НИИИС

МИКРОСХЕМЫ ИНТЕГРАЛЬНЫЕ
1914ВА018
Техническое описание

Содержание

1. Структурная блок-схема микросхемы

1.1 Структурная блок-схема микросхемы показана на рисунке 1.1.

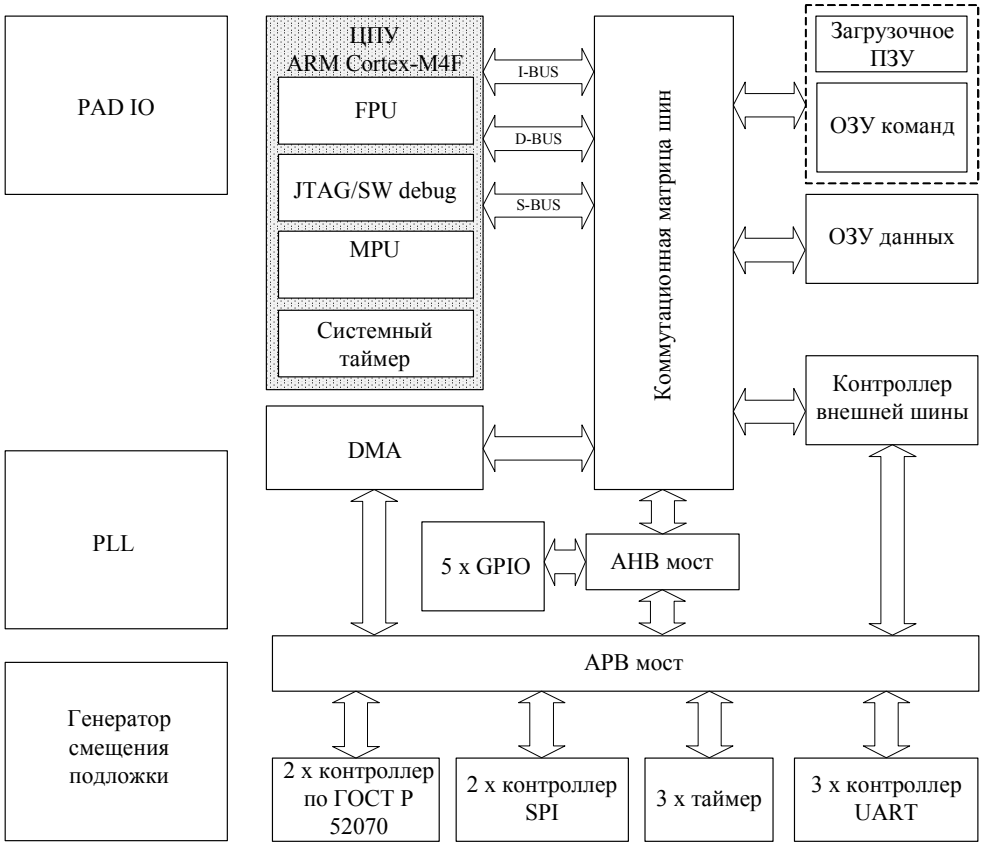

Рисунок 1.1 – Структурная схема микроконтроллера

В состав микроконтроллера входят функциональные элементы (см. рисунок 1.1):

1. процессорное ядро ARM Cortex-M4F с поддержкой набора одноцикловых команд умножения с накоплением, команд централизованного управления потоком данных, арифметических и логических команд и встроенным модулем обработки команд с плавающей запятой с одинарной точностью (блок FPU);
2. загрузочное ПЗУ объемом 4 Кбайт;
3. ОЗУ команд объемом 128 Кбайт;
4. ОЗУ данных объемом 64 Кбайт;
5. контроллер внешней системной шины;
6. 32-канальный контроллер прямого доступа к памяти (DMA);
7. схема сброса и сторожевой таймер;
8. синтезатор частоты на основе генератора с ФАПЧ;
9. три 32-разрядных таймера;
10. отладочный интерфейс JTAG и ARM SWD (Serial Wire Debug);
11. пять 16-разрядных портов ввода/вывода;
12. три последовательных порта UART;
13. два последовательных интерфейса SPI;
14. два контроллера интерфейса по ГОСТ Р 52070-2003.

2. Описание выводов

2.1 Описание выводов микроконтроллера

Таблица 2.1 – Описание выводов микроконтроллера

Вывод	Номер вывода	Тип вывода	Функция вывода
Порт A			
PA0	N19	I/O/Z	Линии порта A
PA1	N25	I/O/Z	
PA2	N20	I/O/Z	
PA3	P25	I/O/Z	
PA4	N17	I/O/Z	
PA5	P24	I/O/Z	
PA6	N15	I/O/Z	
PA7	T25	I/O/Z	
PA8	P21	I/O/Z	
PA9	P22	I/O/Z	
PA10	P20	I/O/Z	
PA11	R24	I/O/Z	
PA12	P16	I/O/Z	
PA13	U25	I/O/Z	
PA14	P19	I/O/Z	
PA15	R23	I/O/Z	
Порт B			
PB0	R21	I/O/Z	Линии порта B
PB1	R22	I/O/Z	
PB2	R18	I/O/Z	
PB3	T24	I/O/Z	
PB4	R17	I/O/Z	
PB5	Y25	I/O/Z	
PB6	R19	I/O/Z	
PB7	U24	I/O/Z	
PB8	R16	I/O/Z	
PB9	AA25	I/O/Z	
PB10	T22	I/O/Z	
PB11	T21	I/O/Z	
PB12	V24	I/O/Z	
PB13	T19	I/O/Z	
PB14	AC25	I/O/Z	
PB15	T17	I/O/Z	
Порт C			
PC0	AD21	I/O/Z	Линии порта C
PC1	W18	I/O/Z	
PC2	AC18	I/O/Z	
PC3	V18	I/O/Z	
PC4	AE24	I/O/Z	
PC5	Y17	I/O/Z	
PC6	AE23	I/O/Z	
PC7	V17	I/O/Z	
PC8	AD19	I/O/Z	
PC9	T16	I/O/Z	
PC10	AC17	I/O/Z	
PC11	W16	I/O/Z	
PC12	AE22	I/O/Z	
PC13	Y16	I/O/Z	
PC14	AD18	I/O/Z	
PC15	V16	I/O/Z	

Продолжение таблицы 2.1

Вывод	Номер вывода	Тип вывода	Функция вывода
Порт D			
PD0	AB16	I/O/Z	Линии порта D
PD1	AA16	I/O/Z	
PD2	AC16	I/O/Z	
PD3	W15	I/O/Z	
PD4	AD17	I/O/Z	
PD5	U15	I/O/Z	
PD6	AE19	I/O/Z	
PD7	V15	I/O/Z	
PD8	AD16	I/O/Z	
PD9	AA15	I/O/Z	
PD10	AB15	I/O/Z	
PD11	AE17	I/O/Z	
PD12	AE16	I/O/Z	
PD13	Y14	I/O/Z	
PD14	AD15	I/O/Z	
PD15	AA14	I/O/Z	
Порт E			
PE0	U11	I/O/Z	Линии порта E
PE1	AE7	I/O/Z	
PE2	W11	I/O/Z	
PE3	AD9	I/O/Z	
PE4	T11	I/O/Z	
PE5	AE6	I/O/Z	
PE6	R11	I/O/Z	
PE7	AB10	I/O/Z	
PE8	AA10	I/O/Z	
PE9	AD8	I/O/Z	
PE10	W10	I/O/Z	
PE11	AE4	I/O/Z	
PE12	U10	I/O/Z	
PE13	AC9	I/O/Z	
PE14	Y9	I/O/Z	
PE15	AB9	I/O/Z	
Контроллер по ГОСТ Р 52070-2003			
M1_BUS1_DP_IN	B13	I	Дифференциальный вход приемника основного канала первого контроллера ГОСТ Р 52070-2003
M1_BUS1_DN_IN	C13	I	
M1_BUS1_DP_OUT	A12	O	Дифференциальный выход передатчика основного канала первого контроллера ГОСТ Р 52070-2003
M1_BUS1_DN_OUT	K13	O	
M1_BUS1_OUT_EN	A11	O	Сигнал разрешения передачи основного канала первого контроллера ГОСТ Р 52070-2003
M1_BUS2_DP_IN	E12	I	Дифференциальный вход приемника резервного канала первого контроллера ГОСТ Р 52070-2003
M1_BUS2_DN_IN	J12	I	
M1_BUS2_DP_OUT	B11	O	Дифференциальный выход передатчика резервного канала первого контроллера ГОСТ Р 52070-2003
M1_BUS2_DN_OUT	D12	O	
M1_BUS2_OUT_EN	F12	O	Сигнал разрешения передачи резервного канала первого контроллера ГОСТ Р 52070-2003
M2_BUS1_DP_IN	F13	I	Дифференциальный вход приемника основного канала второго контроллера ГОСТ Р 52070-2003
M2_BUS1_DN_IN	G13	I	
M2_BUS1_DP_OUT	A14	O	Дифференциальный выход передатчика основного канала второго контроллера ГОСТ Р 52070-2003
M2_BUS1_DN_OUT	A13	O	
M2_BUS1_OUT_EN	J13	O	Сигнал разрешения передачи основного канала второго контроллера ГОСТ Р 52070-2003
M2_BUS2_DP_IN	E14	I	Дифференциальный вход приемника резервного канала второго контроллера ГОСТ Р 52070-2003
M2_BUS2_DN_IN	B14	I	
M2_BUS2_DP_OUT	D14	O	Дифференциальный выход передатчика резервного канала второго контроллера ГОСТ Р 52070-2003
M2_BUS2_DN_OUT	A16	O	
M2_BUS2_OUT_EN	F14	O	Сигнал разрешения передачи резервного канала второго контроллера ГОСТ Р 52070-2003

Продолжение таблицы 2.1

Вывод	**Номер вывода**	**Тип вывода**	**Функция вывода**
Контроллер внешней системной шины			
DATA0	W1	I/O/Z	Шина данных
DATA1	R8	I/O/Z	
DATA2	T2	I/O/Z	
DATA3	R5	I/O/Z	
DATA4	R4	I/O/Z	
DATA5	U1	I/O/Z	
DATA6	T1	I/O/Z	
DATA7	P6	I/O/Z	
DATA8	R2	I/O/Z	
DATA9	P5	I/O/Z	
DATA10	P4	I/O/Z	
DATA11	P9	I/O/Z	
DATA12	R1	I/O/Z	
DATA13	P8	I/O/Z	
DATA14	P1	I/O/Z	
DATA15	N8	I/O/Z	
DATA16	N2	I/O/Z	
DATA17	N10	I/O/Z	
DATA18	N3	I/O/Z	
DATA19	N7	I/O/Z	
DATA20	N1	I/O/Z	
DATA21	N6	I/O/Z	
DATA22	M1	I/O/Z	
DATA23	N9	I/O/Z	
DATA24	M2	I/O/Z	
DATA25	N11	I/O/Z	
DATA26	K1	I/O/Z	
DATA27	M5	I/O/Z	
DATA28	M4	I/O/Z	
DATA29	M6	I/O/Z	
DATA30	L2	I/O/Z	
DATA31	M10	I/O/Z	
ADDR0	J1	O	Шина адреса
ADDR1	M7	O	
ADDR2	L5	O	
ADDR3	L4	O	
ADDR4	L8	O	
ADDR5	K2	O	
ADDR6	L9	O	
ADDR7	F1	O	
ADDR8	L7	O	
ADDR9	J2	O	
ADDR10	L10	O	
ADDR11	E1	O	
ADDR12	K4	O	
ADDR13	K5	O	
ADDR14	H2	O	
ADDR15	K7	O	
ADDR16	C1	O	
ADDR17	K9	O	
ADDR18	J3	O	
ADDR19	J6	O	
ADDR20	J4	O	
ADDR21	J5	O	
ADDR22	B1	O	
ADDR23	J9	O	
ADDR24	H7	O	

Продолжение таблицы 2.1

Вывод	**Номер вывода**	**Тип вывода**	**Функция вывода**
ADDR25	H3	O	
ADDR26	H6	O	
ADDR27	E2	O	
ADDR28	G7	O	
ADDR29	G3	O	
ADDR30	G6	O	
ADDR31	D2	O	
OE	J25	O	Сигнал чтения данных
WE	L22	O	Сигнал записи данных
EXT_MEM_CLOCK	V11	O	Тактовый сигнал внешней шины
BE0	K25	O	Сигнал byte enable 0
BE1	M20	O	Сигнал byte enable 1
BE2	L24	O	Сигнал byte enable 2
BE3	M21	O	Сигнал byte enable 3
ALE	M22	O	Разрешение защелкивания адреса
CLE	M17	O	Разрешение защелкивания команды
BUSY	L25	I	Сигнал занятости NAND Flash-памяти
CS0	M18	O	Сигнал выборки чипа 0
CS1	M25	O	Сигнал выборки чипа 1
CS2	N18	O	Сигнал выборки чипа 2
CS3	N24	O	Сигнал выборки чипа 3
CS4	N16	O	Сигнал выборки чипа 4
CS5	N23	O	Сигнал выборки чипа 5
Контроллер UART			
UART1_RXD	B9	I	Вход приема последовательных данных модуля UART1
UART1_TXD	G11	O	Выход передачи последовательных данных UART1
UART2_RXD	A7	I	Вход приема последовательных данных модуля UART2
UART2_TXD	J11	O	Выход передачи последовательных данных UART2
UART3_RXD	B10	I	Вход приема последовательных данных модуля UART3
UART3_TXD	H11	O	Выход передачи последовательных данных UART3
Контроллер SPI			
SPI1_MISO	B15	I/O/Z	Сигнал MISO контроллера SPI1
SPI1_MOSI	K14	I/O/Z	Сигнал MOSI контроллера SPI1
SPI1_SCK	A17	I/O/Z	Сигнал тактирования контроллера SPI1
SPI1_SS_S	G14	I	Вход выбора устройства (режим ведомого) контроллера SPI1
SPI1_SS0_M	C15	O/Z	Сигнал выбора устройства 0 (режим ведущего) контроллера SPI1
SPI1_SS1_M	E15	O/Z	Сигнал выбора устройства 1 (режим ведущего) контроллера SPI1
SPI1_SS2_M	D15	O/Z	Сигнал выбора устройства 2 (режим ведущего) контроллера SPI1
SPI1_SS3_M	H15	O/Z	Сигнал выбора устройства 3 (режим ведущего) контроллера SPI1
SPI2_MISO	B16	I/O/Z	Сигнал MISO контроллера SPI2
SPI2_MOSI	J15	I/O/Z	Сигнал MOSI контроллера SPI2
SPI2_SCK	A20	I/O/Z	Сигнал тактирования контроллера SPI2
SPI2_SS_S	G15	I	Вход выбора устройства (режим ведомого) контроллера SPI2
SPI2_SS0_M	B17	O/Z	Сигнал выбора устройства 0 (режим ведущего) контроллера SPI2
SPI2_SS1_M	K15	O/Z	Сигнал выбора устройства 1 (режим ведущего) контроллера SPI2
SPI2_SS2_M	A21	O/Z	Сигнал выбора устройства 2 (режим ведущего) контроллера SPI2
SPI2_SS3_M	L15	O/Z	Сигнал выбора устройства 3 (режим ведущего) контроллера SPI2

Продолжение таблицы 2.1

Вывод	Номер вывода	Тип вывода	Функция вывода
Таймер			
TMR1_EXTIN	D10	I	Вход внешнего тактирования блока таймер 1
TMR2_EXTIN	E10	I	Вход внешнего тактирования блока таймер 2
TMR3_EXTIN	C10	I	Вход внешнего тактирования блока таймер 3
Внешние прерывания			
EXT_INT0	E11	I	Входы внешних прерываний
EXT_INT1	D11	I	
EXT_INT2	A9	I	
EXT_INT3	A10	I	
Системное управление			
XTAL1	AC13	I	Вход основного тактового сигнала/вывод для подключения кварцевого резонатора
XTAL2	U13	O	Вывод для подключения кварцевого резонатора
NRST	Y13	I	Вход внешнего сброса микроконтроллера
Генератор смещения подложки			
OUT_NVS_1V	Y6	O	Выход генератора смещения подложки №1
OUT_NVS_2V	V4	O	Выход генератора смещения подложки №2
NVS_PACKAGE	W7	—	Интерфейс к подложке кристалла
Отладочный интерфейс			
JTAG_SEL	D25	I	Сигнал выбора JTAG-контроллера
NTRST	K24	I	JTAG/SW интерфейс
TDI	K22	I	
SWDIOTMS	L21	I/O/Z	
SWCLKTCK	K21	I	
TDO	K18	O/Z	
TRACECLK	G25	O	Трассировочный порт
TRACEDATA3	L17	O	
TRACEDATA2	J24	O	
TRACEDATA1	L19	O	
TRACEDATA0	K23	O	
SWV	H24	O	
Питание			
VDD (NVS_1)	AB2	—	Питание генератора смещения подложки №1
VDD (NVS_2)	V5	—	Питание генератора смещения подложки №2
VDD (IO)	W3, U4, Y1, R3, P2, L1, H1, K3, G2, H4, C7, D9, A6, C11, B12, A15, A18, C16, B19, D18, G23, J22, F25, L23, M24, R25, V25, T23, W24, V22, AC19, AB17, AE20, AC15, AD14, AE12, AE9, AC10, AD7, AB8,	—	Питание площадок ввода-вывода
VDD (CORE)	U7, T9, R6, P10, N5, M9, L6, K8, J7, G9, J10, F11, K12, E13, J14, F15, H16, G17, J19, K17, L20, M16, N21, P17, R20, T18, U19, W17, U16, Y15, T14, AA13, Y11, V10, W9	—	Основное питание

Продолжение таблицы 2.1

Вывод	**Номер вывода**	**Тип вывода**	**Функция вывода**
VDD (PLL)	AD12	—	Питание PLL
GND (NVS_1)	V6	—	Общий
GND (NVS_2)	AA2	—	Общий
GND (IO)	W6, U5, R10, P7, N12, M8, M11, K6, J8, H5, F7, E9, K11, G12, M13, H14, L14, F16, H17, E18, G20, J21, L16, M19, N14, P18, P15, T20, U18, V21, Y19, AA17, T15, W14, P13, V12, R12, Y10, V9, AA8	—	Общий
GND (CORE)	Y2, AA1, V1, P3, N4, M3, G1, D1, F2, B6, A5, A8, C12, D13, C14, A19, A22, B20, F24, E25, H25, M23, N22, P23, W25, AB25, Y24, AD20, AE21, AE18, AC14, AB13, AC12, AE8, AE5, AD6	—	Общий
GND (PLL)	N13	—	Общий
Выводы для тестирования и исследования			
REMAP	K20	I	Сигнал выбора режима работы загрузчика. 0 – из внутреннего ПЗУ, 1 – с внешней шины.
PLL_EN	U12	I	Выводы для исследования блока PLL. Рекомендуется подсоединить к питанию.
TEST_PLL_EN	T13	I	Выводы для исследования блока PLL. Рекомендуется подсоединить к земле.
TEST_PLL_M_5	AB11	I	
TEST_PLL_M_4	AA11	I	
TEST_PLL_M_3	AC11	I	
TEST_PLL_M_2	W12	I	
TEST_PLL_M_1	AE10	I	
TEST_PLL_M_0	T12	I	
TEST_PLL_N_5	AD11	I	
TEST_PLL_N_4	Y12	I	
TEST_PLL_N_3	AB12	I	
TEST_PLL_N_2	AA12	I	
TEST_PLL_N_1	AE11	I	
TEST_PLL_N_0	R13	I	
TEST_PLL_RDY	AE13	O	Выводы для исследования блока PLL. Рекомендуется не подсоединять.
TEST_PLL_FOUT	W13	O	
TEST_MEM_OUTSIDE	AD13	I	Выводы для тестирования внутренней памяти. Рекомендуется подсоединить к земле.
TEST_MEM_NUMBER_3	AE14	I	
TEST_MEM_NUMBER_2	AE15	I	
TEST_MEM_NUMBER_1	U14	I	
TEST_MEM_NUMBER_0	AB14	I	

2.2 Диаграмма расположения выводов в корпусе

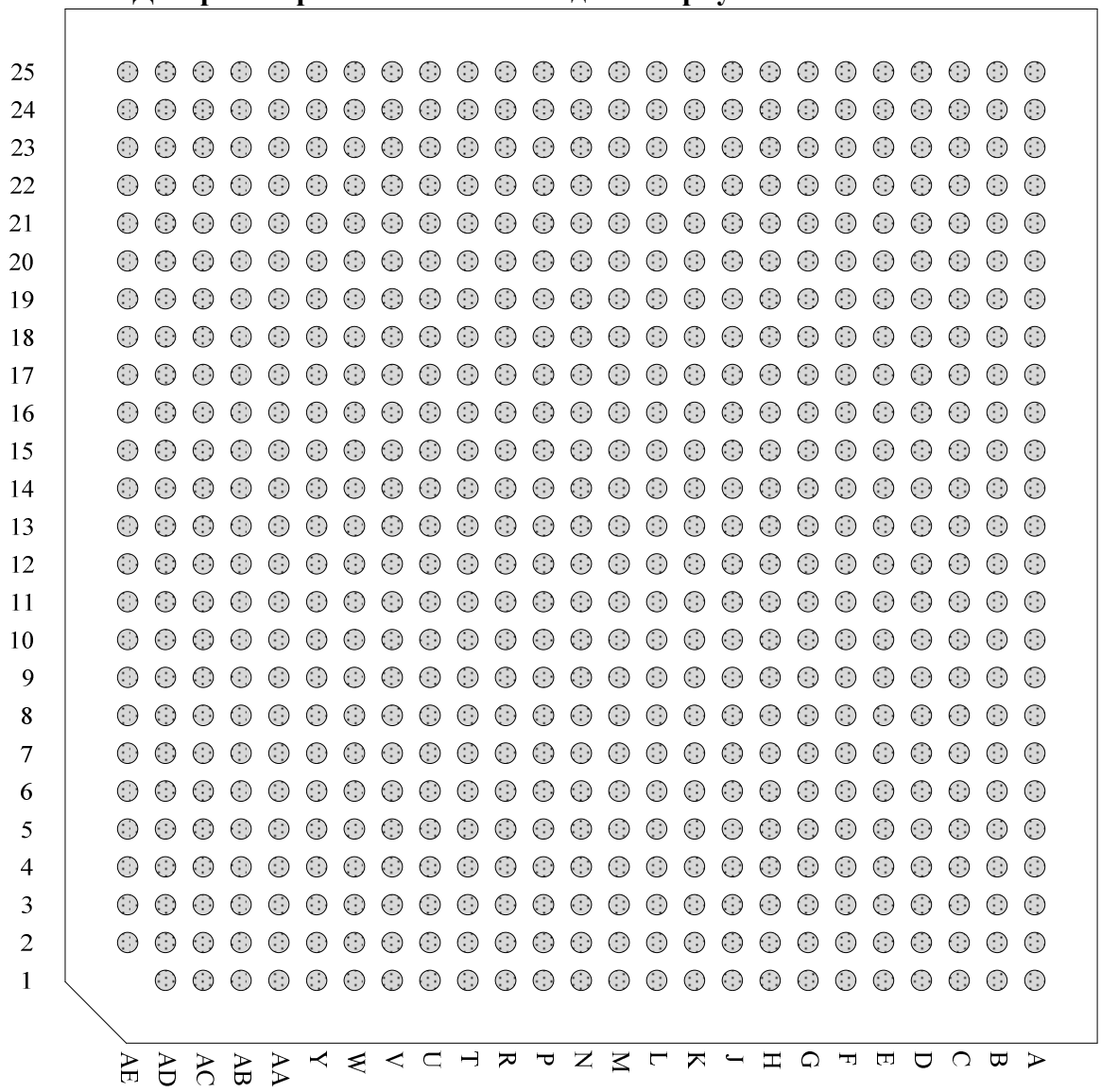

Рисунок 2.1 – Расположение выводов в 624-выводоном корпусе

3. Система питания

3.1 Микроконтроллер имеет несколько типов выводов питания.

VDD (NVS_1): Питание встроенного генератора смещения подложки №1. Входное напряжение должно быть 3,3В ±10%.

VDD (NVS_2): Питание встроенного генератора смещения подложки №2. Входное напряжение должно быть 3,3В ±10%.

VDD (IO): Питание площадок ввода-вывода микросхемы. Входное напряжение должно быть 3,3В ±10%.

VDD (CORE): Основное питание микросхемы. Включает питание внутренней цифровой части, памяти ОЗУ и ПЗУ. Входное напряжение должно быть 3,3В ±10%.

VDD (PLL): Питание схемы PLL. Входное напряжение должно быть 3,3В ±10%.

GND (NVS_1): Земля питания генератора смещения подложки №1.

GND (NVS_2): Земля питания генератора смещения подложки №2.

GND (IO): Земля питания VDD (IO).

GND (CORE): Земля питания VDD (CORE).

GND (PLL): Земля питания VDD (PLL).

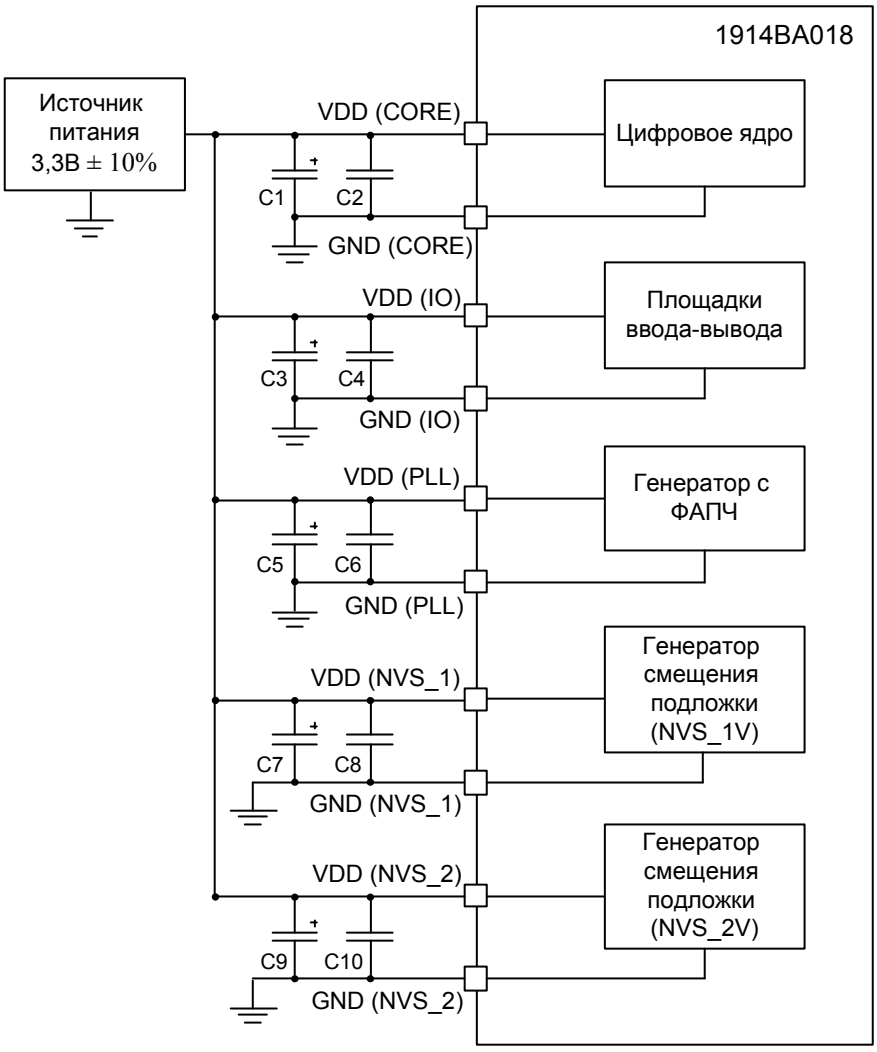

Рисунок 3.1 – структурная блок-схема подачи питания

Примечания:

1. конденсаторы должны быть установлены у каждого вывода питания;
2. конденсаторы C1, C3, C5, C7, C9 = 22 мкФ, C2, C4, C6, C8, C10 = 0,1 мкФ.

4 Система тактирования микроконтроллера

4.1 Описание схемы тактирования

4.1.1 Микроконтроллер имеет осциллятор и встроенный блок умножения/деления частоты (блок PLL). Схема формирования тактовых частот показана на рисунке 4.1

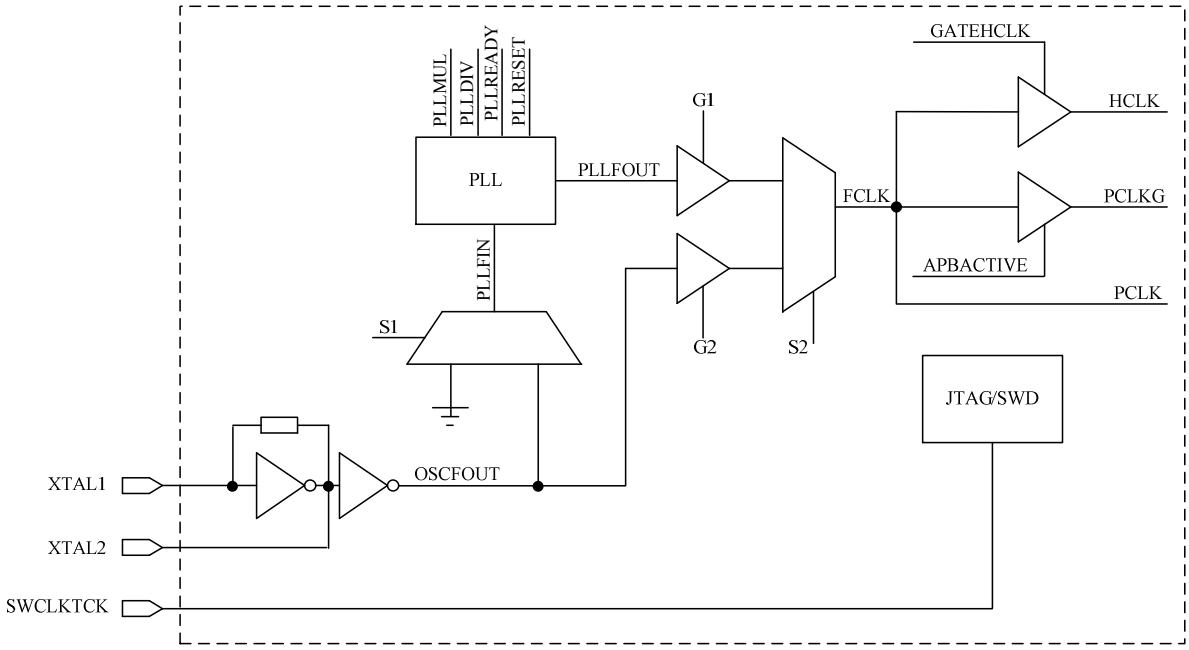

Рисунок 4.1 – Структурная блок-схема формирования тактовых частот

Осциллятор предназначен для выработки периодического сигнала OSCFOUT. Для работы осциллятора снаружи должен быть подключен кварцевый резонатор (выводы XTAL1 и XTAL2) либо подан периодический сигнал от внешнего генератора на вход XTAL1.

Блок PLL предназначен для умножения или деления входной тактовой частоты. Допустимые коэффициенты умножения лежат в диапазоне от 2 до 32 с шагом 1. Допустимые коэффициенты деления частоты лежат в диапазоне от 1 до 32 с шагом 1. Допустимая входная частота блок PLL лежит в диапазоне от 12 до 16 МГц. На вход PLLFIN блока PLL поступает периодический сигнал с выхода осциллятора либо вход PLLFIN подключен к «земле».

Микроконтроллер тактируется системной частотой FCLK. Источниками системной частоты FCLK могут быть выходной сигнал PLLFOUT блока PLL или выходной сигнал осциллятора OSCFOUT. Управление тактовыми сигналами микроконтроллера осуществляет автомат синхронизации. Управляя контрольными сигналами S1, S2, G1, G2, автомат синхронизации осуществляет выбор источников тактового сигнала, управляет работой PLL, а также осуществляет переключение системной частоты.

4.2 Переключение с одной частоты на другую

4.2.1 При включении питания в качестве системной частоты используется тактовая частота сигнала OSCFOUT. Для изменения тактовой частоты микроконтроллера можно воспользоваться блоком PLL.

Вычисление выходной частоты производится по формуле:

$$F_{PLLFOUT} = F_{PLLFIN} \cdot \frac{PLLMUL}{PLLDIV} \qquad (4.1)$$

где F_{PLLFIN} – частота на входе блока PLL;

PLLMUL – значение коэффициента умножения блока PLL;

PLLDIV – значение коэффициента деления блока PLL.

Для перехода на системную частоту от блока PLL необходимо настроить коэффициенты умножения и деления с помощью регистра CLKCTRL->PLLCONF.

Автомат синхронизации переключит источник тактирования блока PLL на сигнал осциллятора OSCFOUT и после стабилизации частоты блока PLL выполнит переключение системной частоты на частоту PLLFOUT.

Для возврата на системную частоту от осциллятора необходимо установить значение 0 для коэффициентов умножения и деления в регистре CLKCTRL->PLLCONF.

4.3 Описание регистров контроллера тактовых сигналов CLKCTRL

Таблица 4.1 – Описание регистров контроллера тактовых сигналов

Базовый адрес	**Название**	**Описание**
0x4002_0000	NIIIS_SYSCTRL	Контроллер тактовых сигналов
Смещение		
0x14	PLL_CONF	Регистр CLKCTRL->PLL_CONF управления работой блока PLL

Таблица 4.2 – Регистр CLKCTRL->PLL_CONF

Номер	31…20	19…14	13…8	7…0
Доступ	U	R/W	R/W	R/W
Сброс	0	0	0	0xFF
	-	**PLL_MUL**	**PLL_DIV**	**PLL_COUNT**

Таблица 4.3 – Описание бит регистра CLKCTRL->PLL_CONF

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…20	—	Зарезервировано
19…14	PLL_MUL	Коэффициент умножения частоты
13…8	PLL_DIV	Коэффициент деления частоты
7…0	PLL_COUNT	Таймаут сигнала Ready

5 Организация памяти

5.1 Структурная схема организации памяти

5.1.1 Структурная схема организации памяти показана на рисунке 5.1.

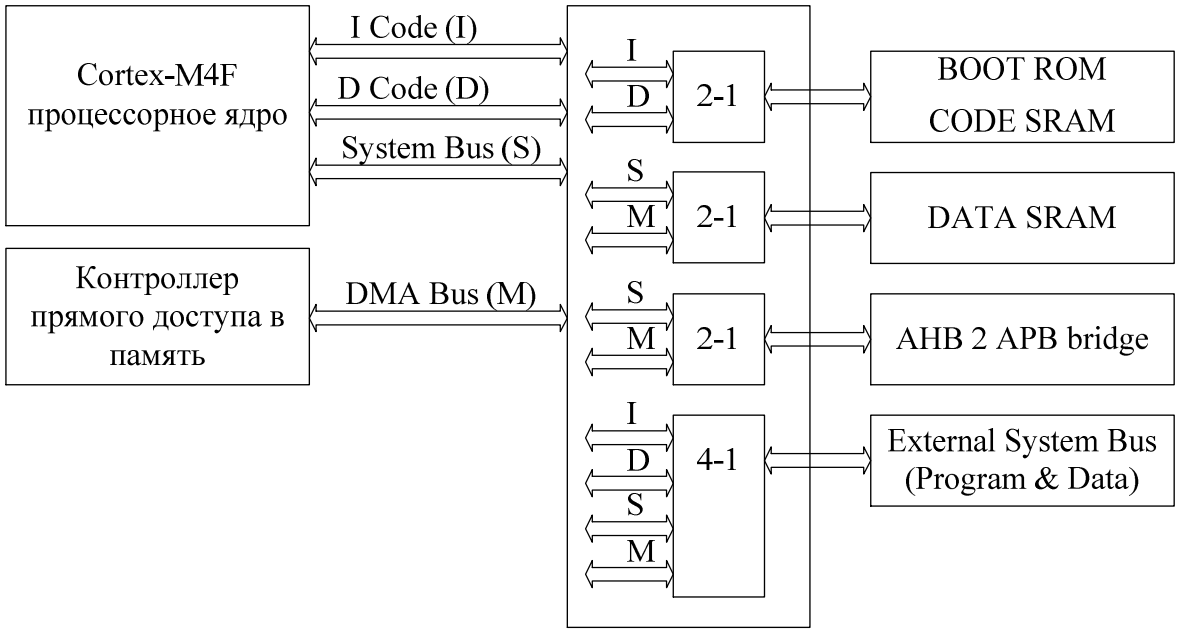

Рисунок 5.1 – Структурная схема организации памяти

Процессорное ядро имеет три системных шины:

– I Code – шина выборки инструкций;

– D Code – шина выборки данных, расположенных в коде программы;

– S Bus – шина выборки данных расположенных в области ОЗУ.

Также в микроконтроллере реализован контроллер прямого доступа в память (DMA), который осуществляет выборку через шину DMA Bus.

Все адресное пространство микроконтроллера едино и имеет максимальный объем 4 Гбайта. В данное адресное пространство отображаются различные модули памяти и периферии.

5.2 Карта распределения основных областей памяти

5.2.1 Карта распределения основных областей памяти показана на рисунке 5.2

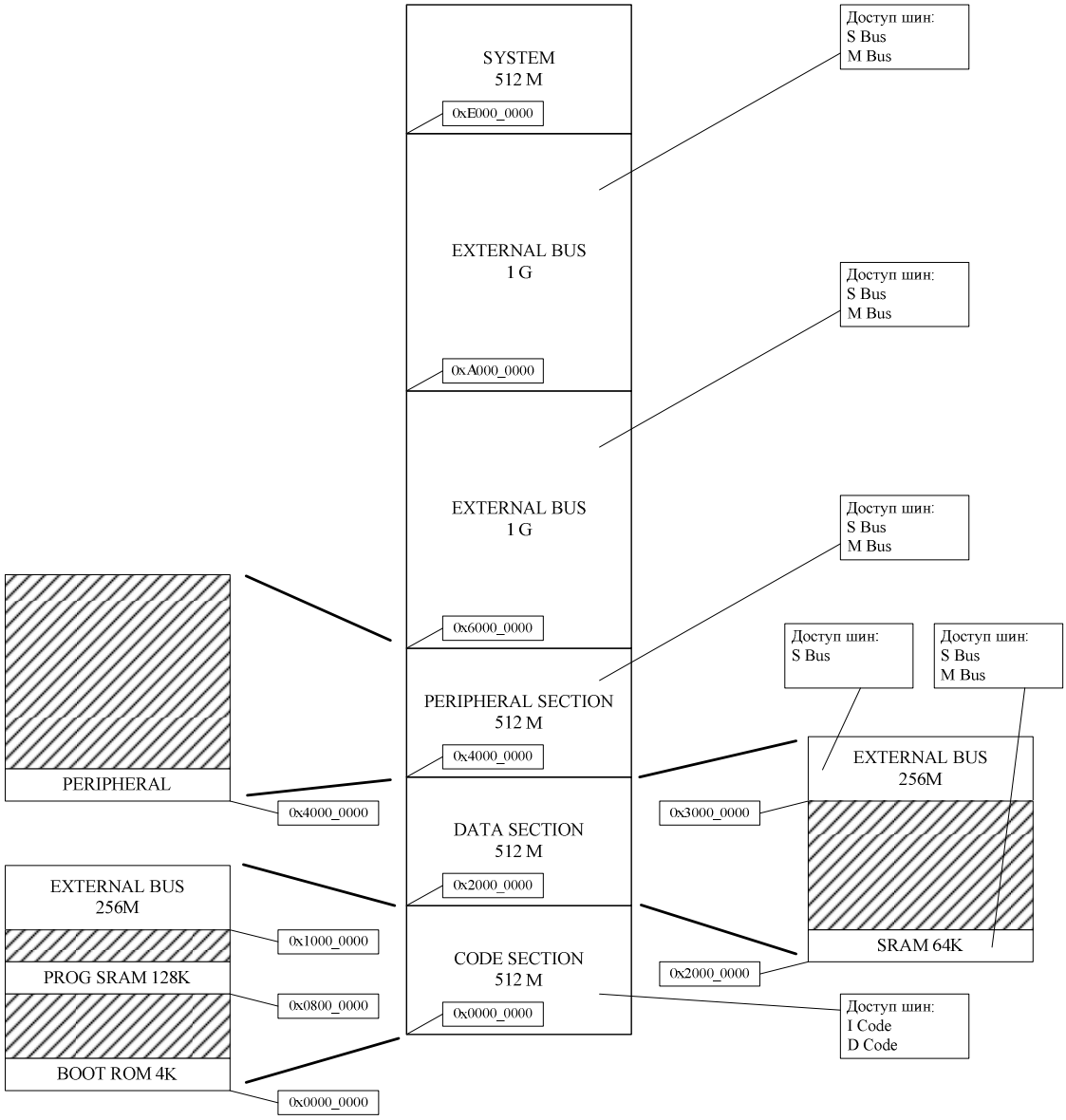

Рисунок 5.2 – Карта распределения основных областей памяти

Секция CODE

Область BOOT ROM

Предназначена для хранения программы запуска микроконтроллера; в ходе выполнения этой программы определяется режим запуска основной программы.

Область PROG SRAM

Основная область памяти программ. Область предназначена для хранения основной рабочей программы.

Область EXTERNAL BUS

Область отображения внешней системной шины в адресное пространство области программы. Предназначена для хранения кода программ во внешних микросхемах памяти, подсоединенных к внешней системной шине.

Секция DATA

Область SRAM

Основная область ОЗУ, предназначенная для хранения данных программы. В данной области также располагаются стек (stack) и «куча» (heap) программы. Адресные диапазоны стека и «кучи» задаются пользователем при написании программы.

Область EXTERNAL BUS

Область отображения внешней системной шины в адресное пространство области данных и предназначена для хранения данных во внешних микросхемах памяти, подсоединенных к внешней системной шине.

Секция PERIPHERAL

Область PERIPHERAL

Область отображения регистров периферии в общее адресное пространство памяти.

Область EXTERNAL BUS

Область отображения внешней системной шины в адресное пространство области периферии и предназначена для хранения данных во внешних микросхемах памяти или для работы с периферийными устройствами, подсоединенными к внешней системной шине.

Секция EXTERNAL BUS

Область EXTERNAL BUS

Область отображения внешней системной шины в адресное пространство области внешней памяти и периферии. Эта секция предназначена для хранения данных во внешних микросхемах памяти или для работы с периферийными устройствами, подсоединенными к внешней системной шине.

Секция SYSTEM

Предназначена для отображения системных регистров ядра и системной периферии.

Память BOOT ROM

Память области BOOT ROM реализована в виде масочной ПЗУ, с занесением информации одним из технологических слоев при изготовлении кристалла микроконтроллера. Скорость доступа к памяти BOOT ROM – 1 цикл системной частоты.

Память PROG SRAM

Память области PROG SRAM реализована в виде блока статической памяти. Скорость доступа к памяти PROG SRAM – 1 цикл системной частоты.

Память SRAM

Память области SRAM реализована в виде блока статической памяти. Скорость доступа к памяти SRAM – 1 цикл системной частоты.

6 Базовые адресы микроконтроллера

Таблица 6.1 – Базовые адресы микроконтроллера

Адрес	Размер	Обозначение		Шины	Примечание
Память программ					
0x0000_0000	4 Кбайт	BOOT ROM		I, D	Область памяти загрузочного ПЗУ, из этой памяти микроконтроллер начинает работу, определяет дальнейший режим функционирования и выполняет необходимые настройки
0x0800_0000	384 Кбайт	PROG SRAM		I, D	Область ОЗУ для основной пользовательской программы
0x1000_0000	256 Мбайт	EXTERNAL BUS		I, D	Область памяти для отображения внешней системной шины в CODE SECTION
Память данных					
0x2000_000	128 Кбайт	DATA SRAM		S, M	Область статического ОЗУ в DATA SECTION, предназначена для данных
0x3000_0000	256 Мбайт	EXTERNAL BUS		S	Область памяти для отображения внешней системной шины в DATA SECTION
Периферия					
0x4000_0000	8 Кбайт	0	MIL-STD-1553B1	S, M	Регистры контроллера интерфейса MIL-STD-1553B1
0x4000_2000	8 Кбайт	1	MIL-STD-1553B2		Регистры контроллера интерфейса MIL-STD-1553B2
0x4000_4000	4 Кбайт	2	UART1		Регистры контроллера интерфейса UART1
0x4000_5000	4 Кбайт	3	UART2		Регистры контроллера интерфейса UART2
0x4000_6000	4 Кбайт	4	UART3		Регистры контроллера интерфейса UART3
0x4000_7000	4 Кбайт	5	SPI1		Регистры контроллера интерфейса SPI1
0x4000_8000	4 Кбайт	6	SPI2		Регистры контроллера интерфейса SPI2
0x4000_9000	4 Кбайт	7	TIMER1		Регистры управления Таймер 1
0x4000_A000	4 Кбайт	8	TIMER2		Регистры управления Таймер 2
0x4001_B000	4 Кбайт	9	TIMER3		Регистры управления Таймер 3
0x4001_C000	4 Кбайт	10	WDT		Регистры управления сторожевого таймера
0x4001_D000	4 Кбайт	11	DMA		Регистры DMA-контроллера
0x4001_E000	4 Кбайт	12	EXTERNAL BUS CONTROLLER		Регистры контроллера внешней системной шины
0x4002_0000	4 Кбайт	13	CLKCTRL		Регистры контроллера тактовых сигналов
0x4002_1000	4 Кбайт	14	PORTA		Регистры управления порта A
0x4002_2000	4 Кбайт	15	PORTB		Регистры управления порта B
0x4002_3000	4 Кбайт	16	PORTC		Регистры управления порта C
0x4002_4000	4 Кбайт	17	PORTD		Регистры управления порта D
0x4002_5000	4 Кбайт	18	PORTE		Регистры управления порта E
Внешняя системная шина					
0x6000_0000	2 Гбайт	EXTERNAL BUS		S, M	Область памяти для отображения внешней системной шины в секции EXTERNAL BUS
SYSTEM REGION					
0xE000_0000	256 Мбайт			-	Системные регистры процессора ARM Cortex-M4F

7 Загрузочное ПЗУ и режимы загрузки

7.1 После включения питания и снятия внешних сигналов сброса, микроконтроллер начинает выполнять программу из *загрузочной области.* Состояние внешнего вывода REMAP определяет загрузочную область микроконтроллера.

7.1.1 Режим внешнего загрузчика. Для переключения микроконтроллера в режим внешнего загрузчика необходимо притянуть вывод REMAP к высокому логическому уровню. Все обращения по адресам 0x0000_0000…0x0000_0FFF выполняются к внешней системной шине. Предполагается, что в режиме внешнего загрузчика на внешней системной шине присутствуют микросхемы памяти с пользовательской загрузочной программой.

7.1.2 Режим внутреннего загрузчика. Для переключения микроконтроллера в режим внешнего загрузчика необходимо притянуть вывод REMAP к низкому логическому уровню. Все обращения по адресам 0x0000_0000…0x0000_0FFF выполняются к загрузочному ПЗУ (BOOT ROM). В режиме внутреннего загрузчика работает загрузочная программа, зашитая в масочном ПЗУ.

В загрузочной программе внутреннего загрузчика микроконтроллер определяет, в каком из режимов он будет функционировать, и переходит в этот режим. Режим функционирования определяется внешними выводами PE[15:13].

Загрузочная программа (Загрузчик) получает управление сразу после включения.

Загрузчик имеет несколько режимов работы определяемых состоянием внешних выводов в момент получения управления.

Таблица 7.1 – Режимы работы внутреннего загрузчика

PE[15:13]	Режим	Описание
000	Непосредственное выполнение программы из внешней памяти. Шина в 32-битном режиме.	Микроконтроллер конфигурирует внешнюю шина на работу в 32-битном режиме со значением WAIT_STATES = 4 и начинает выполнять программу из внешней памяти.
001	Загрузка программы из внешней памяти. Шина в 8-битном режиме.	Микроконтроллер конфигурирует внешнюю шину на работу в 8-битном режиме со значением WAIT_STATES = 4, копирует пользовательскую программу из внешней памяти во внутреннее ОЗУ и передаёт ей управление.
010	Загрузка программы из внешней памяти. Шина в 32-битном режиме.	Микроконтроллер конфигурирует внешнюю шину на работу в 32-битном режиме со значением WAIT_STATES = 4, копирует пользовательскую программу из внешней памяти во внутреннее ОЗУ и передаёт ей управление.
011	Зарезервировано	–
100	Непосредственное выполнение программы из внешней памяти. Шина в 8-битном режиме	Микроконтроллер конфигурирует внешнюю шина на работу в 8-битном режиме со значением WAIT_STATES = 4 и начинает выполнять программу из внешней памяти.
101	Загрузка по UART1	Микроконтроллер через интерфейс UART1 получает код пользовательской программы для записи во внутреннее ОЗУ и передачи ей управления.
110	Загрузка по UART2	Микроконтроллер через интерфейс UART2 получает код пользовательской программы для записи во внутреннее ОЗУ и передачи ей управления.
111	Загрузка по UART3 (режим по умолчанию)	Микроконтроллер через интерфейс UART3 получает код пользовательской программы для записи во внутреннее ОЗУ и передачи ей управления.

В режимах непосредственного выполнения программы из внешней памяти Загрузчик конфигурирует системную шину на разрядность операций 8 или 32 бита и передает управление по фиксированному адресу 0x10000000. По этому адресу должна быть размещена пользовательская программа.

В режимах загрузки программы из внешней памяти Загрузчик конфигурирует системную шину на разрядность операций 8 или 32 бита, получает адрес для загрузки программы из слова по адресу 0x10000000, получает размер программы в байтах из слова по адресу 0x10000004, копирует (побайтно при 8-битном режиме или пословно при 32-битном режиме) программу пользователя из внешней памяти начиная с адреса 0x10000008 во внутреннюю по адресу загрузки и передает ей управление.

UART загрузчик Три режима загрузки по UART различаются только используемым для загрузки портом, выполняемые при этом операции и протокол обмена идентичны.

Для связи по UART выбраны следующие параметры канала связи:

– количество бит данных – 8;

– четность – нет;

– количество Stop бит – 1;

– Загрузчик всегда выступает в качестве подчиненного устройства (Slave), а внешнее устройство, подающее команды, – в качестве ведущего устройства (Master);

– данные передаются младшим битом вперед.

В начале работы Загрузчика в режиме загрузки по UART происходит синхронизация с внешним устройством.

Внешнее устройство постоянно посылает в канал синхросимвол – 0x00.

Загрузчик начинает прием на максимально возможной скорости (значение делителя 16).

Если при приеме пяти байтов хотя бы один не 0x00, скорость приема снижается путем увеличения делителя на 1. Подстройка скорости приема выполняется до приема пяти байтов 0x00 подряд.

После настройки скорости приема Загрузчик выдает приглашение (байт 0x3E ('>')) внешнему устройству переходит в режим ожидания команд.

По приему приглашения Загрузчика внешнее устройство завершает выдачу синхросимволов и может подавать команды согласно протоколу обмена.

В Загрузчике реализован набор команд, необходимых для записи в ОЗУ какой-либо программы, верификации ее и запуска на выполнение, а также имеется возможность явного задания внешним устройством скорости обмена.

Протокол обмена по UART

После синхронизации с внешним устройством, подающим команды (Master), загрузчик переходит в диспетчер команд.

Внешнему устройству доступны следующие команды из таблицы 7.2.

Таблица 7.2 – Команды Загрузчика в режимах UART

Команда	**Код**	**ASCII символ**	**Описание**
CMD_SYNC	x00		Пустая команда. Загрузчик ее принимает, но ничего по ней не делает
CMD_BAUD	x42	'B'	Установка скорости обмена
CMD_LOAD	x4C	'L'	Загрузка массива байт
CMD_VFY	x59	'Y'	Выдача массива байт
CMD_RUN	x52	'R'	Запуск программы на выполнение

Команды (кроме CMD_SYNC) имеют один или два параметра.

Параметры команд – это 4-байтные слова. Параметры передаются младшим байтом вперед. В качестве значения параметра запрещено использовать число 0xFFFFFFFF.

Анализ допустимости параметров, Загрузчик производит только после принятия всех параметров команды.

По завершении обработки команды, Загрузчик выдает в канал связи с внешним устройством либо приглашение (сигнал об успешной обработке команды и ожидании следующей), либо сообщение об ошибке.

Сообщения об ошибках – это 2-байтные слова. Первый байт всегда 0x45 ('E'). Второй байт определяет тип ошибки.

После выдачи сообщения об ошибке Загрузчик переходит в режим ожидания следующей команды, поэтому внешнее устройство, после получения такого сообщения, должно прекратить передачу данных, относящихся к текущей команде.

Команда CMD_SYNC

Пустая команда. Загрузчик принимает ее, но ничего по ней не делает. Код команды соответствует символу синхронизации.

Таблица 7.3 – Команда CMD_SYNC

Код команды	CMD_SYNC = 0x00
ASCII символ, соответствующий коду команды	нет
Количество параметров команды	0
Ответ загрузчика	0x3E ('>')

Команда CMD_BAUD

Установка скорости обмена.

Таблица 7.4 – Команда CMD_BAUD

Код команды	CMD_BAUD = 0x42
ASCII символ, соответствующий коду команды	'B'
Количество параметров команды	1
Параметр	Новое значение делителя частоты
Ответ загрузчика	0x3E ('>') – успешное задание новой скорости обмена 0x45423E (“EB>”) – ошибка. Скорость обмена не изменена

Команда CMD_LOAD

Загрузка массива байт в память микроконтроллера.

Таблица 7.5 – Команда CMD_LOAD

Код команды	CMD_LOAD = 0x4C
ASCII символ, соответствующий коду команды	'L'
Количество параметров команды	4
Параметр 1	Адрес памяти приемника данных.
Параметр 2	Размер массива в байтах
Параметр 3	Массив данных
Параметр 4	Контрольная сумма массива данных
Ответ загрузчика	0x3E ('>') - успех. Данные записаны по заданному адресу 0x45413E (“EA>”) – задан недопустимый адрес. 0x454C3E (“EL>”) – заданный размер данных выходит за границы допустимого диапазона 0x45533E (“ES>”) – посчитанная контрольная сумма записанных данных не совпадает с заданной. Данные записаны по заданному адресу.

Команда CMD_VFY

Выдача массива байт из памяти микроконтроллера.

Таблица 7.6 – Команда CMD_VFY

Код команды	CMD_VFY = 0x59
ASCII символ, соответствующий коду команды	'Y'
Количество параметров команды	2
Параметр 1	Адрес памяти источника данных
Параметр 2	Размер массива в байтах
Ответ загрузчика	Массив запрошенных данных, два байта контрольной суммы, 0x3E ('>') - успех. 0x45413E (“EA>”) – задан недопустимый адрес. 0x454C3E (“EL>”) – запрошенный размер данных выходит за границы допустимого диапазона

Команда CMD_RUN

Запуск программы на выполнение.

Таблица 7.7 – Команда CMD_RUN

Код команды	CMD_RUN = 0x52
ASCII символ, соответствующий коду команды	'R'
Количество параметров команды	1
Параметр 1	Адрес таблицы векторов загруженной программы
Ответ загрузчика	0x3E ('>') - успех. Управление передается по указанному адресу. 0x45413E (“EA>”) – задан недопустимый адрес. Управление остается у загрузчика.

Неизвестная команда

Таблица 7.8 – Неизвестная команда

Код команды	Любой код, отличный от кодов из пп.0, 0, 0, 0, 0
Ответ загрузчика	0x45553E (“EU>”) – принята неизвестная команда.

7.1.3 Алгоритм расчета контрольной суммы данных

Контрольная сумма – это двухбайтное слово, образуемое путем последовательного побайтного сложения поступаемых в порядке приема или передачи данных с потерей бита перехода при переполнении.

7.1.4 Режим отладки. При работе в режиме отладки к микроконтроллеру может быть подключен JTAG/SW адаптер, с помощью которого программные средства разработки позволяют работать с микроконтроллером в отладочном режиме. В этом случае линии JTAG должны быть подтянуты к питанию.

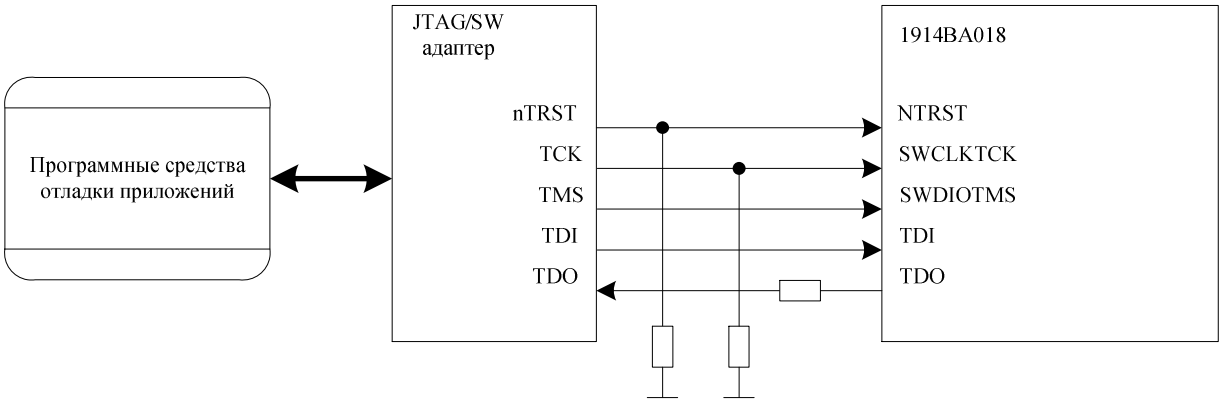

Рисунок 7.1 – Схема работы в режиме отладки

В отладочном режиме можно:

– записывать и считывать содержимое внутренней памяти, периферии;
– выполнять программу в пошаговом режиме;
– запускать программу в нормальном режиме;
– останавливать программу по точкам остановки:
– просматривать переменные выполняемой программы.

8 Процессорное ядро ARM Cortex-M4F

8.1 Краткий обзор процессора и периферии ядра

8.1.1 Cortex-M4 – высокопроизводительный 32-разрядный процессор. Он имеет следующие особенности:

– высокая производительность и быстрая обработка прерываний;

– расширенная система отладки с возможностью установки точек останова и управления потоком данных;

– эффективное ядро для работы системы и памяти;

– низкое энергопотребление;

– защищенная платформа с интегрированным блоком защиты памяти (MPU).

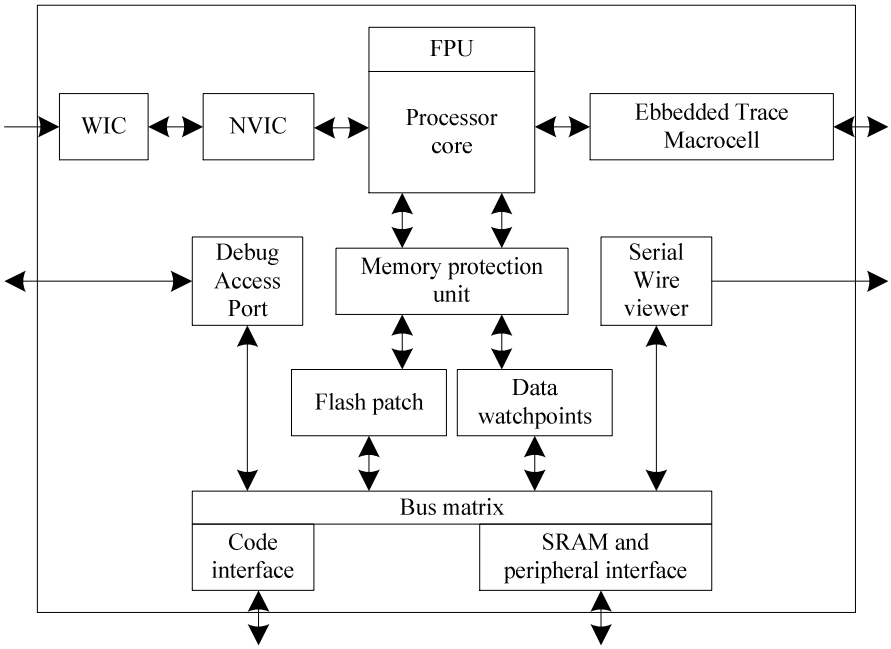

Рисунок 8.1 – Структурная блок-схема процессорного ядра Cortex-M4F

Процессор Cortex-M4 построен на высокопроизводительном ядре с 3-стадийной конвейерной гарвардской архитектурой, которая делает его идеальным для встроенных приложений.

Процессор предлагает большую энергоэффективность, эффективный набор инструкций и оптимальные аппаратные решения, включая совместимый с IEEE754 модуль арифметики с плавающей точкой одинарной точности Floating-Point Unit (FPU). В наборе команд присутствуют оптимизированные команды, большинство из которых выполняются за один цикл, команды SIMD (Single instruction multiple data), команды арифметики с насыщением и специализированное аппаратное деление.

Процессор реализует набор инструкций Thumb-2, обеспечивающих высокую плотность кода и пониженные требования к памяти программ.

Контроллер прерываний NVIC непосредственно интегрирован в процессор. NVIC включает в себя немаскируемое прерывание (NMI) и обеспечивает до 256 приоритетных уровней прерывания. Тесная интеграция ядра процессора и NVIC обеспечивает быстрое выполнение программы обработки прерываний (ISRs), что приводит к существенному уменьшению времени ожидания прерывания. Это достигается с помощью аппаратного сохранения регистров в стеке.

Реализация обработчика прерываний не требует его представления в ассемблерном коде. Оптимизация при помощи механизма сцепления обработки прерываний так же значительно уменьшает задержки при переключении из одной программы обработки прерывания на другую.

8.2 Периферийные блоки ядра

8.2.1 Периферия ядра Cortex-M4 содержит:

а) Nested Vectored Interrupt Controller – контроллер вложенных векторных прерываний (NVIC);

б) System control block – блок управления системой (SCB). Блок предоставляет информацию по реализации и управлению системой, включая конфигурацию, управление и сообщения системных исключений;

в) System timer - системный таймер. 24-разрядный таймером обратного отчета. Таймер используется в системе реального времени (Real Time Operating System - RTOS) или в качестве простого счетчика;

г) Memory protection unit - блок защиты памяти (MPU). Повышает надежность системы, определяя различные атрибуты для различных областей памяти. Поддерживает до 8 различных областей и дополнительную предопределенную «фоновую» область;

д) Floating-point unit – модуль вычисления с плавающей точкой (FPU). Поддерживает операции с одинарной точностью, 32-битные значения с плавающей точкой.

8.3 Программная модель

8.3.1 Режимы функционирования процессора:

а) Режим потока (Thread). Используется для выполнения программ приложений. После сброса процессор входит в режим потока;

б) Режим обработчика (Handler). Используется для обработки исключений. Процессор возвращается в режим потока, когда закончится обработка всех исключительных ситуаций.

Определены следующие уровни привилегий для выполнения программ:

а) Непривилегированный уровень:

– имеет ограниченный доступ к инструкциям MSR и MRS, и не может использовать инструкцию CPS;

– не имеет доступа к таймеру системы, NVIC, или блоку управления системой;

– имеет ограниченный доступ к памяти и периферии.

б) Привилегированный уровень:

На привилегированном уровне программное обеспечение может использовать все инструкции и иметь доступ ко всем ресурсам.

В режиме потока (Thread) регистр управления CONTROL контролирует выполнение как «привилегированного», так и не «непривилегированного» программное обеспечение. В режиме обработчика выполняется только «привилегированное» программное обеспечение.

Только «привилегированное» программное обеспечение может выполнять запись в регистр управления CONTROL, чтобы изменить уровень привилегий для выполнения программного обеспечения в режиме потока. «Непривилегированное» программное обеспечение может использовать инструкцию SVC для вызова супервизора, чтобы передать управление «привилегированному» программному обеспечению.

8.4 Стек

8.4.1 Процессор использует полный убывающий стек. Это означает, что указатель стека указывает на адрес последнего положенного в стек элемента в память. Когда процессор записывает новый элемент в стек, он уменьшает значение указателя стека, затем записывает элемент в новое местоположение в памяти. В процессоре реализовано два стека: основной стек (main) и стек процесса (process). Указатель для каждого сохраняется в независимом регистре.

В режиме потока (Thread) регистр управления CONTROL определяет, какой из стеков (основной стек или стек процесса) использует процессор. В режиме обработчика (Handler) процессор всегда использует основной стек (main). В таблице 8.1 представлены опции для операций процессора.

Таблица 8.1 – Соотношение режима процессора, уровня привилегий выполнения программного обеспечения, опций использования стека

Режим процессора	**Использование**	**Уровень привилегии для программного обеспечения**	**Используемый стек**
Режим потока (Thread)	прикладная программа	привилегированный или непривилегированный	основной стек (main) или стек процесса (process)
Режим обработчика (Handler)	обработчик исключений	всегда привилегированный	основной стек (main)

8.5 Регистры ядра

8.5.1 На рисунке 8.2 представлены регистры ядра процессора. В таблице 8.2 представлены установки регистров ядра.

Таблица 8.2 – Установки регистров ядра

Имя	**Тип** [1)]	**Требуемые привилегии** [2)]	**Значение после сброса**	**Описание**
R0-R12	RW	Любой	Неизвестно	Регистры общего назначения
MSP	RW	Привилегированный	См. описание	Указатель стека main
PSP	RW	Любой	Неизвестно	Указатель стека process
LR	RW	Любой	0xFFFFFFFF	Регистр связи Link register
PC	RW	Любой	См. описание	Счетчик команд Program counter
PSR	RW	Привилегированный	0x01000000	Программный регистр состояний Program Status Register
ASPR	RW	Любой	Неизвестно	Программный регистр состояния приложения Application Program Status Register
IPSR	RO	Привилегированный	0x00000000	Программный регистр состояния прерывания Interrupt Program Status Register
EPSR	RO	Привилегированный	0x01000000	Программный регистр состояния выполнения Execution Program Status Register
PRIMASK	RW	Привилегированный	0x00000000	Регистр маски приоритетов Priority Mask Register
FAULTMASK	RW	Привилегированный	0x00000000	Регистр маски сбоев Fault Mask Register
BASEPRI	RW	Привилегированный	0x00000000	Регистр базового приоритета маски
CONTROL	RW	Привилегированный	0x00000000	Регистр управления CONTROL Register

[1)] Определяет режим доступа при исполнении программы в потоковом режиме и в режиме обработки запросов.

[2)] Значение «Любой» обозначает, что доступ к регистру имеет как привилегированное так и непривилегированное программное обеспечение

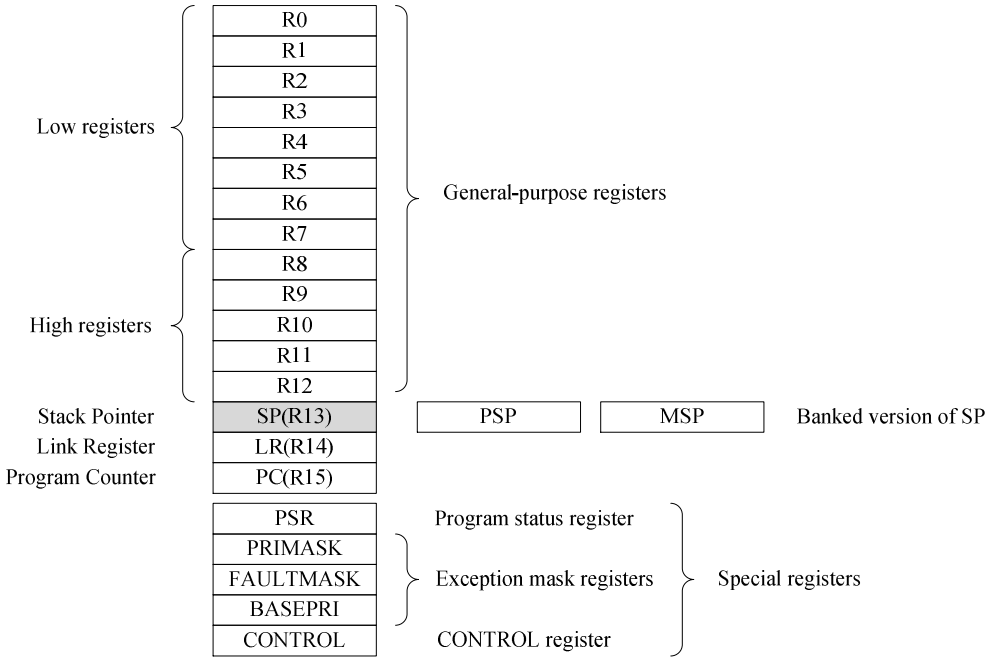

Рисунок 8.2 – Регистры ядра процессора

8.5.1.1 Регистры общего назначения. Регистры R0-R12 являются 32-битными регистрами общего назначения для операций с данными.

8.5.1.2 Указатель стека. Указателем стека (SP) является регистр R13. В режиме потока (Thread), бит 1 регистра управления CONTROL определяет какой указатель стека используется:

0 – регистр SP используется как указатель основного стека MSP. Это значение сразу после сброса;

1 – регистр SP используется как указатель стека процесса PSP.

При сбросе, процессор загружает в MSP значение из адреса 0x00000000.

Примечание:

Регистр R13 (указатель стека) реализован в виде банка из двух регистров (в каждый момент времени доступен только один из регистров).

Они объединены в банк, поэтому в каждый момент времени виден только один из них:

- основной указатель стека (Main Stack Pointer — MSP) — указатель стека, используемый ядром операционной системы и обработчиками исключительных ситуаций.

- указатель стека процесса (Process Stack Pointer — PSP) — указатель стека, используемый прикладной программой.

Два младших бита указателей стека всегда сброшены в 0, т.е. эти указатели всегда выровнены по границе 32-битного слова.

8.5.1.3 Регистр связи. Регистр R14 - регистр связи (LR). Он хранит адрес возврата при работе подпрограмм, вызове функций и исключений (ошибочных ситуаций).

При сбросе, процессор устанавливает в LR значение 0xFFFFFFFF.

8.5.1.4 Счетчик команд. Регистр R15 - счетчик команд (PC). Он содержит адрес выполняемой в данный момент команды.

При сбросе процессор загружает в PC значение из вектора сброса, который находится по адресу 0x00000004.

8.5.1.5 Регистр состояния программы. Регистр состояния программы (PSR) объединяет в себе:

– регистр состояния прикладной программы Application Program Status Register (APSR)

– регистр состояния прерывания Interrupt Program Status Register (IPSR)

– регистр состояния выполнения программы Execution Program Status Register (EPSR).

Эти регистры являются взаимно исключающими битовыми полями в 32-битном PSR.

Доступ к этим регистрам отдельно или совместно к двум или ко всем трем регистрам обеспечивается через использование имени регистра как аргумента для команд MSR или MRS.

Например:

– чтение всех регистров, используя PSR инструкцией MRS;

– запись в биты N, Z, C, V, Q регистра APSR, используя APSR_nzcvq в инструкции MSR.

В таблице 8.3 представлены комбинации и атрибуты доступа к регистру PSR.

Таблица 8.3 – Комбинация доступа к регистру PSR

Регистр	**Тип**	**Комбинация**
PSR	RW[1), 2)]	APSR, EPSR и IPSR
IEPSR	RO	EPSR и IPSR
IAPSR	RW[1)]	APSR и IPSR
EAPSR	RW[2)]	APSR и EPSR

_____

[1)] процессор игнорирует запись в IPSR биты;

[2)] чтение битов EPSR возвращает 0 и процессор игнорирует запись в эти биты.

Регистр состояния прикладкой программы APSR

Регистр APSR содержит текущее состояние флагов, измененных в результате выполнения предыдущих инструкций.

Таблица 8.4 – Регистр APSR

Номер	31	30	29	28	27	26…0
Доступ	R/W	R/W	R/W	R/W	R/W	
Сброс	0	0	0	0	0	
	N	**Z**	**C**	**V**	**Q**	**–**

Таблица 8.5 – Описание бит регистра APSR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31	N	Флаг отрицательного числа 0 – результат операции положительный или нулевой; 1 – результат операции отрицательный.
30	Z	Флаг нуля 0 – результат операции не нулевой; 1 – результат операции нулевой.
29	C	Признак переполнения; признак переноса или заема (единицы старшего разряда при вычитании) 0 – при суммировании не было переноса, либо при вычитании не было заема; 1 – при суммировании был перенос, либо при вычитании был заем.
28	V	Флаг переполнения 0 – в результате операции не было переполнения; 1 – в результате операции было переполнение.
27	Q	Переполнение DSP и флаг насыщения
26…0	–	Зарезервировано

Регистр состояния прерываний IPSR

Регистр IPSR содержит номер типа исключения текущей программы обработки прерываний (ISR).

Таблица 8.6 – Регистр IPSR

Номер	31…9	8…0
Доступ	–	R
Сброс	–	0
	–	**ISR_NUMBER**

Таблица 8.7 – Описание бит регистра IPSR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…9	–	Зарезервировано
8…0	ISR_NUMBER	Номер текущего исключения: 0 – Thread mode, режим потока; 1 – Зарезервировано; 2 – NMI; 3 – HardFault, устойчивая неисправность, отказ (по причине дефекта); 4 – MemManage; 5 – BusFault; 6 – UsageFault; 7-10 – Зарезервировано; 11 – SVCall; 12 – Зарезервировано для отладки; 13 – Зарезервировано; 14 – PendSV; 15 – SysTick; 16 – IRQ0; … 48 –IRQ31;

Регистр состояния выполнения программы EPSR

EPSR содержит бит состояния Thumb инструкции и биты состояния выполнения для:

– команды *If-Then* (IT);

– поля *Interruptible-Continuable Instruction* (ICI) прерванной команды множественного сохранения и считывания.

Таблица 8.8 – Регистр EPSR

Номер	31…27	26…25	24	23…16	15…10	9…0
Доступ	–	R		–		–
Сброс	–	0		–		–
	–	**ICI/IT**	**T**	–	**ICI/IT**	–

Таблица 8.9 – Описание бит регистра EPSR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…27	–	Зарезервировано
26…25	ICI/IT	ICI: Биты инструкции Interruptible-continuable IT: Показывает биты состояния выполнения (программы) для инструкции IT
24	T	Бит состояния Thumb
23…16	–	Зарезервировано
15…10		ICI: Биты инструкции Interruptible-continuable IT: Показывает биты состояния выполнения (программы) для инструкции IT
9…0	–	Зарезервировано

Попытка считать EPSR напрямую при помощи прикладного программного обеспечения, используя инструкцию MSR, всегда возвращает 0. Попытки записи EPSR, используя инструкцию MSR в прикладном программном обеспечении, игнорируются.

8.5.1.6 Регистры маскирования прерываний. Регистры маскирования прерываний блокируют обработку прерываний процессором. Прерывания блокируются, когда они могут оказать воздействие на задачи критичные по времени.

Для доступа к регистрам маскирования прерываний используются команды MSR и MRS, или команда CPS для изменения значения PRIMASK или FAULTMASK.

Регистр маски приоритетов PRIMASK

Регистр PRIMASK запрещает все прерывания с переконфигурируемым приоритетом.

Таблица 8.10 – Регистр PRIMASK

Номер	31…1	0
Доступ	U	R/W
Сброс	0	0
	–	**PRIMASK**

Таблица 8.11 – Описание бит регистра PRIMASK

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…1	–	Зарезервировано
0	PRIMASK	0 – не действует 1 – предотвращает активизацию всех исключений с переконфигурируемым приоритетом

Регистр маскирования неисправностей FAULTMASK

Регистр FAULTMASK запрещает все исключения кроме немаскируемых прерываний *Non-Maskable Interrupt* (NMI).

Таблица 8.12 – Регистр FAULTMASK

Номер	31…1	0
Доступ	U	R/W
Сброс	0	0
	–	**FAULTMASK**

Таблица 8.13 – Описание бит регистра FAULTMASK

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…1	-	Зарезервировано
0	FAULTMASK	0 – не действует 1 – запрещает все прерывания, кроме NMI

Процессор очищает бит FAULTMASK в 0 по выходу из любой программы обработки прерываний, кроме выхода из программы обработки NMI.

Регистр маскирования базового приоритета

Регистр BASEPRI определяет минимальный приоритет для обработки исключений. Когда BASEPRI установлен в ненулевое значение, он запрещает все прерывания, имеющие уровень приоритета равный или меньший заданному в BASEPRI.

Таблица 8.14 – Регистр BASEPRI

Номер	31…8	7…0
Доступ	U	R/W
Сброс	0	0
	–	**BASEPRI**

Таблица 8.15 – Описание бит регистра BASEPRI

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…8	–	Зарезервировано
7…0	BASEPRI	Биты маски приоритета: 0x00 = Не действует; «Ненулевой» - определяет базовый приоритет для обработки исключений Процессор не обрабатывает исключения, значение приоритета которых больше или равно BASEPRI

8.5.1.7 Регистры маскирования прерываний. Регистр управления CONTROL управляет использованием стеков и уровнем привилегий для исполняемого ПО, когда процессор находится в режиме потока и указывает на активность FPU.

Таблица 8.16 – Регистр CONTROL

Номер	31…2	2	1	0
Доступ	U	R/W	R/W	R/W
Сброс	0	0	0	0
	–	**FPCA**	**SPSEL**	**nPRIV**

Таблица 8.17 – Описание бит регистра CONTROL

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31-2	–	Зарезервировано
2	FPCA	Указывает, является ли текущий контекст операции с плавающей точкой активным: 0 = не активен 1 = активен
1	SPSEL	Определяет текущий активный указатель стека: В режиме обработки запросов этот бит считывается как 0 и игнорируется при записи. Процессор обновляет этот бит автоматически при возврате из исключения. 0 = MSP является текущим указателем стека 1 = PSP является текущим указателем стека
0	nPRIV	Определяет в режиме потока уровень привилегии 0 = «Привилегированный» 1 = «Непривилегированный»

Режим обработчика всегда использует MSP, поэтому процессор игнорирует явную запись в бит активного указателя стека регистра управления в режиме обработчика. Механизмы входа и вывода из обработки исключений автоматически обновляют регистр управления.

При работе с операционными средами рекомендуется, чтобы потоки, запущенные в режиме потока, использовали стек процесса, а ядро и обработчик исключений использовали основной стек.

По умолчанию режим потока использует MSP. Для переключения указателя стека, используемого в режиме потока в PSP необходимо одно из двух:

– использовать команду MSR, чтобы установить бит активного указателя стека в 1;

– выполнить возврат из исключения в режиме потока с соответствующим значением EXC_RETURN.

Примечание:

Когда изменяется указатель стека, программное обеспечение должно использовать команду ISB непосредственно после команды MSR. Это гарантирует что команды, следующие после команды ISB, исполняются, используя новый указатель стека.

8.6 Исключения и прерывания

8.6.1 Процессор поддерживает прерывания и системные исключения. Процессор и NVIC назначают приоритеты и обрабатывают все исключения. Исключения изменяют нормальный порядок исполнения программы. Процессор использует режим обработчика, чтобы обработать все исключения кроме «сброса». Регистры NVIC управляют обработкой прерываний.

8.7 Типы данных

8.7.1 Процессор поддерживает следующие типы данных:

– 32-битные слова;

– 16-битные полуслова (любая половина машинного слова);

– 8-битные байты;

Процессор управляет всей памятью данных доступной как с прямым, так и обратным порядком байтов. Обращения к памяти команд и *Private Peripheral Bus* (PPB) всегда выполняется в прямом порядке (little-endian).

9 Система команд

9.1 Поддерживаемые команды представлены в таблице 9.1.

В таблице используются следующие обозначения:

в угловых скобках, <>, приведены альтернативные формы операндов;

– в фигурных скобках {} приведены необязательные операнды;

– столбец Операнды не является полным;

– операнд Op2 может быть как регистром, так и константой;

– большинство инструкций может содержать суффикс кода условного выполнения.

Более подробная информация об инструкциях и операндах приведена в описании инструкций.

Таблица 9.1 – Система команд

Мнемоника	Операнды	Краткое описание	Флаги
ADC, ADCS	{Rd,} Rn, Op2	Сложение с переносом	N,Z,C,V
ADD, ADDS	{Rd,} Rn, Op2	Сложение	N,Z,C,V
ADD, ADDW	{Rd,} Rn, #imm12	Сложение	—
ADR	Rd, label	Загрузка PC-относительного адреса	—
AND, ANDS	{Rd,} Rn, Op2	Логическое И	N,Z,C
ASR, ASRS	Rd, Rm, <Rs|#n>	Арифметический сдвиг вправо	N,Z,C
B	label	Ветвление (переход)	—
BFC	Rd, #lsb, #width	Очистить битовое поле	—
BFI	Rd, Rn, #lsb, #width	Вставить битовое поле	—
BIC, BICS	{Rd,} Rn, Op2	Очистить бит	N,Z,C
BKPT	#imm	Контрольная точка	—
BL	label	Переход со ссылкой	—
BLX	Rm	Косвенный переход со ссылкой	—
BX	Rm	Косвенный переход	—
CBNZ	Rn, label	Сравнить и перейти, если не ноль	—
CBZ	Rn, label	Сравнить и перейти, если ноль	—
CLREX	—	Сброс монопольного доступа	—
CLZ	Rd, Rm Count	Ведущие Нули	—
CMN	Rn, Op2	Сравнить Отрицательные	N,Z,C,V
CMP	Rn, Op2	Сравнить	N,Z,C,V
CPSID	i	Изменить Состояние Процессора, Отключить прерывания	—
CPSIE	i	Изменить Состояние Процессора, Включить прерывания	—
DMB	—	Граница (Барьер) Памяти Данных	—
DSB	—	Граница (Барьер) Синхронизации Данных	—
EOR, EORS	{Rd,} Rn, Op2	Исключающее ИЛИ	N,Z,C
ISB	—	Граница (Барьер) Синхронизации Инструкций	—
IT	—	Условный блок If-Then	—
LDM	Rn{!}, reglist	Загрузка нескольких регистров, увеличить после	—
LDMDB, LDMEA	Rn{!}, reglist	Загрузка нескольких регистров, уменьшить до	—
LDMFD, LDMIA	Rn{!}, reglist	Загрузка нескольких регистров, увеличить после	—
LDR	Rt, [Rn, #offset]	Загрузка регистр словом	—
LDRB, LDRBT	Rt, [Rn, #offset]	Загрузка регистр байтом	—
LDRD	Rt, Rt2, [Rn, #offset]	Загрузка регистр двумя байтами	—
LDREX	Rt, [Rn, #offset]	Монопольная загрузка регистра	—
LDREXB	Rt, [Rn]	Монопольная загрузка регистра байтом	—

Продолжение таблицы 9.1

Мнемоника	Операнды	Краткое описание	Флаги
LDREXH	Rt, [Rn]	Монопольная загрузка регистра полусловом	—
LDRH, LDRHT	Rt, [Rn, #offset]	Загрузка регистр полусловом	—
LDRSB, LDRSBT	Rt, [Rn, #offset]	Загрузка регистр байтом со знаком	—
LDRSH, LDRSHT	Rt, [Rn, #offset]	Загрузка регистр полусловом со знаком	—
LDRT	Rt, [Rn, #offset]	Загрузка регистр словом	—
LSL, LSLS	Rd, Rm, <Rs|#n>	Логический сдвиг влево	N,Z,C
LSR, LSRS	Rd, Rm, <Rs|#n>	Логический сдвиг вправо	N,Z,C
MLA	Rd, Rn, Rm, Ra	Умножение со сложением, 32-битный результат	—
MLS	Rd, Rn, Rm, Ra	Умножение с вычитанием, 32-битный результат	—
MOV, MOVS	Rd, Op2	Пересылка	N,Z,C
MOVT	Rd, #imm16	Пересылка старшего полуслова	—
MOVW, MOV	Rd, #imm16	Пересылка 16-разрядной константы	N,Z,C
MRS	Rd, spec_reg	Пересылка из специального регистра в общий регистр	—
MSR	spec_reg, Rm	Пересылка из общего регистра в специальный регистр	N,Z,C,V
MUL, MULS	{Rd,} Rn, Rm	Умножение, 32-разрядный результат	N,Z
MVN, MVNS	Rd, Op2	Пересылка с инверсией	N,Z,C
NOP	—	Нет операции	-
ORN, ORNS	{Rd,} Rn, Op2	Логическое ИЛИ с инверсией (OR NOT)	N,Z,C
ORR, ORRS	{Rd,} Rn, Op2	Логическое ИЛИ	N,Z,C
PKHTB, PKHBT	{Rd,} Rn, Rm, Op2	Упаковать Полслова	—
POP	reglist	Извлечение регистра из стека	—
PUSH	reglist	Загрузка регистра в стек	—
QADD	{Rd,} Rn, Rm	Знаковое сложение с насыщением 32-разрядное	Q
QADD16	{Rd,} Rn, Rm	Знаковое сложение с насыщением 16-разрядное	—
QADD8	{Rd,} Rn, Rm	Знаковое сложение с насыщением 8-разрядное	—
QASX	{Rd,} Rn, Rm	Сложение и вычитание с насыщением и перестановкой	—
QDADD	{Rd,} Rn, Rm	Удвоение и сложение с насыщением, знаковое	Q
QDSUB	{Rd,} Rn, Rm	Удвоение и вычитание с насыщением, знаковое	Q
QSAX	{Rd,} Rn, Rm	Вычитание и сложение с насыщением и перестановкой	—
QSUB	{Rd,} Rn, Rm	Вычитание с насыщением	Q
QSUB16	{Rd,} Rn, Rm	Вычитание с насыщением 16-разрядное	—
QSUB8	{Rd,} Rn, Rm	Вычитание с насыщением 8-разрядное	—
RBIT	Rd, Rn	Перестановка битов	—
REV	Rd, Rn	Перестановка байтов слова	—
REV16	Rd, Rn	Перестановка байтов в каждом полуслове	—
REVSH	Rd, Rn	Перестановка байтов в младшем полуслове и расширение знака	—
ROR, RORS	Rd, Rm, <Rs|#n>	Циклический сдвиг вправо	N,Z,C

Продолжение таблицы 9.1

Мнемоника	Операнды	Краткое описание	Флаги
RRX, RRXS	Rd, Rm	Расширенный циклический сдвиг вправо	N,Z,C
RSB, RSBS	{Rd,} Rn, Op2	Обратное вычитание	N,Z,C,V
SADD16	{Rd,} Rn, Rm	Знаковое сложение 16-разрядных целых чисел	GE
SADD8	{Rd,} Rn, Rm	Знаковое сложение 8	GE
SASX	{Rd,} Rn, Rm	Знаковое сложение и вычитание с перестановкой	GE
SBC, SBCS	{Rd,} Rn, Op2	Вычитание с переносом	N,Z,C,V
SBFX	Rd, Rn, #lsb, #width	Извлечение знакового битового поля	—
SDIV	{Rd,} Rn, Rm	Знаковое деление	—
SEL	{Rd,} Rn, Rm	Выбор байта	—
SEV	—	Послать событие	—
SHADD16	{Rd,} Rn, Rm	Полусумма знаковых 16-битных целых чисел	—
SHADD8	{Rd,} Rn, Rm	Полусумма знаковых 8-битных целых чисел	—
SHASX	{Rd,} Rn, Rm	Сложение и вычитание с перемещением и делением пополам	—
SHSAX	{Rd,} Rn, Rm	Вычитание и сложение с перемещением и делением пополам	—
SHSUB16	{Rd,} Rn, Rm	Полуразность знаковых 16-битных целых чисел	—
SHSUB8	{Rd,} Rn, Rm	Полуразность знаковых 8-битных целых чисел	—
SMLABB, SMLABT, SMLATB, SMLATT	Rd, Rn, Rm, Ra	Знаковое умножение с сложением (полуслов)	Q
SMLAD, SMLADX	Rd, Rn, Rm, Ra	Знаковое умножение с двойным сложением (и перестановкой)	Q
SMLAL	RdLo, RdHi, Rn, Rm	Знаковое умножение с накоплением (32 x 32 + 64), 64-разрядный результат	—
SMLALBB, SMLALBT, SMLALTB, SMLALTT	RdLo, RdHi, Rn, Rm	Знаковое умножение с длинным сложением (полуслов)	—
SMLALD, SMLALDX	RdLo, RdHi, Rn, Rm	Знаковое умножение с длинным двойным сложением (и перестановкой)	—
SMLAWB, SMLAWT	Rd, Rn, Rm, Ra	Знаковое умножение со сложением (слово+полуслово)	Q
SMLSD	Rd, Rn, Rm, Ra	Знаковое вычитание двух произведений полуслов	Q
SMLSLD	RdLo, RdHi, Rn, Rm	Знаковое вычитание двух произведений полуслов (64-битный результат)	
SMMLA	Rd, Rn, Rm, Ra	Сложение со знаковым старшим значимым словом произведения, 32 + (32 x 32) = 32 (старшие разряды)	—
SMMLS, SMMLR	Rd, Rn, Rm, Ra	Знаковое Умножение с Вычитанием Наиболее значимого слова	—
SMMUL, SMMULR	{Rd,} Rn, Rm	Вычитание со знаковым старшим значимым словом произведения. 32 -(32 x 32) = 32 (старшие разряды)	—
SMUAD	{Rd,} Rn, Rm	Сложение результатов произведений полуслов операндов (16 x 16) + (16 x 16) = 32	Q
SMULBB, SMULBT, SMULTB, SMULTT	{Rd,} Rn, Rm	Знаковое перемножение полуслов 16 x 16 = 32	—

Продолжение таблицы 9.1

Мнемоника	Операнды	Краткое описание	Флаги
SMULL	RdLo, RdHi, Rn, Rm	Знаковое умножение (32 x 32), 64-разрядный результат	—
SMULWB, SMULWT	{Rd,} Rn, Rm	Знаковое перемножение слова с полусловом	—
SMUSD, SMUSDX	{Rd,} Rn, Rm	Вычитание результатов произведений полуслов операндов (16 x 16) - (16 x 16) = 32	—
SSAT	Rd, #n, Rm {,shift #s}	Насыщение в знаковом диапазоне	Q
SSAT16	Rd, #n, Rm	Знаковое насыщение до любой позиции бита для двух полуслов	Q
SSAX	{Rd,} Rn, Rm	Знаковое вычитание и сложение с Перестановкой	GE
SSUB16	{Rd,} Rn, Rm	Знаковое вычитание 16-разрядных целых чисел	—
SSUB8	{Rd,} Rn, Rm	Знаковое вычитание 8 разрядных целых чисел	—
STM	Rn{!}, reglist	Сохранение нескольких регистров, увеличить после	—
STMDB, STMEA	Rn{!}, reglist	Сохранение нескольких регистров, уменьшить до	—
STMFD, STMIA	Rn{!}, reglist	Сохранение нескольких регистров, увеличить после	—
STR	Rt, [Rn, #offset]	Сохранение регистра, слово	—
STRB, STRBT	Rt, [Rn, #offset]	Сохранение регистра, байт	—
STRD	Rt, Rt2, [Rn, #offset]	Сохранение регистра, два слова	—
STREX	Rd, Rt, [Rn, #offset]	Монопольное сохранение регистра	—
STREXB	Rd, Rt, [Rn]	Монопольное сохранение регистра, байт	—
STREXH	Rd, Rt, [Rn]	Монопольное сохранение регистра, полуслово	—
STRH, STRHT	Rt, [Rn, #offset]	Сохранение регистра, полуслова	—
STRT	Rt, [Rn, #offset]	Сохранение регистра, слово	—
SUB, SUBS	{Rd,} Rn, Op2	Вычитание	N,Z,C,V
SUB, SUBW	{Rd,} Rn, #imm12	Вычитание	—
SVC	#imm	Вызов супервизора	—
SXTAB	{Rd,} Rn, Rm,{,ROR #}	Расширение 8 разрядов до 32 и сложение	—
SXTAB16	{Rd,} Rn, Rm,{,ROR #}	Сдвоенное расширение 8 разрядов до 16 и сложение	—
SXTAH	{Rd,} Rn, Rm,{,ROR #}	Расширение 16 разрядов до 32 и сложение	—
SXTB16	{Rd,} Rm {,ROR #n}	Двойное расширение 8 разрядов до 16	—
SXTB	{Rd,} Rm {,ROR #n}	Расширение байта знаком	—
SXTH	{Rd,} Rm {,ROR #n}	Расширение полслова знаком	—
TBB	[Rn, Rm]	Табличный переход байта	—
TBH	[Rn, Rm, LSL #1]	Табличный переход полуслова	—
TEQ	Rn, Op2	Проверка на равенство	N,Z,C
TST	Rn, Op2	Проверка битов	N,Z,C
UADD16	{Rd,} Rn, Rm	Беззнаковое сложение 16-разрядных целых чисел	GE
UADD8	{Rd,} Rn, Rm	Беззнаковое сложение 8-разрядных целых чисел	GE
USAX	{Rd,} Rn, Rm	Беззнаковое сложение и вычитание с перестановкой	GE

Продолжение таблицы 9.1

Мнемоника	Операнды	Краткое описание	Флаги
UHADD16	{Rd,} Rn, Rm	Полусумма беззнаковых 16-битных целых чисел	—
UHADD8	{Rd,} Rn, Rm	Полусумма беззнаковых 8-битных целых чисел	—
UHASX	{Rd,} Rn, Rm	Беззнаковое сложение и вычитание с перестановкой и делением пополам.	—
UHSAX	{Rd,} Rn, Rm	Беззнаковое вычитание и сложение с перестановкой и делением пополам.	—
UHSUB16	{Rd,} Rn, Rm	Полуразность беззнаковых 16-разрядных целых чисел	—
UHSUB8	{Rd,} Rn, Rm	Полуразность беззнаковых 8-разрядных целых чисел	—
UBFX	Rd, Rn, #lsb, #width	Беззнаковое извлечение битового поля	—
UDIV	{Rd,} Rn, Rm	Беззнаковое деление	—
UMAAL	RdLo, RdHi, Rn, Rm	Беззнаковое длинное умножение с двойным сложением (32 x 32 +32+32), 64-разрядный результат	—
UMLAL	RdLo, RdHi, Rn, Rm	Беззнаковое умножение со сложением (32 x 32 + 64), 64-разрядный результат	—
UMULL	RdLo, RdHi, Rn, Rm	Беззнаковое умножение (32 x 32), 64-разрядный результат	—
UQADD16	{Rd,} Rn, Rm	Беззнаковые сложение 16-разрядных целых с насыщением	—
UQADD8	{Rd,} Rn, Rm	Беззнаковые сложение 8-разрядных целых с насыщением	—
UQASX	{Rd,} Rn, Rm	Сложение и вычитание с перестановкой и насыщением, беззнаковые	—
UQSAX	{Rd,} Rn, Rm	Вычитание и сложение с перестановкой и насыщением, беззнаковые	—
UQSUB16	{Rd,} Rn, Rm	Беззнаковые вычитание 16-разрядных целых с насыщением	—
UQSUB8	{Rd,} Rn, Rm	Беззнаковые вычитание 8-разрядных целых с насыщением	—
USAD8	{Rd,} Rn, Rm	Безнаковоая сумма абсолютных разностей 8-разрядных целых	—
USADA8	{Rd,} Rn, Rm, Ra	Безнаковоая сумма абсолютных разностей 8-разрядных целых с дополнительным сложением	—
USAT	Rd, #n, Rm {,shift #s}	Насыщение в беззнаковом диапазоне	Q
USAT16	Rd, #n, Rm	Беззнаковое насыщение до любой позиции бита для двух полуслов	Q
UASX	{Rd,} Rn, Rm	Беззнаковое сложение и вычитание с перестановкой	GE
USUB16	{Rd,} Rn, Rm	Беззнаковое вычитание 16-разрядных целых чисел	GE
USUB8	{Rd,} Rn, Rm	Беззнаковое вычитание 8-разрядных целых чисел	GE
UXTAB	{Rd,} Rn, Rm,{,ROR #}	Циклический сдвиг, расширение 8 разрядов до 32 и сложение	—
UXTAB16	{Rd,} Rn, Rm,{,ROR #}	Циклический сдвиг, сдвоенное расширение 8 разрядов до 16 и сложение	—

Продолжение таблицы 9.1

Мнемоника	Операнды	Краткое описание	Флаги
UXTAH	{Rd,} Rn, Rm,{,ROR #}	Циклический сдвиг, беззнаковое расширение и сложение полуслова	—
UXTB	{Rd,} Rm {,ROR #n}	Расширение нулем байта	—
UXTB16	{Rd,} Rm {,ROR #n}	Беззнаковое расширение байта, 16-разрядное	—
UXTH	{Rd,} Rm {,ROR #n}	Расширение нулем полуслова	—
VABS.F32	Sd, Sm	Модуль числа с плавающей точкой	—
VADD.F32	{Sd,} Sn, Sm	Сложение с плавающей точкой	—
VCMP.F32	Sd, <Sm | #0.0>	Сравнение двух регистров с плавающей точкой или регистра с числом с плавающей точкой и нуля	FPSCR
VCMPE.F32	Sd, <Sm | #0.0>	Сравнение двух регистров с числами с плавающей точкой или одного регистра с плавающей точкой и нуля с проверкой на недопустимость операции	FPSCR
VCVT.S32.F32	Sd, Sm	Преобразование числа с плавающей точкой в целое	—
VCVT.S16.F32	Sd, Sd, #fbits	Преобразование числа с плавающей точкой и в число с фиксированной точкой	—
VCVTR.S32.F32	Sd, Sm	Преобразование числа с плавающей точкой в целое с округлением	—
VCVT<B|H>.F32.F16	Sd, Sm	Преобразование значения с половинной точностью в значение с одинарной точностью	—
VCVTT<B|T>.F32.F16	Sd, Sm	Преобразование значения с одинарной точностью в значение с половинной точностью	—
VDIV.F32	{Sd,} Sn, Sm	Деление с плавающей точкой	—
VFMA.F32	{Sd,} Sn, Sm	Совмещенное умножение с плавающей точкой с накоплением	—
VFNMA.F32	{Sd,} Sn, Sm	Совмещенное умножение чисел с плавающей точкой с накоплением и инверсией	—
VFMS.F32	{Sd,} Sn, Sm	Совмещенное умножение с плавающей точкой с вычитанием	—
VFNMS.F32	{Sd,} Sn, Sm	Совмещенное умножение с плавающей точкой с вычитанием и инверсией	—
VLDM.F<32|64>	Rn{!}, list	Загрузка нескольких регистров с плавающей точкой	—
VLDR.F<32|64>	<Dd|Sd>, [Rn]	Загрузка регистра расширения из памяти	—
VLMA.F32	{Sd,} Sn, Sm	Умножение чисел с плавающей точкой с накоплением	—
VLMS.F32	{Sd,} Sn, Sm	Умножение чисел с плавающей точкой с вычитанием	—
VMOV.F32	Sd, #imm	Переместить непосредственно число с плавающей точкой в регистр	—
VMOV	Sd, Sm	Переместить регистр с плавающей точкой в регистр	—
VMOV	Sn, Rt	Скопировать регистр ядра ARM в регистр с плавающей точкой с одинарной точностью	—

Продолжение таблицы 9.1

Мнемоника	Операнды	Краткое описание	Флаги
VMOV	Sm, Sm1, Rt, Rt2	Скопировать два регистра ядра ARM в два регистра с плавающей точкой с одинарной точностью	—
VMOV	Dd[x], Rt	Скопировать регистр ядра ARM в скаляр	—
VMOV	Rt, Dn[x]	Скопировать скаляр в регистр ядра ARM	—
VMRS	Rt, FPSCR	Переместить FPSCR в регистр ядра ARM или APSR	N,Z,C,V
VMSR	FPSCR, Rt	Переместить из регистра ядра ARM в FPSCR	FPSCR
VMUL.F32	{Sd,} Sn, Sm	Умножение чисел с плавающей точкой	—
VNEG.F32	Sd, Sm	Инверсия числа с плавающей точкой	—
VNMLA.F32	Sd, Sn, Sm	Умножение и сложение чисел с плавающей точкой	—
VNMLS.F32	Sd, Sn, Sm	Умножение и вычитание чисел с плавающей точкой	—
VNMUL	{Sd,} Sn, Sm	Умножение чисел с плавающей точкой	—
VPOP	List	Извлечение из стека регистров расширения	—
VPUSH	List	Помещение в стек регистров расширения	—
VSQRT.F32	Sd, Sm	Вычисление квадратного корня из числа с плавающей точкой	—
VSTM	Rn{!}, list	Сохранение нескольких регистров с числами с плавающей точкой	—
VSTR.F<32|64>	Sd, [Rn]	Сохранение регистра расширения в память	—
VSUB.F<32|64>	{Sd,} Sn, Sm	Вычитание чисел с плавающей точкой	—
WFE	—	Ожидание события	—
WFI	—	Ожидание прерывания	—

9.1 Функции CMSIS

Стандарт ANSI языка Си не позволяет получить доступ к некоторым командам Cortex-M4 напрямую. Этот раздел описывает встроенные функции, которые могут формировать такие команды. Если компилятор Си не поддерживает подходящую внутреннюю функцию, для доступа к некоторым командам необходимо использовать ассемблерные вставки.

В CMSIS предусмотрены внутренние функции, представленные в таблице 9.2

Таблица 9.2 – Функции CMSIS для формирования некоторых команд Cortex-M4

Инструкция	Функция CMSIS
CPSIE I	void __enable_irq(void)
CPSID I	void __disable_irq(void)
CPSIE F	void __enable_fault_irq(void)
CPSID F	void __disable_fault_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
REV	uint32_t __REV(uint32_t int value)
REV16	uint32_t __REV16(uint32_t int value)
REVSH	uint32_t __REVSH(uint32_t int value)
RBIT	uint32_t __RBIT(uint32_t int value)
SEV	void __SEV(void)
WFE	void __WFE(void)
WFI	void __WFI(void)

Кроме этого, CMSIS предоставляет несколько функций, представленных в таблице 9.3, для доступа к специальным регистрам с помощью команд MRS и MSR.

Таблица 9.3 – Функции CMSIS для доступа к специальным регистрам

Специальный регистр	Доступ	Функция CMSIS
PRIMASK	Чтение	uint32_t __get_PRIMASK (void)
	Запись	void __set_PRIMASK (uint32_t value)
FAULTMASK	Чтение	uint32_t __get_FAULTMASK (void)
	Запись	void __set_FAULTMASK (uint32_t value)
BASEPRI	Чтение	uint32_t __get_BASEPRI (void)
	Запись	void __set_BASEPRI (uint32_t value)
CONTROL	Чтение	uint32_t __get_CONTROL (void)
	Запись	void __set_CONTROL (uint32_t value)
MSP	Чтение	uint32_t __get_MSP (void)
	Запись	void __set_MSP (uint32_t TopOfMainStack)
PSP	Чтение	uint32_t __get_PSP (void)
	Запись	void __set_PSP (uint32_t TopOfProcStack)

9.2 Краткое описание команд

9.2.1 Операнды

9.2.1.1 Операндом команды может быть регистр, константа, или другой специальный параметр. Процессор применяет команду к операндам и, как правило, сохраняет результат в регистре-получателе. В случаях, когда команда предусматривает наличие регистра-получателя, он как парвило определяется перед операндами.

В некоторых командах операнды имеют гибкий формат, т.е. могут быть как регистром, так и константой.

9.2.2 Ограничения при использовании PC и SP

9.2.2.1 Многие команды имеют ограничения на использование счетчика команд (PC) или указателем стека (SP) в качестве регистра-получателя. Более подробная информация содержится в описание команды.

Для правильного выполнения команд бит [0] любого загруженного адреса в PC с помощью команд BX, BLX, LDM, LDR или POP должен быть равен 1, т.к. этот бит указывает на требуемый набор команд, а процессор Cortex-M4 поддерживает только команды из набора Thumb.

9.2.3 Формат второго операнда

Многие команды обработки данных поддерживают гибкий формат второго операнда. Такой операнд в дальнейшем будет обозначаться как Operand2.

Operand2 может быть:

– константой;

– регистром с необязательным параметром сдвига.

9.2.3.1 Константа

Константа в Operand2 определяется в следующем виде:

#constant

где constant может быть:

– любая константа, которая может быть произведена путем смещением 8-разрядного числа влево на любое количество бит в пределах 32-разрядного слова;

– любая константа формы 0x00XY00XY;

– любая константа формы 0xXY00XY00;

– любая константа формы 0xXYXYXYXY.

Во всех вышеописанных случаях X и Y – шестнадцатеричные цифры.

Кроме того, в некоторых командах константа может принимать более широкий диапазон значений. Подробности изложены в описании конкретной команды.

Когда константный операнд Operand2 используется в командах MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST в случае, если константа больше 255 и может быть получена путем сдвига 8-разрядного числа, значение бита [31] константы влияет на значение флага переноса. Эти команды не влияют на флаг переноса, для всех остальных значений Operand2.

9.2.3.2 Замещение команды

Ассемблер может сгенерировать эквивалентную команду в том случае, если была указана константа, не удовлетворяющая требованиям, описанным в 9.2.3.1. Например, ассемблер мог преобразовать команду CMP Rd, #0xFFFFFFFE в эквивалентную команду CMN Rd, #0x2.

9.2.3.3 Регистр с необязательным сдвигом

В данном случае второй операнд записывается в виде:

Rm {, *shift*}

где:

Rm – регистр, содержащий значение для второго операнда;

Shift – необязательный параметр, определяющий сдвиг данных регистра Rm. Он может принимать одно из следующих значений:

– ASR #n - арифметический сдвиг вправо на n битов , $1 \le n \le 32$.

– LSL #n - логический сдвиг влево на n битов, $1 \le n \le 31$.

– LSR #n - логический сдвиг вправо на n битов, $1 \le n \le 32$.

– ROR # - циклический сдвиг вправо на n битов, $1 \le n \le 31$.

– RRX - расширенный циклический сдвиг вправо на один бит.

Если сдвиг не указан, случай эквивалентен заданию сдвига LSL #0. При этом команда использует значение, находящееся в регистре Rm.

При указании операции сдвига она выполняется над содержимым регистра Rm, а итоговое 32-битное значение используется в команде. При этом содержимое регистра Rm не изменяется. При использовании определенных команд определение регистра со сдвигом также изменяет состояние флага переноса.

9.2.4 Операции сдвига

9.2.4.1 Операции сдвига регистра выполняют сдвиг содержимого этого регистра влево или вправо на заданное число бит, называемое длиной сдвига. Сдвиг регистра может осуществляться:

– непосредственно командами ASR, LSR, LSL, ROR, и RRX с записью результата в регистр-получатель;

– при вычислении Operand2 командами, использующими в качестве второго операнда результат сдвига.

Допустимые значения длины сдвига зависят от типа сдвига и от команды. Если длина сдвига равна нулю, то сдвиг не осуществляется. Операции сдвига регистра изменяют состояние флага переноса, за исключением тех случаев, когда заданная длина сдвига равна нулю.

9.2.4.2 ASR. Арифметический сдвиг вправо на n бит перемещает 32-n старших бит регистра Rm на n позиций вправо, то есть на место крайних справа 32-n. Бит[31] исходного значения регистра копируется в n старших бит результата. См.рисунок 9.1.

Операцию ASR #*n* можно использовать для деления содержимого регистра Rm на 2^n с округлением результата к ближайшему меньшему целому.

При использовании команды ASRS, а также в случае, если сдвиг ASR #n используется при вычислении второго операнда команд MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST, флаг переноса принимает значение последнего бита, выдвинутого в результате операции сдвига, т.е. бита [n-1] регистра Rm.

Если n больше или равно 32, то во все биты результата устанавливаются в значение бита[31] регистра Rm;

Если n больше или равно 32 и операция влияет на флаг переноса, то в этот флаг загружается значение бита [31] регистра Rm.

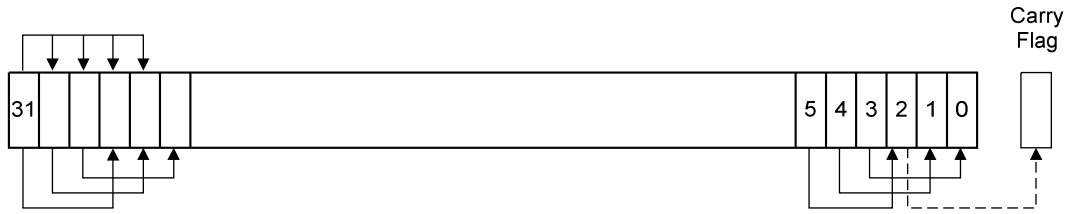

Рисунок 9.1 – Команда ASR #3

9.2.4.3 LSR. Логический сдвиг вправо на n бит перемещает левые 32-n бит регистра Rm на n позиций вправо, на место крайних справа 32-n бит. Старшие n битов обнуляются. См. рисунок 9.2.

Операцию LSR #n можно использовать для деления содержимого регистра Rm на 2^n, если содержимое регистра рассматривается как целое число без знака.

При использовании команды LSRS а также в случае, если сдвиг LSR #n используется при вычислении второго операнда команд MOVS, MVNS,ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST во флаг переноса загружается последний выдвинутый бит регистра Rm, т.е. бит[n-1].

Если n больше или равно 32, тогда все биты результата сбрасываются в 0.

Если n больше или равно 32 и операция влияет на флаг переноса, то он сбрасывается в 0.

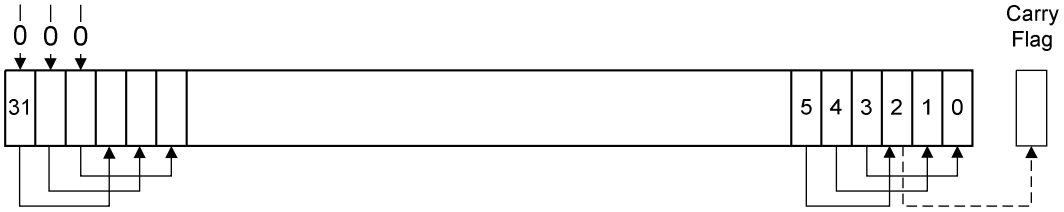

Рисунок 9.2 – Команда LSR #3

9.2.4.4 LSL. Логический сдвиг влево на n бит перемещает правосторонние (младшие) 32-n бит регистра Rm на n позиций влево, то есть на место крайних слева 32-n бит. Правосторонние (младшие) n битов результата обнуляются. См. рисунок 9.3.

Операцию LSL #n можно использовать для умножения содержимого регистра Rm на 2^n, если содержимое регистра рассматривается как целое число со знаком,

представленное в дополнительном коде. При переполнении никаких предупреждений не выдается.

В случае команды LSLS или при использовании команды LSL #n при вычислении второго операнда команд MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST во флаг переноса загружается последний выдвинутый бит регистра Rm, т.е. бит[32-n]. Эта операция не влияет на состояние флага переноса, если n =0 (LSL #0).

Если n больше или равно 32, тогда все биты результата сбрасываются в 0.

Если n больше или равно 33 и операция влияет на флаг переноса, то он сбрасывается в 0.

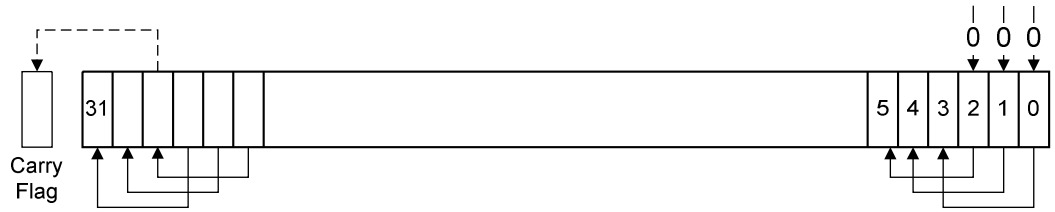

Рисунок 9.3 – Команда LSL #3

9.2.4.5 ROR. Циклический сдвиг вправо на n битов перемещает левые (старшие) 32-n битов регистра Rm на n позиций вправо, то есть на место крайних справа 32-n бит. Одновременно n правосторонних битов регистра копируются в левые n битов результата. См. рисунок 9.4.

Если n равно 32, тогда результат равен содержимому регистра Rm. При этом если операция влияет на флаг переноса, то в него загружается значение бита[31] регистра Rm.

Операция ROR при величине сдвига n больше 32 эквивалентна операции ROR с величиной сдвига n-32.

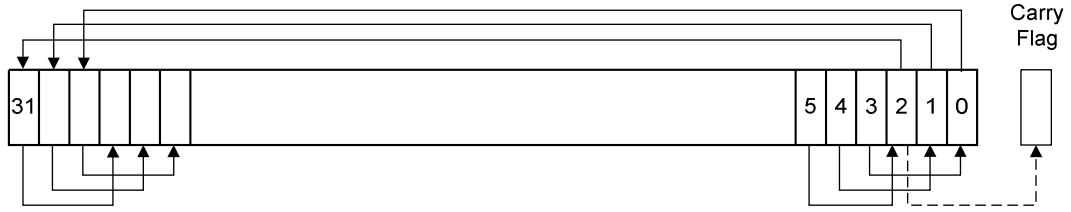

Рисунок 9.4 – Команда ROR #3

9.2.4.6 RRX. Расширенный циклический сдвиг вправо перемещает все биты регистра Rm на одну позицию вправо и копирует содержимое флага переноса в бит[31] результата. См. рисунок 9.5.

При использовании команды RRXS, а также в случае, когда сдвиг RRX #n используется при вычислении второго операнда команд MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST, во флаг переноса загружается бит[0] регистра Rm.

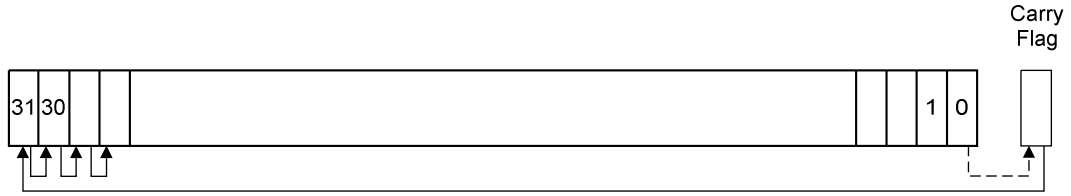

Рисунок 9.5 – Команда RRX

9.2.5 Выравнивание адресов

9.2.5.1 Под выровненным доступом понимают операции, в которых чтение и запись слов, двойных слов и более длинных последовательностей слов осуществляется по адресам, выровненным на границу слова, или же операции, при которых для обращения к полуслову используется адрес, выровненный на границу полуслова. Обращения к байтам выровнены по определению.

Процессор Cortex-M4 поддерживает обращение к не выровненным данным только для следующих команд:

– LDR, LDRT

– LDRH, LDRHT
– LDRSH, LDRSHT
– STR, STRT
– STRH, STRHT

Все остальные команды загрузки и сохранения при обращении к не выровненному адресу генерируют исключение UsageFault, поэтому они должны использоваться только с выровненными адресами.

Обращения к не выровненным данным, как правило, осуществляются медленнее, чем к выровненным данным. Кроме того, некоторые области адресного пространства могут не поддерживать обращение к не выровненным адресам. В связи с этим ARM рекомендует программистам всегда использовать выровненные данные. Для избежания непреднамеренных обращений по не выровненным адресам используется бит UNALIGN_TRP регистра CCR.

9.2.6 Адресация относительно счетчика команд PC

9.2.6.1 В системе команд предусмотрена адресация команды или области данных в виде суммы значения счетчика команд PC плюс/минус численное смещение. Ассемблер автоматически вычисляет требуемое смещение, исходя из адреса метки и адреса текущей команды. Если величина смещения получается слишком большой, то ассемблер выдает сообщение об ошибке.

Примечание:

1. для команд B, BL, CBNZ и CBZ значением PC является адрес текущей команды плюс 4 байта;

2. для всех других команд значением PC является адрес текущей команды плюс 4 байтов, при этом бит[1] результата сбрасывается в 0 для выравнивания адреса по границе слова;

3. ассемблер может поддерживать другие формы записи, как например «метка плюс/минус число» или запись вида [PC, #number].

9.2.7 Условное выполнение

9.2.7.1 Большинство команд обработки данных могут изменять флаги условий в регистре состояния прикладной программы (APSR) в зависимости от результата операции.

Одни команды влияют на все флаги, другие – только на некоторые из них. Если команда не влияет на флаг, то его состояние остается неизменным. Информация, на какие флаги влияет та или иная команда, указана в описании команды.

Команда может быть выполнена условно в соответствии со значением флагов, измененных другой командой, либо в одном из двух случаев:

– сразу же после команды, изменившей флаги;

– после нескольких промежуточных команд, не влияющих на состояние флагов условий.

Условное выполнение реализуется командами условного перехода или добавлением к мнемонике обычных команд суффиксов, определяющих условие выполнения команды. Эти суффиксы перечислены в таблице . Код условия, определяемый суффиксом, позволяет процессору проверять соответствие состояния флагов заданному условию. Если код условия условно выполняемой команды не соответствует состоянию флагов, то команда:

– не выполняется;

– ничего не сохраняет в регистре-приёмнике;

– не воздействует ни на какие флаги;

– не генерирует никаких исключительных ситуаций.

9.2.7.2 Флаги условия. В регистре APSR содержатся следующие флаги условия:

– **N** - устанавливается в 1, если результат последней операции был отрицательный, иначе сбрасывается в 0.

– **Z** - устанавливается в 1, если результат последней операции был равен нулю, иначе сбрасывается в 0.

– **C** - устанавливается в 1, если в результате последней операции произошел перенос, иначе сбрасывается в 0.

– **V** - Устанавливается в 1, если в результате последней операции произошло переполнение, иначе сбрасывается в 0.

Перенос возникает в следующих случаях:

– если результат сложения больше или равен 2^{32};

– если результат вычитания положительный или равен нулю;

– в результате работы внутренней логики циклического сдвига при выполнении команд пересылки данных или команд логических операций.

Переполнение возникает в случаях, если результат сложения, вычитания или сравнения больше или равен 2^{31}, либо меньше -2^{31}.

Операции сравнения полностью эквивалентны командам вычитания (в случае CMP) или сложения (в случае CMN), за исключением того, что результат операции не сохраняется. Для более подробной информации обратитесь к описанию команд.

Примечание - Большинство команд изменяют флаги состояния только в том случае, если в мнемонике команды указан суффикс S.

9.2.7.3 Суффиксы условного выполнения. Команды, допускающие условное выполнение, имеют необязательное поле кода условия, представленного в синтаксических описаниях как {cond}. Условное выполнение требует предварительного выполнения команды IT. Команда, содержащая код условия, выполняется только в том случае, если состояние флагов условий регистра APSR соответствует заданному условию. Все допустимые суффиксы условия выполнения приведены в таблице 9.4.

Условные команды рекомендуется применять для уменьшения числа команд переходов в коде программы.

Кроме того, в таблице 9.4 показана взаимосвязь между суффиксами условия выполнения и флагами N, Z, C и V.

Таблица 9.4 – Суффиксы условного выполнения

Суффиксы	Флаги	Значение
EQ	Z = 1	Равно
NE	Z = 0	Не равно
CS или HS	C = 1	Перенос/выше или то же (беззнаковое)
CC или LO	C = 0	Нет переноса/ниже (беззнаковое)
MI	N = 1	Минус/отрицательное значение
PL	N = 0	Плюс/положительное значение
VS	V = 1	Переполнение
VC	V = 0	Нет переполнения
HI	C = 1 и Z = 0	Выше (беззнаковое)
LS	C = 0 или Z = 1	Ниже или то же (беззнаковое)
GE	N = V	Больше или равно (знаковое)
LT	N != V	Меньше (знаковое)
GT	Z = 0 и N = V	Больше (знаковое)
LE	Z = 1 и N != V	Меньше или равно (знаковое)
AL	Может принимать любое значение	Всегда. Безусловное выполнение

Пример: Вычисление абсолютного значения.

```
MOVS    R0, R1          ; R0 = R1, изменяет состояние флагов
IT      MI              ; пропускает следующую команду, если
                        ; значение больше или равно 0 или
                        ; положительное
RSBMI   R0, R0, #0      ; Если значение меньше 0, то R0 = -R0
```

Пример: сравнение и изменение значения регистра.

```
CMP     R0, R1          ; Сравниваются R0 и R1.
```

```
                    ; Инструкция изменяет состояние флагов
ITT     GT          ; Пропуск следующих двух команд, если условие
                    ; GT не выполняется
CMPGT   R2, R3      ; Если 'больше', сравниваем R2 и R3,
                    ; Устанавливаем флаги
MOVGT   R4, R5      ; Если все еще 'больше', копируем R4 = R5
```

9.2.8 Выбор разрядности команды

9.2.8.1 Многие команды могут формировать как 16-битный, так и 32-битный машинный код в зависимости от используемых операндов и регистра-приёмника. Для некоторых из этих команд можно явно задать разрядность с помощью специального суффикса. Суффикс .W указывает на необходимость генерации 32-битного кода. Суффикс .N указывает на необходимость генерации 16-битного кода.

Если ассемблер не сможет сформировать машинный код заданной разрядности, то он выдаст сообщение об ошибке.

Иногда явное указание суффикса .W необходимо, например, если операнд является меткой или константой, как в случае с командами перехода. Эта необходимость обусловлена тем, что ассемблер может оказаться не в состоянии автоматически сгенерировать машинный код требуемой разрядности.

При использовании суффикса, определяющего разрядность команды, он помещается непосредственно после мнемоники команды и суффикса условия выполнения, при наличии последнего.

Пример: запись команды с использованием суффикса разрядности.

```
BCS.W    label      ; генерирует 32-битный код для
                    ; короткого перехода
ADDS.W R0, R0, R1   ; генерирует 32-битный код, даже если та же самая
                    ; операция может быть выполнена посредством
                    ; 16-битной команды
```

9.3 Команды доступа к памяти

9.3.1 В таблице 9.5 перечислены поддерживаемые команды доступа к памяти.

Таблица 9.5 – Команды доступа к памяти

Мнемоника	Краткое описание
ADR	Загрузка адреса, заданного относительно счётчика команд PC
CLREX	Сброс эксклюзивного доступа
LDM{mode}	Загрузка нескольких регистров
LDR{type}	Загрузка регистра с использованием непосредственного смещения
LDR{type}	Загрузка регистра с использованием смещения в регистре
LDR{type}T	Загрузка регистра с непривилегированным доступом
LDR	Загрузка регистра с использованием относительного адреса
LDRD	Загрузка регистра с использованием относительного адреса
LDREX{type}	Эксклюзивное чтение регистра
POP	Извлечение регистра из стека
PUSH	Загрузка регистра в стек
STM{mode}	Сохранение нескольких регистров
STR{type}	Сохранение регистра с использованием непосредственного смещения
STR{type}	Сохранение регистра с использованием смещения в регистре
STR{type}T	Сохранение регистра с непривилегированным доступом
STREX{type}	Эксклюзивная запись регистра

9.3.2 ADR

9.3.2.1 Загрузка адреса, заданного относительно счетчика команд PC.

Синтаксис

ADR{*cond*} *Rd*, *label*

где:

cond – необязательный суффикс условия выполнения;

Rd – регистр-приёмник;

label – выражение, определяемое относительно PC;

Описание

Команда ADR формирует адрес, прибавляя непосредственно заданного значения смещения к значению счётчика команд PC, после чего записывает результат в регистр-приёмник.

Эта команда упрощает генерацию переместимого кода, поскольку адрес в данном случае задаётся относительно значения PC.

Если команда ADR используется для формирования адреса, используемого командами BX или BLX**,** то для корректного выполнения программист должен быть уверен, что бит [0] сформированного адреса будет установлен в 1.

Значение *label* должно находиться в пределах от минус 4095 до плюс 4095 относительно адреса в счётчике команд.

Для использования всего диапазона значений смещения или для формирования адресов, не выровненных на границу слова, может потребоваться использование суффикса .W**.**

Ограничения

Нельзя использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве регистра *Rd*.

Флаги

Команда не влияет на состояние флагов.

Пример

```
ADR Rl,  TextMessage ; Заносит в Rl адрес ячейки памяти, обозначенной
                     ; меткой TextMessage
```

9.3.3 LDR и STR, «непосредственное смещение»

9.3.3.1 Загрузка или сохранение регистров в режиме адресации с непосредственным смещением, адресации с пред- или постиндексацией.

Синтаксис

```
op{type}{cond} Rt, [Rn {, #offset}]      ; непосредственное смещение
op{type}{cond} Rt, [Rn, #offset]!        ; прединдексация
op{type}{cond} Rt, [Rn], #offset         ; постиндексация
opD{cond} Rt, Rt2, [Rn {, #offset}]      ; непосредственное смещение, два слова
opD{cond} Rt, Rt2, [Rn, #offset]!        ; прединдексация, два слова
opD{cond} Rt, Rt2, [Rn], #offset         ; постиндексация, два слова
```

где:

op – операция:

– LDR -загрузка регистра;

– STR - сохранение регистра;

type – тип операнда:

B - беззнаковый байт, при загрузке дополняется нулями до 32-битного значения;

SB - знаковый байт, расширяется до 32-битного значения (только LDR);

H - беззнаковое полуслово, при загрузке дополняется нулями до 32-битного значения;

SH - знаковое полуслово, расширяется до 32-битного значения (только LDR);

без суффикса – 32-разрядное слово;

cond – необязательный суффикс условия выполнения;

Rt – загружаемый или сохраняемый регистр;

Rn – регистр, содержащий базовый адрес памяти;

offset – смещение относительно *Rn.* Если смещение не указано, то адрес равен содержимому *Rn*;

Rt2 – дополнительный регистр для загрузки или сохранения при операциях с двойнымсловами.

Описание

Команда LDR загружает один или два регистра данными из памяти. Команда STR сохраняет содержимое одного или двух регистров в память.

Команды загрузки и сохранения с непосредственным смещением могут использовать следующие режимы адресации:

1) Адресация со смещением

Величина смещения прибавляется или вычитается из адреса, находящегося в регистре *Rn.* Результат используется в качестве адреса, по которому производится обращение к памяти. Регистр *Rn* не изменяется. Синтаксис задания данного режима:

[Rn, #offset]

2) Адресация с прединдексированием

Величина смещения прибавляется или вычитается из адреса, находящегося в регистре *Rn.* Результат используется в качестве адреса, по которому производится обращение к памяти, и сохраняется в регистре *Rn.* Синтаксис задания данного режима:

[Rn, #offset]!

3) Адресация с постиндексированием

Значение, находящееся в регистре *Rn,* используется в качестве адреса, по которому производится обращение к памяти. После выполнения операции величина смещения прибавляется или вычитается из этого значения и результат сохраняется в регистре *Rn.* Синтаксис языка ассемблера для данного режима имеет вид:

[Rn], #offset

Загружаемое или сохраняемое значение может иметь разрядность байт, полуслово, слово или двойное слово. В командах загрузки байты и полуслова могут быть как знаковыми, так и беззнаковыми.

Допустимые значения смещения для всех режимов адресации приведены в таблице 9.6.

Таблица 9.6 – Допустимые значения смещения

Тип операнда	Непосредственное смещение	Прединдексирование	Постиндексирование
Слово, полуслово (со знаком и без знака), байт (со знаком и без знака)	от -255 до 4095	от -255 до 255	от -255 до 255
Двойное слово	Кратное 4 в диапазоне от -1020 до 1020	Кратное 4 в диапазоне от -1020 до 1020	Кратное 4 в диапазоне от -1020 до 1020

Ограничения

Для команд загрузки:

– использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве регистра *Rt* можно только в командах загрузки слова;

– при загрузке двойных слов *Rt* должен отличаться от *Rt2*;

– в режимах адресации с пред- и постиндексированием *Rn* должен отличаться от *Rt* и *Rt2*.

В случае, когда счетчик команд используется в качестве *Rt*:

– дял корректного выполнения команды бит [0] загружаемого значения должен быть установлен в 1;

– переход производится по адресу, соответствующему значению бита [0] в 0;

– если команда условно выполняемая, то она должна быть последней командой в IT-блоке.

Для команд сохранения:

– использовать регистр SP (указатель стека) в качестве *Rt* можно только в командах записи слова;

– использовать регистр PC (счетчик команд) в качеств *Rt* и *Rn* нельзя;

– *Rn* должен отличаться от *Rt* и *Rt2* в режимах адресации с пред- и постиндексированием.

Флаги условий

Команда не влияет на состояние флагов.

Примеры

```
LDR    R8, [R10]              ; Загружает R8 из памяти по адресу, находящемуся в R10
LDRNE R2, [R5, #960]!         ; Загружает (условно) R2 из слова , расположенного со
                              ; смещением +960 байт относительно адреса,
                              ; находящегося в R5, и увеличивает R5 на 960
STR    R2, [R9,#const-struc]  ; const-struc это вычисляемое выражение, результатом
                              ; которого является константа из диапазона 0-4095
STRH   R3,  [R4], #4          ; Сохраняет R3 как полуслово по адресу, находящемуся
                              ; в R4, затем увеличивает R4 на 4
LDRD  R8, R9,  [R3, #0x20]    ; Загружает R8 из слова, расположеннего со смещением
                              ; +32 байта относительно адреса, находящегося в R3,
                              ; и загружает R9 расположеннего со смещением
                              ; +36 байт
                              ; относительно адреса, находящегося в R3
STRD    R0,  R1, [R8],  #-16  ; Сохраняет R0 по адресу, находящемуся в R8,
                              ; и сохраняет R1 в слове памяти по адресу,
                              ; который на 4 байта больше адреса,
                              ; находящегося в
                              ; R8, после чего уменьшает R8 на 16.
```

9.3.4 LDR и STR, «регистровое смещение»

9.3.4.1 Загрузка и сохранение регистров в режиме адресации со смещением, заданным в регистре.

Синтаксис

op{*type*}{*cond*} *Rt*, [*Rn*, *Rm* {, LSL #*n*}]

где:

op - операция:

– LDR - Загрузка регистра

– STR - Сохранение регистра

type - тип операнда:

– B - беззнаковый байт, при загрузке дополняется нулями до 32-битного значения

– SB - знаковый байт, расширяется до 32-битного значения (только LDR)

– H - беззнаковое полуслово, при загрузке дополняется нулями до 32-битного значения

– SH - знаковое полуслово, расширяется до 32-битного значения (только LDR)

– Без суффикса – 32-разрядное слово

cond - необязательный суффикс условия выполнения;

Rt - регистр для загрузки или сохранения.

Rn - регистр, содержащий базовый адрес памяти.

Rm - регистр, содержащий значение используемого смещения.

LSL #*n* - необязательный параметр сдвига, где *n* лежит в диапазоне от 0 до 3.

Описание

Команда LDR загружает регистр данными из памяти.

Команда STR сохраняет содержимое регистра в память.

Адрес памяти для загрузки или сохранения данных задаётся относительно базового адреса, находящегося в регистре *Rn.* Величина смещения определяется содержимым регистра *Rm*, которое может быть сдвинуто на 0...3 бита влево посредством команды LSL.

Загружаемое или сохраняемое значение может иметь разрядность байт, полуслово или слово. В командах загрузки байты и полуслова могут быть как знаковыми, так и беззнаковыми.

Ограничения

– нельзя использовать регистр PC (счетчик команд) в качестве *Rn*;

– нельзя использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве *Rm*;

– использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве *Rt* можно только в командах загрузки/сохранения слова;

– использовать регистр PC (счетчик команд) в качестве *Rt* можно только в командах загрузки слова.

В случае, когда регистр PC (счетчик команд) используется в качетсве *Rt*:

– для корректного выполнения команды бит [0] загружаемого значения должен быть установлен в 1. При этом переход осуществляется по адресу, выровненному на по границе полуслова;

– если команда условно выполняемая, то она должна быть последней командой в IT-блоке.

Флаги условий

Команда не влияет на состояние флагов.

Примеры

```
STR     R0, [R5, R1]      ; Сохраняет содержимое R0 по адресу равному сумме R5 и R1
LDRSB  R0, [R5, R1, LSL #1] ; Читает однобайтовое значение по адресу, равному
                                  ; R5 +2*R1, выполняет расширение знака до размеров
                                  ; слова и помещает результат в R0
STR     R0, [R1, R2, LSL #2]   ; Сохраняет R0 по адрес равному R1+ 4* R2
```

9.3.5 LDR и STR, «непривилегированный доступ»

9.3.5.1 Загрузка и сохранение регистров в режиме непривилегированного доступа.

Синтаксис

op{*type*}T{*cond*} *Rt*, [*Rn* {, #*offset*}] ; Непосредственное смещение

где

op - операция:

– LDR загрузка регистра;

– STR сохранение регистра;

type - тип операнда :

– B - беззнаковый байт, при загрузке дополняется нулями до 32-битного

– значения;

– SB - знаковый байт, расширяется до 32-битного значения (только LDR);

– H - беззнаковое полуслово, при загрузке дополняется нулями до 32-битного значения;

– SH - знаковое полуслово, расширяется до 32-битного значения (только LDR);

– без суффикса - 32-разрядное слово;

cond - необязательный суффикс условия выполнения;

Rt - регистр для загрузки или сохранения;

Rn - регистр, содержащий базовый адрес памяти;

offset - смещение относительно *Rn* может быть от 0 до 255. Если смещение не указано, то адрес равен содержимому *Rn*.

Описание

Эти команды выполняют те же операции, что и команды доступа к памяти с непосредственно задаваемым смещением. Отличие заключается в том, что эти команды осуществляют обращение к памяти исключительно на непривилегированном уровне, даже в случае, если они выполняются в привилегированном программном коде.

При использовании в программном коде, выполняющемся в непривилегированном режиме, эти команды полностью идентичны обычным командам доступа к памяти с непосредственным смещением.

Ограничения

– нельзя использовать регистр PC (счетчик команд) в качестве *Rn*;

– нельзя использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве *Rt*.

Флаги условий

Команда не влияет на состояние флагов.

Примеры

```
STRBTEQ  R4, [R7]     ; Сохраняет (условно) младший значащий байт содержимого
                      ; R4 по адресу, содержащемуся в R7,
                      ; на непривилегированном уровне
LDRHT    R2, [R2, #8] ; Загружает в R2 полуслово с адреса равного R2 + 8
                      ; на непривилегированном уровне.
```

9.3.6 LDR и STR, адресация относительно счетчика команд PC

9.3.6.1 Загрузка регистра из памяти.

Синтаксис

LDR{*type*}{*cond*} *Rt*, *label*

LDRD{*cond*} *Rt*, *Rt2*, *label* ; загрузка двух слов

где:

type - тип операнда:

– B - беззнаковый байт, при загрузке дополняется нулями до 32-битного значения;

– SB - знаковый байт, расширяется до 32-битного значения (только LDR);

– H - беззнаковое полуслово, при загрузке дополняется нулями до 32-битного значения;

– SH - знаковое полуслово, расширяется до 32-битного значения (только LDR);

– без суффикса - 32-разрядное слово;

cond - необязательный суффикс условия выполнения;

Rt - регистр для загрузки или сохранения;

Rt2 - второй регистр для загрузки или сохранения;

label – относительный адрес.

Описание

Команда LDR загружает в регистр данные из памяти. Адрес ячейки памяти определяется меткой или смещением относительно PC.

Загружаемое или хранимое значение может иметь разрядность байт, полуслово или слово. Для команд загрузки байты и полуслова могут быть как знаковыми, так и беззнаковыми.

Значение *label* должно находиться в допустимом для данной команды диапазоне. Допустимые величины смещения *label* относительно PC указаны в таблице 9.7.

Таблица 9.7 – Допустимые значения смещения

Тип операнда	Диапазон значений смещения
Слово, полуслово (со знаком и без знака), байт (со знаком и без знака)	от - 4095 до 4095
Двойное слово	от -1020 до 1020

Для использования всего диапазона значений смещения может потребоваться использование суффикса .W.

Ограничения

– использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве *Rt* можно только в командах загрузки слова;

– нельзя использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве *Rt2*;

– *Rt* должен отличаться от *Rt2*.

Когда PC (счетчик команд) используется в качестве *Rt* в команде загрузки слова:

– для корректного выполнения команды бит [0] загружаемого значения должен быть установлен в 1. При этом переход осуществляется по этому адресу, выровненному по границе полуслова;

– если команда условно выполняемая, то она должна быть последней командой в IT-блоке.

Флаги условий:

Команда не влияет на состояние флагов.

Примеры:

```
LDR   R0,  LookUpTable      ; Загружает в R0 слово, расположенное по адресу,
                            ; помеченному меткой LookUpTable
LDRSB R7,  localdata        ; Считывает байт с адреса, помеченного меткой
                            ; localdata, выполняет расширение знака до размера
                            ; слова
                            ; и помещает результат в R7.
```

9.3.7 **LDM и STM**

9.3.7.1 Загрузка и сохранение нескольких регистров.

Синтаксис

op{*addr_mode*}{*cond*} *Rn*{!}, *reglist*

где:

op - операция:

– LDM загрузка нескольких регистров;

– STM сохранение нескольких регистров;

addr_mode - режим адресации:

– IA инкрементирование адреса после каждого обращения к памяти (режим по умолчанию);

– DB декрементирование адреса перед каждым обращением к памяти;

cond - необязательный суффикс условия выполнения;

Rn - регистр, содержащий базовый адрес;

! - необязательный суффикс обратной записи значения базового регистр. В случае присутствия в мнемонике команды символа «!» последний адрес, по которому производилось чтение или запись данных, сохраняется обратно в регистре *Rn*;

reglist - заключённый в фигурные скобки список из одного или нескольких регистров, загружаемых из памяти или сохраняемых в память. Может включать в себя диапазоны номеров регистров. При наличии в списке более одного регистра или диапазона регистров элементы списка разделяются запятыми.

LDM и LDMFD являются синонимами LDMIA. Мнемоника LDMFD указывает на использование данной команды для извлечения данных из «полного» убывающего стека (Full Descending stacks).

LDMEA является синонимом LDMDB и указывает на использование данной команды для извлечения данных из «пустого» возрастающего стека (Empty Ascending stacks.).

STM и STMEA являются синонимами STMIA. Мнемоника STMEA указывает на использование данной команды для помещения данных в «пустой» возрастающий стек (Empty Ascending stacks).

STMFD является синонимом STMDB и указывает на использование данной команды для помещения данных в «полный» убывающий стек (Full Descending stacks).

Описание

Команды LDR загружают в регистры, указанные в *reglist*, слова из памяти, начиная с адреса, указанного в *Rn.*

Команды STM сохраняют содержимое регистров, указанных в *reglist*, в памяти, начиная с адреса, указанного в *Rn.*

Команды LDM, LDMIA, LDMFD, STM, STMIA и STMEA для доступа используют адреса ячеек памяти в диапазоне от *Rn* до *Rn* + 4 * (n -1) с интервалом в 4 байта, где n – количество регистров в *reglist.* Обращения к памяти производятся в порядке возрастания номеров регистров, при этом регистру с наименьшим номером соответствует самый младший адрес, а регистру с наибольшим номером — самый старший адрес. При наличии в мнемонике команды суффикса, разрешающего обратную запись («!»), значение *Rn + 4*(n-1)* записывается обратно в регистр *Rn.*

Для команд LDMDB, LDMEA, STMDB и STMFD адреса ячеек памяти, по которым производятся обращения, располагаются в диапазоне от *Rn* до *Rn - 4 * (n* -1) с интервалом в 4 байта, где n — число регистров в *reglist.* Обращения к памяти производятся в порядке убывания номеров регистров, при этом регистру с наибольшим номером соответствует самый старший адрес, а регистру с наименьшим номером — самый младший адрес. При наличии в мнемонике команды суффикса, разрешающего обратную запись, значение *Rn-4* (n-1)* сохраняется в регистре *Rn.*

Инструкции PUSH и POP могут быть выражены через инструкции LDM и STM.

Ограничения

– нельзя использовать регистр PC (счетчик команд) в качестве *Rn*;

– список регистров reglist не должен содержать SP (указатель стека);

– Rt должен отличаться от Rt2;

– в любой команде STM в списке регистров reglist нельзя указывать PC (счетчик команд);

– в любой команде LDM в списке регистров reglist нельзя указывать одновременно PC и LR;

– reglist не должен содержать *Rn,* если в команде указан суффикс обратной записи “!”.

В случае, если команда LDM содержит в списке *reglist* регистр PC (счетчик команд):

– для корректного выполнения команды бит [0] загружаемого значения в PC должен быть установлен в 1, переход при этом осуществляется по адресу, выровненному по границе полуслова;

– если команда условно выполняемая, то она должна быть последней командой в 1T-блоке.

Флаги условий

Команда не влияет на состояние флагов.

Примеры

```
LDM   R8, {R0,R2,R9}              ; LDMIA синоним для LDM
STMDB R1!,{R3-R6,R11,R12}
```

Пример некорректной записи:

```
STM   R5!,{R5,R4,R9}              ; Значение, сохраняемое в R5 непредсказуемо
LDM   R2, {}                      ; В списке должен быть хотя бы один.
```

9.3.8 **PUSH и POP**

9.3.8.1 Загрузка регистров в стек и извлечение регистров из стека.

Синтаксис:

PUSH{*cond*} *reglist*

POP{*cond*} *reglist*

где:

cond - необязательный суффикс условия выполнения;

reglist - непустой список регистров, заключённый в фигурные скобки.

Может включать в себя диапазоны регистров. При наличии в списке более одного регистра или диапазона регистров элементы списка разделяются запятыми.

Команды PUSH и POP являются синонимами команд STMDB и LDM (или LDMIA), в которых базовый адрес памяти содержится в регистре указателя стека SP.

Описание

Команда PUSH сохраняет содержимое регистров в стеке, при этом регистр с наименьшим номером сохраняется в слове памяти с самым младшим адресом, а регистр с наибольшим номером — в слове с самым старшим адресом.

Команда POP загружает регистры из стека, при этом регистр с наименьшим номером загружается из слова памяти с самым младшим адресом, а регистр с наибольшим номером — из слова с самым старшим адресом.

Старшим адресом для команды PUSH является содержимое регистра SP, уменьшенное на 4, а младшим адресом для команды POP — содержимое регистра SP, т.е. эти команды реализуют «полный» убывающий стек. После завершения операции команда PUSH изменяет регистр SP таким образом, чтобы тот указывал на слово памяти, в котором был сохранён регистр с наименьшим номером. Команда POP после завершения операции изменяет регистр SP таким образом, чтобы тот указывал на слово памяти, расположенное выше того слова, из которого был считан регистр с наибольшим номером.

Если в списке *reglist* команды POP присутствует регистр PC, то переход по соответствующему адресу будет произведён после завершения команды POP. При этом бит [0] считанного из стека значения заносится в бит T регистра APSR. Для корректной работы процессора указанный бит должен быть установлен в 1.

Ограничения

– список регистров reglist не должен содержать SP (указатель стека);

– список регистров reglist команды PUSH не должен содержать PC (счетчик команд);

– список регистров reglist команды POP не должен одновременно содержать PC (счетчик команд) и LR.

В случае, если в списке регистров в *reglist* команды POP содержится PC (счетчик команд):

– для корректного выполнения команды бит [0] загружаемого значения должен быть установлен в 1, при этом переход осуществляется по этому адресу, выровненному по границе полуслова;

– если команда условно выполняемая, то она должна быть последней командой в IT-блоке.

Флаги условий

Команда не влияет на состояние флагов.

Примеры

```
PUSH {R0,R4-R7}          ; Помещает содержимое регистров
                         ; R0,R4,R5,R6,R7 в стек
PUSH {R2,LR}             ; Помещает содержимое регистра R2 и регистра
                         ; LR в стек
POP  {R0,R6,PC}          ; Загружает регистры R0,R6 и PC из стека,
                         ; где в скобках новое значение PC.
```

9.3.9 **LDREX и STREX**

9.3.9.1 Запись и чтение регистров в режиме эксклюзивного доступа.

Синтаксис

LDREX{*cond*} *Rt*, [*Rn* {, #*offset*}]

STREX{*cond*} *Rd*, *Rt*, [*Rn* {, #*offset*}]

LDREXB{*cond*} *Rt*, [*Rn*]

STREXB{*cond*} *Rd*, *Rt*, [*Rn*]
LDREXH{*cond*} *Rt*, [*Rn*]
STREXH{*cond*} *Rd*, *Rt*, [*Rn*]
где:
cond - необязательный суффикс условия выполнения;
Rd - регистр-приёмник для сохранения возвращённого кода состояния;
Rt - загружаемый или сохраняемый регистр;
Rn - регистр, содержащий базовый адрес памяти;
offset - смещение относительно *Rn.* Если смещение не указано, то адрес равен содержимому *Rn*.

Описание

Команды LDREX, LDREXB и LDREXH загружают из памяти в регистр слово, байт и полуслово соответственно.

Команды STREX, STREXB и STREXH осуществляют попытку записи в память слова, байта и полуслова соответственно. Адрес, используемый в любой команде эксклюзивной записи данных, должен быть идентичен адресу последней выполненной команды эксклюзивного чтения. Значение, сохраняемое в памяти командой эксклюзивной записи, должно иметь такую же разрядность, что и значение, считанное последней командой эксклюзивного чтения. То есть для решения задач синхронизации команда эксклюзивной записи должна всегда использоваться в паре с командой эксклюзивного чтения.

При успешном выполнении команда эксклюзивной записи заносит 0 в регистр-приёмник. Если попытка записи была неудачной, то в регистр-приёмник заносится 1. Нулевое значение, возвращённое командой записи в регистр-приёмник, гарантирует, что ни один процесс в системе не сможет получить доступ к использованной ячейке памяти между командами эксклюзивного чтения и эксклюзивной записи.

В интересах быстродействия число команд, располагающихся между командой эксклюзивной записи и эксклюзивного чтения, должно быть сведено к минимуму.

Результат выполнения команды эксклюзивной записи при обращении по адресу, отличному от использованного в предшествующей команде эксклюзивного чтения, будет непредсказуем.

Ограничения

- нельзя использовать PC (счетчик команд);
- нельзя использовать регистр SP (указатель стека) в качестве *Rd*;
- нельзя использовать регистр SP (указатель стека) в качестве *Rt*;
- в командах STREX регистр *Rd* должен отличаться от *Rt* и от *Rn*;
- величина *offset* должна находиться в диапазоне 0 - 1020 и быть кратной 4.

Флаги условий

Команда не влияет на состояние флагов.

Примеры

```
MOV R1, #0x1                ; инициализируем значение
                            ; для признака блокировки try
LDREX R0, [LockAddr]        ; читаем значение признака блокировки
CMP R0, #0                  ; блокировка снята?
ITT EQ                      ; IT –блок для команд STREXEQ и CMPEQ
STREXEQ R0, R1, [LockAddr]  ; пробуем поставить блокировку
CMPEQ R0, #0                ; успешно?
BNE try                     ; No – пробуем снова
...                         ; Yes – поставили блокировку
```

9.3.10 **CLREX**

9.3.10.1 Сброс эксклюзивного доступа.

Синтаксис:

CLREX{*cond*}

где:

cond - необязательный суффикс условия выполнения; см. п.3.3.7.

Описание:

После использования команды CLREX любая последующая команда STREX, STREXB и STREXH заносит 1 в регистр-приёмник и, соответственно, не осуществляет запись в память. Принудительное блокирование эксклюзивной записи в память может потребоваться в коде обработчика исключений в том случае, если исключение возникнет во время операции синхронизации между командой эксклюзивной записи и соответствующей командой эксклзивного чтения.

Флаги условий:

Команда не влияет на состояние флагов.

Примеры:

CLREX

9.4 Команды обработки данных

9.4.1 В таблице 9.8 перечислены все команды обработки данных.

Таблица 9.8 – Команды обработки данных

Мнемоника	Краткое описание
ADC	Сложение с переносом
ADD	Сложение
ADDW	Сложение
AND	Логическое И (AND)
ASR	Арифметический сдвиг вправо
BIC	Очистка битов
CLZ	Определение количества ведущих нулей
CMN	Сравнение с противоположным знаком
CMP	Сравнение
EOR	Исключающее ИЛИ
LSL	Логический сдвиг влево
LSR	Логический сдвиг вправо
MOV	Загрузка
MOVT	Загрузка в старшее полуслово
MOVW	Загурзка 16-разрядной константы
MVN	Загрузка инверсного значения
ORN	Логическое ИЛИ с инверсией
ORR	Логическое ИЛИ
RBIT	Перестановка бит
REV	Изменение порядка байтов в слове на обратный
REV16	Изменение порядка байтов в полусловах на обратный
REVSH	Изменение порядка байтов в младшем полуслове. Распространение знакового бита в старшее полуслово
ROR	Циклический сдвиг вправо
RRX	Циклический сдвиг вправо на один обит с учетом переноса
RSB	Вычитание с обратным порядком аргументов
SADD16	Сложение 16-разрядных целых чисел со знаком
SADD8	Сложение 8-разрядных целых чисел со знаком
SASX	Сложение и вычитание чисел со знаком с перестановкой
SSAX	Вычитание и сложение чисел со знаком с перестановкой
SBC	Вычитание с переносом
SEL	Выбор байт
SHADD16	Полусумма 16-разрядных целых чисел со знаком

Продолжение таблицы 9.8

Мнемоника	Краткое описание
SHADD8	Полусумма 8-разрядных целых чисел со знаком
SHASX	Сложение и вычитание целых чисел со знаком с перестановкой и делением пополам.
SHSAX	Вычитание и сложение целых чисел со знаком с перестановкой и делением пополам.
SHSUB16	Полуразность 16-разрядных целых чисел со знаком
SHSUB8	Полуразность 8-разрядных целых чисел со знаком
SSUB16	Вычитание 16-разрядных целых чисел со знаком
SSUB8	Вычитание 8- разрядных целых чисел со знаком
SUB	Вычитание
SUBW	Вычитание
TEQ	Проверка на равенство
TST	Проверка значения бит по маске
UADD16	Сложение 16-разрядных целых чисел без знака
UADD8	Сложение 8-разрядных целых чисел без знака
UASX	Сложение и вычитание 16-разрядных чисел без знака с перестановкой
USAX	Вычитание и сложение 16-разрядных чисел без знака с перестановкой
UHADD16	Полусумма 16-разрядных целых чисел без знака
UHADD8	Полусумма 8-разрядных целых чисел без знака
UHASX	Сложение и вычитание целых чисел без знака с перестановкой и делением пополам.
UHSAX	Вычитание и сложение целых чисел без знака с перестановкой и делением пополам.
UHSUB16	Полуразность 16-разрядных целых чисел без знака
UHSUB8	Полуразность 8-разрядных целых чисел без знака
USAD8	Безнаковоая сумма абсолютных разностей 8-разрядных целых
USADA8	Безнаковоая сумма абсолютных разностей 8-разрядных целых с дополнительным сложением
USUB16	Вычитание 16-разрядных целых чисел без знака
USUB8	Вычитание 8-разрядных целых чисел без знака

9.4.2 **ADD, ADC, SUB, SBC и RSB**

9.4.2.1 Сложение, сложение с переносом, вычитание, вычитание с переносом, вычитание с противоположным порядком аргументов.

Синтаксис

op{S}{*cond*} {*Rd*,} *Rn*, *Operand2*

op{*cond*} {*Rd*,} *Rn*, *#imm12* ; только команды ADD и SUB

где:

op - операция:

– ADD - сложение;

– ADC - сложение с переносом;

– SUB - вычитание;

– SBC - вычитание с переносом;

– RSB – вычитание с противоположным порядком аргументов;

S - необязательный суффикс. При наличии в мнемонике команды суффикса S состояние флагов условий изменяется в соответствии с результатом операции;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приёмник. Если *Rd* отсутствует, то регистром-приёмником является *Rn*;

Rn - регистр, содержащий первый операнд;

Operand2 - второй операнд.

imm12 - любое число в диапазоне 0 - 4095.

Описание

Команда ADD складывает *Operand2* или *imml2* с содержимым *Rn.*

Команда ADC складывает *Operand2* с содержимым *Rn* и прибавляет к результату значение флага переноса.

Команда SUB вычитает *Operand2* или *imml2* из содержимого *Rn.*

Команда SBC вычитает *Operand2* из содержимого *Rn*. Если флаг переноса сброшен, дополнительно уменьшает результат на единицу.

Команда RSB вычитает значение, находящееся в *Rn* из *Operand2.* Достоинством этой команды является большое число опций, поддерживаемых операндом *Operand2*.

Команды ADC и SBC используются для реализации вычисления с повышеноӥ разрядностью.

Мнемоники ADDW и SUBW эквивалентны мнемоникам ADD и SUB соответственно при использовании в качестве операнда константы *imm12*.

Ограничения

– нельзя использовать регистры PC (счетчик команд) и SP (указатель стека) в качестве *Operand2*;

– регистр SP (указатель стека) может использоваться в качестве *Rd* только в командах ADD и SUB со следующими ограничениями:

– в качестве *Rn* также должен использоваться SP (указатель стека);

– любые операции сдвига в *Operand2* должны быть ограничены сдвигом влево не более чем на 3 бита при использовании LSL;

– регистр SP (указатель стека) может использоваться в качестве *Rn* только в командах ADD и SUB;

– регистры PC (счетчик команд) может использоваться в качестве *Rd* только в команде ADD{cond} PC, PC, Rm, где:

– использование суффикса S не допускается;

– нельзя использовать регистры PC (счетчик команд) и SP (указатель стека) в качестве *Rm*;

– если команда условно выполняемая, то она должна быть последней командой в IT-блоке;

– в качетсве *Rn* можно использовать PC (счетчик команд) только в командах ADD и SUB (за исключением команды «ADD{cond} PC, PC, *Rm»)* и только со следующими ограничениями:

– использование суффикса S не допускается;

– второй операнд должен быть константой из диапазона 0 - 4095.

При использовании PC (счетчика команд) в операциях сложения и вычитания биты [1:0] регистра PC сбрасываются в 0b00 перед выполнением операции, выравнивая тем самым базовый адрес по границе полуслова.

При необходимости генерации адреса команды необходимо скорректировать значение константы в соответствии со значением PC. ARM рекомендует использовать команду ADR, т.к. в этом случае ассемблер автоматически вычисляет правильное значение константы.

При использовании PC (счетчика команд) в качестве *Rd* в команде ADD{cond} PC, PC, Rm:

– бит [0] значения, загружаемого в PC, игнорируется;

– переход производится по адресу, соответствующему нулевому значению этого бита.

Флаги условия

В сдучае, если в комаде указан суффикс S, эти команды изменяют состояние флагов N, Z, C и V в соответствии с результатом операции.

Примеры

```
ADD     R2, R1, R3
SUBS    R8, R6, #240    ; Устанавливает флаги в
                        ; соответствии с результатом
```

```
RSB     R4, R4, #1280     ; Вычитает содержимое R4 из константы 1280
ADCHI   R11, R0, R3       ; Выполняется только если флаг C установлен, а
                          ; флаг Z сброшен.
```

В следующем примере показаны две команды, которые выполняют сложение 64-битного целого, находящегося в регистрах R2 и R3, с другим 64-битным целым, находящимся в регистрах R0 и R1, и помещают результат в регистры R4 и R5.

```
ADDS  R4, R0, R2          ; складываем младшие слова
ADC   R5, R1, R3          ; складываем старшие слова, учитывая перенос
```

Данные с повышенной разрядностью не обязательно должны храниться в последовательно расположенных регистрах. В приведенном ниже примере показан фрагмент кода, который вычитает одно 96-битное целое, хранящееся в регистрах R9, R1 и R11, из другого, хранящегося в регистрах R6, R2 и R8. Результат вычитания сохраняется в регистрах R6, R9 и R2.

```
SUBS   R6, R6, R9         ; вычитаем младшие слова
SBCS   R9, R2, R1         ; вычитаем средние слова, учитывая перенос
SBC    R2, R8, R11        ; вычитаем старшие слова, учитывая перенос
```

9.4.3 **AND, ORR, EOR, BIC и ORN**

9.4.3.1 Логические операции И, ИЛИ, Исключающее ИЛИ, сброс бит по маске, ИЛИ с инверсией.

Синтаксис

op{S}{*cond*} {*Rd*,} *Rn*, *Operand2*

где:

op - операция :

– AND - логическое И;

– ORR - логическое ИЛИ;

– EOR - исключающее ИЛИ;

– BIC – сброс бит по маске;

– ORN - логическое ИЛИ с инверсией (OR NOT);

S - необязательный суффикс. При наличии в мнемонике команды суффикса S состояние флагов условий изменяется в соответствии с результатом операции;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приёмник;

Rn - регистр, содержащий первый операнд;

Operand2 - второй операнд;

Команды AND, EOR и ORR выполняют побитовые операции «И», «ИЛИ» и «Исключающее ИЛИ» между содержимым *Rn* и *Operand2*.

Команда BIC выполняет операцию «И» между содержимым *Rn* и инверсным значением *Operand2*.

Команда ORN выполняет операцию «ИЛИ» между содержимым *Rn* и инверсным значением *Operand2*.

Ограничения

Нельзя использовать SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Если в команде указан суффикс S, то эти команды:

– изменяют флаги N и Z в соответствии с результатом операции;

– могут изменять состояние флага C при вычислении Operand2;

– не влияют на состояние флага V.

Примеры

```
AND     R9, R2, #0xFF00
ORREQ   R2, R0, R5
ANDS    R9, R8, #0x19
EORS    R7, R11, #0x18181818
```

```
BIC     R0, R1, #0xab
ORN     R7, R11, R14, ROR #4
ORNS    R7, R11, R14, ASR #32
```

9.4.4 **ASR, LSL, LSR, ROR и RRX**

9.4.4.1 Арифметический сдвиг вправо, логический сдвиг влево, логический сдвиг вправо, циклический сдвиг вправо и циклический сдвиг вправо с переносом.

Синтаксис

op{S}{cond} Rd, Rm, Rs
op{S){cond} Rd, Rm, #n
RRX{*S}{cond} Rd, Rm*
где:

op - операция:

– ASR - арифметический сдвиг вправо;

– LSL - логический сдвиг влево;

– LSR - логический сдвиг вправо;

– ROR - циклический сдвиг вправо;

S - необязательный суффикс. При наличии в мнемонике команды суффикса S состояние флагов условий изменяется в соответствии с результатом операции;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приёмник;

Rm - регистр, содержащий сдвигаемое значение;

Rs - регистр, в котором хранится величина сдвига для содержимого *Rm.* Используется только младший байт регистра, соответственно, величина сдвига может быть в диапазоне 0 - 255;

n - величина сдвига. Диапазон допустимых значений зависит от команды:

– ASR от 1 до 32

– LSL от 0 до 31

– LSR от 1 до 32

– ROR от 1 до 31

Вместо команды LSLS Rd, Rm, #0 лучше использовать команду MOVS Rd, Rm.

Описание

Команды ASR, LSL, LSR и ROR сдвигают содержимое регистра *Rm* влево или вправо на число бит, определяемое константой *n* или регистром *Rs.*

Команда RRX сдвигает содержимое регистра *Rm* на один бит вправо с учетом переноса.

Все рассматриваемые команды сохраняют результат в регистре *Rd,* при этом содержимое регистра *Rm* не изменяется.

Ограничения

Нельзя использовать SP (указатель стека) либо PC (счетчик команд).

Флаги условия

В случае, если в команде указан суффикс S:

– изменяются состояния флагов N и Z в соответствии с результатом;

– во флаг C заносится значение последнего выдвинутого бита, если величина сдвига не равна нулю;

Примеры

```
ASR   R7, R8, #9      ; Арифметический сдвиг вправо на 9 бит
LSLS  R1, R2, #3      ; Логический сдвиг влево на 3 бита с обновлением флага
LSR   R4, R5, #6      ; Логический сдвиг вправо на 6 бит
ROR   R4, R5, R6      ; Циклический сдвиг вправо на значение в младших битах R6
RRX   R4, R5          ; Расширенный циклический сдвиг вправо.
```

9.4.5 **CLZ**

9.4.5.1 Подсчёт количества ведущих нулей.

Синтаксис

CLZ{*cond*} *Rd*, *Rm*

где:

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rm - регистр операнда.

Описание

Команда CLZ подсчитывает число ведущих нулевых битов в двоичной записи содержимого *Rm* и возвращает результат в *Rd.* Если ни один бит регистра *Rm* не установлен, результат будет равен 32. Если установлен только бит [31] регистра, результат будет равен 0.

Ограничения

Нельзя использовать SP (указатель стека)либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
CLZ    R4,R9
CLZNE R2,R3
```

9.4.6 **CMP и CMN**

9.4.6.1 Сравнение и сравнение с противоположным знаком.

Синтаксис

CMP{*cond*} *Rn*, *Operand2*

CMN{*cond*} *Rn*, *Operand2*

где:

cond - необязательный суффикс условия выполнения;

Rn - регистр, содержащий первый операнд;

Operand2 - «гибкий» второй операнд.

Описание

Эти команды выполняют сравнение содержимого регистра *Rn* со значением *Operand2.* Они изменяют флаги условий в соответствии с результатом сравнения, но не сохраняют сам результат в регистре.

Команда CMP вычитает *Operand2* из содержимого регистра *Rn,* т.е. выполняет ту же операцию, что и команда SUBS, но без сохранения результата вычитания.

Команда CMN складывает *Operand2* с содержимым регистра *Rn,* т.е. выполняет ту же операцию, что и команда ADDS, но без сохранения результата сложения.

Ограничения

- нельзя использовать регистр PC (счетчик команд);
- нельзя использовать регистр SP (указатель стека) в качетстве Operand2.

Флаги условия

Эти команды изменяют состояние флагов N, Z, C и V в соответствии с результатом операции.

Примеры

```
CMP     R2, R9
CMN     R0, #6400
CMPGT SP, R7, LSL #2
```

9.4.7 **MOVT**

9.4.7.1 Запись в старшее полуслово регистра.

Синтаксис

MOVT{*cond*} *Rd*, *#imm16*

где:

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

imm16 - любое число в диапазоне 0 - 65535.

Описание

Команда MOVT записывает 16-разрядную константу *imm16* в старшее полуслово регистра-приёмника *Rd* (*Rd* [31:16]). Состояние битов *Rd* [15:0] при этом остается неизменным.

Комбинация команд MOV и MOVT позволяет загружать в регистры произвольные 32-битные константы.

Ограничения

Нельзя использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве *Rd*.

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
MOVT R3, #0xF123        ; Записывает 0xF123 в старшее полуслово регистра R3,
                        ; младшее полуслово и регистр APSR не изменяются.
```

9.4.8 REV, REV16, REVSH и RBIT

9.4.8.1 Изменение порядка бит или байтов в слове.

Синтаксис

op{*cond*} *Rd*, *Rn*

где:

op - операция:

– REV – изменение на обратный порядок байтов в слове;

– REV16 – изменение на обратный порядок байтов в полуслове;

– REVSH – изменить на обратный порадок байтов в младшем полуслове и произвести распространение знакового бита в старшее полуслово;

– RBIT – изменить порядок бит в 32-разрядном слове.

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий операнд.

Описание

Эти команды предназначены для изменения формата представления данных (endianness):

– REV преобразует 32-разрядное число в формате big-endian в число в формате little-endian и наоборот;

– REV16 преобразует 16-разрядное число в формате big-endian в число в формате little-endian и наоборот;

– REVSH выполняет одно из преобразований:

– 16-разрядное число со знаком в формате big-endian в 32-разрядное число со знаком в формате little-endian;

– 16-разрядное число со знаком в формате little-endian в 32-разрядное число со знаком в формате big-endian.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
REV     R3, R7          ; Меняет порядок байтов содержимого R7 и записывает в R3;
REV16   R0, R0          ; Меняет порядок байтов каждого 16-битного
                        ; полуслова в регистре R0
```

```
REVSH  R0, R5        ; Меняет порядок байтов знакового полуслова
REVHS  R3, R7        ; Переставляет байты при условии
RBIT   R7, R8        ; Меняет порядок битов содержимого R8 и записывает R7.
```

9.4.9 **SADD16 и SADD8**

9.4.9.1 Сложение 16-разрядных целых чисел со знаком, сложение 8-разрядных целых чисел со знаком.

Синтаксис

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– SADD16 - сложение 16-разрядных целых чисел со знаком;

– SADD8 - сложение 8-разрядных целых чисел со знаком;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Эти команды выполняют параллельное сложение полуслов или байтов.

Команда SADD16:

1. складывает каждое полуслово первого операнда с соответствующим полусловом второго операнда;
2. результат записывается в соответствующие полуслова регистра-приемника.

Команда SADD8:

1. складывает каждый байт первого операнда с соответствующим байтом второго операнда;
2. результат записывается в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SADD16 R1, R0        ; Сложение полуслов из регистра R0 с соответствующими
                     ; полусловами регистра R1, запись в
                     ; соответствующие полуслова регистра R1.
SADD8 R4, R0, R5     ; Сложение байт регистра R0 с соответствующими
                     ; байтами регистра R5 и запись в соответствующие
                     ; байты регистра R4.
```

9.4.10 **SHADD16 и SHADD8**

9.4.10.1 Полусумма 16-разрядных целых чисел со знаком, полусумма 8-разрядных целых чисел со знаком.

Синтаксис

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– SHADD16 - полусумма 16-разрядных целых чисел со знаком;

– SHADD8 - полусумма 8-разрядных целых чисел со знаком;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Эти команды выполняют параллельное сложение 16-ти или 8-ми разрядных данных, затем делят результат пополам перед его записью в регистр-приемник:

Команда SHADD16:

1. складывает каждое полуслово первого операнда с соответствующим полусловом второго операнда;
2. сдвигает полученные результаты сложения вправо на 1 бит, что приводит к их делению на 2;
3. записывает полученные после сдвига полуслова в соответствующие полуслова регистра-приемника.

Команда SHADD8:

1. складывает каждый байт первого операнда с соответствующим байтом второго операнда;
2. сдвигает полученные результаты сложения вправо на 1 бит, что приводит к их делению на 2;
3. записывает полученные после сдвига байты в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SHADD16 R1, R0      ; Сложение полуслов из регистра R0 с соответствующими
                    ; полусловами регистра R1, запись
                    ; результата, деленного пополам, в
                    ; соответствующие полуслова регистра R1.
SHADD8 R4, R0, R5   ; Сложение байт регистра R0 с
                    ; соответствующими байтами регистра
                    ; R5 и запись результата,
                    ; деленного пополам, в соответствующие
                    ; байты регистра R4.
```

9.4.11 **SHASX и SHSAX**

9.4.11.1 Сложение и вычитание целых чисел со знаком с перестановкой и делением пополам, вычитание и сложение целых чисел со знаком с перестановкой и делением пополам.

Синтаксис

op{cond} {*Rd*}, *Rn*, *Rm*

где:

op - операция:

−SHASX - сложение и вычитание целых чисел со знаком с перестановкой и делением пополам;

−SHSAX - вычитание и сложение целых чисел со знаком с перестановкой и делением пополам;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда SHASX:

1. складывает старшее полуслово первого операнда с младшим полусловом второго операнда;

2. записывает результат сложения полуслов, свдинутый вправо на 1 бит, в старшее полуслово регистра-приемника;

3. вычитает старшее полуслово второго операнда из младшего полуслова первого операнда;

4. записывает результат вычитания полуслов, сдвинутый вправо на 1 бит, в младшее полуслово регистра-приемника.

Команда SHSAX:

1. вычитает младшее полуслово второго операнда из старшего полуслова первого операнда;

2. записывает результат вычитания полуслов, сдвинутый вправо на 1 бит, в младшее полуслово регистра-приемника;

3. складывает младшее полуслово первого операнда со старшим полусловом второго операнда;

4. записывает результат сложения полуслов, сдвинутый вправо на 1 бит, в старшее полуслово регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SHASX R7, R4, R2    ; Складывает старшее полуслово регистра R4 с младшим
                    ; полусловом регистра R2 и записывает результат, деленный
                    ; пополам, в старшее полуслово регистра R7
                    ; Вычитает старшее полуслово регистра R2 из младшего
                    ; полуслова регистра R4 и записывает половинный результат
                    ; в младшее полуслово регистра R7
SHSAX R0, R3, R5    ; Вычитает младшее полуслово регистра R5 из старшего
                    ; полуслова регистра R3 и записывает половинный результат
                    ; в старшее полуслово регистра R0
                    ; Складывает старшее полуслово регистра R5 с младшим
                    ; полусловом регистра R3 и записывает результат, деленный
                    ; пополам, в старшее полуслово регистра R0
```

9.4.12 SHSUB16 и SHSUB8

9.4.12.1 Полуразность 16-разрядных целых чисел со знаком, полуразность 8-разрядных целых чисел со знаком.

Синтаксис

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– SHSUB16 - полуразность 16-разрядных целых чисел со знаком;

– SHSUB8 - полуразность 8-разрядных целых чисел со знаком;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда SHSUB16:

1. вычитает каждое полуслово второго операнда из соответствующего полуслова первого операнда;

2. записывает результаты вычитаний полуслов, сдвинутые вправо на 1 бит, в соответствующие полуслова регистра-приемника;

Команда SHSUB8:

1. вычитает каждый байт второго операнда из соответствующего байта первого операнда;
2. записывает результаты вычитаний байт, сдвинутые вправо на 1 бит, в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SHSUB16 R1, R0      ; Вычитает полуслова регистра R0 из соответствующих
                    ; полуслов регистра R1 и сдвинутые на 1 бит
                    ; вправо результаты записывает в соответствующие
                    ; полуслова регистра R1
SHSUB8  R4, R0, R5  ; Вычитает байты регистра R5 из соответствующих байт
                    ; регистра R0 и сдвинутые на 1 бит
                    ; вправо результаты записывает в соответствующие
                    ; байты регистра R4.
```

9.4.13 **SSUB16 и SSUB8**

9.4.13.1 Вычитание 16-разрядных целых чисел со знаком, вычитание 8-разрядных целых чисел со знаком.

Синтаксис

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– SSUB16 - вычитание 16-разрядных целых чисел со знаком;

– SSUB8 - вычитание 8-разрядных целых чисел со знаком;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда SSUB16:

1. вычитает каждое полуслово второго операнда из соответствующего полуслова первого операнда;
2. записывает результаты вычитаний полуслов в соответствующее полуслова регистра-приемника.

Команда SSUB8:

1. вычитает каждый байт второго операнда из соответствующего байта первого операнда;
2. записывает результаты вычитаний байт в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SSUB16  R1, R0      ; Вычитает полуслова регистра R0 из соответствующих
                    ; полуслов регистра R1 и записывает
                    ; результаты в соответствующие
                    ; полуслова регистра R1
SSUB8   R4, R0, R5  ; Вычитает байты регистра R5 из соответствующих байт
```

; регистра R0 и записывает
; результаты в соответствующие
; байты регистра R4.

9.4.14 **SASX и SSAX**

9.4.14.1 Сложение и вычитание чисел со знаком с перестановкой, вычитание и сложение чисел со знаком с перестановкой.

Синтаксис

op{cond} {*Rd*}, *Rn*, *Rm*

где:

op - операция:

– SASX - сложение и вычитание чисел со знаком с перестановкой;

– SSAX - вычитание и сложение чисел со знаком с перестановкой;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда SASX:

1) прибавляет старшее полуслово первого операнда к младшему полуслову второго операнда;
2) записывает результат сложения в старшее полуслово регистра-приемника;
3) вычитает старшее полуслово второго операнда из младшего полуслова первого операнда;
4) записывает результат вычитания в младшее полуслово регистра-приемника.

Команда SSAX:

1) вычитает младшее полуслово второго операнда из старшего полуслова первого операнда;
2) записывает результат вычитания в старшее полуслово регистра приемника;
3) прибавляет старшее полуслово второго операнда к младшему полуслову первого операнда;
4) записывает результат сложения в младшее полуслово регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SASX R0, R4, R5     ; Сложение старшего полуслова регисра R4
                    ; и младшего полуслова регистра R5,
                    ; запись результата в старшее полуслово регистра R0,
                    ; вычитание старшего полуслова регистра R5
                    ; из младшего полуслова регистра R4,
                    ; и запись результата в младшее полуслово регистра R0
SSAX R7, R3, R2     ; Вычитание младшего полуслова регистра R2
                    ; из старшего полуслова регистра R3,
                    ; запись результата в старшее полуслово регистра R7
                    ; Сложение младшего полуслова регистра R3
                    ; и старшего полуслова регистра R2
                    ; и запись результата в младшее полуслово регистра R7.
```

9.4.15 **TST и TEQ**

9.4.15.1 Проверка значений бит по маске, проверка равенства.

Синтаксис

TST{*cond*} *Rn*, *Operand2*

TEQ{*cond*} *Rn*, *Operand2*

где:

cond - необязательный суффикс условия выполнения;

Rn - регистр, содержащий первый операнд;

Operand2 - второй операнд.

Описание

Эти команды выполняют сравнение содержимого регистра *Rn* с учетом знечния второго операнда *Operand2*. Они изменяют флаги условий в соответствии с результатом сравнения, но не сохраняют результат в регистре.

Команда TST выполняет побитную операцию логического И между содержимым *Rn* и значением *Operand2*. Она аналогична команде ANDS, за исключением того, что результат операции не сохраняется.

Для проверки состояния отдельного бита *Rn* используют команду TST, второй операнд которой является константой. В этой константе проверяемый бит установлен в 1, а остальные биты сброшены в 0.

Команда TEQ выполняет побитную операцию Исключающее ИЛИ между содержимым *Rn* и значением *Operand2*. Она аналогична команде EORS, за исключением того, что результат операции не сохраняется.

Команда TEQ используется для проверки равенства двух значений, не изменяя при этом флагов C или V.

Команду TEQ так же можно использовать для проверки знака числа. После сравнения флаг N будет равен результату операции Исключающее ИЛИ между знаковыми битами обоих операндов.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Команды TST и TEQ:

– изменяют флаги N и Z в соответствии с результатом операции;

– могут изменять состояние флага C при вычислении Operand2;

– не влияют на состояние флага V.

Примеры

```
TST R0, #0x3F8      ; Выполняет побитовое AND регистра R0 и 0x3F8,
                    ; регистр APSR изменяется, но результат не сохраняется
TEQEQ R10, R9       ; Условно проверяет эквивалентность значения регистра
                    ; R10 и значения регистра R9, регистр
                    ; APSR изменяется, но результат не сохраняется.
```

9.4.16 **UADD16 и UADD8**

9.4.16.1 Беззнаковое сложение 16-битных целых чисел, беззнаковое сложение 8-битных целых.

Синтаксис

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– UADD16 - сложение двух беззнаковых 16-битных целых чисел;

– UADD8 - сложение четырех беззнаковых 8-битных целых чисел;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Эти команды выполняют параллельное сложение полуслов или байтов:

Команда UADD16:

1. складывает каждое полуслово первого операнда с соответствующим полусловом второго операнда;
2. беззнаковые результаты сложений записываются в соответствующие полуслова регистра-приемника.

Команда UADD8:

1. складывает каждый байт первого операнда с соответствующим байтом второго операнда;
2. беззнаковые результаты сложений записываются в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
UADD16 R1, R0       ; Сложение полуслов регистра R0 с соответствующими
                    ; полусловами регистра R1, запись результатов сложения в
                    ; соответствующие полуслова регистра R1.
UADD8 R4, R0, R5    ; Сложение байт регистра R0 с соответствующими байтами
                    ; регистра R5, запись в соответствующие байты регистра R4.
```

9.4.17 UASX и USAX

9.4.17.1 Беззнаковое сложение и вычитание с перестановкой, беззнаковое вычитание и сложение с перестановкой.

Синтаксис

op{cond} {*Rd*}, *Rn*, *Rm*

где:

op - операция:

– UASX - беззнаковое сложение и вычитание с перемещением;

– USAX - беззнаковое вычитание и сложение с перемещением;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда UASX:

1. вычитает старшее полуслово второго операнда из младшего полуслова первого операнда;
2. записывает беззнаковый результат вычитания в младшее полуслово регистра приемника;
3. прибавляет е старшее полуслово первого операнда к младшему полуслову второго операнда;
4. записывает беззнаковый результат суммы в старшее полуслово регистра-приемника.

Команда USAX:

1. прибавляет младшее полуслово первого операнда к старшему полуслову второго операнда;

2. записывает беззнаковый результат суммы в младшее полуслово регистра-приемника;

3. вычитает младшее полуслово второго операнда из старшего полуслова первого операнда;

4. записывает беззнаковый результат вычитания в старшее полуслово регистра приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
UASX R0, R4, R5     ; Сложение старшего полуслова регистра R4
                    ; и младшего полуслова регистра R5,
                    ; запись результата в старшее полуслово регистра R0
                    ; вычитание младшего полуслова регистра R5
                    ; из старшего полуслова регистра R4,
                    ; запись результата в младшее полуслово регистра R0
USAX R7, R3, R2     ; Вычитание младшего полуслова регистра R2
                    ; из старшего полуслова регистра R3,
                    ; запись результата в старшее полуслово регистра R7
                    ; Сложение младшего полуслова R3 и старшего
                    ; полуслова регистра R2,
                    ; запись результата в младшее полуслово регистра R7.
```

9.4.18 **UHADD16 и UHADD8**

9.4.18.1 Полусумма 16-разрядных целых чисел без знака, полусумма 8-разрядных целых чисел без знака.

Синтаксис

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– UHADD16 - полусумма 16-разрядных целых чисел без знака;

– UHADD8 - полусумма 8-разрядных целых чисел без знака;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Эти команды выполняют сложение 16 или 8 битных данных, затем перед записью результата в регистр-приемник делят его пополам результат:

Команда UHADD16:

1. складывает каждое полуслово первого операнда с соответствующим полусловом второго операнда;

2. сдвигает результаты сложений вправо на 1 бит, что приводит к их делению 2;

3. записывает результирующие полуслова в соответствующие полуслова регистра-приемника.

Команда UHADD8:

1. складывает каждый байт первого операнда с соответствующим байтом второго операнда;

2. сдвигает результаты сложений вправо на 1 бит, что приводит к их делению на 2;

3. записывает результирующие байты в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
UHADD16 R7, R3      ; Сложение полуслов из регистра R7
                    ; с соответствующими полусловами регистра R3,
                    ; запись результатов, поделенных на 2
                    ; в соответствующие полуслова регистра R7.
UHADD8 R4, R0, R5   ; Сложение байт регистра R0
                    ; с соответствующими байтами регистра R5,
                    ; запись результатов, поделенных на 2 в
                    ; соответствующие байты регистра R4.
```

9.4.19 **UHASX и UHSAX**

9.4.19.1 Сложение и вычитание целых чисел без знака с перестановкой и делением пополам, вычитание и сложение целых чисел без знака с перестановкой и делением пополам.

Синтаксис

op{cond} {*Rd*}, *Rn*, *Rm*

где:

op - операция:

– UHASX - сложение и вычитание целых чисел без знака с перестановкой и делением пополам;

– UHSAX - вычитание и сложение целых чисел без знака с перестановкой и делением пополам;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда UHASX:

1. складывает старшее полуслово первого операнда с младшим полусловом второго операнда;
2. сдвигает, результат сложения вправо на 1 бит, что приводит к делению данных на 2;
3. записывает результирующее полуслово в старшее полуслово регистра-приемника;
4. вычитает старшее полуслово второго операнда из младшего полуслова первого операнда;
5. сдвигает, результат вычитания вправо на 1 бит, что приводит к делению данных на 2;
6. записывает результирующее полуслово в младшее полуслово регистра-приемника.

Команда UHSAX:

1. вычитает младшее полуслово второго операнда из старшего полуслова первого операнда;
2. сдвигает, результат вычитания вправо на 1 бит, что приводит к делению данных на 2;
3. записывает результирующее полуслово в старшее полуслово регистра-приемника;
4. складывает младшее полуслово первого операнда со старшим полусловом второго операнда;
5. сдвигает результат сложения вправо на 1 бит, что приводит к делению данных на 2;

6. записывает результирующее полуслово в младшее полуслово регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
UHASX R7, R4, R2    ; Складывает старшее полуслово регистра R4
                    ; с младшим полусловом регистра R2,
                    ; записывает результат деленный пополам в старшее
                    ; полуслово регистра R7
                    ; Вычитает старшее полуслово регистра R2
                    ; из младшего полуслова регистра R7,
                    ; записывает результат деленный пополам в младшее
                    ; полуслово регистра R7
UHSAX R0, R3, R5    ; Вычитает младшее полуслово регистра R5
                    ; из старшего полуслова регистра R3,
                    ; записывает результат деленный пополам в старшее
                    ; полуслово регистра R0
                    ; Складывает старшее полуслово регсистра R5
                    ; с младшим полусловом регистра R3,
                    ; записывает результат деленный пополам в младшее
                    ; полуслово регистра R0
```

9.4.20 UHSUB16 и UHSUB8

9.4.20.1 Полуразность 16-разрядных целых чисел без знака, полуразность 8-разрядных целых чисел без знака.

Синтаксис

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– UHSUB16 - полуразность 16-разрядных целых чисел без знака;

– UHSUB8 - полуразность 8-разрядных целых чисел без знака;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда UHSUB16:

1. вычитает каждое полуслово второго операнда из соответствующего полуслова первого операнда;

2. сдвигает результаты вычитаний вправо на 1 бит, что приводит к делению данных на 2;

3. записывает результирующие полуслова в соответствующие полуслова регистра-приемника.

Команда UHSUBB8:

1. вычитает каждый байт второго операнда из соответствующего байта первого операнда;

2. сдвигает результаты вычитаний вправо на 1 бит, что приводит к делению данных на 2;

3. записывает результирующие байты в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
UHSUB16 R1, R0      ; Вычитает полуслова регистра R0 из соответствующих
                    ; полуслов регистра R1, результаты, поделенные
                    ; на 2 записывает в соответствующие
                    ; полуслова регистра R1
UHSUB8  R4, R0, R5  ; Вычитает байты регистра R5 из соответствующих байт
                    ; регистра R0, результаты, поделенные на 2 записывает
                    ; в соответствующие байты регистра R4.
```

9.4.21 **SEL**

9.4.21.1 Выбирает байты. В качестве результата выбирает байты из первого или второго операнда в зависимости от значений флага GE.

Синтаксис

SEL{<c>}{<q>} {<Rd>,} <Rn>, <Rm>

где:

<c>, <q> - поля со стандартным ассемблерным синтаксисом;

<Rd> - регистр-приемник;

<Rn> - регистр, содержащий первый операнд;

<Rm> - регистр, содержащий второй операнд.

Описание

Команда SEL:

1) читает значение каждого бита в APSR.GE;

2) в зависимости от значения в APSR.GE, назначает регистру-приемнику значение первого или второго регистра.

Ограничения

Нет.

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SADD16  R0, R1, R2      ; Устанавливает биты GE в зависимости от результата
SEL     R0, R0, R3      ; Выбирает байты из R0 или R3, в зависимости от GE.
```

9.4.22 **USAD8**

9.4.22.1 Безнаковоая сумма абсолютных разностей 8-разрядных целых

Синтаксис

USAD8{*cond*} {*Rd*,} *Rn*, *Rm*

где:

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда USAD8:

- вычитает каждый байт второго операнда из соответствующего байта первого операнда;
- складывает абсолютные значения разностей;
- записывает результат в регистр-приемник.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
USAD8 R1, R4, R0    ; Вычитает каждый байт R0 из соответствующего байта R4
                    ; складывает разности и записывает в R1
USAD8 R0, R5        ; Вычитает каждый байт R5 из соответствующего байта R0,
                    ; складывает разности и записывает в R0.
```

9.4.23 **USADA8**

9.4.23.1 Безнаковоая сумма абсолютных разностей 8-разрядных целых с дополнительным сложением.

Синтаксис

USADA8{*cond*}{*Rd*,} *Rn*, *Rm*, *Ra*

где:

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд;

Ra - регистр, содержащий значение для сложения.

Описание

Команда USADA8:

1. вычитает каждый байт второго операнда из соответствующего байта первого операнда;
2. складывает беззнаковые абсолютные значения разностей;
3. добавляет значение регистра *Ra* к сумме абсолютных разностей;
4. записывает результат в регистр-приемник.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
USADA8      R1, R0, R6      ; Вычитает байты регистра R0 из соответствующих
                            ; байт регистра R1, складывает разности,
                            ; суммирует со значением регистра R6, записывает
                            ; в регистр R1
USADA8      R4, R0, R5, R2      ; Вычитает байты регистра R5
                                ; из соответствующих байт регистра R0
                                ; складывает разности, суммирует со
                                ; значением регистра R2
                                ; записывает результат в регистр R4.
```

9.4.24 **USUB16 и USUB8**

9.4.24.1 Вычитание 16-разрядных целых чисел без знака, вычитание 8-разрядных целых чисел без знака.

Синтаксис:

op{*cond*}{*Rd*,} *Rn*, *Rm*

где:

op - операция:

– USUB16 - вычитание 16-разрядных целых чисел без знака;

– USUB8 - вычитание 8-разрядных целых чисел без знака;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда USUB16:

1. вычитает каждое полуслово второго операнда из соответствующего полуслова первого операнда;

2. записывает результаты вычитания полуслов в оответствующие полуслова регистра-приемника.

Команда USUB8:

1. вычитает каждый байт второго операнда из соответствующего байта первого операнда;

2. записывает результаты вычитаний байт в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
USUB16  R1, R0      ; Вычитает полуслова регистра R0 из соответствующих
                    ; полуслов регистра R1 и записывает
                    ; результаты вычитаний в соответствующие
                    ; полуслова регистра R1
USUB8   R4, R0, R5  ; Вычитает байты регистра R5 из соответствующих байт
                    ; регистра R0 и записывает результаты
                    ; вычитаний в соответствующие байты R4.
```

9.5 Команды умножения и деления

9.5.1 В таблице 9.9 представлены команды умножения и деления.

Таблица 9.9 – Команды умножения и деления

Мнемоника	Краткое описание
MLA	Умножение со сложением (32 x 32 + 32), 32-разрядный результат
MLS	Умножение с вычитанием (32 – 32 x 32), 32- разрядный результат
MUL	Умножение (32 x 32), 32- разрядный результат
SDIV	Деление чисел со знаком
SMLA[B,T]	Умножение чисел со знаком со сложением (16 x 16 + 32), 32-разрядный результат
SMLAD, SMLADX	Умножение чисел со знаком с двойным сложением (16 x 16 + 16 x 16 + 32), 32-разрядный результат
SMLAL	Умножение чисел со знаком со сложением (32 x 32 + 64), 64-разрядный результат
SMLAL[B,T]	Умножение чисел со знаком со сложением (16 x 16 + 64), 64-разрядный результат
SMLALD, SMLALDX	Умножение чисел со знаком с двойным сложением (16 x 16 + 16 x 16 + 64), 64-разрядный результат
SMLAW[B|T]	Умножение чисел со знаком со сложением (32 x 16 + 32), 32-разрядный результат
SMLSD	Вычитание результатов умножений чисел со знаком со сложением (16 x 16 – 16 x 16 + 32), 32-разрядный результат
SMLSLD	Вычитание результатов умножений чисел со знаком со сложением (16 x 16 – 16 x 16 + 64), 64-разрядный результат
SMMLA	Сложение со знаковым старшим значимым словом произведения, (32 x 32 + 32), 32-разрядный результат
SMMLS, SMMLSR	Вычитание знакового старшего значимого слова произведения (32 - 32 x 32), 32-разрядный результат

Продолжение таблицы 9.9

Мнемоника	Краткое описание
SMUAD, SMUADX	Умножение чисел со знаком со сложением (16 x 16 + 16 x 16), 32-разрядный результат
SMUL[B,T]	Умножение чисел со знаком (16 x 16), 32-разрядный результат
SMMUL, SMMULR	Умножение чисел со знаком с выделение старшего знакового слова (32 x 32), 32-разрядный результат
SMULL	Умножение чисел со знаком (32 x 32), 64-разрядный результат
SMULWB, SMULWT	Умножение чисел со знаком (32 x 16), 32-разрядный результат
SMUSD, SMUSDX	Вычитание результатов умножений чисел со знаком (полуслов) (16 x 16 - 16 x 16), 32-разрядный результат
UDIV	Деление чисел без знака
UMAAL	Умножение чисел без знака с двойным сложением (32 x 32 + 32 + 32), 64-разрядный результат
UMLAL	Умножение чисел без знака со сложением (32 x 32 + 64), 64-разрядный результат
UMULL	Умножение чисел без знака (32 x 32), 64-разрядный результат

9.5.2 **MUL, MLA, MLS**

9.5.2.1 Умножение, умножение со сложением, умножение с вычитанием. 32-разрядные операнды, 32-разрядный результат.

Синтаксис

MUL{S}{*cond*} {*Rd*,} *Rn*, *Rm* ; Умножение
MLA{*cond*} *Rd*, *Rn*, *Rm*, *Ra* ; Умножение и сложение
MLS{*cond*} *Rd*, *Rn*, *Rm*, *Ra* ; Умножение и вычитание

где:

cond - необязательный суффикс условия выполнения;

S - необязательный суффикс. При наличии в мнемонике команды суффикса S, состояние флагов условий изменяется в соответствии с результатом операции;

Rd - регистр-приёмник;

Rn, Rm - регистры, содержащий значения для умножения;

Ra - регистр, содержащий значение, к которому должно быть прибавлено или из которого должно быть вычтено произведение.

Описание

Команда MUL перемножает значения регистров *Rn* и *Rm* и помещает 32 младших значащих бита результата в регистр *Rd*.

Команда MLA перемножает значения регистров *Rn* и *Rm*, прибавляет к произведению значение из *Ra*, помещает 32 младших значащих бита результата в регистр *Rd*.

Команда MLS перемножает значения регистров *Rn* и *Rm*, вычитает результат умножения из значения в регистре *Ra,* помещает 32 младших значащих бита результата в регистр *Rd*.

Результат выполнения этих команд не зависит от того, имели операнды знак или нет.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

При использовании суффикса S с командой MUL:

– *Rd*, *Rn* и *Rm* должны находиться в диапазоне R0...R7;

– *Rd* должен быть тем же регистром, что и *Rm*;

– использование суффикса *cond* не допускается.

Флаги условия

Если суффикс S указан, то команда MUL:

– изменяет флаги N и Z в соответствии с результатом операции;

– не влияет на состояние флагов C и V.

Примеры

```
MUL     R10, R2, R5       ; Умножение, R10 = R2 x R5
MLA     R10, R2, R1, R5   ; Умножение со сложением, R10 = (R2 x R1)+ R5
MULS    R0, R2, R2        ; Умножение с изменением флага, R0 = R2 x R2
MULLT   R2, R3, R2        ; Умножение (условное), R2 = R3 x R2
MLS     R4, R5, R6, R7    ; Умножение с вычитанием, R4 = R7 - (R5 x R6)
```

9.5.3 UMULL, UMAAL, UMLAL

9.5.3.1 Умножение чисел без знака, умножение чисел без знака с двойным сложением, умножение чисел без знака со сложением. 32-разрядные операнды, 64-разрядный результат.

Синтаксис

op{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

где:

op - операция:

UMULL - умножение чисел без знака;

UMAAL - умножение чисел без знака с двойным сложением;

UMLAL - умножение чисел без знака со сложением;

cond - необязательный суффикс условия выполнения;

RdHi, *RdLo* - пара регистров-приемников. Для команд UMAAL и UMLAL они также содержат значения для суммирования;

Rn, Rm - регистры, содержащие операнды для умножения.

Описание

Эти команды интерпретируют значения из регистров *Rn* и *Rm* как беззнаковые 32-разрядные целые.

Команда UMULL:

- перемножает два беззнаковых значения регистров *Rn* и *Rm*;
- записывает 32 младших значащих бита результата в *RdLo*;
- записывает 32 старших значащих бита результата в *RdHi*.

Команда UMAAL:

- перемножает два беззнаковых 32-разрядных целых значения регистров *Rn* и *Rm*;
- суммирует беззнаковое 32-разрядное целое в регистре *RdHi* с 64-разрядным результатом умножения;
- суммирует беззнаковое 32-разрядное целое в регистре *RdLo* с 64-разрядным результатом сложения;
- записывает старшие 32 бита результата в *RdHi*;
- записывает младшие 32 бита результата в *RdLo*.

Команда UMLAL:

- перемножает два беззнаковых 32-разрядных целых значения регистров *Rn* и *Rm*;
- суммирует 64-разрядный результат перемножение с 64-разрядным беззнаковым целым, находящимся в паре регистров *RdHi* (старшие 32 бита) и *RdLo* (младшие 32 бита);
- записывает результат суммирования обратно в пару регистров *RdHi* (старшие 32 бита) и *RdLo* (младшие 32 бита).

Ограничения

- Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд);
- *RdHi* и *RdLo* должны быть разными регистрами.

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
UMULL R0, R4, R5, R6    ; Умножает R5 и R6, записывает старшие 32 бита в R4,
                        ; а младшие 32 бита в R0
UMAAL R3, R6, R2, R7    ; Умножает R2 and R7, добавляет R6, добавляет R3,
                        ; записывет старшие 32 бита в R6,
```

```
                                ; младшие 32 бита в R3
UMLAL R2, R1, R3, R5            ; Умножает R5 и R3, добавляет R1:R2,
                                ; записывает R1:R2.
```

9.5.4 **Команды SMLA и SMLAW**

9.5.4.1 Умножение чисел со знаком со сложением.

Синтаксис

op{*XY*}{*cond*} *Rd*, *Rn*, *Rm*

op{*Y*}{*cond*} *Rd*, *Rn*, *Rm*, *Ra*

где

op - операция:

– SMLA - умножение чисел со знаком со сложением (32 + 16 × 16);

X и *Y* - определяют, какая половина регистров источников *Rn* и *Rm* используется в качестве первого и второго операндов для умножения.

Если *X* = *B*, то используется младшее полуслово, биты [15:0] регистра *Rn*.

Если *X* = *T*, то используется старшее полуслово, биты [31:16] регистра *Rn*.

Если *Y* = *B*, то используется младшее полуслово, биты [15:0] регистра *Rm*.

Если *Y* = *T*, то используется старшее полуслово, биты [31:16] регистра *Rm*.

– SMLAW умножение чисел со знаком со сложением (32 + 32 × 16);

Y – определяет, какое полуслово регистра-источника *Rm* используется в качестве второго операнда для умножения.

Если *Y* = *T*, то используется старшее полуслово, биты [31:16] регистра *Rm*.

Если *Y* = *B*, то используется младшее полуслово, биты [15:0] регистра *Rm*;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn, Rm - регистры, содержащие перемножаемые значения;

Ra - регистр, содержащий 32-разрядное число со знаком.

Описание

Команды SMLABB, SMLABT, SMLATB, SMLATT:

– перемножают заданные знаковые полуслова (старшие или младшие значения из *Rn* и *Rm*);

– суммируют результат перемножения со значением *Ra*;

– записывают результат сложения в *Rd*.

Незаданные полуслова регистров *Rn* и *Rm* игнорируются.

Команды SMLAWB и SMLAWT:

1. перемножают 32-разрядное значение в регистре *Rn* с:
   – старшим знаковым полусловом регистра *Rm*, если суффикс команды - *T*;
   – младшим знаковым полусловом регистра *Rm*, если суффикс команды - *B*;
2. суммируют старшие 32 бита 48-разрядного произведения с 32-разрядным знаковым значением регистра *Ra*;
3. записывают результат сложения в *Rd*.

Младшие 16 бит 48-разрядного результата игнорируются.

Если во время прибавления аккумулируемого значения происходит переполнение, команда устанавливает флаг Q в APSR. При перемножении переполнение произойти не может.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд);

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SMLABB R5, R6, R4, R1     ; Перемножает младшие полуслова R6 и R4, добавляет
                          ; R1 и записывает в R5
SMLATB R5, R6, R4, R1     ; Перемножает старшее полуслово R6
                          ; с младшим полусловом R4, добавляет R1 и
                          ; записывает в R5
SMLATT R5, R6, R4, R1     ; Перемножает старшие полуслова R6 и R4, добавляет
                          ; R1 и записывает сумму в R5
SMLABT R5, R6, R4, R1     ; Умножает младшее полуслово R6
                          ; со страшим полусловом
                          ; R4, добавляет R1 и записывает в R5
SMLABT R4, R3, R2         ; Перемножает младшее полуслово R4 со старшим
                          ; полусловом R3, добавляет R2 и записывает в R4
SMLAWB R10, R2, R5, R3    ; Перемножает R2 с младшим полусловом R5, добавляет
                          ; R3 к результату и записывает старшие 32 бита в R10
SMLAWT R10, R2, R1, R5    ; Перемножает R2 со старшим
                          ; полусловом R1, добавляет R5
                          ; и записывает старшие 32 бита в R10.
```

9.5.5 **SMLAD**

9.5.5.1 Умножение чисел со знаком с двойным сложением.

Синтаксис

op{*X*}{*cond*} *Rd*, *Rn*, *Rm*, *Ra*

где:

op - операция:

– SMLAD – умножение чисел со знаком с двойным сложением;

– SMLADX – умножение чисел со знаком с двойным сложением и перестановкой;

X – опеределяет какое полуслово регистра-источника *Rn* используется в качестве операнда при умножении.

Если *X* пропущен, то умножаются полуслова: младший × младший и старший × старший.

Если *X* присутствует, то умножаются полуслова младшее × старшее и старшее × младшее;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn, Rm - регистры, содержащие перемножаемые значения;

Ra - регистр, содержащий 32-разрядное число со знаком.

Описание

Команды SMLAD и SMLADX рассматривают *Rn, Rm* как четыре 16-битных полуслова.

Команды SMLAD и SMLADX:

– если *X* отсутствует, перемножают старшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm* , а младшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm*;

– если *X* присутствует, перемножают старшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm* и младшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm*;

– суммируют оба результата умножения со знаковым 32-разрядным значением в регистре *Ra*;

– записывают знаковый 32-разрядный результат сложения в *Rd*.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SMLAD       R10, R2, R1, R5     ; Перемножаются два полуслова регистра R2 с
                                ; соответствующими полусловами регистра R1,
                                ; суммимруется с регистром R5 и
                                ; записывается в R10
SMLALDX     R0, R2, R4, R6      ; Перемножается старшее полуслово
                                ; регистра R2 с младшим полусловом
                                ; регистра R4, и младшее полуслово R2
                                ; со старшим полусловом R4, суммируется
                                ; с R6 и записывается в R0.
```

9.5.6 SMLAL and SMLALD

9.5.6.1 Умножение чисел со знаком со сложением, умножение чисел со знаком с двойным сложением.

Синтаксис

op{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*
op{XY}{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*
op{*X*}{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

где:

op - операция:

– SMLAL - знаковое умножение со сложением;

– SMLAL - знаковое умножение со сложением (полуслов, опции X и Y);

X и *Y* - определяют, какие полуслова источников *Rn* и *Rm* используются в качестве первого и второго операндов для умножения.

Если *X* = *B*, то используется младшее полуслово, биты [15:0] регистра *Rn*.

Если *X* = *T*, то используется старшее полуслово, биты [31:16] регистра *Rn*.

Если *Y* =, то используется младшее полуслово, биты [15:0] регистра *Rm*.

Если *Y* = *T*, то используется старшее полуслово, биты [31:16] регистра *Rm*;

SMLALD - знаковое умножение с двойным сложением;

SMLALDX - знаковое умножение с двойным сложением с реверсом.

Если *X* пропущен, то умножаются полуслова: младший × младший и старший × старший.

Если *X* присутствует, то умножаются полуслова младшее × старшее и старшее × младшее;

cond - необязательный суффикс условия выполнения;

RdHi, *RdLo* – пара регистров-приемников;

RdLo - младшие 32 бита 64-разрядного целого;

RdHi - старшие 32 бита 64-разрядного целого.

Для команд SMLAL, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLALD и SMLALDX *RdLo* и *RdHi* также содержат значения для суммирования;

Rn, Rm - регистры, содержащие перемножаемые значения.

Описание

Команда SMLAL:

– перемножает два знаковых слова из регистров *Rn* и *Rm*;

– суммирует результат умножения с 64-разрядным значением из пары регистов *RdLo* (младшие 32 бита) и *RdHi* (старшие 32 бита);

– записывает 64-разрядный результат суммирования в пару регистров *RdLo* (младшие 32 бита) и *RdHi* (старшие 32 бита).

Команды SMLALBB, SMLALBT, SMLALTB и SMLALTT:

1. перемножают заданные знаковые полуслова из регистров *Rn* и *Rm*;

2. суммируют 32-разрядный результат умножения с 64-разрядным значением из пары регистров *RdLo* (младшие 32 бита) *и RdHi* (старшие 32 бита);

3. записывают 64-разрядный результат сложения в пару регистров *RdLo* (младшие 32 бита) и *RdHi* (старшие 32 бита).

Незаданные полуслова регистров *Rn* и *Rm* игнорируются.

Команды SMLALD и SMLALDX интерпретируют значения из *Rn* и *Rm* как четыре полуслова.

−если *X* не указано, перемножается старшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm* и младшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm*;

−если *X* указано, перемножается старшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm* и младшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm*;

−суммируются два полученных результата умножения и знаковое 64-рзарядное значение, определяемое парой регистров *RdLo* (младшие 32 бита) и *RdHi* (старшие 32 бита);

−записывает 64-разрядный результат сложения в пару регистров *RdLo* (младшие 32 бита) и *RdHi* (старшие 32 бита).

Ограничения

−Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд);

−*RdHi* и *RdLo* должны быть разными регистрами.

Флаги условия

Эта команда не влияет на состояние флагов.

Примеры

```
SMLAL   R4, R5, R3, R8      ; Умножает R3 на R8, суммирует с парой R5:R4 и
                            ; записывает в пару R5:R4
SMLALBT R2, R1, R6, R7      ; Умножает младшее полуслово R6
                            ; на старшее полуслово R7, расширяет знак
                            ; до 32-бит, суммирует с парой R1:R2
                            ; и записывает в R1:R2
SMLALTB R2, R1, R6, R7      ; умножает старшее полуслово R6
                            ; на младшее полуслово R7, расширяет знак
                            ; до 32-бит, суммирует с парой R1:R2
                            ; и записывает в R1:R2
SMLALD  R6, R8, R5, R1      ; перемножает старшие полуслова R5 и R1, младшие
                            ; полуслова R5 и R1, суммирует
                            ; результаты перемножений, добавляет к паре R8:R6 и
                            ; записывает результат в R8:R6
SMLALDX R6, R8, R5, R1      ; Перемножает старшее полуслово R5 с
                            ; младшим полусловом R1, младшее полуслово R5
                            ; со старшим полусловом R1, суммирует
                            ; результаты перемножений, добавляет
                            ; к паре R8:R6 и записывает в R8:R6.
```

9.5.7 **SMLSD и SMLSLD**

9.5.7.1 Вычитание результатов умножений чисел со знаком с последующим сложением с 32-разрядным знаковым целым числом, вычитание результатов умножений чисел со знаком с последующим сложением с 64-разрядным знаковым целым числом.

Синтаксис

op{*X*}{*cond*} *Rd*, *Rn*, *Rm*, *Ra*

где:

op - операция:

– SMLSD - вычитание результатов умножений чисел со знаком с последующим сложением с 32-разрядным знаковым целым числом;

– SMLSDX - вычитание результатов умножений чисел со знаком с последующим сложением с 32-разрядным знаковым целым числом с перестановкой;

– SMLSLD - вычитание результатов умножений чисел со знаком с последующим сложением с 64-разрядным знаковым целым числом;

– SMLSLDX - вычитание результатов умножений чисел со знаком с последующим сложением с 64-разрядным знаковым целым числом с перестановкой.

Если *X* указано, перемножается младшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm* и старшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm*.

Если *X* не указано, перемножается младшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm* и старшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm*.

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn, Rm - регистры, содержащие перемножаемые значения;

Ra - регистр, содержащий 32-разрядное число со знаком.

Описание

Команда SMLSD интерпретирует значения из *Rn* и *Rm* как четыре знаковых полуслова. Эта команда:

– при наличии X переставляет полуслова второго операнда;

– выполняет перемножение двух пар знаковых полуслов 16 × 16-бит;

– вычитает регультат перемножения верхних полуслов из результата перемножения нижних полуслов;

– суммирует полученную разность со знаковым 32-разрядным значением в регистре *Ra*;

– записывает результат в регистр-приемник.

Команда SMLSLD интерпретирует значения из *Rn* и *Rm* как четыре знаковых полуслова. Эта команда:

– при наличии X переставляет полуслова второго операнда;

– выполняет перемножение двух пар знаковых полуслов 16 × 16-бит;

– вычитает регультат перемножения верхних полуслов из результата перемножения нижних полуслов;

– суммирует полученную разность со знаковым 64-разрядным значение из пары регистров *RdHi* (старшие 32 бита) и *RdLo* младшие 32 бита);

– записывает 64-разрядный результат в пару регистров *RdHi* (старшие 32 бита) и *RdLo* младшие 32 бита).

Ограничения

– Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд);

– *Rd* и *Rn* должны быть разными регистрами.

Флаги условия

Эта команда устанавливает флаг Q, если происходит переполнение при операции суммирования. Переполнения не может произойти при умножении или вычитании.

Для набора инструкций Thumb эти команды не действуют на флаги условного кода.

Примеры

```
SMLSD   R0, R4, R5, R6      ; Перемножает младшее полуслово R4
                            ; с младшим полусловом
                            ; R5, старшее полуслово R4
                            ; со старшим полусловом R5, вычитает
                            ; второе из первого добавляе R6, записывает в R0
SMLSDX R1, R3, R2, R0       ; Перемножает младшее полуслово R3 со старшим
```

```
                                ; полусловом R2, старшее полуслово R3
                                ; с младшим полусловом R2, вычитает
                                ; второе из первого добавляет R0, записывает в R1
SMLSLD  R3, R6, R2, R7          ; Перемножает младшее полуслово R6
                                ; с младшим полусловом R2, старшее полуслово R6
                                ; со старшим полусловом R2, вычитает второе
                                ; из первого, добавляет R6:R3, записывает в R6:R3
SMLSLDX R3, R6, R2, R7          ; Перемножает младшее полуслово R6 с
                                ; старшим полусловом R2, старшее полуслово R6
                                ; с младшим полусловом R2, вычитает
                                ; второе из первого, добавляет R6:R3,
                                ; записывает в R6:R3
```

9.5.8 **SMMLA и SMMLS**

9.5.8.1 Сложение со знаковым старшим значимым словом произведения, вычитание знакового старшего значимого слова произведения.

32 ± (32 x 32) = 32 (старшие разряды).

Синтаксис

op{*R*}{*cond*} *Rd*, *Rn*, *Rm*, *Ra*

где:

op - операция:

– SMMLA - сложение со знаковым старшим значимым словом произведения;

– SMMLS - вычитание знакового старшего значимого слова произведения;

R - флаг ошибки округления. Если *R* задан, то вместо усечения результат округляется. В этом случае перед извлечением старшего значащего слова результат складывается с константой 0x80000000;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn, Rm - регистры, содержащие перемножаемые значения;

Ra - регистр, содержащий 32-разрядное число со знаком.

Описание

Команда SMMLA интерпретирует значения *Rn* и *Rm* как знаковые 32-разрядные слова. Команда SMMLA:

– перемножает значения в *Rn* и *Rm*;

– при наличии опции R округляет результат прибавлением к произведению константы 0x80000000;

– извлекает старшие значащие 32 бита результата;

– суммирует извлеченное значение и значение в регистре *Ra*;

– записывает результат суммирования в *Rd*.

Команда SMMLS интерпретирует значения *Rn* и *Rm* как знаковые 32-разрядные слова. Команда SMMLS:

– перемножает значения в *Rn* и *Rm*;

– при наличии опции R округляет результат прибавлением к произведению константы 0x80000000;

– извлекает старшие значащие 32 бита результата;

– вычитает извлеченное значение из значения регистра *Ra*;

– записывает результат вычитания в *Rd*.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на установку флагов.

Примеры

```
SMMLA R0, R4, R5, R6            ; Перемножает R4 и R5, извлекает старшие
                                ; 32 бита, добавляет R6, отбрасывает и записывает в R0
```

```
SMMLAR R6, R2, R1, R4     ; Перемножает R2 и R1, извлекает старшие
                          ; 32 бита, добавляет R4, округляет и записывает в R6
SMMLSR R3, R6, R2, R7     ; Перемножает R6 и R2, извлекает старшие 32 бита,
                          ; вычитает R7, округляет и записывает в R3
SMMLS R4, R5, R3, R8      ; Перемножает R5 и R3, извлекает старшие 32 бита,
                          ; вычитает R8, отбрасывает и записывает в R4.
```

9.5.9 **SMMUL**

9.5.9.1 Умножение чисел со знаком с выделение старшего знакового слова.

Синтаксис

op{*R*}{*cond*} *Rd*, *Rn*, *Rm*

где:

op - операция:

SMMUL - знаковое умножение старших слов;

R - флаг ошибки округления. Если *R* задан, то вместо усечения результат округляется. В этом случае перед извлечением старшего значащего слова к результату вычислений прибавляется константа 0x80000000;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn, Rm - регистры, содержащие перемножаемые значения.

Операция

Команда SMMUL интерпретирует значения *Rn* и *Rm* как знаковые 32-разрядные слова.

Команда SMMUL:

– перемножает значения в *Rn* и *Rm*;

– при наличии опции R округляет результат прибавлением к произведению константы 0x80000000;

– извлекает старшие значащие 32 бита результата;

– записывает результат в регистр *Rd*.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на установку флагов.

Примеры

```
SMULL R0, R4, R5          ; Перемножает R4 и R5, извлекает старшие 32 бита
                          ; и записывает в R0
SMULLR R6, R2             ; Перемножает R2 и R6, извлекает старшие 32 бита,
                          ; округляет и записывает в R6
```

9.5.10 **SMUAD и SMUSD**

9.5.10.1 Умножение чисел со знаком со сложением, вычитание результатов умножений чисел со знаком.

Синтаксис

op{X}{*cond*} *Rd*, *Rn*, *Rm*

где:

op - операция:

– SMUAD - умножение чисел со знаком со сложением;

– SMUADX - умножение чисел со знаком со сложением с перестановкой;

– SMUSD - вычитание результатов умножений чисел со знаком;

– SMUSDX - вычитание результатов умножений чисел со знаком с перестановкой.

Если *X* указано, перемножается младшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm* и старшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm*.

Если *X* не указан, перемножается младшее знаковое полуслово регистра *Rn* с младшим знаковым полусловом регистра *Rm* и старшее знаковое полуслово регистра *Rn* со старшим знаковым полусловом регистра *Rm*;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn, Rm - регистры, содержащие перемножаемые значения.

Описание

Команда SMUAD интерпретирует значения из регистров *Rn* и *Rm* как два знаковых целых полуслова в каждом регистре.

Эта команда:

– при наличии X переставляет полуслова регистра *Rm*;

– выполняет перемножение двух пар знаковых полуслов 16 × 16-бит;

– суммирует результаты перемножения;

– записывает результат сложения в регистр-приемник.

Команда SMUSD интерпретирует значения из регистров *Rn* и *Rm* как два знаковых целых полуслова в каждом регистре.

Эта команда:

– по наличию X переставляет полуслова регистра *Rm*;

– выполняет перемножение двух пар знаковых полуслов 16 × 16-бит;

– вычитает результат умножения верхних полуслов из результата перемножения нижних полуслов;

– записывает результат вычитания в регистр-приемник.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Устанавливают флаг Q, если происходит переполнение при суммировании. Перемножение не может дать переполнение.

Примеры

```
SMUAD R0, R4, R5    ; Перемножает младшее полуслово R4 с младшим
                    ; полусловом R5, складывает с произведением старшего
                    ; полуслова R4 со старшим полусловом R5,
                    ; записывает в R0
SMUADX R3, R7, R4   ; Перемножает младшее полуслово R7
                    ; со старшим полусловом R4,
                    ; складывает с произведением старшего R7
                    ; с младшим полусловом R4, записывает в R3
SMUSD R3, R6, R2    ; Перемножает младшее полуслово R4
                    ; с младшим полусловом R6, вычитает
                    ; произведение старшего полуслова R6 со
                    ; старшим полусловом R3, записывает в R3
SMUSDX R4, R5, R3   ; Перемножает младшее полуслово R5
                    ; со старшим полусловом R3, вычитает
                    ; произведение старшего полуслова R5
                    ; с младшим полусловом R3, записывает в R4.
```

9.5.11 **SMUL и SMULW**

9.5.11.1 Умножение чисел со знаком (16 x 16), умножение чисел со знаком (32 x 16).

Синтаксис

op{*XY*}{*cond*} *Rd*,*Rn*, *Rm*

op{*Y*}{*cond*} *Rd*. *Rn*, *Rm*

Только для *SMULXY*:

op - операция:

– SMUL{*XY*} - Знаковое перемножение полуслов;

X и *Y* определяют какие полуслова регистров источников *Rn* и *Rm* используются в качестве первого и второго операндов при умножении.

Если *X* = B, то используется младшее полуслово, биты [15:0] регистра *Rn*.

Если *X* = T, то используется старшее полуслово, биты [31:16] регистра *Rn*.

Если *Y* = B, то используется младшее полуслово, биты [15:0] регистра *Rm*.

Если *Y* = T, то используется старшее полуслово, биты [31:16] регистра *Rm*;

– SMULW{Y} Знаковое перемножение слова и полуслова;

Y определет какое полуслово регистра *Rm* используется в качестве второго операнда.

Если *Y* = B, то используется младшее полуслово, биты [15:0] регистра *Rm*.

Если *Y* = T, то используется старшее полуслово, биты [31:16] регистра *Rm*;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn, Rm - регистры, содержащие перемножаемые значения.

Описание

Команды SMULBB, SMULTB, SMULBT и SMULTT интерпретируют значения регистров *Rn* и *Rm* как четыре знаковых целых 16-разрядных числа.

Эти команды:

– перемножают заданные знаковые полуслова регистров *Rn* и *Rm*;

– записывает 32-разрядный результат умножения в *Rd.*

Команды SMULWT и SMULWB интерпретируют значение регистра *Rn* как 32-разрядное целое и значение регистра *Rm* как два полуслова с 16-разрядными знаковыми целыми.

Эти команды:

– перемножают значение регистра *Rn* со старшим (при наличии суффикса T), или с младшим (при наличии суффикса B) полусловом регистра *Rm*;

– записывает знаковые старшие значащие 32 бита из 48-разрядного результата в регистр-приемник.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на установку флагов.

Примеры

```
SMULBT R0, R4, R5   ; Перемножает младшее полуслово R4 со страшим
                    ; полусловом R5, результат записывается в R0
SMULBB R0, R4, R5   ; Перемножает младшее полуслово R4
                    ; и младшее полуслово R5, результат записывается в  R0
SMULTT R0, R4, R5   ; Перемножает старшее полуслово R4
                    ; со старшим полусловом R5, результат
                    ; записывается в R0
SMULTB R0, R4, R5   ; Перемножает старшее полуслово R4
                    ; с младшим полусловом R5, результат
                    ; записывается в R0
SMULWT R4, R5, R3   ; Умножает R5 на старшее полуслово R3,
                    ; выделяет старших 32 бита и записывает в R4
SMULWB R4, R5, R3   ; Умножает R5 на младшее полуслово R3,
                    ; выделяет старших 32 бита и записывает в R4.
```

9.5.12 **UMULL, UMLAL, SMULL**

9.5.12.1 Умножение чисел без знака, умножение чисел без знака со сложением, умножение чисел со знаком.

Синтаксис

op{*cond*} *RdLo*, *RdHi*, *Rn*, *Rm*

где:

op - операция:

– UMULL - умножение чисел без знака;

– UMLAL - умножение чисел без знака со сложением;

– SMULL - умножение чисел со знаком;

cond - необязательный суффикс условия выполнения;

RdHi, *RdLo* – пара регистров-приемников. Для команды UMLAL они так же содержат значения для суммирования;

Rn, Rm - регистры, содержащие перемножаемые значения.

Описание

Команда UMULL интерпретирует значения регистров *Rn* и *Rm* как целые числа без знака. Она перемножает эти целые числа и размещает младшие значащие 32 бита результата в регистре *RdLo*, а старшие значащие 32 бита результата в регистре *RdHi*.

Команда UMLAL интерпретирует значения регистров *Rn* и *Rm* как целые числа без знака. Она перемножает эти целые числа, суммирует произведение с 64-разрядным числом без знака, содержащимся в регистрах *RdHi* (старшие 32 бита) и *RdLo* (младшие 32 бита) и записывает результат обратно в *RdHi* (старшие 32 бита) и *RdLo* (младшие 32 бита).

Команда SMULL интерпретирует значения регистров *Rn* и *Rm* как целые числа со знаком. Она перемножает эти целые числа и размещает младшие значащие 32 бита произведения в *RdLo*, а старшие значащие 32 бита произведения в *RdHi*.

Ограничения

– Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

– *RdHi* и *RdLo* должны быть разными регистрами.

Флаги условия

Эта команда не влияет на установку флагов.

Примеры

```
UMULL R0, R4, R5, R6        ; R4,R0 = R5 x R6 (без знака)
SMLAL R4, R5, R3, R8        ; R5,R4 = (R5,R4) + R3 x R8 (со знаком)
```

9.5.13 **Команды SDIV и UDIV**

9.5.13.1 Деление чисел со знаком, деление чисел без знака.

Синтаксис

SDIV{*cond*} {*Rd*,} *Rn*, *Rm*

UDIV{*cond*} {*Rd*,} *Rn, Rm*

где:

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник. Если *Rd* отсутствует, то регистром приемником служит *Rn*;

Rn - регистр, содержащий делимое;

Rm - регистр, содержащий делитель.

Описание

SDIV выполняет деление знакового целого значения регистра *Rn* на знаковое целое значение регистра *Rm*.

UDIV выполняет деление целого значения регистра *Rn* без знака на значение регистра *Rm* без знака. Для обеих команд, если значение в *Rn* не делится на значение в *Rm*, результат округляется в сторону нуля.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Эта команда не влияет на установку флагов.

Примеры:

```
SDIV R0, R2, R4      ; R0 = R2/R4 (со знаком)
UDIV R8, R8, R1      ; R8 = R8/R1 (без знака)
```

9.6 Команды насыщения

9.6.1 В таблице 9.10 перечислены команды насыщения (фиксации крайних значений).

Таблица 9.10 – Команды насыщения

Мнемоника	Краткое описание
SSAT	Преобразование 32-разрядного числа в n-разрядное со знаком
SSAT16	Преобразование двух 16-разрядных чисел в n-разрядные со знаком
USAT	Преобразование 32-разрядного числа в n-разрядное без знака
USAT16	Преобразование двух 16-разрядных чисел в n-разрядные без знака
QADD	Сложение чисел со знаком с насыщением результата
QSUB	Вычитание чисел со знаком с насыщением результата
QASX	Сложение и вычитание чисел со знаком с перестановкой и насыщением результата
QSUB16	Вычитание 16-разрядных чисел со знаком с насыщением результата
QSAX	Вычиатние и сложение чисел со знаком с перестановкой и насыщением результата
QDADD	Удвоение и сложение чисел со знаком с насыщением результата
QDSUB	Удвоение и вычитание чисел со знаком с насыщением результата
UQADD16	Сложение 16-разрядных целых чисел без знака с насыщением результата
UQADD8	Сложение 8-разрядных целых чисел без знака с насыщением результата
UQASX	Сложение и вычитание чисел без знака с перестановкой и насыщением результата
UQSAX	Вычитание и сложение чисел без знака с перестановкой и насыщением результата
UQSUB16	Вычитание 16-разрядных целых чисел без знака с насыщением результата
UQSUB8	Вычитание 8-разрядных целых чисел без знака с насыщением результата

Знаковое *n*-битное насыщение означает:

– если значение, которое должно быть насыщено, меньше чем -2^{n-1}, то результатом будет -2^{n-1};

– если значение, которое должно быть насыщено больше чем $2^{n-1}-1$, то результатом будет $2^{n-1}-1$;

– в остальных случаях, результатом остается тоже самое значение, которое подвергалось насыщению.

Беззнаковое *n*-битное насыщение означает:

– если значение, которое должно быть насыщено меньше чем 0, то результатом будет 0;

– если значение, которое должно быть насыщено больше чем $2^{n}-1$, то результатом будет $2^{n}-1$;

– в остальных случаях, результатом остается тоже самое значение, которое подвергалось насыщению.

Если возвращаемое значение отличается от того, которое подвергалось насыщению, то оно называется насыщенным. Если насыщение произошло, то команда устанавливает флаг Q в 1 в APSR. В противном случае, флаг Q не изменяется.

Чтобы очистить флаг Q в 0, необходимо использовать команду MSR.

Чтобы прочитать состояние флага Q необходимо использовать команду MRS.

9.6.2 **SSAT и USAT**

9.6.2.1 Преобразование 32-разрядного числа в n-разрядное со знаком, преобразование 32-разрядного числа в n-разрядное без знака.

Синтаксис

op{*cond*} *Rd*, #*n*, *Rm* {, *shift* #s}

где:

op - операция:

– SSAT - преобразование 32-разрядного числа в n-разрядное со знаком;

– USAT - преобразование 32-разрядного числа в n-разрядное без знака;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник.;

n - определяет позицию бита для насыщения:

– *n* в диапазоне от 1 до 32 для команды SSAT;

– *n* в диапазон от 0 до 31 для кманды USAT;

Rm - регистр, содержащий значение для насыщения;

shift #s - необязательный сдвиг в *Rm* перед операцией насыщения. Он должен быть одним из:

– ASR #*s* где *s* число в диапазоне от 1 до 31;

– LSL #*s* где *s* число в диапазоне от 0 до 31.

Описание

Эти команды преобразуют 32-разрядное число в n-разрядное число со знаком или без знака. Преобразование осуществляется с насыщением результата.

Команда SSAT выполняет заданный сдвиг, затем насыщает в знаковом диапазоне $-2^{n-1} \le x \le 2^{n-1}-1$.

Команда USAT выполняет заданный сдвиг, затем насыщает в беззнаковом диапазоне $0 \le x \le 2^{n}-1$.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Если при преобразовании данных выполнилось насыщение, то эти команды устанавливают флаг Q в 1.

Примеры

```
SSAT R7, #16, R7, LSL #4        ; Логический сдвиг влево на 4 в R7, затем
                                ; приведение его к 16-разрядному
                                ; числу со знаком с насыщением,
                                ; запись обратно в R7
USATNE R0, #7, R5               ; Условное насыщение значения
                                ; в R5 к 7-разрядному числу без знака,
                                ; запись результата в R0.
```

9.6.3 **SSAT16 и USAT16**

9.6.3.1 Преобразование двух 16-разрядных чисел в n-разрядные со знаком, преобразование двух 16-разрядных чисел в n-разрядные без знака.

Синтаксис:

op{*cond*} *Rd*, #*n*, *Rm*

где:

op - операция:

– SSAT16 - преобразование двух 16-разрядных чисел в n-разрядные со знаком;

– USAT16 - преобразование двух 16-разрядных чисел в n-разрядные без знака;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

n - определяет позицию бита для насыщения:

– *n* в диапазоне от 1 до 16 для команды SSAT;

– *n* в диапазон от 0 до 15 для кманды USAT;

Rm - регистр, содержащий значение для насыщения.

Описание

Команда SSAT16:

1. выполняет преобразование двух 16-разрядных чисел со знаком в регистре *Rm* в n-разрядные числа со знаком с насыщением результата;

2. записывает результат как два 16-разрядных числа со знаком в регистр-приемник.

Команда USAT16:

1. выполняет преобразование двух 16-разрядных чисел без знака в регистре *Rm* в n-разрядные числа без знака с насыщением результата;

2. записывает результат как два 16-разрядных числа без знака в регистр-приемник.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Команды не влияют на флаги кода условий.

Если насыщение выполнилось, то устанавливается флаг Q в 1.

Примеры

```
SSAT16 R7, #9, R2        ; Насыщает старшее и младшее полуслова регистра R2
                         ; как 9-разрядные значения, записывает
                         ; результат преобразования в R7
USAT16NE R0, #13, R5     ; По условию насыщает старшее и младшее полуслова
                         ; регистра R5 как 13-разрядные значения,
                         ; записывает результаты преобразований в R0.
```

9.6.4 **Команды QADD и QSUB**

9.6.4.1 Сложение чисел со знаком с насыщением результата, вычитание чисел со знаком с насыщением результата.

Синтаксис

op{*cond*} {*Rd*}, *Rn*, *Rm*

op{*cond*} {*Rd*}, *Rn*, *Rm*

где:

op - операция:

– QADD – сложение 32-разрядных чисел со знаком с насыщением;

– QADD8 – сложение четырех 8-разрядных чисел со знаком с насыщением;

– QADD16 – сложение двух 16-разрядных чисел со знаком с насыщением;

– QSUB – вычитание 32-разрядных чисел со знаком с насыщением;

– QSUB8 – вычитание четырех 8-разрядных чисел со знаком с насыщением;

– QSUB16 – вычитание 16-разярдных чисел со знаком с насыщением;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Эти команды складывают или вычитают два, четыре или восемь значений из первого и второго операндов, а затем записывают насыщенные значения со знаком в регистр-приемник.

Команды QADD и QSUB выполняют суммирование или вычитание, а затем насыщают результат до знакового диапазона $-2^{n-1} \le x \le 2^{n-1}-1$.

Если возвращаемое значение отличается от того, которое подвергалось насыщению, то оно называется насыщенным.

Если насыщение произошло, то команда устанавливает флаг Q в 1 в APSR. В противном случае, флаг Q не изменяется. 8-разрядные и 16-разрядные команды QADD и QSUB всегда сохраняют флаг Q неизмененным.

Чтобы очистиь флаг Q в 0, необходимо использовать команду MSR. Чтобы прочитать состояние флага Q необходимо использовать команду MRS.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Команды не влияют на флаги кода условий.

Если насыщение выполнилось, то устанавливается флаг Q в 1.

Примеры

```
QADD16 R7, R4, R2   ; Складывает полуслова R4 с
                    ; соответствующими полусловами
                    ; R2, насыщает до 16 бит и
                    ; записывает результаты преобразования
                    ; в соответствующие полуслова регистра R7
QADD8 R3, R1, R6    ; Складывает байты R1 с соответствующими
                    ; байтами R6, насыщает до 8 бит и записывает
                    ; результаты насыщения в соответствующие
                    ; байты регистра R3
QSUB16 R4, R2, R3   ; Вычитает полуслова R3 из соответствующих полуслов
                    ; R2, насыщает до 16 бит,
                    ; записывает результаты насыщения в соответствующие
                    ; полуслова регистра R4
QSUB8 R4, R2, R5    ; Вычитает байты R5 из соответствующих байт
                    ; R2, насыщает до 8 бит ,
                    ; записывает результаты насыщения в соответствующие
                    ; байты регистра R4.
```

9.6.5 **Команды QASX и QSAX**

9.6.5.1 Сложение и вычитание чисел со знаком с перестановкой и насыщением результата, вычиатние и сложение чисел со знаком с перестановкой и насыщением результата.

Синтаксис

op{cond} {*Rd*}, *Rm*, *Rn*

где:

op - операция:

−QASX – сложение и вычитание чисел со знаком с перестановкой и насыщением результата;

−QSAX – вычитание и сложение чисел со знаком с перестановкой и насыщением результата;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание

Команда QASX:

1. суммирует старшее полуслово первого операнда с младшим полусловом второго операнда;

2. вычитает старшее полуслово второго операнда из младшего полуслова первого операнда;

3. насыщает результат вычитания до 16-разрядного целого со знаком в диапазоне $-2^{15} \le x \le 2^{15} - 1$ и записывает в младшее полуслово регистра-приемника;

4. насыщает результата сложения до 16-разрядного целого со знаком в диапазоне $-2^{15} \le x \le 2^{15} - 1$ и записывает в старшее полуслово регистра-приемника.

Команда QSAX:

1. вычитает младшее полуслово второго операнда из старшего полуслова первого операнда;

2. суммирует младшее полуслово первого операнда со старшим полусловом второго операнда;

3. насыщает результат сложения до 16-разрядного целого со знаком в диапазоне $-2^{15} \le x \le 2^{15} - 1$ и записывает в младшее полуслово регистра-приемника;

4. насыщает результат вычитания до 16-разрядного целого со знаком в диапазоне $-2^{15} \le x \le 2^{15} - 1$ и записывает в старшее полуслово регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Команды не влияют на флаги кода условий.

Примеры

```
QASX R7, R4, R2     ; Сложение старшего полуслова R4
                    ; с младшим полусловом R2,
                    ; насыщение до 16 бит, запись в старшее
                    ; полуслово регистра R7
                    ; Вычитает старшее полуслово R2 из младшего полуслова
                    ; регистра R4, насыщает до 16 бит и записывает в младшее
                    ; полуслово регистра R7
QSAX R0, R3, R5     ; Вычитает младшее полуслово регистра R5 из старшего
                    ; полуслова R3, насыщает до 16 бит, записывает в старшее
                    ; полуслово R0
                    ; Складывает младшее полуслово R3 со старшим полусловом
                    ; R5, насыщает до 16 бит, записывает
                    ;в младшее полуслово R0.
```

9.6.6 **Команды QDADD и QDSUB**

9.6.6.1 Удвоение и сложение чисел со знаком с насыщением результата, удвоение и вычитание чисел со знаком с насыщением результата.

Синтаксис

op{*cond*} {*Rd*}, *Rm*, *Rn*

где:

op - операция:

– QDADD – удвоение и сложение чисел со знаком с насыщением результата;

– QDSUB – удвоение и вычитание чисел со знаком с насыщением результата;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rm - регистр, содержащий второй операнд;

Rn - регистр, содержащий первый операнд.

Описание

Команда QDADD:

– удваивает значение второго операнда;

– насыщает результат удвоения и суммирует его с первым операндом;

– записывает результат в регистр-приемник.

Команда QDSUB:

1. удваивает значение второго операнда;
2. вычитает результат удвоения из первого операнда с насыщеннием результата;
3. записывает результат в регистр-приемник.

Результаты удвоения и операций сложения/вычитания насыщаются до 32-битного целого со знаком в диапазоне $-2^{31} \le x \le 2^{31} - 1$.

Если в какой либо из операций произведено насыщение, то команда устанавливает флаг Q в APSR.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Если насыщение выполнилось, то эти команды устанавливают флаг Q в 1.

Примеры

```
QDADD R7, R4, R2    ; Удваивает и насыщает R4 до 32 бит, складывает с R2,
                    ; насыщает до 32 , записывает в R7
QDSUB R0, R3, R5    ; Вычитает удвоенное и насыщенное до
                    ; 32 бит значение R3 из R5, насыщает
```

; до 32 бит, записывает R0.

9.6.7 **UQASX и UQSAX**

9.6.7.1 Сложение и вычитание чисел без знака с перестановкой и насыщением результата, вычитание и сложение чисел без знака с перестановкой и насыщением результата.

Синтаксис:

op{cond} {*Rd*}, *Rm*, *Rn*

где:

op - операция:

– UQASX – сложение и вычитание чисел без знака с перестановкой и насыщением результата;

– UQSAX - вычитание и сложение чисел без знака с перестановкой и насыщением результата;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Описание:

Команда UQASX:

1. складывает старшее полуслово первого операнда с младшим полусловом второго операнда;

2. вычитает старшее полуслово второго операнда из младшего полуслова первого операнда;

3. насыщает результат вычитания до 16-разрядного целого без знака в диапазоне $0 \le x \le 2^{16} - 1$ и записывает в младшее полуслово регистра-приемника;

4. насыщает результат суммы до 16-разрядного целого без знака в диапазоне $0 \le x \le 2^{16} - 1$ и записывает в старшее полуслово регистра-приемника.

Команда UQSAX:

1. вычитает младшее полуслово второго операнда из старшего полуслова первого операнда;

2. складывает младшее полуслово первого операнда со старшим полусловом второго операнда;

3. насыщает результат сложения до 16-разрядного целого без знака в диапазоне $0 \le x \le 2^{16} - 1$ и записывает в младшее полуслово регистра-приемника;

4. насыщает результат вычитания до 16-разрядного целого без знака в диапазоне $0 \le x \le 2^{16} - 1$ и записывает в старшее полуслово регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Команды не действуют на флаги кода условий.

Примеры

```
UQASX R7, R4, R2    ; Складыает старшее полуслово R4 с младшим полусловом R2,
                    ; насыщает до 16 бит, записывает в старшее полуслово R7
                    ; Вычитает старшее полуслово R2 из младшего полуслова
                    ; R4, насыщает до 16 бит, записывает в младшее полуслово R7
UQSAX R0, R3, R5    ; Вычитает младшее полуслово R5 из старшего полуслова R3,
                    ; насыщает до 16 бит, записывает в старшее полуслово R0
                    ; Складывает младшее полуслово R3 со старшим полусловом
                    ; R5, насыщает до16 бит, записывает в младшее полуслово R0.
```

9.6.8 **Команды UQADD и UQSUB**

9.6.8.1 Удвоение и сложение чисел со знаком с насыщением результата, удвоение и вычитание чисел со знаком с насыщением результата.

Синтаксис

op{*cond*} {*Rd*}, *Rn*, *Rm*

op{*cond*} {*Rd*}, *Rn*, *Rm*

где:

op - операция:

– UQADD8 – сложение четырех 8-разрядных чисел без знака с насыщением;

– UQADD16 – сложение двух 16-разрядных чисел без знака с насыщением;

– UQSUB8 – вычитание четырех 8-разрядных чисел без знака с насыщением;

– UQSUB16 – вычитание двух 16-разрядных чисел без знака с насыщением;

cond - необязательный суффикс условия выполнения;

Rd - регистр-приемник;

Rn - регистр, содержащий первый операнд;

Rm - регистр, содержащий второй операнд.

Операция

Эти команды складывают или вычитают два или четыре числе, затем записывают насыщенные значения без знака в регистр-приемник.

Команда UQADD16:

1. складывает соответствующие старшие и младшие полуслова первого и второго операндов;

2. насыщает результаты сложения до целых без знака в диапазоне $0 \le x \le 2^{16} - 1$ и записывает результаты преобразования в соответствующие полуслова регистра-приемника.

Команда UQADD8:

1. складывает соответствующие байты первого и второго операндов;

2. насыщает результаты суммы до целых без знака в диапазоне $0 \le x \le 2^8 - 1$ и записывает результаты преобразования в соответствующие байты регистра-приемника.

Команда UQSUB16:

1. вычитает полуслова второго операнда из соответствующих полуслов первого операнда;

2. насыщает результаты вычитаний до целых без знака в диапазоне $0 \le x \le 2^{16} - 1$ и записывает результаты преобразований в соответствующие полуслова регистра-приемника.

Команда UQSUB8:

1. вычитает байты второго операнда из соответствующих байтов первого операнда;

2. насыщает результаты вычитаний до целых без знака в дипазоне $0 \le x \le 2^8 - 1$ и записывает результаты преобразований в соответствующие байты регистра-приемника.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги условия

Команды не действуют на флаги кода условий.

Примеры

```
UQADD16 R7, R4, R2 ; Складывает полуслова в R4 с соответствующими,
                   ; полусловами в R2, насыщает до 16 бит, записывает в
                   ; соответствующие полуслова R7
UQADD8 R4, R2, R5  ; Складывает байты R2 с соответствующими
                   ; байтами R5, насыщает до 8 бит, записывает в
                   ; соответствующие байты R4
UQSUB16 R6, R3, R0 ; Вычитает полуслова R0 из соответствующих полуслов
                   ; в R3, насыщает до 16 бит, записывает в
                   ;  соответствующие полуслова в регистр R6
```

```
UQSUB8 R1, R5, R6   ; Вычитает байты R6 из соответствующих
                    ; байтов регистра R5, насыщает до 8 бит, записывает
                    ; в соответствующие байты R1.
```

9.7 Команды упаковки и распаковки

9.7.1 В таблице 9.11 приведены команды, которые управляют упаковкой и распаковкой данных.

Таблица 9.11 – Команды упаковки и распаковки

Мнемоника	Краткое описание
PKH	Упаковать полуслово
SXTAB	Расширение байта со знаком в слово со сложением
SXTAB16	Двойное расширение байтов со знаком в полуслова со сложением
SXTAH	Расширение полуслова со знаком в слово со сложением
SXTB	Расширение байта со знаком в слово
SXTB16	Двойное расширение байтов со знаком в полуслова
SXTH	Расширение полуслова со знаком в слово
UXTAB	Расширение байта без знака в слово со сложением
UXTAB16	Двойное расширение байтов без знака в полуслова со сложением
UXTAH	Расширение полуслова без знака в слово со сложением
UXTB	Расширение байта без знака в слово
UXTB16	Двойное расширение байтов без знака в полуслова
UXTH	Расширение полуслова без знака в слово

9.7.2 **Команды PKHBT и PKHTB**

9.7.2.1 Упаковать полуслово.

Синтаксис

op{cond} {Rd}, Rn, Rm {, LSL #imm}

op{cond} {Rd}, Rn, Rm {, ASR #imm}

где:

op - может быть:

– PKHBT - упаковать полуслово, нижнее и верхнее со сдвигом;

– PKHTB - упаковать полуслово, верхнее и нижнее со сдвигом;

cond - дополнительный код условия;

Rd - регистр назначения;

Rn - регистр первого операнда;

Rm - регистр второго операнда, содержащий значение, которое может быть дополнительно сдвинуто;

Imm - длина сдвига. Тип длины сдвига зависит от инструкции.

Для PKHBT:

LSL сдвиг влево с длиной сдвига от 1 до 31, 0 означает без сдвига.

Для PKHTB:

ASR арифметический сдвиг вправо с длиной сдвига от 1 до 32, сдвиг на 32 бита кодируется как 0b00000.

Описание

Инструкция PKHBT:

1) записывает значение младшего полуслова первого операнда в младшее полуслово регистра-приемника;

2) если задано смещение, сдвинутое значение второго операнда записано в старшее полуслово регистра-приемника.

Инструкция PKHTB:

1) записывает значение старшего полуслова первого операнда в старшее полуслово регистра-приемника;

2) если задано смещение, сдвинутое значение второго операнда записывает в младшее полуслово регистра-приемника.

3)

Ограничения

Нельзя использовать регистры SP (указатель стека) и PC (счетчик команд) в качестве *Rd*.

Флаги состояний

Эта инструкция не меняет состояние флагов.

Примеры

```
PKHBT R3, R4, R5 LSL #0     ; Записывает нижнее полуслово R4
                            ; в нижнее полуслово R3,
                            ; записывает верхнее полуслово R5,
                            ; без сдвига, в верхнее полуслово R3.
PKHTB R4, R0, R2 ASR #1     ; Записывает R2, сдвинутое вправо на 1 бит, в нижнее
                            ; полуслово R4, и записывает верхнее полуслово R0 в
                            ; верхнее полуслово R4.
```

9.7.3 **Команды SXT и UXT**

9.7.3.1 Расширение чисел со знаком.

Синтаксис

op{cond} {Rd,} Rm {, ROR #n}
op{cond} {Rd}, Rm {, ROR #n}

где:

op - является одним из:

- SXTB – расширение байта со знаком в слово;
- SXTH – расширение полуслова со знаком в слово;
- SXTB16 – двойное расширение байтов со знаком в полуслова;
- UXTB – расширение байта без знака в слово;
- UXTH – расширение полуслова без знака в слово;
- UXTB16 – двойное расширение байтов без знака в полуслова;

cond - является дополнительным кодом условия;

Rd – регистр-приемник;

Rm - регистр содержащий значение для расширения;

- ROR *#n* - является одним из:
- ROR #8 - циклический сдвиг значения из Rm вправо на 8 бит;
- ROR #16 - циклический сдвиг значения из Rm вправо на 16 бит;
- ROR #24 - циклический сдвиг значения из Rm вправо на 24 бита.

Если ROR *#n* опущено, циклический сдвиг не выполняется.

Описание

Эти инструкции выполняют следующее:

1) выполняет циклический сдвиг значения из Rm вправо на 0, 8, 16 или 24 бита;

2) извлекает биты из полученного значения:

- SXTB извлекает биты [7:0] и преобразует их в 32-разрядное число со знаком;
- UXTB извлекает биты [7:0] и преобразует их в 32-разрядное число без знака;
- SXTH извлекает биты [15:0] и преобразует их в 32-разрядное число со знаком;
- UXTH извлекает биты [15:0] и преобразует их в 32-разрядное число без знака;
- SXTB16 извлекает биты [7:0], преобразует их в 16-разрядное число со знаком, извлекает биты [23:16] и преобразует их в 16-разрядное число со знаком;
- UXTB16 извлекает биты [7:0], преобразует их в 16-разрядное число без знака, извлекает биты [23:16] и преобразует их в 16-разрядное число со знаком.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги состояний

Эта инструкция не меняет состояние флагов.

Примеры

```
SXTH R4, R6, ROR #16      ; Выполняет циклический сдвиг R6 вправо
                          ; на 16 бит, получает нижнее полуслово
                          ; результата, расширяет знаком до 32
                          ; разрядов и записывает в R4
UXTB R3, R10              ; Извлекает нижний байт значения в R10,
                          ; расширяет нулем, и
                          ; записывает в R3.
```

9.7.4 **Команды SXTA и UXTA**

9.7.4.1 Расширение числе со знаком и без знака со сложением.

Синтаксис

op{cond} {Rd,} Rn, Rm {, ROR *#n}*

op{cond} {Rd,} Rn, Rm {, ROR *#n}*

где:

op - может быть одним из:

- SXTAB - расширение байта со знаком в слово со сложением;
- SXTAH - расширение полуслова со знаком в слово со сложением;
- SXTAB16 - двойное расширение байтов со знаком в полуслова со сложением;
- UXTAB - расширение байта без знака в слово со сложением;
- UXTAH - расширение полуслова без знака в слово со сложением;
- UXTAB16 - двойное расширение байтов без знака в полуслова со сложением;

cond - является дополнительным кодом условия;

Rd - является регистром назначения;

Rn - является регистром первого операнда;

Rm - является регистром, содержащим значение для циклического сдвига и расширения;

ROR *#n* является одним из:

ROR #8 - циклический сдвиг значения из Rm вправо на 8 бит;

ROR #16 - циклический сдвиг значения из Rm вправо на 16 бит;

ROR #24 - циклический сдвиг значения из Rm вправо на 24 бит.

Описание

Эти инструкции выполняют следующее:

1. выполняет циклический сдвиг значения из Rm вправо на 0, 8, 16 или 24 бита;
2. извлекает биты из полученного значения:

- SXTAB извлекает биты [7:0] и преобразует их в 32-разрядное число со знаком;
- UXTAB извлекает биты [7:0] и преобразует их в 32-разрядное число без знака;
- SXTAH извлекает биты [15:0] и преобразует их вы 32-разрядное число со знаком;
- UXTAH извлекает биты [15:0] и преобразует их вы 32-разрядное число без знака;
- SXTAB16 извлекает биты [7:0], преобразует их в 16-разрядное число со знаком, извлекает биты [23:16] из *Rm* , преобразует в 16-разрядное число со знаком;
- UXTAB16 извлекает биты [7:0], преобразует их в 16-разрядное число без знака, извлекает биты [23:16] из *Rm* , преобразует в 16-разрядное число без знака;

3. прибавляет расширенное знаком или нулем значение к слову или соответствующему полуслову *Rn* и записывает результат в *Rd*.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги состояний

Эта инструкция не меняет состояние флагов.

Примеры

```
SXTAH R4, R8, R6, ROR #16 ; Выполняет циклический сдвиг R6 вправо
                          ; на 16 бит,
                          ; получает нижнее полуслово, расширяет
                          ; знаком до 32 бит, прибавляет R8,
```

```
                                ; и записывает в R4.
UXTAB R3, R4, R10               ; Извлекает нижний байт R10 и расширяет
                                ; нулем до 32 бит, прибавляет R4, и
                                ; записывает в R3.
```

9.8 Команды битовых полей

9.8.1 В таблице 9.12 приведены инструкции, которые обрабатывают смежные наборы бит в регистрах или битовых полях.

Таблица 9.12 – Инструкции битовых полей

Мнемоника	Краткое описание
BFC	Запись нуля в битовое поле
BFI	Запись заданного значения битового поля
SBFX	Чтение значения битового поля, интерпретируемого как число со знаком
SXTB	Преобразовать байт со знаком в слово
SXTH	Преобразовать полуслово со знаком в слово
UBFX	Чтение значения битового поля, интерпретируемого как число без знака
UXTB	Преобразовать байт без знака в слово
UXTH	Преобразовать полуслово без знака в слово

9.8.2 **Команды BFC и BFI**

9.8.2.1 Очистка битового поля, запись заданного значения битового поля.

Синтаксис

BFC*{cond} Rd, #lsb, #width*

BFI*{cond} Rd, Rn, #lsb, #width*

где:

cond - является дополнительным кодом условия;

Rd – регистр-приемник;

Rn - регистр-источник;

lsb - позиция младшего значащего разряда битового поля. lsb должен быть в диапазоне от 0 до 31;

width - ширина битового поля, значения которой должны быть в диапазоне от 1 до 32 - lsb.

Описание

BFC очищает битовое поле регистра *Rd*, имеющее длину width бит, начиная с бита с номером lsb. Остальные биты *Rd* остаются без изменения.

BFI копирует битовое поле в один регистр из другого. Команда заменяет *width* бит *Rd*, начиная с бита с номером *lsb*, width битами из *Rn*, начиная с бита [0]. Остальные биты *Rd* остаются без изменения.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги состояний

Эта инструкция не меняет состояние флагов.

Примеры

```
BFC R4, #8, #12                 ; Очищает биты с 8 по 19 (12 бит) R4 в 0.
BFI R9, R2, #8, #12             ; Замещает биты с 8 по 19 (12 бит) R9 значениями
                                ; бит с 0 по 11 из R2.
```

9.8.3 **Команды SBFX и UBFX**

9.8.3.1 Чтение значения битового поля, интерпретируемого как число со знаком или без знака.

Синтаксис

SBFX*{cond} Rd, Rn, #lsb, #width*

UBFX*{cond} Rd, Rn, #lsb, #width*

где:

cond - является дополнительным кодом условия;

Rd – регистр-приемник;

Rn - регистр-источник;

lsb - позиция младшего значащего разряда битового поля. *lsb* должен быть в диапазоне от 0 до 31;

width - ширина битового поля и должен быть в диапазоне от 1 до 32 - *lsb*.

Описание

SBFX извлекает битовое поле из регистра-источника, преобразует его в 32-разрядное число со знаком и записывает результат в регистр-приемник.

UBFX извлекает битовое поле из одного регистра, преобразует его в 32-разрядное число без знака и записывает результат в регистр назначения.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги состояний

Эта инструкция не меняет состояние флагов.

Примеры

```
SBFX R0, R1, #20, #4        ; Извлекает биты с 20 по 23 (4 бита) из R1 и расширяет
                            ; знаком до 32 бит и затем записывает результат в R0.
UBFX R8, R11, #9, #10       ; Извлекает биты с 9 по 18 (10 бит) из R11 и расширяет
                            ; нулем до 32 бит и затем записывает результат в R8.
```

9.8.4 **Команды SXT и UXT**

9.8.4.1 Преобразование байта или полуслова в слово со знаком или без знака.

Синтаксис

SXT*extend{cond} {Rd,} Rm {,* ROR *#n}*

UXT*extend{cond} {Rd}, Rm {,* ROR *#n}*

где:

extend - является одним из:

– B – преобразование 8-разрядного значения в 32-разрядное;

– H – преобразование 16-разрядного значения в 32-разрядное;

cond - является дополнительным кодом условия;

Rd – регистр-приемник;

Rm – регистр, содержащий значение для расширения;

ROR *#n* является одним из:

– ROR #8 – циклический сдвиг значения из Rm вправо на 8 бит;

– ROR #16 – циклический сдвиг значения из Rm вправо на 16 бит;

– ROR #24 – циклический сдвиг значения из Rm вправо на 24 бита.

Если ROR *#n* опущено, циклический сдвиг не выполняется.

Описание

Эти инструкции выполняют следующее:

1. выполняет циклический сдвиг значения из Rm вправо на 0, 8, 16 или 24 бита;
2. извлекает биты из полученного значения:

– SXTB извлекает биты [7:0] и преобразует их в 32-разрядное число со знаком;

– UXTB извлекает биты [7:0] и преобразует их в 32-разрядное число без знака;

– SXTH извлекает биты [15:0] и преобразует их в 32-разрядное число со знаком;

– UXTH извлекает биты [15:0] и преобразует их в 32-разрядное число без знака.

Ограничения

Нельзя использовать регистры SP (указатель стека) либо PC (счетчик команд).

Флаги состояний

Эта инструкция не меняет состояние флагов.

Примеры

```
SXTH R4, R6, ROR #16        ; Выполняет циклический сдвиг R6 вправо на 16 бит,
                            ; получает нижнее полуслово результата, расширяет
                            ; знаком до 32 разрядов и записывает в R4
UXTB R3, R10                ; Извлекает нижний байт значения в R10,
                            ; расширяет нулем, и записывает в R3.
```

9.9 Команды передачи управления

9.9.1 В таблице 9.13 показаны инструкции ветвления и управления:

Таблица 9.13 – Инструкции ветвления и управления

Мнемоника	Краткое описание
B	Переход
BL	Переход со связью
BLX	Косвенный переход со связью
BX	Косвенный переход
CBNZ	Сравнение с нулем и переход по неравенству
CBZ	Сравнение с нулем и переход по равенству
IT	Начало блока условно исполняемых конструкций
TBB	Табличный переход по индексу, смещения – байты
TBH	Табличный переход по индексу, смещения – полуслова

9.9.2 **Команды B, BL, BX и BLX**

9.9.2.1 Команды ветвления.

Синтаксис

B*{cond} label*

BL*{cond} label*

BX*{cond} Rm*

BLX*{cond} Rm*

где:

B - переход (прямой);

BL - переход со связью (прямой);

BX – косвенный переход по адресу, заданному значением регистра;

BLX – косвенный переход со связью;

cond - дополнительный код условия;

label – относиельный адрес (относительно PC).

Rm - регистр, содержащий адрес, на который необходимо передать управление. Бит[0] значения в *Rm* должен быть 1, но передача управления будет выполнена по адресу, соответствующему нулевому значению бита [0].

Описание

Все эти инструкции вызывают переход на *label*, или на адрес, указанный в *Rm*. Кроме того:

−инструкции BL и BLX записывают адрес следующей инструкции в LR (регистр ссылки, R14);

−инструкции BX и BLX вызывает исключение UsageFault, если бит[0] *Rm* равен 0.

Команда вида *B cond label* является условной инструкцией, которая может быть как внутри, так и вне IT-блока. Все остальные условно исполняемые инструкции перехода должны быть внутри блока IT, а за пределами этого блока должны использоваться только в безусловной форме.

В таблице 9.14 приведены диапазоны для различных инструкций перехода.

Таблица 9.14 – Диапазоны переходов

Инструкция	Диапазон перехода
B label	−16 MB до +16 MB относительно текущей позиции
Bcond label (вне блока IT)	−1 MB до +1 MB относительно текущей позиции
Bcond label (в блоке IT)	−16 MB до +16 MB относительно текущей позиции
BL{cond} label	−16 MB до +16 MB относительно текущей позиции
BX{cond} Rm	Любое значение, записанное в регистре
BLX{cond} Rm	Любое значение, записанное в регистре
Примечание: Необходимо использовать суффикс .W для получения максимального диапазона перехода.	

Ограничения

– в команде BLX не допускается использование регистра PC;

– в командах BX и BLX, бит[0] регистра Rm должен быть установлен в 1, но переход происходит на целевой адрес, соответствующий нулевому значению бита[0];

– любая из этих инструкций, находящаяся внутри блока IT, должна быть последней.

Примечание - *Bcond* является единственной условной инструкцией, применение которой допустимо за пределами блока IP. Однако, она имеет более широкий диапазон переходов, если расположена внутри блока IT.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
B loopA         ; Перейти на метку loopA
BLE ng          ; Условно перейти к метке ng
B.W target      ; Перейти на метку target в диапазоне +/- 16MB
BEQ target      ; Условно перейти на метку target
BEQ.W target    ; Условно перейти на метку target в диапазоне +/- 1MB
BL funC         ; Перейти со ссылкой (вызов функции) funC, адрес возврата
                ; будет записан в LR
BX LR           ; Возврат из вызова функции
BXNE R0         ; Условно перейти к адресу, сохраненному в R0
BLX R0          ; Перейти со ссылкой (вызов функции) по адресу
                ; сохраненному в R0.
```

9.9.3 **Команды CBZ и CBNZ**

9.9.3.1 Сравнение и условная передача управлени по равенству или неравенству нулю.

Синтаксис

CBZ *Rn, label*

CBNZ *Rn, label*

где:

Rn – регистр, содержащий операнд;

Label – метка, на которую должен быть осуществлен переход.

Описание

Инструкции CBZ или CBNZ позволяют выполнить проверку на равенство нулю с условным переходом, при этом не влияя на значения флагов и снижая общее количество инструкций.

CBZ *Rn, label* не изменяет флаги состояния и в остальном эквивалентна следующей последовательности:

– CMP Rn, #0

– BEQ *label*

CBNZ *Rn, label* не изменяет флаги состояния и в остальном эквивалентна следующей последовательности:

– CMP Rn,#0

– BNE *label*

Ограничения

– *Rn* должен быть регистром из диапазона R0…R7;

– адрес перехода должен быть расположен на расстоянии от 4 до 130 байт;

– данные инструкции нельзя использовать внутри блока IT.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
CBZ R5, target      ; Переход вперед, если R5 равен нулю
CBNZ R0, target     ; Переход вперед, если R0 не равен нулю
```

9.9.4 **Команда IT**

9.9.4.1 Начало блока условно исполняемых инструкций.

Синтаксис

IT*{x{y{z}}} cond*

где:

x определяет выбор условия для второй инструкции в блоке IT;

y определяет выбор условия для третьей инструкции в блоке IT;

z определяет выбор условия для четвертой инструкции в блоке IT;

cond - определяет условие для первой инструкции в блоке IT.

Суффиксы выбора условия для второй, третьей и четвертой инструкции блока IT может быть:

T - Then. Инструкция выполняется, если условие *cond* истинно;

E - Else. Инструкция выполняется, если условие *cond* ложно.

Примечание - в инструкции IT существует возможность использовать AL (*всегда истинное*) на месте *cond*. В этом случае все инструкции в блоке IT должны быть безусловными, и каждый из *x*, *y* и *z* должны быть T или пропущен.

Описание

Инструкция IT содержит до четырех следующих за ней инструкций. Условия могут быть либо все одинаковыми либо некоторые из них могут быть логически противоположны. Условные инструкции, следующие за инструкцией IT, формируют блок IT.

Инструкции в блоке IT, включая любые переходы, должны определять условие в части {*cond*} их синтаксиса.

Примечание:

Ассемблеры некоторых производителей способны автоматически генерировать необходимыеинструкции IT, предшествующие условно исполняемым командам, избавляя разработчика от необходимости делать эту работу вручную. Подробности следует уточнить в документации на Ваш ассемблер.

Инструкция BKPT в блоке IT, всегда выполняются, незаивисимо от истинности её условия.

Между инструкцией IT и соответствующим блоком IT, или внутри блока IT может быть вызвана обработка исключения. В результате возникновения таких исключений выполнения передается на соответствующий обработчик исключения, с предварительным сохранением регистра PSR в стеке и и необходимой информации в регистре LR.

Инструкции спроектированы для возможности использования возврата из исключения как обычного для возврата из исключения, и правильного возобновления выполнения блока IT. Это единственный допустимый способ передачи управления внутрь блока IT с помощью команд, модицицирующих счетчик команд PC.

Ограничения

Следующие инструкции не разрешены в блоке IT:

– IT;

– CBZ и CBNZ;

– CPSID и CPSIE;

Переход или любая инструкция, которая изменяет PC, должна передавать управление за пределы блока IT, либо на последнюю инструкцию блока IT. Инструкции, модифицирующие счетчик команд:

– ADD PC, PC, *Rm*;

– MOV PC, *Rm*;

– B, BL, BX, BLX;

– любая инструкция LDM, LDR или POP, выполняющая запись в PC;

– TBB и TBH;

Не допускается пердавать управление на инструкцию внутри блока IT, за исключением случаев возврата из обработчика исключения;

Все условные инструкции за исключением B cond должны быть внутри блока IT. Bcond может быть как вне, так и внутри блока IT, но имеет более широкий диапазон перехода, если располагается внутри;

Каждая инструкция внутри блока IT должна определять суффикс условного исполнения с кодом, который может быть либо таким же либо логически противоположным коду условия блока IT.

Примечание:

Ассемблер может помещать дополнительные ограничения на использование блоков IT, таких как запрещение использования директив ассемблера внутри них.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
ITTE NE          ; Следующие 3 инструкции являются условными
ANDNE R0, R0, R1 ; ANDNE не обновляют флаги состояния
ADDSNE R2, R2, #1 ; ADDSNE обновляет флаги состояния
MOVEQ R2, R3     ; Условное копирование
CMP R0, #9       ; Преобразовать шестнадцатеричное значение (от 0 до 15) R0 в
                 ; ASCII ('0'-'9', 'A'-'F')
ITE GT           ; Следующие 2 инструкции являются условными
ADDGT R1, R0, #55 ; Преобразует 0xA -> 'A'
ADDLE R1, R0, #48 ; Преобразует 0x0 -> '0'
IT GT            ; Блок IT с одной условной инструкцией
ADDGT R1, R1, #1 ; Увеличить R1 на 1 при условии
ITTEE EQ         ; Следующие 4 инструкции являются условными
MOVEEQ R0, R1    ; Условное копирование
ADDEQ R2, R2, #10 ; Условное сложение
ANDNE R3, R3, #1 ; Условное логическое И
BNE.W dloop      ; Инструкция перехода. Должна быть только
                 ; последней инструкцией блока IT
IT NE            ; Следующая инструкция является условной
ADD R0, R0, R1   ; Ошибка синтаксиса: не используется код условия в блоке IT
```

9.9.5 **Команды TBB и TBH**

9.9.5.1 Табличный переход по индексу.

Синтаксис

TBB *[Rn, Rm]*

TBH *[Rn, Rm,* LSL *#1]*

где:

Rn - регистром, содержащий адрес таблицы длин переходов.

Если *Rn* является PC, то первый байт таблицы переходов следует непосредственно после инструкции TBB или TBH;

Rm – регистр, содержащий индекс в таблице переходов. Для таблиц, содержащих полуслова, добавляется операция сдвига LSL #1, что обеспечивает корректную адресацию по смещению в таблице.

Описание

Данные инструкции позволяют выполнить переход вперед относительно текущего значения счетчика команд PC на заданное смещение, выбранное из таблицы смещений, имеющих размер байта (для команды TBB) или полуслова (для команды TBH).

Регистр Rn содержит указатель на начало таблицы, а регистр Rm - индекс требуемого элемента.

Для команды TBB смещение вычисляется путем умножения на два значения байта из заданной ячейки таблицы, интерпретируемого как целое число без знака.

Для команды TBH смещение вычисляется путем умножения на два значения полуслова из заданной ячейки таблицы, интерпретируемого как целое число без знака.

Передача управления по соответствующему смещению осуществляется немедленно после выполнения инструкции TBB или TBH.

Ограничения

– в качестве регистра *Rn* нельзя использовать SP;

– в качестве регистра *Rm* нельзя использовать SP и PC;

– когда любая из этих инструкций используется внутри блока IT, они должны быть последней инструкцией блока.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
ADR.W R0, BranchTable_Byte
TBB [R0, R1]                ; R1 является индексом, R0 является основным
                            ; адресом таблицы переходов
Case1
                            ; последовательность инструкций
Case2
                            ; последовательность инструкций
Case3
                            ; последовательность инструкций
BranchTable_Byte
DCB 0                       ; вычисление смещения Case1
DCB ((Case2-Case1)/2)       ; вычисление смещения Case2
DCB ((Case3-Case1)/2)       ; вычисление смещения Case3

TBH [PC, R1, LSL #1]        ; R1 является индексом, PC используется в качестве
                            ; основного адреса таблицы переходов
BranchTable_H
DCI ((CaseA - BranchTable_H)/2)     ; вычисление смещения CaseA
DCI ((CaseB - BranchTable_H)/2)     ; вычисление смещения CaseB
DCI ((CaseC - BranchTable_H)/2)     ; вычисление смещения CaseC
CaseA
                            ; последовательность инструкций
CaseB
                            ; последовательность инструкций
CaseC
                            ; последовательность инструкций
```

9.10 Команды с плавающей точкой

9.10.1 В таблице 9.15 показаны инструкции с плавающей точкой.

Таблица 9.15 – Инструкции с плавающей точкой

Мнемоника	Краткое описание
VABS	Модуль числа с плавающей точкой
VADD	Сложение чисел с плавающей точкой
VCMP	Сравнение двух регистров с плавающей точкой, или регистра с плавающей точкой и нуля
VCMPE	Сравнение двух регистров с плавающей точкой, или регистра с плавающей точкой и нуля с проверкой на недопустимость операции
VCVT	Преобразование между числами с плавающей точкой и целым
VCVT	Преобразование между числами с плавающей точкой и с фиксированной точкой
VCVTR	Преобразование между числами с плавающей точкой и целым с округлением
VCVTB	Преобразование между значением с половинной точностью и значением с одинарной точностью
VCVTT	Преобразование значения с одинарной точностью в значение с половинной точностью
VDIV	Деление с плавающей точкой
VFMA	Совмещенное умножение чисел с плавающей точкой со сложением
VFNMA	Совмещенное умножение чисел с плавающей точкой со сложением и инверсией
VFMS	Совмещенное умножение чисел с плавающей точкой с вычитанием
VFNMS	Совмещенное умножение чисел с плавающей точкой с вычитанием и инверсией
VLDM	Загрузка нескольких регистров расширения с плавающей точкой
VLDR	Загрузка регистра расширения из памяти
VLMA	Умножение двух чисел с плавающей точкой с накоплением
VLMS	Умножение двух чисел с плавающей точкой с вычитанием
VMOV	Перемещение непосредственно числа с плавающей точкой в регистр
VMOV	Перемещение регистра с плавающей точкой в регистр
VMOV	Скопировать регистр ядра ARM в регистр с плавающей точкой с одинарной точностью
VMOV	Скопировать 2 регистра ядра ARM в два регистра с плавающей точкой с одинарной точностью
VMOV	Скопировать из регистра ядра ARM в скаляр
VMOV	Скопировать скаляр в регистр ядра ARM
VMRS	Переместить из FPSCR в регистр ядра ARM
VMSR	Переместить из регистра ядра ARM в FPSCR
VMUL	Умножение чисел с плавающей точкой
VNEG	Инверсия числа с плавающей точкой
VNMLA	Умножение с плавающей точкой с последующим сложением и сменой знака
VNMLS	Умножение с плавающей точкой с последующим вычитанием и сменой знака
VNMUL	Умножение с плавающей точкой с последующей сменой знака
VPOP	Извлечение из стека регистров расширения
VPUSH	Помещение в стек регистров расширения
VSQRT	Вычисление квадратного корня из числа с плавающей точкой
VSTM	Сохранение нескольких регистров расширения
VSTR	Сохранение регистра расширения в память
VSUB	Вычитание чисел с плавающей точкой

9.10.2 **Команда VABS**

9.10.2.1 Модуль числа с плавающей точкой.

Синтаксис

VABS*{cond}*.F32 *Sd, Sm*

где:

cond – дополнительный код условия;

Sd, Sm – результирующее значение и операнд с плавающей точкой.

Описание

Эта инструкция:

1) берет модуль значения операнда из регистра с плавающей точкой;
2) помещает результат с плавающей точкой в регистр назначения.

Ограничения

Ограничений нет.

Флаги состояния

Инструкция с плавающей точкой очищает бит знака.

Примеры

VABS.F32 S4, S6

9.10.3 **Команда VADD**

9.10.3.1 Сложение чисел с плавающей точкой.

Синтаксис

VADD*{cond}*.F32 *{Sd,} Sn, Sm*

где:

cond – дополнительный код условия;

Sd – результирующее значения с плавающей точкой;

Sn, Sm – значения операндов с плавающей точкой.

Описание

Эта инструкция:

1. складывает значения двух регистров операндов с плавающей точкой;
2. помещает результат в регистр назначения с плавающей точкой.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

VADD.F32 S4, S6, S7

9.10.4 **Команды VCMP, VCMPE**

9.10.4.1 Сравнение двух регистров с плавающей точкой, или регистра с плавающей точкой и нуля, сравнение двух регистров с плавающей точкой, или регистра с плавающей точкой и нуля с проверкой на недопустимость операции.

Синтаксис

VCMP*{E}{cond}*.F32 *Sd, Sm*

VCMP*{E}{cond}*.F32 *Sd*, #0.0

где:

cond – дополнительный код условия;

E – если установлен, любой операнд NaN вызывает исключение Invalid Operation. В противном случае, только сигнализация NaN вызывает исключение;

Sd - является операндом с плавающей точкой для сравнения;

Sm - является операндом с плавающей точкой, с которым выполняется сравнение.

Описание

Эта инструкция:

1. сравнивает:

– два регистра с плавающей точкой;

– один регистр с плавающей точкой и ноль;

2. записывает результат во флаг FPSCR.

Ограничения

Эта инструкция может дополнительно генерировать исключение Invalid Operation, если хотя бы один операнд содержит любой тип NaN. Всегда генерирует исключение Invalid Operation, если хотя бы один операнд является сигнальным NaN.

Флаги состояния

При записи этой инструкцией результата во флаги FPSCR, значения в нормальном режиме пересылаются во флаги ARM с помощью последующей инструкции VMRS.

Примеры

VCMP.F32 S4, #0.0
VCMP.F32 S4, S2

9.10.5 **Команды VCVT, VCVTR между числом с плавающей точкой и целым**

9.10.5.1 Преобразование между числами с плавающей точкой и целым.

Синтаксис

VCVT*{R}{cond}*.*Tm*.F32 *Sd, Sm*
VCVT*{cond}*.F32.*Tm Sd, Sm*

где:

R – если R указано, операция использует режим округления, определенный с помощью FPSCR. Если R отсутствует, операция использует режим округления Round towards Zero;

cond – дополнительный код условия.

Tm –тип данных для операнда. Должен быть одним из:

– S32 – 32-разрядное значение со знаком;
– U32 – 32-разрядное значение без знака;

Sd, Sm – регистр-приемник и регистр операнда.

Описание

Эти инструкции:

– либо преобразуют содержимое регистра из значения с плавающей точкой в целое 32-разрядное;
– либо преобразуют из 32-разрядного целого в значение с плавающей точкой;
– помещают результат во второй регистр.

Операция преобразования числа с плавающей точкой в целое в нормальном режиме использует режим округления Round towards Zero, но может дополнительно использовать режим округления, определенный с помощью FPSCR.

Операция преобразования целого в число с плавающей точкой использует режим округления, определенный с помощью FPSCR.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.6 **Команда VCVT между числом с плавающей точкой и числом с фиксированной точкой.**

9.10.6.1 Преобразование между числами с плавающей точкой и с фиксированной точкой.

Синтаксис

VCVT*{cond}*.*Td*.F32 *Sd*, *Sd*, *#fbits*
VCVT*{cond}*.F32.*Td Sd*, *Sd*, *#fbits*

где:

cond – дополнительный код условия;

Td – тип данных для числа с фиксированной точкой. Он может быть одним из:

– S16 – 16-разрядное значение со знаком;
– U16 – 16разрядное значение без знака;
– S32 – 32-разрядное значение со знаком;
– U32 – 32-разрядное значение без знака;

Sd – регистр-приемник и регистром операнда;

fbits – количество разрядов дроби в числе с фиксированной точкой:

– если *Td* равно S16 или U16, *fbits* должно быть в диапазоне 0-16;

– если *Td* равно S32 или U32, *fbits* должно быть в диапазоне 1-32.

Описание

Эти инструкции:

– либо преобразуют значение в регистре из числа с плавающей точкой в число с фиксированной точкой;

– либо преобразуют значение в регистре из числа с фиксированной точкой в число с плавающей точкой;

– помещают результат во второй регистр.

Значения числа с плавающей точкой имеют одинарную точность.

Числа с фиксированной точкой могут быть 16-разрядные или 32-разрядные. Преобразование из значения числа с фиксированной точкой берет операнд из нижней последовательности разрядов исходного регистра и отбрасывает все оставшиеся биты.

Знаковое преобразование в значение с фиксированной точкой расширяет результат знаком до ширины регистра назначения.

Беззнаковое преобразование в значение с фиксированной точкой расширяет результат нулем до ширины регистра назначения.

Операция преобразования числа с плавающей точкой в число с фиксированной точкой использует режим округления Round towards Zero. Операция преобразования числа с фиксированной точкой в число с плавающей точкой использует режим округления Round to Nearest.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.7 **Команды VCVTB, VCVTT**

9.10.7.1 Преобразование между значением с половинной точностью и значением с одиночной точностью.

Синтаксис

VCVT*{y}{cond}*.F32.F16 *Sd, Sm*

VCVT*{y}{cond}*.F16.F32 *Sd, Sm*

где:

y - определяет какая половина регистра операнда *Sm* или регистра назначения *Sd* используется для операнда или назначения:

– если *y* = B, то используется нижняя половина, биты [15:0] *Sm* или *Sd*;

– если *y* = T, то используется верхняя половина, биты [31:16] *Sm* или *Sd*;

cond – дополнительный код условия;

Sd – регистр-приемник;

Sm – регистр операнда.

Описание

Инструкция с суффиксом .F16.F32:

1. преобразует значение с половинной точностью в верхней или нижней половине регистра с одиночной точностью в значение одиночной точности;
2. записывает результат в регистр одиночной точности.

Эта инструкция с суффиксом .F32.F16:

1. преобразует значение в регистре одиночной точности в значение половинной точности;
2. записывает результат в верхнюю или нижнюю половину регистра с одиночной точностью, сохраняя неизменной другую половину целевого регистра.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.8 **Команда VDIV**

9.10.8.1 Деление значений с плавающей точкой.

Синтаксис:

VDIV*{cond}*.F32 *{Sd,} Sn, Sm*

где:

cond – дополнительный код условия;

Sd – регистр-приемник;

Sn, *Sm* – регистры операндов.

Описание:

Эта инструкция:

1) делит одно значение с плавающей точкой на другое значение с плавающей точкой;
2) записывает результат в регистр назначения с плавающей точкой.

Ограничения:

Ограничений нет.

Флаги состояния:

Эта инструкция не меняет состояние флагов.

9.10.9 **Команды VFMA, VFMS**

9.10.9.1 Совмещенное умножение со сложением чисел с плавающей точкой или с вычитанием.

Синтаксис

VFMA*{cond}*.F32 *{Sd,} Sn, Sm*

VFMS*{cond}*.F32 *{Sd,} Sn, Sm*

где

cond – дополнительный код условия;

Sd – регистр-приемник;

Sn, Sm – регистры операнды.

Операция

Инструкция VFMA:

1. умножает значения с плавающей точкой из регистров операндов;
2. накапливает результат в регистре назначения.

Результат умножения не округляется перед накоплением.

Инструкция VFMS:

1. инвертирует регистр первого операнда;
2. умножает значения с плавающей точкой из регистров первого и второго операндов;
3. прибавляет произведение к значению в регистре-приемнике;
4. помещает результат в регистр-приемник.

Результат умножения не округляется перед сложением.

Ограничения:

Ограничений нет.

Флаги состояния:

Эта инструкция не меняет состояние флагов.

9.10.10 **Команды VFNMA, VFNMS**

9.10.10.1 Совмещенное умножение чисел с плавающей точкой со сложением и инверсией, совмещенное умножение чисел с плавающей точкой с вычитанием и инверсией.

Синтаксис

VFNMA*{cond}*.F32 *{Sd,} Sn, Sm*

VFNMS*{cond}*.F32 *{Sd,} Sn, Sm*

где:

cond – дополнительный код условия;

Sd – регистр-приемник;

Sn, *Sm* – регистры операндов.

Описание

Инструкция VFNMA:

1. изменяет знак первого операнда с плавающей точкой в регистре;
2. умножает первый операнд с плавающей точкой на второй операнд с плавающей точкой;
3. прибавляет регистр назначения с плавающей точкой с обратным знаком к произведению;
4. помещает результат в регистр назначения.

Результат произведения не округляется перед сложением.

Инструкция VFNMS:

1. умножает первый операнд с плавающей точкой на второй операнд с плавающей точкой;
2. прибавляет значение с плавающей точкой с обратным знаком из регистра назначения к произведению;
3. помещает результат в регистр назначения.

Результат умножения не округляется перед сложением.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.11 **Команда VLDM**

9.10.11.1 Загрузка нескольких регистров расширения с плавающей точкой.

Синтаксис

VLDM*{mode}{cond}{.size} Rn{!}, list*

где:

mode – режим адресации:

– IA – «Инкремент После». Последовательные адреса начинаются с адреса указанного в *Rn*;

– DB – «Декремент Перед». Последовательные адреса заканчиваются непосредственно перед адресом указанным в *Rn*;

cond – дополнительный код условия;

size – дополнительный определитель размера данных;

Rn – базовый регистр. Может использоваться SP;

! – команда для записи измененного значения обратно в *Rn*. Обязательна, если *mode* == DB, и необязательна, если *mode* == IA;

list – список регистров области памяти, которые необходимо загрузить, в виде последовательно пронумерованных регистров, размером в два или одно слово, разделенных запятыми и окруженных скобками.

Описание

Эта инструкция загружает:

– множество регистров памяти из последовательных ячеек памяти, используя адрес из регистра ядра ARM в качестве базового адреса.

Ограничения

– если указан *size*, он должен быть равен размеру регистров в *list* в битах, 32 или 64;

– для базового адреса может использоваться SP. В наборе инструкций ARM если не указан !, то может использоваться PC;

– *list* должен содержать хотя бы один регистр. Если он содержит регистры длинной два слова, он не должен содержать более 16 регистров;

– при использовании режима адресации Decrement Before, флаг обратной записи (!) должен быть прикреплен к определению базового регистра.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.12 **Команда VLDR**

9.10.12.1 Загрузка регистра расширения из памяти.

Синтаксис:

VLDR*{cond}{.64} Dd, [Rn{#imm}]*
VLDR*{cond}{.64} Dd, label*
VLDR*{cond}{.64} Dd, [*PC*, #imm}]*
VLDR*{cond}{.32} Sd, [Rn {, #imm}]*
VLDR*{cond}{.32} Sd, label*
VLDR*{cond}{.32} Sd, [*PC*, #imm]*

где:

cond – дополнительный код условия;

64, 32 – дополнительные определители размера данных;

Dd – регистр назначения для загрузки двойного слова;

Sd – регистр назначения для загрузки одного слова;

Rn – базовый регистр. Может использоваться SP;

Imm – положительное или отрицательное непосредственное смещение, используемое для вычисления адреса. Разрешенные значения адреса кратны 4 и находятся в диапазоне от 0 до 1020;

Label – метка элемента символьных данных для загрузки.

Операция

Эта инструкция:

– загружает один регистр расширения из памяти, используя базовый адрес из регистра ядра ARM, с дополнительным смещением.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.13 **Команды VLMA, VLMS**

9.10.13.1 Умножение двух значений с плавающей точкой с накоплением или вычитанием результата.

Синтаксис

VLMA*{cond}*.F32 *Sd, Sn, Sm*
VLMS*{cond}*.F32 *Sd, Sn, Sm*

где

cond – дополнительный код условия;

Sd – результат с плавающей точкой;

Sn, Sm – операнды с плавающей точкой.

Описание

Инструкция VLMA:

1. перемножает два значения с плавающей точкой;
2. прибавляет результат к результату с плавающей точкой.

Инструкция VLMS:

1. перемножает два значения с плавающей точкой;

2. вычитает произведение из результата с плавающей точкой;
3. помещает результат в регистр назначения.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.14 **Команда VMOV «Константа»**

9.10.14.1 Перемещение непосредственно числа с плавающей точкой в регистр.

Синтаксис

VMOV*{cond}*.F32 *Sd, #imm*

где:

cond – дополнительный код условия;

Sd – результат с плавающей точкой;

Imm – константа с плавающей точкой.

Описание

Эта инструкция копирует константное значение в регистр с плавающей точкой.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.15 **Команда VMOV «Регистр»**

9.10.15.1 Перемещение регистра с плавающей точкой в регистр.

Синтаксис

VMOV*{cond}*.F64 *Dd, Dm*

VMOV*{cond}*.F32 *Sd, Sm*

где:

cond – дополнительный код условия;

Dd - регистр назначения, для операции с двухсловными операндами;

Dm - регистр источник, для операции с двухсловными операндами;

Sd - регистр назначения, для операции с однословными операндами;

Sm - регистр источник, для операции с однословными операндами.

Операция

Эта инструкция копирует содержимое одного регистра с плавающей точкой в другой.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.16 **Команда VMOV «Скаляр в регистр Ядра ARM»**

9.10.16.1 Пересылает одно слово двусловного регистра с плавающей точкой в регистр ядра ARM.

Синтаксис

VMOV*{cond} Rt, Dn[x]*

где:

cond – дополнительный код условия;

Rt – регистр назначения ядра ARM;

Dn – 64-разрядный двухсловный регистр;

x – определяет половину двухсловного регистра, которую необходимо использовать:

– если x равен 0, использовать нижнюю половину двухсловного регистра;

– если x равен 1, использовать верхнюю половину двухсловного регистра.

Описание

Эта инструкция пересылает:

– одно слово из верхней или нижней половины двухсловного регистра с плавающей точкой в регистр ядра ARM.

Ограничения

Нельзя использовать PC (счетчик команд) или SP (указатель стека) в качестве *Rt*.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.17 **Команда VMOV «регистр ядра ARM с одинарной точностью»**

9.10.17.1 Пересылает регистр с одинарной точностью в регистр и из регистра ядра ARM.

Синтаксис

VMOV*{cond} Sn, Rt*

VMOV*{cond} Rt, Sn*

где:

cond –дополнительный код условия;

Sn - регистр одинарной точности с плавающей точкой;

Rt - регистр ядра ARM.

Операция

Эта инструкция пересылает:

– содержимое регистра с одинарной точностью в регистр ядра ARM;

– содержимое регистра ядра ARM в регистр одинарной точности.

Ограничения

Нельзя использовать PC (счетчик команд) или SP (указатель стека) в качестве *Rt*.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.18 **Команда VMOV «два регистра ядра ARM в два регистра с одинарной точностью»**

9.10.18.1 Пересылает два последовательно перечисленных регистра одиночной точности в- и из- двух регистров ядра ARM.

Синтаксис:

VMOV*{cond} Sm, Sm1, Rt, Rt2*

VMOV*{cond} Rt, Rt2, Sm, Sm*

где:

cond – дополнительный код условия;

Sm – первый регистр одиночной точности;

Sm1 – второй регистр с одиночной точностью. Это следующий регистр одиночной точности после *Sm*;

Rt – регистр ядра ARM, в который или из которого пересылается *Sm*;

Rt2 – регистр ядра ARM, в который или из которого пересылается *Sm1*.

Операция

Эта инструкция пересылает:

– содержимое двух последовательно перечисленных регистров одиночной точности в два регистра ядра ARM;

– содержимое двух регистров ядра ARM в пару регистров одиночной точности.

Ограничения

Ограничениями являются:

– регистры с плавающей точкой должны быть последовательными, один после другого;

– регистры ядра ARM не обязаны быть последовательными;

– нельзя использовать PC (счетчик команд) или SP (указатель стека) в качестве *Rt*.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.19 **Команда VMOV «регистр ядра ARM в скаляр»**

9.10.19.1 Пересылает одно слово в регистр с плавающей точкой из регистра ядра ARM.

Синтаксис

VMOV*{cond}{*.32*}* *Dd[x], Rt*

где:

cond – дополнительный код условия;

32 – дополнительный определитель размера данных;

Dd[x] – результат, где [*x*] определяет половину двойного слова для пересылки следующим образом:

– если *x* равен 0, извлекается нижняя половина;

– если *x* равен 1, извлекается верхняя половина;

Rt – регистр-источник ядра ARM.

Операция

Эта инструкция пересылает одно слово в верхнюю или нижнюю половину двухсловного регистра с плавающей точкой из регистра ядра ARM.

Ограничения

Нельзя использовать PC (счетчик команд) или SP (указатель стека) в качестве *Rt*.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.20 **Команда VMRS**

9.10.20.1 Перемещает в регистр ядра ARM из системного регистра с плавающей точкой.

Синтаксис

VMRS{cond} *Rt*, FPSCR

VMRS*{cond}* APSR_*nzcv*, FPSCR

где:

cond – дополнительный код условия;

Rt – регистр назначения ядра ARM. Этим регистром могут быть R0-R14;

APSR_*nzcv* – пересылает флаги с плавающей точкой во флаги APSR.

Описание

Эта инструкция выполняет одно из следующих действий:

– копирует значение FPSCR в регистр общего назначения;

– копирует значения бит флага FPSCR во флаги APSR N, Z, C и V.

Ограничения

Нельзя использовать PC (счетчик команд) или SP (указатель стека) в качестве *Rt*.

Флаги состояния

Эти инструкции могут менять флаги: N, Z, C, V.

9.10.21 **Команда VMSR**

9.10.21.1 Перемещает в системный регистр с плавающей точкой из регистра ядра ARM.

Синтаксис

VMSR*{cond}* FPSCR, *Rt*

где:

cond – дополнительный код условия;

Rt – регистр общего назначения для пересылки в FPSCR;

Описание
Эта инструкция перемещает значение регистра общего назначения в FPSCR.
Ограничения
Нельзя использовать PC (счетчик команд) или SP (указатель стека) в качестве *Rt*.
Флаги состояния
Эта инструкция обновляет FPSCR.

9.10.22 **Команда VMUL**

9.10.22.1 Умножение чисел с плавающей точкой.
Синтаксис
VMUL*{cond}*.F32 *{Sd,} Sn, Sm*
где:
cond – дополнительный код условия;
Sd – результат с плавающей точкой;
Sn, *Sm* – операнды с плавающей точкой.
Описание
Эта инструкция:
1. перемножает два значения с плавающей точкой;
2. помещает результат в регистр назначения.

Ограничения
Ограничений нет.
Флаги состояния:
Эта инструкция не меняет состояние флагов.

9.10.23 **Команда VNEG**

9.10.23.1 Инверсия числа с плавающей точкой.
Синтаксис
VNEG*{cond}*.F32 *Sd, Sm*
где:
cond – дополнительный код условия;
Sd – результат с плавающей точкой;
Sm – операнд с плавающей точкой.
Описание
Эта инструкция:
1. меняет знак значения с плавающей точкой;
2. помещает результат во второй регистр с плавающей точкой.

Инструкция с плавающей точкой инвертирует бит знака.
Ограничения
Ограничений нет.
Флаги состояния
Эта инструкция не меняет состояние флагов.

9.10.24 **Команды VNMLA, VNMLS, VNMUL**

9.10.24.1 Умножение чисел с плавающей точкой с последующим сложением или вычитанием и сменой знака.
Синтаксис
VNMLA*{cond}*.F32 *Sd, Sn, Sm*
VNMLS*{cond}*.F32 *Sd, Sn, Sm*
VNMUL*{cond}*.F32 *{Sd,} Sn, Sm*
где:
cond – дополнительный код условия;
Sd – регистр-приемник с плавающей точкой;
Sn, Sm – регистры операндов с плавающей точкой.

Описание

Инструкция VNMLA:

1. перемножает значения двух регистров с плавающей точкой;
2. прибавляет значение регистра-приемника с плавающей точкой, взятое с противоположным знаком, к произведению, взятому с противоположным знаком;
3. записывает результат обратно в регистр назначения.

Инструкция VNMLS:

1. перемножает значения двух регистров с плавающей точкой;
2. прибавляет значение регистра-приемника с плавающей точкой, взятое с противоположным знаком, к произведению;
3. записывает результат обратно в регистр-приемник.

Инструкция VNMUL:

1. перемножает вместе значения двух регистров с плавающей точкой;
2. записывает результат, взятый с противоположным знаком, в регистр-приемник.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.25 **Команда VPOP**

9.10.25.1 Извлечение из стека регистров расширения.

Синтаксис

VPOP*{cond}{.size} list*

где:

cond – дополнительный код условия;

size – дополнительный определитель размера данных.

Если указан, то он должен быть равен размеру регистров в *list* в битах, 32 или 64;

list - является списком регистров расширения, которые необходимо загрузить, в виде списка последовательно перечисленных двухсловных или однословных регистров, разделенных запятыми и окруженных скобками.

Описание

Эта инструкция загружает множество последовательных регистров расширения из стека.

Ограничения

Список должен содержать хотя бы один регистр, но не более 16 регистров.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.26 **Команда VPUSH**

9.10.26.1 Помещение в стек регистров расширения.

Синтаксис

VPUSH*{cond}{.size} list*

где:

cond – дополнительный код условия;

size – дополнительный определитель размера данных.

Если указан, то он должен быть равен размеру в битах регистров в *list*, 32 или 64;

list – список регистров расширения, которые необходимо сохранить, в виде списка последовательно перечисленных двухсловных или однословных регистров, разделенных запятыми и окруженных скобками.

Описание

Эта инструкция:

– сохраняет множество последовательных регистров расширения в стек.

Ограничения

– список должен содержать хотя бы один регистр, но не более 16 регистров.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.27 **Команда VSQRT**

9.10.27.1 Вычисление квадратного корня из числа с плавающей точкой.

Синтаксис

VSQRT*{cond}*.F32 *Sd, Sm*

где:

cond - дополнительный код условия;

Sd – результат с плавающей точкой;

Sm – результат с плавающей точкой.

Описание

Эта инструкция:

– вычисляет квадратный корень числа с плавающей точкой в регистре;

– записывает результат в другой регистр с плавающей точкой.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.28 **Команда VSTM**

9.10.28.1 Сохранение нескольких регистров расширения.

Синтаксис

VSTM*{mode}{cond}{.size} Rn{!}, list*

где:

mode – режим адресации:

– IA «Инкремент После». Последовательные адреса начинаются с адреса, указанного в *Rn*. Это по умолчанию и может быть опущено;

– DB «Декремент До». Последовательные адреса заканчиваются непосредственно перед адресом, определенном в *Rn*;

cond – дополнительный код условия;

size – дополнительный определитель размера данных.

Если указан, то он должен быть равен размеру регистров в *list* в битах, 32 или 64;

Rn – является базовым регистром;

! - является функцией, которая предписывает инструкции записывать измененное значение обратно в *Rn*.

Необходимо если *mode* == DB;

list – список регистров расширения, которые необходимо сохранить, в виде списка последовательно перечисленных двухсловных и однословных регистров, разделенных запятыми и окруженных скобками.

Описание

Эта инструкция:

– сохраняет множество регистров расширения в последовательные ячейки памяти, используя базовый адрес из регистра ядра ARM.

Ограничения

Ограничениями являются:

– *list* должен содержать как минимум один регистр. Если он содержит двухсловные регистры, то он не должен содержать более 16 регистров.

– Не рекомендуется использовать регистр PC (программный счетчик) в качестве *Rn*.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.29 **Команда VSTR**

9.10.29.1 Сохранение регистра расширения в память.

Синтаксис

VSTR*{cond}{.32} Sd, [Rn{, #imm}]*

VSTR*{cond}{.64} Dd, [Rn{, #imm}]*

где:

cond – дополнительный код условия;

32, 64 - дополнительные определители размера данных;

Sd - исходный регистр для сохранения однословного значения;

Dd - исходный регистр для сохранения двусловного значения;

Rn - базовый регистр. Может использоваться SP;

Imm – положительное или отрицательное непосредственное смещение, используемое для вычисления адреса. Значения кратны 4 в диапазоне 0-1020. *imm* может быть пропущено, означая смещение +0.

Описание

Эта инструкция:

−сохраняет один регистр расширения в памяти, используя адрес из регистра ядра ARM, с дополнительным смещением, определенным в *imm*.

Ограничения

Не рекомендуется использовать регистр PC (программный счетчик) в качестве *Rn*.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.10.30 **Команда VSUB**

9.10.30.1 Вычитание чисел с плавающей точкой.

Синтаксис

VSUB*{cond}*.F32 *{Sd,} Sn, Sm*

где:

cond – дополнительный код условия;

Sd – результат с плавающей точкой;

Sn, Sm – операнды с плавающей точкой;

Описание

Эта инструкция:

−вычитает одно значение с плавающей точкой из другого значения с плавающей точкой;

−помещает результат в регистр назначения с плавающей точкой.

Ограничения

Ограничений нет.

Флаги состояния

Эта инструкция не меняет состояние флагов.

9.11 Прочие команды

9.11.1 В таблице 9.16 представлены прочие команды Cortex-M4.

Таблица 9.16 – Прочие команды

Мнемоника	Краткое описание
BKPT	Точка останова
CPSID	Изменить состояние процессора, отключить прерывания
CPSIE	Изменить состояние процессора, включить прерывания
DMB	Барьер синхронизации доступа к памяти данных
DSB	Барьер синхронизации доступа к памяти данных
ISB	Барьер синхронизации доступа к инструкциям
MRS	Загрузка из специального регистра в регистр общего назначения
MSR	Записать регистр общего назначения в специальный регистр
NOP	Нет операции
SEV	Установить признак события
SVC	Вызов супервизора
WFE	Ожидать событие
WFI	Ожидать прерывание

9.11.2 **Команда BKPT**

9.11.2.1 Точка останова.

Синтаксис

BKPT *#imm*

где:

imm – целое число в диапазоне 0-255 (8-разрядное значение).

Описание

Инструкция BKPT заставляет процессор перейти в режим отладки. Инструменты отладки используют это для исследования состояния системы, когда достигнута инструкция по определенному адресу.

imm игнорируется процессором. При необходимости, отладчик может использовать его для сохранения дополнительной информации о контрольной точке.

Инструкция BKPT может быть помещена в блок IT, но при этом она выполняется безусловно.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
BKPT #0x3       ; Контрольная точка с непосредственным значением
                ; равным 0x3 (отладчик может извлечь непосредственное значение,
                ; путем определения его позиции, используя PC)
```

9.11.3 **Команда CPS**

9.11.3.1 Изменить состояние процессора.

Синтаксис

CPS*effect iflags*

где:

effect – один из возможных суффиксов:

– IE – сбрасывает специальный регистр в 0;

– ID – устанавливает специальный регистр в 1;

iflags - последовательность флагов:

– i – устанавливает или очищает регистр PRIMASK;

– f - устанавливает или очищает регистр FAULTMASK.

Описание

Команда CPS позвоялет изменить значения специальных регистров PRIMASK и FAULTMASK.

Ограничения

Команда CPS доступна только из привиленгированного приложения, при вызове из непривилегированного приложения она игнорируется.

Команда CPS не допускает условного исполнения, поэтому не должна использоваться внутри блока IT.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
CPSID i     ; Запрещает прерывания и конфигурируемые
            ; обработчики отказов
CPSID f     ; Запрещает прерывания и все обработчики
            ; отказов
CPSIE i     ; Разрешает прерывания и конфигурируемые
            ; обработчики отказов
CPSIE f     ; Разрешает прерывания и обработчики отказов
```

9.11.4 **Команда DMB**

9.11.4.1Барьер синхронизации доступа к памяти данных.

Синтаксис

DMB*{cond}*

где:

cond – дополнительный код условия.

Описание

Команда DMB выполняет функцию барьерной синхронизации доступа к памяти данных. Она гарантирует, что все явные обращения к памяти, которые были инициированы перед инструкцией DMB, будут завершены перед любым явным обращением к памяти, которые появляются в программе, после инструкции DMB. DMB не меняет порядок исполнения инструкций, которые не обращаются к памяти.

Флаги состояния

Данная инструкция не меняет состояние флагов.

Примеры

```
DMB         ; Барьер синхронизации доступа к памяти данных
```

9.11.5 **Команда DSB**

9.11.5.1 Барьер синхронизации доступа к памяти данных.

Синтаксис

DSB*{cond}*

где:

cond – дополнительный код условия.

Операция

Команда DSB выполняет функцию барьерной синхронизации доступа к памяти данных. Инструкции, которые следуют после DSB, не начнут выполнться до завершения инструкции DSB. Инструкция DSB завершается после того, как завершатся все явные обращения к памяти до неё.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
DSB         ; Барьер синхронизации доступа к памяти данных
```

9.11.6 **Команда ISB**

9.11.6.1Барьер синхронизации доступа к инструкциям.

Синтаксис

ISB*{cond}*

где:

cond – дополнительный код условия.

Описание

Команда ISB выполняет функцию барьерной синхронизации выполнения команд. Она очищает конвейер инструкций процессора. Таким образом, все инструкции, следующие после ISB, по завершению инструкции ISB выбираются заново.

Флаги состояния

Данная инструкция не меняет состояние флагов.

Примеры

```
ISB                 ; Барьер синхронизации доступа к инструкциям
```

9.11.7 **Команда MRS**

9.11.7.1Считать содержимое специального регистра в регистр общего назначения.

Синтаксис

MRS*{cond} Rd, spec_reg*

где:

cond – дополнительный код условия;

Rd – регистром-приемник;

spec_reg – один из регистров: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK или CONTROL.

Описание

Инструкции MRS в комбинации с MSR используются для последовательности чтения-модификации-записи элементов PSR, например, для очистки флага Q.

В коде, отвечающем за переключение процессов, необходимо обеспечить сохранение состояния приостановленного процесса, и восстановление состояния активизированного процесса. Необходимой составной частью сохраняемой (восстанавливаемой) информации является значение регистра PSR. При этом на этапе сохранения состояния используется команда MRS, а на этапе восстановления - команда MSR.

Примечание - при использовании инструкцией MRS регистр BASEPRI_MAX является синонимом регистра BASEPRI.

Ограничения

В качестве регистра *Rd* нельзя использовать регистр SP (указатель стека) и регистр PC (счетчик команд)

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
MRS R0, PRIMASK             ; Считывает значение PRIMASK и записывает его в R0
```

9.11.8 **Команда MSR**

9.11.8.1 Записать регистр общего назначения в специальный регистр.

Синтаксис

MSR*{cond} spec_reg, Rn*

где:

cond – дополнительный код условия;

Rn – регистр-источник данных;

*spec_re*g – один из регистров: APSR_nzcvq, APSR_g, APSR_nzcvqg, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, или CONTROL.

Описание

Доступ к специальным регистрам в команде MSR зависит от уровня привилегий. Непривилегированное приложение может обращаться только к APSR. При этом попытки записи в нераспределенные биты, а также в EPSR игнорируются. Привилегированное приложение может обращаться ко всем специальным регистрам.

При записи данных в регистр BASEPRI_MAX инструкция записывает данные в регистр BASEPRI только если выполняется одно из двух условий:

– *Rn* отлично от нуля и текущее значение BASEPRI равно 0;

– *Rn* отлично от нуля и меньше чем текущее значение BASEPRI.

Ограничения

В качестве регистра *Rn* нельзя использовать регистр SP (указатель стека) и регистр PC (счетчик команд).

Флаги состояния

Данная инструкция не влиияет на состояние флагов.

Примеры

```
MSR CONTROL, R1          ; Считывает значение R1, записывает
                         ; его в регистр CONTROL
```

9.11.9 **Команда NOP**

9.11.9.1Нет операции.

Синтаксис

NOP*{cond}*

где:

cond – дополнительный код условия.

Описание

NOP не делает ничего. Процессор может удалить эту инструкцию из конвейера до достижения ступени выполнения.

Команду NOP рекомендуется использовать для заполнения, например с целью разместить очередную инструкцию по адресу, выровненному по 64-разрядной границе.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
NOP                      ; Нет операции
```

9.11.10 **Команда SEV**

9.11.10.1 Послать событие

Синтаксис

SEV*{cond}*

где:

cond – дополнительный код условия.

Описание

Инструкция SEV используется для передачи информации о событии всем процессорам в составе многопроцессорной системы. Кроме того, он устанавливает собственный регистр события в 1.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
SEV                      ; Посылает Событие
```

9.11.11 **Команда SVC**

9.11.11.1Вызов супервизора.

Синтаксис

SVC*{cond} #imm*

где:

cond – дополнительный код условия;

imm – константное выражение, целое число в диапазоне 0-255 (8-разрядное значение).

Описание

Инструкция SVC вызывает формирование исключения SVC. Параметр *imm* игнорируется процессором. При необходимости он может быть получен обработчиком исключения для определения запрошенного приложением варианта обслуживания.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
SVC #0x32       ; Вызов Супервизора (обработчик SVCall
                ; может извлечь непосредственное значение
                ; путем определения его позиции через помещенный в стек PC).
```

9.11.12 **Команда WFE**

9.11.12.1 Ожидать событие.

Синтаксис

WFE*{cond}*

где:

cond – дополнительный код условия.

Описание

В случае если регистр события равен 0, выполнение команды WFE приводит к приостановке исполнения команд до тех пор, пока не произойдет одно из следующих событий:

– исключение, если не маскированное регистром маскировки исключения или текущим уровнем приоритета;

– перевод исключения в состояние ожидания обслуживания при установленном в 1 бите SEVONPEND в регистре управления системой SCR;

– получение запроса на переход в режим отладки, если отладка разрешена;

– получение сигнала о событии от периферийного устройства или от другого процессора (по команде SEV) в многопроцессорной системе.

Если регистр события равен 1, WFE очищает его в 0 после чего завершает свое функционирование без приостановки процессора.

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
WFE             ; Ожидание события
```

9.11.13 **Команда WFI**

9.11.13.1 Ожидание прерывания.

Синтаксис

WFI*{cond}*

где:

cond – дополнительный код условия.

Описание

Команда WFI приостанавливает процессор до тех пор, пока не наступит одно из следующих событий:

– исключение;

– запрос на перевод в режим отладки, вне зависимости от того, разрешен ли этот режим;

Флаги состояния

Эта инструкция не меняет состояние флагов.

Примеры

```
WFI             ; Ожидание прерывания
```

10 Прерывания и исключения

10.1 Состояния исключений

10.1.1 Каждое исключение может быть в одном из следующих состояний:

– **Inactive (неактивный) -** исключение не находится в стадии Active или Pending;

– **Pending (ожидание)** - исключение находится в режиме ожидания обработки процессором. Запрос прерывания с периферийного устройства или из программного обеспечения может изменить состояние соответствующего прерывания на состояние ожидания;

– **Active (активный) –** исключение обслуживается процессором, но не является завершенным. Обработчик исключений может прервать выполнение другого обработчика исключений. В этом случае оба исключения находятся в активном состоянии.

– **Active and pending (активный и в ожидании) -** исключение обслуживается процессором, но появилось новое исключение в состоянии Pending от того же источника.

10.2 Типы исключений

10.2.1 **Reset**

10.2.1.1 Reset – сброс. Вызывается подачей сигнала сброса. Модель исключений интерпретирует reset как специальную форму исключения. При появлении исключения reset, работа процессора останавливается. Такое возможно в любой момент выполнения команды. После выхода процессора из состояния сброса, исполнение команд перезапускается с адреса, заданного в таблице векторов для сброса. Исполнение перезапускается как привилегированное исполнение в режиме потока.

10.2.2 **NMI**

10.2.2.1 NMI **–** немаскируемое прерывание. Оно может быть вызвано периферийным устройством или установлено программой. Оно имеет самый высокий приоритет исключения после reset. Оно постоянно разрешено и имеет фиксированный приоритет равный -2.

NMI не может быть:

– замаскированным или предотвращено от активации из другого исключения;

– перехваченным любым другим исключением кроме Reset.

10.2.3 **Hard Fault**

10.2.3.1 Hard Fault (тяжелый отказ) – исключение, которое происходит из-за ошибки в процессе обработки исключения или по причине невозможности обработки исключения каким-либо другим механизмом. Hard Fault имеет фиксированный приоритет равный -1, означающий, что оно имеет более высокий приоритет, чем любое исключение с конфигурируемым приоритетом.

10.2.4 **Mem Manage**

10.2.4.1 MemManage **(отказ системы управления памятью) –** исключение, которое возникает из-за **нарушения правил доступа к памяти**. Блок MPU или фиксированные защитные настройки определяют это исключение как для данных, так и для инструкций. Исключение используется для того, чтобы прервать доступ за инструкцией в область памяти с атрибутом *Execute Never* (XN).

10.2.5 **Bus Fault**

10.2.5.1 BusFault (отказ шины) – исключение, которое возникает при ошибке памяти во время обмена по шине при выборке команды или данных. Например, при обращении в несуществующую память.

10.2.6 **Usage Fault**

10.2.6.1 UsageFault (отказ программы) – исключение, которое возникает из-за отказа при выполнении команды. Например:

– неопределенная команда;

– запрещенное невыровненное обращение;

– недопустимое состояние при выполнении команды;

– ошибка при возврате из исключения.

Следующие ситуации могут вызвать UsageFault, если ядро сконфигурировано для сообщения о них:

– невыровненный адрес доступа к слову и полуслову в памяти;

– деление на ноль.

10.2.7 **SVCall**

10.2.7.1 SVCall (вызов супервизора командой SVC) – исключение, которое возникает при выполнении инструкции SVC. В приложениях с использованием операционных сред инструкция SVC может использоваться для того, чтобы получить доступ к функциям ядра ОС и драйверам устройств.

10.2.8 **PendSV**

10.2.8.1 Исключение PendSV является прерыванием запросов сервисов системного уровня. В приложениях с использованием ОС PendSV используется для переключения контекстов, когда нет других активных исключений.

10.2.9 **SysTick**

10.2.9.1 Исключение SysTick генерируется системным таймером, когда он обнуляется. Программное беспечение также может генерировать исключение SysTick. В приложениях с использованием ОС процессор может использовать это исключение для подсчета системных циклов.

10.2.10 **Interrupt (IRQ)**

10.2.10.1 Прерывания или IRQ - это исключения, вызываемые периферийными устройствами или программными запросами. Все прерывания асинхронны по отношению к выполняемым инструкциям. В системе прерывания используются для коммуникации периферии и процессора.

Таблица 10.1 – Свойства различных типов исключений

Номер исключения	**Номер IRQ**	**Тип**	**Приоритет**	**Адрес вектора обработчика (смещение)**	**Активация**
1	-	Reset	-3, наивысший	0x0000_0004	Асинхронный
2	-14	NMI	-2	0x0000_0008	Асинхронный
3	-13	Hard Fault	-1	0x0000_000C	-
4	-12	Mem Manage	Конфигурируемый	0x0000_0010	Синхронный
5	-11	Bus Fault	Конфигурируемый	0x0000_0014	Синхронный/ Асинхронный
6	-10	Usage Fault	Конфигурируемый	0x0000_0018	Синхронный
7-10	-	-	-	Зарезервировано	-
11	-5	SVCall	Конфигурируемый	0x0000_002C	Синхронный
12-13	-	-	-	Зарезервировано	-
14	-2	PendSV	Конфигурируемый	0x0000_0038	Асинхронный
15	-1	SysTick	Конфигурируемый	0x0000_003C	Асинхронный
16 и выше	0	IRQ	Конфигурируемый	0x0000_0040	Асинхронный

Для асинхронных исключений, кроме RESET, процессор может выполнить другие инструкции между возникновением сигнала исключения и входом в обработчик.

Программа в Privileged режиме может запретить прерывания, имеющие конфигурируемый приоритет.

10.3 Обработчики исключений

10.3.1 Для обработки исключений используются:

– Процедуры обработки прерываний (Interrupt Service Routines - ISRs). Прерывания с IRQ0 по IRQ31 обрабатываются процедурами ISR .

– Обработчики ошибок (Fault Handlers). Обрабатывают исключения Hard fault, memory management fault, usage fault и bus fault.

– Системные обработчики (System handlers). Обрабатывают исключения NMI, PendSV, SVCall и SysTick.

10.4 Таблица векторов

10.4.1 Таблица векторов содержит указатель стека, вектор входа по RESET и стартовые адреса обработчиков, также называемых векторами. На рисунке 10.1 представлена последовательность векторов в таблице. Младший бит всех векторов должен быть равен 1, указывая на то, что обработчик выполняется в Thumb режиме.

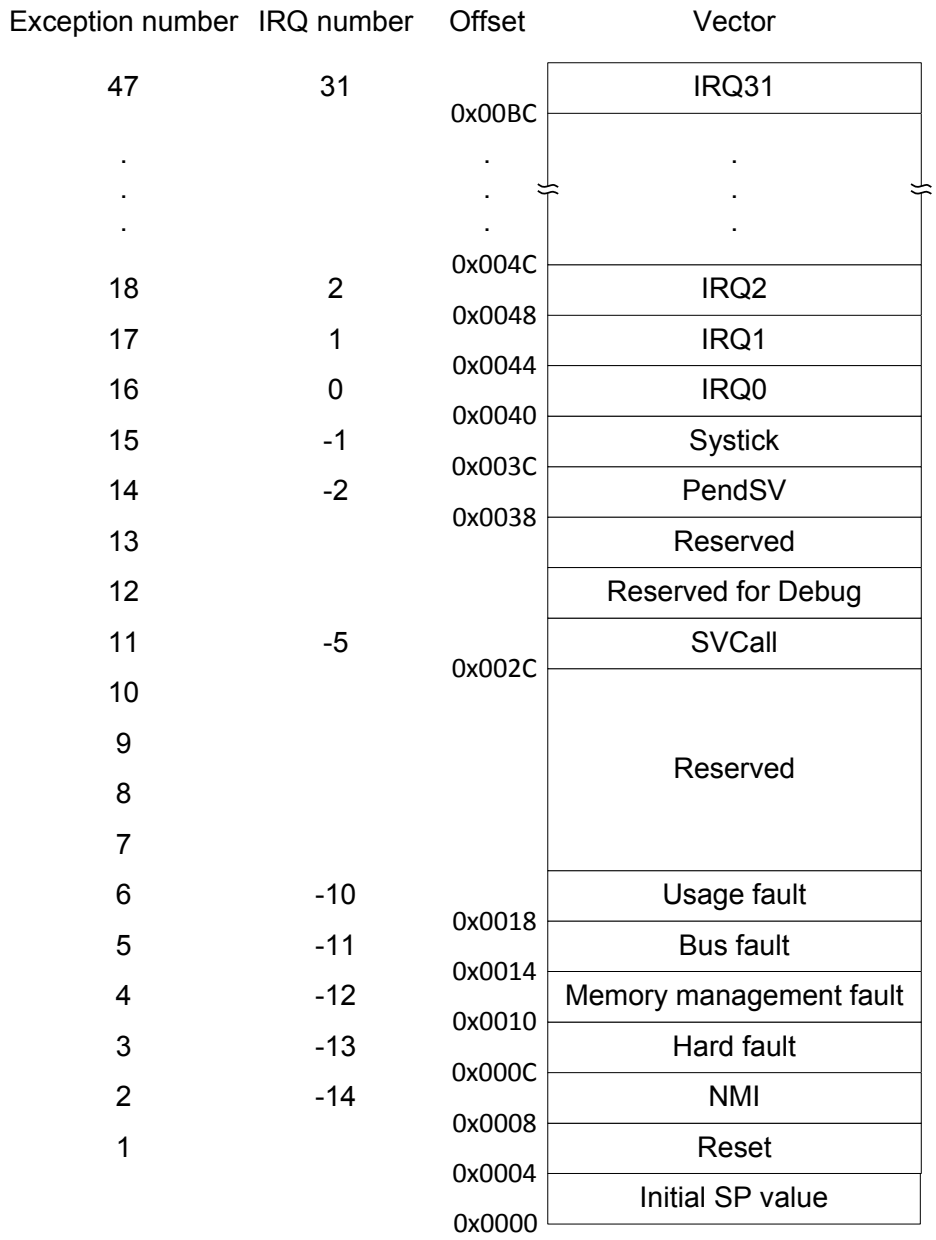

Рисунок 10.1 – Таблица векторов исключений и прерываний

При системном сбросе таблица векторов располагается по фиксированному адресу 0x00000000. Программное обеспечение в privileged режиме может перенести таблицу в другое место памяти через регистр VTOR. Таблица может располагаться в адресах от 0x00000080 до 0x3fffff80. Подробнее в описании регистра VTOR.

10.5 Приоритеты исключений

10.5.1 Как показано в таблице 10.1 все исключения имеют приоритет такой, что:

– меньшее значение уровня приоритета указывает на более высокий приоритет;

– программируемые уровни приоритетов имеют все исключения, кроме Reset, HardFault и NMI.

Если программное обеспечение не программирует приоритеты, тогда все исключения с программируемым приоритетом имеют приоритет 0.

Примечание - значения программируемых приоритетов находятся в дипазоне 0-255. Это означает, что Reset, HardFault и NMI, имеющие отрицательное значение приоритета, всегда имеют больший приоритет, чем любое другое исключение.

Назначение более высокого значения приоритета для IRQ[0] и более низкого значения приоритета для IRQ[1] означает, что IRQ[1] имеет более высокий приоритет, чем IRQ[0]. Если IRQ[1] и IRQ[0] находятся в состоянии ожидания, то IRQ[1] обрабатывается перед IRQ[0].

Если несколько исключений, находящихся в режиме ожидания, имеют одинаковый приоритет, то исключение с меньшим порядковыим номером будет иметь превосходство. Например, если IRQ[0] и IRQ[1] имеют одинаковый приоритет, IRQ[0] будет обработан перед IRQ[1].

Если процессор выполняет обработку исключений и появляется исключение с более высоким приоритетом, то происходит переход на обработчик с большим приоритетом. Если при выполнении обработчика произошло исключение с таким же приоритетом, то это исключение будет выполнено по завершению текущего обработчика, несмотря на порядковый номер исключения.

10.6 Группировка приоритетов прерываний

10.6.1 Для увеличения управляемости приоритетов в системах с прерываниями контроллер прерываний NVIC поддерживает группировку приоритетов. Это достигается за счет разбиения регистра приоритета прерывания на две части:

– верхняя часть определяет группу приоритетов;

– нижняя часть задает подприоритет в группе.

Только приоритет группы определяет последовательность обработки прерываний. Когда процессор выполняет обработку прерывания, другое прерывание с таким же приоритетом группы не прервет обработку первоначального обработчика. При возникновении нескольких прерываний, имеющих одинаковый приоритет группы, подприоритеты определяют последовательность их обработки. При возникновении нескольких прерываний с одинаковым приоритетом группы и подприоритетом первым обрабатывается прерывание с меньшим номером.

10.7 Вход в обработчик и выход из обработчика

10.7.1 **Приоритетное прерывание обслуживания**

10.7.1.1 Когда процессор выполняет программу обработки исключений, другое исключение может приостановить выполнение обработчика прерывания, если его приоритет выше, чем приоритет выполняемого обработчика.

В случае, если внутри обработчика исключения возникает прерывание более высокого приоритета, возникает ситуация, называемая вложенным исключением.

10.7.2 **Возврат из прерывания**

10.7.2.1 Возврат из программы-обработчика осуществляется по завершении обработки исключительной ситуации, с одновременноым выполнением следующих условий:

– нет исключения, находящегося в режиме ожидания с достаточным приоритетом для обработки;

– завершенный обработчик исключений не обрабатывал «опоздавшее» исключение.

Процессор обращается к стеку и восстанавливает состояние, имевшее место до вызова обработчика.

10.7.3 **Передача управления без восстановления контекста (tail-chaining)**

10.7.3.1 Данный механизм ускоряет обслуживание исключений. Если исключение возникает в тот момент, когда процессор занят обработкой исключительной ситуации с таким же или более высоким приоритетом, то оно переводится в состояние ожидания (откладывается). После завершения текущего обработчика процессор может приступить к обработке отложенного прерывания. В этом случае вместо того, чтобы восстановить состояние процессора из стека, а затем снова сохранить его в стек, процессор, пропуская эти операции, сразу переходит к обработчику отложенного прерывания.

10.7.4 **«Опоздавшее» исключение (Late-arriving)**

10.7.4.1 Этот механизм ускоряет приоритетное прерывание обслуживания. Если исключение с более высоким приоритетом возникает в течение сохранения состояния для предшествующего исключения, то процессор переключается на обработку исключения с более высоким приоритетом и инициализирует векторную выборку для этого исключения. При выходе из обработчика исключений «опоздавшего» исключения, применяются правила стандартного механизма передачи управления без восстановления контекста.

10.7.5 **Вход в процедуру обработки исключения**

10.7.5.1 Вызов процедуры обработки исключения происходит в случае, когда есть ожидающее исключение с достаточным приоритетом и при выполнении одного из следующих условий:

- процессор находится в режиме приложения (thread mode);
- новое исключение имеет более высокий приоритет, чем исключение, которое должно быть обработано. В этом случае новое исключение перехватывает обработку.

Когда одно исключение захватывает другое, то исключения называются вложенными.

При необходимости вызова обработчика, за исключением случаев обработки запоздавшего исключения и передачи управления на обработчик без восстановления контекста, процессор заносит в текущий стек восемь слов данных, называемые далее стековым фреймом. Этот фрейм включает в себя следующие значения:

– регистры R0-R3, R12;
– адрес возврата;
– регистр PSR;
– регистр LR.

Указанная операция далее будет называться сохранением контекста. Непосредственно после ее выполнения указатель стека равен младшему адресу стекового фрейма.

Стековый фрейм содержит адрес возврата, указывающий на ближайшую невыполненную инструкцию прерванной программы. По завершении процедуры обработки исключения значение адреса возврата заносится в счетчик команд, после чего выполнение программы возобновляется с прерванной точки. Одновременно с сохранением контекста процессор осуществляет выборку адреса точки входа в процедуру обработки исключения из таблицы векторов исключений. По завершении операции сохранения контекста процессор передает управление на полученный из таблицы адрес.

При использовании процедур обработки чисел с плавающей точкой процессор автоматически сохраняет состояние плавающей точки при входе в исключение. На рисунке 10.2 показано формирование стекового фрейма, когда состояние с плавающей точкой сохраняется в стек в результате прерывания или исключения.

Примечание - в случаях, когда стековое пространство для состояния плавающей точки не определено, стековый фрейм является таким же, как и для реализации архитектуры ARMv7-M без FPU.

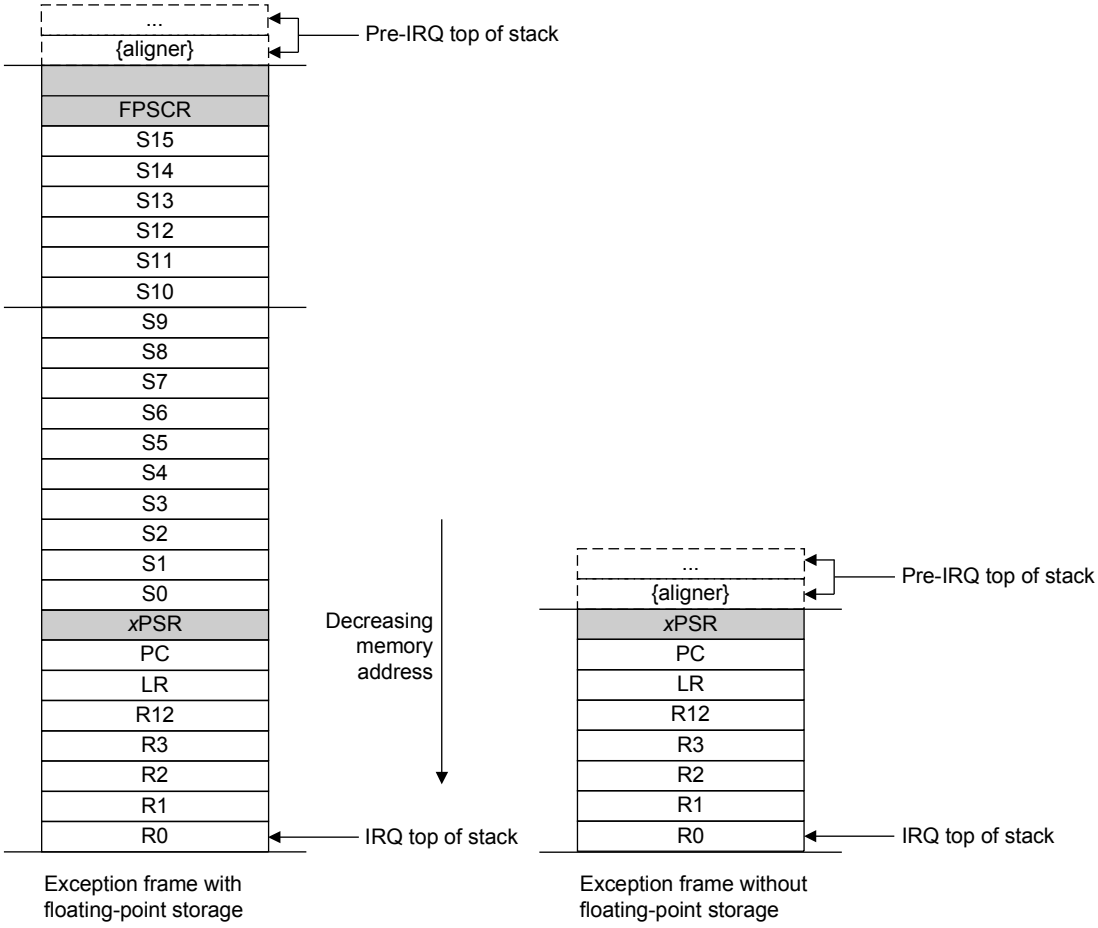

Рисунок 10.2 – Стековый фрейм

Если никакое другое исключение с более высоким приоритетом не возникает во время входа в исключение, процессор начинает выполнять обработку исключения и автоматически изменяет статус соответствующего прерывания из состояния pending в состояние active.

Если другое исключение с более высоким приоритетом возникает во время входа в исключение, процессор начинает выполнять обработку этого исключения и не изменяет статус pending более раннего исключения. Это случай «опоздавшего» исключения.

Возврат из обработчика прерывания производится в случае, если процессор находится в режиме обработчика и выполняет одну из инструкций, позволяющих загрузить значение EXC_RETURN в регистр PC:

– инструкции LDM или POP с аргументом PC;
– инструкцию LDR с регистром PC в качестве приемника;
– инструкцию BX с любым аргументом.

EXC_RETURN – значение, загружаемое в регистр LR, при входе в обработчик прерывания. Механизм обработки исключений использует это значение для того, чтобы опеределить завершил ли процессор выполнение процедуры обработки исключения. Пять младших разрядов этого значения содержат информацию о состоянии стека и режиме работы процессора. Информация о назначении разрядов EXC_RETURN и особенности процесса возврата из обработчика исключения представлены в таблице 10.2.

Все значения EXC_RETURN имеют разряды [31:5] установленные в единицу. Когда это значение загружено в PC, оно показывает процессору, что процедура обработки исключения завершена, после чего процессор инициирует соответствующую последовательность действий, необходимых для *возврата из обработчика прерывания*.

Таблица 10.2 – Возврат из обработчика исключений

EXC_RETURN[31:0]	**Описание**
0xFFFFFFF1	Возврат в режим обработчика. Восстановление контекста (не для плавающей точки) осуществляется из стека MSP.
0xFFFFFFF9	Возврат в режим потока. Восстановление контекста (не для плавающей точки) осуществляется из стека MSP.
0xFFFFFFFD	Возврат в режим потока. Восстановление контекста (не для плавающей точки) осуществляется из стека PSP.
0xFFFFFFE1	Возврат в режим обработчика. Восстановление контекста (для плавающей точки) осуществляется из стека MSP.
0xFFFFFFE9	Возврат в режим потока. Восстановление контекста (для плавающей точки) осуществляется из стека MSP.
0xFFFFFFED	Возврат в режим потока. Восстановление контекста (для плавающей точки) осуществляется из стека PSP.

10.8 Обработка отказов

10.8.1 Отказ (ошибка) – это подкласс исключений .

Отказы возникают по следующим причинам:

1. ошибка шины в процессе:
   - выборки команды или загрузки вектора обработчика;
   - доступа к данным
2. обнаруженная внутренняя ошибка, например, неопределенная команда;
3. попыткой исполнить команду из области памяти отмеченную как Non-Executable (неисполняемая);
4. отказ блока защиты памяти MPU вследствие нарушения прав доступа или вследствие попытки доступа к неподдерживаемой области адресного пространства.

10.8.2 **Типы отказов**

10.8.2.1 В таблице 10.3 показаны типы отказов, обработчики, вызываемые при их возникновении, соответствующие данному типу отказа регистры состояния и биты регистра, указывающие на конкретный отказ.

Таблица 10.3 – Отказы

Отказ	**Обработчик**	**Наименование бита регистра**	**Регистр отказа**
Ошибка доступа к шине при чтении вектора	Тяжелый отказ	VECTTBL	«Регистр состояния тяжелых отказов»
Эскалация отказа		FORCED	
Ошибка доступа к памяти:	Отказ доступа к памяти	-	«Регистр состояния отказов доступа к памяти», «Регистр адреса отказа доступа к памяти»
- при чтении команды		IACCVIOL	
-при доступе к данным		DACCVIOL	
- при сохранении контекста		MSTKERR	
- при восстановлении контекста		MUNSKERR	
Ошибка шины:	Отказ доступа к шине	-	«Регистр состояния отказа доступа к шине», «Регистр адреса отказа доступа к шине»
- при сохранении контекста		STKERR	
- при восстановлении контекста		UNSTKERR	
- при загрузке инструкции		IBUSERR	
Локализованная ошибка шины данных		PRECISERR	
Нелокализованная ошибка шины данных		IMPRECISERR	
Попытка доступа к сопроцессору	Отказ, вызванный ошибками программирования	NOCP	«Регистр состояния отказов, вызванных ошибками программирования»
Неизвестная инструкция		UNDEFINSTR	
Попытка выбора неверного набора инструкций		INVSTATE	
Неверное значение EXC_RETURN		INVPC	
Запись или чтение по неверно выровненному адресу		UNALIGNED	
Деление на 0		DIVBYZERO	

10.9 Эскалация отказов

10.9.1 Все исключительные ситуации по отказу кроме HardFaul имеют настраиваемый приоритет обработки.

Как правило, приоритет обработки исключения, вместе со значениями регистров маски определяет, будет ли процессор входить в обработчик отказа, а также – сможет ли он прервать выполнение другого обработчика.

В некоторых ситуациях, отказ с пограммируемым приоритетом обрабатывается как HardFault. Такая ситуация называется эскалацией отказа. Это возможно в следующих случаях:

– обработчик отказов во время своего выполнения вызывает отказ того же типа. Этот тип эскалации обусловлен тем фактом, что обработчик не может прервать собственное выполнение, так как его приоритет равен текущему;

– обработчик отказа вызвал отказ другого типа с приоритетом, меньшим или равным собственному. В этом случае новый обработчик также не может быть активизирован вследствие недостаточного уровня приоритета;

– обработчик исключительной ситуации вызвал отказ с приоритетом обработки, меньшим или равным текущему;

– возник отказ, обработчик которого не разрешен.

Если отказ обращения к шине возник во время загрузки данных в стек при передаче

управления на обработчик отказа доступа к шине - эскалации не происходит. Таким образом, в случае, если отказ возник вследствие разрушения стека, передача управления на обработчик отказа выполняется несмотря на то, что сохранение контекста не было осуществлено.

Обработка тяжелых отказов имеет фиксированный приоритет. Она может быть прервана только по сигналу сброса Reset или немаскируемого прерывания NMI. Сам обработчик способен прерывать обработку любых исключительных ситуаций, кроме ситуаций сброса Reset, NMI, а также другого тяжелого отказа.

10.9.2 **Регистры состояния и адреса отказа**

10.9.2.1 Регистры состояния отказа указывают причину отказа. Для отказов Bus Faults и Mem Manage регистр адреса отказа указывает адрес доступа к операции, которая вызвала отказ, как показано в таблице 10.4.

Таблица 10.4

Обработчик	**Регистр состояния**	**Регистр адреса**	**Описание регистров**
Тяжелый отказ	HFSR	-	«Регистр состояния тяжелых отказов»
Отказ доступа к памяти	MMFSR	MMFAR	«Регистр состояния отказов доступа к памяти» «Регистр адреса отказа доступа к памяти»
Отказ доступа к шине	BFSR	BFAR	«Регистр состояния отказов доступа к шине» «Регистр адреса отказа доступа к шине»
Отказ, вызванный ошибками программирования	UFSR	-	«Регистр состояни отказов, вызванных ошибками программирования»

10.10 Блокировка

10.10.1 Процессор переходит в состояние блокировки в случае, если тяжелый отказ возник во время выполнения программы-обработчика тяжелого отказа.

После перехода в состояние блокировки процессор перестает выполнять какие-либо команды. В этом состоянии он будет находиться до момента сброса.

10.11 Прерывания от периферийных блоков

10.11.1 Периферийные блоки формируют прерывания с IRQ0 до IRQ31.

Таблица 10.5 – Прерывания от периферийных блоков

Прерывания	**Блок**	**Принцип формирования**
IRQ0	MIL-STD-1553B1	Прерывание от MIL-STD-1553B1
IRQ1	MIL-STD-1553B2	Прерывание от MIL-STD-1553B2
IRQ2…IRQ4	Зарезервировано	-
IRQ5	UART1	Прерывание от UART1
IRQ6	UART2	Прерывание от UART2
IRQ7	UART3	Прерывание от UART3
IRQ8…IRQ10	Зарезервировано	-
IRQ11	SPI1	Прерывание от SPI1
IRQ12	SPI2	Прерывание от SPI2
IRQ13…IRQ14	Зарезервировано	-
IRQ15	DMA	Прерывания от DMA (DMA_ERR или DMA_DONE)
IRQ16	TIMER1	Прерывание от таймера 1
IRQ17	TIMER2	Прерывание от таймера 2
IRQ18	TIMER3	Прерывание от таймера 3
IRQ19	PORTA	Прерывание от порта A
IRQ20	PORTB	Прерывание от порта B
IRQ21	PORTC	Прерывание от порта C
IRQ22	PORTD	Прерывание от порта D
IRQ23	PORTE	Прерывание от порта E
IRQ24…IRQ27	Зарезервировано	-
IRQ28	Внешнее прерывание	Внешнее прерывание 0 (вход EXT_INT0, активный уровень - низкий)
IRQ29	Внешнее прерывание	Внешнее прерывание 1 (вход EXT_INT1, активный уровень - низкий)
IRQ30	Внешнее прерывание	Внешнее прерывание 2 (вход EXT_INT2, активный уровень - высокий)
IRQ31	Внешнее прерывание	Внешнее прерывание 3 (вход EXT_INT3, активный уровень - высокий)

11 Контроллер прерываний NVIC

11.1 В разделе описан векторный контроллер прерываний с возможностью вложения (NVIC – Nested Vectored Interrupt Controller) и используемые им регистры.

Контроллер обеспечивает следующие возможности:

– програмное задание уровня приоритета в диапазоне от 0 до 15 независимо каждому прерыванию. Более высокое значение уровня соответствует меньшему приоритету, таким образом, уровень 0 отвечает наивысшему приоритету прерывания;

– срабатывание сигнала прерывания по импульсу и по уровню;

– динамическое изменение приоритета прерываний;

– разделение исключений по группам с одинаковым приоритетом и по подгруппам внутри одной группы;

– передача управления из одного обработчика исключения в другой без восстановления контекста.

Процессор автоматически сохраняет в стеке свое состояние (контекст) по входу в обработчик прерывания и восстанавливает его по завершению обработчика, без необходимости непосредственного программирования этих операций. Это обеспечивает обработку исключительных ситуаций с малой задержкой.

Назначение регистров контроллера прерываний представлено в таблице 11.1.

Таблица 11.1 – Обобщённая информация о регистрах контроллера NVIC

Адрес	**Название**	**Тип**	**Доступ**	**Значение после сброса**	**Описание**
0xE000_E100	NVIC				Контроллер прерываний NVIC
0x000	ISER[0]	RW	Привилегированный	0x0000_0000	Регистр разрешения прерываний ISER
…					
0x01C	ISER[7]				
…					
0x080	ICER[0]	RW	Привилегированный	0x0000_0000	Регистр запрета прерываний ICER
…					
0x09C	ICER[7]				
…					
0x100	ISPR[0]	RW	Привилегированный	0x0000_0000	Регистр установки состояния ожидания для прерывания ISPR
…					
0x11C	ISPR[7]				
…					
0x180	ICPR[0]	RW	Привилегированный	0x0000_0000	Регистр сброса состояния ожидания для преырвания ICPR
…					
0x19C	ICPR[7]				
…					
0x200	IABR[0]	RO	Привилегированный	0x0000_0000	Регистр активных прерываний IABR
…					
0x21C	IABR[7]				
…					
0x300	IP[3], IP[2], IP[1], IP[0]	RW	Привилегированный	0x0000_0000	Регистр приоритета прерываний IP
…					
0x3F0	IP[239], IP[238], IP[237], IP[236]				
…					
0xE00	STIR	WO	В зависимости от конфигурации	0x0000_0000	Регистр программного формирования прерываний STIR

11.2 Упрощенный доступ к регистрам контроллера прерываний

11.2.1 В целях повышения эффективности разработки программного обеспечения в CMSIS предусмотрен упрощенный доступ к регистрам контроллера прерываний NVIC из среды разработки программного обеспечения:

- регистры разрешения, запрета, установки и сброса состояния ожидания прерываний, а также регистр активных прерываний отображаются на массивы 32-разрядных целых чисел, а именно:
- массив ISER[0] соответствует регистру ISER0;
- массив ICER[0] соответствует регистру ICER0;
- массив ISPR[0] соответствует регистру ISPR0;
- массив ICPR[0] соответствует регистру ICPR0;
- массив IABR[0] соответствует регистру IABR0;
- 4-битные поля регистра приоритета прерываний отображаются на массив 4-разрядных целых чисел, а именно:
- массив IP[0]...IP[29] соответствует регистрам IPR0-IPR7, причем элемент массива IP[n] соответствует приоритету прерывания с номером n.

CMSIS генерирует код, гарантированно обеспечивающий в условиях многозадачности корректный непрерываемый (atomic) доступ к регистрам приоритета.

В таблице 11.2 показано отображение прерываний (номеров запросов IRQ) на регистры прерываний и соответствующие переменные CMSIS, для которых предусмотрено по одному биту на прерывание.

Таблица 11.2 – Распределение прерываний в переменных прерывания

Номер прерывания	**Элементы массивов CMSIS**				
	Разрешение	**Запрет**	**Установка режима ожидания**	**Сброс режима ожидания**	**Признак активности**
0-29	ISER[0]	ICER[0]	ISPR[0]	ICPR[0]	IABR[0]

Каждый элемент массива соответствует одному регистру контроллера прерываний NVIC, например элемент ICER[1] соответствует регистру ICER1.

11.3 NVIC->ISER[x]

11.3.1 Регистр ISER0 предназначен для разрешения прерываний (запись) и определения, какие из прерываний разрешены (чтение).

Таблица 11.3 – Регистр разрешения прерываний

Номер	31…0
Доступ	R/W
Сброс	0
	SETENA bits

Назначение бит SETENA:

запись: 0 – не влияет, 1 – разрешение прерывания;

чтение: 0 – прерывание запрещено, 1 – прерывание разрешено.

При разрешении прерывания, находящегося в состоянии ожидания обработки, контроллер NVIC активизирует его в зависимости от приоритета. Запрос запрещенного прерывания переводит его в состояние ожидания обработки, однако контроллер NVIC не активизирует его вне зависимости от приоритета.

11.4 NVIC->ICER[x]

11.4.1 Регистр запрета прерываний

Регистр ICER0 предназначен для запрета прерываний (запись) и определения, какие из прерываний разрешены (чтение).

Таблица 11.4 – Регистр запрета прерываний

Номер	31…0
Доступ	R/W
Сброс	0
	CLRENA bits

Назначение бит CLRENA:
запись: 0 – не влияет, 1 – запрет прерывания;
чтение: 0 – прерывание запрещено, 1 – прерывание разрешено.

11.5 NVIC->ISPR[x]

11.5.1 Регистр установки состояния ожидания для прерывания.

Регистр ISPR0 предназначен для принудительного перевода прерываний в состояние ожидания обслуживания (запись) и определения, какие из прерываний находятся в этом состоянии (чтение).

Таблица 11.5 – Регистр установки состояния ожидания для прерывания

Номер	31…0
Доступ	R/W
Сброс	0
	SETPEND

Назначение бит SETPEND:
запись: 0 – не влияет, 1 – перевод прерывания в состояние ожидания;
чтение: 0 – прерывание не ожидает обслуживания, 1 – прерывание ожидает обслуживания.
Запись 1 в бит регистра ISPR, соответствующий:
- прерыванию, уже ожидающему обслуживания – не влияет на работу системы;
- запрещенному прерыванию – переводит его в состояние ожидания.

11.6 NVIC->ICPR[x]

11.6.1 Регистр сброса состояния ожидания для прерывания.

Регистр ICPR0 предназначен для принудительного сброса состояния ожидания обслуживания прерывания (запись) и определения, какие из прерываний находятся в состоянии ожидания (чтение).

Таблица 11.6 – Регистр сброса состояния ожидания для прерывания

Номер	31…0
Доступ	R/W
Сброс	0
	CLRPEND

Назначение бит CLRPEND:
запись: 0 – не влияет, 1 – сброс состояния ожидания;
чтение: 0 – прерывание не ожидает обслуживания, 1 – прерывание ожидает обслуживания.
Запись 1 в разряд регистра ICPR, соответствующий прерыванию в активном состоянии, не влияет на работу системы.

11.7 NVIC->IABR[x]

11.7.1 Регистр активных прерываний

Регистр IABR0 показывает, какие из прерываний находятся в активном состоянии. Этот регистр доступен только для чтения.

Таблица 11.7 – Регистр активных прерываний

Номер	31…0
Доступ	RO
Сброс	0
	ACTIVE

Назначение бит ACTIVE:

чтение: 0 – прерывание не активно;

1 – прерывание активно и обслуживается, либо активно и ожидает обслуживания.

11.8 NVIC->IP[x]

11.8.1 Регистры приоритета прерываний

Регистры IPR0-IPR7 представляют собой набор 4-битных полей, каждое из которых соответствует одному прерыванию. Регистры доступны побайтно.

Каждый из регистров содержит четыре поля приоритета, которые отображаются на четыре элемента массива IP[0] .. IP[29] CMSIS, как показано ниже.

Таблица 11.8 – Регистры приоритета прерываний

Номер	31…16	15…8	7…0
Доступ	U	R/W	R/W
Сброс	0		
	-	**IP[29]**	**IP[28]**

Номер	31…24	23…16	15…8	7…0
Доступ	U	R/W	R/W	R/W
Сброс	0	0	0	0
	IP[4m+3]	**IP[4m+2]**	**IP[4m+1]**	**IP[4m]**

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	0	0	0	0
	IP[3]	**IP[2]**	**IP[1]**	**IP[0]**

Каждое поле содержит значение приоритета в диапазоне от 0 до 15, причем меньшие значения соответствуют более высокому приоритету соответствующего прерывания. Процессор обеспечивает доступ только к битам [7:5] приоритета, биты [4:0] при чтении всегда равны нулю, а при записи игнорируются.

Номер регистра IPR и смещение данных в регистре для заданного номера прерывания N определяются следующими соотношениями:

– номер M соответствующего регистра приоритета равен M = N DIV 4;

– смещение данных в регистре в зависимости от значения N MOD 4 равно:

0 – биты регистра [7:0];

1 – биты регистра [15:8];

2 – биты регистра [23:16];

3 – биты регистра [31:24].

11.9 NVIC->STIR

11.9.1 Регистр программного формирования прерывания

Запись в регистр STIR приводит к формированию в системе программного прерывания (SGI – Software Generated Interrupt).

В случае, если бит USERSETMPEND в регистре SCR установлен в 1, возможен доступ к регистру STIR из непривилегированных приложений (см. “Регистр управления системой”).

Установка этого бита возможна только из привилегированного режима работы процессора.

Таблица 11.9 – Регистр программного формирования прерывания

Номер	31…9	8…0
Доступ	U	R/W
Сброс	0	0
	-	**INTID**

INTID – идентификатор формируемого прерывания в диапазоне 0 – 239. Например: значение b000000011 соответствует прерыванию IRQ3.

11.10 Прерывания, срабатывающие по уровню сигнала

11.10.1 Процессор способен обрабатывать прерывания, сформированные по уровню сигнала.

Прерывание такого типа считается активным до тех пор, пока периферийное устройство не снимет активный уровень сигнала запроса. Как правило, это происходит после соответствующего обращения процедуры обработки прерывания к периферийному устройству.

После того, как процессор передал управление на обработчик, он автоматически снимает признак ожидания обслуживания прерывания (см. раздел “Аппаратное и программное управление прерываниями”). Если прерывание формируется по уровню сигнала, а сигнал запроса не снят до возврата из обработчика, процессор вновь переведет прерывание в состояние ожидания обслуживания, что, в свою очередь, приведет к повторному вызову его обработчика.

Таким образом, периферийное устройство может поддерживать сигнал запроса прерывания в активном состоянии до тех пор, пока не перестанет нуждаться в обслуживании.

11.11 Аппаратное и проргаммное управление прерываниями

11.11.1 Процессор Cortex-M3 регистрирует все поступающие прерывания. Перевод прерывания, сформированного периферийным устройством, в состояние ожидания обслуживания осуществляется в одном из следующих случаев:

– контроллер прерываний NVIC обнаруживает, что сигнал запроса имеет высокий логический уровень, а прерывание неактивно;

– контроллер прерываний NVIC обнаруживает передний фронт сигнала запроса прерывания;

– программное обеспечение осуществляет запись в соответствующий разряд регистра ISPR0 или соответствующего значения в регистр STIR.

Прерывание находится в состоянии ожидания до тех пор, пока не произойдет одно из следующих событий:

– процессор передаст управление процедуре обработки прерывания. В этом случае прерывание переходит в активное состояние, после чего:

– по завершении обработки прерывания, срабатывающего по уровню, контроллер NVIC проверяет состояние сигнала запроса на прерывание. Если этот сигнал активен, прерывание вновь переводится в состояние ожидания обслуживания, что приводит к

немедленной повторной передаче управления на обработчик. В противном случае прерывание переводится в неактивное состояние;

– если в период выполнения процедуры обработки прерывания, настроенного на срабатывание по фронту, не было зафиксировано импульсов на линии запроса, прерывание переводится в неактивное состояние.

– программное обеспечение осуществляет запись в соответствующий разряд регистра сброса состояния ожидания прерывания.

11.12 Рекомендации по работе с контроллером прерываний

11.12.1 Доступ к регистрам контроллера из программного обеспечения должен осуществляться по корректно выровненным адресам. Процессор не поддерживает возможность доступа к контроллеру по невыровненным адресам. Требования по выравниванию приведены в описании регистров.

Прерывание может быть переведено в состояние ожидания обслуживания даже в случае, если оно запрещено.

Перед установкой нового адреса таблицы векторов прерывания необходимо убедиться, что элементы новой таблицы корректно проинициализированы адресами обработчиков отказов и всех разрешенных исключений, в частности, прерываний.

Программное разрешение или запрещение прерываний может осуществляться с помощью инструкций CPSIE I и CPSID I. В CMSIS предусмотрены следующие встроенные функции, генерирующие эти инструкции:

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

Кроме того, в CMSIS имеется ряд дополнительных функций, обеспечивающих управление контроллером прерываний NVIC:

Таблица 11.10 – Функции CMSIS для управления контроллером прерываний

Функция	Описание
void NVIC_SetPriorityGrouping (uint32_t riority_grouping)	Установить группировку приоритетов
void NVIC_EnableIRQ(IRQn_t IRQn)	Разрешить IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Запретить IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Вернуть TRUE, если прерывание IRQn ожидает обслуживания, FALSE – в противном случае
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Перевести IRQn в состояние ожидания обслуживания
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Сбросить состояние ожидания обслуживания для IRQn
uint32_t NVIC_GetActive (IRQn_t IRQn)	Вернуть номер IRQ текущего активного прерывания
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Установить приоритет для IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Считать приоритет IRQn
void NVIC_SystemReset (void)	Сбросить систему

12 Блок управления системой

12.1 Блок управления системой SCB обеспечивает доступ к информации о конфигурации и управление работой системы. Регистры блока управления системой представлены в таблице 12.1.

Таблица 12.1 – Обобщенная информация о регистрах блока управления системой

Адрес	Имя	Тип	Доступ	Значение после сброса	Описание
0xE000_E000	InterruptType				
0x008	ACTLR	RW	Привилегированный	0x0000_0000	Дополнительный регистр управления
0xE000ED00	SCB				Блок управления системой
0x000	CPUID	RO	Привилегированный	0x412F_C230	Регистр идентификации процессора
0x004	ICSR	RW	Привилегированный	0x0000_0000	Регистр управления прерываниями
0x008	VTOR	RW	Привилегированный	0x0000_0000	Регистр смещения таблицы векторов прерываний
0x00C	AIRCR	RW	Привилегированный	0xFA05_0000	Регистр управления прерываниями и программного сброса
0x010	SCR	RW	Привилегированный	0x0000_0000	Регистр управления системой
0x014	CCR	RW	Привилегированный	0x0000_0200	Регистр конфигурации и управления
0x018	SHPR1	RW	Привилегированный	0x0000_0000	Регистр №1 приоритета системных обработчиков
0x01C	SHPR2	RW	Привилегированный	0x0000_0000	Регистр №2 приоритета системных обработчиков
0x020	SHPR3	RW	Привилегированный	0x0000_0000	Регистр №3 приоритета системных обработчиков
0x024	SHCRS	RW	Привилегированный	0x0000_0000	Регистр управления и состояния системных обработчиков
0x028	CFSR	RW	Привилегированный	0x0000_0000	Регистр состояния отказов с конфигурируемым приоритетом
0x028	MMSR	RW	Привилегированный	0x00	Регистр состояния отказов доступа к памяти
0x029	BFSR	RW	Привилегированный	0x00	Регистр состояния отказов доступа к шине
0x02A	UFSR	RW	Привилегированный	0x0000	Регистр состояния отказов, вызванных ошибками программирования
0x02C	HFSR	RW	Привилегированный	0x0000_0000	Регистр состояния тяжелого отказа
0x034	MMAR	RW	Привилегированный	Не определено	Регистр адреса отказа доступа к памяти
0x038	BFAR	RW	Привилегированный	Не определено	Регистр адреса отказа доступа к шине

12.2 Упрощенный доступ к регистрам блока управления системой

12.2.1 В целях повышения эффективности в CMSIS предусмотрен упрощенный доступ к регистрам SCB из среды разработки программного обеспечения, а именно, регистры SHPR1-SHPR3 в CMSIS отображаются на массив байтов SHP[0]...SHP[12].

12.2.2 **InterrupType->ACTLR**

12.2.2.1 Дополнительный регистр управления

Регистр ACTLR позволяет разрешить или запретить следующие возможности процессора:

– вложение условных инструкций (IT folding);

– использование буферизации записи в режиме отображения памяти по умолчанию (default memory map);

– прерывание многоэлементных инструкций чтения и записи регистров.

Обобщенные данные о регистре ACTLR представлены далее в таблице 12.2

Таблица 12.2 – Дополнительный регистр управления

Номер	31…3	2	1	0
Доступ	U	R/W	R/W	R/W
Сброс	0	0	0	0
	-	**DISFOLD**	**DISDEFWBUF**	**DISMCYCINT**

DISFOLD – установка разряда в 1 запрещает вложение условных инструкций (IT folding).

DISDEFWBUF – установка в 1 запрещает использование буфера записи при работе в режиме отображения памяти по умолчанию (default memory map). Это обеспечивает возможность локализовать любые отказы шины, однако приводит к снижению производительности системы, так как все операции записи данных в память должны быть завершены до того, как процессор перейдет к выполнению следующей инструкции. Данный бит влияет исключительно на функционирование буферов записи, реализованных в процессоре Cortex-M4.

DISMCYCINT – установка бита в 1 запрещает прерывание многоэлементных инструкций чтения и записи регистров (LDM и STM). Это приводит к увеличению задержки обработки прерываний, вследствие необходимости завершения выполнения инструкций LDM или STM перед началом сохранения контекста и передачи управления обработчику прерывания.

В некоторых случаях процессор может начать выполнение первой инструкции в IT-блоке, все еще выполняя инструкцию IT. Эта возможность, называемая далее вложением условных инструкций (IT folding), позволяет увеличить производительность системы, однако может привести к непостоянству времени выполнения тела цикла программы («джиттеру»). В случае, если в разрабатываемом приложении это нежелательно, следует установить бит DISFOLD в 1.

12.2.3 **SCB->CPUID**

12.2.3.1 Регистр идентификации процессора.

Регистр CPUID содержит информацию о модели процессора, версии и варианте его реализации. Подробная информация о регистре представлена в таблице 12.3.

Таблица 12.3 – Регистр идентификации процессора

Номер	31…24	23…20	19…16	15…4	3…0
Доступ	RO	RO	RO	RO	RO
Сброс	0x41	0x2	0xF	0xC24	0x0
	Implementer	**Variant**	**Constant**	**PartNo**	**Revision**

Implementer – код разработчика 0x41 = ARM.

Variant – значение r в номере версии rnpn изделия: 0x2 = r2p0;

Constant – постоянное значение 0xF;

PartNo – номер модели процессора: 0xC24 = Cortex-M4;

Revision – значение p в номере версии rnpn изделия: 0x0 = r2p0.

12.2.4 **SCB->ICSR**

12.2.4.1 Регистр управления прерываниями.

Регистр ICSR обеспечивает возможность установки и сброса состояния ожидания обслуживания для исключений PendSV и SysTick, а также доступ к следующей информации:

– номер текущего обрабатываемого исключения;
– наличие активных исключений, обработка которых была прервана;
– номер исключения, ожидающего обслуживания, с наивысшим приоритетом;
– наличие прерываний, ожидающих обслуживания.

Таблица 12.4 – Регистр управления прерываниями

Номер	31…29	28	27	26	25	24	23	22	21…12	11	10	9	8…0
Доступ	RO	R/W	R/W	R/W	R/W	U	R/W	R/W	R/W	R/W	U	U	R/W
Сброс	0	0	0	0	0	0	0	0	0	0	0	0	0
	-	**PENDSVSET**	**PENDSVCLR**	**PENDSTSET**	**PENDSTCLR**	-	**Reserved for Debug**	**ISRPENDING**	**VECTPENDING**	**RETTOBASE**	-	-	**VECTACTIVE**

PENDSVSET (RW) – бит установки состояния ожидания обслуживания для исключения PendSV.

Запись 0 – не влияет на работу системы, 1 – переводит исключение PendSV в состояние ожидания обслуживания.

Чтение 0 – исключение PendSV не ожидает обслуживания, 1 – ожидает.

Запись 1 – это единственно возможный способ перевода исключения PendSV в состояние ожидания обслуживания.

PENDSVCLR (WO) – бит сброса состояния ожидания обслуживания для исключения PendSV.

Запись 0 – не влияет на работу системы.

Запись 1 – сбрасывает состояние ожидания обслуживания для исключения PendSV.

PENDSTSET (RW) – бит установки состояния ожидания обслуживания для исключения SysTick.

Запись 0 – не влияет на работу системы.

Запись 1 – переводит исключение SysTick в состояние ожидания обслуживания.

Чтение 0 – исключение SysTick не ожидает обслуживания. 1 – ожидает.

PENDSTCLR (WO) – бит сброса состояния ожидания обслуживания для исключения SysTick.

Запись 0 – не влияет на работу системы.

Запись 1 – сбрасывает состояние ожидания обслуживания для исключения SysTick.

Данный бит доступен только для записи, при чтении результат не определен.

Reserved for Debug use (RO) – этот бит зарезервирован для целей отладки, при чтении вне режима отладки возвращает значение 0.

ISRPENDING (RO) – флаг наличия в системе прерываний (за исключением отказов), ожидающих обслуживания. 0 – ожидающие обслуживания прерывания отсутствуют, 1 – присутствуют.

VECTPENDING (RO) – содержит номер ожидающего обслуживания исключения с наивысшим приоритетом, обработка которого в системе разрешена. 0 – необслуженных исключений нет, другое число – номер ожидающего обслуживания исключения.

Значение данного поля формируется с учетом полей BASEPRI и FAULTMASK, однако не учитывает влияние поля PRIMASK.

RETTOBASE (RO) – показывает наличие в системе активных исключений, обслуживание которых было прервано. 0 – присутствуют, 1 – отсутствуют.

VECTACTIVE (RO) – содержит номер активного исключения. 0 – режим приложения, другое число – номер текущего обслуживаемого исключения. Для получения номера запроса прерывания (IRQ) из значения VECTACTIVE необходимо вычесть 16.

Запись в регистр ICSR может привести к непредсказуемым результатам в случае:

– одновременной установки в 1 бит PENDSVSET и PENDSVCLR;

– одновременной установки в 1 бит PENDSTSET и PENDSTCLR.

12.2.5 **SCB->VTOR**

12.2.5.1 Регистр смещения таблицы векторов прерываний

Регистр VTOR содержит смещение базового адреса таблицы векторов прерываний относительно адреса 0x00000000.

Таблица 12.5 – Регистр смещения таблицы векторов прерываний

Номер	31	30	29	7	6…0
Доступ	U	U	R/W	R/W	R/W
Сброс	0	0	0	0	0
	-		**TBLOFF**		**Reserved**

TBLOFF – смещение базового адреса таблицы векторов относительно нижней границы карты распределения памяти. Собственно смещение хранится в битах [28:7]. Бит [29] определяет, размещена ли таблица в области кода или в области памяти SRAM: 0 = область кода, 1 = SRAM. Бит [29] может также обозначаться как TBLBASE.

При установке значения TBLOFF требуется обеспечить выравнивание базового адреса таблицы векторов. Минимальный размер выравнивания – по границе блока из 32 слов, достаточен для хранения 16 векторов прерываний. Для поддержки большего количества прерываний необходимо увеличить размер выравнивания до ближайшей степени двойки, большей или равной размеру таблицы. Например, для хранения 21 вектора прерываний таблицу следует выровнять по границе блока из 64 слов, так как ее объем составляет 37 слов, а ближайшая степень двойки, большая или равная 37, равна 64.

Учитывая описанные выше требования по выравниванию, разряды [6…0] смещения всегда равны нулю.

12.2.6 **SCB->AIRCR**

12.2.6.1 Регистр управления прерываниями и программного сброса

Регистр AIRCR позволяет группировать исключения по приоритетам, задавать порядок следования байтов в слове (endian) при доступе к данным, а также управлять процессом сброса системы.

Для записи данных в регистр необходимо установить его поле VECTKEY в значение 0x05FA, в противном случае попытка записи будет проигнорирована процессором.

Таблица 12.6 – Регистр управления прерываниями и программного сброса

Номер	31…16	15	14…11	10…8	7…3	2	1	0
Доступ	R/W	R/W	U	R/W	U	R/W	R/W	R/W
Сброс	0	0	0	0	0	0	0	0
	On Read: VECTKEYSTAT, On Write: VECTKEY	**ENDIANESS**	-	**PRIGROUP**	-	**SYSRESETREQ**	**VECTCLRACTIVE**	**VECTRESET**

VECTKEYSTAT – ключ доступа к регистру. При чтении возвращает 0xFA05.

VECTKEY – ключ доступа к регистру. При записи должен быть равен 0x05FA, в противном случае попытка записи в регистр будет проигнорирована процессором.

ENDIANESS (RO) – порядок следования значащих разрядов при доступе к данным. 0 – младший байт идет первым (little-endian), 1 – старший байт идет первым (big-endian). Значение поля устанавливается, исходя из уровня конфигурационного сигнала BIGEND в момент сброса системы.

PRIGROUP (RW) – группировка приоритетов исключений. Значение данного поля определяет положение двоичной точки, разделяющей поле приоритета на поля номера группы и подгруппы приоритетов.

PRIGROUP - определяет позицию двоичной точки, разделяющей поля PRI_n регистров приоритета прерываний на два подполя – номер группы и номер подгруппы. Зависимость этого разбиения от значения PRIGROUP показана в таблице 12.7.

Таблица 12.7 – Группировка приоритетов прерываний

PRIGROUP	**Значение приоритета в поле PRI_N[7:0]**			**Общее количество**	
	Положение двоичной точки	**Биты номера группы**	**Биты номера подгруппы**	**групп**	**подгрупп**
b011	bxxxx.0000	[7:4]	None	16	1
b100	bxxx.y0000	[7:5]	[4]	8	2
b101	bxx.yy0000	[7:6]	[5:4]	4	4
b110	bx.yyy0000	[7]	[6:4]	2	8
b111	b.yyyy0000	None	[7:4]	1	16

SYSRESETREQ (WO) – запрос сброса системы. 0 – не влияет на работу, 1 – инициирует сигнал сброса процессора. При чтении возвращает 0.

VECTCLRACTIVE (WO) – зарезервировано для целей отладки. При чтении возвращает 0. При записи данных в регистр значение поля должно быть равно 0, в противном случае результат непредсказуем.

VECTRESET (WO) – зарезервировано для целей отладки. При чтении возвращает 0. При записи данных в регистр значение поля должно быть равно 0, в противном случае результат непредсказуем.

12.2.7 **SCB->SCR**

12.2.7.1 Регистр управления системой

Регистр SCR позволяет определить требования к переходу в режим и выходу из режима пониженного энергопотребления.

Таблица 12.8 – Регистр управления системой

Номер	31…5	4	3	2	1	0
Доступ	U	R/W	U	R/W	R/W	U
Сброс	0	0	0	0	0	0
	-	**SEVONPEND**	**-**	**SLEEPDEEP**	**SLEEONEXIT**	**-**

SEVONPEND – разрешает или запрещает формирование сигнала события при переводе исключения в состояние ожидания обработки. 0 – выход из режима пониженного энергопотребления по прерыванию могут инициировать только разрешенные прерывания или события; 1 – выход может инициироваться разрешенными событиями и любыми, в том числе запрещенными, прерываниями.

Перевод прерывания в состояние ожидания обслуживания формирует событие, что в свою очередь приводит к выходу процессора из режима пониженного потребления, инициированного инструкцией WFE, либо к регистрации факта события, если эта инструкция еще не выполнялась.

Кроме того, процессор может быть выведен из режима пониженного энергопотреблении при поступлении внешнего события, а также после выполнения инструкции SEV.

SLEEPDEEP – определяет режим пониженного энергопотребления процессора:

0 – спящий режим (Sleep);

1 – режим глубокого сна (Deep Sleep).

SLEEPONEXIT – разрешает или запрещает перевод процессора в режим пониженного энергопотребления при выходе из обработчика события в режим выполнения прикладной программы: 0 – не переводить, 1 – переводить.

12.2.8 **SCB->CCR**

12.2.8.1 Регистр конфигурации и управления

Регистр CCR управляет процессом перехода процессора в режим приложения, а также позволяет запретить или разрешить:

– игнорирование отказов доступа к шине в обработчиках тяжелых отказов и при эскалации отказа по FAULTMASK;

– генерацию исключений при делении на ноль и при доступе по невыровненному адресу;

– доступ к регистру STIR из непривилегированного приложения.

Таблица 12.9 – Регистр конфигурации и управления

Номер	31…10	9	8	7…5	4	3	2	1	0
Доступ	U	R/W	R/W	U	R/W	R/W	U	R/W	R/W
Сброс	0	0	0	0	0	0	0	0	
	-	**STKALIGN**	**BFHFNMIGN**	**-**	**DIV_0_TRP**	**UNALIGN_TRP**	**-**	**USERSETMPEND**	**NONBASETHRDENA**

STKALIGN определяет режим выравнивания адреса стека при обработке исключений: 0 - выравнивание по границе 4 байт; 1 - выравнивание по границе 8 байт. При передаче управления на обработчик исключения процессор анализирует бит [9] сохраненного в стеке слова состояния PSR и определяет по нему режим выравнивания стека. При возврате из обработчика процессор использует сохраненный в стеке бит этого слова для восстановления требуемого режима выравнивания.

BFHFNMIGN разрешает обработчикам с уровнем приоритета 1 и 2 игнорировать отказы доступа к шине, вызванные инструкциями чтения и записи. Бит влияет на функционирование обработчиков тяжелых отказов и при эскалации отказов по FAULTMASK. 0 = отказы доступа к шине данных, вызванные инструкциями чтения или записи, приводят к блокировке процессора;

1 = обработчики с уровнем приоритета 1 и 2 игнорируют указанные отказы доступа к шине данных. Данный бит следует устанавливать лишь в том случае, если обработчик и используемые им данные размещены в абсолютно безопасной области памяти. Как правило, данный бит используется для локализации и исправления проблем доступа к системным устройствам и мостам ввода-вывода.

DIV_0_TRP разрешает процессору формировать отказ или останавливаться в случае деления на ноль при выполнении инструкций SDIV или UDIV. 0 = не обрабатывать деление на 0. 1 = обрабатывать. В случае, если бит установлен в 0, при делении на ноль процессор устанавливает частное в 0.

UNALIGN_TRP разрешает процессору формировать отказ при невыровненном доступе к данным. 0 = не обрабатывать невыровненный доступ к словам или полусловам данных. 1 = обрабатывать. Если бит равен 1, то невыровненный доступ приводит к отказу, вызванному ошибкой программирования (usage fault).

В случае невыровненного доступа по инструкциям LDM, STM, LDRD или STRD отказ формируется всегда, вне зависимости от значения бита UNALIGN_TRP.

USERSETMPEND разрешает доступ к регистру STIR из непривилегированного приложения. 0 = доступ запрещен, 1 = разрешен.

NONEBASETHRDENA определяет процедуру перехода процессора в режим приложения (Thread mode): 0 = процессор может перейти в режим приложения только в случае отсутствия активных исключений, 1 = процессор может перейти в режим приложения из обработчика любого уровня, в соответствии со значением слова EXC_RETURN.

12.2.9 **SCB->SHP[x]**

12.2.9.1 Регистры приоритета системных обработчиков

Регистры приоритета системных обработчиков SHPR1-SHPR3 позволяют установить уровень приоритета обработки исключений.

Доступ к регистрам осуществляется побайтно.

Поля PRI_N регистров имеют ширину 8 бит, однако в процессоре реализована поддержка доступа только к старшему полубайту [7…4], при чтении данных из младшего полубайта [3…0] процессор возвращает нули, запись в этот полубайт игнорируется.

Таблица 12.10 – Поля приоритета обработчиков системных отказов

Обработчик отказа	Поле	Описание регистра
Отказ доступа к памяти	SHP[4]	Регистр №1 приоритета системных обработчиков
Отказ доступа к шине	SHP[5]	
Ошибка программирования (usage fault)	SHP[6]	
Вызов SVCall	SHP[11]	Регистр №2 приоритета системных обработчиков
Вызов PendSV	SHP[14]	Регистр №3 приоритета системных обработчиков
Вызов SysTick	SHP[15]	

Регистр №1 приоритета систмных обработчиков.

Таблица 12.11 – Регистр №1 приоритета системных обработчиков

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	0	0	0	0
	PRI_7: Резерв	**PRI_6**	**PRI_5**	**PRI_4**

PRI_7 - Резерв.
PRI_6 - Приоритет системного обработчика 6, ошибка программирования
PRI_5 - Приоритет системного обработчика 5, отказ доступа к шине
PRI_4 - Приоритет системного обработчика 4, отказ доступа к памяти

Регистр №2 приоритета системных обработчиков.
Таблица 12.12 – Регистр №2 приоритета системных обработчиков

Номер	31…24	23…0
Доступ	R/W	U
Сброс	0	0
	PRI_11	**-**

PRI_11 - Приоритет системного обработчика 11, вызов SVCall

Регистр №3 приоритета системных обработчиков.

Таблица 12.13 – Регистр №3 приоритета системных обработчиков

Номер	31…24	23…16	15…0
Доступ	R/W	U	
Сброс	0	0	
	PRI_11	**-**	

PRI_15 Приоритет системного обработчика 15, вызов SysTick
PRI_14 Приоритет системного обработчика 14, вызов PendSV

12.2.10 **SCB->SHCSR**

12.2.10.1 Регистр управления и состояния системных обработчиков.

Регистр SHCSR позволяет разрешить или запретить работу системных обработчиков, а также содержит сведения:

– о наличии ожидающих обработки отказов доступа к шине, управления памятью, а также вызов SVCall;

– об активных системных обработчиках.

Таблица 12.14 – Регистр управления и состояния системных обработчиков

Номер	31…19	18	17	16	15	14	13	12	11	10	9	8	7	6…4	3	2	1	0
Доступ	U	R/W	R/W	U	R/W	R/W	U	R/W	R/W	R/W	U	R/W	R/W	U	R/W	U	R/W	R/W
Сброс	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	-	**USGFAULTENA**	**BUSFAULTENA**	**MEMFAULTENA**	**SVCALLPENDED**	**BUSFAULTPENDED**	**MEMFAULTPENDED**	**USGFAULTPENDED**	**SYSTICKACT**	**PENDSVACT**	**-**	**MONITORACT**	**CVCALLAVCT**	**-**	**USGFAULTACT**		**BUSFAULTACT**	**MEMFAULTACT**

USGFAULTENA – разрешение обработки отказов, вызванных ошибками программирования, 1 – разрешено, 0 – запрещено.

BUSFAULTENA – разрешение обработки отказа доступа к шине, 1 – разрешено, 0 – запрещено.

MEMFAULTENA – разрешение обработки отказа доступа к памяти, 1 – разрешено, 0 – запрещено.

SVCALLPENDED – признак ожидания обработки вызова SVC, возвращает 1, если вызов ожидает обработки.

BUSFAULTPENDED – признак ожидания обработки отказа доступа к шине, возвращает 1, если отказ ожидает обработки.

MEMFAULTPENDED – признак ожидания обработки отказа доступа к памяти, возвращает 1, если отказ ожидает обработки.

USGFAULTPENDED – признак ожидания обработки отказа, вызванного ошибками программирования, возвращает 1, если отказ ожидает обработки.

SYSTICKACT – признак активности обработчика исключения SysTick, возвращает 1, если обработчик активен.

PENDSVACT – признак активности обработчика исключения PendSV, возвращает 1, если обработчик активен.

MONITORACT – признак активности монитора отладчика, возвращает 1, если монитор отладчика активен.

SVCALLACT – признак активности обработчика вызова SVC, возвращает 1, если обработчик активен.

USGFAULTACT – признак активности обработчика отказа, вызванного ошибкой программирования, возвращает 1, если обработчик активен.

BUSFAULTACT – признак активности обработчика отказа доступа к шине, возвращает 1, если обработчик активен.

MEMFAULTACT – признак активности обработчика отказа доступа к памяти, возвращает 1, если обработчик активен.

Примечания:

1. Установка бита разрешения в 1 разрешает обработку исключения, установка в 0 – запрещает.

2. Чтение 1 из бита-признака активности свидетельствует об активности исключения, 0 – о его неактивности. Существует возможность записи значения в данный бит для принудительного перевода исключения в активное состояние, однако при этом следует предпринять меры предосторожности, описанные далее в разделе.

3. Чтение 1 из бита-признака ожидания свидетельствует о том, что исключение находится в состоянии ожидания обработки. Существует возможность принудительного перевода исключения в состояние ожидания путем записи 1 в данный бит.

Если в системе возникло исключение (отказ), обработчик которого запрещен, процессор формирует запрос на обработку тяжелого отказа.

Существует возможность принудительного перевода того или иного системного исключения в состояние ожидания обработки или активное состояние путем записи в соответствующий разряд регистра SHCSR.

Например, ядро операционной системы может осуществлять запись в биты – признаки активности для того, чтобы осуществить переключение контекста со сменой типа обрабатываемого исключения.

Программа, меняющая значение бит – признаков активности исключения, должна обеспечить необходимую корректировку содержимого стека, в противном случае процессор может сгенерировать отказ. Необходимо убедиться, что программа сохраняет и впоследствии корректно восстанавливает текущее значение признаков активности исключений.

После разрешения системных обработчиков все дальнейшие манипуляции с битами регистра необходимо производить, последовательно выполняя операции чтения, модификации и обратной записи, гарантирующие изменение только необходимых разрядов регистра.

12.2.11 **SCB->CFSR**

12.2.11.1 Регистр состояния отказов с конфигурируемым уровнем приоритета.

Регистр CFSR содержит информацию о причине возникновения отказов управления памятью, отказов доступа к шине и ошибок программирования (usage fault).

Таблица 12.15 – Регистр состояния отказов с конфигурируемым уровнем приоритета

Номер	31…16	15…8	7…0
Доступ	RO	RO	RO
Сброс	0	0	0
	Usage Fault Status Register: UFSR	**Bus Fault Status Register: BFSR**	**Memory Management Fault Status Register: MMFSR**

Регистр CFSR доступен побайтно. Возможны следующие варианты доступа к регистру CFSR и его отдельным элементам:

- слово по адресу 0xE000ED28 – полный регистр CFSR;
- байт по адресу 0xE000ED28 – регистр MMFSR;
- полуслово по адресу 0xE000ED28 – регистры MMFSR и BFSR;
- байт по адресу 0xE000ED29 – регистр BFSR;
- полуслово по адресу 0xE000ED2A – регистр UFSR.

В последующих подразделах подробно описаны элементы, составляющие регистр CFSR:

- регистр состояния отказов доступа к памяти;
- регистр состояния отказов доступа к шине;
- регистр состояния отказов, вызванных ошибками программирования.

12.2.11.2 Поле MMFSR. Регистр состояния отказов доступа к памяти. Регистр MMFSR содержит набор флагов, указывающих на различные причины отказа доступа к памяти.

Таблица 12.16 – Регистр состояния отказов доступа к памяти

Номер	7	6	5	4	3	2	1	0
Доступ	RO	U	U	RO	RO	U	RO	RO
Сброс	0	0	0	0	0	0	0	0
	MMARVALID	-		**MSTKERR**	**MUNSTKERR**	-	**DACCVIOL**	**IACCVIOL**

MMARVALID признак корректности значения в регистре адреса отказа доступа к памяти (MMAR): 0 = значение в MMAR не содержит корректный адрес отказа, 1 = содержит.

В случае, если произошла эскалация отказа доступа к памяти, обработчик тяжелого отказа должен установить этот бит в 0. В противном случае после возврата в обработчик

отказа доступа к памяти возможна его некорректная работы, так как значение регистра MMAR будет изменено.

MSTKERR признак отказа на этапе сохранения в стеке контекста при передаче управления на обработчик исключения: 0 = отсутствует, 1 = попытка сохранения в стеке контекста при вызове обработчика исключения вызвала одно или несколько нарушений доступа к памяти. В случае, если бит равен 1, значение указателя стека SP по прежнему корректно, однако содержимое стека может быть неверным. Адрес отказа в регистр MMAR не записывается.

MUNSTKERR признак отказа на этапе восстановления контекста из стека при выходе из обработчика исключения: 0 = отсутствует, 1 = попытка восстановления контекста из стека вызвала одно или несколько нарушений доступа к памяти.

Передача управления на обработчик данного отказа осуществляется без сохранения контекста.

Таким образом, в случае, если данный бит равен 1, состояние стека сохраняется, значение указателя стека не меняется, контекст не сохраняется.

Адрес отказа в регистр MMAR не записывается.

DACCVIOL признак нарушения доступа к памяти данных: 0 = отсутствует, 1 = процессор попытался прочесть или записать данные в области, для которой не разрешон такой тип доступа. Если бит равен 1, значение счетчика команд PC, сохраненное в стеке, указывает на инструкцию, вызвавшую отказ. В регистре MMAR содержится адрес, по которому была осуществлена попытка доступа к памяти.

IACCVIOL признак нарушения доступа к памяти команд: 0 = отсутствует, 1 = процессор попытался считать очередную команду из области памяти, для которой не разрешено выполнение. Этот отказ возникает всякий раз при доступе к области, помеченной как неразрешенная для выполнения (XN), даже в случае, если блок защиты памяти MPU не активен (disabled) или отсутствует. Если бит равен 1, значение счетчика команд PC, сохраненное в стеке, указывает на инструкцию, вызвавшую отказ. Адрес отказа в регистр MMAR не записывается.

12.2.11.3 Поле BFSR. Регистр состояния отказов доступа к шине. Регистр BFSR содержит набор флагов, указывающих на различные причины отказа доступа к шине.

Таблица 12.17 – Регистр состояния отказов доступа к шине

Номер	7	6	5	4	3	2	1	0
Доступ	RO	U	U	RO	RO	RO	RO	RO
Сброс	0	0	0	0	0	0	0	0
	BFRVALID			**STKERR**	**UNSTKERR**	**IMPRECISERR**	**PRECISERR**	**IBUSERR**

BFARVALID признак корректности значения в регистре адреса отказа доступа к шине (BFAR): 0 = значение в BFAR не содержит корректный адрес отказа, 1 = содержит.

Процессор устанавливает этот бит в 1 в случае, если известен адрес, при доступе по которому произошел отказ. Возникновение впоследствии других отказов, например отказов управления памятью, может сбросить этот бит в 0.

В случае, если возникла эскалация отказа, обработчик тяжелого отказа должен установить этот бит в 0. В противном случае после возврата в обработчик отказа доступа к шине возможна его некорректная работы, так как значение регистра MMAR будет изменено.

STKERR признак отказа на этапе сохранения в стеке контекста при передаче управления на обработчик исключения: 0 = отсутствует, 1 = попытка сохранения в стеке контекста при вызове обработчика исключения вызвала одно или несколько нарушений доступа к шине. В случае, если бит равен 1, значение указателя стека SP по прежнему корректно, однако содержимое стека может быть неверным. Адрес отказа в регистр BFAR не записывается.

UNSTKERR признак отказа на этапе восстановления контекста из стека при выходе из обработчика исключения: 0 = отсутствует, 1 = попытка восстановления контекста из стека вызвала одно или несколько нарушений доступа к шине.

Передача управления на обработчик данного отказа осуществляется без сохранения контекста.

Таким образом, в случае, если данный бит равен 1, состояние стека сохраняется, значение указателя стека не меняется, контекст не сохраняется.

Адрес отказа в регистр BFAR не записывается.

IMPRECISERR признак нелокализованной ошибки доступа к шине данных. 0 = отсутствует, 1 = произошла ошибка доступа к шине данных, однако адрес возврата в стековом фрейме не указывает на инструкцию, вызвавшую ошибку. В случае, если процессор установил этот бит в 1, адрес отказа в регистр BFAR не записывается. Данный отказ является асинхронным, таким образом, если он возник внутри процесса, приоритет которого выше, чем приоритет обработки отказа шины, процессор переводит его в состояние ожидания обслуживания до завершения более приоритетных процессов. В случае, если до передачи управления на обработчик возникла также локализованная ошибка доступа к шине, процессор устанавливает оба соответствующих флага.

PRECISERR признак локализованной ошибки доступа к шине данных. 0 = отсутствует, 1 = произошла ошибка доступа к шине данных, при этом адрес возврата в стековом фрейме указывает на инструкцию, вызвавшую ошибку. В случае, если процессор установил этот бит в 1, он также записывает адрес отказа в регистр BFAR.

IBUSERR признак ошибки доступа к шине инструкций. 0 = отсутствует, 1 = произошла ошибка доступа к шине инструкций. Процессор обнаруживает факт ошибки доступа к шине инструкций на этапе выборки очередной команды, однако признак IBUSERR устанавливается только после попытки выполнения этой инструкции. В случае, если процессор установил этот бит в 1, адрес отказа в регистр BFAR не записывается.

12.2.11.4 Поле UFSR. Регистр состояния отказов, вызванных ошибками программирования. Регистр UFSR содержит набор флагов, указывающих на различные причины отказа.

Таблица 12.18 – Регистр состояния отказов, вызванных ошибками программирования

Номер	15…10	9	8	7…4	3	2	1	0
Доступ	U	RO	RO	U	RO	RO	RO	RO
Сброс	0	0	0	0	0	0	0	0
	-	**DIVBYZERO**	**UNALIGNED**	-	**NOCP**	**INVPC**	**INVSTATE**	**UNDEFINSTR**

DIVBYZERO – признак деления на ноль: 0 = деления на ноль не было, либо обработка данного типа ошибки запрещена, 1 = процессор выполнил инструкцию SDIV или UDIV с делителем равным 0. Если бит равен 1, значение счетчика команд PC,

сохраненное в стеке, указывает на инструкцию, вызвавшую отказ. Разрешить либо запретить обработку деления на ноль можно путем установки в 1 бита DIV_0_TRP регистра CCR.

UNALIGNED – признак доступа к памяти по невыровненному адресу: 0 = не было, либо обработка данного типа ошибки запрещена, 1 = процессор попытался обратиться к памяти по невыровненному адресу. Разрешить либо запретить обработку этой ошибки можно путем установки в 1 бита UNALIGN_TRP регистра CCR. Инструкции LDM, STM, LDRD, и STRD, пытающиеся обратиться по невыровненному адресу, вызывают исключение всегда, вне зависимости от значения бита UNALIGN_TRP.

NOCP – попытка обращения к сопроцессору. Процессор не поддерживает инструкции, требующие наличия сопроцессора. 0 = не было, 1 = была.

INVPC – загрузка неверного значения в счетчик команд PC. 0 = не было, 1 = процессор попытался загрузить в счетчик команд PC неверное значение EXC_RETURN, вследствие неправильного восстановления контекста, либо неверного значения EXC_RETURN. Если бит равен 1, значение счетчика команд PC, сохраненное в стеке, указывает на инструкцию, пытавшуюся загрузить неверное значение в PC.

INVSTATE – неверное состояние: 0 = не было, 1 = процессор попытался выполнить инструкцию, связанную с неверным использованием регистра EPSR. Если бит равен 1, значение счетчика команд PC, сохраненное в стеке, указывает на инструкцию, попытавшуюся некорректно использовать регистр EPSR.

UNDEFINSTR – попытка выполнения неверной инструкции. 0 = не было, 1 = процессор попытался выполнить неверную инструкцию. Если бит равен 1, значение счетчика команд PC, сохраненное в стеке, указывает на инструкцию, вызвавшую отказ. Под неверной понимается инструкция, которую процессор не смог декодировать.

После установки в 1 биты регистра UFSR сохраняют это значение до тех пор, пока не будут принудительно сброшены путем записи в них 1, либо до сброса системы.

12.2.12 **SCB->HFSR**

12.2.12.1 Регистр состояния тяжелого отказа.

Регистр HFSR содержит сведения о причинах вызова обработчика тяжелого отказа.

Особенностью данного регистра является то, что для сброса в 0 его разрядов необходимо записать в них значение 1.

Таблица 12.19 – Регистр состояния тяжелого отказа

Номер	31	30	29…2	1	0
Доступ	R/W	R/W	U	R/W	R/W
Сброс	0	0	0	0	0
	DEBUGEVT	**FORCED**	**-**	**VECTTBL**	**Reserved**

DEBUGEVT – бит зарезервирован для отладки. При записи в регистр данный бит должен быть равен 0, в противном случае поведение процессора непредсказуемо.

FORCED – признак тяжелого отказа, возникшего вследствие эскалации отказа с конфигурируемым уровнем приоритета, который не может быть обработан (запрещен или имеет недостаточно высокий приоритет): 0 = нет, 1 = да.

Если этот бит равен 1, то для определения причины отказа обработчику следует прочитать значения остальных разрядов регистров HFSR.

VECTTBL – признак возникновения отказа шины при попытке доступа к таблице векторов исключений: 0 = не было, 1 = было. Эта ошибка всегда вызывает передачу управления на обработчик тяжелого отказа. Если бит равен 1, значение счетчика команд PC, сохраненное в стеке, указывает на инструкцию, выполнение которой было прервано для обработки исключения.

После установки в 1 биты регистра HFSR сохраняют это значение до тех пор, пока не будут принудительно сброшены путем записи в них 1, либо до сброса системы.

12.2.13 **SCB->MMFAR**

12.2.13.1 Регистр адреса отказа доступа к памяти.

Регистр MMFAR содержит адрес, при обращении по которому возникла ошибка управления памятью.

Таблица 12.20 – Регистр адреса отказа доступа к памяти

Номер	31…0
Доступ	RO
Сброс	0
	ADDRESS

ADDRESS если бит MMARVALID регистра MMFSR равен 1, это поле содержит адрес, при обращении по которому возникла ошибка управления памятью. В случае ошибки доступа по невыровненному адресу поле содержит фактическое значение адреса, вызвавшего отказ.

Учитывая, что одна единственная операция чтения или записи может быть разбита процессором на несколько операций доступа по выровненному адресу, в регистре адреса отказа может находиться любое значение в диапазоне адресов, по которым осуществлялась попытка доступа.

Флаги регистра MMFSR содержат информацию о причине отказа, а также сообщают, является ли значение MMFAR корректным.

12.2.14 **SCB->BFAR**

12.2.14.1 Регистр адреса отказа доступа к шине. Регистр BFAR содержит адрес, при обращении по которому возникла ошибка доступа к шине.

Таблица 12.21 – Регистр адреса отказа доступа к шине

Номер	31…0
Доступ	RO
Сброс	0
	ADDRESS

ADDRESS если бит BFARVALID регистра BFSR равен 1, это поле содержит адрес, при обращении по которому возникла ошибка доступа к шине. В случае ошибки доступа по невыровненному адресу поле содержит значение адреса, запрошенного командой процессора, даже если оно не совпадает с адресом, вызвавшем отказ.

Флаги регистра BFSR содержат информацию о причине отказа, а также сообщают, является ли значение BFAR корректным.

12.3 Рекомендации по программированию блока управления системой

12.3.1 Необходимо убедиться, что программа использует для обращения к регистрам блока управления системой доступ по корректно выровненным адресам. Обращение ко всем регистрам, за исключением CFSR и SHPR1-SHPR3, должно быть выровнено по границе слова. Регистры CFSR и SHPR1-SHPR3 допускают как побайтный доступ, так и доступ по адресам, выровненным по границе слова или полуслова.

Для того, чтобы определить истинный адрес, вызвавший отказ, в обработчике необходимо выполнить следующие действия:

– считать и сохранить значения регистров MMFAR или BFAR;

– проверить значение бита MMARVALID регистра MMFSR, либо бита BFARVALID регистра BFSR. Значения MMFAR или BFAR корректны только в случае, если соответствующие биты равны 1.

Рекомендуется именно такая последовательность операций, так как возникновение исключения с более высоким приоритетом может изменить значения в регистрах MMFAR и BFAR, например, в случае возникновения сбоя в обработчике более высокоприоритетного исключения.

13 Системный таймер SysTick

13.1 Процессор имеет 24-х разрядный системный таймер, SysTick, который считает вниз от загруженного в него значения до нуля; перезагрузка (возврат в начало) значения в регистр LOAD происходит по следующему фронту синхросигнала, затем счёт продолжается по последующему фронту.

Когда процессор остановлен для отладки, таймер не декрементируется.

13.2 Описание регистров системного таймера

Таблица 13.1 – Описание регистров системного таймера SysTick

Адрес	**Название**	**Тип**	**Доступ**	**Значение после сброса**	**Описание**
0xE000E000	SysTick				Системный таймер SYSTICK
0x000	CTRL	RW	привилегированный	0x00000004	SysTick->CTRL
0x004	LOAD	RW	привилегированный	0x00000000	SysTick->LOAD
0x008	VAL	RW	привилегированный	0x00000000	SysTick->VAL
0x00C	CALIB	RO	привилегированный	0x00028B0A	SysTick->CAL

13.2.1 **SysTick->CTRL**

13.2.1.1 Регистр CTRL разрешает основные функции системного таймера.

Назначение бит:

Таблица 13.2 – Регистр контроля и статуса CTRL

Номер	31…17	16	15…3	2	1	0
Доступ						
Сброс						
	-	**COUNTFLAG**	**-**	**CLKSOURCE**	**TICKINT**	**ENABLE**

COUNTFLAG

Возвращает 1, если таймер досчитал до нуля с последнего момента чтения.

CLKSOURCE

Указывает источник синхросигнала:

0 - LSI

1 - HCLK

TCKINT

Разрешает запрос на прерывание от системного таймера:

0 - таймер досчитает до нуля и прерывание не возникнет;

1 - таймер досчитывает до нуля и возникает запрос на прерывание.

Программное обеспечение может использовать бит COUNTFLAG, чтобы определить, досчитал таймер до нуля или нет.

ENABLE

Разрешает работу таймера:

0 - работа таймера запрещена;

1 - работа таймера разрешена.

Когда ENABLE установлен в единицу, таймер загружает значение RELOAD из регистра LOAD и затем начинает декрементироваться. По достижению значения 0 таймер устанавливает бит COUNTFLAG и в зависимости от TCKINT генерирует запрос на прерывание. Затем загружается значение RELOAD и продолжается счёт.

13.2.2 **SysTick->LOAD**

13.2.2.1 Регистр LOAD устанавливает стартовое значение, загружаемое в регистр VAL.

Таблица 13.3 – Регистр перегружаемого значения LOAD

Номер	31…24	23…0
Доступ		
Сброс		
	-	**RELOAD**

RELOAD

Значение, загружаемое в регистр VAL, когда таймер разрешён и когда достигается значение нуля.

Расчёт значения RELOAD

Значение RELOAD может быть любым в диапазоне 0x00000001-0x00FFFFFF. Значение 0 допустимо, но не оказывает эффекта, потому что запрос на прерывание и активизация бита COUNTFLAG происходит только при переходе таймера из состояния 1 в 0.

Расчёт значения RELOAD происходит в соответствии с использованием таймера:

– Для формирования мультикороткого таймера с периодом N процессорных циклов применяется значение RELOAD, равное N-1. Например, если требуется прерывание каждые 100 циклов, то устанавливается значение RELOAD, равное 99;

– Для формирования одиночного прерывания после задержки в N тактов процессора используется значение N. Например, если требуется прерывание после 400 тактов процессора, то устанавливается RELOAD, равное 400.

13.2.3 **SysTick->VAL**

13.2.3.1 Регистр VAL содержит текущее значение системного таймера.

Таблица 13.4 – Регистр текущего значения таймера VAL

Номер	31…24	23…0
Доступ		
Сброс		
	-	**CURRENT**

CURRENT

Чтение возвращает текущее значение системного таймера.

Запись любого значения очищает регистр в ноль, и также очищает бит COUNTFLAG регистра CTRL.

13.2.4 **SysTick->CAL**

13.2.4.1 Регистр CALIB показывает калибровочное значение системного таймера.

Таблица 13.5 – Регистр калибровочного значения таймера CAL

Номер	31	30	29…24	23…0
Доступ				
Сброс				
	NOREF	**SKEW**	**-**	**TENMS**

NOREF

Читается как ноль.

SKEW

Читается как ноль.

TENMS

Читается как 0x00028B0A.

14 Модуль защиты памяти MPU

14.1 MPU делит карту памяти на регионы и определяет положение, размер, разрешение на доступ и атрибуты памяти для каждого из них. Поддерживается:

– независимая установка атрибута для каждого региона;

– наложение (перекрытие) регионов;

– экспортирование атрибутов памяти в систему.

Атрибуты памяти влияют на доступ к памяти в регионе. MPU определяет:

– восемь независимых регионов, 0-7;

– фоновый регион.

Если регионы памяти перекрываются, на доступ к памяти влияют атрибуты региона с большим номером. Например, атрибуты региона 7 получают первенство над атрибутами любых других регионов, перекрывающихся с 7.

Фоновый регион имеет такие же атрибуты доступа к памяти, как и default карта памяти, но доступен только через привилегированные инструкции программы.

Карта памяти Cortex-M4 унифицированная. Это означает, что атрибуты доступа к инструкциям и данным одинаковые.

Если происходит программный запрос в запрещённую область памяти MPU, процессор генерирует ошибку управления памятью. Это вызывает прерывание по ошибке и может вызвать прерывание процессов в переменном окружении OS.

В переменном окружении OS,ядро может обновлять настройки MPU региона динамически, основываясь на выполняемых процессах. Обычно встроенные OS используют MPU для защиты памяти.

Конфигурация MPU регионов основывается на типе памяти.

Таюлица 14.1 показывает возможные атрибуты MPU регионов. Здесь включены такие атрибуты памяти, как shareable и кэшируемость, которые не существенны во многих реализациях микроконтроллеров.

Таблица 14.1 – Атрибуты памяти

Тип памяти	**Атрибут shareable**	**Другие атрибуты**	**Описание**
Строго упорядоченная	-	-	Весь доступ к строго упорядоченной памяти осуществляется под программным управлением. Все строго упорядоченные регионы могут быть общими
Устройство	Общая	-	Общая периферийная память для нескольких процессоров
	Не общая	-	Периферийная память только для одного процессора
Обычная	Общая		Обычная общая память для нескольких процессоров
	Не общая		Обычная память только для одного процессора

14.2 Описание регистров MPU

14.2.1 Применяются следующие MPU регистры для определения регионов и их атрибутов.

Таблица 14.2 – Обзор регистров MPU

Адрес	**Обозначение**	**Тип**	**Доступ**	**Значение после сброса**	**Описание**
0xE000ED90	MPU				Модуль защиты памяти
0x000	TYPE	RO	привилегированный	0x00000800	MPU->TYPE
0x004	CTRL	RW	привилегированный	0x00000000	MPU->CTRL
0x008	RNR	RW	привилегированный	0x00000000	MPU->RNR
0x00C	RBAR	RW	привилегированный	0x00000000	MPU->RBAR
0x010	RASR	RW	привилегированный	0x00000000	MPU->RASR
0x014	RBAR_A1	RW	привилегированный	0x00000000	Обозначение RBAR
0x018	RASR_A1	RW	привилегированный	0x00000000	Обозначение RASR
0x01C	RBAR_A2	RW	привилегированный	0x00000000	Обозначение RBAR
0x020	RASR_A2	RW	привилегированный	0x00000000	Обозначение RASR
0x24	RBAR_A3	RW	привилегированный	0x00000000	Обозначение RBAR
0x28	RASR_A3	RW	привилегированный	0x00000000	Обозначение RASR

14.2.2 **MPU->TYPE**

14.2.2.1 Регистр TYPE показывает, присутствует или нет MPU, и как много регионов поддерживается.

Таблица 14.3 – Регистр TYPE

Номер	31…24	23…16	15…8	7…1	0
Доступ					
Сброс					
	-	**IREGION**	**DREGION**	**-**	**SEPARATE**

IREGION

Указывает количество поддерживаемых MPU регионов инструкций.

Всегда содержит 0x00. Карта памяти MPU унифицированная и описывается полем DREGION.

DREGION

Указывает количество поддерживаемых MPU регионов данных.

0x08 - Восемь MPU регионов.

SEPARATE

Указывает, поддерживается унифицированная или раздельная карта памяти для инструкций и данных:

0 - унифицированная.

14.2.3 **MPU->CTRL**

14.2.3.1 Регистр CTRL:

- разрешает MPU;
- разрешает default карту памяти как фоновый регион;
- разрешает применение MPU, при возникновении аппаратной ошибки, немаскируемое прерывание (NMI), FAULTMASK вызываемый обработчик.

Таблица 14.4 – Регистр CTRL

Номер	31…4	3	2	1	0
Доступ					
Сброс					
	-	**PRIVDEFENA**		**HFNMIENA**	**ENABLE**

PRIVDEFENA

Разрешает привилегированный программный доступ к default карте памяти:

0 - если MPU разрешен, запрещение применяется к default карте памяти. Любой доступ к памяти, не покрываемой разрешённым регионом, вызывает ошибку;

1 - если MPU разрешон, разрешает применение default карты памяти как фонового региона для привилегированного программного доступа.

Когда разрешено, фоновый регион считается как номер региона -1. Любой регион, который определён и разрешен, имеет приоритет выше этой default памяти.

Если MPU запрещён, то процессор игнорирует этот бит.

HFNMIENA

Разрешает операции MPU во время возникновения аппаратной ошибки, NMI, и FAULTMASK обработчик.

Если MPU разрешён:

0 - MPU запрещён во время возникновения аппаратной ошибки, NMI, FAULTMASK обработчик, несмотря на значения бита ENABLE;

1 - MPU разрешён во время возникновения аппаратной ошибки, NMI, FAULTMASK обработчик.

Если MPU запрещён и этот бит устанавливается в единицу, то поведение непредсказуемо.

ENABLE
Разрешает MPU:
0 - MPU запрещён;
1 - MPU разрешён.

Если ENABLE и PRIVDEFENA одновременно установлены в единицу:

Любой неадресованый доступ привилегированным программным обеспечением к разрешённому региону, ведёт себя как определено default картой памяти. Любой неадресованный доступ непривилегированным программным обеспечением к разрешённому региону вызывает ошибку управления памятью.

XN и строго упорядоченные правила всегда применяются к управляющему системному пространству несмотря на значение бита ENABLE.

Когда ENABLE установлен в единицу, по крайней мере, один регион карты памяти должен быть разрешён для системных функций, за исключением PRIVDEFENA установлен в единицу.

Если PRIVDEFENA установлен в единицу и нет разрешённых регионов, тогда только привилегированное программное обеспечение может исполняться.

Когда ENABLE установлен в ноль, система использует default карту памяти. Это аналогично памяти с атрибутами, как если бы MPU не применялся. К default карте памяти доступ осуществляется как с помощью привилегированного, так и непривилегированного программного обеспечения.

Когда MPU разрешён, доступ к системному пространству управления и таблице векторов всегда разрешён. К другим областям доступ базируется на регионах и состоянии бита PRIVDEFENA.

За исключением случая HFNMIENA установленного в 1, MPU не разрешает процессору выполнять обработчики прерываний с приоритетом -1 или -2. Эти приоритеты допустимы только когда обрабатывается прерывание аппаратной ошибки или NMI, или когда FAULTMASK разрешён. Установка бита HFNMIENA в единицу разрешает действовать этим двум приоритетам.

14.2.4 **MPU->RNR**

14.2.4.1 Регистр RNR выбирает, на какой регион памяти ссылаются регистры RBAR и RASR.

Таблица 14.5 – Регистр номера региона RNR

Номер	31…8	7…0
Доступ		
Сброс		
	-	**REGION**

REGION
Указывает MPU регион, на который ссылаются регистры RBAR и RASR.

MPU поддерживает 8 регионов памяти, поэтому разрешённое значение для этого поля от 0 до 7.

Обычно вы записываете требуемое значение номера региона в этот регистр перед обращением в RBAR и RASR. Однако вы можете изменить номер региона записью в RBAR с установленным в единицу битом VALID. Эта запись обновляет значение поля REGION.

14.2.5 **MPU->RBAR**

14.2.5.1 Регистр RBAR определяет базовый адрес MPU региона, выбранного RNR, вы можете изменить значение RNR. Запись RBAR с битом VALID установленным в единицу изменяет текущий номер региона и обновляет RNR.

Таблица 14.6 – Регистр базового адреса RBAR

Номер	31…N	N-1…6	5	4	3…0
Доступ					
Сброс					
	ADDR	**-**	**VALID**		**REGION**

ADDR

Поле базового адреса региона. Значение N зависит от размера региона.

VALID

Бит верности номера региона MPU:

Запись:

0 - RNR не изменяется, и процессор:

- обновляет базовый адрес для региона, определённого в RNR;
- игнорирует значение поля REGION.

1 - процессор:

- обновляет значение RNR на значение из поля REGION;
- обновляет базовый адрес региона, определённого в поле REGION.

Всегда читается как ноль.

REGION

Поле MPU региона:

- поведение при записи описано выше (см. описание бита VALID);
- при чтении возвращает текущий номер региона, который определён в регистре RNR.

Поле ADDR

Поле ADDR это [31:N] бит регистра RBAR. Размер региона определяется полем SIZE в регистре RASR, как

N = Log2 (Размер региона в байтах),

Если размер региона сконфигурирован равным 4 ГБ, в RASR, то значение поля ADDR неверно. В этом случае, регион занимает всю карту памяти, и базовый адрес его равен 0x00000000.

Базовый адрес выравнивается под размер региона. Например, 64КБ регион должен быть кратно 64КБ, например, 0x00010000 или 0x00020000.

14.2.6 MPU->RASR

14.2.6.1 Регистр RASR определяет размер и атрибуты памяти MPU региона, выбранного RNR, а также разрешает регион и любые подрегионы.

RASR доступен в режиме слова или полуслова:

- старшее значащее полуслово содержит атрибуты региона;
- младшее значащее полуслово содержит размер региона и биты разрешения региона и подрегионов.

Таблица 14.7 – Назначение бит регистра RASR

Номер	31	30	29	28	27	26	24	23	22	21	19	18	17	16	15	8	7	6	5	1	0
Доступ																					
Сброс																					
	-		**XN**	-		**AP**		-		**TEX**		**S**	**C**	**B**	**SRD**		-		**SIZE**		**ENABLE**

XN

Бит запрещения доступа инструкций:

0 - выборка инструкций разрешена;

1 - выборка инструкций запрещена.

AP

Поле разрешения доступа, см.. Таблица 76 – Кодирование привилегий доступа в поле AP.

TEX, C, B

Атрибуты доступа к памяти, см. Таблица 74 – Кодирование бит разрешения доступа.

S

Бит общего доступа, см. Таблица 73 – Пример значений поля SIZE.

SRD

Бит запрещения подрегиона. Для каждого бита в этом поле:

0 - соответствующий подрегион разрешон;

1 - соответствующий подрегион запрещён.

Регион размером 128 байт и менее не поддерживает подрегионы. Когда записываются атрибуты для такого региона, записывайте поле SRD равным 0x00.

SIZE

Определяет размер MPU региона. Минимальное разрешённое значение 3(b00010).

ENABLE

Бит разрешения региона.

Для более подробной информации о разрешении доступа.

Значения поля SIZE

Поле SIZE определяет размер памяти MPU региона выбранного регистром RNR следующим образом:

(Region size in bytes)= $2^{(SIZE+1)}$

Наименьший разрешенный размер региона 32 байт, соответствует значению SIZE, равному 4. В таблице 14.8 представлены примеры значений SIZE, соответствующие размеру региона и значению N регистра RBAR.

Таблица 14.8 – Пример значений поля SIZE

Значение SIZE	**Размер региона**	**Значение N**	**Комментарий**
b00100 (4)	32 байт	5	Минимальный разрешенный размер
b01001 (9)	1 кбайт	10	-
b10011 (19)	1 Мбайт	20	-
b11101 (29)	1 Гбайт	30	-
b11111 (31)	4 Гбайт	b01100	Максимальный разрешенный размер

14.2.7 Атрибуты разрешения доступа MPU

14.2.7.1 Раздел описывает атрибуты разрешения доступа. Биты разрешения доступа TEX, C, B, S, AP и XN регистра RASR контролируют доступ к соответствующему региону памяти. Если происходит доступ к области памяти без разрешения доступа, то MPU генерирует ошибку доступа.

Таблица 14.9 – Кодирование бит разрешения доступа TEX, C, B, S

TEX	C	B	S	Тип памяти	Возможность общего доступа	Другие атрибуты
b000	0	0	x[1)]	Строго упорядоченная	Общий доступ	
		1	x[1)]	Устройство	Общий доступ	
	1	0	0	Обычная	Не общий доступ	Внешний и внутренний кэш, синхронное обновление памяти. Запись без кэширования
			1		Общий доступ	
		1	0	Обычная	Не общий доступ	Внешний и внутренний кэш, отложенное обновление памяти. Запись без кэширования
			1		Общий доступ	
b001	0	0	0	Обычная	Не общий доступ	
			1		Общий доступ	
		1	x[1)]	Зарезервировано		-
	1	0	x[1)]	Реализация определяется атрибутами		-
		1	0	Обычная	Не общий доступ	Внешний и внутренний кэш, отложенное обновление памяти. Запись и чтение пакетные
			1		Общий доступ	
b010	0	0	x[1)]	Устройство	Не общий доступ	Индивидуальное устройство
		1	x[1)]	Зарезервировано		-
	1	x[1)]	x[1)]	Зарезервировано		-
b1BB	A	A	0	Обычное	Не общий доступ	
			1		Общий доступ	

[1)] MPU игнорирует значение этих бит.

Таблица 14.10 поясняет кодирование режима кэша атрибутом TEX в диапазоне значений атрибута от 4 до 7.

Таблица 14.10 – Кодирование режима кэша атрибутом TEX

Значение AA или BB при TEX=1xx	Соответствующий режим кэша
00	Некэшируемая
01	Отложенное обновление, запись и чтение пакетные
10	Синхронное обновление, запись без кэширования
11	Отложенное обновление, запись без кэширования

Таблица 14.11 поясняет кодирование бит AP, определяющих разрешение на доступ для привилегированного и непривилегированного программного обеспечения (ПО).

Таблица 14.11 – Кодирование привилегий доступа в поле AP

AP[2:0]	Привилегированный доступ	Непривилегированный	Описание
000	нет доступа	нет доступа	Любой доступ приводит к ошибке доступа
001	RW	нет доступа	Доступ только для привилегированного ПО
010	RW	RO	Запись непривилегированным ПО приводит к ошибке
011	RW	RW	Полный доступ
100	непредсказуемо	непредсказуемо	Зарезервировано
101	RO	нет доступа	Чтение только привилегированным ПО
110	RO	RO	Только чтение и припривилегированным и непривилегированным ПО
111	RO	RO	Только чтение и припривилегированным и непривилегированным ПО

14.2.8 **Несоответствие MP**

14.2.8.1 Когда происходит нарушение разрешения доступа MPU, процессор генерирует ошибку управления памятью, см. раздел “Прерывания и исключения”. Регистр MMFSR указывает причину ошибки.

14.2.9 **Обновление MPU региона**

14.2.9.1 Атрибуты для MPU региона обновляют через регистры RNR, RBAR и RASR. Можно программировать каждый регистр независимо или использовать для программирования возможность множественной записи всех этих регистров. Можно использовать обозначения RBAR и RASR, чтобы запрограммировать до 4 регионов одновременно, используя инструкцию STM.

14.2.9.2 Обновление MPU региона через отдельные регистры.

Простой код для одного региона:

```
; R1 = номер региона
; R2 = размер/разрешение
; R3 = атрибуты
; R4 = адрес
LDR R0,=MPU_RNR   ; 0xE000ED98, регистр номера региона MPU
STR R1, [R0, #0x0]     ; номер региона
STR R4, [R0, #0x4]     ; базовый адрес региона
STRH R2, [R0, #0x8]   ; размер региона и разрешение
STRH R3, [R0, #0xA]   ; атрибуты региона
```

Запрещение региона перед записью новых настроек в MPU, если этот регион перед этим был разрешён. Например:

```
; R1 = номер региона
; R2 = размер/разрешение
; R3 = атрибуты
; R4 = адрес
LDR R0,=MPU_RNR ; 0xE000ED98, регистр номера региона MPU
STR R1, [R0, #0x0] ; номер региона
BIC R2, R2, #1 ; запрещение
STRH R2, [R0, #0x8] ; размер региона и разрешение
STR R4, [R0, #0x4] ; базовый адрес региона
STRH R3, [R0, #0xA] ; атрибуты региона
ORR R2, #1 ; разрешение
STRH R2, [R0, #0x8] ; размер региона и разрешение.
```

Программное обеспечение должно применить barrier инструкции:

– если перед установкой MPU будет невыполненная пересылка в память, такая как буферная запись, то это может повлиять на изменение настроек MPU;

– после установки MPU, если это включает пересылку в память, должны использоваться новые настройки MPU.

Однако не требуются barrier инструкции памяти, если процесс установки MPU начинается с помощью входа в обработчик прерывания, или сопровождается возвращением из прерывания, потому что вход и выход из прерывания сопровождается механизмом barrier для памяти.

Программному обеспечению не требуются barrier инструкции памяти во время установки MPU, потому что этот доступ осуществляется через PPB, который строго упорядоченный регион памяти.

Например, если вы хотите, чтобы все изменения доступа к памяти имели место непосредственно после программной последовательности, используйте инструкции DSR и ISB.

Инструкции DSB требуются после изменения настроек MPU, в конце переключения контекста.

Инструкции ISB требуются, если код, который программирует MPU регион или регионы вызывается с использованием инструкций перехода (branch) или вызова подпрограммы (call).

Если программная последовательность вызывается инструкцией выхода из прерывания (return), или прерыванием, то ISB не требуется.

14.2.9.3 Обновление MPU региона через множественную запись регистров.

Можно программировать напрямую, используя запись множества регистров, в зависимости от того, как распределена информация.

```
; R1 = номер региона
; R2 = адрес
; R3 = размер, атрибуты
LDR R0, =MPU_RNR ; 0xE000ED98, регистр номера региона MPU
STR R1, [R0, #0x0] ; номер региона
STR R2, [R0, #0x4] ; базовый адрес региона
STR R3, [R0, #0x8] ; атрибут региона, размер и разрешение.
```

Оптимизация при использовании STM инструкции:

```
; R1 = номер региона
; R2 = адрес
; R3 = размер, атрибуты
LDR R0, =MPU_RNR ; 0xE000ED98, регистр номера региона MPU
STM R0, {R1-R3} ; номер региона, адрес, атрибут, размер и разрешение.
```

Можно использовать два слова для предварительной упаковки информации. Это значит, что RBAR содержит требуемый номер региона и имеет бит VALID, установленный в единицу. Это применимо, если данные упакованы статически, например, в начальном загрузчике:

```
; R1 = адрес и номер региона;
; R2 = размер и атрибуты;
LDR R0, =MPU_RBAR ; 0xE000ED9C, регистр базового адреса MPU
STR R1, [R0, #0x0] ; базовый адрес и номер региона,
 ; совмещённые с битом VALID, установленным в 1
STR R2, [R0, #0x4] ; атрибут региона, размер и разрешение.
```

Оптимизация при использовании STM инструкции:

```
; R1 = адрес и номер региона
; R2 = размер и атрибуты
LDR R0,=MPU_RBAR ; 0xE000ED9C, регистр базового адреса MPU
STM R0, {R1-R2} ; базовый адрес региона, номер региона и бит VALID,
; и атрибут региона, размер и разрешение.
```

14.2.9.4 Подрегионы

Регионы величиной в 256 байт или более делятся на восемь равных подрегионов. Установите соответствующий бит в поле SRD регистра RASR для запрещения подрегиона. Младший значащий бит SRD контролирует первый подрегион, и старший значащий бит контролирует последний подрегион. Запрещение подрегиона означает, что другой регион перекрывает запрещённую область. Если другой разрешённый регион не перекрывает запрещённый регион, то MPU вырабатывает ошибку.

Регионы размером 32, 64 и 128 не поддерживают подрегионы, с этими регионами необходимо установить поле SRD равным 0x00, иначе поведение MPU непредсказуемо.

14.2.9.5 Пример применения SRD

Два региона с одинаковым базовым адресом перекрываются. Регион размером 128 КВ и регион размером 512 КВ. Убедитесь, что атрибуты для региона один установлены для первых 128 КВ, установите SRD поле для региона два в значение b00000011 для запрещения первых двух подрегионов, как показано на рисунке ниже.

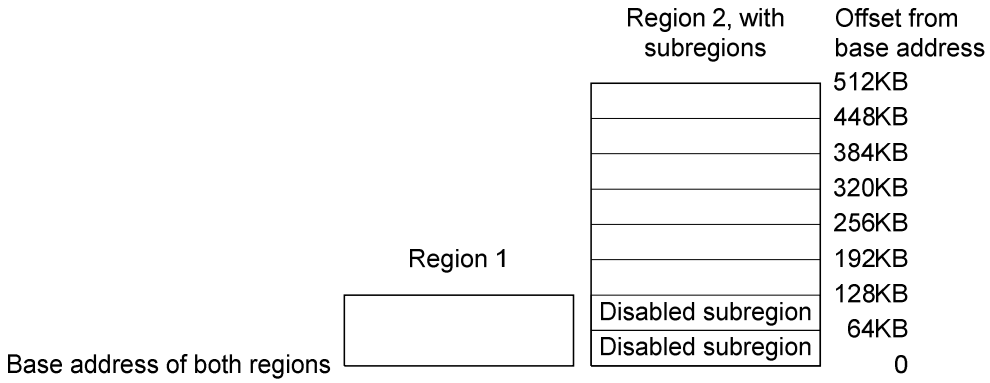

Рисунок 14.1 – Применение SRD

14.3 Советы и особенности применения MPU

14.3.1 Во избежание непредвиденных ситуаций, запретите прерывания перед обновлением атрибутов региона, к которому может осуществляется доступ в обработчике прерываний.

Убедитесь, что программное обеспечение использует корректный доступ, соответствующий размеру регистров MPU:

– за исключением RASR, необходимо использовать доступ по словам;

– для RASR может использоваться доступ по байтам, полусловам или словам.

Процессор не поддерживает невыровненный доступ к регистрам MPU.

Если MPU перенастраивается, то запретите неиспользуемые регионы для предотвращения любых предыдущих настроек регионов от их влияния на новые настройки.

14.3.2 Конфигурация MPU для микроконтроллера

Обычно, микроконтроллерные системы имеют только один процессор и не имеют кэша. В таких системах MPU программируется следующим образом:

Таблица 14.12 – Атрибуты регионов памяти для микроконтроллера

Регион памяти	**TEX**	**C**	**B**	**S**	**Типы памяти и атрибут**
Флеш-память	b000	1	0	0	Обычная память, не общий доступ, сквозная запись
Внутренняя SRAM	b000	1	0	1	Обычная память, общий доступ, сквозная запись
Внешняя SRAM	b000	1	1	1	Обычная память, общий доступ, обратная запись, выделенная запись
Периферия	b000	0	1	1	Память устройства, общий доступ

В большинстве микроконтроллерных приложениях, установка атрибутов общего доступа и кэширования не влияет на поведение системы. Однако применение этих настроек для MPU регионов может сделать код приложений более переносимым. Это имеет большую важность в обычных ситуациях. В специальных системах, таких как многопроцессорные или с отдельным DMA устройством, атрибуты общего доступа очень важны. В этих случаях обращайтесь к рекомендациям производителей устройств памяти.

15 Порты ввода-вывода GPIO

15.1 В состав микроконтроллера входят 5 портов ввода-вывода GPIO. Порты 16-разрядные с отдельным управлением для каждого вывода. Порты GPIO имеют следующие особенности:

– программируемая генерация прерываний;

– поддержка битового маскирования на основе адреса;

– раздельная установка/сброс контрольных регистров.

15.2 Генерация прерываний

15.2.1 В портах ввода-вывода реализована программируемая генерация прерываний. Для работы с прерываниями реализованы три контрольных регистра. Каждый регистр имеет раздельные адреса для установки и сброса бит. Используя эти три регистра, любой бит порта может быть сконфигурирован для генерации прерываний.

Таблица 15.1 – Генерация прерываний

Interrupt enable[n]	Interrupt polarity[n]	Interrupt type[n]	Признак прерывания
0	-	-	Отключено
1	0	0	Низкий уровень
1	0	1	Спад
1	1	0	Высокий уровень
1	1	1	Фронт

Возникшее прерывание вызывает установку соответствующего бита регистра INTSTATUS. Этот бит может быть сброшен в обработчике прерывания записью 1 в соответствующий бит регистра INTCLEAR, располагающегося по тому же адресу, что и INTSTATUS. Признаки прерываний всех линий порта объединены по схеме ИЛИ в один сигнал – комбинированное прерывание порта. Комбинированное прерывание каждого порта подключено к контроллеру NVIC.

15.3 Доступ по маске

15.3.1 Возможность доступа по маске позволяет выполнить запись/чтение отдельных бит порта за одну операцию. Это исключает необходимость использования операции вида чтение-модификация-запись. При доступе по маске 16 разрядов порта разделены на две половины – младший и старший байты. Адресное пространство битовых масок определено в виде двух массивов по 256 слов каждый.

Например, для того, чтобы установить биты [1:0] и сбросить биты [7:6] за одну операцию, необходимо выполнить запись в регистр доступа по маске, соответствующий младшему байту. Битам [1:0] и [7:6] соответствует битовая маска 0xC3. Таким образом, необходимая операция имеет вид MASKLOWBYTE[0xC3] = 0x03, см. рисунок 15.1.

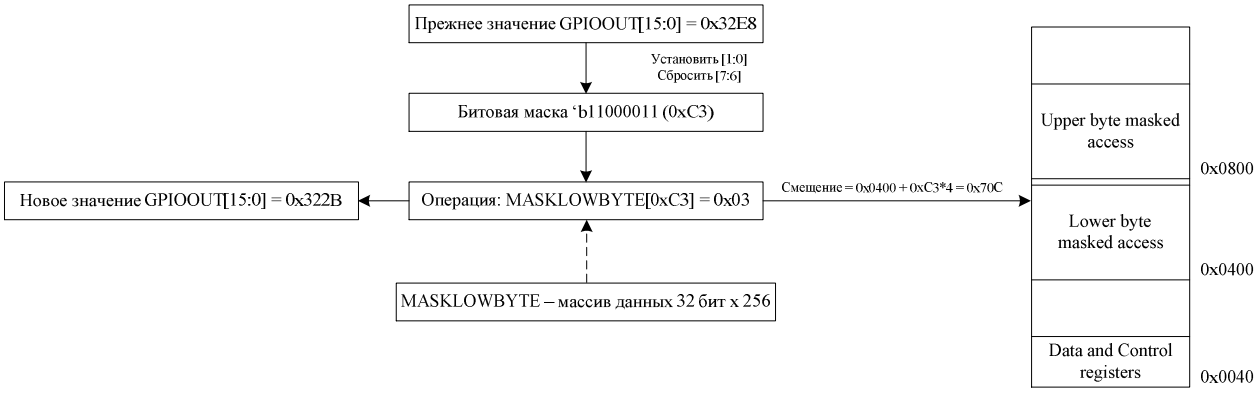

Рисунок 15.1 – Доступ по маске (младший байт)

Аналогично, для того, чтобы изменить некоторые разряды в старшем байте порта ввода-вывода, необходимо выполнить запись в массив MASKHIGHBYTE как показано на рисунке 15.2.

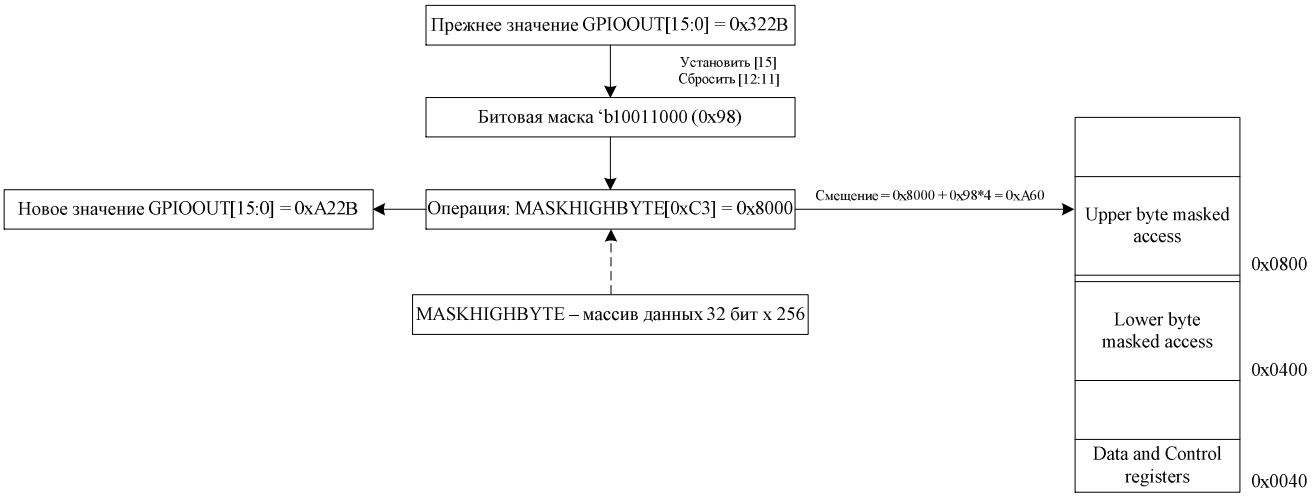

Рисунок 15.2 – Доступ по маске (старший байт)

15.4 Описание регистров портов ввода-вывода

Таблица 15.2 – Описание регистров портов ввода-вывода

Базовый адрес	**Название**	**Описание**
0x4002_1000	GPIOA	Порт A
0x4002_2000	GPIOB	Порт B
0x4002_3000	GPIOC	Порт C
0x4002_4000	GPIOD	Порт D
0x4002_5000	GPIOE	Порт E
Смещение		
0x0000	DATA[15:0]	Регистр GPIOx->DATA данных порта
0x0004	DATA_OUT[15:0]	Регистр GPIOx->DATAOUT выходных данных порта
0x0008-x000C	Зарезервировано	
0x0010	OUTENSET[15:0]	Регистр GPIOx->OUTENSET установки бит разрешения выхода
0x0014	OUTENCLR[15:0]	Регистр GPIOx-> OUTENCLR очистки бит разрешения выхода
0x0020	INTENSET[15:0]	Регистр GPIOx->INTENSET установки бит разрешения прерываний
0x0024	INTENCLR[15:0]	Регистр GPIOx->INTENCLR очистки бит разрешения прерываний
0x0028	INTTYPESET[15:0]	Регистр GPIOx->INTTYPESET установки бит типа прерываний
0x002C	INTTYPECLR[15:0]	Регистр GPIOx->INTTYPECLR очистки бит типа прерываний
0x0030	INTPOLSET[15:0]	Регистр GPIOx->INTPOLSET установки бит полярности прерываний
0x0034	INTPOLCLR[15:0]	Регистр GPIOx->INTPOLCLR очистки бит полярности прерываний
0x0038	INTSTATUS[15:0]	Регистр GPIOx->INTSTATUS статуса прерываний
0x0400-0x07FC	MASKLOWBYTE[15:0]	Регистр GPIOx->MASKLOWBYTE маскирования младшего байта порта
0x0800-0x0BFC	MASKHIGHBYTE[15:0]	Регистр GPIOx->MASKHIGHBYTE маскирования старшего байта порта
0x0C00-0x0FCF	Зарезервировано	

15.4.1 **GPIOx->DATA**

Таблица 15.3 – Регистр GPIOx->DATA

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	-
	-	**DATA**

Таблица 15.4 – Описание бит регистра GPIOx->DATA

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	DATA	При **чтении** – данные, сэмплируемые на внешних выводах порта При **записи** – данные помещаются в выходной буфер порта

15.4.2 **GPIOx->DATAOUT**

Таблица 15.5 – Регистр GPIOx->DATAOUT

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**DATAOUT**

Таблица 15.6 – Описание бит регистра GPIOx->DATAOUT

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	DATAOUT	При **чтении** – данные, хранящиеся в выходном буфере порта При **записи** – данные помещаются в выходной буфер порта

15.4.3 **GPIOx->OUTENSET**

Таблица 15.7 – Регистр GPIOx->OUTENSET

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**OUTENSET**

Таблица 15.8 – Описание бит регистра GPIOx->OUTENSET

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	OUTENSET	Направление передачи данных на выводах порта. При **чтении:** **0 –** порт работает как вход **1 –** порт работает как выход При **записи**: **0** – не влияет на направление передачи данных **1** – установить направление порта как выход

15.4.4 **GPIOx->OUTENCLR**

Таблица 15.9 – Регистр GPIOx->OUTENCLR

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**OUTENCLR**

Таблица 15.10 – Описание бит регистра GPIOx->OUTENCLR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	OUTENCLR	Направление передачи данных на выводах порта. При **чтении:** **0 –** порт работает как вход **1 –** порт работает как выход При **записи**: **0** – не влияет на направление передачи данных **1** – установить направление порта как вход

15.4.5 **GPIOx->INTENSET**

Таблица 15.11 – Регистр GPIOx->INTENSET

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**INTENSET**

Таблица 15.12 – Описание бит регистра GPIOx->INTENSET

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	INTENSET	Разрешение прерываний При **чтении:** **0 –** прерывания от порта запрещены **1 –** прерывания от порта разрешены При **записи**: **0** – не влияет на разрешение прерываний **1** – разрешить прерывания от порта

15.4.6 **GPIOx->INTENCLR**

Таблица 15.13 – Регистр GPIOx->INTENCLR

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**INTENCLR**

Таблица 15.14 – Описание бит регистра GPIOx->INTENCLR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	INTENCLR	Запрет прерываний При **чтении:** **0 –** прерывания от порта запрещены **1 –** прерывания от порта разрешены При **записи**: **0** – не влияет на запрет прерываний **1** – запретить прерывания от порта

15.4.7 **GPIOx->INTTYPESET**

Таблица 15.15 – Регистр GPIOx->INTTYPESET

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**INTTYPESET**

Таблица 15.16 – Описание бит регистра GPIOx->INTTYPESET

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	INTTYPESET	Установка признака прерывания При **чтении:** **0 –** прерывание генерируется при НИЗКОМ или ВЫСОКОМ уровне на внешних выводах порта **1 –** прерывание генерируется при положительном или отрицательном перепаде на внешних выводах порта При **записи**: **0** – не влияет на признак прерывания **1** – установить в качестве признака прерывания положительный или отрицательный перепад на внешних выводах порта

15.4.8 **GPIOx->INTTYPECLR**

Таблица 15.17 – Регистр GPIOx->INTTYPECLR

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**INTTYPECLR**

Таблица 15.18 – Описание бит регистра GPIOx->INTTYPECLR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	-	Зарезервировано
15…0	INTTYPECLR	Установка признака прерывания При **чтении:** **0 –** прерывание генерируется при НИЗКОМ или ВЫСОКОМ уровне на внешних выводах порта **1 –** прерывание генерируется при положительном или отрицательном перепаде на внешних выводах порта При **записи**: **0** – не влияет на признак прерывания **1** – установить в качестве признака прерывания наличие НИЗКОГО или ВЫСОКОГО уровня на внешних выводах порта

15.4.9 **GPIOx->INTPOLSET**

Таблица 15.19 – Регистр GPIOx->INTPOLSET

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**INTPOLSET**

Таблица 15.20 – Описание бит регистра GPIOx->INTPOLSET

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…16	-	Зарезервировано
15…0	INTPOLSET	Установка полярности прерывания При **чтении:** **0 –** прерывание генерируется при НИЗКОМ уровне или при отрицательном перепаде на внешних выводах порта **1 –** прерывание генерируется при ВЫСОКОМ уровне или при положительном перепаде на внешних выводах порта При **записи**: **0** – не влияет на полярность прерывания **1** – установить в качестве полярности прерывания ВЫСОКИЙ уровень или положительный перепад на внешних выводах порта

15.4.10 **GPIOx->INTPOLCLR**

Таблица 15.21 – Регистр GPIOx->INTPOLCLR

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**INTPOLCLR**

Таблица 15.22 – Описание бит регистра GPIOx->INTPOLCLR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…16	-	Зарезервировано
15…0	INTPOLCLR	Установка полярности прерывания При **чтении:** **0 –** прерывание генерируется при НИЗКОМ уровне или при отрицательном перепаде на внешних выводах порта **1 –** прерывание генерируется при ВЫСОКОМ уровне или при положительном перепаде на внешних выводах порта При **записи**: **0** – не влияет на полярность прерывания **1** – установить в качестве полярности прерывания НИЗКИЙ уровень или отрицательный перепад на внешних выводах порта

15.4.11 **GPIOx->INTSTATUS**

Таблица 15.23 – Регистр GPIOx->INTSTATUS

Номер	31:16	15:0
Доступ	U	R/W
Сброс	0	0
	-	**INTSTATUS**

Таблица 15.24 – Описание бит регистра GPIOx->INTSTATUS

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…16	-	Зарезервировано
15…0	INTSTATUS	Статус прерывания При **чтении –** статус прерываний от порта **1 –** прерывание генерируется при ВЫСОКОМ уровне или при положительном перепаде на внешних выводах порта При **записи**: **0** – не влияет на статус прерывания **1** – очистка запроса прерывания

15.4.12 **GPIOx->MASKLOWBYTE**

Таблица 15.25 – Регистр GPIOx->MASKLOWBYTE

Номер	31:8	7:0
Доступ	U	R/W
Сброс	0	0
	-	**MASKLOWBYTE**

Таблица 15.26 – Описание бит регистра GPIOx->MASKLOWBYTE

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…8	-	Зарезервировано
7…0	MASKLOWBYTE	Младший байт регистра доступа по маске

15.4.13 **GPIOx->MASKHIGHBYTE**

Таблица 15.27 – Регистр GPIOx->MASKHIGHBYTE

Номер	31:8	7:0
Доступ	U	R/W
Сброс	0	0
	-	**MASKHIGHBYTE**

Таблица 15.28 – Описание бит регистра GPIOx->MASKHIGHBYTE

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…8	-	Зарезервировано
7…0	MASKLOWBYTE	Старший байт регистра доступа по маске

16 Внешняя системная шина

16.1 Внешняя системная шина позволяет работать с внешними микросхемами памяти и периферийными устройствами. В таблице 16.1 приведены области адресного пространства микроконтроллера, предназначенные для работы с внешней системной шиной.

Таблица 16.1 – Адресные диапазоны внешней системной шины

Адресный диапазон	**Размер**	**Описание**
0x0000_0000 – 0x0000_0FFF	4 Кбайт	Область памяти секции CODE отображаемая на внешнюю системную шину в режиме REMAP=1. Формируется сигнал выборки чипа CS0.
0x1000_0000 – 0x1FFF_FFFF	256 Мбайт	Области памяти секции CODE, отображаемая на внешнюю системную шину с доступом через I code и D code шины. В режиме микропроцессора из этой области начинает выполняться программа. Формируется сигнал выборки чипа CS0.
0x3000_0000 – 0x3FFF_FFFF	256 Мбайт	Область памяти секции DATA, отображаемая на внешнюю системную шину с доступом через S Bus. Формируется сигнал выборки чипа CS1.
0x6000_0000 – 0x7FFF_FFFF	512 Мбайт	Область памяти секции EXTERNAL BUS, отображаемая на внешнюю системную шину с доступом через S Bus. Формируется сигнал выборки чипа CS2.
0x8000_0000 – 0x9FFF_FFFF	512Мбайт	Область памяти секции EXTERNAL BUS, отображаемая на внешнюю системную шину с доступом через S Bus. Формируется сигнал выборки чипа CS3.
0xA000_0000 – 0xBFFF_FFFF	512Мбайт	Область памяти секции EXTERNAL BUS, отображаемая на внешнюю системную шину с доступом через S Bus. Формируется сигнал выборки чипа CS4.
0xC000_0000 – 0xDFFF_FFFF	512 Мбайт	Область памяти секции EXTERNAL BUS, отображаемая на внешнюю системную шину с доступом через S Bus. Формируется сигнал выборки чипа CS5.

Для каждого из 6 регионов можно задавать собственные настройки длительности транзакции на шине.

16.2 Работа с внешними статическими ОЗУ, ПЗУ и периферийными устройствами

16.2.1 Для работы контроллера внешней системной шины с внешними микросхемами статического ОЗУ, ПЗУ или внешними периферийными устройствами необходимо задать режим работы через регистр EXT_BUS_CONTROL. В зависимости от скорости работы ядра микроконтроллера и внешних устройств необходимо задать времена транзакции на внешней системной шине через биты регистра WAIT_STATE для соответствующего региона. После этого все обращения в область памяти, отображаемой на внешнюю системную шину, будут транслироваться на выводы внешней системной шины ADDR, DATA, сигналы управления OE, WE, BE[3:0] и сигнал синхронизации CLOCK. Полярность сигнала на выходе CLOCK задаётся битом CPOL.

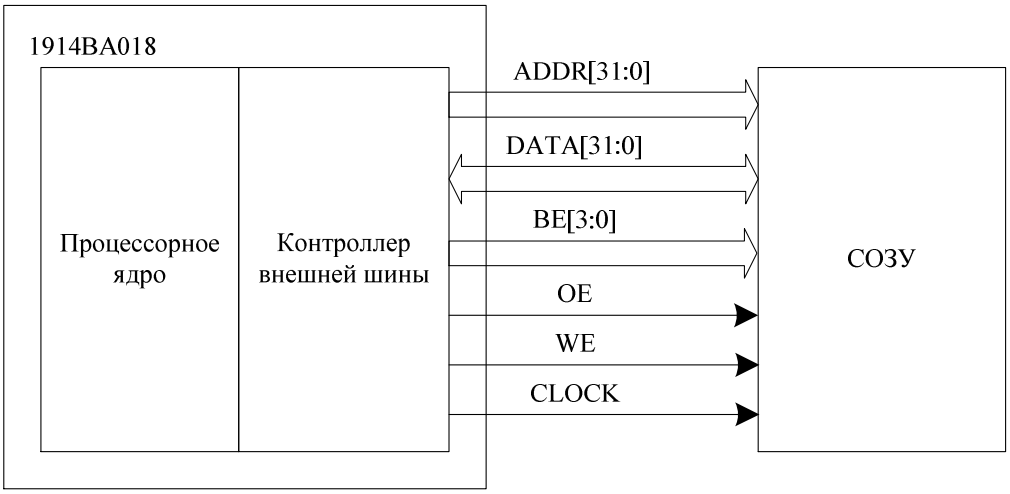

Рисунок 16.1 – Обмен по внешней системной шине

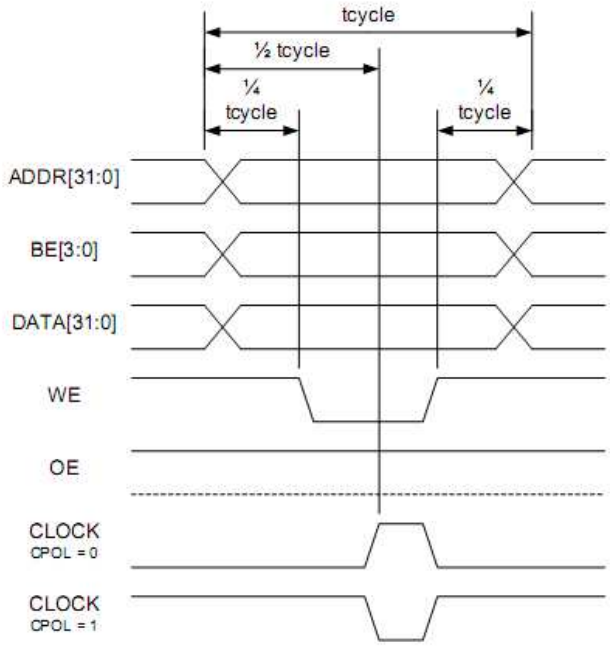

Рисунок 16.2 – Диаграмма записи

Время цикла записи tcycle задаётся битами WAIT_STATE[3:0]. Активный уровень сигналов WE, OE, BE[3:0] низкий. Если сигнал CLOCK не требуется, он может не использоваться.

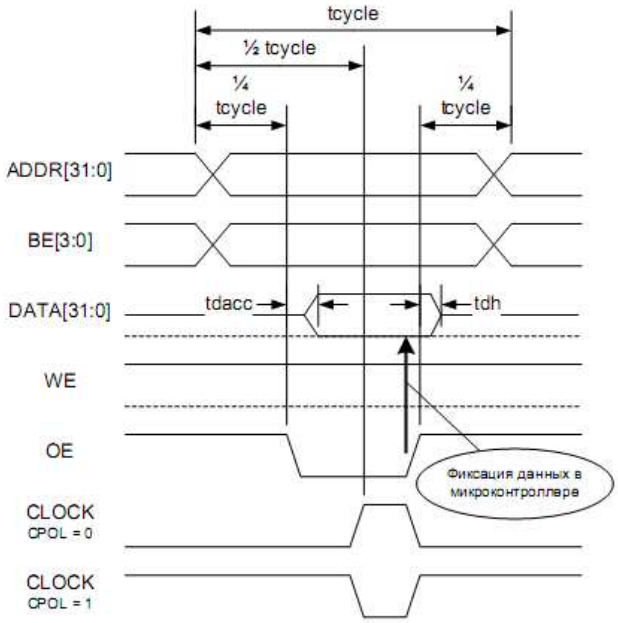

Рисунок 16.3 – Диаграмма чтения

При чтении по внешней системной шине необходимо выбрать такую длительность времени tcycle, чтобы удовлетворить время доступа к памяти.

Таблица 16.2 – Длительность фаз обращения в тактах процессора для соответствующего региона

WAIT_STATE	**Предустановка адреса и данных перед сигналом WE или OE**	**Длительность WE или OE**	**Удержание адреса и данных после сигнала WE или OE**
0	1	1	0
1	1	1	1
2	1	1	1
3	1	2	1
4	2	2	1
5	2	3	1
6	2	3	2
7	2	4	2
8	3	4	2
9	3	5	2
10	3	5	3
11	3	6	3
12	4	6	3
13	4	7	3
14	4	7	4
15	4	8	4

16.3 Работа с внешней NAND Flash – памятью

16.3.1 Для работы контроллера внешней системной шины с внешними NAND Flash микросхемами необходимо задать режим работы через регистр EXT_BUS_CONTROL. Бит NAND разрешает работу с внешними NAND Flash микросхемами. В зависимости от скорости работы ядра микроконтроллера и внешних устройств необходимо задать времена выполнения различных этапов работы NAND Flash – памяти через регистр NAND_CYCLES. После этого обращения в область памяти, отображаемой на внешнюю системную шину будут перекодироваться в командные, адресные и обмена данными циклы обращения с NAND Flash через выводы внешней системной шины DATA[7:0], ALE, CLE, BUSY.

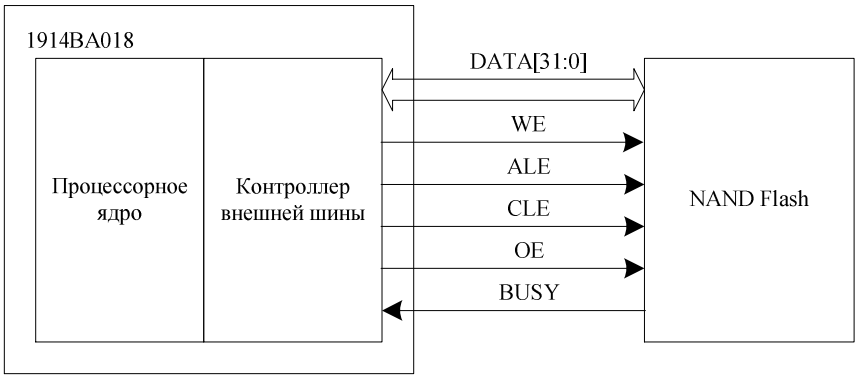

Рисунок 16.4 – Подключение внешней NAND Flash

При работе с NAND Flash – памятью тип выполняемой операции кодируется адресом обращения, а данные и адрес передаются данными при записи и чтении памяти. Формат кодирования адреса обращения представлен в Таблице.

Таблица 16.3 – Формат кодирования адреса обращения

Адрес обращения	**Фаза команды**	**Фаза адреса**	**Фаза данных**
ADDR[31:24]	Не имеет значения, но должно попадать в адресные диапазоны внешней системной шины: 0x10…0x1F		
ADDR[23]	Hold CS: 1 – удержание линии в активном состоянии 0 – линия CS переводится в неактивное состояние по окончании передачи		
ADDR[22]	–	Hold ALE: 1 – удержание линии ALE в активном состоянии 0 – линия ALE переводится в неактивное состояние по окончании передачи	–
ADDR[21]	Выбор фазы команды (CMD)		
ADDR[20]	Выбор фазы адреса (ADDR)		
ADDR[19]	Выбор фазы данных (DATA)		
ADDR[18]	–		
ADDR[17:10]	Код команды		
ADDR[9]	1 – учитывать Trr (en_trr)		
ADDR[8]	1 – учитывать Talea (en_talea)		
ADDR[7]	1 – учитывать Twhr (en_twhr)		
ADDR[6:0]	Не имеет значения		

16.4 Описание регистров блока контроллера внешней системной шины

Таблица 16.4 – Описание регистров блока контроллера внешней системной шины

Базовый адрес	**Название**	**Описание**
0x4001_E000	EBC	Контроллер внешней системной шины
Смещение		
0x00	CONTROL	Регистр EBC->CONTROL управления внешней системной шиной
0x04	NAND_CYCLES	Регистр EBC->NAND_CYCLES управления работой с NAND Flash
0x08	LOW8	Регистр EBC->LOW8

16.4.1 **EBC->CONTROL**

Таблица 16.5 – Регистр CONTROL

Номер	31…28	27…24	23…20	19…16	15…12	11…8
Доступ	R/W	R/W	R/W	R/W	R/W	R/W
Сброс						
	wait_state5	**wait_state4**	**wait_state3**	**wait_state2**	**wait_state1**	**wait_state0**

Номер	7	6…4	3	2	1	0
Доступ	R/W	U	R/W	R/W	R/W	U
Сброс		0	0	0	0	0
	busy	**-**	**cpol**	**nand**	**ram**	**-**

Таблица 16.6 – Описаиние бит регистра CONTROL

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…28	wait_state5	Длительность фаз обращений для региона памяти, соответствующего сигналу выборки CS5
27…24	wait_state4	Длительность фаз обращений для региона памяти, соответствующего сигналу выборки CS4
23…20	wait_state3	Длительность фаз обращений для региона памяти, соответствующего сигналу выборки CS3
19…16	wait_state2	Длительность фаз обращений для региона памяти, соответствующего сигналу выборки CS2
15…12	wait_state1	Длительность фаз обращений для региона памяти, соответствующего сигналу выборки CS1
11…8	wait_state0	Длительность фаз обращений для региона памяти, соответствующего сигналу выборки CS0
7	busy	Сигнал занятости NAND Flash памяти 1 – Операция завершена 0 – Операция не завершена
6…4	-	Резерв
3	cpol	Бит задания полярности сигнала CLOCK 1 – Отрицательная полярность 0 – Положительная полярность
2	nand	Бит глобального разрешения памяти NAND Flash 1 – Память NAND Flash выбрана 0 – Память NAND Flash не выбрана
1	ram/rom	Бит глобального разрешения памяти RAM 1 – Память RAM выбрана 0 – Память RAM не выбрана
0	-	Резерв

16.4.2 **EBC->NAND_CYCLES**

Таблица 16.7 – Регистр NAND_CYCLES

Номер	31…28	27…24	23…20	19…16	15…12	11…8	7…4	3…0
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс		0	0	0	0	0	0	0
	-	**t_rr**	**t_alea**	**t_whr**	**t_wp**	**t_rea**	**t_wc**	**t_rc**

Таблица 16.8 – Описание бит регистра NAND_CYCLES

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…28		Зарезервировано
27…24	t_rr[3:0]	Время от снятия BUSY до операции чтения: 0000 – 0 HCLK циклов 0001 – 1 HCLK цикл … 1111 – 15 HCLK циклов Типовое значение для памяти NAND Flash составляет 20 нс
23…20	t_alea[3:0]	Время доступа к регистрам ID. Аналогично t_rr. Типовое значение для памяти NAND Flash составляет 100 нс
19…16	t_whr[3:0]	Время доступа к регистру статуса. Аналогично t_rr. Типовое значение для памяти NAND Flash составляет 60 нс
15…12	t_wp[3:0]	Время доступа по записи. Аналогично t_rr. Типовое значение для памяти NAND Flash составляет 25 нс
11…8	t_rea[3:0]	Время доступа по чтению. Аналогично t_rr. Типовое значение для памяти NAND Flash составляет 35 нс
7…4	t_wc[3:0]	Время цикла записи. Аналогично t_rr. Типовое значение для памяти NAND Flash составляет 60 нс
3…0	t_rc[3:0]	Время цикла чтения. Аналогично t_rr. Типовое значение для памяти NAND Flash составляет 60 нс

16.4.3 **EBC->LOW8**

Таблица 16.9 – Регистр LOW8

Номер	31…6	5	4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W
Сброс		0	0	0	0	0	0
	-	**low8_5**	**low8_4**	**low8_3**	**low8_2**	**low8_1**	**low8_0**

Таблица 16.10 – Описание бит регистра LOW8

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…6	-	Резерв
5	low8_5	Переключение между 8/32 битным режимами для региона памяти, соответствующего сигналу выборки CS5 0 – 32-битный режим работы 1 – 8-битный режим работы
4	low8_4	Переключение между 8/32 битным режимами для региона памяти, соответствующего сигналу выборки CS4 0 – 32-битный режим работы 1 – 8-битный режим работы
3	low8_3	Переключение между 8/32 битным режимами для региона памяти, соответствующего сигналу выборки CS3 0 – 32-битный режим работы 1 – 8-битный режим работы
2	low8_2	Переключение между 8/32 битным режимами для региона памяти, соответствующего сигналу выборки CS2 0 – 32-битный режим работы 1 – 8-битный режим работы
1	low8_1	Переключение между 8/32 битным режимами для региона памяти, соответствующего сигналу выборки CS1 0 – 32-битный режим работы 1 – 8-битный режим работы
0	low8_0	Переключение между 8/32 битным режимами для региона памяти, соответствующего сигналу выборки CS0 0 – 32-битный режим работы 1 – 8-битный режим работы

17 Таймеры общего назначения

17.1 В состав микроконтроллера входят три таймера общего назначения. Таймеры имеют разрядность 32 бита и представляют собой счетчик обратного счета со следующими особенностями:

– возможность генерации сигнала преырвания по достижении счетчиком значения 0. Запрос на прерывание удерживается до тех пор, пока он не будет снят в обработчике записью в регистр INTCLEAR;

– в качестве разрешения работы таймера может выступать перепад на выводе TMRx_EXTIN;

– если счетчик таймера достиг значения 0 и в тоже самое время ПО очистило предыдущий статус прерывания, статус прерывания будет вновь установлен в 1;

– в режиме внешнего тактирования частота тактового сигнала, поступающего на вход TMRx_EXTIN, должна быть как минимум вдвое меньше системной частоты, т.к. пройдя через логику двойного защёлкивания внешний тактовый сигнал сэмплируется внутренней логикой детектирования перепадов.

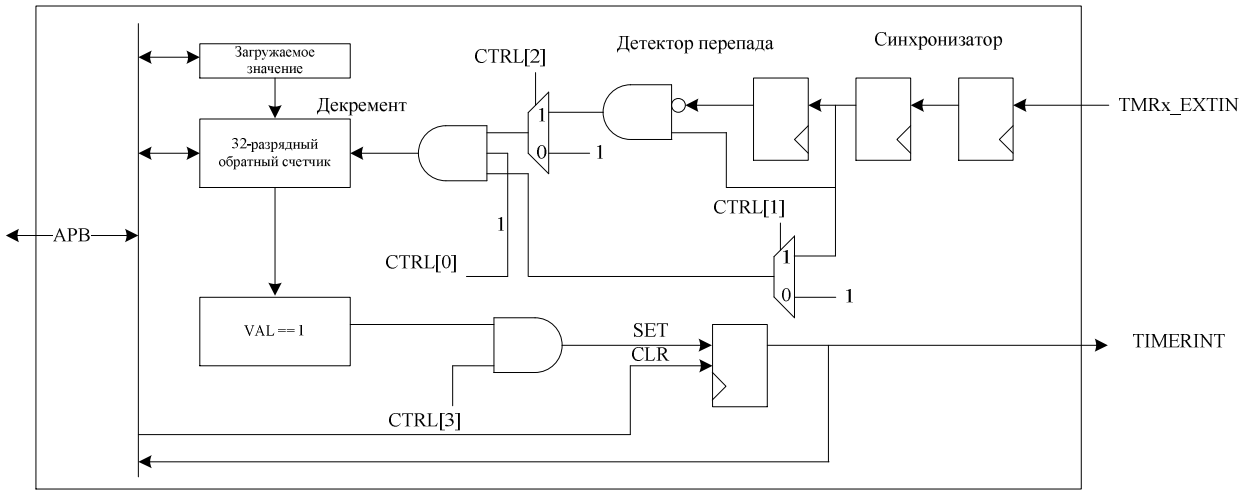

Рисунок 17.1 – Структурная схема таймера

17.2 Описание регистров блока таймера

Таблица 17.1 – Базовые адреса и смещения регистров таймера

Базовый адрес	**Название**	**Описание**
0x4000_9000	TIMER1	Контроллер TIMER1
0x4000_A000	TIMER2	Контроллер TIMER2
0x4000_B000	TIMER3	Контроллер TIMER3
Смещение		
0x00	CTRL	Регистр контроля TIMERx->CTRL
0x04	VALUE	Регистр TIMERx->VALUE
0x08	RELOAD	Регистр TIMERx->RELOAD
0x0C	INTSTATUS	Регистр прерываний TIMERx->INTSTATUS

17.2.1 TIMERx->CTRL

Таблица 17.2 – Регистр TIMERx->CTRL

Номер	31…4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0
	-	**INT_EN**	**EXT_CLK**	**EXT_EN**	**ENABLE**

Таблица 17.3 – Описание бит регистра TIMERx->CTRL

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…4	0	Зарезервировано
3	INT_EN	Разрешение прерывания от таймера **0** – запретить генерацию прерывания **1** – разрешить генерацию прерывания
2	EXT_CLK	Тактирования таймера от внешнего вывода TMRx_EXTIN **0** – запретить тактирования от TMRx_EXTIN **1** – разрешить тактирование от TMRx_EXTIN
1	EXT_EN	Разрешение работы таймера с внешнего вывода TMRx_EXTIN **0** – не использовать вход TMRx_EXTIN как разрешающий вход **1** – использовать перепад на входе TMRx_EXTIN в качестве сигнала разрешения для работы таймера
0	ENABLE	Разрешение работы блока таймера **0** – таймер отключен **1** – таймер включен

17.2.2 TIMERx->VALUE

Таблица 17.4 – Регистр TIMERx->VALUE

Номер	31…0
Доступ	R/W
Сброс	0
	VALUE

Таблица 17.5 – Описание бит регистра TIMERx->VALUE

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	VALUE	Текущее значение счетчика таймера

17.2.3 TIMERx->RELOAD

Таблица 17.6 – Регистр TIMERx->RELOAD

Номер	31…0
Доступ	R/W
Сброс	0
	RELOAD

Таблица 17.7 – Описание бит регистра TIMERx->RELOAD

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	RELOAD	Перезагружаемое значение. Запись в этот регистр меняет текущее значение счетчика таймера.

17.2.4 TIMERx->INTSTATUS

Таблица 17.8 – Регистр TIMERx->INTSTATUS

Номер	31…1	0
Доступ	U	R/W
Сброс	0	0
	-	**INTSTATUS**

Таблица 17.9 – Описание бит регистра TIMERx->INTSTATUS

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…1	-	Зарезервировано
0	INTSTATUS	Статус прерывания. При **чтении:** **0** – нет прерывания от таймера **1** – есть прерывание от таймера При **записи:** **0** – не влияет на статус прерывания **1** – очистка запроса прерывания

18 Контроллер интерфейса SPI

18.1 Модуль контроллера выполняет функции интерфейса последовательной синхронной связи в режимах ведущего и ведомого устройства и обеспечивает обмен данными с подключенными ведомым или ведущим периферийным устройством в соответствии с протоколом интерфейса SPI фирмы Motorola.

Основные характеристики модуля SPI:

- может функционировать как в ведущем, так и в ведомом режимах;
- программное управление скоростью обмена;
- состоит из независимых буферов приёма и передачи с организацией доступа типа FIFO (First In First Out – первый вошёл, первый вышел);
- программируемая длительность информационного кадра от 4 до 16 бит;
- независимое маскирование прерываний от буфера FIFO передатчика, буфера FIFO приемника, а также по переполнению буфера приёмника.

18.2 Структура контроллера SPI

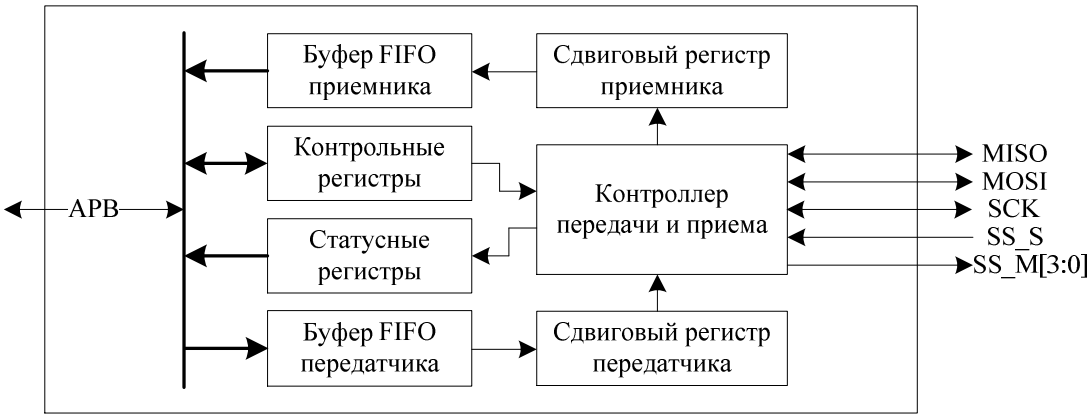

Рисунок 18.1 – Структурная схема контроллера SPI

18.3 Формат информационного кадра

18.3.1 Протокол SPI позволяет выбрать состояния и фазы сигнала SCK в режиме ожидания путем задания бит PL и PH регистра управления CR1.

Выбор полярности тактового сигнала – бит PL. Если бит PL равен 0, то в режиме ожидания линия SCK переводится в низний логический уровень. В противном случае при отсутствии обмена данными линия SCK переводится в высокий логический уровень.

Выбор фазы тактового сигнала – бит PH. Значение бита PH определяет фронт тактового сигнала, по которому осуществляется выборка данных и изменение состояние на выходе линии. В случае, если бит PH установлен в 0, регистрация данных приемником осуществляется после первого обнаружения фронта тактового сигнала, в противном случае – после второго.

18.3.2 Синхронный обмен при PH=0, PL=0

18.3.2.1 На рисунке 18.2 показаны временные диаграммы последовательного синхронного обмена в режиме PH=0, PL=0.

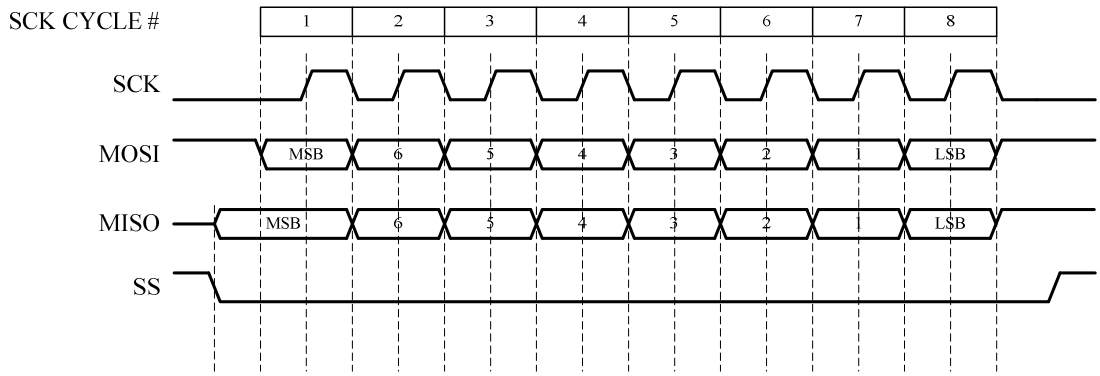

Рисунок 18.2 – Формат синхронного обмена (PH=0, PL=0)

18.3.3 Синхронный обмен при PH=1, PL=0

18.3.3.1 На рисунке 18.3 показаны временные диаграммы последовательного синхронного обмена в режиме PH=1, PL=0.

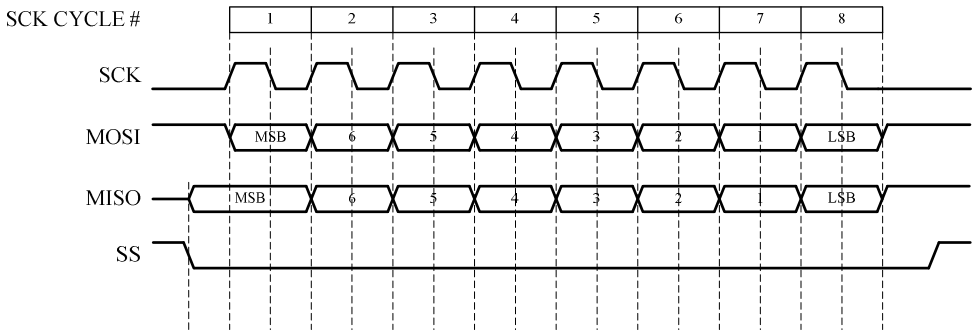

Рисунок 18.3 – Формат синхронного обмена (PH=1, PL=0)

18.3.4 Синхронный обмен при PH=0, PL=1

18.3.4.1 На рисунке 18.4 показаны временные диаграммы последовательного синхронного обмена в режиме PH=0, PL=1.

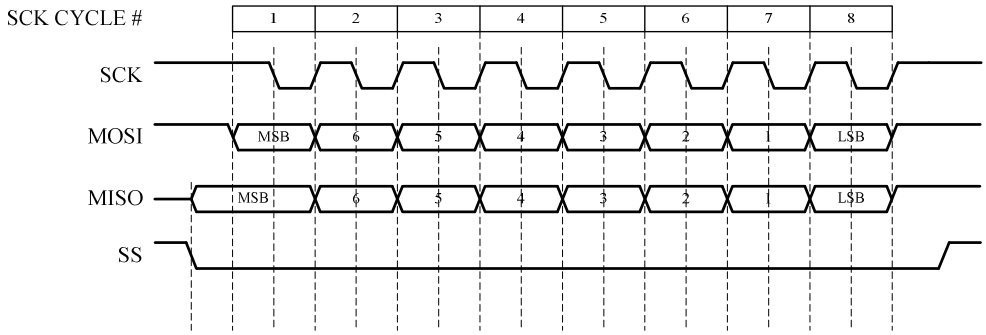

Рисунок 18.4 – Формат синхронного обмена (PH=0, PL=1)

18.3.5 Синхронный обмен при PH=1, PL=1

18.3.5.1 На рисунке 18.5 показаны временные диаграммы последовательного синхронного обмена в режиме PH=1, PL=1.

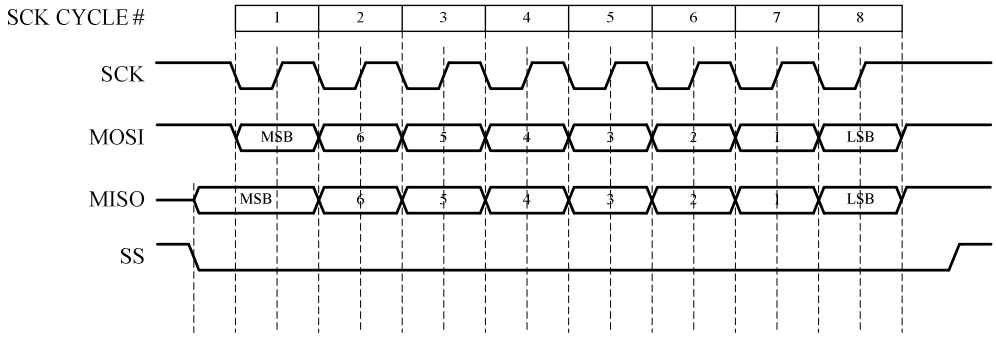

Рисунок 18.5 – Формат синхронного обмена (PH=1, PL=1)

18.4 Буферы приема и передачи

18.4.1 Для хранения передаваемых и принятых данных в контроллере SPI имеются два 16-разрядных буфера, организованных по принципу FIFO. Каждый буфер может хванить до восьми слов данных. Буфер для передаваемых данных со стороны процессора доступен только для записи, а буфер принятых данных – только для чтения.

18.4.2 FIFO-буфер передатчика

18.4.2.1 FIFO-буфер передатчика имеет ширину 16 бит и глубину 8 слов. Буфер построен по схеме «первый вошел, первый вышел». Данные от микроконтроллера сохраняются в буфере до тех пор, пока не будут считаны передатчиком.

Данные для передачи записываются в буфер через регистр SPIx->DR. Состояние буфера можно контролировать с помощью статусных бит регистра состояния SPIx->SR.

18.4.3 FIFO-буфер приемника

18.4.3.1 FIFO-буфер приемника имеет ширину 16 бит и глубину 8 слов. Буфер построен по схеме «первый вошел, первый вышел». Данные, принятые от периферийного устройства, сохраняются в буфере до тех пор, пока не будут прочитаны со стороны процессора.

Извлечь данные из буфера возможно чтением регистра SPIx->DR. Состояние буфера можно контролировать с помощью статусных бит регистра состояния SPIx->SR

18.5 Функционирование

18.5.1 Формирование тактового сигнала обмена данными

18.5.1.1 В контроллере SPI предусмотрена возможность программной настройки скорости передачи/приема и параметров тактового сигнала SPIx_SCK. Для обеспечения необходимой скорости обмена необходимо настроить значения полей SCK_HIGH и SCK_LOW регистра SPIx->CPSR в тактах системной частоты.

18.5.2 Прием и передача данных

18.5.2.1 Размер передаваемого кадра данных может быть от 4 до 16 бит, что задается полем DL регистра SPIx->CR1. Если выбран размер кадра менее 16 бит, данные выравниваются по правой границе, а неиспользуемые биты игнорируются. В контроллере SPI реализованы 4 линии выбора ведомого. Битовые поля SS_3, SS_2, SS_1, SS_0 регистра SPIx->CR1 позволяют в режиме ведущего выбрать активную в ходе обмена линию выбора ведомого либо их комбинацию. В зависимости от состояни битового поля SS регистра SPIx->CR1 в режиме непрерывной передачи данных линия выбора ведомого SPIx-_SSn_M может либо постоянно находиться в низком логическом уровне либо на ней могут формируются импульсы высокого уровня между передачами каждого из слов данных. По окончании приема последнего бита блока данных соответствующие линии выбора ведомого SPIx_SSn_M переводятся в состояние, соответствующее режиму ожидания.

В режиме ведущего блок передатчика последовательно считывает данные из FIFO-буфера передатчика и преобразует их из параллельной формы в последовательную, после чего поток последовательных данных вместе с тактовым сигналом SPIx_SCK передается по линии SPIx_MOSI.

Блок приемника выполняет преобразование данных, поступающих синхронно с линии SPIx_MISO, из последовательной в параллельную форму, после чего загружает их в FIFO-буфер приемника.

В режиме ведомого тактовый сигнал обмена данными формируется одним из подключенных к модулю периферийных устройств и поступает по линии SPIx_SCK.

Передатчик, тактируемый этим внешним сигналом, считывает данные из FIFO-буфера, преобразует их из параллельной формы в последовательную и выдает поток последовательных данных в линию SPIx_MISO.

Блок приемника выполняет преобразование данных, поступающих с линии SPIx_MOSI синхронно с сигналом SPIx_SCK, из последовательной в параллельную форму, после чего загружеает их в FIFO-буфер приемника.

18.5.3 Прерывания

18.5.3.1 Контроллер SPI генериурет независимые прерывания с активным высоким уровнем. Все эти прерывания объединяются по схеме ИЛИ в комбинированное прерывание. Комбинированное прерывание подключено к контроллеру прерываний NVIC. Любое независимое прерывание, в том числе и комбинированное может быть маскировано путем задания соответствующих настроек в регистре SPIx->IMSC.

Сброс прерывания осуществляется путем записи 1 в соответствующий бит регистра SPIx->INTCLEAR.

18.6 Описание регистров контроллера SPI

Таблица 18.1 – Базовые адреса и смещения регистров контроллеров SPI

Базовый адрес	**Название**	**Описание**
0x4000_7000	SPI1	Контроллер SPI1
0x4000_8000	SPI2	Контроллер SPI2
Смещение		
0x00	DR	Буфер FIFO приемника (чтение) Буфер FIFO передатчика (запись) SPIx->DR
0x04	SR	Регистр SPIx->SR состояния
0x08	CR1	Регистр SPIx->CR1 управления
0x0C	CR2	Регистр SPIx->CR2 управления
0x10	CPSR	Регистр SPIx->CPSR делителя тактовой частоты
0x14	IMSC	Регистр SPIx->IMSC установки и сброса маски прерывания
0x18	INTSTATUS	Регистр SPIx->ISC состояния и сброса прерывания
0x18	INTCLEAR	

18.6.1 SPIx->DR

Таблица 18.2 – Регистр SPIx->DR

Номер	31…16	15…0
Доступ	U	R/W
Сброс	0	0
	-	**DATA**

Таблица 18.3 – Описание бит регистра SPIx->DR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	0	Зарезервировано
15…0	DATA	Принимаемые данные **(чтение)** Передаваемые данные **(запись)** При чтении выполняется доступ к последней несчитанной ячейке буфера FIFO приемника. При записи очередное слово помещается в буфер FIFO передатчика. При длине информационного слова менее 16 бит перед записью необходимо обеспечить выравнивание данных по правой границе. Неиспользуемые биты игнорируются передатчиком. Принятые информационные слова выравниваются по правой границе автоматически.

18.6.2 SPIx->SR

Таблица 18.4 – Регистр SPIx->SR

Номер	31…5	4	3	2	1	0
Доступ	U	R	R	R	R	R
Сброс	0	0	0	1	0	1
	-	**BUSY**	**RFF**	**RNE**	**TNF**	**TNE**

Таблица 18.5 – Описание бит регистра SPIx->SR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…5	0	Зарезервировано
4	BUSY	Флаг активности модуля: **1** – модуль активен **0** – модуль не активен
3	RFF	Буфер приемника заполнен: **1** – заполнен **0** – не заполнен
2	RNE	Буфер приемника не пуст: **1** – пуст **0** – не пуст
1	TNF	Буфер передатчика не заполнен: **1** – заполнен **0** – не заполнен
0	TNE	Буфер передатчика пуст: **1** – пуст **0** – не пуст

18.6.3 SPIx->CR1

Таблица 18.6 – Регистр SPIx->CR1

Номер	31…11	10	9	8	7	6	5	4	3…0
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс	0	1	1	1	1	0	0	0	0
	-	**SS_3**	**SS_2**	**SS_1**	**SS_0**	**PH**	**PL**	**SS**	**DL**

Таблица 18.7 – Описание бит регистра SPIx->CR1

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…11	0	Зарезервировано
10	SS_3	Активность сигнала выбора ведомого SPIx_SS3_M в режиме ведущего: 1 – сигнал SPIx_SS3_M активен; 0 – сигнал SPIx_SS3_M не активен.
9	SS_2	Активность сигнала выбора ведомого SPIx_SS2_M: 1 – сигнал SPIx_SS2_M активен; 0 – сигнал SPIx_SS2_M не активен.
8	SS_1	Активность сигнала выбора ведомого SPIx_SS1_M: 1 – сигнал SPIx_SS1_M активен; 0 – сигнал SPIx_SS1_M не активен.
7	SS_0	Активность сигнала выбора ведомого SPIx_SS0_M: 1 – сигнал SPIx_SS0_M активен; 0 – сигнал SPIx_SS0_M не активен.
6	PH	Фаза сигнала SCK
5	PL	Полярность сигнала SCK
4	SS	Переключение сигнала SS между посылками слов: **1** – включено **0** – отключено
3…0	DL	Размер слова данных: 0000 – резерв 0001 – резерв 0010 – резерв 0011 – 4 бита 0100 – 5 бит 0101 – 6 бит 0110 – 7 бит 0111 – 8 бит 1000 – 9 бит 1001 – 10 бит 1010 – 11 бит 1011 – 12 бит 1100 – 13 бит 1101 – 14 бит 1110 – 15 бит 1111 – 16 бит

18.6.4 SPIx->CR2

Таблица 18.8 – Регистр SPIx->CR2

Номер	31…2	1	0
Доступ	U	R/W	R/W
Сброс	0	0	0
	-	**MS**	**EN**

Таблица 18.9 – Описание бит регистра SPIx->CR2

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…2	0	Зарезервировано
1	MS	Выбор режима ведущего/ведомого **1** – ведущий **0** – ведомый
0	EN	Разрешение работы приемопередатчика **1** – работа разрешена **0** – работа запрещена

18.6.5 SPIx->CPSR

Таблица 18.10 – Регистр SPIx->CPSR

Номер	31…16	15…8	7…0
Доступ	U	R/W	R/W
Сброс	0	0x4	0x4
	-	**SCK_HIGH**	**SCK_LOW**

Таблица 18.11 – Описание бит регистра SPIx->CPSR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…16	0	Зарезервировано
15…8	SCK_HIGH	Длительность высокого уровня SCK в тактах системной частоты
7…0	SCK_LOW	Длительность низкого уровня SCK в тактах системной частоты

18.6.6 SPIx->IMSC

Таблица 18.12 – Регистр SPIx->IMSC

Номер	31…4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W
Сброс	0	1	1	1	1
	-	**COMBIM**	**TXIM**	**RXIM**	**RORIM**

Таблица 18.13 – Описание бит регистра SPIx->IMSC

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…4	0	Зарезервировано
3	COMBIM	Маска комбинационного прерывания **1** – прерывание маскировано **0** – прерывание не маскировано
2	TXIM	Маска прерывания по событию «буфер передатчика пуст»: **1** – прерывание маскировано **0** – прерывание не маскировано
1	RXIM	Маска прерывания по событию «буфер при`емника не пуст»: **1** – прерывание маскировано **0** – прерывание не маскировано
0	RORIM	Маска прерывания по событию «буфер приемника полон» **1** – прерывание маскировано **0** – прерывание не маскировано

18.6.7 SPIx->INTSTATUS

Таблица 18.14 – Регистр SPIx->INTSTATUS

Номер	31…4	3	2	1	0
Доступ	U	R	R	R	R
Сброс	0	1	1	0	0
	-	**COMBIS**	**TXIS**	**RXIS**	**RORIS**

Таблица 18.15 – Описание бит регистра SPIx->INTSTATUS

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…4	0	Зарезервировано
3	COMBIS	Состояние комбинационного прерывания
2	TXIS	Состояние прерывания по событию «буфер передатчика пуст»
1	RXIS	Состояние прерывания по событию «буфер приемника не пуст»
0	RORIS	Состояние прерывания по событию «буфер приемника полон»

18.6.8 SPIx->INTCLEAR

Таблица 18.16 – Регистр SPIx->INTCLEAR

Номер	31…4	3	2	1	0
Доступ	U	W	W	W	W
Сброс	0	1	1	0	0
	-	**COMBIS**	**TXIS**	**RXIS**	**RORIS**

Таблица 18.17 – Описание бит регистра SPIx->INTCLEAR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…4	0	Зарезервировано
3	COMBIS	1 – сброс комбинационного прерывания 0 – не влияет на признак комбинационного прерывания
2	TXIS	1 – сброс прерывания по событию «буфер передатчика пуст» 0 – не влияет на признак прерывания по событию «буфер передатчика пуст»
1	RXIS	1 – сброс прерывания по событию «буфер приемника не пуст» 0 – не влияет на признак прерывания по событию «буфер приемника не пуст»
0	RORIS	1 – сброс прерывания по событию «буфер приемника полон» 0 – не влияет на признак прерывания по событию «буфер приемника полон»

19 Контроллер интерфейса UART

19.1 В состав микроконтроллера входят 3 универсальных асинхронных приемопередатчика: UART1, UART2, UART3.

Приемопередатчики UART2 и UART3 отличаются от приемопередатчика UART1 тем, что их в их состав входят независимыве буферы типа FIFO – для приемника (32x8) и передатчика (32x8). Буферы приемопередатчика UART1 имеют размерность 1 байт.

19.2 Основные характеристики контроллера UART

- UART2 и UART3 содержат независимые буферы приема (32x8) и передачи (32x8) типа FIFO (First In First Out), что позволяет снизить интенсивность прерываний центрального процессора;
- программное управление скоростью обмена;
- поддержка стандартных элементов асинхронного протокола связи – стартового и стопового бита, которые добавляются перед передачей и удаляются после приема;
- независимое маскирование прерываний от буфера FIFO передатчика, буфера FIFO приемника;
- данные длиной 8 бит;
- контроль четности отсутствует;
- количество стоп-бит: 1.

19.3 Структура контроллера UART

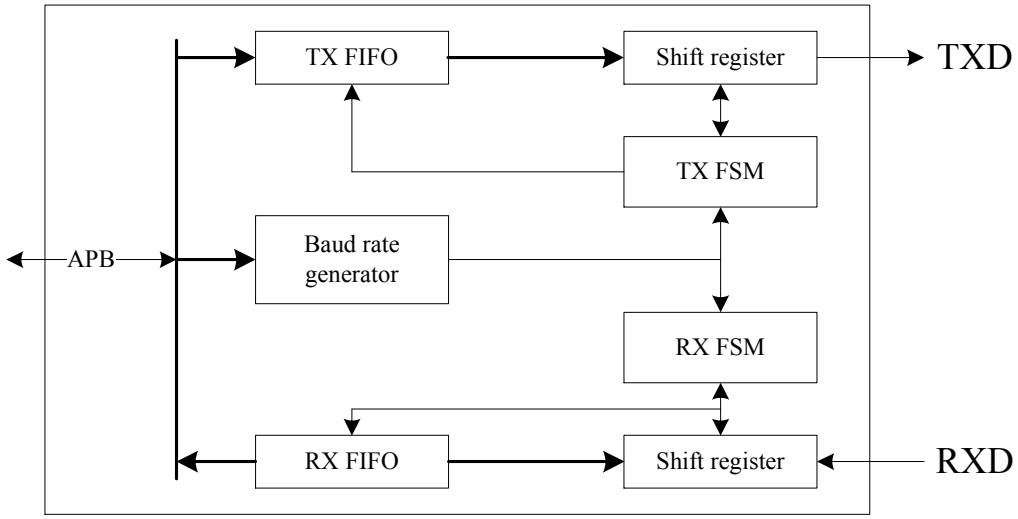

Рисунок 19.1 – Структурная схема приемопередатчика UART

19.4 Функционирование

19.4.1 Скорость обмена

19.4.1.1 Перед включением приемопередатчика UART необходимо настроить скорость обмена, выполнив запись в регистр делителя UARTx->BAUDDIV. Например если системная частота равна 12 МГц, а желаемая скорость обмена 9600 бод, то в регистр UARTx->BAUDDIV должно быть записано значение 12000000/9600 = 1250. Контроллер UART также поддерживает высокоскоростной режим. Для его активации предназначено поле HS_MODE регистра UARTx->CTRL.

19.4.2 Буфер FIFO передатчика

19.4.2.1 Буфер передатчика контроллеров UART2, UART3 имеет ширину 8 бит, глубину 32 слова, схему организации доступа типа «первый вошел, первый вышел». Данные от центрального процессора, записанные через шину APB, сохраняются в буфере до тех пор, пока не будут считаны логической схемой передачи данных.

19.4.3 Буфер FIFO приемника

19.4.3.1 Буфер приемника имеет ширину 8 бит, глубину 32 слова, схему организации доступа типа «первый вошел, первый вышел». Принятые от периферийного устройства данные сохраняются логикой приема данных в нем до тех пор, пока не будут считаны центральным процессором через шину APB.

19.4.4 Блок передатчика

19.4.4.1 Логические схемы передатчика осуществляют преобразование данных, считанных из буфера передатчика, из параллельной в последовательную форму. Управляющая логика выдает последовательный поток бит в порядке: стартовый бит, биты данных, начиная с младшего значащего разряда, стоповый бит.

19.4.5 Блок приемника

19.4.5.1 Логические схемы приемника преобразуют данные, полученные от периферийного устройства, из последовательной в параллельную форму после обнаружения корректного стартового импульса. Кроме того, производят проверки переполнения буфера.

19.4.6 Формирование прерываний

19.4.6.1 Контроллер UART1 генерирует 4, а контроллеры UART2, UART3 – 5 независимых прерываний с активным высоким уровнем. Каждый из независимых сигналов запроса на прерывание может быть маскирован путем установки соответствующего бита в регистре UARTx->CTRL. Установка бита в 1 разрешает соответствующее прерывание, в 0 – запрещает.

В каждом контроллере эти прерывания объединяются по схеме ИЛИ в комбинированное прерывание. Комбинированное прерывание каждого контроллера подключено к контроллеру прерываний NVIC.

19.4.6.2 Прерывания UART1

В модуле предусмотрено 4 маскируемых источника прерывания.

UARTTXINTR – прерывание от передатчика. Прерывание возникает при опустошении буфера передатчика и сигнализирует о готовности принять новые данные от процессора для передачи. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

UARTRXINTR – прерывание от приемника. Прерывание возникает в случае возникновения события, когда принят символ данных. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

UARTTXOVRINTR – переполнение передающего буфера. Прерывание возникает при обнаружении ошибки, вызванной фактом переполнения буфера FIFO передатчика. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

UARTRXOVRINTR – переполнение приемного буфера. Прерывание возникает при обнаружении ошибки, вызванной фактом переполнения буфера FIFO приемника. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

19.4.6.3 Прерывания UART2, UART3

UARTTXINTR – прерывание от передатчика. Состояние прерывания от передатчика может измениться в случае возникновения события опустошения буфера FIFO передатчика. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

Примечание: В случае, если модуль и прерывания от него разрешены до осуществления записи данных в буфер FIFO передатчика, прерывание не формируется. Прерывание возникает только при опустошении буфера FIFO передатчика.

UARTRXINTR – прерывание от приемника. Состояние прерывания от приемника может измениться в случае возникновения события, когда принят символ данных. При

этом линия прерывания переходит в высокое состояние. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

UARTTXOVRINTR – переполнение передающего буфера. Прерывание по обнаружению ошибки, вызванной фактором переполнения буфера FIFO передатчика. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

UARTRXOVRINTR – переполнение приемного буфера. Прерывание по обнаружению ошибки, вызванной фактором переполнения буфера FIFO приемника. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

UARTRXFULLINTR – прерывание от приемника. Состояние прерывания от приемника изменяется в случае возникновения события когда буфер FIFO приемника заполнен полностью. Сигнал прерывания переходит в низкое состояние после сброса прерывания.

19.5 Описание регистров контроллера

Таблица 19.1 – Базовые адреса и смещения регистров контроллеров UART

Базовый адрес	**Название**	**Описание**
0x4000_4000	UART1	Контроллер UART1
0x4000_5000	UART2	Контроллер UART2
0x4000_6000	UART3	Контроллер UART3
Смещение		
0x00	DATA	Регистр UARTx->DATA доступа к буферам данных. **чтение** – принятые данные **запись** – передаваемые данные
0x04	STATE	Регистр UARTx->STATE состояния
0x08	CTRL	Регистр UARTx->CTRL управления
0x0C	INTSTATUS INTCLEAR	Регистры UARTx->INTSTATUS состояния и сброса прерывания
0x10	BAUDDIV	Регистр UARTx->BAUDDIV делителя

19.5.1 UARTx->DATA

Таблица 19.2 – Регистр UARTx->DATA

Номер	31…8	7…0
Доступ	U	R/W
Сброс	0	0
	-	**DATA**

Таблица 19.3 – Описание бит регистра UARTx->DATA

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…8	0	Зарезервировано
7…0	DATA	Принимаемые данные **(чтение)** Передаваемые данные **(запись)** При чтении выполняется доступ к последней несчитанной ячейке буфера FIFO приемника. При записи очередное слово помещается в буфер FIFO передатчика.

19.5.2 UARTx->STATE (контроллер UART1)

Таблица 19.4 – Регистр UARTx->STATE

Номер	31…4	3	2	1	0
Доступ	U	R/W	R/W	R	R
Сброс	0	0	0	0	0
	-	**RXOVR**	**TXOVR**	**RXF**	**TXF**

Таблица 19.5 – Описание бит регистра UARTx->STATE

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…4	-	Зарезервировано
3	RXOVR	Флаг переполненения премного буфера: **1** – произошло переполнение приемного буфера **0** – переполнения приемного буфера не было Запись 1 обнуляет статусный бит.
2	TXOVR	Флаг переполнения передающего буфера: **1** – произошло переполнение передающего буфера **0** – переполнения передающего буфера не было Запись 1 обнуляет статусный бит.
1	RXF	Буфер приемника заполнен: **1** – приемный буфер заполнен **0** – приемный буфер пуст
0	TXF	Буфер передатчика заполнен: **1** – передающий буфер заполнен **0** – передающий буфер пуст

19.5.3 UARTx->STATE (контроллеры UART2, UART3)

Таблица 19.6 – Регистр UARTx->STATE

Номер	31…7	6	5	4	3	2	1	0
Доступ	U	R	R	R	R/W	R/W	R/W	R/W
Сброс	0	0	1	1	0	0	0	0
	-	**TX_BUSY**	**RXE**	**TXE**	**RXOVR**	**TXOVR**	**RXF**	**TXF**

Таблица 19.7 – Описание бит регистра UARTx->STATE

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…7	-	Зарезервировано
6	TX_BUSY	Передатчик занят 1 – идёт передача байт 0 – передачи байт нет, в состоянии ожидания
5	RXE	Буфер приемника пуст: **1** – приемный буфер пуст **0** – приемный буфер не пуст
4	TXE	Буфер передатчика пуст: **1** – передающий буфер пуст **0** – передающий буфер не пуст
3	RXOVR	Флаг переполненения приемного буфера: **1** – произошло переполнение приемного буфера **0** – переполнения приемного буфера не было Запись 1 обнуляет статусный бит.
2	TXOVR	Флаг переполнения передающего буфера: **1** – произошло переполнение передающего буфера **0** – переполнения передающего буфера не было Запись 1 обнуляет статусный бит.
1	RXF	Буфер приемника заполнен: **1** – приемный буфер заполнен **0** – приемный буфер не заполнен Запись 1 очищает приемный буфер и статусный бит
0	TXF	Буфер передатчика заполнен: **1** – передающий буфер заполнен **0** – передающий буфер не заполнен Запись 1 очищает передающий буфер и статусный бит

19.5.4 UARTx->CTRL (контроллер UART1)

Таблица 19.8 – Регистр UARTx->CTRL

Номер	31…7	6	5	4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0	0	0	0
	-	**HSTM**	**RXOVRIE**	**TXOVRIE**	**RXIE**	**TXIE**	**RXEN**	**TXEN**

Таблица 19.9 – Описание бит регистра UARTx->CTRL

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…7	-	Зарезервировано
6	HSTM	Высокоскоростной режим передатчика **1** – высокоскоростной режим включен **0** – высокоскоростной режим отключен
5	RXOVRIE	Маска прерывания по событию «переполнение приемного буфера» (UARTRXOVRINTR) **1** – прерывание маскировано **0** – прерывание не маскировано
4	TXOVRIE	Маска прерывания по событию «переполнение передающего буфера» (UARTTXOVRINTR) **1** – прерывание маскировано **0** – прерывание не маскировано
3	RXIE	Маска прерывания по событию «прерывание от приемника» (UARTRXINTR) **1** – прерывание маскировано **0** – прерывание не маскировано
2	TXIE	Маска прерывания по событию «прерывание от передатчика» **1** – прерывание маскировано **0** – прерывание не маскировано
1	RXEN	Разрешение работы приемника **1** – работа приемника разрешена **0** – работа приемника запрещена
0	TXEN	Разрешение работы передатчика **1** – работа передатчика разрешена **0** – работа передатчика запрещена

19.5.5 UARTx->CTRL (контроллеры UART2, UART3)

Таблица 19.10 – Регистр UARTx->CTRL

Номер	31…8	7	6	5	4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0	0	0	0	0
	-	**RXFIE**	**HSTM**	**RXOVRIE**	**TXOVRIE**	**RXIE**	**TXIE**	**RXEN**	**TXEN**

Таблица 19.11 – Описание бит регистра UARTx->CTRL

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…8	-	Зарезервировано
7	RXFIE	Маска прерывания по событию «буфер приемника заполнен полностью» (UARTRXFULLINTR) **1** – прерывание маскировано **0** – прерывание не маскировано
6	HSTM	Высокоскоростной режим передатчика **1** – высокоскоростной режим включен **0** – высокоскоростной режим отключен
5	RXOVRIE	Маска прерывания по событию «переполнение приемного буфера» (UARTRXOVRINTR) **1** – прерывание маскировано **0** – прерывание не маскировано
4	TXOVRIE	Маска прерывания по событию «переполнение передающего буфера» (UARTTXOVRINTR) **1** – прерывание маскировано **0** – прерывание не маскировано
3	RXIE	Маска прерывания по событию «прерывание от приемника» (UARTRXINTR) **1** – прерывание маскировано **0** – прерывание не маскировано
2	TXIE	Маска прерывания по событию «прерывание от передатчика» **1** – прерывание маскировано **0** – прерывание не маскировано
1	RXEN	Разрешение работы приемника **1** – работа приемника разрешена **0** – работа приемника запрещена
0	TXEN	Разрешение работы передатчика **1** – работа передатчика разрешена **0** – работа передатчика запрещена

19.5.6 UARTx->INTSTATUS (контроллер UART1)

Таблица 19.12 – Регистр UARTx->INTSTATUS

Номер	31…4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0
	-	**RXOVRI**	**TXOVRI**	**RXI**	**TXI**

Таблица 19.13 – Описание бит регистра UARTx->INTSTATUS

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…4	-	Зарезервировано
3	RXOVRI	При **чтении** – состояние прерывания по событию «переполнение приемного буфера» (UARTRXOVRINTR) При **записи**: 1 – сброс прерывания по событию «переполнение приемного буфера» 0 – не влияет на признак прерывания по событию «переполнение приемного буфера»
2	TXOVRI	При **чтении** – состояние прерывания по событию «переполнение передающего буфера» (UARTTXOVRINTR) При **записи**: 1 – сброс прерывания по событию «переполнение передающего буфера» 0 – не влияет на признак прерывания по событию «переполнение передающего буфера»
1	RXI	При **чтении** – состояние прерывания по событию «прерывание от приемника» (UARTRXINTR) При **записи**: 1 – сброс прерывания по событию «прерывание от приемника» 0 – не влияет на признак прерывания по событию «прерывание от приемника»
0	TXI	При **чтении** – состояние прерывания по событию «прерывание от передатчика» (UARTTXINTR) При **записи**: 1 – сброс прерывания по событию «прерывание от передатчика» 0 – не влияет на признак прерывания по событию «прерывание от передатчика»

19.5.7 UARTx->INTSTATUS (UART2, UART3)

Таблица 19.14 – Регистр UARTx->INTSTATUS

Номер	31…5	4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W	R/W
Сброс	0	1	1	1	1	
	-	**RXFI**	**RXOVRI**	**TXOVRI**	**RXI**	**TXI**

Таблица 19.15 – Описание бит регистра UARTx->INTSTATUS

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…5	-	Зарезервировано
4	RXFI	При **чтении** – состояние прерывания по событию «приемный буфер заполнен полностью» (UARTRXFULLINTR) При **записи**: 1 – сброс прерывания по событию «приемный буфер заполнен полностью» 0 – не влияет на признак прерывания по событию «приемный буфер заполнен полностью»
3	RXOVRI	При **чтении** – состояние прерывания по событию «переполнение приемного буфера» (UARTRXOVRINTR) При **записи**: 1 – сброс прерывания по событию «переполнение приемного буфера» 0 – не влияет на признак прерывания по событию «переполнение приемного буфера»
2	TXOVRI	При **чтении** – состояние прерывания по событию «переполнение передающего буфера» (UARTTXOVRINTR) При **записи**: 1 – сброс прерывания по событию «переполнение передающего буфера» 0 – не влияет на признак прерывания по событию «переполнение передающего буфера»
1	RXI	При **чтении** – состояние прерывания по событию «прерывание от приемника» (UARTRXINTR) При **записи**: 1 – сброс прерывания по событию «прерывание от приемника» 0 – не влияет на признак прерывания по событию «прерывание от приемника»
0	TXI	При **чтении** – состояние прерывания по событию «прерывание от передатчика» (UARTTXINTR) При **записи**: 1 – сброс прерывания по событию «прерывание от передатчика» 0 – не влияет на признак прерывания по событию «прерывание от передатчика»

19.5.8 UARTx->BAUDDIV

Таблица 19.16 – Регистр UARTx->BAUDDIV

Номер	31…20	19…0
Доступ	U	R/W
Сброс	0	0
	-	**BAUDDIV**

Таблица 19.17 – Описание бит регистра UARTx->BAUDDIV

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…20	-	Зарезервировано
19…0	BAUDDIV	Значение делителя скорости передачи. Минимальное значение – 16.

20 Контроллер интерфейса по ГОСТ Р52070-2003

20.1 В состав микроконтроллера входят два независимых контроллера интерфейса по ГОСТ Р 52070-2003.

20.2 Основные характеристики контроллера

20.2.1 Контроллер реализует обработку сообщений интерфейса ГОСТ Р 52070-2003 в одном из трех режимов: контроллер шины (КШ), оконечное устройство (ОУ), монитор шины (МШ).

В режиме КШ осуществляется обработка только одного сообщения, запрограммированного пользователем.

В режиме ОУ контроллер фильтрует сообщения в соответствии со своим адресом ОУ, обрабатывает команды управления и принимает/передает данные. Слова данных со стороны пользователя и со стороны шины хранятся раздельно, что позволяет реализовать дуплексный обмен данными для каждого интерфейсного подадреса (0-31).

В режиме МШ контроллер только принимает сообщения и сохраняет их во внутренней памяти для анализа пользователем. Максимальное количество одновременно хранимых сообщений равно 32.

Стандартная скорость обмена равна 1 Мбит/с. Поддерживается программная настройка временных параметров контроллера.

Для подключения к основной и резервной шинам интерфейса ГОСТ Р 52070-2003 реализованы два приемо-передающих канала. Прием сообщений одновременно по двум шинам не предусмотрен.

20.3 Описание регистров и памяти контроллера

Таблица 20.1 – Базовые адреса и смещения регистров контроллеров

Базовый адрес	**Название**	**Описание**
0x4000_0000	MIL1	Контроллер интерфейса по ГОСТ Р 52070-2003 MIL1
0x4000_2000	MIL2	Контроллер интерфейса по ГОСТ Р 52070-2003 MIL2
Смещение		
0x1FD0 : 0x0000	DATA	Память DATA контроллера
0x1FD4	DBGDEC_D	Регистр MILx->DBGDEC_D
0x1FD8	DBGDEC_C	Регистр MILx->DBGDEC_C
0x1FDC	DBGDEC_B	Регистр MILx->DBGDEC_B
0x1FE0	DBGDEC_A	Регистр MILx->DBGDEC_A
0x1FE4	TIME_D	Регистр MILx->TIME_D
0x1FE8	INTERRUPT	Регистр MILx->INTERRUPT
0x1FEC	TIME_C	Регистр MILx->TIME_C
0x1FF0	TIME_B	Регистр MILx->TIME_B
0x1FF4	TIME_A	Регистр MILx->TIME_A
0x1FF8	CONTROL	Регистр MILx->CONTROL
0x1FFC	STATUS	Регистр MILx->STATUS статуса

20.3.1 **MILx->STATUS**

Таблица 20.2 – Регистр MILx->STATUS

Номер	31…21	20	19…14	13…9	8	7	6	5	4	3	2	1	0
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0	0	0	0	0	0	0	0	0
	-	**RT_BUS**	**RT_WCMC**	**RT_SUBADDR**	**RT_TR**	**RT_BROAD**	**RT_MC**	**ERR_NOWORD**	**ERR_NOGAP**	**ERR_SYNC**	**ERR_PAR**	**ERR_M2**	**MSG_OK**

Таблица 20.3 – Описание бит регистра MILx->STATUS

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…21	-	Зарезервировано
20	RT_BUS	**1** – в режиме ОУ сообщение выполнено по второй шине интерфейса. **0** – в режиме ОУ сообщение выполнено по первой шине интерфейса.
19…14	RT_WCMC	В режиме ОУ: если в сообщении была команда управления, то RT_WCMC[18:14] = стандартный код команды управления; в остальных случаях RT_WCMC[19:14] = количество принятых/переданных слов данных.
13…9	RT_SUBADDR	В режиме ОУ определяет подадрес, указанный в сообщении.
8	RT_TR	В режиме ОУ определяет направление передачи слов данных: **0** – в ОУ. **1** – из ОУ.
7	RT_BROAD	**0** – в режиме ОУ сообщение негрупповое. **1** – в режиме ОУ сообщение групповое.
6	RT_MC	**0** – не определено. **1** – в режиме ОУ "обработана команда управления"
5	ERR_NOWORD	**0** – нет ошибки. **1** – для режимов КШ и ОУ "нет ответа": не принято ответное слово или слово данных в течении времени, определяемого полем NOWORD (регистр TIME_B).
4	ERR_NOGAP	**0** – нет ошибки. **1** – для режимов КШ и ОУ "нет паузы": ответное слово или новое сообщение принято без паузы, определяемой полем NOGAP (регистр TIME_B).
3	ERR_SYNC	**0** – нет ошибки. **1** – для режимов КШ и ОУ "ошибка типа слова": должно быть слово данных, а получено командное/ответное или наоборот.
2	ERR_PAR	**0** – нет ошибки. **1** – для режимов КШ и ОУ "ошибка бита четности".
1	ERR_M2	**0** – нет ошибки. **1** – для режимов КШ и ОУ "ошибка Манчестер II кодирования".
0	MSG_OK	**0** – не определено. **1** – для режимов КШ и ОУ "сообщение выполнено успешно". Также в режиме ОУ через регистр INTERRUPT выбирается для каких команд управления этот бит будет устанавливаться. Для режима МШ обозначает "принята группа сообщений". Количество сообщений в группе выбирается полем MT_MSGINTV (регистр CONTROL).

20.3.2 **MIL->CONTROL**

Таблица 20.4 – Регистр MILx->CONTROL

Номер	31…25	24	23	22…19	18	17	16…12	11	10	9	8	7	6…2	1…0
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	-	**OUT_POLAR**	**EN_POLAR**	**BC_MSGTYPE**	**BC_BUS**	**BS_START**	**MT_MSGINTV**	**RT_BUSY**	**RT_DYNBC**	**RT_TERMF**	**RT_SUBSYSF**	**RT_SERVREQ**	**RT_ADDR**	**MODE**

Таблица 20.5 – Описание бит регистра MILx->CONTROL

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…25	-	Зарезервировано
24	OUT_POLAR	Определяет состояние прямого и инверсного выходов, когда передача отсутствует. **0** – прямой и инверсный выход находятся в 0. **1** – прямой и инверсный выход находятся в 1.
23	EN_POLAR	Определяет состояние выхода включения передатчика, когда передача отсутствует. **0** – выход включения передатчика находится в 0. **1** – выход включения передатчика находится в 1.
22…19	BC_MSGTYPE	В режиме КШ определяет формат сообщения. **0** – передача данных от КШ к ОУ. **1** – передача данных от ОУ к КШ. **2** – передача данных от ОУ к ОУ. **3** – передача команды управления. **4** – передача команды управления и прием слова данных от ОУ. **5** – передача команды управления со словом данных оконечному устройству. **6** – передача данных (в групповом сообщении) от КШ к ОУ. **7** – передача данных (в групповом сообщении) от ОУ к ОУ. **8** – передача групповой команды управления. **9** – передача групповой команды управления со словом данных.
18	BC_BUS	**0** – в режиме КШ сообщение выполнить по первой шине интерфейса. **1** – в режиме КШ сообщение выполнить по второй шине интерфейса.
17	BC_START	**0** – не определено. **1** – начать сообщение. После начала сообщения бит автоматически сбрасывается контроллером.
16…12	MT_MSGINTV	В режиме МШ определяет количество принятых сообщений, после которого будет установлен MSG_OK = 1 (регистр STATUS).
11	RT_BUSY	В режиме ОУ определяет значение бита "абонент занят" в ответном слове.
10	RT_DYNBC	В режиме ОУ определяет значение бита "принято управление интерфейсом" в ответном слове.
9	RT_TERMF	В режиме ОУ определяет значение бита "неисправность ОУ" в ответном слове.
8	RT_SUBSYSF	В режиме ОУ определяет значение бита "неисправность абонента" в ответном слове.
7	RT_SERVREQ	В режиме ОУ определяет значение бита "запрос на обслуживание" в ответном слове.
6…2	RT_ADDR	В режиме ОУ определяет адрес ОУ.
1…0	MODE	**0** – программный сброс контроллера, не сбрасываются значения адресуемых регистров контроллера. **1** – режим ОУ. **2** – режим МШ. **3** – режим КШ.

20.3.3 **MILx->INTERRUPT**

Таблица 20.6 – Регистр MILx->INTERRUPT

Номер	31…21	20	19	18	17	16	15	14	13	12	11	10
Доступ	U	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0	0	0	0	0	0	0	0
	-	**EVTMC_DYNBC**	**EVTMC_SYNC**	**EVTMC_SENDSW**	**EVTMC_INITST**	**EVTMC_ENCON**	**EVTMC_ENCOFF**	**EVTMC_MASKON**	**EVTMC_OVMSKOFF**	**EVTMC_RESET**	**EVTMC_SENDVEC**	**EVTMC_SENDCW**

Номер	9	8	7	6	5	4	3	2	1	0
Доступ	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Сброс	0	0	0	0	0	0	0	0	0	0
	EVTMC_SYNCDW	**EVTMC_SENDBITW**	**EVTMC_SENCON**	**EVTMC_SENCOFF**	**INT_ERR_NOWORD**	**INT_ERR_NOGAP**	**INT_ERR_SYNC**	**INT_ERR_PAR**	**INT_ERR_M2**	**INT_MSG_OK**

Таблица 20.7 – Описание бит регистра MILx->INTERRUPT

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…21	-	Зарезервировано
20	EVTMC_DYNBC	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "принять управление интерфейсом".
19	EVTMC_SYNC	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "синхронизация".
18	EVTMC_SENDSW	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "передать ответное слово".
17	EVTMC_INITST	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "начать самоконтроль ОУ".
16	EVTMC_ENCON	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "разблокировать передатчик".
15	EVTMC_ENCOFF	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "блокировать передатчик".
14	EVTMC_MASKON	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "блокировать признак неисправности ОУ".

Продолжение таблицы 20.7

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
13	EVTMC_OVMSKOFF	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "разблокировать признак неисправности ОУ".
12	EVTMC_RESET	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "установить ОУ в исходное состояние".
11	EVTMC_SENDVEC	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "передать векторное слово".
10	EVTMC_SENDCW	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "передать последнюю команду".
9	EVTMC_SYNCDW	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "синхронизация с СД".
8	EVTMC_SENDBITW	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "передать слово встроенной системы контроля".
7	EVTMC_SENCON	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "разблокировать i-ый передатчик".
6	EVTMC_SENCOFF	**0** – не определено. **1** – в режиме ОУ разрешение установки бита MSG_OK (регистр STATUS) для команды управления "блокировать i-ый передатчик".
5	INT_ERR_NOWORD	**0** – не определено. **1** – прерывание по ERR_NOWORD (регистр STATUS).
4	INT_ERR_NOGAP	**0** – не определено. **1** – прерывание по ERR_NOGAP (регистр STATUS).
3	INT_ERR_SYNC	**0** – не определено. **1** – прерывание по ERR_SYNC (регистр STATUS).
2	INT_ERR_PAR	**0** – не определено. **1** – прерывание по ERR_PAR (регистр STATUS).
1	INT_ERR_M2	**0** – не определено. **1** – прерывание по ERR_M2 (регистр STATUS).
0	INT_MSG_OK	**0** – не определено. **1** – прерывание по MSG_OK (регистр STATUS).

20.3.4 **MILx->DATA память данных в режиме КШ**

Таблица 20.8 – Память DATA в режиме КШ

Адрес смещения	Доступ	Имя	Описание
0x00001FD0:0x00000094	R/W		Не используется.
0x00000090:0x00000014	R/W	DW	32 передаваемых/принимаемых слова данных. Первое слово, принятое с шины или передаваемое на шину, размещается по адресу=0x00000014, второе по адресу=0x00000018 и т.д.
0x00000010	R/W	SW1	Второе ответное слово используется в форматах сообщений от ОУ к ОУ.
0x0000000C	R/W	SW0	Первое ответное слово.
0x00000008	R/W	CW1	Второе командное слово используется в форматах сообщений от ОУ к ОУ.
0x00000004	R/W	CW0	Первое командное слово.
0x00000000	R/W		Не используется.

Слова хранятся без синхрополя и без бита паритета, только 2 байта данных. Адреса слов выравнены по 4-байтным границам и хранят 4 байта данных (little-endian): старшие 2 байта заполнены нулями, младшие 2 байта занимает слово.

20.3.5 **MIL->DATA память данных в режиме ОУ**

Таблица 20.9 – Память DATA в режиме ОУ

Подадрес	**Адрес смещения**	**Доступ**	**Имя**	**Описание**
31	0x00000FFC:0x00000F94	R/W	-	Не используется.
31	0x00000F90	R/W	TROFFW	Слово данных, принятое по команде управления "блокировать i передатчик" в 31 подадрес.
31	0x00000F8C	R/W	TRONW	Слово данных, принятое по команде управления "разблокировать i передатчик" в 31 подадрес.
31	0x00000F88	R/W	BITW	Слово данных, передаваемое по команде управления "передать слово встроенной системы контроля" в 31 подадрес.
31	0x00000F84	R/W	VECW	Слово данных, передаваемое по команде управления "передать векторное слово" в 31 подадрес.
31	0x00000F80	R/W	SYNCW	Слово данных, принятое по команде управления "синхронизация с словом данных" в 31 подадрес.
30	0x00000F7C:0x00000F00	R/W	DW	При обмене данными с конкретным подадресом первое слово, принятое с шины или передаваемое на шину, размещается по младшему адресу.
…	+0x80	R/W	DW	
2	0x0000017C:0x00000100	R/W	DW	
1	0x000000FC:0x00000080	R/W	DW	
0	0x0000007C:0x00000018	R/W	-	Не используется.
0	0x00000010	R/W	TROFFW	Слово данных, принятое по команде управления "блокировать i передатчик" в 0 подадрес.
0	0x0000000C	R/W	TRONW	Слово данных, принятое по команде управления "разблокировать i передатчик" в 0 подадрес.
0	0x00000008	R/W	BITW	Слово данных, передаваемое по команде управления "передать ВСК слово" в 0 подадрес.
0	0x00000004	R/W	VECW	Слово данных, передаваемое по команде управления "передать векторное слово" в 0 подадрес.
0	0x00000000	R/W	SYNCW	Слово данных, принятое по команде управления "синхронизация с СД" в 0 подадрес.

Каждому адресу соответствует две ячейки А и Б. Пользователь, выполняя запись по адресу, производит запись ячейки А, из которой данные могут быть прочитаны со стороны шины. Когда пользователь выполняет чтение по адресу, он читает из ячейки Б, в которую данные могут записаны со стороны шины. Это позволяет для каждого адреса осуществлять дуплексный обмен данными.

При обмене данными в командном слове указывается номер подадреса (0-31), откуда/куда читать/писать слова данных. Нулевой подадрес начинается с 0x00000000, первый подадрес с 0x00000080, начало каждого подадреса следует с интервалом +0x80, что соответствует 32 словам данных, хранимых в каждом подадресе. Слова данных хранятся без синхрополя и без бита паритета, только 2 байта данных. Адреса слов выравнены по 4-байтным границам и хранят 4 байта данных (little-endian): старшие 2 байта заполнены нулями, младшие 2 байта занимает слово.

20.3.6 **MIL->DATA память данных в режиме МШ**

Таблица 20.10 – Память DATA в режиме МШ

Блок	**Адрес смещения**	**Доступ**	**Имя**	**Описание**
31	0x00001FD0: 0x00001F00	R/W		Поля такие же как у нулевого блока.
…	+0x100	R/W		
2	0x000002FC: 0x00000200	R/W		
1	0x000001FC: 0x00000100	R/W		
0	0x000000FC: 0x00000094	R/W		Не используется.
0	0x00000090: 0x00000014	R/W	DW	32 слова данных. Первое слово данных принятое с шины, размещается по младшему адресу.
0	0x00000010	R/W	SW1	Второе ответное слово используется в форматах сообщений от ОУ к ОУ.
0	0x0000000C	R/W	SW0	Первое ответное слово.
0	0x00000008	R/W	CW1	Второе командное слово используется в форматах сообщений от ОУ к ОУ.
0	0x00000004	R/W	CW0	Первое командное слово.
0	0x00000000	R/W	MSGST	Монитор-слово сообщения.

Слова хранятся без синхрополя и без бита паритета, только 2 байта данных. Адреса слов выравнены по 4-байтным границам и хранят 4 байта данных (little-endian): старшие 2 байта заполнены нулями, младшие 2 байта занимает слово.

20.3.7 **MILx->TIME_A**

Таблица 20.11 – Регистр MILx->TIME_A

Номер	31…24	23…8	7…0
Доступ	U	R/W	R/W
Сброс	0	500	50
	-	**PAUSE**	**HALFBIT**

Таблица 20.12 – Описание бит регистра MILx->TIME_A

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…24	-	Зарезервировано
23…8	PAUSE	Пауза перед передачей ответного слова в режиме ОУ и пауза перед началом передачи нового сообщения в режиме КШ. Стандартный диапазон от 4×HALFBIT до 20×HALFBIT, что соответствует временному интервалу 4-12 мкс по ГОСТ Р 52070-2003.
7…0	HALFBIT	Длительность половины бита на шине интерфейса.

Временные параметры подробно описаны в подразделе 20.3 «Инициализация контроллера»

20.3.8 **MILx->TIME_B**

Таблица 20.13 – Регистр MILx->TIME_B

Номер	31…16	15…0
Доступ	R/W	R/W
Сброс	300	1400
	NOGAP	**NOWORD**

Таблица 20.14 – Описание бит регистра MILx->TIME_B

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…16	NOGAP	Минимальная пауза приема ответного слова и приема нового сообщения. Рекомендуемое значение 2×HALFBIT.
15…0	NOWORD	Максимальная пауза приема ответного слова или слова данных. Стандартное значение должно быть не менее 24×HALFBIT, что соответствует временному интервалу не менее 14 мкс по ГОСТ Р 52070-2003.

Временные параметры подробно описаны в подразделе 20.3 «Инициализация контроллера»

20.3.9 **MILx->TIME_C**

Таблица 20.15 – Регистр MILx->TIME_C

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	15	15	15	15
	JTR_NBMAX	**JTR_NBMIN**	**JTR_1BMAX**	**JTR_1BMIN**

Таблица 20.16 – Описание бит регистра MILx->TIME_C

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…24	JTR_NBMAX	Определяет правую границу окна приема NB. Рекомендуемое значение HALFBIT/2.
23…16	JTR_NBMIN	Определяет левую границу окна приема NB. Рекомендуемое значение HALFBIT/2.
15…8	JTR_1BMAX	Определяет правую границу окна приема 1B. Рекомендуемое значение HALFBIT/2.
7…0	JTR_1BMIN	Определяет левую границу окна приема 1B. Рекомендуемое значение HALFBIT/2.

20.3.10 **MILx->TIME_D**

Таблица 20.17 – Регистр MILx->TIME_D

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	15	15	15	15
	JTR_NSMAX	**JTR_NSMIN**	**JTR_1SMAX**	**JTR_1SMIN**

Таблица 20.18 – Описание бит регистра MILx->TIME_D

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…24	JTR_NSMAX	Определяет правую границу окна приема NS. Рекомендуемое значение 2×HALFBIT.
23…16	JTR_NSMIN	Определяет левую границу окна приема NS. Рекомендуемое значение 2×HALFBIT.
15…8	JTR_1SMAX	Определяет правую границу окна приема 1S. Рекомендуемое значение 2×HALFBIT.
7…0	JTR_1SMIN	Определяет левую границу окна приема 1S. Рекомендуемое значение 2×HALFBIT.

20.3.11 **MIL->DBGDEC_A**

Таблица 20.19 – Регистр MILx->DBGDEC_A

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	0	0	0	0
	TR_NS	**TR_NB**	**TR_1B**	**TR_1S**

Таблица 20.20 – Описание бит регистра MILx->DBGDEC_A

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…24	TR_NS	Время центрального перепада в окне приема NS.
23…16	TR_NB	Время центрального перепада в окне приема NB.
15…8	TR_1B	Время центрального перепада в окне приема 1B.
7…0	TR_1S	Время центрального перепада в окне приема 1S.

20.3.12 **MIL->DBGDEC_B**

Таблица 20.21 – Регистр MILx->DBGDEC_B

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	0	0	0	0
	TR_NSANY	**TR_NBANY**	**TR_1BANY**	**TR_1SANY**

Таблица 20.22 – Описание бит регистра MILx->DBGDEC_B

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…24	TR_NSANY	Время перепада, который не попал в окно приема NS.
23…16	TR_NBANY	Время перепада, который не попал в окно приема NB.
15…8	TR_1BANY	Время перепада, который не попал в окно приема 1B.
7…0	TR_1SANY	Время перепада, который не попал в окно приема 1S.

20.3.13 **MIL->DBGDEC_C**

Таблица 20.23 – Регистр MILx->DBGDEC_C

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	0	0	0	0
	TR_NS	**TR_NB**	**TR_1B**	**TR_1S**

Таблица 20.24 – Описание бит регистра MILx->DBGDEC_C

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…24	TR_NS	Время центрального перепада в окне приема NS.
23…16	TR_NB	Время центрального перепада в окне приема NB.
15…8	TR_1B	Время центрального перепада в окне приема 1B.
7…0	TR_1S	Время центрального перепада в окне приема 1S.

20.3.14 **MIL->DBGDEC_D**

Таблица 20.25 – Регистр MILx->DBGDEC_D

Номер	31…24	23…16	15…8	7…0
Доступ	R/W	R/W	R/W	R/W
Сброс	0	0	0	0
	TR_NSANY	**TR_NBANY**	**TR_1BANY**	**TR_1SANY**

Таблица 20.26 – Описание бит регистра MILx->DBGDEC_D

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…24	TR_NS	Время перепада, который не попал в окно приема NS.
23…16	TR_NB	Время перепада, который не попал в окно приема NB.
15…8	TR_1B	Время перепада, который не попал в окно приема 1B.
7…0	TR_1S	Время перепада, который не попал в окно приема 1S.

20.4 Инициализация контроллера

20.4.1 Сброс контроллера осуществляется сигналом сброса от процессора или программно: установкой поля MODE = 0 в регистре CONTROL, программный сброс не сбрасывает значение адресуемых регистров контроллера.

Далее требуется установить временные параметры контроллера в регистрах TIME_A, TIME_B, TIME_C, TIME_D. Все времена в этих регистрах рассчитываются по формуле: записанное число × период тактового сигнала контроллера.

В регистре TIME_A поле HALFBIT определяет длительность половины бита (Thb) на шине интерфейса. На рисунке 20.1 показано начало слова с указанием времени половины бита и времени бита Tb = 2×Thb.

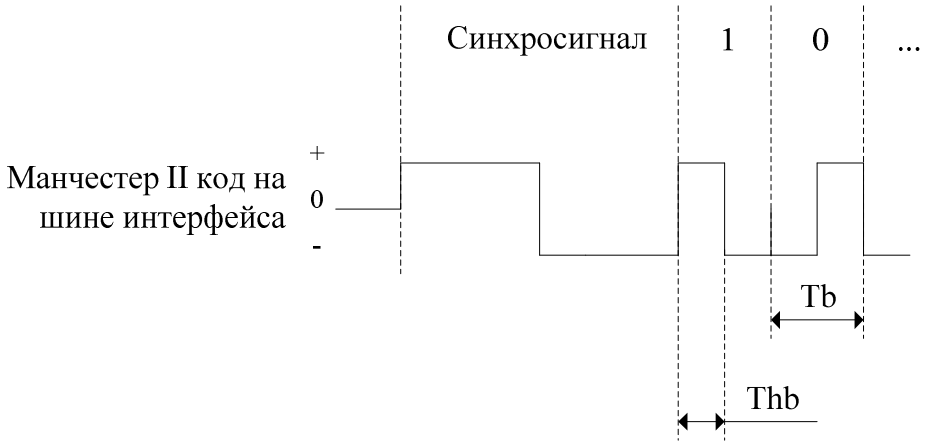

Рисунок 20.1 – Время бита

Поле PAUSE определяет паузу (Tpause) перед передачей ответного слова в режиме ОУ и паузу перед началом нового сообщения в режиме КШ. На рисунке 20.2 показана пауза между двумя словами, которую генерирует контроллер. При установке значений нужно учитывать, что начало и конец интервала отличаются от стандартных, которые замеряются по центральным перепадам бита паритета и синхрополя. Другими словами, контроллер создает временной интервал без учета половины бита паритета (0,5 мкс) и половины синхрополя (1,5 мкс). Стандартный диапазон лежит в пределах от 4×HALFBIT до 20×HALFBIT, что соответствует временному интервалу 4-12 мкс по ГОСТ Р 52070-2003.

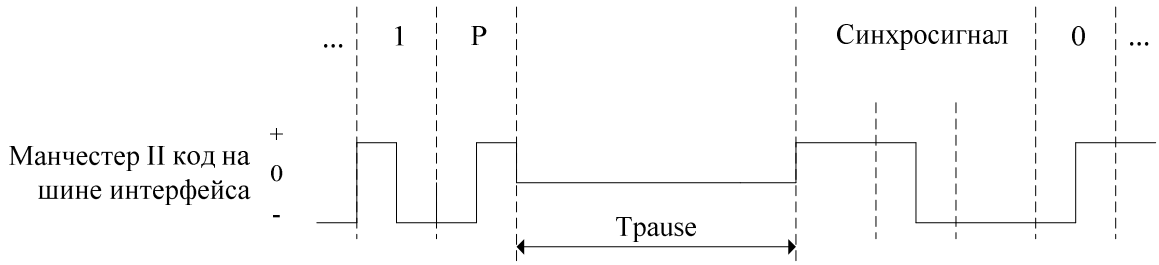

Рисунок 20.2 – Время паузы

В регистре TIME_B поле NOWORD задает максимальную паузу приема ответного слова или слова данных (Tnoword). Если ответ не получен, то генерируется ошибка "нет ответа" бит ERR_NOWORD регистра STATUS. На рисунке 20.3 показана ситуация, когда контроллер в режиме КШ передал данные, а ответа от ОУ не пришло. В этом случае контроллер сгенерирует ошибку "нет ответа", и начнет передавать следующее указанное сообщение. При установке значений нужно учитывать, что начало и конец интервала отличаются от стандартных, которые замеряются по центральным перепадам бита паритета и синхрополя. Другими словами, контроллер замеряет временной интервал без учета половины бита паритета (0,5 мкс) и половины синхрополя (1,5 мкс). Стандартное

значение должно быть не менее 24×HALFBIT, что соответствует временному интервалу не менее 14 мкс по ГОСТ Р 52070-2003.

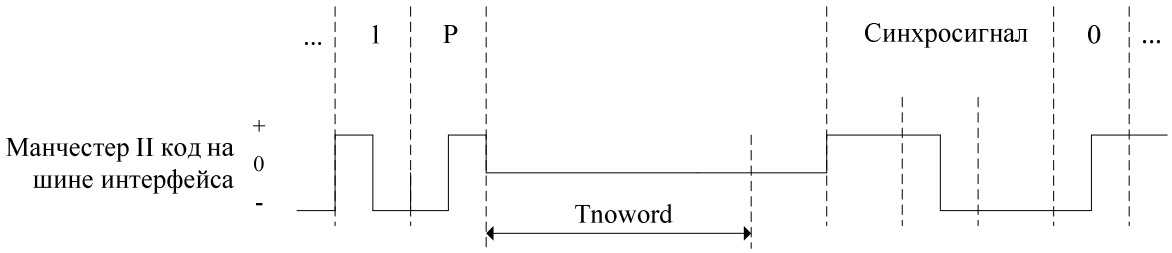

Рисунок 20.3 – Максимальная пауза приема

Поле NOGAP задает минимальную паузу приема ответного слова или нового командного слова (Tnogap). Если ответ получен в течение этого времени, то генерируется ошибка "ответ без паузы" бит ERR_NOGAP регистра STATUS. На рисунке 20.4 показана ситуация, когда контроллер в режиме КШ передал данные, а ответ от ОУ пришел раньше минимальной паузы. В этом случае контроллер сгенерирует ошибку "ответ без паузы", и проигнорирует ответное слово. Рекомендуемое значение 2×HALFBIT.

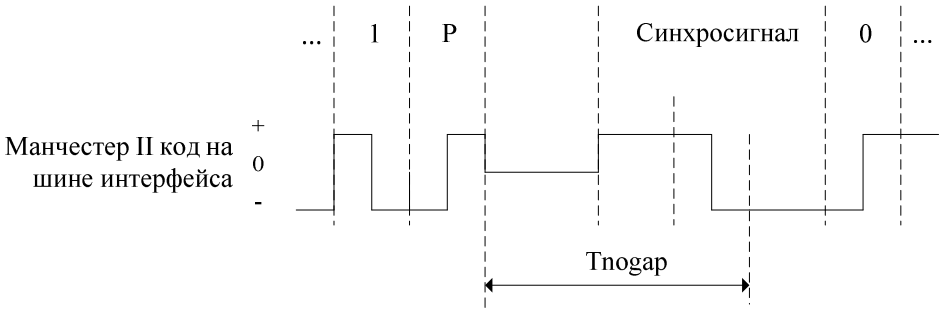

Рисунок 20.4 – Минимальная пауза приема

Контроллер имеет возможность настройки параметров приема Манчестер II кода через регистры TIME_C, TIME_D. На рисунке 20.5 показана пачка из 2 слов.

Вначале шина переходит из состояния простоя в первую половину синхрополя первого слова. Контроллер регистрирует этот перепад и запускает счетчик времени. Номинальная длительность половины синхрополя 3×HALFBIT, затем должен произойти центральный перепад и начаться вторая половина. Регистрация этого перепада осуществляется в окне 1S: с левой границей (3×HALFBIT)−JTR_1SMIN и правой границей (3×HALFBIT)+JTR_1SMAX. Перепады вне этого окна считаются ошибкой "Манчестер II кодирования" (ERR_M2 = 1). Минимальное значение левой границы равно началу синхрополя (JTR_1SMIN = 3×HALFBIT), максимальное значение правой ограничено только разрядностью счетчика времени. Чем больше границы, тем более искаженное синхрополе примет контроллер. Рекомендация JTR_1SMIN = 2×HALFBIT, JTR_1SMAX = 2×HALFBIT. Посмотреть значение счетчика времени для окна 1S можно в регистрах DBGDEC_A, DBGDEC_C поле TR_1S.

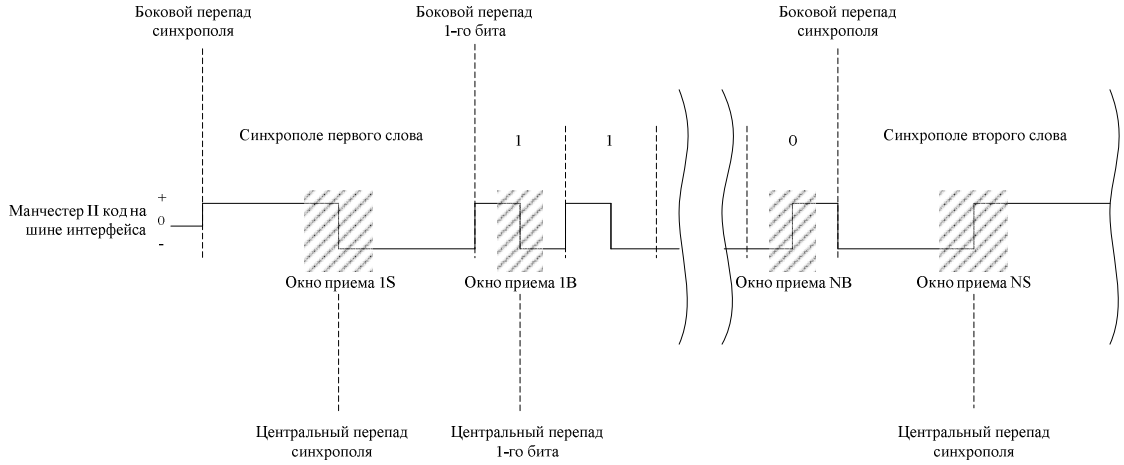

Рисунок 20.5 – Диаграмма окон приема

После центрального перепада синхрополя следует вторая половина синхрополя (длительность 3×HALFBIT) плюс первая половина 1 бита (длительность HALFBIT). Контроллер ожидает центрального перепада 1 бита в окне 1B: с левой границей (4×HALFBIT)−JTR_1BMIN и правой границей (4×HALFBIT)+JTR_1BMAX. Перепады справа этого окна считаются ошибкой "Манчестер II кодирования" (ERR_M2 = 1). Минимальное значение левой границы равно началу первого бита (JTR_1SMIN = HALFBIT), максимальное значение правой границы равно началу второго бита (JTR_1SMAX = HALFBIT). Чем больше левая граница, тем левее контроллер может зафиксировать центральный перепад, но тем жестче становятся требования для бокового перепада 1 бита, который может попасть в расширенное окно. Чем больше правая граница, тем правее контроллер может зафиксировать центральный перепад. Рекомендация JTR_1BMIN = HALFBIT/2, JTR_1BMAX = HALFBIT/2. Посмотреть значение счетчика времени для окна 1B можно в регистрах DBGDEC_A, DBGDEC_C поле TR_1B.

После центрального перепада 1 бита следует вторая половина 1 бита (длительность HALFBIT) плюс первая половина 2 бита (длительность HALFBIT). Контроллер ожидает центрального перепада 2 и последующих битов в окне NB: с левой границей (2×HALFBIT)−JTR_NBMIN и правой границей (2×HALFBIT)+JTR_NBMAX. Перепады справа этого окна считаются ошибкой "Манчестер II кодирования" (ERR_M2 = 1). Минимальное значение левой границы равно началу второго бита (JTR_NBMIN = HALFBIT), максимальное значение правой границы равно началу второго бита (JTR_NBMAX = HALFBIT). Чем больше левая граница, тем левее контроллер может зафиксировать центральный перепад, но тем жестче становятся требования для бокового перепада 2 бита, который может попасть в расширенное окно. Чем больше правая граница, тем правее контроллер может зафиксировать центральный перепад. Рекомендация JTR_NBMIN = HALFBIT/2, JTR_NBMAX = HALFBIT/2. Окно NB используется для остальных битов также. Посмотреть значение счетчика времени для окна NB можно в регистрах DBGDEC_A, DBGDEC_C поле TR_NB.

После центрального перепада последнего бита следует вторая половина этого бита (длительность HALFBIT) плюс первая половина синхрополя второго слова (длительность 3×HALFBIT). Контроллер ожидает центрального перепада синхрополя в окне NS: с левой границей (4×HALFBIT)−JTR_NSMIN и правой границей (4×HALFBIT)+JTR_NSMAX. Перепады справа этого окна считаются ошибкой "Манчестер II кодирования" (ERR_M2 = 1). Минимальное значение левой границы равно началу синхрополя (JTR_NBMIN = 3×HALFBIT), максимальное значение правой границы ограничено только разрядностью счетчика времени. Чем больше левая граница, тем левее контроллер может зафиксировать центральный перепад, но тем жестче становятся требования для бокового перепада синхрополя, который может попасть в расширенное окно. Чем больше правая граница, тем правее контроллер может зафиксировать центральный перепад.

Рекомендация JTR_NBMIN = 2×HALFBIT, JTR_NBMAX = 2×HALFBIT. Окно NS используется для остальных синхрополей в пачке. Посмотреть значение счетчика времени для окна NS можно в регистрах DBGDEC_A, DBGDEC_C поле TR_NS.

Пример установки временных параметров при периоде тактового сигнала Tclk = 20 нс:

HALFBIT (регистр TIME_A): так как стандартное время половины бита равно 500 нс, то HALFBIT = Thb / Tclk = 500 нс / 20 нс = 25.

PAUSE (регистр TIME_A): стандартный диапазон лежит в пределах от 4×HALFBIT до 20×HALFBIT, значение выбираем например чуть больше нижней границы PAUSE = 5×HALFBIT = 5×25 = 125.

NOWORD (регистр TIME_B): стандартное значение должно быть не менее 24×HALFBIT, выбираем чуть больше NOWORD = 26×HALFBIT = 26×25 = 650.

NOGAP (регистр TIME_B): NOGAP = 2×HALFBIT = 2×25 = 50.

JTR_1BMIN, JTR_NBMIN, JTR_1BMAX, JTR_NBMAX (регистр TIME_C):

JTR_1BMIN = JTR_NBMIN = JTR_1BMAX = JTR_NBMAX = HALFBIT/2 = 12.

JTR_1SMIN, JTR_NSMIN, JTR_1SMAX, JTR_NSMAX (регистр TIME_D):

JTR_1SMIN = JTR_NSMIN = JTR_1SMAX = JTR_NSMAX = 2×HALFBIT = 50.

После установки временных параметров контроллера выбирается один из трех режимов работы в регистре CONTROL:

MODE = 1: режим оконечного устройства.

MODE = 2: режим монитора шины. Одновременно или до установки этого поля пользователь должен установить значение поля MT_MSGINTV.

MODE = 3: режим контроллера шины.

20.5 Режим контроллера шины (КШ)

20.5.1 Проведя инициализацию контроллера, пользователь записывает поля памяти DATA (командные слова, слова данных) в соответствии с форматом требуемого сообщения. Далее в регистре CONTROL указывает формат сообщения в поле BC_MSGTYPE, шину интерфейса в поле BC_BUS и устанавливает бит BC_START = 1 для старта сообщения. Результат выполнения сообщения контроллер выводит в регистр STATUS: "сообщение выполнено успешно" бит MSG_OK = 1 или "сообщение не выполнено" один из битов ошибок ERR_M2, ERR_PAR, ERR_SYNC, ERR_NOGAP, ERR_NOWORD равен 1 и бит MSG_OK = 0. После успешно выполненного сообщения пользователь читает поля памяти DATA (ответные слова, слова данных). Контроллер не декодирует содержимого ответных слов и слов данных, он проверяет правильность Манчестер II кодирования (ERR_M2), тип синхрополя (ERR_SYNC), бит паритета (ERR_PAR) и время ответа (ERR_NOWORD, ERR_NOGAP).

Пример сообщения ГОСТ Р 52070-2003 формат 1 — передача двух слов данных от контроллера шины к оконечному устройству:

Запись командного слова в поле CW0 памяти DATA (0x00000004).

Запись первого слова данных в поле DW память DATA (0x00000014).

Запись второго слова данных в поле DW память DATA (0x00000018).

Запись регистра CONTROL: BC_MSGTYPE = 0 (тип сообщения КШ к ОУ), BC_BUS = 0 (по первой шине), BC_START = 1 (начать сообщение).

Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

Запись нулей в регистр STATUS: сброс события.

Чтение ответного слова из памяти DATA поле SW0 (0x0000000C).

Пример сообщения ГОСТ Р 52070-2003 формат 2 — передача двух слов данных от оконечного устройства к контроллеру шины:

Запись командного слова в поле CW0 памяти DATA (0x00000004).

Запись регистра CONTROL: BC_MSGTYPE = 1 (тип сообщения ОУ к КШ), BC_BUS = 0 (по первой шине), BC_START = 1 (начать сообщение).

Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

Запись нулей в регистр STATUS: сброс события.

Чтение ответного слова из памяти DATA поле SW0 (0x0000000C).

Чтение первого слова данных из памяти DATA поле DW (0x00000014).

Чтение второго слова данных из памяти DATA поле DW (0x00000018).

Пример сообщения ГОСТ Р 52070-2003 формат 3 — передача двух слов данных от одного оконечного устройства к другому оконечному устройству:

Запись первого командного слова с адресом принимающей стороны в поле CW0 памяти DATA (0x00000004).

Запись второго командного слова с адресом передающей стороны в поле CW1 памяти DATA (0x00000008).

Запись регистра CONTROL: BC_MSGTYPE = 2 (тип сообщения ОУ к ОУ), BC_BUS = 0 (по первой шине), BC_START = 1 (начать сообщение).

Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

Запись нулей в регистр STATUS: сброс события.

Чтение ответного слова от ОУ, передававшего данные, поле SW0 (0x0000000C).

Чтение ответного слова от ОУ, принимавшего данные поле SW1 (0x00000010).

Контроллер сохраняет данные, передающего ОУ, и их можно прочитать из памяти DATA поле DW.

Пример сообщения ГОСТ Р 52070-2003 формат 4 — передача команды управления:

Запись команды управления в поле CW0 памяти DATA (0x00000004).

Запись регистра CONTROL: BC_MSGTYPE = 3, BC_BUS = 0 (по первой шине), BC_START = 1 (начать сообщение).

Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

Запись нулей в регистр STATUS: сброс события.

Чтение ответного слова из памяти DATA поле SW0 (0x0000000C).

Пример сообщения ГОСТ Р 52070-2003 формат 5 — передача команды управления и прием слова данных от оконечного устройства:

Запись команды управления в поле CW0 памяти DATA (0x00000004).

Запись регистра CONTROL: BC_MSGTYPE = 4, BC_BUS = 0 (по первой шине), BC_START = 1 (начать сообщение).

Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

Запись нулей в регистр STATUS: сброс события.

Чтение ответного слова из памяти DATA поле SW0 (0x0000000C).

Чтение слова данных из памяти DATA поле DW (0x00000014).

Пример сообщения ГОСТ Р 52070-2003 формат 6 — передача команды управления со словом данных:

Запись команды управления в поле CW0 памяти DATA (0x00000004).

Запись первого слова данных в поле DW память DATA (0x00000014).

Запись регистра CONTROL: BC_MSGTYPE = 5, BC_BUS = 0 (по первой шине), BC_START = 1 (начать сообщение).

Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

Запись нулей в регистр STATUS: сброс события.

Чтение ответного слова из памяти DATA поле SW0 (0x0000000C).

20.6 Режим монитора шины (МШ)

20.6.1 После инициализации контроллера поле MT_MSGINTV (регистр CONTROL) указывает количество выполненных сообщений, после которых будет происходить оповещение. Контроллер записывает первое принятое сообщение в нулевой блок памяти DATA, второе в первый блок памяти DATA и т.д. (смотри память DATA в режиме МШ). После того, как количество принятых сообщений равно значению поля MT_MSGINTV, генерируется событие MSG_OK = 1 (регистр STATUS). Далее пользователь читает слова сообщения из памяти DATA. После записи 32 сообщения в 31 блок памяти DATA контроллер заново начинает записывать нулевой блок. Поэтому пользователь должен успеть прочитать сообщение из нулевого блока перед тем, как контроллер начнет повторную запись этого блока. Для каждого сообщения контроллер генерирует монитор-слово. Формат монитор-слова приведен в таблице 20.27.

Таблица 20.27 – Формат монитор-слова сообщения

Номер	**Имя**	**Описание**
31:13		Не используется.
12:9	TYPE	Формат сообщения. 0: передача данных от КШ к ОУ. 1: передача данных от ОУ к КШ. 2: передача данных от ОУ к ОУ. 3: передача команды управления. 4: передача команды управления и прием слова данных от ОУ. 5: передача команды управления со словом данных оконечному устройству. 6: передача данных (в групповом сообщении) от КШ к ОУ. 7: передача данных (в групповом сообщении) от ОУ к ОУ. 8: передача групповой команды управления. 9: передача групповой команды управления со словом данных.
8	BUS	0: сообщение выполнено по первой шине интерфейса. 1: сообщение выполнено по второй шине интерфейса.
7	ERR_MC	0: нет ошибки. 1: "нестандартная команда управления".
6	ERR_NOWORD	0: нет ошибки. 1: "нет ответа": не принято ответное слово или слово данных в течении времени, определяемого NOWORD (регистр TIME_B).
5	ERR_NOGAP	0: нет ошибки. 1: "нет паузы": ответное слово или новое сообщение принято без паузы, определяемой NOGAP (регистр TIME_B).
4	ERR_SYNC	0: нет ошибки. 1: "ошибка типа слова": должно быть слово данных, а получено командное/ответное слово или наоборот.
3	ERR_PAR	0: нет ошибки. 1: "ошибка бита четности".
2	ERR_M2	0: нет ошибки. 1: "ошибка Манчестер II кодирования".
1	MSG_OK	0: не определено. 1: "сообщение выполнено успешно".
0	MSG_RCV	0: не определено. 1: "сообщение завершено".

Пример приема трех сообщений (ГОСТ Р 52070-2003 формат 1, 2, 3) с оповещением через каждые три сообщения:

–Поле MT_MSGINTV = 3 (оповещение для группы из 3 сообщений). Изменение этого поля разрешается проводить только во время инициализации (сброса) контроллера.

– Ожидание приема группы из трех сообщений (настройка прерывания или периодический опрос регистра STATUS). На шине произошло 3 сообщения: первое ГОСТ Р 52070-2003 формата 1 — передача двух слов данных от контроллера шины к оконечному устройству, второе ГОСТ Р 52070-2003 формата 2 — передача двух слов данных от оконечного устройства к контроллеру шины, третье ГОСТ Р 52070-2003 формата 3 — передача двух слов данных от одного оконечного устройства к другому оконечному устройству. По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

– Запись нулей в регистр STATUS: сброс события.

– У пользователя должен быть индекс сообщений, который будет хранить адрес начала блока откуда читать сообщения, после сброса контроллера он должен быть равен нулю (0x00000000). В начале каждого блока находится монитор-слово, которое хранит параметры сообщения.

– Чтение монитор-слова по индексу = 0x00000000. Из него можно узнать об успешности выполнения MSG_OK = 1 и тип сообщения TYPE = 0 (формат 1).

– Чтение командного слова CW0 (0x0000004). Из него можно узнать количество слов данных.

– Чтение ответного слова поле SW0 (0x000000C).

– Чтение первого слова данных поле DW (0x0000014).

– Чтение второго слова данных поле DW (0x0000018).

– Подготовка к чтению второго сообщения, увеличение индекса на 0x100 (начало второго блока 0x00000100).

– Чтение монитор-слова по индексу=0x00000100. Из него можно узнать об успешности выполнения MSG_OK = 1 и тип сообщения TYPE = 1 (формат 2).

– Чтение командного слова CW0 (0x0000104). Из него можно узнать количество слов данных.

– Чтение ответного слова поле SW0 (0x000010C).

– Чтение первого слова данных поле DW (0x0000114).

– Чтение второго слова данных поле DW (0x0000118).

– Подготовка к чтению третьего сообщения, увеличение индекса на 0x100 (начало третьего блока 0x00000200).

– Чтение монитор-слова по индексу=0x00000200. Из него можно узнать об успешности выполнения MSG_OK = 1 и тип сообщения TYPE = 2 (формат 3).

– Чтение командного слова CW0 (0x0000204). Из него можно узнать количество слов данных.

– Чтение командного слова CW1 (0x0000208).

– Чтение ответного слова оконечного устройства, передававшего данные, поле SW0 (0x000020C).

– Чтение ответного слова оконечного устройства, принимавшего данные, поле SW1 (0x0000210).

– Чтение первого слова данных поле DW (0x0000214).

– Чтение второго слова данных поле DW (0x0000218).

– Подготовка к чтению четвертого сообщения, увеличение индекса на 0x100 (начало четвертого блока 0x00000300).

– Ожидание приема следующей группы из трех сообщений.

Пример приема трех сообщений (ГОСТ Р 52070-2003 формат 4, 5, 6) с оповещением через каждое одно сообщение:

– Поле MT_MSGINTV = 1 (оповещение для 1 сообщения). Изменение этого поля разрешается проводить только во время инициализации контроллера.

– Ожидание приема сообщения (настройка прерывания или периодический опрос регистра STATUS). На шине произошло сообщение: ГОСТ Р 52070-2003 формат 4 —

передача команды управления. По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

– Запись нулей в регистр STATUS: сброс события.

– У пользователя должен быть индекс сообщений, который будет хранить адрес начала блока откуда читать сообщения, после сброса контроллера он должен быть равен нулю (0x00000000). В начале каждого блока находится монитор-слово, которое хранит параметры сообщения.

– Чтение монитор-слова по индексу = 0x00000300 (продолжение предыдущего примера, в котором было уже принято 3 сообщения). Из него можно узнать об успешности выполнения MSG_OK = 1 и тип сообщения TYPE = 3 (формат 4).

– Чтение команды управления CW0 (0x0000304). Из него можно узнать количество слов данных.

– Чтение ответного слова поле SW0 (0x000030C).

– Подготовка к чтению пятого сообщения, увеличение индекса на 0x100 (начало пятого блока 0x00000400).

– Ожидание приема сообщения. На шине произошло сообщение: ГОСТ Р 52070-2003 формат 5 — передача команды управления. По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

– Чтение монитор-слова по индексу = 0x00000400. Из него можно узнать об успешности выполнения MSG_OK = 1 и тип сообщения TYPE = 4 (формат 5).

– Чтение команды управления CW0 (0x0000404).

– Чтение ответного слова поле SW0 (0x000040C).

– Чтение слова данных поле DW (0x0000414).

– Подготовка к чтению шестого сообщения, увеличение индекса на 0x100 (начало шестого блока 0x00000500).

– Ожидание приема сообщения. На шине произошло сообщение: формат 6 — передача команды управления. По завершению в регистре STATUS устанавливается бит MSG_OK = 1.

– Чтение монитор-слова по индексу=0x00000500. Из него можно узнать об успешности выполнения MSG_OK = 1 и тип сообщения TYPE = 5 (формат 6).

– Чтение команды управления CW0 (0x0000504).

– Чтение ответного слова поле SW0 (0x000050C).

– Чтение слова данных поле DW (0x0000514).

– Подготовка к чтению седьмого сообщения, увеличение индекса на 0x100 (начало седьмого блока 0x00000600).

– Ожидание приема следующего сообщения.

20.7 Режим оконечного устройства (ОУ)

20.7.1 После инициализации контроллера пользователь записывает адрес ОУ: поле RT_ADDR регистра CONTROL.

20.7.2 **Прием/передача данных (ГОСТ Р 52070-2003 формат 1, 2, 3, 7, 8)**

20.7.2.1 Для передачи данных пользователь записывает данные в память DATA в требуемые подадреса. Затем, когда контроллер получает сообщение от контроллера шины на передачу данных, он отсылает ответное слово (если не групповое сообщение) и запрошенные данные. По окончании сообщения контроллер выводит в регистре STATUS: "сообщение выполнено успешно" бит MSG_OK = 1 или "сообщение не выполнено" один из битов ошибок ERR_M2, ERR_PAR, ERR_SYNC, ERR_NOGAP, ERR_NOWORD равен 1 и бит MSG_OK = 0.

При приеме данных, когда контроллер получает сообщение от контроллера шины на прием данных, он сохраняет данные в памяти DATA по подадресу, указанному в командном слове, и передает ответное слово (если не групповое сообщение). По окончании сообщения контроллер выводит в регистре STATUS: "сообщение выполнено

успешно" бит MSG_OK = 1 или "сообщение не выполнено" один из битов ошибок ERR_M2, ERR_PAR, ERR_SYNC, ERR_NOGAP, ERR_NOWORD равен 1 и бит MSG_OK = 0.

Также при приеме/передаче данных по окончании сообщения в регистре STATUS выводится:

- RT_MC: признак того, что сообщение не является командой управления.
- RT_SUBADDR: подадрес, участвовавший в обмене.
- RT_WCMC: количество принятых/переданных слов данных.
- RT_TR: направление обмена (прием/передача).
- RT_BROAD: единица, если сообщение было групповым.
- RT_BUS: номер шины, на которой было сообщение.

При обработке сообщений контроллер самостоятельно обрабатывает командные слова, передает ответные слова и передает/принимает слова данных. При обработке групповых сообщений контроллер устанавливает в ответном слове бит "Принята групповая команда" в 1, но не передает ответное слово. При обработке следующего негруппового сообщения ответное слово передается с битом "Принята групповая команда", установленным в ноль. Для чтения ответного слова группового сообщения контроллер шины может использовать команды управления "Передать ответное слово" и “Передать последнюю команду”. Пользователь должен своевременно обновлять данные в подадресах памяти DATA, участвующих в обмене, чтобы не произошло перезаписи или чтения старых данных. Если пользователь не успевает обработать данные имеется возможность установить бит RT_BUSY регистра CONTROL. После этого контроллер начинает передавать каждое ответное слово с битом "Абонент занят" равным единице и переставать принимать/передавать слова данных.

Пример сообщения ГОСТ Р 52070-2003 формат 1 — передача двух слов данных от контроллера шины к оконечному устройству по 27 подадресу:

- Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). При успешном завершении в регистре STATUS устанавливается бит MSG_OK = 1.
- Запись нулей в регистр STATUS: сброс события.
- Декодирование параметров сообщения из регистра STATUS: RT_MC = 0: не команда управления, RT_TR = 0 и RT_SUBBADDR = 27: данные приняты в 27 подадрес, RT_WCMC = 2: количество слов данных равно 2.
- Чтение первого слова данных из памяти DATA по адресу 27*0x80 = 0x00000D80.
- Чтение второго слова данных из памяти DATA по адресу 0x00000D84.

Пример сообщения ГОСТ Р 52070-2003 формат 2 — передача двух слов данных от оконечного устройства к контроллеру шины по 3 подадресу:

- Запись первого слова данных по адресу 3*0x80 = 0x00000180.
- Запись второго слова данных по адресу 0x00000184.
- Ожидание завершения сообщения (настройка прерывания или периодический опрос регистра STATUS). При успешном завершении в регистре STATUS устанавливается бит MSG_OK = 1.
- Запись нулей в регистр STATUS: сброс события.
- Декодирование параметров сообщения из регистра STATUS: RT_MC = 0: не команда управления, RT_TR = 1 и RT_SUBBADDR = 3: данные переданы из 3 подадреса, RT_WCMC = 2: количество слов данных равно 2.

Аналогичные действия, как в примерах выше, совершаются для сообщений ГОСТ Р 52070-2003 форматов 3, 7, 8.

20.7.3 **Обработка команд управления (ГОСТ Р 52070-2003 формат 4, 5, 6, 9, 10)**

20.7.3.1 Кроме приема/передачи данных контроллер осуществляет обработку команд управления. Через регистр INTERRUPT включается/выключается информирование пользователя об успешной обработке команд управления, а именно будет ли контроллер обрабатывать команду "втихую" или будет устанавливать в регистре STATUS бит MSG_OK = 1, и обновлять поля:

- RT_MC: признак того, что сообщение является командой управления.
- RT_SUBADDR: подадрес, участвовавший в приеме/передаче слова данных.
- RT_WCMC: код команды управления.
- RT_TR: направление обмена (прием/передача).
- RT_BROAD: единица, если сообщение было групповым.
- RT_BUS: номер шины, на которой было сообщение.

Команда **b_00000**: принять управление интерфейсом. Используется в устройствах, которые могут переходить в режим контроллера шины и брать управление шиной на себя. Если поддерживается то, пользователь устанавливает бит RT_DYNBC = 1 регистра CONTROL. В ответ на команду контроллер передает ответное слово со значением бита "Принято управление интерфейсом", определяемое битом RT_DYNBC. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_DYNBC (регистр INTERRUPT) равно 1.

Команда **b_00001**: синхронизация. Используется для сообщения о каком-либо условленном событии. Для негрупповой команды контроллер передает ответное слово. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SYNC (регистр INTERRUPT) равно 1.

Команда **b_00010**: передать ответное слово. Команда не изменяет биты ответного слова, таким образом ответное слово передается такое же, как и в предыдущем сообщении. Может использоваться контроллером шины для анализа ошибок. В ответ на команду контроллер передает ответное слово последнего сообщения, не считая данной команды. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SENDSW (регистр INTERRUPT) равно 1.

Команда **b_00011**: начать самоконтроль оконечного устройства. Используется для устройств, в которых есть механизм самотестирования. Для негрупповой команды контроллер передает ответное слово. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_INITST (регистр INTERRUPT) равно 1.

Команда **b_00100**: блокировать передатчик. Используется, когда в системе имеется две шины и позволяет заблокировать передатчик на другой шине. После приема команды контроллер блокирует передатчик шины альтернативной той, по которой команда была принята. Для негрупповой команды контроллер передает ответное слово. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_ENCOFF (регистр INTERRUPT) равно 1.

Команда **b_00101**: разблокировать передатчик. Используется, когда в системе имеется две шины и позволяет разблокировать передатчик на другой шине. Контроллер разблокирует передатчик шины альтернативной той, по которой команда была принята. Для негрупповой команды контроллер передает ответное слово. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_ENCON (регистр INTERRUPT) равно 1.

Команда **b_00110**: блокировать признак неисправности оконечного устройства. Используется для того, чтобы предотвратить повторные прерывания и анализ ошибки, когда ошибка уже обработана контроллером шины и система переконфигурирована. (Пользователь имеет возможность установить бит RT_TERMF регистра CONTROL, после чего контроллер начинает отвечать ответными словами с установленным в единицу битом "неисправность ОУ"). Контроллер маскирует бит "Неисправность ОУ" и для негрупповой

команды передает ответное слово с битом "Неисправность ОУ", сброшенным в ноль. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_MASKON (регистр INTERRUPT) равно 1.

Команда **b_00111**: разблокировать признак неисправности оконечного устройства. Используется, чтобы демаскировать в ответном слове ранее замаскированный бит "Неисправность ОУ". Контроллер демаскирует бит "Неисправность ОУ" и для негрупповой команды передает ответное слово со значением бита "Неисправность ОУ" равным биту RT_TERMF регистра CONTROL. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_OVMSKOFF (регистр INTERRUPT) равно 1.

Команда **b_01000**: установить оконечное устройство в исходное состояние. Для негрупповой команды контроллер передает ответное слово. Затем производится сброс контроллера, кроме всех адресуемых регистров контроллера. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_RESET (регистр INTERRUPT) равно 1.

Команда **b_10000**: передать векторное слово. Используется вместе с битом ответного слова "запрос на обслуживание". Если оконечное устройство имеет функции обслуживания, то с помощью бита RT_SERVREQ регистра CONTROL оно может установить бит ответного слова "запрос на обслуживание". После этого контроллер в каждом сообщении держит бит "запрос на обслуживание" в единице. Далее контроллер шины может запросить векторное слово, которое кодирует требуемое обслуживание. При запросе контроллер читает векторное слово из поля VECW памяти DATA. Затем контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SENDVEC (регистр INTERRUPT) равно 1.

Команда **b_10001**: синхронизация со словом данных. Используется вместе со словом данных для сообщения о каком-либо условленном событии. Значение битов слова данных может быть любым, например, это может быть время системных часов. После приема слова данных контроллер помещает его в поле SYNCW памяти DATA в подадрес, определяемый соответствующим полем команды ("Подадрес/Режим управления"). Для негрупповой команды контроллер передает ответное слово. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SYNCDW (регистр INTERRUPT) равно 1.

Команда **b_10010**: передать последнюю команду. Используется как механизм восстановления системы после ошибок: оконечное устройство должно передать ответное слово и командное слово последнего сообщения, не считая этого. В ответ на команду контроллер передает ответное слово и слово данных, содержащее командное слово последнего сообщения. Если сообщений не было, то контроллер передает нули и в ответном слове, и в слове данных. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SENDCW (регистр INTERRUPT) равно 1.

Команда **b_10011**: передать слово встроенной системы контроля оконечного устройства. Используется как расширенная информация об ошибках оконечного устройства. В ответ на команду контроллер передает ответное слово и слово данных, содержащее слово встроенной системы контроля. Слово встроенной системы контроля контроллер читает из поля BITW памяти DATA из подадреса, определяемого соответствующим полем команды ("Подадрес/Режим управления"). Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SENDBITW (регистр INTERRUPT) равно 1.

Команда **b_10100**: блокировать передатчик с номером i. Используется, когда в системе имеется количество шин больше двух и позволяет заблокировать передатчик на любой другой шине. Контроллер записывает принятое слово данных в поле TROFFW памяти DATA. Для негрупповой команды контроллер передает ответное слово. Далее

контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SENCOFF (регистр INTERRUPT) равно 1.

Команда **b_10101**: разблокировать передатчик с номером i. Используется, когда в системе имеется количество шин больше двух и позволяет заблокировать передатчик на любой другой шине. Контроллер записывает принятое слово данных в поле TROFFW памяти DATA. Для негрупповой команды контроллер передает ответное слово. Далее контроллер выставляет бит MSG_OK = 1 и обновляет соответствующие поля регистра STATUS, если EVTMC_SENCON (регистр INTERRUPT) равно 1.

Зарезервированные коды команд управления b_01001-b_01111 и b_10110-b_11111. Для негрупповых сообщений контроллер передает ответное слово с битом "Ошибка в сообщении" равным 1. Для групповых сообщений ОУ в ответном слове устанавливает биты "Принята групповая команда" и "Ошибка в сообщении" в 1, но не передает ответное слово.

20.8 Подключение контроллера к шинам интерфейса ГОСТ Р 52070-2003

Блок схема подключения контроллера интерфейса ГОСТ Р 52070-2003 приведена на рисунке 20.6.

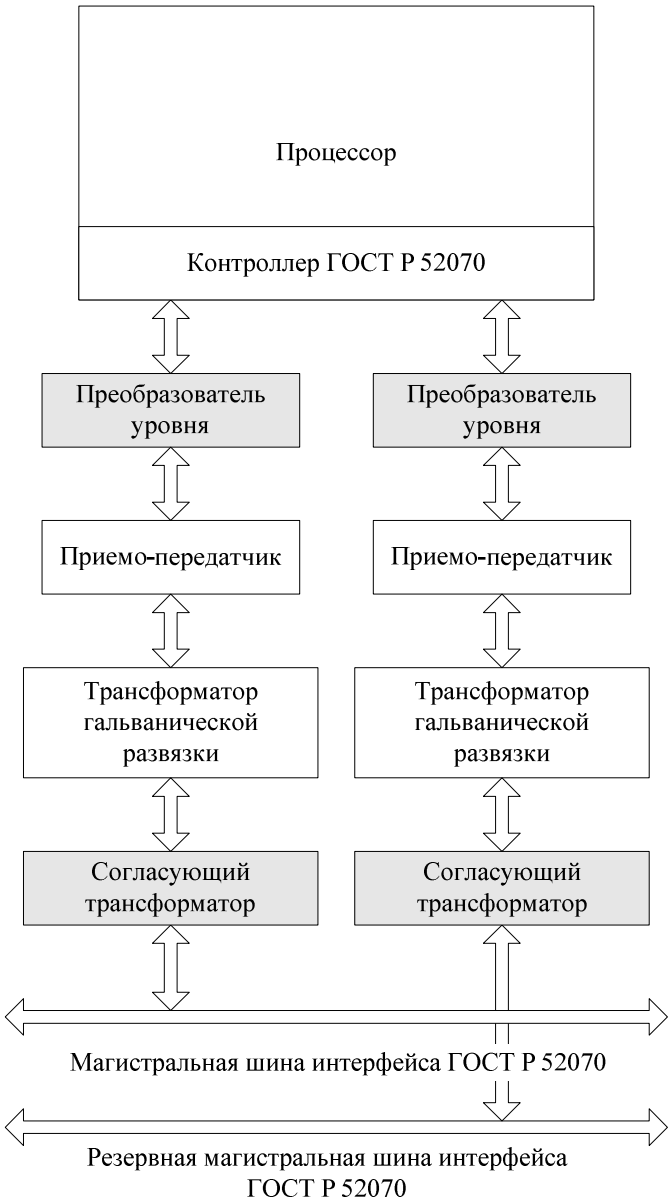

Рисунок 20.6 – Подключение к шинам интерфейса ГОСТ Р52070-2003.

Приемо-передатчик вместе с трансформаторами при передаче битов на магистральную шину преобразует цифровые уровни сигналов контроллера в уровни,

требуемые интерфейсом ГОСТ Р 52070-2003. И наоборот при приеме битов из магистральной шины происходит преобразование уровней интерфейса ГОСТ Р 52070-2003 в цифровые уровни контроллера. Дополнительно могут использоваться микросхемы преобразования цифровых уровней, если приемо-передатчик и контроллер имеют разные уровни напряжения питания.

В зависимости от используемых приемо-передатчиков пользователь настраивает логические состояния выходов контроллера с помощью битов OUT_POLAR, EN_POLAR регистра CONTROL.

На рисунке 20.7 показана временная диаграмма состояния выходов, когда OUT_POLAR = 0: в состоянии бездействия прямой/инверсный выходы будут находится в 0, и EN_POLAR = 0: в состоянии бездействия выход включения передачи будет находится в 0.

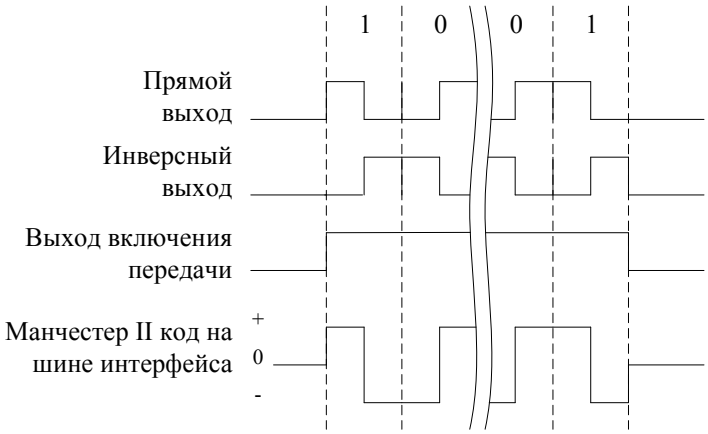

Рисунок 20.7 – Временная диаграмма состояния выходов при OUT_POLAR=0, EN_POLAR=0.

На рисунке 20.8 показана временная диаграмма состояния выходов, когда OUT_POLAR=1: в состоянии бездействия прямой/инверсный выходы будут находится в 1, и EN_POLAR = 1: в состоянии бездействия выход включения передачи будет находится в 1.

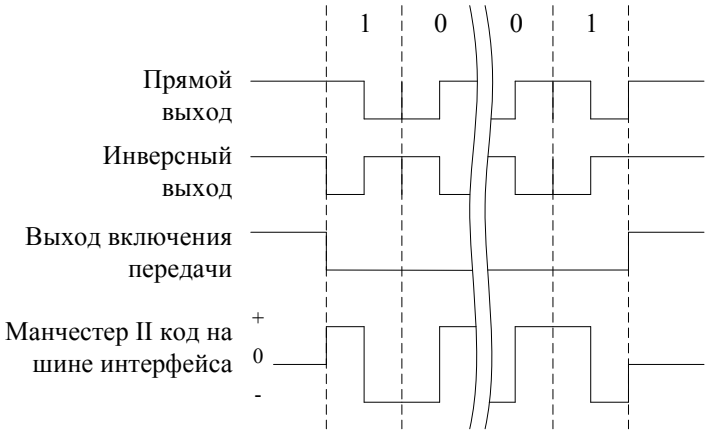

Рисунок 20.8 – Временная диаграмма состояния выходов при OUT_POLAR=1, EN_POLAR=1.

21 Сторожевой таймер

21.1 В состав микроконтроллера входит сторожевой таймер (далее WDT).

WDT построен на основе 32-разрядного счетчика обратного счета, который инициализируется значением из регистра RELOAD. В зависимости от запрограммированного значения блок WDT систематически вырабатывает прерывания. Состояние счетчика уменьшается на единицу по каждому такту системной частоты.

WDT контролирует сигнал прерывания и по достижению счетчиком нулевого значения устанавливает сигнал сброса микроконтроллера. В следующем такте счетчик перезагружается значением из регистра WDOGLOAD и обратный счет повторяется. Если запрос на прерывание не снят к моменту достижения счетчиком нулевого значения, WDT устанавливает сигнал сброса повторно.

WDT служит для сброса системы при возниконовении программных сбоев, обеспечивая тем самым возврат системы в известное состояние. В зависимости от требований WDT может быть включен либо выключен.

На рисунке показан алгоритм работы сторожевого таймера.

Рисунок 21.1 – Алгоритм работы сторожевого таймера

21.2 Описание регистров блока WDT

Таблица 21.1 – Базовые адреса и смещения регистров WDT

Базовый адрес	**Название**	**Описание**
0x4001_C000	WDT	Сторожевой таймер
Смещение		
0x000	WDOGLOAD	Регистр WDT-> WDOGLOAD загрузки
0x004	WDOGVALUE	Регистр WDT-> WDOGVALUE значения
0x008	WDOGCONTROL	Регистр WDT-> WDOGCONTROL управления
0x00C	WDOGINTCLR	Регистр WDT->WDOGINTCLR очистки прерываний
0x010	WDOGRIS	Регистр WDT->WDOGRIS состояния необработанного состояни прерываний
0x014	WDOGMIS	Регистр WDT->WDOGMIS состояния прерывания
0xC00	WDOGLOCK	Регистр WDT->WDOGLOCK блокировки

21.2.1 **WDT->WDOGLOAD**

Таблица 21.2 – Регистр WDT->WDOGLOAD

Номер	31…0
Доступ	R/W
Сброс	0xFFFFFFFF
	WDOGLOAD

Таблица 21.3 – Описание бит регистра WDT->WDOGLOAD

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	WDOGLOAD	Значение, с которого начинается обратный счет. Всякий раз, когда этот регистр перезаписывается, счетчик незамедлительно перезапускает процесс обратного счета с нового значения. Минимальное допустимое значение для WDOGLOAD равно 1.

21.2.2 **WDT->WDOGVALUE**

Таблица 21.4 – Регистр WDT->WDOGVALUE

Номер	31…0
Доступ	R
Сброс	0xFFFFFFFF
	WDOGVALUE

Таблица 21.5 – Описание бит регистра WDT->WDOGVALUE

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	WDOGVALUE	Текущее состояние счетчика обратного счета

21.2.3 **WDT->WDOGCONTROL**

Таблица 21.6 – Регистр WDT->WDOGCONTROL

Номер	31…2	1	0
Доступ	U	R/W	R/W
Сброс	0	0	0
	-	**RES_EN**	**INTEN**

Таблица 21.7 – Описание бит регистра WDT->WDOGCONTROL

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…2	-	Зарезервировано
1	RES_EN	Разрешение установки сигнала сброса WDOGRES **0** – установка сброса запрещена **1** – установка сброса разрешена
0	INTEN	Разрешение генерации прерывания WDOGINT **0** – счет и генерация прерывания WDOGINT запрещены **1** – счет и генерация прерывания WDOGINT разрешены

21.2.4 **WDT->WDOGINTCLR**

Таблица 21.8 – Регистр WDT->WDOGINTCLR

Номер	31…0
Доступ	W
Сброс	-
	WDOGINTCLR

Таблица 21.9 – Описание бит регистра WDT->WDOGINTCLR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	WDOGINTCLR	Запись любого значения в регистр снимает запрос прерывания и перезагружает счетчик значением из WDT->WDOGLOAD

21.2.5 **WDT-> WDOGRIS**

Таблица 21.10 – Регистр WDT->WDOGRIS

Номер	31…1	0
Доступ	U	R
Сброс	0	0
	-	**WDOGRIS**

Таблица 21.11 – Описание бит регистра WDT->WDOGRIS

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…1	-	Зарезервировано
0	WDOGRIS	Статус немаскируемого прерывания от счетчика.

21.2.6 **WDT->WDOGMIS**

Таблица 21.12 – Регистр WDT->WDOGMIS

Номер	31…1	0
Доступ	U	R
Сброс	0	0
	-	**WDOGMIS**

Таблица 21.13 – Описание бит регистра WDT->WDOGMIS

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…1	-	Зарезервировано
0	WDOGMIS	Статус маскируемого прерывания от счетчика. Представляет собой результат сложения по схеме «логического И» бита состояния немаскируемого прерывания из регистра WDT->WDOGRIS с битом разрешения прерывания из регистра WDT->WDOGCONTROL

21.2.7 **WDT->WDOGLOCK**

Таблица 21.14 – Регистр WDT->WDOGLOCK

Номер	31…1	0
Доступ	W	W
Сброс	0	0
	WRENREG	**WRENREGSTATE**

Таблица 21.15 – Описание бит регистра WDT->WDOGLOCK

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…1	WRENREG	Разрешение записи в регистры контроллера WDT. Запись значения 0x1ACCE551 разрешает запись во все остальные регистры контроллера WDT. Запись любого другого значения запрещает запись во все остальные регистры контроллера WDT.
0	WRENREGSTATE	Состояние разрешения записи в регистры **0** – запись во все остальные регистры контроллера WDT разрешена **1** – запись во все остальные регистры контроллера WDT запрещена

22 Контроллер ПДП

22.1 В составе микроконтроллера имеется контроллер прямого доступа в память (контроллер DMA). Контроллер DMA имеет следующие особенности:

– 32 канала DMA;

– каждый канал DMA имеет свои сигналы управления передачей данных;

– каждый канал DMA имеет программируемый уровень приоритета;

– каждый уровень приоритета обрабатывается, исходя из уровня приоритета, определяемого номером канала DMA;

– поддержка различного типа передачи данных: память – память, память – периферия, периферия – память;

– поддержка различных типов DMA циклов;

– поддержка передачи данных различной разрядности;

– каждому каналу DMA доступна первичная и альтернативная структура управляющих данных канала;

– все управляющие данные канала хранятся в системной памяти;

– разрядность данных приемника равна разрядности данных передатчика;

– количество передач в одном цикле DMA может программироваться от 1 до 1024;

– инкремент адреса передачи может быть больше чем разрядность данных.

22.2 Термины и определения

22.2.1 В описании контроллера DMA применяются следующие термины.

Таблица 22.1 – Термины и определения

Термин	Определение
Альтернативная	Альтернативная структура управляющих данных канала. Изменить тип структуры данных можно путем установки соответствующего регистра (см. раздел «Структура управляющих данных канала»)
C	Идентификатор номера канала прямого доступа. Например: C=1 - канал DMA 1 C=23 - канал DMA 23
Канал	Возможны конфигурации контроллера с числом каналов до 32.
Управляющие данные канала	Структура данных находится в системной памяти. Эту структуру данных можно программировать так, чтобы контроллер выполнял передачу данных по каналу DMA в желаемом режиме. Контроллер должен иметь доступ к области системной памяти, где находится эта информация. Примечание: Любое упоминание в документе структуры данных означает управляющие данные канала.
Цикл DMA	Все передачи DMA, которые контроллер должен выполнить для передачи N пакетов данных.
Передача DMA	Пересылка одного байта, полуслова или слова. Общее количество передач DMA, которые контроллер выполняет для канала.
Пинг-понг	Режим работы для выбранного канала, при котором контроллер получает начальный запрос и затем выполняет цикл DMA, используя первичную или альтернативную структуру данных. После завершения этого цикла DMA контроллер начинает выполнять новый цикл DMA, используя другую структуру данных. Контроллер сигнализирует об окончании каждого цикла DMA, позволяя главному процессору перенастраивать неактивную структуру данных. Контроллер продолжает переключаться от первичной к альтернативной структуре данных и обратно до тех пор, пока он не прочитает «неправильную» структуру данных или пока он не завершит цикл без переключения к другой структуре

Продолжение таблицы 22.1

Термин	Определение
Первичная	Первичная структура управляющих данных канала. Контроллер использует эту структуру данных, если соответствующий разряд в регистре chnl_pri_alt_set установлен в 0.
R	Степень числа 2, устанавливающее число передач DMA, которые могут произойти перед сменой арбитража. Количество передач DMA программируется в диапазоне от 1 до 1024 двоичными шагами от 2 в степени 0 до 2 в степени 10.
Исполнение с изменением конфигурации	Режим работы для выбранного канала, при котором контроллер получает запрос от периферии и выполняет 4 DMA передачи, используя первичную структуру управляющих данных, которые настраивают альтернативную структуру управляющих данных. После чего контроллер начинает цикл DMA, используя альтернативную структуру данных. После того, как цикл закончится и если периферия устанавливает новый запрос на обслуживание, контроллер выполняет снова 4 DMA передачи, используя первичную структуру управляющих данных, которые опять перенастраивают альтернативную структуру управляющих данных. После чего контроллер начинает цикл DMA, используя альтернативную структуру данных. Контроллер будет продолжать работать вышеописанным способом до тех пор, пока не прочитает неправильную структуру данных или процессор не установит альтернативную структуру данных для обычного цикла. Контроллер устанавливает флаг dma_done, если окончание подобного режима работы происходит после выполнения обычного цикла

22.3 Функциональное описание

22.3.1 Структурная схема контроллера показана на рисунке 22.1.

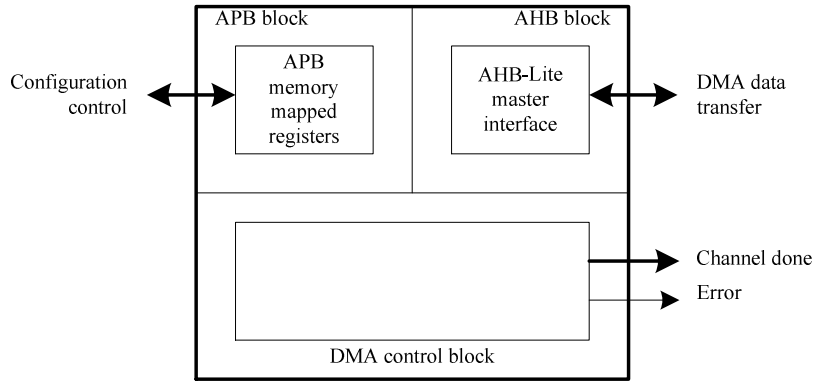

Рисунок 22.1 – Структурная схема контроллера DMA

Контроллер состоит из следующих основных функциональных блоков:

– блок, подключенный к шине APB;

– блок, подключенный к шине AHB;

– управляющий блок DMA.

22.3.2 **Распределение каналов**

22.3.2.1 Каналы не имеют аппаратных источников запросов. Все 32 канала контроллера DMA являются программными.

22.3.3 **APB интерфейс**

22.3.3.1 Блок содержит набор регистров, позволяющих настраивать контроллер, используя ведомый APB интерфейс. Регистры занимают адресное пространство емкостью 4 кбайт.

22.3.4 **AHB интерфейс**

22.3.4.1 Контроллер содержит один блок типа «ведущий» шины DMA Bus, который позволяет, используя 32-разрядную шину, передавать данные от источника к приемнику. Источник и приемник являются ведомыми шины AHB.

22.3.4.2 Типы передач. Контроллер интерфейса не поддерживает пакетные передачи. Контроллер выполняет одиночные передач. Отсутствие возможности осуществлять пакетные передачи оказывает минимальное влияние на производительность системы, так как пакетные передачи более эффективны в одноуровневых системах с шиной AHB, где блоки должны «захватывать» шину или обращаться к внешней памяти. В тоже время контроллер DMA предназначен для использования в многоуровневых системах с шиной AHB, включающих встроенную память.

22.3.4.3 Разрядность передач данных. Контроллер интерфейса предоставляет возможность осуществлять передачу 8, 16 и 32 разрядных данных.

Контроллер всегда использует передачи 32-х разрядными данными при обращении к управляющим данным канала. Необходимо устанавливать разрядность данных источника, соответствующую разрядности данных приемника.

22.3.4.4 Управление защитой данных. Контроллер позволяет устанавливать режимы защиты данных протокола AHB-Lite, определяемые шиной HPROT[3:1]. Возможен выбор следующих режимов защиты:

– кэширование;

– буферизация;

– привилегированный.

Для каждого цикла DMA возможен выбор режимов защиты данных передач источника и приемника. Более подробно это описано в разделе «Настройка управляющих данных».

Для каждого канала DMA также возможен выбор режима защиты данных. Более подробно это описано в разделе Управление DMA.

22.3.4.5 Инкремент адреса. Контроллер позволяет управлять инкрементом адреса при чтении данных из источника и при записи данных в приемник. Инкремент адреса зависит от разрядности передаваемых данных. В таблице 22.2 перечислены возможные комбинации.

Таблица 22.2 – Инкремент адреса

Разрядность данных	**Величина инкремента**
8	Байт, полуслово, слово
16	Полуслово, слово
32	Слово

Минимальная величина инкремента адреса всегда соответствует разрядности передаваемых данных. Максимальная величина инкремента адреса, осуществляемая контроллером, одно слово.

Более подробно о настройке инкремента адреса написано в разделе Настройка управляющих данных. Этот раздел описывает разряды управления величиной инкремента адреса в управляющих данных канала.

Примечание - если необходимо оставлять адрес неизменным при чтении или записи данных, для примера, при работе с FIFO, можно соответствующим образом настроить контроллер на работу с фиксированным адресом (см. раздел «Структура управляющих данных канала»).

22.3.5 **Управляющий блок DMA**

22.3.5.1 Этот блок содержит схему управления, позволяющую реализовать следующие функции:

– осуществление арбитража поступающих запросов;

– индикацию активного канала;

– индикацию завершения обмена по каналу;

– индикацию состояния ошибки обмена по шине DMA Bus;

– разрешение медленным устройствам приостанавливать исполнение цикла DMA;

−ожидание запроса на очистку до завершения цикла DMA;

−осуществление одиночных или множественных передач DMA для каждого запроса;

−осуществление следующих типов DMA передач: память – память, память – периферия, периферия – память.

22.3.6 **Управление DMA контроллером**

22.3.6.1 Правила арбитража DMA. Контроллер имеет возможность настройки момента арбитража при передачах DMA. Эта возможность позволяет уменьшить время отклика при обслуживании каналов с высоким приоритетом.

Контроллер имеет 4 разряда, которые определяют количество транзакций по шине AHB до повторения арбитража. Эти разряды задают степень R числа 2; изменение R напрямую устанавливает периодичность арбитража как 2 в степени R. Для примера, если R равно 4, то арбитраж будет проводиться через каждые 16 передач DMA.

Таблица 22.3 показывает возможную периодичность арбитража.

Таблица 22.3 – Периодичность арбитража в единицах передач по шине AHB

Значение R	**Периодичность арбитража каждые x передач DMA**
b0000	1
b0001	2
b0010	4
b0011	8
b0100	16
b0101	32
b0110	64
b0111	128
b1000	256
b1001	512
b1010-b1111	1024

Примечание: Необходимо с осторожностью устанавливать большие значения R для низкоприоритетных каналов, так как это может привести к невозможности обслуживать запросы по высокоприоритетным каналам.

При N > 2R (N- номер передачи) и если результат деления 2^R на N не целое число, контроллер всегда выполняет последовательность из 2^R передач до тех пор, пока не станет верным $N<2^R$. Контроллер выполняет оставшиеся N передач в конце цикла DMA.

Разряды степени R числа 2 находятся в структуре управляющих данных канала. Местонахождение этих разрядов описано в разделе «Управляющие данные канала».

22.3.6.2 Приоритет. При проведении арбитража определяется канал для обслуживания в следующем цикле DMA. На выбор следующего канала влияют:

− номер канала

− уровень приоритета, присвоенного каналу.

Каждому каналу может быть присвоен уровень приоритета по умолчанию (низкий) или высокий уровень приоритета. Присвоение уровня приоритета осуществляется установкой или сбросом разряда chnl_priority_set.

Канал номер 0 имеет высший уровень приоритета и уровень приоритета снижается с увеличением номера канала. Таблица 22.4 показывает уровень приоритета каналов DMA в порядке его уменьшения.

Таблица 22.4 – Уровень приоритета каналов DMA

Уровень приоритета в порядке его уменьшения	**Номер канала**	**Настройка уровеня приоритета**
Наивысший уровень приоритета	0	Высокий
-	1	Высокий
-	2	Высокий
……..	……..	……..
-	30	Высокий
-	31	Высокий
-	0	По умолчанию
-	1	По умолчанию
-	2	По умолчанию
……..	……..	……..
-	30	По умолчанию
Низший уровень приоритета	31	По умолчанию

После окончания цикла DMA контроллер выбирает следующий для обслуживания канал из всех включенных каналов DMA. На рисунке 22.2 показан процесс выбора следующего канала для обслуживания.

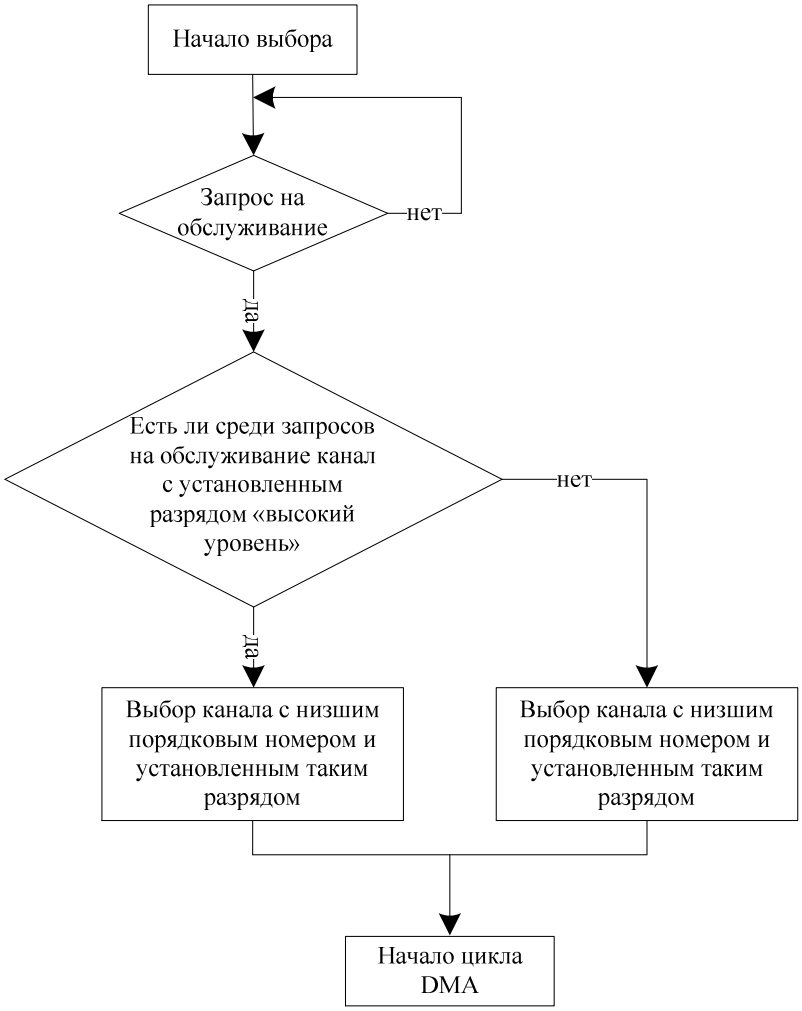

Рисунок 22.2 – Алгоритм выбора следующего канала для обслуживания

22.3.7 **Типы циклов DMA**

22.3.7.1 Разряды cycle_ctrl определяют, как контроллер будет выполнять циклы DMA. Описание значений этих разрядов приведено в таблице 22.5.

Таблица 22.5 – Типы циклов DMA

cycle_ctrl	**Описание**
b000	Структура управляющих данных канала в запрещенном состоянии
b001	Обычный цикл DMA
b010	Авто-запрос
b011	Режим «пинг-понг»
b100	Работа с памятью в режиме «Исполнение с изменением конфигурации» с использованием первичных управляющих данных канала
b101	Работа с памятью в режиме «Исполнение с изменением конфигурации» с использованием альтернативных управляющих данных канала
b110	Работа с периферией в режиме «Исполнение с изменением конфигурации» с использованием первичных управляющих данных канала
b111	Работа с периферией в режиме «Исполнение с изменением конфигурации» с использованием альтернативных управляющих данных канала

Примечание - разряды cycle_ctrl находятся в области памяти, отведенной под channel_cfg.

Для всех типов циклов DMA повторный арбитраж происходит после 2^R передач DMA. Если установить длинный период арбитража на низкоприоритетном канале, то это заблокирует все запросы на обработку от других каналов до тех пор, пока не будут выполнены 2^R передач DMA по данному каналу. Поэтому, устанавливая значение R, необходимо учитывать, что это может привести к повышенному времени отклика на запрос на обработку от высокоприоритетных каналов.

Данный раздел описывает следующие типы циклов DMA:

- недействительный;
- основной;
- авто-запрос;
- «пинг-понг»;
- работа с памятью в режиме «исполнение с изменением конфигурации»;
- работа с периферией в режиме «исполнение с изменением конфигурации».

22.3.7.2 Недействительный. После окончания цикла DMA контроллер устанавливает тип цикла в значение «недействительный» для предотвращения повтора выполненного цикла DMA.

22.3.7.3 Основной. В этом режиме контроллер работает только с основными или альтернативными управляющими данными канала. После того, как разрешена работа канала и контроллер получил запрос на обработку, цикл DMA выглядит следующим образом:

1. Контроллер выполняет 2^R передач. Если число оставшихся передач 0, контроллер переходит к шагу 3.
2. Осуществление арбитража:

- если высокоприоритетный канал выдает запрос на обработку, то контроллер начинает обслуживание этого канала;
- если периферийный блок или программное обеспечение выдает запрос на обработку (повторный запрос на обработку по каналу), то контроллер переходит к шагу 1.

3. Контроллер устанавливает dma_done[C] в состояние 1 на один такт сигнала hclk. Это указывает центральному процессору на завершение цикла DMA.

22.3.7.4 Авто-запрос. Функционируя в данном режиме, контроллер ожидает получения одиночного запроса на обработку для разрешения работы и выполнения цикла DMA. Такая работа позволяет выполнять передачу больших пакетов данных без существенного увеличения времени отклика на обслуживание высокоприоритетных запросов и не требует множественных запросов на обработку от процессора или периферийных блоков.

Контроллер позволяет выбрать для использования первичную или альтернативную структуру управляющих данных канала. После того, как разрешена работа канала и контроллер получил запрос на обработку, цикл DMA выглядит следующим образом:

1. Контроллер выполняет 2^R передач для канала C. Если число оставшихся передач 0, контроллер переходит к шагу 3.
2. Осуществление арбитража:

– если высокоприоритетный канал выдает запрос на обработку, то контроллер начинает обслуживание этого канала;

– если периферийный блок или программное обеспечение выдает запрос на обработку (повторный запрос на обработку по каналу), то контроллер переходит к шагу 1.

3. Контроллер устанавливает dma_done[C] в состояние 1 на один такт сигнала hclk. Это указывает центральному процессору на завершение цикла DMA.

22.3.7.5 Пинг-понг. В данном режиме контроллер выполняет цикл DMA, используя одну из структур управляющих данных, а затем выполняет еще один цикл DMA, используя другую структуру управляющих данных. Контроллер выполняет циклы DMA с переключением структур до тех пор, пока не считает «неправильную» структуру данных или пока процессор не запретит работу канала.

Рисунок демонстрирует пример функционирования контроллера в режиме «пинг-понг».

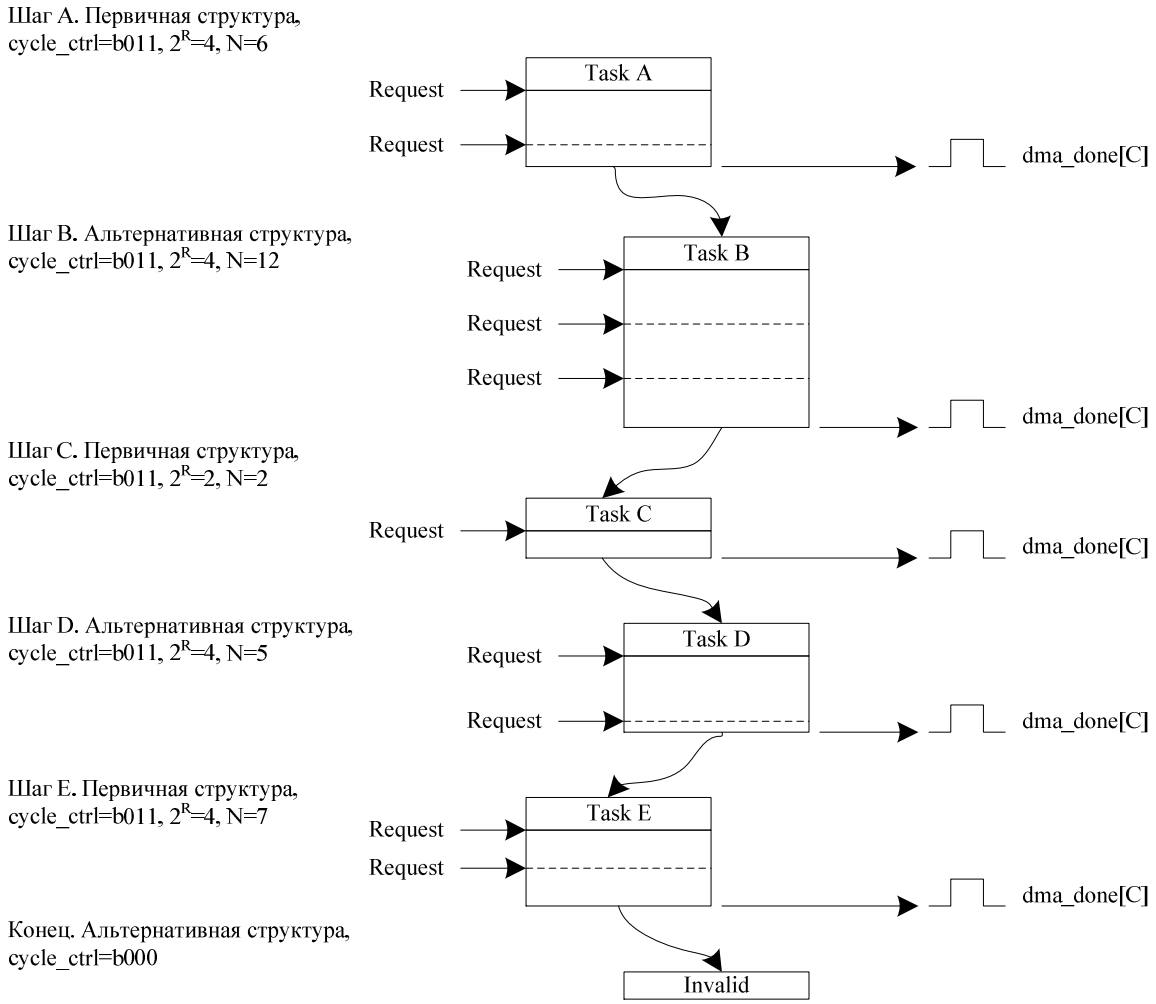

Рисунок 22.3 – Пример функционирования контроллера в режиме «пинг-понг»

Шаг A

1. Процессор устанавливает первичную структуру управляющих данных для шага A.
2. Процессор устанавливает альтернативную структуру управляющих данных для шага B. Это позволит контроллеру переключиться к шагу B незамедлительно после выполнения шага A, при условии, что контроллер не получит запрос на обработку от высокоприоритетного канала.
3. Контроллер получает запрос и выполняет четыре передачи DMA.
4. Контроллер выполняет арбитраж. После получения запроса на обработку от этого же канала, контроллер продолжает цикл в ситуации отсутствия высокоприоритетных запросов.
5. Контроллер выполняет оставшиеся две передачи DMA.
6. Контроллер устанавливает dma_done[C] в состояние 1 на один такт сигнала синхронизации hclk и входит в процедуру арбитража

После выполнения шага A процессор может установить первичные управляющие данные канала для шага C. Это позволит контроллеру переключиться к шагу C незамедлительно после выполнения шага B, при условии, что контроллер не получит запрос на обработку от высокоприоритетного канала.

После получения нового запроса на обработку от канала при условии его наивысшего приоритета исполняется шаг B.

Шаг B

7. Контроллер выполняет четыре передачи DMA.
8. Контроллер выполняет арбитраж. После получения запроса на обработку от этого же канала контроллер продолжает цикл в ситуации отсутствия высокоприоритетных запросов.
9. Контроллер выполняет четыре передачи DMA.
10. Контроллер выполняет арбитраж. После получения запроса на обработку от этого же канала контроллер продолжает цикл в ситуации отсутствия высокоприоритетных запросов.
11. Контроллер выполняет оставшиеся 4 передачи DMA.
12. Контроллер устанавливает dma_done[C] в состояние 1 на один такт сигнала синхронизации hclk и входит в процедуру арбитража

После выполнения шага B процессор может установить альтернативные управляющие данные канала для шага D.

После получения нового запроса на обработку от канала при условии его наивысшего приоритета исполняется шаг C.

Шаг C

13. Контроллер выполняет 2 передачи DMA.
14. Контроллер устанавливает dma_done[C] в состояние 1 на один такт сигнала синхронизации hclk и входит в процедуру арбитража

После выполнения шага C процессор может установить первичные управляющие данные канала для шага E.

После получения нового запроса на обработку от канала при условии его наивысшего приоритета исполняется шаг D.

Шаг D

15. Контроллер выполняет 4 передачи DMA.
16. Контроллер выполняет арбитраж. После получения запроса на обработку от этого же канала контроллер продолжает цикл в ситуации отсутствия высокоприоритетных запросов
17. Контроллер выполняет оставшуюся передачу DMA.
18. Контроллер устанавливает dma_done[C] в состояние 1 на один такт сигнала синхронизации hclk и входит в процедуру арбитража

После получения нового запроса на обработку от канала при условии его наивысшего приоритета исполняется шаг E.

Шаг E

19. Контроллер выполняет 4 передачи DMA.
20. Контроллер выполняет арбитраж. После получения запроса на обработку от этого же канала контроллер продолжает цикл в ситуации отсутствия высокоприоритетных запросов.
21. Контроллер выполняет оставшиеся 3 передачи DMA.
22. Контроллер устанавливает dma_done[C] в состояние 1 на один такт сигнала синхронизации hclk и входит в процедуру арбитража

Если контроллер получит новый запрос на обработку от данного канала и этот запрос будет самым приоритетным, контроллер предпримет попытку выполнения следующего шага. Однако из-за того, что процессор не установил альтернативные управляющие данные, и по окончанию шага D контроллер установил cycle_ctrl в состояние b000, передачи DMA прекращаются.

Примечание: Для прерывания цикла DMA, исполняемого в режиме «пинг-понг», также возможен перевод режима работы контроллера на шаге E в режим «Основной цикл DM» путем установки cycle_ctrl в 3'b001.

22.3.7.6 Режим работы с памятью «исполнение с изменением конфигурации». В данном режиме контроллер, получая начальный запрос на обработку, выполняет 4 передачи DMA, используя первичные управляющие данные. По окончании этих передач контроллер начинает цикл DMA используя альтернативные управляющие данные. Затем контроллер выполняет еще 4 передачи DMA, используя первичные управляющие данные. Контроллер продолжает выполнять циклы ПДА, меняя структуры управляющих данных, пока не произойдет одно из следующих условий:

– процессор переведет контроллер в режим «Основной» во время цикла с альтернативной структурой

– контроллер считает «неправильную» структуру управляющих данных.

Примечание - после исполнения контроллером N передач с использованием первичных управляющих данных он делает эти управляющие данные «неправильными» путем установки cycle_ctrl в b000.

Контроллер устанавливает флаг dma_done[C] в этом режиме работы только тогда, когда передача DMA заканчивается с использованием основного цикла.

В данном режиме контроллер использует первичные управляющие данные для программирования альтернативных управляющих данных. Таюлица 22.6 перечисляет области памяти channel_cfg, как те, которые должны быть определены константами, так и те, значения которых определяются пользователем.

Таблица 22.6 – Channel_cfg для первичной структуры управляющих данных в режиме работы с памятью «исполнение с изменением конфигурации»

№ бита	Обозначение	Значение	Описание
Области с константными значениями			
31…30	dst_inc	b10	Контроллер производит инкремент адреса пословно
29…28	dst_size	b10	Контроллер осуществляет передачу пословно
27…26	src_inc	b10	Контроллер производит инкремент адреса пословно
25…24	src_size	b10	Контроллер осуществляет передачу пословно
17…14	R_power	b0010	Контроллер выполняет 4 передачи DMA
3	next_useburst	b0	Для данного режима этот разряд должен быть равен 0
2…0	cycle_ctrl	b100	Контролер работает в режиме работы с периферией «исполнение с изменением конфигурации»
Области со значениями, определяемыми пользователем			
23…21	dst_prot_ctrl	-	Определяет состояние HPROT при записи данных в приемник
20…18	src_prot_ctrl	-	Определяет состояние HPROT при чтении данных из источника
13…4	n_minus_1	N*)	Настраивает контроллер на выполнение N передач DMA, где N кратно 4

*) - Так как R_power задает значение 4, то необходимо задавать значение N, кратное 4. Число, равное N/4, это количество раз, которое нужно настраивать альтернативные управляющие данные.

Рисунок 22.4 демонстрирует пример функционирования в режиме работы с памятью «Исполнение с изменением конфигурации».

1. Настройка первичных управляющих данных для разрешения копирования A, B, C и D: cycle_ctrl=b100, 2^R=4, N=16.
2. Запись первичных данных в память с использованием структуры, показанной в таблице ниже.

Data for Task A	0x0AE00000	cycle_ctrl=b101, 2^R=4, N=3	0xXXXXXXXX
Data for Task B	0x0BE00000	cycle_ctrl=b101, 2^R=2, N=8	0xXXXXXXXX
Data for Task C	0x0CE00000	cycle_ctrl=b101, 2^R=8, N=5	0xXXXXXXXX
Data for Task D	0x0DE00000	cycle_ctrl=b101, 2^R=4, N=4	0xXXXXXXXX

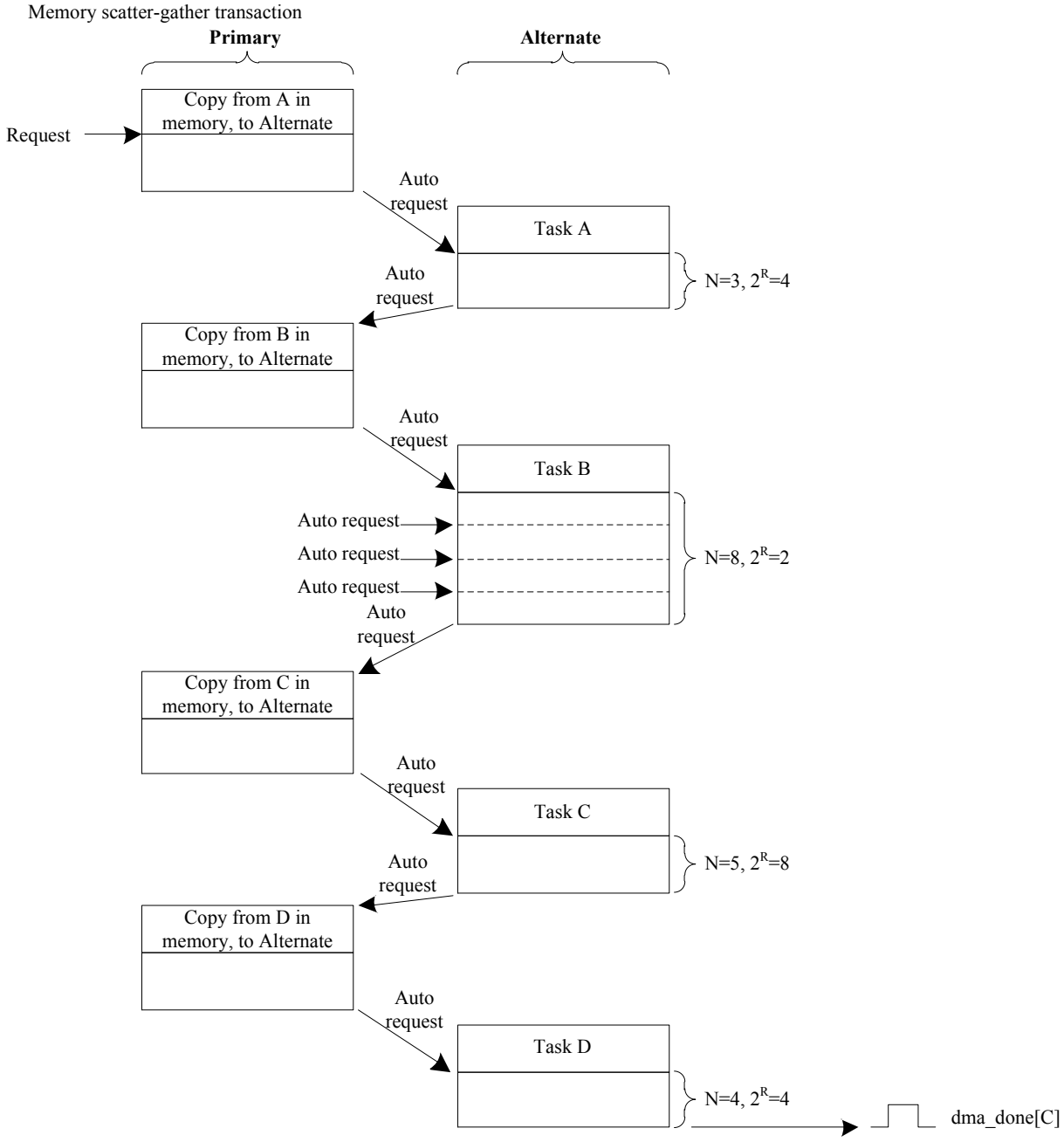

Рисунок 22.4 – Пример работы DMA в режиме «Исполнение с изменением конфигурации»

Инициализация

1. Процессор настраивает первичную структуру управляющих данных для работы в режиме работы с памятью «исполнение с изменением конфигурации» путем установки cycle_ctrl в b100. Так как управляющие данные канала состоят из четырех слов,

необходимо установить 2^R в 4. В этом примере количество задач равно 4 и поэтому N установлен в 16.

2. Процессор записывает управляющие данные для шагов A, B, C, D в область памяти с адресом, указанным в src_data_end_ptr.

3. Процессор разрешает работу канала DMA.

Передачи в данном режиме начинают исполняться при получении контроллером запроса на обслуживание по запросу от процессора. Порядок выполнения следующий:

Первичная, копирование A

1. По получению запроса на обслуживание контроллер выполняет четыре передачи DMA. Эти передачи записывают альтернативную структуру управляющих данных для шага A.

2. Контроллер генерирует автозапрос для канала, после чего проводит процедуру арбитража.

Шаг A

3. Контроллер выполняет шаг A. По окончании контроллер генерирует автозапрос для канала и проводит процедуру арбитража.

Первичная, копирование B

4. Контроллер выполняет четыре передачи DMA. Эти передачи записывают альтернативную структуру управляющих данных для шага B.

5. Контроллер генерирует автозапрос для канала, после чего проводит процедуру арбитража.

Шаг B

6. Контроллер выполняет шаг B. По окончании контроллер генерирует автозапрос для канала и проводит процедуру арбитража.

Первичная, копирование C

7. Контроллер выполняет четыре передачи DMA. Эти передачи записывают альтернативную структуру управляющих данных для шага C.

8. Контроллер генерирует автозапрос для канала, после чего проводит процедуру арбитража.

Шаг C

9. Контроллер выполняет шаг C. По окончании контроллер генерирует автозапрос для канала и проводит процедуру арбитража.

Первичная, копирование D

10. Контроллер выполняет четыре передачи DMA. Эти передачи записывают альтернативную структуру управляющих данных для шага D.

11. Контроллер устанавливает cycle_ctrl первичных управляющих данных в b000 для индикации о том, что эта структура управляющих данных является «неправильной».

12. Контроллер генерирует автозапрос для канала, после чего проводит процедуру арбитража.

Шаг D

13. Контроллер выполняет шаг D, используя основной цикл DMA.

14. Контроллер устанавливает флаг dma_done[C] в состояние 1 на один такт сигнала hclk и входит в процедуру арбитража.

22.3.8 **Индикация ошибок**

22.3.8.1 При получении контроллером по шине AHB ответа об ошибке, он выполняет следующие действия:

– отключает канал, связанный с ошибкой;

– устанавливает флаг dma_err в состояние 1.

После обнаружения процессором флага dma_err процессор должен определить номер канала, который был активен в момент появления ошибки. Это может быть сделано следующим образом:

1. Чтение регистра chnl_enable_set с целью создания списка отключенных каналов.

Когда канал установливает флаг dma_done[], контроллер должен отключить этот канал. Программа, выполняемая процессором, должна всегда хранить данные о каналах, которые недавно установили флаги dma_done[].

2. Процессор должен сравнить список выключенных каналов, полученный на шаге 1, с данными о каналах, которые недавно устанавливали флаги dma_done[]. Канал, по которому отсутствуют данные об установке флага dma_done[] – есть канал, с которым связана ошибка.

22.3.9 **Структура управляющих данных канала**

22.3.9.1 В системной памяти должна быть отведена область для хранения управляющих данных каналов. Системная память должна:

− предоставлять смежную область системной памяти, к которой контроллер и процессор имеют доступ;

− иметь базовый адрес, который целочисленно кратен общему размеру структуры управляющих данных канала.

Рисунок 22.5 показывает область памяти, необходимую контроллеру для структур управляющих данных канала, при использовании всех 32 каналов и опциональной альтернативной структуры управляющих данных.

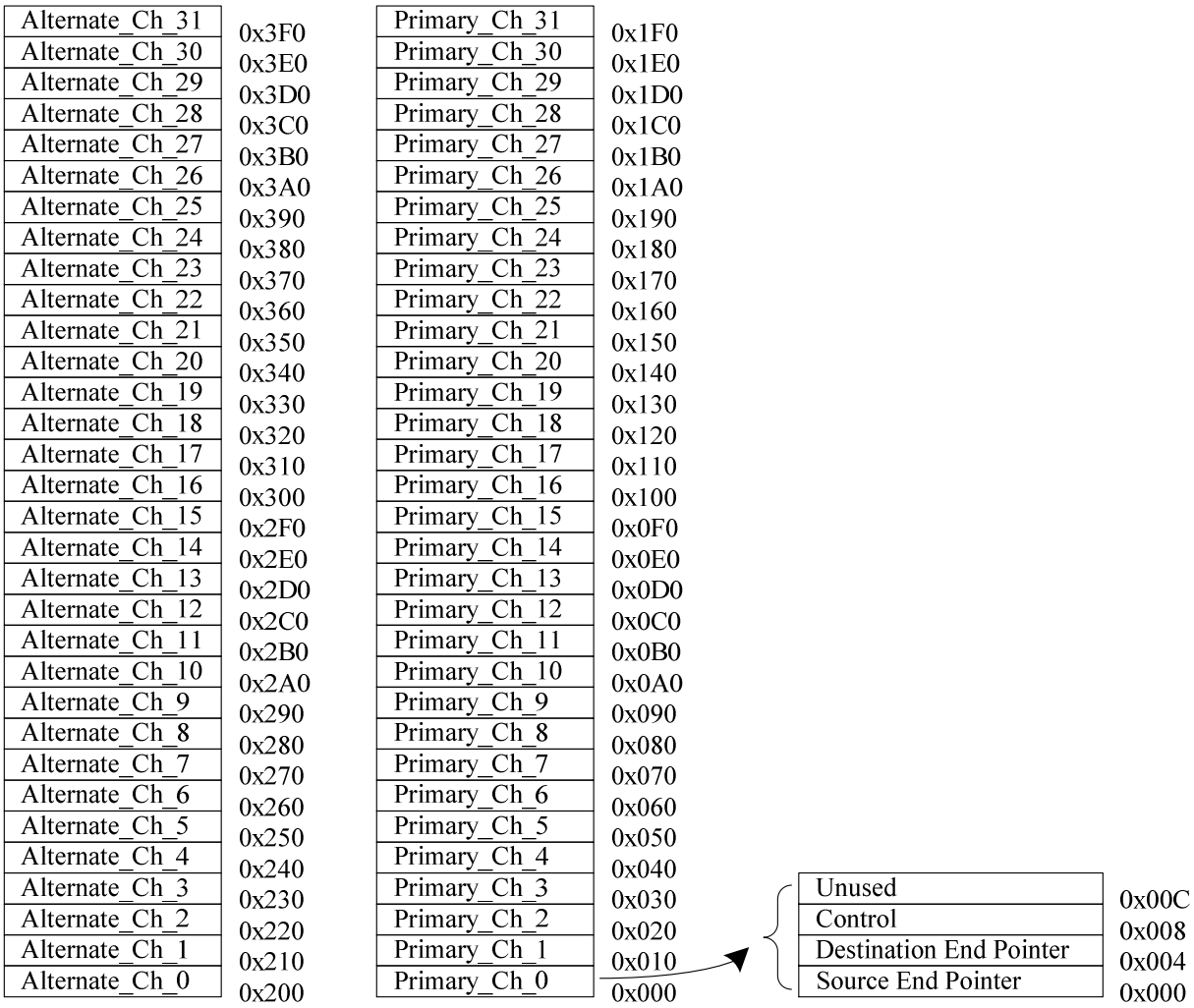

Рисунок 22.5 – Карта памяти для 32-х каналов, включая альтернативную структуру

Обозначения на рисунке:

Destination end pointer - указатель конца данных приемника

Source end pointer - указатель конца данных источника

Control – управление.

Пример, показанный на рисунке, использует 1 Кбайт системной памяти. В этом примере контроллер использует младшие 10 разрядов адреса для доступа ко всем элементам структуры управляющих данных, поэтому базовый адрес структуры должен быть 0xXXXXX000, 0xXXXXX400, 0xXXXXX800 или 0xXXXXXC00.

Базовый адрес для первичной структуры управляющих данных можно установить путем записи соответствующего значения в регистр CTRL_BASE_PTR.

Необходимый размер области системной памяти зависит:

– от количества каналов, используемых в контроллере;

– от того, используется или нет альтернативная структура управляющих данных.

Таблица 22.7 перечисляет разряды адреса, обеспечивающие контроллеру доступ к различным элементам структуры управляющих данных, в зависимости от количества каналов, используемых в контроллере.

Таблица 22.7 – Разряды адреса, соответствующие элементам структуры управляющих данных

	Разряды адреса						
Количество каналов, используемых в контроллере	**[9]**	**[8]**	**[7]**	**[6]**	**[5]**	**[4]**	**[3:0]**
1						A	0x0
2					A	C[0]	0x4
3-4				A	C[1]	C[0]	0x8
5-8			A	C[2]	C[1]	C[0]	
9-16		A	C[3]	C[2]	C[1]	C[0]	
17-32	A	C[4]	C[3]	C[2]	C[1]	C[0]	

Где A выбирает одну из структур управляющих данных канала:

– A = 0 выбирает первичную структуру управляющих данных;

– A = 1 выбирает альтернативную структуру управляющих данных.

C[x:0] Выбирает канал DMA.

Address[3:0] выбирает один из управляющих элементов:

– 0x0 выбирает указатель конца данных источника;

– 0x4 выбирает указатель конца данных приемника;

– 0x8 выбирает конфигурацию управляющих данных;

– 0xC контроллер не имеет доступа к этому адресу. Если это необходимо, то возможно разрешить процессору использовать эти адреса в качестве системной памяти.

Примечание: Совсем не обязательно вычислять базовый адрес альтернативной структуры управляющих данных, так как регистр alt_ctrl_base_ptr содержит эту информацию.

Контроллер использует системную память для доступа к двум указателям адреса конца данных и разрядам управления каждого канала. Эти 32-х разрядные области памяти и процедуру вычисления контроллером адреса передачи DMA описывают следующие подразделы:

– указатель конца данных источника;

– указатель конца данных приемника;

– разряды управления;

– вычисление адреса.

22.3.9.1 Указатель конца данных источника. Область памяти под названием src_data_end_ptr содержит указатель на последний адрес месторасположения данных источника.

Таблица 22.8 – Значения разрядов src_data_end_ptr

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…0	src_data_end_ptr	Указатель последнего адреса данных источника

Перед тем, как контроллер выполнит передачу DMA, необходимо определить эту область памяти. Контроллер считывает значение этой области перед началом 2^R передачи DMA.

Примечание - контроллер не имеет доступа по записи в эту область памяти.

22.3.9.2 Указатель конца данных приемника. Область памяти под названием dst_data_end_ptr содержит указатель на последний адрес месторасположения данных приемника.

Таблица 22.9 – Значения разрядов dst_data_end_ptr

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…0	dst_data_end_ptr	Указатель последнего адреса данных приемника

Перед тем, как контроллер выполнит передачу DMA, необходимо определить эту область памяти. Контроллер считывает значение этой области перед началом 2^R передачи DMA.

Примечание - контроллер не имеет доступа по записи в эту область памяти.

22.3.9.3 Контрольные данные. Область памяти под названием channel_cfg обеспечивает управление каждой передачей DMA.

Таблица 22.10 – Название разрядов области памяти channel_cfg

Номер	31…30	29…28	27…26	25…24	23…21	20…18	17…14	13…4	3	2…0
Доступ										
Сброс										
	dst_inc	**dst_size**	**src_inc**	**src_size**	**dst_prot_ctrl**	**src_prot_ctrl**	**R_power**	**n_minus_1**	**next_useburt**	**cycle_ctrl**

Таблица 22.11 – Назначение разрядов channel_cfg

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…30	dst_inc	Шаг инкремента адреса приемника. Шаг инкремента адреса зависит от разрядности данных источника. Разрядность данных источника = байт: b00 = байт; b01 = полуслово (16 разрядов); b10 = слово (32 разряда); b11 = нет инкремента. Адрес остается равным значению области памяти dst_data_end_ptr. Разрядность данных источника = полуслово: b00 = зарезервировано; b01 = полуслово; b10 = слово; b11 = нет инкремента. Адрес остается равным значению области памяти dst_data_end_ptr. Разрядность данных источника = слово: b00 = зарезервировано; b01 = зарезервировано; b10 = слово (32 разряда); b11 = нет инкремента. Адрес остается равным значению области памяти dst_data_end_ptr
29…28	dst_size	Размерность данных приемника Примечание: Значение этого поля должно быть равно значению поля src_size.
27…26	src_inc	Шаг инкремента адреса источника. Шаг инкремента адреса зависит от разрядности данных источника. Разрядность данных источника = байт: b00 = байт; b01 = полуслово; b10 = слово (32 разряда); b11 = нет инкремента. Адрес остается равным значению области памяти src_data_end_ptr. Разрядность данных источника = полуслово: b00 = зарезервировано b01 = полуслово b10 = слово b11 = нет инкремента. Адрес остается равным значению области памяти src_data_end_ptr. Разрядность данных источника = слово: b00 = зарезервировано; b01 = зарезервировано; b10 = слово; b11 = нет инкремента. Адрес остается равным значению области памяти src_data_end_ptr
25…24	src_size	Задает размерность данных источника: b00 = байт; b01 = полуслово (в русском обычно слово); b10 = слово (в русском обычно двойное слово); b11 = зарезервировано

Продолжение таблицы 22.11

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
23…21	dst_prot_ctrl	Задает состояние HPROT[3:1], когда контроллер записывает данные в приемник. Разряд 23 управляет разрядом HPROT[3]: 0 = HPROT[3] в состоянии 0 и доступ не кэшируется; 1 = HPROT[3] в состоянии 1 и доступ кэшируется. Разряд 22 управляет разрядом HPROT[2]: 0 = HPROT[2] в состоянии 0 и доступ не буферизуется; 1 = HPROT[2] в состоянии 1 и доступ буферизуется. Разряд 21 управляет разрядом HPROT[1]: 0 = HPROT[1] в состоянии 0 и доступ непривилегированный; 1 = HPROT[1] в состоянии 1 и доступ привилегированный
20…18	src_prot_ctrl	Задает состояние HPROT[3:1], когда контроллер считывает данные из источника. Разряд 20 управляет разрядом HPROT[3]: 0 = HPROT[3] в состоянии 0 и доступ не кэшируется; 1 = HPROT[3] в состоянии 1 и доступ кэшируется. Разряд 19 управляет разрядом HPROT[2]: 0 = HPROT[2] в состоянии 0 и доступ не буферизуется; 1 = HPROT[2] в состоянии 1 и доступ буферизуется. Разряд 18 управляет разрядом HPROT[1]: 0 = HPROT[1] в состоянии 0 и доступ непривилегированный; 1 = HPROT[1] в состоянии 1 и доступ привилегированный
17…14	R_power	Задает количество передач DMA до выполнения контроллером процедуры арбитража. Возможные значения: b0000 - арбитраж производится после каждой передачи DMA; b0001 - арбитраж производится после 2 передач DMA; b0010 - арбитраж производится после 4 передач DMA; b0011 - арбитраж производится после 8 передач DMA; b0100 - арбитраж производится после 16 передач DMA; b0101 - арбитраж производится после 32 передач DMA; b0110 - арбитраж производится после 64 передач DMA; b0111 - арбитраж производится после 128 передач DMA; b1000 - арбитраж производится после 256 передач DMA; b1001 - арбитраж производится после 512 передач DMA; b1010 - b1111 - арбитраж производится после 1024 передач DMA. Это означает, что арбитраж не производится, так как максимальное количество передач DMA равно 1024
13…4	n_minus_1	Перед выполнением цикла DMA эти разряды указывают общее количество передач DMA, из которых состоит цикл DMA. Необходимо установить эти разряды в значение, соответствующее размеру желаемого цикла DMA. 10-разрядное число плюс 1 задает количество передач DMA. Возможные значения: b0000000000 = 1 передача DMA; b0000000001 = 2 передачи DMA; b0000000010 = 3 передачи DMA; b0000000011 = 4 передачи DMA; b0000000100 = 5 передач DMA; b0000000101 = 6 передач DMA; …. b1111111111 = 1024 передачи DMA. Контроллер обновит это поле перед тем, как произвести процесс арбитража. Это позволяет контроллеру хранить количество оставшихся передач DMA до завершения цикла DMA

Продолжение таблицы 22.11

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
3	–	Зарезервировано
2…0	cycle_ctrl	Режим работы при выполнении цикла DMA: b000 – **Стоп**. Означает, что структура управляющих данных является «неправильной»; b001 – **Основной**. Контроллер должен получить новый запрос для окончания цикла DMA, перед этим он должен выполнить процедуру арбитража; b010 – **Авто-запрос**. Контроллер автоматически осуществляет запрос на обработку по соответствующему каналу в течение процедуры арбитража. Это означает, что начального запроса на обработку достаточно для выполнения цикла DMA; b011 – **Пинг-понг**. Контроллер выполняет цикл DMA использую одну из структур управляющих данных. По окончании выполнения цикла DMA, контроллер выполняет следующий цикл DMA, используя другую структуру. Контроллер сигнализирует об окончании каждого цикла DMA, позволяя процессору перенастраивать неактивную структуру данных. Контроллер продолжает выполнять циклы DMA, до тех пор, пока он не прочитает «неправильную» структуру данных или пока процессор не изменит cycle_ctrl поле в состояние b001 или b 010; b100 – Режим работы с памятью «Исполнение с изменением конфигурации». При работе контроллера в данном режиме значение этого поля в первичной структуре управляющих данных должно быть b100; b101 – Режим работы с памятью «Исполнение с изменением конфигурации». При работе контроллера в данном режиме значение этого поля в альтернативной структуре управляющих данных должно быть b101; b110 – Режим работы с периферией «исполнение с изменением конфигурации». При работе контроллера в данном режиме значение этого поля в первичной структуре управляющих данных должно быть b110; b111 – Режим работы с периферией «исполнение с изменением конфигурации». При работе контроллера в данном режиме значение этого поля в альтернативной структуре управляющих данных должно быть b111

В начале цикла DMA или 2^R передачи DMA контроллер считывает значение channel_cfg из системной памяти. После выполнения 2^R или N передач он сохраняет обновленное значение channel_cfg в системную память.

Контроллер не поддерживает значений dst_size, отличных от значений src_size. Если контроллер обнаруживает неравные значения этих полей, он использует значение src_size в качестве размера данных и приемника, и источника и при ближайшем обновлении поля n_minus_1, он также устанавливает значение поля dst_size, равное src_size.

После выполнения контроллером N передач, контроллер устанавливает значение поля cycle_ctrl в b000, делая тем самым channel_cfg данные «неправильными». Это позволяет избежать повторения выполненной передачи DMA.

22.3.9.4 Вычисление адреса. Для вычисления адреса источника передачи DMA, контроллер выполняет сдвиг влево значения n_minus_1 на количество разрядов, соответствующее полю src_inc, и затем вычитает получившееся значение от значения указателя адреса конца данных источника. Подобным образом вычисляется адрес приемника передачи DMA, контроллер выполняет сдвиг влево значения n_minus_1 на количество разрядов, соответствующее полю dst_inc, и затем вычитает получившееся значение от значения указателя адреса конца данных приемника.

В зависимости от значения полей src_inc и dst_inc вычисления адресов приемника и источника выполняются по следующим уравнениям:

src_inc=b00 and dst_inc=b00
- адрес источника = src_data_end_ptr - n_minus_1
- адрес приемника = dst_data_end_ptr - n_minus_1.

src_inc=b01 and dst_inc=b01
- адрес источника = src_data_end_ptr – (n_minus_1<<1)
- адрес приемника = dst_data_end_ptr – (n_minus_1<<1).

src_inc=b01 and dst_inc=b10
- адрес источника = src_data_end_ptr – (n_minus_1<<2)
- адрес приемника = dst_data_end_ptr – (n_minus_1<<2).

src_inc=b11 and dst_inc=b11
- адрес источника = src_data_end_ptr
- адрес приемника = dst_data_end_ptr.

В таблице 22.12 перечислены адреса приемника цикла DMA для 6 слов.

Таблица 22.12 – Цикл DMA для 6 слов с пословным инкрементом

Начальные значения channel_cfg перед циклом DMA				
src_size=b10, dst_inc=b10, n_minus_1=b101, cycle_ctrl=1				
DMA передачи	**Указатель конца данных**	**Счетчик**	**Отличие**[*)]	**Адрес**
	0x2AC	5	0x14	0x298
	0x2AC	4	0x10	0x29C
	0x2AC	3	0xC	0x2A0
	0x2AC	2	0x8	0x2A4
	0x2AC	1	0x4	0x2A8
	0x2AC	0	0x0	0x2AC
Конечные значения channel_cfg после цикла DMA				
src_size=b10, dst_inc=b10, n_minus_1=0, cycle_ctrl=0				

[*)] это значение, полученное после сдвига влево значения счетчика на количество разрядов, соответствующее dst_inc.

В таблице 22.13 перечислены адреса приемника для передач DMA 12 байт с использованием «полусловного» инкремента.

Таблица 22.13 – Цикл DMA для 12 байт с «полусловным» инкрементом

Начальные значения channel_cfg перед циклом DMA				
src_size=b00, dst_inc=b01, n_minus_1=b1011, cycle_ctrl=1, R_power=b11				
DMA передачи	**Указатель конца данных**	**Счетчик**	**Отличие*)**	**Адрес**
	0x5E7	11	0x16	0x5D1
	0x5E7	10	0x14	0x5D3
	0x5E7	9	0x12	0x5D5
	0x5E7	8	0x10	0x5D7
	0x5E7	7	0xE	0x5D9
	0x5E7	6	0xC	0x5DB
	0x5E7	5	0xA	0x5DD
	0x5E7	4	0x8	0x5DF
Значения channel_cfg после 2^R передач DMA				
src_size=b00, dst_inc=b01, n_minus_1=b011, cycle_ctrl=1, R_power=b11				
DMA передачи	0x5E7	3	0x6	0x5E1
	0x5E7	2	0x4	0x5E3
	0x5E7	1	0x2	0x5E5
	0x5E7	0	0x0	0x5E7
Конечные значения channel_cfg после цикла DMA				
src_size=b00, dst_inc=b01, n_minus_1=0, cycle_ctrl=0**), R_power=b11				

*) это значение, полученное после сдвига влево значения счетчика на количество разрядов, соответствующее dst_inc.

**) после окончания цикла DMA контроллер делает channel_cfg «неправильным», сбрасывая в 0 поле cycle_ctrl.

22.4 Описание регистров контроллера DMA

Таблица 22.14 – Базовый адрес и смещения регистров контроллера DMA

Базовый адрес	**Название**	**Описание**
0x4001_D000	DMA	Контроллер DMA
Смещение		
0x000	DMA_STATUS	Статусный регистр DMA->DMA_STATUS
0x004	DMA_CFG	Регистр конфигурации DMA->DMA_CFG
0x008	CTRL_BASE_PTR	Регистр базового адреса управляющих данных каналов DMA->CTRL_BASE_PTR
0x00C	ALT_CTRL_BASE_PTR	Регистр базового адреса альтернативных управляющих данных каналов DMA->ALT_CTRL_BASE_PTR
0x014	CHNL_SW_REQUEST	Регистр программного запроса на обработку каналов DMA->CHNL_SW_REQUEST
0x028	CHNL_ENABLE_SET	Регистр установки разрешения каналов DMA->CHNL_ENABLE_SET
0x02C	CHNL_ENABLE_CLR	Регистр сброса разрешения каналов DMA->CHNL_ENABLE_CLR
0x030	CHNL_PRI_ALT_SET	Регистр установки первичной/альтернативной структуры управляющих данных DMA->CHNL_PRI_ALT_SET
0x034	CHNL_PRI_ALT_CLR	Регистр сброса первичной/альтернативной структуры управляющих данных DMA->CHNL_PRI_ALT_CLR
0x038	CHNL_PRIORITY_SET	Регистр установки приоритета каналов DMA->CHNL_PRIORITY_SET
0x03C	CHNL_PRIORITY_CLR	Регистр сброса приоритета каналов DMA->CHNL_PRIORITY_SET
0x040-0x048	Зарезервировано	
0x04C	ERR_CLR	Регистр сброса флага ошибки DMA->ERR_CLR
0x050-0x0DFC	Зарезервировано	

22.4.1 **DMA->DMA_STATUS**

22.4.1.1 Данный регистр имеет доступ только на чтение. При чтении регистр возвращает состояние контроллера. Если контроллер находится в состоянии сброса, то чтение регистра запрещено.

Таблица 22.15 – Регистр DMA->DMA_STATUS

Номер	31…28	27…21	20…16	15…8	7…4	3…1	0
Доступ	R	U	R	U	R	U	R
Сброс	0	0	0	0	0	0	0
	TEST_STATUS	-	**CHNLS_MINUS1**	-	**STATE**	-	**MASTER_ENABLE**

Таблица 22.16 – Описание бит регистра DMA->DMA_STATUS

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…28	TEST_STATUS	При чтении: **0x0** – контроллер не имеет интегрированной схемы тестирования **0x1** – контроллер имеет интегрированную схему тестирования **0x2** – не определено
27…21	-	Зарезервировано
20…16	CHNLS_MINUS1	Количество доступных каналов DMA минус 1.
15…8	-	Зарезервировано
7…4	STATE	Текущее состояние автомата управления контроллера. **b0000** – в покое; **b0001** – чтение управляющих данных канала; **b0010** – чтение указателя конца данных источника; **b0011** – чтение указателя конца данных приемника; **b0100** – чтение данных источника; **b0101** – запись данных в приемник; **b0110** – ожидание запроса на выполнение DMA; **b0111** – запись управляющих данных канала; **b1000** – приостановлен; **b1001** – выполнен; **b1010** – режим работы с периферией «Исполнение с изменением конфигурации»; **b1011-b1111** – не определено
3…1	-	Зарезервировано
0	MASTER_ENABLE	Состояние контроллера: **0** – работа контроллера запрещена **1** – работа контроллера разрешена

22.4.2 **DMA->DMA_CFG**

22.4.2.1 Данный регистр имеет доступ только на запись. Регистр определяет состояние контроллера.

Таблица 22.17 – Регистр DMA->DMA_CFG

Номер	31…8	7…5	4…1	0
Доступ	U	W	U	W
Сброс	0	0	0	0
	-	**CHNL_PROT_CTRL**	**-**	**MASTER_ENABLE**

Таблица 22.18 – Описание бит регистра DMA->DMA_CFG

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…8	-	Зарезервировано
7…5	CHNL_PROT_CTRL	Определяет уровни индикации сигналов HPROT[3:1] защиты шины AHB-Lite: **Разряд [7]** – управляет сигналом HPROT[3], с целью индикации о появлении доступа с кэшированием; **Разряд [6]** – управляет сигналом HPROT[2], с целью индикации о появлении доступа с буферизацией; **Разряд [5]** – управляет сигналом HPROT[1], с целью индикации о появлении привилегированного доступа. Примечание: Если разряд [n] = 1, то соответствующий сигнал HPROT в состоянии 1; Если разряд [n] = 0, то соответствующий сигнал HPROT в состоянии 0
4…1	-	Зарезервировано
0	MASTER_ENABLE	Определяет состояние контроллера: **0** – запрещает работу контроллера; **1** – разрешает работу контроллера

22.4.3 **DMA->CTRL_BASE_PTR**

22.4.3.1 Данный регистр имеет доступ на запись и чтение. Регистр определяет базовый адрес системной памяти для размещения управляющих данных каналов.

Примечание - контроллер не содержит внутреннюю память для хранения управляющих данных каналов.

Размер системной памяти, предназначенной контроллеру, зависит от количества каналов DMA, использующихся контроллером, а также от возможности использования альтернативных управляющих данных каналов. Поэтому количество разрядов регистра, необходимых для задания базового адреса, варьируется и зависит от варианта построения системы.

Таблица 22.19 – Регистр DMA->CTRL_BASE_PTR

Номер	31…10	9…0
Доступ	R/W	U
Сброс	0	0
	CTRL_BASE_PTR	**-**

Таблица 22.20 – Описание бит регистра DMA->CTRL_BASE_PTR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…10	CTRL_BASE_PTR	Указатель на базовый адрес первичной структуры управляющих данных.
9…0	-	Не определено. Записывать 0.

22.4.4 **DMA->ALT_CTRL_BASE_PTR**

22.4.4.1 Данный регистр имеет доступ только на чтение. Регистр возвращает при чтении указатель базового адреса альтернативных управляющих данных каналов. Если контроллер находится в состоянии сброса, то чтение регистра запрещено. Этот регистр позволяет не производить вычисления базового адреса альтернативных управляющих данных каналов.

Таблица 22.21 – Регистр DMA->ALT_CTRL_BASE_PTR

Номер	31…0
Доступ	R
Сброс	0
	ALT_CTRL_BASE_PTR

Таблица 22.22 – Описание бит регистра DMA->ALT_CTRL_BASE_PTR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…10	ALT_CTRL_BASE_PTR	Указатель базового адреса альтернативной структуры управляющих данных.

22.4.5 **DMA->CHNL_SW_REQUEST**

22.4.5.1 Данный регистр имеет доступ только на запись. Регистр позволяет устанавливать программно запрос на выполнение цикла DMA.

Таблица 22.23 – Регистр DMA->CHNL_SW_REQUEST

Номер	31	…	2	1	0
Доступ	W	…	W	W	W
Сброс	0	…	0	0	0
	CHNL_SW_REQUEST for channel[31]	**…**	**CHNL_SW_REQUEST for channel[2]**	**CHNL_SW_REQUEST for channel[1]**	**CHNL_SW_REQUEST for channel[0]**

Таблица 22.24 – Описание бит регистра DMA->CHNL_SW_REQUEST

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…0	CHNL_SW_REQUEST	Устанавливает соответствующий разряд для генерации программного запроса на выполнение цикла DMA по соответствующему каналу DMA. При записи: **Разряд [C]** = 0 означает, что запрос на выполнение цикла DMA по каналу C не будет установлен; **Разряд [C]** =1 означает, что запрос на выполнение цикла DMA по каналу C будет установлен. Запись разряда соответствующего нереализованному каналу, означает, что запрос на выполнение цикла DMA не будет установлен

22.4.6 **DMA->CHNL_ENABLE_SET**

22.4.6.1 Данный регистр имеет доступ на чтение и запись. Регистр разрешает работу каналов DMA. Регистр возвращает при чтении состояние разрешений работы каналов DMA.

Таблица 22.25 – Регистр DMA->CHNL_ENABLE_SET

Номер	31	…	2	1	0
Доступ	W	…	W	W	W
Сброс	0	…	0	0	0
	CHNL_ENABLE_SET for channel[31]	**…**	**CHNL_ENABLE_SET for channel[2]**	**CHNL_ENABLE_SET for channel[1]**	**CHNL_ENABLE_SET for channel[0]**

Таблица 22.26 – Описание бит регистра DMA->CHNL_ENABLE_SET

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…0	CHNL_ENABLE_SET	Разрешает работу каналов DMA и возвращает при чтении состояние этих настроек. При чтении: **Разряд [C]** = 0 означает, что канал DMA C отключен; **Разряд [C]** = 1 означает, что работа канала DMA C разрешена. При записи: **Разряд [C]** = 0 не дает эффекта. Необходимо использовать DMA->CHNL_ENABLE_CLR регистр для отключения канала; **Разряд [C]** = 1 разрешает работу канала DMA C. Запись разряда соответствующего нереализованному каналу не дает никакого эффекта.

22.4.7 **DMA->CHNL_ENABLE_CLR**

22.4.7.1 Данный регистр имеет доступ только на запись. Регистр запрещает работу каналов DMA.

Таблица 22.27 – Регистр DMA->CHNL_ENABLE_CLR

Номер	31	…	2	1	0
Доступ	W	…	W	W	W
Сброс	0	…	0	0	0
	CHNL_ENABLE_CLR for channel[31]	**…**	**CHNL_ENABLE_CLR for channel[2]**	**CHNL_ENABLE_CLR for channel[1]**	**CHNL_ENABLE_CLR for channel[0]**

Таблица 22.28 – Описание бит регистра DMA->CHNL_ENABLE_CLR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	CHNL_ENABLE_CLR	Установка соответствующего разряда запрещает работу соответствующего канала DMA. При записи: **Разряд [C]** = 0 не дает эффекта. Необходимо использовать CHNL_ENABLE_SET регистр для разрешения работы канала; **Разряд [C]** = 1 запрещает работу канала DMA C. Запись разряда, соответствующего нереализованному каналу, не дает никакого эффекта. Примечание: Контроллер может отключить канал DMA, установив соответствующий разряд в следующих случаях: -при завершении цикла DMA; -при чтении из channel_cfg с полем cycle_ctrl установленным в b000; -при появлении ошибки на шине AHB-Lite

22.4.8 **DMA->CHNL_PRI_ALT_SET**

22.4.8.1 Данный регистр имеет доступ на запись и чтение. Регистр разрешает работу канала DMA с использованием альтернативной структуры управляющих данных. Чтение регистра возвращает состояние каналов DMA (какую структуру управляющих данных использует каждый канал DMA).

Таблица 22.29 – Регистр DMA->CHNL_PRI_ALT_SET

Номер	31	…	2	1	0
Доступ	W	…	W	W	W
Сброс	0	…	0	0	0
	CHNL_PRI_ALT_SET for channel[31]	**…**	**CHNL_PRI_ALT_SET for channel[2]**	**CHNL_PRI_ALT_SET for channel[1]**	**CHNL_PRI_ALT_SET for channel[0]**

Таблица 22.30 – Описание бит регистра DMA->CHNL_PRI_ALT_SET

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	CHNL_PRI_ALT_SET	Установка соответствующего разряда подключает использование альтернативных управляющих данных для соответствующего канала DMA, чтение возвращает состояние этих настроек. При чтении: **Разряд [C]** = 0 означает, что канал DMA C использует первичную структуру управляющих данных; **Разряд [C]** = 1 означает, что канал DMA C использует альтернативную структуру управляющих данных. При записи: **Разряд [C]** = 0 не дает эффекта. Необходимо использовать chnl_pri_alt_clr регистр для сброса разряда [C] в 0; **Разряд [C]** = 1 подключает использование альтернативной структуры управляющих данных каналом DMA C. Запись разряда соответствующего нереализованному каналу не дает никакого эффекта. Примечание: Контроллер может переключить значение разряда CHNL_PRI__ALT_SET[C] в следующих случаях: -при завершении 4-х передач DMA указанных в первичной структуре управляющих данных при выполнении цикла DMA в режимах работы с памятью или периферией «исполнение с изменением конфигурации»; -при завершении всех передач DMA указанных в первичной структуре управляющих данных при выполнении цикла DMA в режиме «Пинг-понг»; -при завершении всех передач DMA указанных в альтернативной структуре управляющих данных при выполнении цикла DMA в режимах: -«пинг-понг»; -работа с памятью «Исполнение с изменением конфигурации»; -работа с периферией «Исполнение с изменением конфигурации;

22.4.9 **DMA->CHNL_PRI_ALT_CLR**

22.4.9.1 Данный регистр имеет доступ только на запись. Регистр разрешает работу канала DMA с использованием первичной структуры управляющих данных.

Таблица 22.31 – Регистр DMA->CHNL_PRI_ALT_CLR

Номер	31	…	2	1	0
Доступ	W	…	W	W	W
Сброс	0	…	0	0	0
	CHNL_PRI_ALT_CLR for channel[31]	**…**	**CHNL_PRI_ALT_CLR for channel[2]**	**CHNL_PRI_ALT_CLR for channel[1]**	**CHNL_PRI_ALT_CLR for channel[0]**

Таблица 22.32 – Описание бит регистра DMA->CHNL_PRI_ALT_CLR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	CHNL_PRI_ALT_CLR	Установка соответствующего разряда подключает использование первичных управляющих данных для соответствующего канала DMA. При записи: Разряд [C] = 0 не дает эффекта. Необходимо использовать chnl_pri_alt_set регистр для выбора альтернативных управляющих данных; Разряд [C] = 1 подключает использование первичной структуры управляющих данных каналом DMA C. Запись разряда соответствующего нереализованному каналу не дает никакого эффекта. Примечание: Контроллер может переключить значение разряда chnl_pri_alt_clr[C] в следующих случаях: -при завершении 4-х передач DMA указанных в первичной структуре управляющих данных при выполнении цикла DMA в режимах работы с памятью или периферией «исполнение с изменением конфигурации»; -при завершении всех передач DMA указанных в первичной структуре управляющих данных при выполнении цикла DMA в режиме «пинг-понг»; -при завершении всех передач DMA указанных в альтернативной структуре управляющих данных при выполнении цикла DMA в режимах: -«пинг-понг» -работа с памятью «Исполнение с изменением конфигурации» -работа с периферией «Исполнение с изменением конфигурации»

22.4.10 **DMA->CHNL_PRIORITY_SET**

22.4.10.1 Данный регистр имеет доступ на запись и чтение. Регистр позволяет присвоить высокий приоритет каналу DMA. Чтение регистра возвращает состояние приоритета каналов DMA.

Таблица 22.33 – Регистр DMA->CHNL_PRIORITY_SET

Номер	31	…	2	1	0
Доступ	W	…	W	W	W
Сброс	0	…	0	0	0
	CHNL_PRIORITY_SET for channel[31]	**…**	**CHNL_PRIORITY_SETfor channel[2]**	**CHNL_PRIORITY_SETfor channel[1]**	**CHNL_PRIORITY_SETfor channel[0]**

Таблица 22.34 – Описание бит регистра DMA->CHNL_PRIORITY_SET

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	CHNL_PRIORITY_SET	Установка высокого приоритета каналу DMA, чтение возвращает состояние приоритета каналов DMA. При чтении: **Разряд [C]** = 0 означает, что каналу DMA C присвоен уровень приоритета по умолчанию; **Разряд [C]** = 1 означает, что каналу DMA C присвоен высокий уровень приоритета. При записи: **Разряд [C]** = 0 не дает эффекта. Необходимо использовать chnl_priority_clr регистр для установки каналу C уровня приоритета по умолчанию; **Разряд [C]** = 1 устанавливает каналу DMA C высокий уровень приоритета. Запись разряда соответствующего нереализованному каналу не дает никакого эффекта

22.4.11 **DMA->CHNL_PRIORITY_CLR**

22.4.11.1 Данный регистр имеет доступ только на запись. Регистр позволяет присвоить каналу DMA уровень приоритета по умолчанию.

Таблица 22.35 – Регистр DMA->CHNL_PRIORITY_CLR

Номер	31	…	2	1	0
Доступ	W	…	W	W	W
Сброс	0	…	0	0	0
	CHNL_PRIORITY_CLR for channel[31]	**…**	**CHNL_PRIORITY_CLR for channel[2]**	**CHNL_PRIORITY_CLR for channel[1]**	**CHNL_PRIORITY_CLR for channel[0]**

Таблица 22.36 – Описание бит регистра DMA->CHNLPRIORITY_CLR

№ бита	Функциональное имя бита	Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений
31…0	CHNL_PRIORITY_CLR	Установка разряда присваивает соответствующему каналу DMA уровень приоритета по умолчанию. При записи: **Разряд [C]** = 0 не дает эффекта. Необходимо использовать chnl_priority_set регистр для установки каналу C высокого уровня приоритета. **Разряд [C]** = 1 устанавливает каналу DMA C уровень приоритета по умолчанию. Запись разряда соответствующего нереализованному каналу не дает никакого эффекта

22.4.12 **DMA->ERR_CLR**

22.4.12.1 Данный регистр имеет доступ на запись и чтение. Регистр позволяет сбрасывать сигнал dma_err в 0. Чтение регистра возвращает состояние сигнала dma_err.

Таблица 22.37 – Регистр DMA->ERR_CLR

Номер	31…1	0
Доступ	U	R/W
Сброс	0	0
	-	**ERR_CLR**

Таблица 22.38 – Описание бит регистра DMA->ERR_CLR

№ бита	**Функциональное имя бита**	**Расшифровка функционального имени бита, краткое описание назначения и принимаемых значений**
31…1	-	Не определено. Следует записывать 0.
0	ERR_CLR	Установка сигнала в состояние 0, чтение возвращает состояние сигнала (флага) dma_err. При чтении: **Разряд [C]** = 0 означает, что dma_err находится в состоянии 0; **Разряд [C]** = 1 означает, что dma_err находится в состоянии 1. При записи: **Разряд [C]** =0 не дает эффекта. Состояние dma_err останется неизменным; **Разряд [C]** =1 сбрасывает сигнал (флаг) dma_err в состояние 0. Для целей тестирования возможно использовать регистр err_set, чтобы установить сигнал dma_err в состояние 1. Примечание: При сбросе сигнала dma_err одновременно с появлением ошибки на шине AHB-Lite, то приоритет отдается ошибке и следовательно, значение регистра (и dma_err) останется неизменным (несброшенным)

Лист регистрации изменений

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий номер сопроводительного документа и дата	Подп.	Дата
	изменённых	заменённых	новых	аннулированных					