ПРИМЕНЕНИЕ SPI ФЛЭШ-ПАМЯТИ S25FL256 СОВМЕСТНО С МИКРОСХЕМОЙ 1892ВМ10Я

1. ВВЕДЕНИЕ

Документ описывает порядок работы с SPI флэш-памятью S25FL256, подключенной к порту MFBSP микросхемы 1892ВМ10Я на отладочном модуле NVCom-02TEM-3U.

Проект MCStudio 3M для данного примера доступен на официальном сайте multicore.ru в разделе «Техподдержка → Программное обеспечение → Примеры программирования».

2. ПОРТ MFBSP НА МИКРОСХЕМЕ 1892ВМ10Я

Многофункциональный буферизированный последовательный порт (MFBSP) позволяет обмениваться информацией с внешними устройствами по последовательным интерфейсам (SPI, I2S) в дуплексном режиме, с возможностью независимой настройки приёмника и передатчика, а также может вести обмен параллельно-последовательным кодом с другими микросхемами по линковому интерфейсу (LPORT).

Более подробно порт MFBSP описан в руководстве пользователя на микросхему 1892ВМ10Я.

3. SPI ФЛЭШ-ПАМЯТЬ S25FL256

Флэш-память S25FL256 – энергонезависимое запоминающее устройство, которое соединяется с ведущей системой через последовательный интерфейс SPI. Данное устройство поддерживает вид подключения, при котором для передачи и приема данных используется одна (Single I/O), две(Dual I/O) или четыре линии связи (Quad I/O).

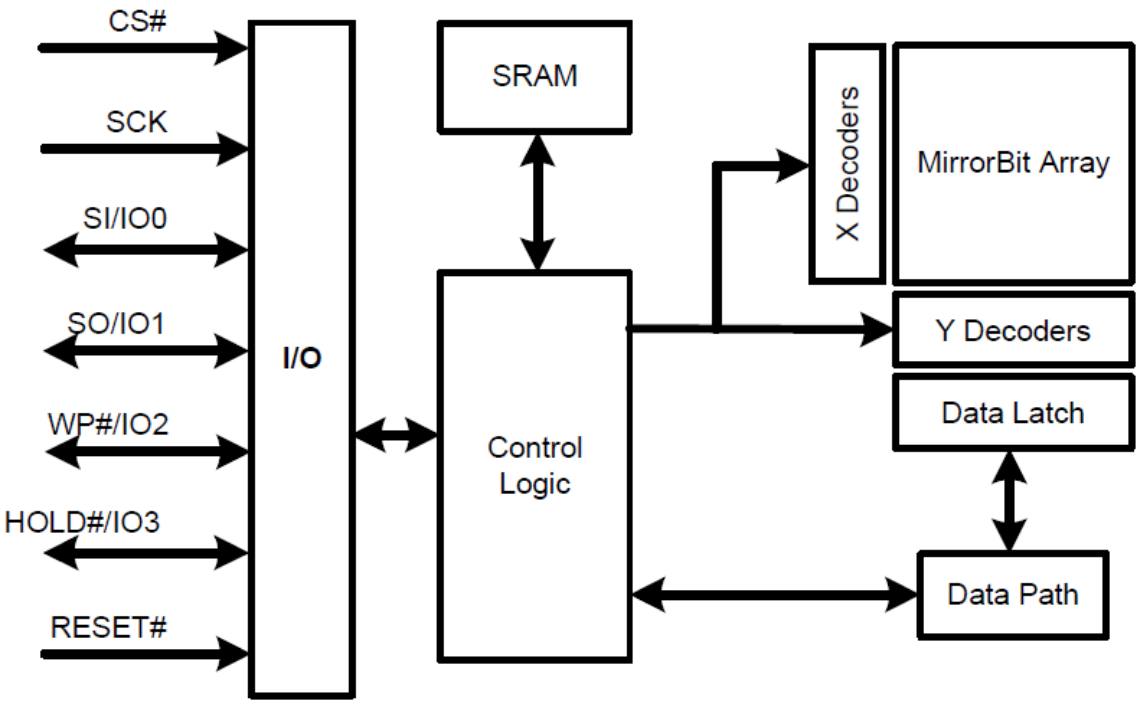

Рисунок 3.1 Структурная схема флэш-памяти S25FL256

Таблица 3.1 Наименования и назначения выводов микросхемы S25FL256

Выводы микросхемы		Назначение	
IO0	SI	Выводы для обмена данными в режиме «Dual I/O mode»	Вход для приема данных от ведущего устройства
IO1	SO		Выход для передачи данных ведущему устройству
IO2		Выводы для обмена данными в режиме «Quad I/O mode»	
IO2			
CS		Выбор ведомого устройства	
SCK		Сигнал синхронизации	

Возможны четыре комбинации фазы (CPHA) и полярности (CPOL) сигнала SCK по отношению к сигналам данных. Режимы работы определяются комбинацией бит CPHA и CPOL:

- CPOL = 0 — сигнал синхронизации начинается с низкого уровня;
- CPOL = 1 — сигнал синхронизации начинается с высокого уровня;
- CPHA = 0 — выборка данных производится по переднему фронту сигнала синхронизации;
- CPHA = 1 — выборка данных производится по заднему фронту сигнала синхронизации

Для обозначения режимов работы интерфейса SPI принято следующее соглашение:

- режим 0 (CPOL = 0, CPHA = 0);
- режим 1 (CPOL = 0, CPHA = 1);
- режим 2 (CPOL = 1, CPHA = 0);
- режим 3 (CPOL = 1, CPHA = 1).

Поддерживаемые режимы работы SPI-флэш S25FL256:

- режим 0 (CPOL = 0, CPHA = 0);
- режим 3 (CPOL = 1, CPHA = 1)

Для управления SPI-флэш используются команды, которые представляют собой последовательность битов, передаваемых через SPI. Полный список команд указан в документации на S25FL256.

В рамках данного примера рассмотрены следующие команды:

Таблица 3.2 Список рассмотренных команд управления микросхемой S25FL256

Условное обозначение	Значение	Описание	Максимальная частота (МГц)
READ_ID	0x90	Чтение ID устройства	133
4READ	0x13	Чтение данных	50
4PP	0x12	Запись страницы	133
4SE	0xDC	Очистка сектора	133
RDSR1	0x05	Чтение статусного регистра SR1	133
WRDI	0x04	Блокировка флэш	133
WREN	0x06	Разблокировка флэш	133

Некоторые из инструкций вслед за кодом команды ожидают значение адреса. Одна и та же инструкция может принимать 24- или 32-разрядное значение адреса.

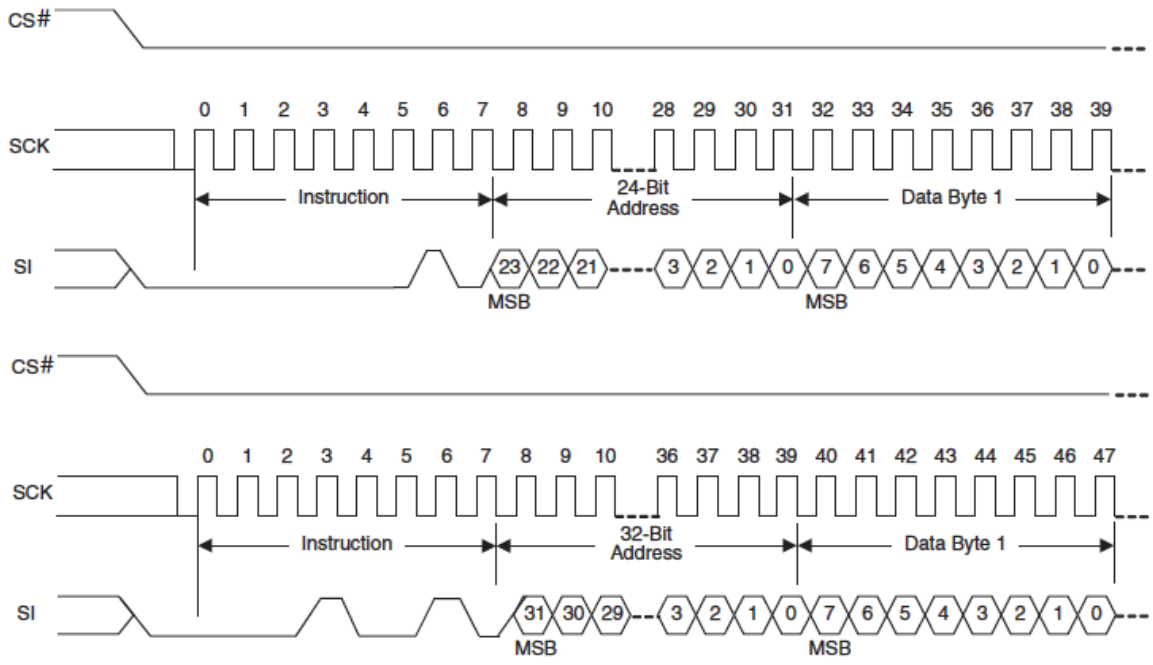

Рисунок 3.2 Пример одинаковых инструкций с разной разрядностью адреса

Код команды таких инструкций будет различаться.

Например:

- READ = 0x03 принимает 24-разрядное значение адреса;
- 4READ = 0x13 принимает 32-разрядное значение адреса.

В данном примере используются команды с 32-разрядным адресом, так как объем адресуемой памяти - 32 Мбайт, а 24-разрядный адрес обеспечивает доступ к объему памяти не более чем 16 Мбайт.

4. ПОДКЛЮЧЕНИЕ SPI ФЛЭШ-ПАМЯТИ S25FL256, К МИКРОСХЕМЕ 1892ВМ10Я В СОСТАВЕ МОДУЛЯ NVCOM-02TEM-3U

Микросхема флэш-памяти S25FL256 в составе модуля NVCom-02TEM-3U подключена к порту MFBSP0.

Для переключения режима загрузки на модуле установлен переключатель SA1. В положении «OFF» загрузка при подаче питания производится из параллельной флэш-памяти. В положении «ON» загрузка при подаче питания производится из SPI-флэш. В этом случае процессор копирует из SPI-флэш 64 32-разрядных слова, начиная с адреса 0x1800_0000. После этого происходит переход на адрес 0x1800_0000.

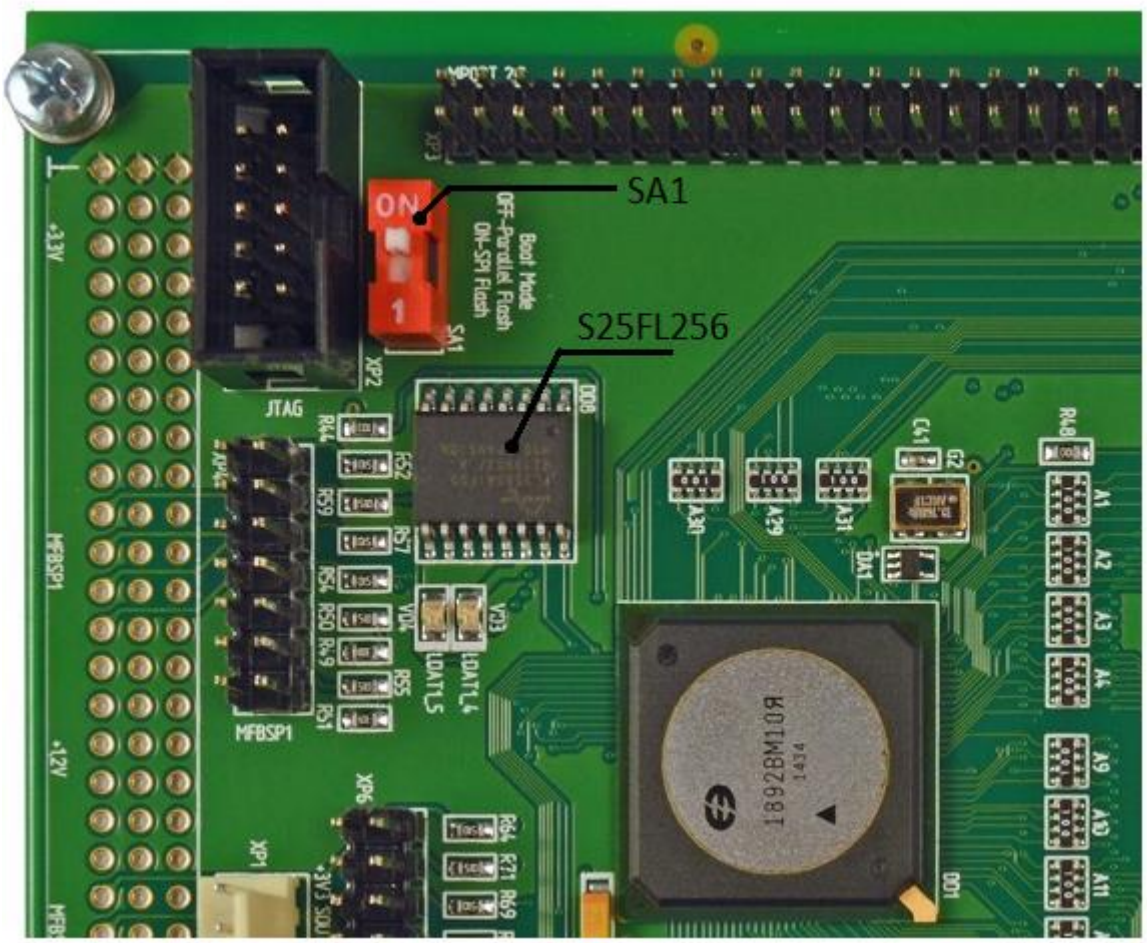

Рисунок 4.1 Лицевая сторона модуля NVCom-02TEM-3U

Если нет необходимости в использовании загрузчика при подаче питания, то для корректного программирования флэш рекомендуется переключатель SA1 перевести в положение «OFF».

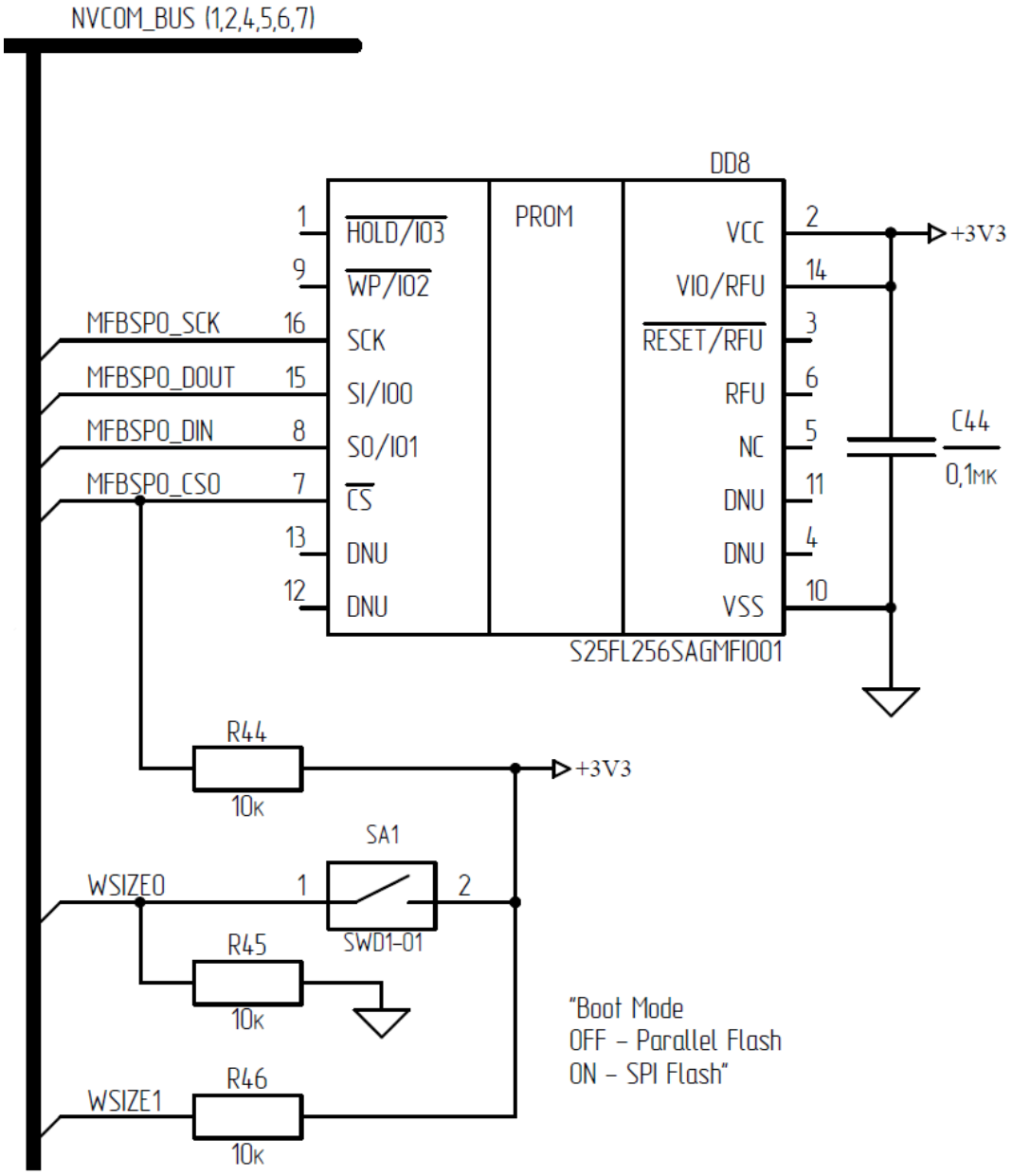

Рисунок 4.2 Принципиальная схема подключения флэш-памяти S25FL256 в составе модуля NVCom-02TEM-3U

Для флэш-памяти S25FL256 доступен вид подключения «Dual I/O mode» и «Quad I/O mode». Порт MFBSP0 для работы в таком режиме подключения не предназначен. Поэтому в составе модуля NVCom-02TEM-3U SPI-флэш работает в стандартном режиме «Single I/O», где SI – вход для данных с передатчика, а SO – вывод данных из флэш.

5. ИНИЦИАЛИЗАЦИЯ MFBSP ДЛЯ РАБОТЫ В РЕЖИМЕ SPI

Для инициализации регистров MFBSP используется следующая структура:

```
typedef struct {
  volatile uint FIFO;
  volatile uint CSR_MFBSP;
  volatile uint DIR;
  volatile uint GPIO_DR;
  volatile uint TCTR;
  volatile uint RCTR;
  volatile uint TSR;
  volatile uint RSR;
  volatile uint TCTR_RATE;
  volatile uint RCTR_RATE;
  uint reserved[54];
} __attribute__((aligned(8))) MFBSP_regs;
```

Совместно с приведенной структурой используется функция, которая возвращает указатель на экземпляр такой структуры для определенного порта MFBSP.

```
//Макрос, возвращающий указатель на структуру регистров DMA_MFBSP для
определенного номера порта
#define DMA_MFBSP_channel(_port_id, _base_addr) ((DMA_MFBSP_regs*
volatile )_base_addr + _port_id)

MFBSP_regs *get_MFBSP_regs (uint _port_id, uint _base_addr){

  if (_port_id < 0 || _port_id > 3) {
      return 0;
  }
  return MFBSP_channel(_port_id, _base_addr);
}
```

Для установки значений регистров введены следующие макросы:

```
//Направление выводов:
typedef enum {
  IN = 0,
  OUT = 1
}DIRECTION;
#define SET_MOSI_DIR(_dir) _dir<<5
#define SET_MISO_DIR(_dir) _dir<<4
#define SET_SS0_DIR(_dir)  _dir<<3
#define SET_TSCK_DIR(_dir) _dir<<1
#define SET_RSCK_DIR(_dir) _dir

//Значение Slave Select0
#define SET_SS0(_bit) _bit<<30

//Работа в режиме:  0 - LPORT; 1 - SPI/I2S
#define SET_SPI_I2S_EN(_bit) _bit<<9

//Режим работы:  0 - I2C; 1 - SPI
#define SET_TMODE(_bit)  _bit<<1
#define SET_RMODE(_bit)  _bit<<1

//Разрешение работы передатчика: 0 – передатчик выключен; 1 –
передатчик включен
#define SET_TEN(_bit)     _bit
#define SET_REN(_bit)     _bit

//Режим работы приемника: 0,0 - асинхронно с передатчиком; 1,1 -
синхронно
#define SET_RCS_CP(_bit)   _bit<<3
#define SET_RCLK_CP(_bit)  _bit<<2

//Порядок следования битов при приеме и передаче: 0 - младшим битом
вперед; 1 - старшим битом вперед
#define SET_TMBF(_bit)    _bit<<19
#define SET_RMBF(_bit)    _bit<<19

//Режим управления сигналом CS: 0 - автоматичекий, 1 - программный
#define SET_SS_DO(_bit)   _bit<<3

//Фронт сигнала синхронизации для передатчика/приемника (CPHA): 0 -
передний фронт; 1 - задний
#define SET_TDEL(_bit)    _bit<<10
#define SET_RDEL(_bit)    _bit<<10

//Начало сигнала синхронизации для передатчика/приемника (CPOL): 0 -
низкий уровень; 1 - высокий
#define SET_TNEG(_bit)    _bit<<11
#define SET_RNEG(_bit)    _bit<<11
```

Функция *SPI_Init (uint _port_id, uint _cpu_freq, uint _spi_freq)* использует вышеприведенные макросы для инициализации MFBSP.

```
//Конфигурация порта MFBSP для работы в режиме SPI
void SPI_Init(uint _port_id, uint _cpu_freq, uint _spi_freq){

  MFBSP_regs* MFBSP  = get_MFBSP_regs(_port_id, MFBSP_BASE_ADDR);

  //Установка MFBSP в режим SPI
  MFBSP->CSR_MFBSP = SET_SPI_I2S_EN(1);

  //Конфигурация направлений выводов
  MFBSP->DIR = (
    SET_MOSI_DIR(OUT) |
    SET_MISO_DIR(IN)  |
    SET_SS0_DIR(OUT)  |
    SET_TSCK_DIR(OUT) |
    SET_RSCK_DIR(OUT)
    );

  //Конфигурация передатчика
  MFBSP->TCTR = (
    SET_SS0(1)    |
    SET_TMBF(1)   |
    SET_TNEG(1)   |
    SET_TDEL(1)   |
    SET_SS_DO(1)  |
    SET_TMODE(1)  |
    SET_TEN(1)
    );

  //Конфигурация приемника
  MFBSP->RCTR = (
    SET_RMBF(1)     |
    SET_RNEG(1)     |
    SET_RDEL(1)     |
    SET_RCS_CP(1)   |
    SET_RCLK_CP(1)  |
    SET_RMODE(1)    |
    SET_REN(1)
    );

  //Делитель частоты передатчика
  MFBSP->TCTR_RATE = (uint)((_cpu_freq/(2*_spi_freq))-1);

  // Slave Select0  = 1
  SPI_set_SS0(_port_id, 1);
}
```

5.1 Таблицы с описанием и значениями регистров.

Таблица 5.1 Конфигурация регистра CSR_MFBSP0

Номер разряда	Условное обозначение	Значение	Описание
CSR_MFBSP0[9]	SPI_I2S_EN	1	Режим SPI/I2S
CSR_MFBSP0[1]	LTRAN	0	LSTAT показывает состояние буфера приёма

Таблица 5.2 Конфигурация регистра DIR_MFBSP0

Номер разряда	Условное обозначение	Значение	Описание
DIR_MFBSP0[5]	TD_DIR	1	MOSI – является выходом для передачи последовательных данных
DIR_MFBSP0[4]	RD_DIR	0	Последовательные данные принимаются со входа MISO
DIR_MFBSP0[3]	TCS_DIR	0	Управляющий сигнал для передатчика снимается с вывода SS[0]
DIR_MFBSP0[1]	TCLK_DIR	1	Тактовый сигнал TSCK формируется передатчиком
DIR_MFBSP0[0]	RCLK_DIR	1	Тактовый сигнал RSCK формируется приёмником

Таблица 5.3 Конфигурация регистра TCTR0

Номер разряда	Условное обозначение	Значение	Описание
TCTR0[24:20]	TWORDLEN	Устанавливается перед передачей	Число бит в передаваемом слове равно TWORDLEN + 1
TCTR0[19]	TMBF	1	Порядок следования бит при передаче - старшим битом вперед
TCTR0[17:12]	TWORDCNT	При программном управлении SS – значение не важно	Число передаваемых слов равно TWORDCNT + 1
TCTR0[11]	TDEL	1	Выборка данных производится по заднему фронту сигнала синхронизации (Эквивалентно CPHA = 1 в спецификации Motorola)
TCTR0[10]	TNEG	1	Сигнал синхронизации начинается с высокого уровня (Эквивалентно CPOL = 1 в спецификации Motorola)
TCTR0[3]	SS_DO	1	Значения бит SS напрямую передаются на внешние выводы
TCTR0[1]	TMODE	1	Режим SPI
TCTR0[0]	TEN	1	Разрешение работы

Таблица 5.4 Конфигурация регистра RCTR0

Номер разряда	Условное обозначение	Значение	Описание
RCTR0[24:20]	RWORDLEN	Устанавливается перед приемом	Число бит в принимаемом слове равно RWORDLEN + 1
RCTR0[19]	RMBF	1	Порядок следования бит при приеме - старшим битом вперед
RCTR0[17:12]	RWORDCNT	При программном управлении SS – значение не важно	Число принимаемых слов равно RWORDCNT + 1
RCTR0[11]	RDEL	1	Выборка данных производится по заднему фронту сигнала синхронизации (Эквивалентно CPHA = 1 в спецификации Motorola)
RCTR0[10]	RNEG	1	Сигнал синхронизации начинается с высокого уровня (Эквивалентно CPOL = 1 в спецификации Motorola)
RCTR0[03]	RCS_CP	1	Приёмник осуществляет приём данных синхронно с передатчиком
RCTR0[2]	RCLK_CP	1	RSCK приёмника дублирует TSCK передатчика
RCTR0[1]	RMODE	1	Режим SPI
RCTR0[0]	REN	1	Разрешение работы

Таблица 5.5 Конфигурация регистра TCTR_RATE0

Номер разряда	Условное обозначение	Значение	Описание
TCTR_RATE0[15:12]	TSS_RATE	0	Если сигнал SS формируется передатчиком, то задает время удержания сигнала SS в высоком уровне между передачами слов.
TCTR_RATE0[9:0]	TCLK_RATE		В случае, если частота формируется самим передатчиком, определяет частоту передатчика

6. ПОРЯДОК ОБМЕНА ДАННЫМИ С SPI-ФЛЭШ

Для передачи слов определенной разрядности необходимо:

В регистрах TCTR и RCTR установить разрядность передаваемого и принимаемого слова. Значения должны быть равны, кратны 8 и не должны быть больше 64.

```
void SPI_set_wordlen(uint _port_id, uint _len){

  MFBSP_regs* MFBSP  = get_MFBSP_regs(_port_id, MFBSP_BASE_ADDR);
  MFBSP->TCTR =  (MFBSP->TCTR&(~(0x1F<<20)))|((_len)<<20);
  MFBSP->RCTR =  (MFBSP->RCTR&(~(0x1F<<20)))|((_len)<<20);
}
```

Установить SS0 в «0».

```
void SPI_set_SS0(uint _port_id, uint _bit){

  MFBSP_regs* MFBSP  = get_MFBSP_regs(_port_id, MFBSP_BASE_ADDR);
  MFBSP->TCTR = ((MFBSP->TCTR)&(~(0x3<<30)))|(_bit<<30);
}
```

Приемник сконфигурирован в режим синхронной работы с передатчиком. Чтобы начать обмен, необходимо записать данные в буфер передачи (TX_MFBSP). К моменту окончания передачи слова заданной разрядности, в буфере приемника с SPI-флэш будет получено слово такой же разрядности. Проверяя регистры состояния приема и передачи, необходимо дождаться завершения передачи слова. А затем считать полученное значение из FIFO приемника (RX_MFBSP).

```
  MFBSP_regs* MFBSP  = get_MFBSP_regs(_port_id, MFBSP_BASE_ADDR);

  uint i = 0;
  char buf;
  for (i = 0; i < _data_len; i++){
    //Если указатель существует
    if((uint)_td)
      MFBSP->FIFO = *_td++;  //Передача слова в FIFO
    else
      MFBSP->FIFO = 0;       //Передача 0 в FIFO

    //Ожидание окончания передачи и приема
    while(MFBSP->TSR&(1<<7) | MFBSP->RSR&(1<<7) );

    //Если указатель существует
    if((uint)_rd)
      *_rd++ = MFBSP->FIFO;  //Считывание принятых данных из FIFO
    else
      buf = MFBSP->FIFO;     //Считывание принятых данных из FIFO во
временный буфер

  }
```

Установить SS0 в «1», если необходимо завершить обмен.

```
  //Если _end = 1 - закончить передачу(иначе продолжить)
  if(_end) SPI_set_SS0(_port_id, 1); // Slave Select - SET_SS0 = 1

}
```

6.1 Обмен данными с использованием DMA

Для обмена 32-разрядными словами можно использовать DMA. Порядок действий аналогичен приведенному выше. Отличие в способе обмена информацией с FIFO приемника и передатчика. Далее рассматривается пример настройки DMA приемника и передатчика.

Для инициализации регистров DMA MFBSP используется следующая структура:

```
typedef struct {
  volatile  uint CSR_DMA;
  volatile  uint CP;
  volatile  uint IR;
  volatile  uint RUN;
} __attribute__((aligned(8))) DMA_MFBSP_regs;
```

Совместно с приведенной структурой используется функция, которая возвращает указатель на экземпляр такой структуры для определенного порта DMA MFBSP.

```
DMA_MFBSP_regs *get_DMA_MFBSP_regs (uint _port_id, uint _base_addr){

  if (_port_id < 0 || _port_id > 3) {
      return 0;
  }
  return DMA_MFBSP_channel(_port_id, _base_addr);
}
```

Для установки значений регистров введены следующие макросы:

```
//Количество передаваемых слов: WCX + 1;
#define SET_WCX(_val) _val<<16
//Число слов данных (пачка), которое передается за одно предоставление
прямого доступа
#define SET_WN(_val)  _val<<2
//Состояние работы канала DMA
#define SET_RUN(_bit) _bit
```

Конфигурация регистров DMA_MFBSP_TX/ DMA_MFBSP_RX:

```
  DMA_MFBSP_regs* DMA_MFBSP_TX =  get_DMA_MFBSP_regs(_port_id,
BASE_DMA_MFBSP_TX);
  DMA_MFBSP_TX->CSR_DMA  = (
    SET_WCX(_data_len/2 - 1) |
    SET_WN(7)                |
    SET_RUN(0)
    );
  //Запись адреса передаваемых данных
  DMA_MFBSP_TX->IR = sys_kernel_va_to_pa((uint)_td);
```

```
  DMA_MFBSP_regs* DMA_MFBSP_RX =  get_DMA_MFBSP_regs(_port_id,
BASE_DMA_MFBSP_RX);
  DMA_MFBSP_RX->CSR_DMA  = (
    SET_WCX(_data_len/2 - 1) |
    SET_WN(7)                |
    SET_RUN(0)
    );
  //Запись адреса передаваемых данных
  DMA_MFBSP_RX->IR = sys_kernel_va_to_pa((uint)_rd);
```

Таблица 6.1 Конфигурация регистра CSR DMA

Номер разряда	Условное обозначение	Значение	Описание
31:16	WCX	_data_len/2 - 1	Количество передаваемых слов: WCX + 1;
5:2	WN	7	Число слов данных (пачка)
0	RUN	0	Состояние работы канала DMA

Таблица 6.2 Конфигурация регистра IR DMA

Номер разряда	Условное обозначение	Значение	Описание
31:0	IR	Физический адрес передаваемых/принимаемых данных	Регистр индекса (адрес памяти)

Важно: перед началом передачи необходимо также установить разрядность передаваемых/принимаемых слов равной 32.

```
//Разрядность передаваемого/принимаемого слова
SPI_set_wordlen(_port_id, 32 - 1);
```

Далее необходимо запустить DMA передатчика:

```
DMA_MFBSP_TX->CSR_DMA  |= 1;
```

Ждать, пока в буфере приема появятся данные.

```
while(MFBSP->RSR&1);
```

Запустить DMA приемника.

```
DMA_MFBSP_RX->CSR_DMA  |= 1;
```

Ждать завершения передачи и приема данных

```
while(MFBSP->RSR&(1<<7) | MFBSP->TSR&(1<<7) );
```

7. ЧТЕНИЕ ID

Чтобы считать данные с флэш-памяти S25FL256 необходимо подать команду READ_ID(0x90) и 24-разрядное значение параметра ADD, которое должно быть равно 0 или 1.

Если ADD = 0, то сначала устройство на линию данных выдаст «Manufacturer ID», а затем «Device ID». Если ADD = 1, то сначала устройство выдаст «Device ID», а затем «Manufacturer ID»

Для S25FL256:

- Manufacturer ID = 0x01;
- Device ID = 0x18.

Пример функции **S25FL256_read_id():**

```
//Чтение ID SPI-флэш памяти S25FL256
ushort S25FL256_read_id(uint _add){

    S25FL256_cmd(    //Передача команды и модификатора ADD
        READ_ID, //Передача команды READ_ID (0x90)
        0);      //Продолжить операцию (SS0 = "0")

    S25FL256_addr(
        _add,    //Значение ADD
        3,    //Разрядность - 3 байта
        0);   //Продолжить операцию (SS0 = "0")

    char id[2];
    SPI_Transfer(    //Прием данных
        MFBSP0,//Id порта MFBSP
        0,  //Передаваемых данных нет
        id, //Указатель на массив id, для приема данных
        2,  //Количество слов - 2
        8,  //Разрядность слов - 8
        0,  //Без использования DMA
        1); //Завершить операцию (SS0 = "1")
    return id[0]<<8 | id[1];
}
```

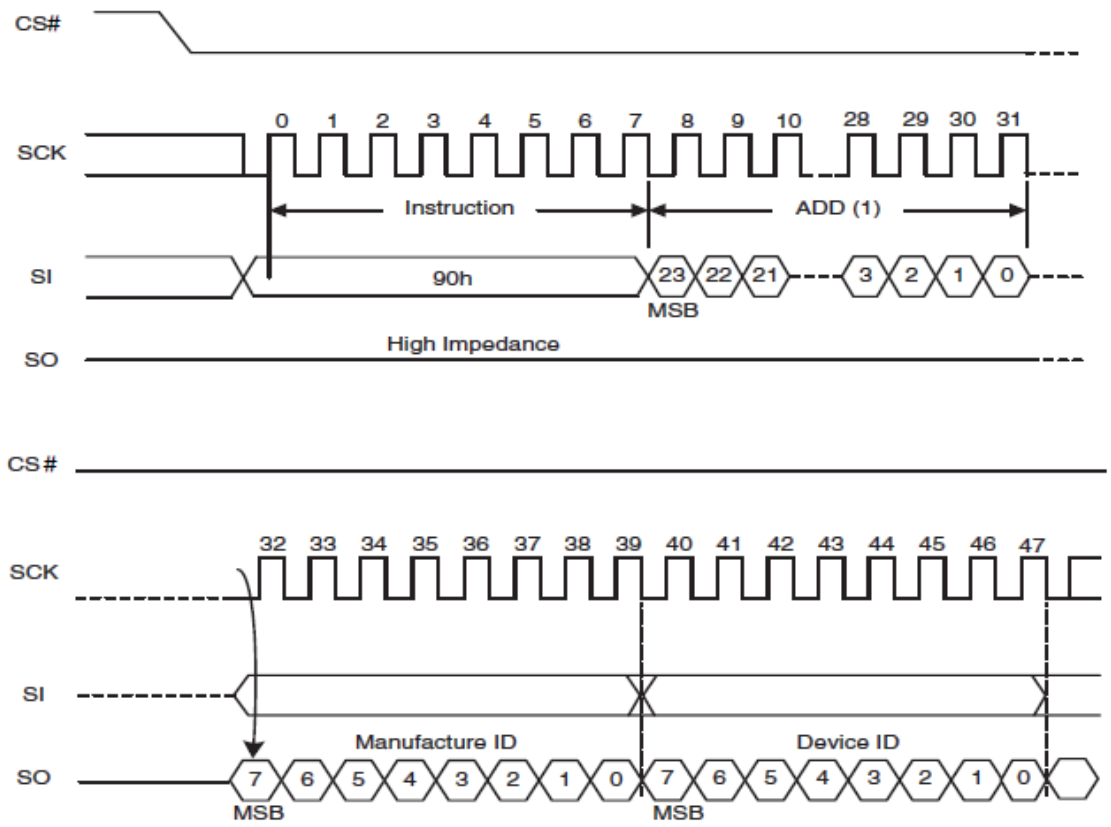

Рисунок 7.1 Временная диаграмма для команды READ_ID

В функции чтения ID для передачи значения модификатора ADD используется функция побайтовой передачи адреса. Функция принимает 32-разрядное целочисленное значение _addr и число байт, которое будет передано. Начиная с младшего байта, указанное количество байт из значения _addr передается в SPI-флэш.

```
//Побайтовая передача адреса
void S25FL256_addr(uint _addr, const uint _addr_len, uint _end){

    //Объявление байтового массива
    char addr[_addr_len];
    //Побайтовое выделение адреса из 32-разрядного слова
    //с учетом _addr_len
    int i = 0;
    for (i = 0; i <_addr_len; i++){
        addr[i] = (char)(_addr>>((_addr_len-1-i)*8));
    }

    //SPI обмен
    SPI_Transfer(
        MFBSP0,     //Id порта MFBSP
        addr,       //Ссылка на массив с адресом
        0,          //Принимаемых данных нет
        _addr_len, //Количесво слов - _addr_len
        8,          //Разрядность слова - 8
        0,          //Без использования DMA
        _end);      //Признак завершения операции
                    //(Необходимо ли установить SS0 = 1 после
передачи?)

}
```

8. ЧТЕНИЕ ДАННЫХ

Чтобы считать данные с флэш-памяти S25FL256 необходимо подать команду 4READ(0x13), а затем 32-разрядное значение адреса ячейки памяти. Адрес может указываться с точностью до байта.

Данные из памяти микросхемы S25FL256 будут последовательно поступать на вывод SO, пока сигнал выбора ведомого устройства(SS0) остается в «0». После передачи одного байта, адрес внутри флэш инкрементируется, и на выход передается следующий байт из памяти. Когда значение адреса достигнет максимума – адрес обнуляется и передача продолжается уже с адреса 0x0000_0000. Таким образом, допустимо начать чтение с любого байта и прочитать всю память флэш.

Пример функции **S25FL256_read():**

```
//Чтение данных
void S25FL256_read(uint _addr, void* _rd, uint _data_len, uint
 word len, uint  dma en){
        S25FL256_cmd(//Передача команды и адреса
        READ,      //Передача команды READ (0x13)
        0);        //Продолжить операцию (SS0 = "0")
    S25FL256_addr(
        _addr,     //Значение адреса
        4,         //Разрядность адреса - 4 байта
        0);        //Продолжить операцию (SS0 = "0")
    SPI_Transfer( //Прием данных
        MFBSP0,    //Id порта MFBSP
        0,         //Передаваемых данных нет
        _rd,       //Указатель на принимаемые данные
        _data_len,//Количество слов -  data len
        _word_len,//Разрядность слов - _word_len
        _dma_en,   //Разрешение работы DMA
        1);        //Завершить операцию (SS0 = "1")
}
```

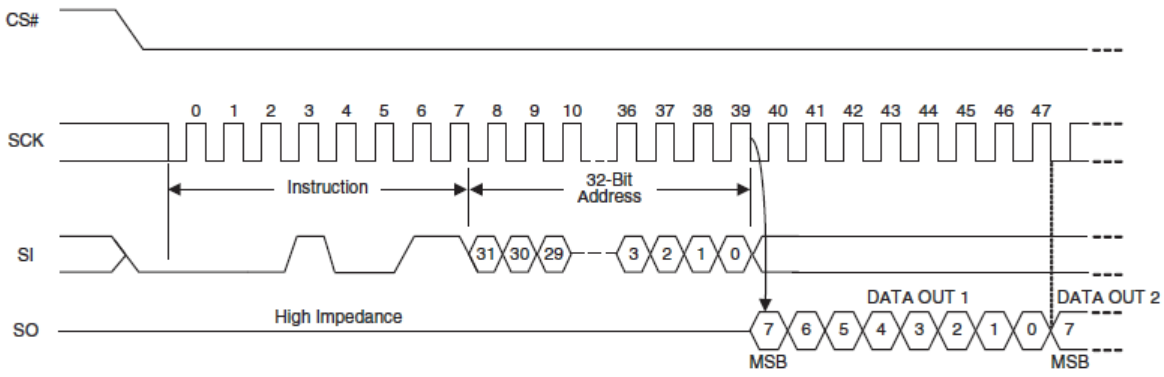

Рисунок 8.1 Временная диаграмма для команды 4READ

9. СТАТУСНЫЙ РЕГИСТР SR1

Таблица 9.1 Назначение разрядов статусного регистра SR1

Номер разряда	Условное обозначение	Назначение
SR1[7]	SRWD	Запрет записи в статусный регистр
SR1[6]	P_ERR	Ошибка при записи
SR1[5]	E_ERR	Ошибка при очистке
SR1[4]	BP2	Защита блока памяти
SR1[3]	BP1	
SR1[2]	BP0	
SR1[1]	WEL	Разрешение команд записи/очистки
SR1[0]	WIP	Статус операции записи/очистки

Пример чтения статусного регистра:

```
    char sr1;
    S25FL256_cmd(
        RDSR1,     //Передача команды RDSR1 (0x05)
        0);        //Продолжить операцию (SS0 = "0")
    do{
        SPI_Transfer(
            MFBSP0,//Id порта MFBSP
            0,     //Передаваемых данных нет
            &sr1, //Ссылка на 8-разрядную переменную sr1
            1,     //Количество слов - 1
            8,     //Разрядность слова - 8
            0,     //Без использования DMA
            0);    //Продолжить операцию (SS0 = "0")
    }while(sr1&1);//Выполнять пока SR1[0] != 0
```

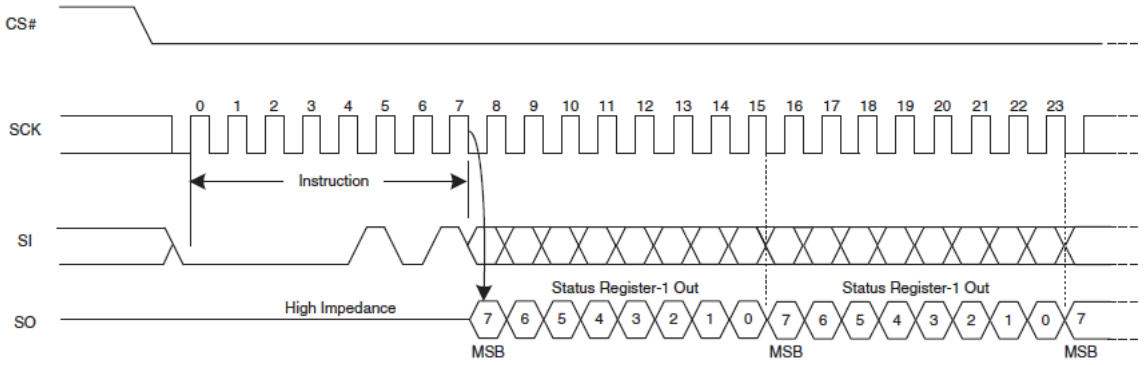

Рисунок 9.1 Временная диаграмма для команды RDSR1

Пока сигнал выбора ведомого устройства(SS0) остается в «0» - флэш будет передавать на линию данных 8-разрядное значение статусного регистра. По значению статусного регистра осуществляется проверка окончания записи/очистки.

10. РАЗБЛОКИРОВКА УСТРОЙСТВА

Перед тем, как передать во флэш команды записи/очистки – необходимо разблокировать устройство. Делается это с помощью записи «1» в поле WEL статусного регистра SR1. Для этого существует специальная команда – WREN(0x06).

```
    //Разблокировка
    S25FL256_cmd(
        WREN,       //Передача команды WREN (0x06)
        1);         //Завершить операцию (SS0 = "1")
```

Поле WEL регистра RDSR1 установится в «1» и команды записи/очистки будут корректно приняты устройством.

Для блокировки флэш существует команда WRDI(0x04).

```
    //Блокировка
    S25FL256_cmd(
        WRDI,       //Передача команды WRDI (0x04)
        1);         //Завершить операцию (SS0 = "1")
```

Поле WEL регистра RDSR1 установится в «0» и команды записи/очистки будут игнорироваться.

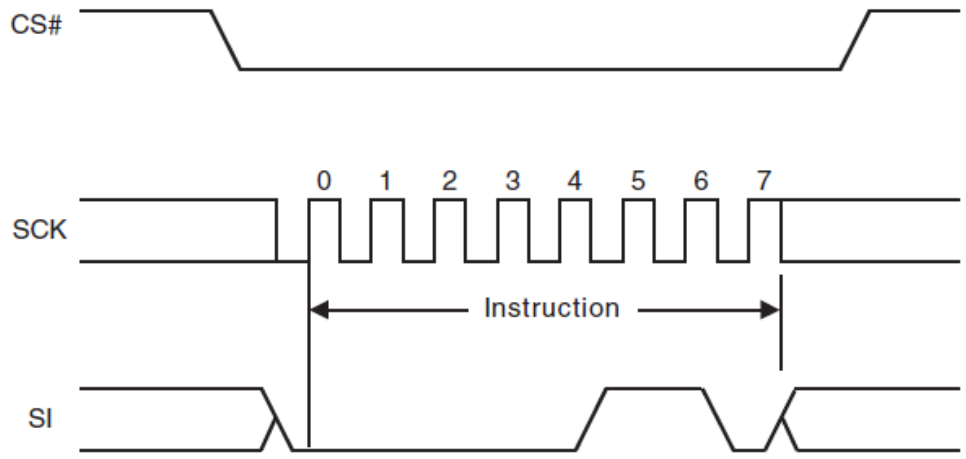

Рисунок 10.1 Временная диаграмма команд WREN, WRDI

11. ЗАПИСЬ ДАННЫХ

Область памяти флэш условно делится на страницы. Для S25FL256 размер страницы – 256 байт.

Операция записи может устанавливать «1» в «0». Для преобразования «0» в «1» необходимо использовать операцию очистки памяти флэш.

Разбор функции **S25FL256_page_program ():**

Снятие блокировки.

```
    //Разблокировка
    S25FL256_cmd(
        WREN,       //Передача команды WREN (0x06)
        1);         //Завершить операцию (SS0 = "1")
```

Далее, необходимо передать на флэш команду записи (4PP), адрес и данные. Первый байт - команда (4PP) 0x12. Следующие 4 байта – 32-разрядный адрес. Последующее содержимое буфера должно содержать в себе от 0 до 256 байт данных, которые будут записаны в память флэш. Запись должна производиться в пределах одной станицы.

```
    //Передача команды и адреса
    S25FL256_cmd(
        PP,         //Передача команды PP (0x12)
        0);         //Продолжить операцию (SS0 = "0")
    S25FL256_addr(
         _addr,     //Значение адреса
        4,          //Разрядность адреса - 4 байта
        0);         //Продолжить операцию (SS0 = "0")

    //Передача данных
    SPI_Transfer(
        _td,        //Указатель на передаваемые данные
        0,          //Принимаемых данных нет
         _data_len,//Количество слов - _data_len
        _word_len,//Разрядность слов - _word_len
        1);         //Завершить операцию (SS0 = "1")
```

Опрос статусного регистра Ожидание завершения операции. Согласно главе 9, младший бит регистра SR1 является признаком завершения операции записи.

```
    //Ожидание завершения операции
    char sr1;
    S25FL256_cmd(
        RDSR1,    //Передача команды RDSR1 (0x05)
        0);       //Продолжить операцию (SS0 = "0")
    do{
        SPI_Transfer(
            0,    //Передаваемых данных нет
            &sr1, //Ссылка на 8-разрядную переменную sr1
            1,    //Количество слов - 1
            8,    //Разрядность слова - 8
            0);   //Продолжить операцию (SS0 = "0")
    }while(sr1&1);//Выполнять пока SR1[0] != 0
    SET_SS0(1);   //Завершить операцию (SS0 = "1")
```

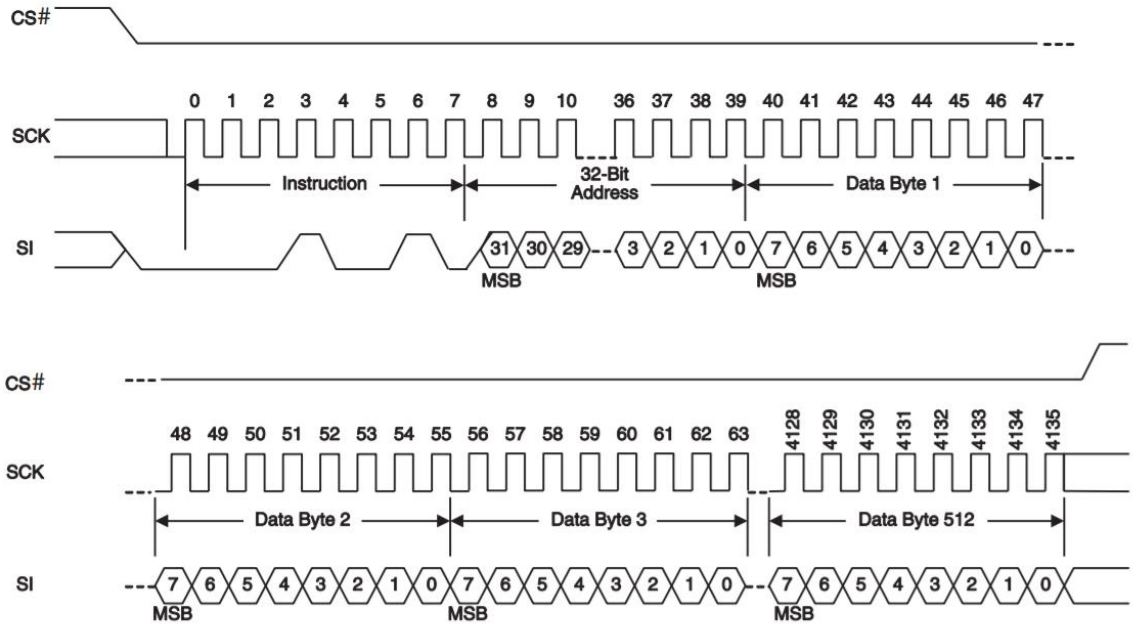

Рисунок 11.1 Временная диаграмма команды 4PP

12. ОЧИСТКА СЕКТОРА

Память флэш делится на сектора. Операция очистки устанавливает во все ячейки памяти сектора значение 0xFF. Вслед за кодом инструкции (4SE) 0xDC должен быть передан 32-разрядный адрес, указывающий на начало сектора.

Таблица 12.1 Карта памяти S25FL256

Размер сектора	Количество секторов	Условное обозначение	Диапазон адресов
4 КБайт	32	SA00	0x0000_0000 - 0x0000_0FFF
		:	:
		SA31	0x0001_F000 - 0x0001_FFFF
64 КБайт	510	SA32	0x0002_0000 - 0x0002_FFFF
		:	:
		SA541	0x01FF_0000 - 0x01FF_FFFF

Пример функции **S25FL256_erase ():**

```
void S25FL256_erase(uint _addr){

    //Разблокировка
    S25FL256_cmd(
        WREN,      //Передача команды WREN (0x06)
        1);        //Завершить операцию (SS0 = "1")

    //Передача команды и адреса
    S25FL256_cmd(
        SE,        //Передача команды SE (0xDC)
        0);        //Продолжить операцию (SS0 = "0")
    S25FL256_addr(
        _addr,     //Значение адреса
        4,         //Разрядность адреса - 4 байта
        1);        //Завершить операцию (SS0 = "1")

    //Ожидание завершения операции
    char sr1;
    S25FL256_cmd(
        RDSR1,     //Передача команды RDSR1 (0x05)
        0);        //Продолжить операцию (SS0 = "0")
    do{
        SPI_Transfer(
            0,     //Передаваемых данных нет
            &sr1, //Ссылка на 8-разрядную переменную sr1
            1,     //Количество слов - 1
            8,     //Разрядность слова - 8
            0);    //Продолжить операцию (SS0 = "0")
    }while(sr1&1);//Выполнять пока SR1[0] != 0
    SET_SS0(1);    //Завершить операцию (SS0 = "1")

    //Блокировка
    S25FL256_cmd(
        WRDI,      //Передача команды WRDI (0x04)
        1);        //Завершить операцию (SS0 = "1")
}
```

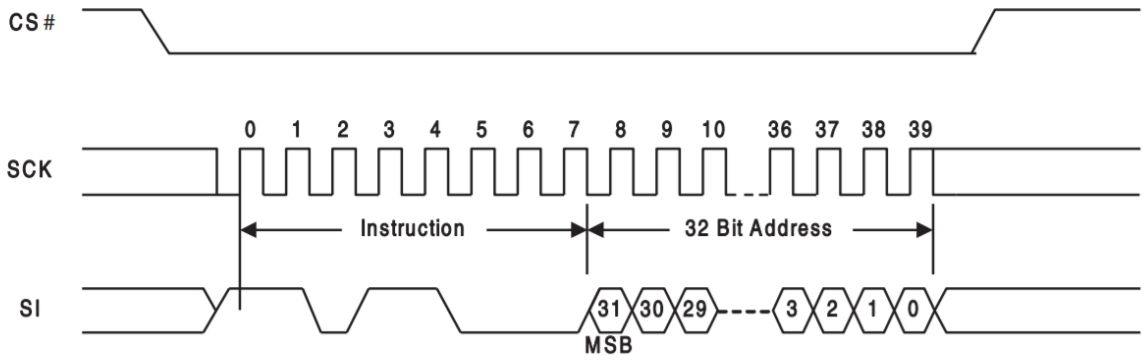

Рисунок 12.1 Временная диаграмма команды 4SE

13. ПРИМЕР РАБОТЫ

Чтение ID устройства:

```
uint main(void){

  //Инициализация регистров MFBSP0 для работы в режиме SPI
  SPI_Init(MFBSP0, get_cpu_freq(), SPI_FREQ);

    //Чтение ID флэш
    short ID = S25FL256_read_id(1);
```

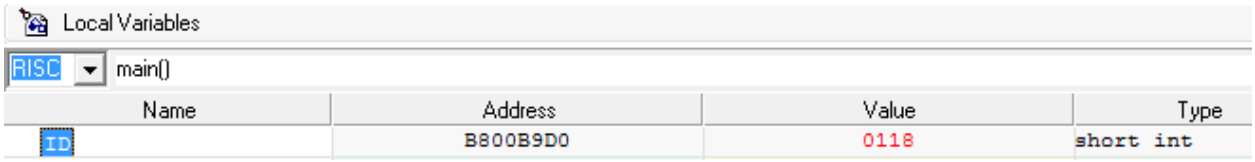

Рисунок 13.1 Результат, полученный функцией чтения ID

Тест записи и чтения 8-разрядными словами:

```
  //Объявление адреса сектора
  //Порядковый номер сектора
  uint j = 1;
  //Адрес сектора = начальный адрес 4кБ сектора + (номер
сектора)*(размер сектора)
  uint addr = BASE_ADDR_64kB + j*SIZE_64kB;

  //Тест записи и чтения 8-разрядными словами
  uint test_res = rw_8b_test(addr, D_8b_LEN);
```

Local Variables

RISC rw_8b_test(unsigned int, const unsigned int)

Name	Address	Value	Type
addr	B800B9C0	00030000	unsigned int
_d_8b_len	B800B9C4	00000010	const unsigned int
⊟ rd_8b	B800B968	[ ]	char[ ]
[0]	B800B968	01	char
[1]	B800B969	02	char
[2]	B800B96A	03	char
[3]	B800B96B	04	char
[4]	B800B96C	05	char
[5]	B800B96D	06	char
[6]	B800B96E	07	char
[7]	B800B96F	08	char
[8]	B800B970	09	char
[9]	B800B971	0A	char
[10]	B800B972	0B	char
[11]	B800B973	0C	char
[12]	B800B974	0D	char
[13]	B800B975	0E	char
[14]	B800B976	0F	char
[15]	B800B977	10	char
⊟ td_8b	B800B978	[ ]	char[ ]
[0]	B800B978	01	char
[1]	B800B979	02	char
[2]	B800B97A	03	char
[3]	B800B97B	04	char
[4]	B800B97C	05	char
[5]	B800B97D	06	char
[6]	B800B97E	07	char
[7]	B800B97F	08	char
[8]	B800B980	09	char
[9]	B800B981	0A	char
[10]	B800B982	0B	char
[11]	B800B983	0C	char
[12]	B800B984	0D	char
[13]	B800B985	0E	char
[14]	B800B986	0F	char
[15]	B800B987	10	char
test_res	B800B998	00000001	unsigned int

Рисунок 13.2 Результат теста записи и чтения 8-разрядными словами

Замер скорости записи и чтения 32-разрядными словами:

```
//Замер скорости записи и чтения 32-разрядными словами
//с использованием DMA и без, с последующей верификацией данных.
int diff_dma_pr = rw_32b_benchmark(addr, D_32b_LEN);
```

Local Variables

RISC | rw_32b_benchmark(unsigned int, const unsigned int)

Name	Address	Value	Type
addr	B800B9C0	196608	unsigned int
_d_32b_len	B800B9C4	16	const unsigned int
finish	B800B92C	2490262237	unsigned int
ok	B800B930	1	unsigned int
⊟ rd_32b_dma	B800B838	[ ]	unsigned int[ ]
[0]	B800B838	1	unsigned int
[1]	B800B83C	2	unsigned int
[2]	B800B840	3	unsigned int
[3]	B800B844	4	unsigned int
[4]	B800B848	5	unsigned int
[5]	B800B84C	6	unsigned int
[6]	B800B850	7	unsigned int
[7]	B800B854	8	unsigned int
[8]	B800B858	9	unsigned int
[9]	B800B85C	10	unsigned int
[10]	B800B860	11	unsigned int
[11]	B800B864	12	unsigned int
[12]	B800B868	13	unsigned int
[13]	B800B86C	14	unsigned int
[14]	B800B870	15	unsigned int
[15]	B800B874	16	unsigned int
⊟ rd_32b_pr	B800B880	[ ]	unsigned int[ ]
[0]	B800B880	1	unsigned int
[1]	B800B884	2	unsigned int
[2]	B800B888	3	unsigned int
[3]	B800B88C	4	unsigned int
[4]	B800B890	5	unsigned int
[5]	B800B894	6	unsigned int
[6]	B800B898	7	unsigned int
[7]	B800B89C	8	unsigned int
[8]	B800B8A0	9	unsigned int
[9]	B800B8A4	10	unsigned int
[10]	B800B8A8	11	unsigned int
[11]	B800B8AC	12	unsigned int
[12]	B800B8B0	13	unsigned int
[13]	B800B8B4	14	unsigned int
[14]	B800B8B8	15	unsigned int
[15]	B800B8BC	16	unsigned int
speed_dma	B800B93C	1977342	unsigned int
speed_program	B800B938	1629535	unsigned int
start	B800B928	2490254469	unsigned int
⊞ td_32b	B800B8C8	[ ]	unsigned int[ ]
time	B800B934	7768	unsigned int

Рисунок 13.3 Результат замера скорости записи и чтения 32-разрядными словами

Итог:

- скорость без использования DMA – 1629535 [байт/с] или 1,55 [Мбайт/с];
- скорость с использования DMA – 1977342 [байт/с] или 1,89 [Мбайт/с].

13.1 uint rw_8b_test(uint _addr, const uint _d_8b_len)

Тест записи и чтения 8-разрядными словами.

Функция последовательно выполняет:

- очистку сектора SPI-флэш памяти S25FL256 по адресу _addr;
- запись в SPI-флэш 8-разрядного массива данных размером _d_8b_len по адресу _addr;
- чтение записанных данных и последующая их верификация;
- возврат результата верификации.

```
uint rw_8b_test(uint _addr, const uint _d_8b_len){

  //Буферы передачи/приема
  char td_8b[_d_8b_len], rd_8b[_d_8b_len];

  //Заполнение массива передачи порядковыми значениями
  array8_fill(td_8b, _d_8b_len);

  //Очистка сектора для записи
  S25FL256_erase(_addr);

  //Запись
  S25FL256_page_program(
    _addr,
    td_8b,
    _d_8b_len,
    8,
    0);

  //Чтение
  S25FL256_read(
    _addr,
    rd_8b,
    _d_8b_len,
    8,
    0);

  //Проверка корректности данных
  uint test_res = array8_compare(td_8b, rd_8b, _d_8b_len);
  return test_res;
}
```

13.2 int rw_32b_benchmark(uint _addr, const uint _d_32b_len)

Замер скорости записи и чтения 32-разрядными словами с использованием DMA и без, с последующей верификацией данных.

Функция последовательно выполняет:

- очистку сектора SPI-флэш памяти S25FL256 по адресу _addr;
- запись во флэш 32-разрядного массива данных размером _d_32b_len по адресу _addr;
- чтение записанных данных без использования DMA,
- подсчет скорости операции чтения и последующая верификация принятных данных;
- чтение записанных данных с использования DMA,
- подсчет скорости операции чтения и последующая верификация принятных данных;
- возврат разницы скоростей [байт/сек] между операциями чтения с использования DMA и без использования DMA

В случае отрицательного результата верификации возвращает 0.

```
int rw_32b_benchmark(uint _addr, const uint _d_32b_len){

  //Величины в тактах процессорной частоты
  //Момент старта обмена
  //Момент окончания передачи и приема
  //Время совершения операциия в размерности тактовой частоты
процессора
  uint start, finish, time;
  //Скорость передачи и приема данных в байтах в секунду
  uint speed_program, speed_dma;

  //Буферы передачи/приема
  uint td_32b[_d_32b_len] __attribute__ ((aligned (8)));
  uint rd_32b_pr[_d_32b_len] __attribute__ ((aligned (8)));
  uint rd_32b_dma[_d_32b_len] __attribute__ ((aligned (8)));

  //Заполнение массива передачи порядковыми значениями
  array32_fill(td_32b, _d_32b_len);

  //Очистка сектора для записи
  S25FL256_erase(_addr);
```

```
  //Запись массива td_32b, заполненного порядковыми номерами
  S25FL256_page_program(
    _addr,
    td_32b,
    _d_32b_len,
    32,
    0);

  //Фиксация старта обмена, обмен, фиксация момента окончания обмена
  start = GetCP0_Count();
    S25FL256_read(
      _addr,
      rd_32b_pr,
      _d_32b_len,
      32,
      0);
  finish = GetCP0_Count();

  //Сравнение исходного массива td_32b со считанным без использования
DMA
  uint ok  = array32_compare(td_32b, rd_32b_pr, _d_32b_len);
  if(!ok)
      return 0;

  time = finish - start;
  //Расчет скорости  обмена в байтах в секунду
  speed_program = (((unsigned long long) _d_32b_len*4)*((unsigned long
long) get_cpu_freq()) / time); //[байт/сек]


 //Фиксация старта обмена, обмен, фиксация момента окончания обмена
  start = GetCP0_Count();
    S25FL256_read(
      _addr,
      rd_32b_dma,
      _d_32b_len,
      32,
      1);
  finish = GetCP0_Count();

  //Сравнение исходного массива td_32b со считанным с использования
DMA
  ok  = array32_compare(td_32b, rd_32b_pr, _d_32b_len);
  if(!ok)
      return 0;

  time = finish - start;
  //Расчет скорости  обмена в байтах в секунду
  speed_dma = (((unsigned long long) _d_32b_len*4)*((unsigned long
long) get_cpu_freq()) / time); //[байт/сек]


  //Расчет разности скоростей обмена dma - program [байт/сек]
  return (int) (speed_dma - speed_program);
}
```