool W25Q80DV::IsBusy()

{

static const uint8 out[2] = { READ\_STATUS\_1, 0 };

static uint8 in[2] = { 0, 0 };

HAL\_SPI1::WriteRead(out, in, 2);

return (in[1] & 0x01);

}

void W25Q80DV::ReadID(uint8 id[2])

{

pinWP.ToHi();

uint8 out[6] = { 0x90, 0, 0, 0, 0, 0 };

uint8 in[6] = { 0, 0, 0, 0, 0, 0 };

HAL\_SPI1::WriteRead(out, in, 6);

id[0] = in[4];

id[1] = in[5];

pinWP.ToLow();

}