МИКРОСХЕМЫ ИНТЕГРАЛЬНЫЕ

1867ВЦ4Т

Руководство пользователя №3

ОПИСАНИЕ ЦЕНТРАЛЬНОГО ПРОЦЕССОРА И ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

2006

Содержание

1 Введение

Настоящий документ содержит подробное описание архитектуры, структуры, режимов работы и способов программирования центрального процессора и всех внешних устройств (буферизированного последовательного порта – BSP, последовательного порта с временным разделением каналов – TDM, аппаратного таймера, host-порта – HPI, генератора циклов ожидания – SWWSR), входящих в состав микросхем 1867ВЦ4Т.

2 Основные характеристики микросхемы 1867ВЦ4Т

1 Улучшенная многошинная архитектура с тремя раздельными 16-битными шинами памяти данных и одной шиной памяти программ.

2 40-битное арифметико-логическое устройство (ALU), включающее 40-битный циклический сдвигатель и два независимых 40-битных аккумулятора.

3 17×17-битный параллельный умножитель, соединенный с 40-битным выделенным сумматором для неконвейеризованной одноцикловой операции умножения с накоплением (MAC).

4 Устройство сравнения, выборки и хранения (CSSU) для реализации операций сложения/сравнения алгоритма Виттерби.

5 Экспоненциальный кодировщик для вычисления значения экспоненты 40-битного аккумулятора в одном цикле.

6 Два генератора адреса с восемью вспомогательными регистрами и два арифметических устройства вспомогательных регистров (ARAU).

7 Шина данных с возможностью удержания шины.

8 Максимальное доступное адресное пространство (192К×16) бит ((64К×16) бит - программное, (64К×16) бит данных, 64К пространства портов ввода/вывода).

9 Внутрикристальное ПЗУ (ROM) .

10 Внутрикристальное ОЗУ двойного доступа (DARAM).

11 Повтор одиночной инструкции и операции блочных повторов для программного кода.

12 Инструкции перемещения блоков памяти для улучшения управления памятью программ и данных.

13 Инструкции для работы с длинными (32-бита) операндами.

14 Инструкции с чтением двух или трех операндов.

15 Арифметические инструкции с параллельным сохранением и параллельной загрузкой.

16 Инструкции хранения условий

17 Быстрый возврат из прерываний.

18 Внутрикристальное периферийное оборудование:

– Программируемый генератор циклов ожидания и программируемый переключатель банков памяти.

– Внутрикристальный генератор тактовых импульсов с ФАПЧ и внутренним осциллятором или внешним источником тактовых импульсов (PLL).

– Мультиплексированный последовательный порт с разделением по времени (TDM).

– Буферизированный последовательный порт (BSP).

– 8-битный параллельный host-порт интерфейс (HPI).

– 16-битный таймер (TIM).

– Управление внешними выводами для запрещения внешних шин данных, адреса и управляющих сигналов.

19 Управление потребляемой мощности инструкциями IDLE1, IDLE2 и IDLE3 в режиме малого потребления энергии.

20 Управление отключением тактовых импульсов (CLKOUT) для запрета CLKOUT.

21 Внутрикристальный эмулятор, реализующий стандарт IEEE Std 1149.1 (JTAG) и логика управления внешними вводами (Boundary Scan Logic).

22 Время выполнения одноцикловой инструкции с фиксированной запятой 25 нс (40 миллионов операций в секунду (MIPS)) для 5-вольтового питания.

3 Конструктивные характеристики микросхемы

Таблица 1 – Функциональное назначение выводов микросхемы

Обозначение вывода на УГО	Тип вывода ИС	Номер вывода корпуса	Функциональное назначение вывода
VCC1	-	1	Вывод питания ядра процессора
NC	-	2	Не задействован
NC	-	3	Не задействован
GNDc	-	4	Общий вывод ядра процессора
VCC2	-	5	Вывод питания буферов ввода/вывода
A10	O/Z	6	Вывод 10 разряда шины адреса
HD7	I/O/Z	7	Вывод 7 разряда двунаправленной шины данных параллельного порта (HPI)
A11	O/Z	8	Вывод 11 разряда шины адреса
A12	O/Z	9	Вывод 12 разряда шины адреса
A13	O/Z	10	Вывод 13 разряда шины адреса
A14	O/Z	11	Вывод 14 разряда шины адреса
NC	-	12	Не задействован
A15	O/Z	13	Вывод 15 разряда шины адреса
HAS#	I	14	Вывод стробирования адреса параллельного порта (HPI)
GNDc	-	15	Общий вывод ядра процессора
VCC1	-	16	Вывод питания ядра процессора
HCS#	I	17	Вывод выбора блока параллельного порта (HPI)
HR/W#	I	18	Вывод сигнала чтение/запись параллельного порта (HPI)
GNDd	-	19	Общий вывод буферов ввода/вывода
READY	I	20	Вывод сигнала готовности внешнего устройства к обмену данными в пространстве адресов ввода/вывода процессора
VCC2	-	21	Вывод питания буферов ввода/вывода
PS#	O/Z	22	Вывод сигнала выбора внешней памяти команд
DS#	O/Z	23	Вывод сигнала выбора внешней памяти данных
IS#	O/Z	24	Вывод сигнала выбора данных в пространстве внешних адресов ввода/вывода
R/W#	O/Z	25	Вывод сигнала чтение/запись по шине данных
MSTRB#	O/Z	26	Вывод сигнала стробирования внешней памяти
IOSTRB#	O/Z	27	Вывод сигнала стробирования в пространстве адресов ввода/вывода
NC	-	28	Не задействован
MSC#	O/Z	29	Вывод сигнала завершения режима ожидания в режиме микрокомпьютера
XF	O/Z	30	Вывод программно управляемого флага
HOLDA#	O/Z	31	Вывод сигнала подтверждения запроса прямого доступа к внешней памяти и к пространству ввода/вывода. Переводит шины адреса, данных и сигналы управления памятью в третье состояние
IAQ#	O/Z	32	Вывод сигнала "адрес инструкции на шине адреса"

Продолжение таблицы 2

Обозначение вывода на УГО	Тип выво-да ИС	Номер вывода корпуса	Функциональное назначение вывода
HOLD#	I	33	Вывод запроса прямого доступа к внешней памяти и к пространству адресов ввода/вывода
BIO#	I	34	Вывод управления условным переходом в программе
MP/MC#	I	35	Вывод выбора режима "микропроцессор/микрокомпьютер"
GNDc	-	36	Общий вывод ядра процессора
GNDc	-	37	Общий вывод ядра процессора
NC	-	38	Не задействован
GNDd	-	39	Общий вывод буферов ввода/вывода
VCC2	-	40	Вывод питания буферов ввода/вывода
HCNTL0	I	41	Вывод сигнала 0 управления доступом к регистрам параллельного порта (HPI)
GNDd	-	42	Общий вывод буферов ввода/вывода
BCLKR	I	43	Вывод сигнала тактирования данных, принимаемых по выводу BDR буферизированного последовательного порта (BSP)
TCLKR	I	44	Вывод сигнала тактирования приема данных, принимаемых по выводу TDR последовательного порта с временным разделением (TDM)
BFSR	I	45	Вывод кадрового синхроимпульса приема данных, принимаемых по выводу BDR буферизированного последовательного порта (BSP)
TFSR/TADD	I/O	46	Вывод кадрового синхроимпульса приема данных/ TDM адреса, принимаемых по выводу BDR последовательного порта с временным разделением (TDM)
BDR	I	47	Вывод приема данных буферизированного последовательного порта (BSP)
HCNTL1	I	48	Вывод сигнала 1 управления доступом к регистрам параллельного порта (HPI))
NC	-	49	Не задействован
TDR	I	50	Вывод приема данных последовательного порта с временным разделением (TDM)
NC	-	51	Не задействован
BCLKX	I/O/Z	52	Вывод сигнала тактирования передаваемых данных на выводе BDX буферизированного последовательного порта (BSP)
TCLKX	I/O/Z	53	Вывод сигнала тактирования передаваемых данных на выводе TDX последовательного порта с временным разделением (TDM)
HINT#	O/Z	54	Вывод сигнала прерывания от параллельного порта (HPI)
BFSX	I/O/Z	55	Вывод кадровых синхроимпульсов передаваемых данных на выводе BDX буферизированного последовательного порта (BSP)

Продолжение таблицы 2

Обозначение вывода на УГО	Тип выво-да ИС	Номер вывода корпуса	Функциональное назначение вывода
TFSX/TFRM	I/O/Z	56	Вывод кадровых синхроимпульсов передаваемых данных на выводе TDX последовательного порта с временным разделением (TDM)
HRDY	O/Z	57	Вывод сигнала готовности параллельного порта (HPI)
GNDc	-	58	Общий вывод ядра процессора
VCC2	-	59	Вывод питания буферов ввода/вывода
VCC1	-	60	Вывод питания ядра процессора
GNDd	-	61	Общий вывод буферов ввода/вывода
HD0	I/O/Z	62	Вывод 0 разряда двунаправленной шины данных параллельного порта (HPI)
BDX	O/Z	63	Вывод передаваемых данных буферизированного последовательного порта (BSP)
TDX	O/Z	64	Вывод передаваемых данных последовательного порта с временным разделением (TDM)
IACK#	O/Z	65	Вывод сигнала подтверждения внешнего прерывания
HBIL	I	66	Вывод идентификации принимаемых байтов по параллельному порту (HPI)
NC	-	67	Не задействован
NMI#	I	68	Вывод немаскируемого внешнего прерывания
INT0#	I	69	Вывод маскируемого внешнего прерывания 0
INT1#	I	70	Вывод маскируемого внешнего прерывания 1
INT2#	I	71	Вывод маскируемого внешнего прерывания 2
INT3#	I	72	Вывод маскируемого внешнего прерывания 3
VCC1	-	73	Вывод питания ядра процессора
HD1	I/O/Z	74	Вывод 1 разряда двунаправленной шины данных параллельного порта (HPI)
NC	-	75	Не задействован
GNDc	-	76	Общий вывод ядра процессора
NC	-	77	Не задействован
GNDd	-	78	Общий вывод буферов ввода/вывода
VCC2	-	79	Вывод питания буферов ввода/вывода
NC	-	80	Не задействован
GNDc	-	81	Общий вывод ядра процессора
NC	-	82	Не задействован
GNDd	-	83	Общий вывод буферов ввода/вывода
CLKMD1	I	84	Вывод сигнала 1 выбора режима тактирования процессора
CLKMD2	I	85	Вывод сигнала 2 выбора режима тактирования процессора
CLKMD3	I	86	Вывод сигнала 3 выбора режима тактирования процессора
CNT	I	87	Вывод сигнала выбора уровня совместимости входов/выходов ТТЛ и КМОП-логики

Продолжение таблицы 2

Обозначение вывода на УГО	Тип выво-да ИС	Номер вывода корпуса	Функциональное назначение вывода
HD2	I/O/Z	88	Вывод 2 разряда двунаправленной шины данных параллельного порта (HPI)
TOUT	O/Z	89	Вывод сигнала таймера
NC	-	90	Не задействован
EMU0	I/O/Z	91	Вывод сигнала 0 прерывания эмулятора
EMU1/OFF#	I/O/Z	92	Вывод сигнала 1 прерывания эмулятора/ переводит выходные буферы выводов в третье состояние
TDO	O/Z	93	Вывод выходных данных из тестового порта (стандарт IEEE1149.1)
TDI	I	94	Вывод входных данных в тестовый порт (стандарт IEEE1149.1)
TRST#	I	95	Вывод сигнала сброса тестового режима (стандарт IEEE1149.1)
TCK	I	96	Вывод сигнала тактирования тестового порта (стандарт IEEE1149.1)
GNDd	-	97	Общий вывод буферов ввода/вывода
TMS	I	98	Вывод сигнала выбора состояния тестового порта (стандарт IEEE1149.1)
VCC2	-	99	Вывод питания буферов ввода/вывода
HPIENA	I	100	Вывод разрешения выбора параллельного порта (HPI)
GNDc	-	101	Общий вывод ядра процессора
CLKOUT	O/Z	102	Вывод сигнала машинного цикла CPU
HD3	I/O/Z	103	Вывод 3 разряда двунаправленной шины данных параллельного порта (HPI)
NC	-	104	Не задействован
X1	-	105	Вывод для подключения кварцевого резонатора
X2/ CLKIN	- I	106	Вывод для подключения кварцевого резонатора/ вход тактового сигнала
VCC1	-	107	Вывод питания ядра процессора
RS#	I	108	Вывод сигнала общего сброса
D0	I/O/Z	109	Вывод 0 разряда шины данных
D1	I/O/Z	110	Вывод 1 разряда шины данных
D2	I/O/Z	111	Вывод 2 разряда шины данных
D3	I/O/Z	112	Вывод 3 разряда шины данных
D4	I/O/Z	113	Вывод 4 разряда шины данных
D5	I/O/Z	114	Вывод 5 разряда шины данных
GNDd	-	115	Общий вывод буферов ввода/вывода
NC	-	116	Не задействован
GNDc	-	117	Общий вывод ядра процессора
VCC1	-	118	Вывод питания ядра процессора
NC	-	119	Не задействован
GNDc	-	120	Общий вывод ядра процессора

Окончание таблицы 2

Обозначение вывода на УГО	Тип выво-да ИС	Номер вывода корпуса	Функциональное назначение вывода
VCC2	-	121	Вывод питания буферов ввода/вывода
NC	-	122	Не задействован
D6	I/O/Z	123	Вывод 6 разряда шины данных
D7	I/O/Z	124	Вывод 7 разряда шины данных
D8	I/O/Z	125	Вывод 8 разряда шины данных
D9	I/O/Z	126	Вывод 9 разряда шины данных
D10	I/O/Z	127	Вывод 10 разряда шины данных
D11	I/O/Z	128	Вывод 11 разряда шины данных
D12	I/O/Z	129	Вывод 12 разряда шины данных
HD4	I/O/Z	130	Вывод 4 разряда двунаправленной шины данных параллельного порта (HPI)
D13	I/O/Z	131	Вывод 13 разряда шины данных
D14	I/O/Z	132	Вывод 14 разряда шины данных
D15	I/O/Z	133	Вывод 15 разряда шины данных
HD5	I/O/Z	134	Вывод 5 разряда двунаправленной шины данных параллельного порта (HPI)
GNDd	-	135	Общий вывод буферов ввода/вывода
GNDc	-	136	Общий вывод ядра процессора
HDS1#	I	137	Вывод сигнала 1 стробирования данных параллельного порта (HPI)
VCC1	-	138	Вывод питания ядра процессора
HDS2#	I	139	Вывод сигнала 2 стробирования данных параллельного порта (HPI)
VCC2	-	140	Вывод питания буферов ввода/вывода
A0	O/Z	141	Вывод 0 разряда шины адреса
A1	O/Z	142	Вывод 1 разряда шины адреса
A2	O/Z	143	Вывод 2 разряда шины адреса
A3	O/Z	144	Вывод 3 разряда шины адреса
NC	-	145	Не задействован
HD6	I/O/Z	146	Вывод 6 разряда двунаправленной шины данных параллельного порта (HPI)
NC	-	147	Не задействован
A4	O/Z	148	Вывод 4 разряда шины адреса
A5	O/Z	149	Вывод 5 разряда шины адреса
A6	O/Z	150	Вывод 6 разряда шины адреса
A7	O/Z	151	Вывод 7 разряда шины адреса
A8	O/Z	152	Вывод 8 разряда шины адреса
A9	O/Z	153	Вывод 9 разряда шины адреса
NC	-	154	Не задействован
GNDd	-	155	Общий вывод буферов ввода/вывода
GNDc	-	156	Общий вывод ядра процессора
Примечание – В графе "Тип вывода": I – вход, O – выход, I/O – вход/выход, O/Z - выход/третье состояние, I/O/Z – вход/выход/третье состояние.			

4 Корпус микросхемы

На рисунке 1 представлен корпус ИМС 1867ВЦ4Т.

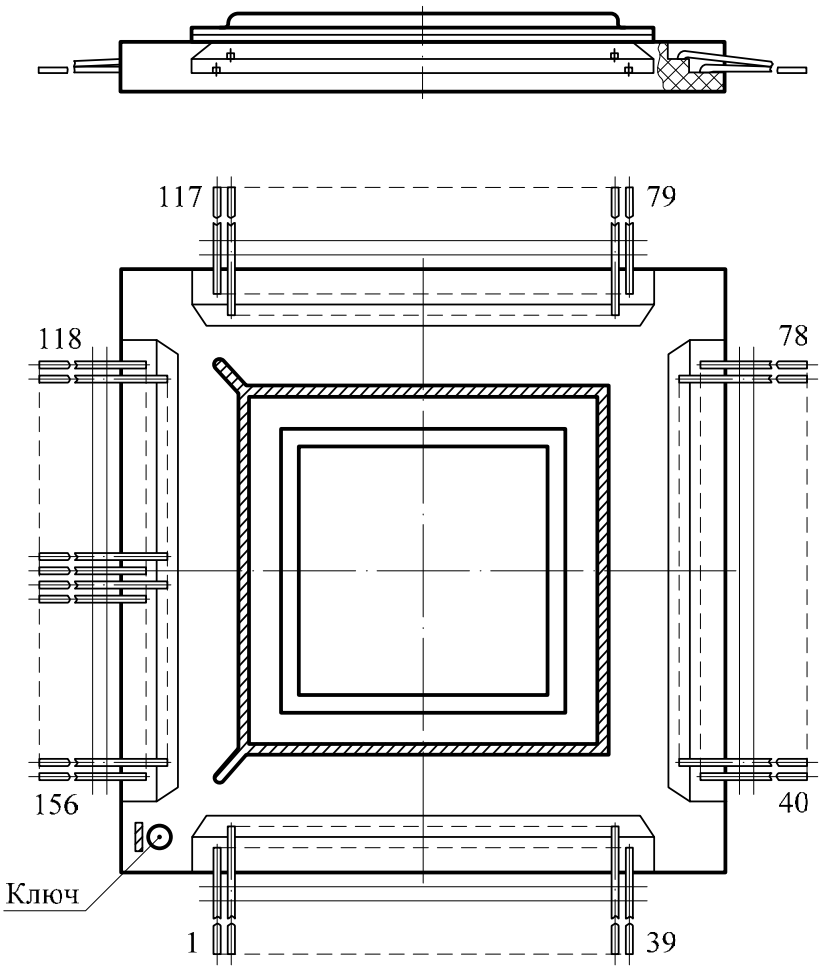


Рисунок 1 – Корпус 4234.156-2 ИМС 1867ВЦ4Т

5 Архитектура микросхемы

Цифровой сигнальный процессор 1867ВЦ4Т использует усовершенствованную модифицированную гарвардскую архитектуру. Разделение областей памяти программ и данных предоставляет одновременный доступ к программным инструкциям и данным, обеспечивая высокую степень параллельности. Например, три операции чтения и одна операция записи могут быть выполнены за один машинный цикл. Инструкции с параллельным сохранением и используемые для специальных приложений в полной мере используют эту архитектуру. Кроме того, данные могут быть перемещены между областями памяти программ и данными. Такой параллелизм обеспечивает широкий набор арифметических и логических операций, поразрядных операций, которые могут быть все выполнены в одном машинном цикле. Кроме того, 1867ВЦ4Т обладает механизмом управления прерываниями, циклическими операциями и вызовом подпрограмм.

На рисунке 2 представлена функциональная блок-схема, где представлены основные блоки и шинная структура.

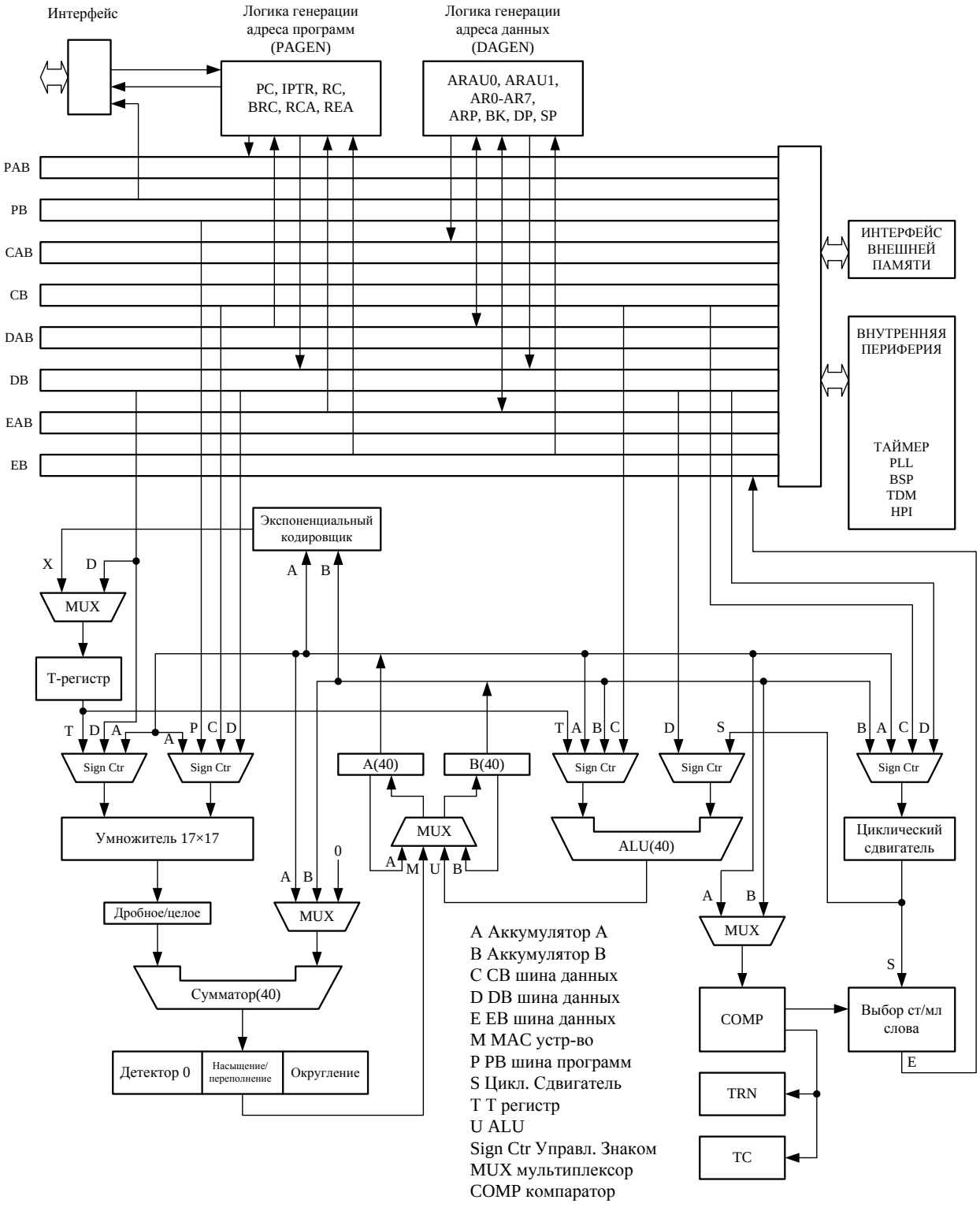


Рисунок 2 – Блок-схема внутренних аппаратных средств 1867ВЦ4Т

5.1 Структура микросхемы

5.1.1 Структура шин

Архитектура 1867ВЦ4Т построена на основе восьми 16-битных шин:

– Программная шина (PB), по которой передаются коды инструкций и непосредственные операнды из памяти программ.

– Три внутренних шины данных (CB, DB и EB), соединяющие различные элементы, такие как ЦПУ, устройства генерации адреса данных и адреса программ, внутрикристальных периферийных устройств кристалла и памяти данных.

— CB и DB предназначены для чтения операндов из памяти данных.

— EB обслуживает данные для записи в память.

Четыре адресных шины (PAB, CAB, DAB, и EAB) предназначены для адресов, необходимых для выполнения инструкций.

1867ВЦ4Т может генерировать до двух адресов памяти данных в одном машинном цикле, используя два арифметических устройства вспомогательных регистров (ARAU0 и ARAU1).

Шина PB может передавать операнды данных, сохраненные в памяти программ (например, таблицу коэффициентов) в умножитель и сумматор для выполнения операций умножения с накоплением или для осуществления в области памяти данных инструкций перемещения данных (MVPD и READA). Эта способность позволяет поддерживать выполнение в одном цикле инструкций с тремя операндами такими, как FIRS.

Процессор 1867ВЦ4Т также имеет встроенную двунаправленную шину для доступа к внутренним периферийным устройствам. Эта шина соединяется с шинами DB и EB через шинный мультиплексор в интерфейсе ЦПУ. Обращения, которые использует эта шина, могут требовать два или более циклов для чтения и записи в зависимости от структуры периферийного устройства.

Таблица 4 суммирует возможности использования шин для различных типов доступа.

Таблица 2 – Использование шин для различного типа доступа

Тип доступа	Шина адреса				Шина данных			
	PAB	CAB	DAB	EAB	PB	CB	DB	EB
Чтение программы	√				√			
Запись программы	√							√
Одиночное чтение данных			√				√	
Двойное чтение данных		√	√			√	√	
Чтение 32-битных данных		√ (hw)	√ (lw)			√ (hw)	√ (lw)	
Одиночная запись данных				√				√
Чтение и запись данных			√	√			√	√
Двойное чтение/ Чтение коэффициентов	√	√	√		√	√	√	
Чтение периферии			√				√	
Запись периферии				√				√
Обозначения: hw – старшее 16-бит слово; lw – младшее 16-бит слово.								

5.1.2 Организация внутренней памяти

Пространство памяти процессора 1867ВЦ4Т организовано в виде трех самостоятельно выбираемых блоков: пространство памяти программы, данных и пространства ввода/вывода.

В области памяти программы содержатся исполняемые команды, а так же исполняемые таблицы. Пространство памяти данных хранит данные, используемые командами. Пространство памяти входа/выхода с помощью интерфейса связывает внешнюю отображаемую память с внешними устройствами, а так же может служить дополнительным устройством для хранения данных.

Полный диапазон адресов памяти устройства 1867ВЦ4Т составляет (192К×16)-битных слов. Пространство памяти разделено на три специфичных сегмента памяти: (64К×16) бит памяти программ, (64К×16) бит памяти данных и (64К×16) бит пространства памяти ввода/вывода.

5.1.3 Внутрикристальное ПЗУ (ROM)

1867ВЦ4Т имеет встроенное ПЗУ (2К×16) бит, во встроенном ПЗУ располагается программа начальной загрузки (boot loader) и таблица векторов прерываний. Программа начальной загрузки может загружаться через последовательные порты, порты ввода/вывода, из внешней памяти или через host-интерфейс.

5.1.4 Внутрикристальное ОЗУ (DARAM)

Процессор 1867ВЦ4Т имеет ОЗУ двойного доступа (DARAM) размером (10К×16) бит. Размещенное на кристалле ОЗУ разделено на отдельные блоки для увеличения производительности, позволяет выбирать два операнда из одного блока DARAM и в том же цикле производить запись в другой блок DARAM. DARAM может картироваться как в область памяти программ, так и в область памяти данных.

5.1.5 Защита внутрикристальной памяти

Устройство 1867ВЦ4Т имеет возможность маскирования для защиты содержимого встроенной памяти. Когда выбрана эта опция, никакая внешняя инструкция не может получить доступ к встроенной памяти.

5.1.6 Картированные в памяти регистры

Пространство памяти данных содержит картированные (отображенные) в памяти регистры ЦПУ и внутрикристальные периферийные устройства. Регистры размещены на нулевой странице памяти данных, что упрощает доступ к ним. Доступ через отображение в памяти обеспечивает оптимально удобный способ сохранять и восстанавливать регистры для переключения контекста и осуществлять обмен информацией между аккумуляторами и другими регистрами.

5.2 Центральное процессорное устройство (CPU)

ЦПУ процессора 1867ВЦ4Т состоит из следующих компонентов:

- 40-разрядное арифметико-логическое устройство (ALU).
- Два 40-разрядных аккумулятора.
- Циклический сдвигатель с поддержкой диапазона сдвига от 16 до 31.
- 17×17-разрядный умножитель.
- 40-разрядный сумматор.
- Устройство сравнения, выборки и хранения (CSSU).
- Устройство генерации адреса данных.
- Устройство генерации адреса программ.

5.2.1 Арифметико-логическое устройство (ALU)

1867ВЦ4Т выполняет арифметические операции в двоичном дополнительном коде, используя 40-разрядное арифметико-логическое устройство (ALU) и два 40-разрядных аккумулятора (ACCA и ACCB). ALU может также выполнять логические операции. ALU получает данные на вход из нескольких источников:

- 16-разрядное непосредственное значение;
- 16-разрядное слово из памяти данных;
- 16-разрядное значение из T-регистра;
- два 16-разрядных слова из памяти данных;
- 32-разрядное слово из памяти данных;
- 40-разрядное слово из любого из аккумуляторов.

Для выполнения арифметических операций ALU может работать в специальном двойном 16-разрядном арифметическом режиме, при котором одновременно (в одном цикле) выполняются две 16-разрядные операции.

5.2.2 Аккумуляторы (ACC)

Аккумуляторы ACCA и ACCB сохраняют выходные данные из ALU или блока умножителя/сумматора. Аккумулятор также может служить дополнительным входом для ALU; аккумулятор ACCA может быть входом для блока умножителя/сумматора. Каждый аккумулятор разделен на три части:

- Биты безопасности (39-32).
- Старшие биты (31-16).
- Младшие биты (15-0).

Инструкции предусматривают сохранение битов безопасности, старших и младших слов аккумулятора в памяти данных и передачу 32-разрядных слов в или из памяти данных. Также, любой из аккумуляторов может быть использован для временного хранения содержимого другого аккумулятора.

5.2.3 Циклический сдвигатель

Циклический сдвигатель имеет 40-разрядный вход, соединенный с аккумулятором или с памятью данных (используя шины CB, DB), и 40-разрядный выход, подключенный к ALU или к памяти данных (используя шину EB). Циклический сдвигатель может производить сдвиг влево в диапазоне от 0 до 31 разрядов и сдвиг вправо в диапазоне от 0 до 16 разрядов для входных данных. Величина сдвига определяется полем счетчика сдвига инструкции, полем счетчика сдвига (ASM) статусного регистра ST1 или Т-регистра (когда он определен как регистр счетчика сдвига).

Сдвигатель и экспоненциальный кодировщик нормализуют содержимое аккумулятора за один машинный цикл. Младшие биты (LSBs) на выходе заполняются нулями, а старшие биты (MSBs) могут быть заполнены либо нулями, либо знаковым расширением, в зависимости от значения бита режима знакового расширения SXM регистра состояния ST1. Кроме того, возможности сдвига позволяют процессору выполнять численное масштабирование, операции извлечения отдельных битов, операции расширенной арифметики и операции предотвращения переполнения.

5.2.4 Умножитель и сумматор

Устройство умножитель/сумматор выполняет 17×17-битную операцию умножения в дополнительном коде с 40-разрядным сложением за один цикл.

Блок умножителя/сумматора состоит из нескольких элементов: умножитель, блок входного контроля наличия знака, блок контроля остатка от деления, детектор нуля, устройство округления (в двоично-дополнительном коде), блок логики переполнения/насыщения и 16-разрядный временный Т-регистр. Умножитель имеет два входа: один вход выбирается из Т-регистра, операнда памяти данных или аккумулятора А; другой выбирается из памяти программ, памяти данных, аккумулятора А или как непосредственное значение.

Быстродействие встроенного умножителя позволяет 1867ВЦ4Т эффективно выполнять такие операции как свертка, корреляция и цифровая фильтрация. Кроме того, умножитель и ALU совместно выполняют операции умножения с накоплением (MAC) и операции в ALU параллельно в одном машинном цикле. Эта функция используется для определения Евклидового кодового расстояния и в фильтрах наименьшего среднего квадратичного отклонения (LMS), которые необходимы для сложных алгоритмов цифровой обработки сигналов.

5.2.5 Устройство сравнения, выбора и хранения (CSSU)

Устройство сравнения, выборки и хранения (CSSU) выполняет операцию сравнения между старшим и младшим словом аккумулятора; позволяет биту флага тест/управление (TC) в статусном регистре (ST0) и регистру переноса (TRN) сохранять их истории переноса и выбирать длинное слово в аккумуляторе для хранения в памяти данных. CSSU также ускоряет вычисление алгоритмов Виттерби (типа “бабочка”) посредством оптимизированного аппаратного обеспечения.

5.3 Адресация данных

1867ВЦ4Т имеет семь основных режимов адресации данных:

– *Непосредственная адресация* использует команды с кодировкой фиксированного значения.

– *Абсолютная адресация* использует команды с кодировкой фиксированного адреса.

– *Адресация по содержимому аккумулятора* использует аккумулятор A для доступа к памяти программ как к данным.

– *Прямая адресация* использует семь разрядов команды для кодировки младших семи разрядов адреса. Семь разрядов используются либо с указателем страницы памяти (DP) или с указателем стека (SP) для определения абсолютного адреса памяти.

– *Косвенная адресация* для доступа к памяти использует вспомогательные регистры.

– *Адресация регистрами, отображенными в памяти*, использует картированные в памяти регистры (MMR) без изменения текущих значений DP и SP.

– *Стековая адресация* управляет добавлением и извлечением элементов из системного стека.

В течение выполнения инструкций, использующих прямую, косвенную адресации или MMR-адресацию, логика генерации адреса данных (DAGEN) вычисляет адреса операндов в памяти данных.

5.4 Адресация программной памяти

Память программ в 1867ВЦ4Т адресуется программным счетчиком (PC). Программный счетчик (PC), который используется для выборки индивидуальной команды, загружается под управлением логики генерации программного адреса (PAGEN). PAGEN увеличивает значение PC, когда выбирается следующая команда. Причиной нарушения последовательности выполнения потока команд могут быть операции перехода, вызова подпрограммы, возврата из подпрограммы, сброс, прерывания, команда повторения блока и т. п. В случае вызова подпрограмм и прерываний текущее значение PC сохраняется в стеке. Когда функция возврата или подпрограмма обработки прерывания завершаются, значение PC, которое было сохранено, восстанавливается из стека командой возврата.

5.5 Операции конвейера

Конвейер команд содержит последовательность операций, которые имеют место во время выполнения команды. Конвейер команд имеет шесть уровней: предвыборка, выборка, декодирование, доступ, чтение и исполнение. На каждом уровне происходят разные операции. Одна из шести команд может быть активна в любом данном цикле, на различных фазах завершения. Обычно конвейер полностью заполнен последовательным набором команд, каждая – на одной из шести фаз. Когда в счетчике команд PC происходит разрыв (при выполнении команд перехода, вызова или возврата), одна из шести фаз может не использоваться.

5.6 Внутрикристальные периферийные устройства

- Контакты ввода/вывода общего назначения.
- Программируемый генератор состояний ожидания.
- Программируемая логика переключения банков памяти.
- Генератор тактовых синхроимпульсов.
- Таймер.
- Мультиплексированный последовательный порт с разделением по времени (TDM).
- Буферизированный последовательный порт (BSP).
- Интерфейс ведущего (host) порта (HPI).

5.6.1 Ввод/вывод общего назначения

Устройство 1867ВЦ4Т имеет два специальных контакта, через которые под управлением программы можно обмениваться данными с внешними устройствами. К ним относятся:

BIO# – общий вход, состояние которого определяет условие перехода в команде ветвления.

XF – выход внешнего флага, который может программным путем устанавливаться в 0 или 1.

BIO# и XF часто используются для функций установления связи.

5.6.2 Программируемый генератор состояний ожидания

Программируемый генератор состояний ожидания растягивает доступ к внешней шине до семи машинных циклов для работы с более медленными внешними устройствами и внешней памятью. Для доступа к внешней памяти число состояний ожидания (от 0 до 7) определяется регистром SWWSR для каждого (32К×16) бит блока памяти программ и памяти данных, а также для (64К×16) бит блока пространства ввода/вывода I/O.

5.6.3 Программируемая логика переключения банков памяти

Программируемая логика переключения банков памяти может автоматически вставлять один цикл при пересечении границы между банками памяти внутри областей памяти программ или памяти данных, переключении из памяти программ в память данных или в выбранных устройствах из одной страницы памяти программ в другую страницу памяти программ. Этот дополнительный цикл предотвращает конфликт между шинами. Размер банка памяти определяется регистром управления переключения банков BSCR.

5.6.4 Аппаратный таймер

Таймер представляет собой 16-разрядную схему, тактируемую через 4-битный предварительный делитель частоты. Счетчик таймера уменьшается на 1 в каждом цикле CLKOUT. Каждый раз, когда счетчик обнуляется, генерируется прерывание таймера. Таймер может быть остановлен, сброшен или запрещен установкой определенных битов состояния в управляющем регистре.

5.6.5 Генератор тактовой синхронизации

Есть два основных варианта для генерации тактовых синхроимпульсов: внутренний генератор или схема с фазовой автоматической подстройкой частоты ФАПЧ (Phase-Locked Loop – PLL). В первом варианте, тактовая частота CPU генерируется делением входного тактового сигнала (через X2/CLKIN) на 1, 2 или 4. Во втором – используется схема PLL для генерации тактовой частоты процессора CPU через умножение частоты. PLL-метод позволяет генерировать из низкочастотного внешнего сигнала высокочастотные внутренние синхроимпульсы для CPU. Использование низкочастотного внешнего тактового сигнала уменьшает потребляемую мощность системы, ослабляет сгенерированные шумы EMI. Требуемые режимы синхронизации первоначально выбираются с помощью выводов CLKMDn.

5.6.6 Интерфейс host-порта

Интерфейс host-порта (HPI) – это параллельный порт, который обеспечивает интерфейс к host-процессору. Обмен данными между host-процессором и 1867ВЦ4Т происходит через внутреннюю память 1867ВЦ4Т, которая доступна для обоих процессоров.

5.6.7 Буферизированный последовательный порт

Буферизированный последовательный порт (BSP) является полнодуплексным с двойной буферизацией данных и с блоком автоматической буферизации (ABU) синхронным последовательным портом, представляющим собой расширенную версию стандартного последовательного порта. Логика блока ABU позволяет порту BSP обмениваться с внешними устройствами массивами данных, размещенных в определенном внутреннем блоке памяти кристалла, на аппаратном уровне без участия процессора. Двойная буферизация данных позволяет BSP обмениваться непрерывным потоком данных длиной в 8, 10, 12 или 16 бит. Максимальная скорость обмена через порт BSP определяется частотой CLKOUT.

5.6.8 Мультиплексированный последовательный порт с разделением по времени

Мультиплексированный последовательный порт с разделением по времени – это расширенный вариант синхронного последовательного порта, который позволяет создать мультипроцессорную сеть из нескольких (до восьми) процессоров, обладающими TDM-портом.

5.6.9 Интерфейс внешней шины

1867ВЦ4Т может адресовать до (64К×16) бит памяти данных, (64К×16) бит памяти программ и до (64К×16) бит 16-разрядных параллельных портов I/O. Доступ к любой внешней памяти или портам ввода/вывода I/O осуществляется через внешний интерфейс. Индивидуальные сигналы выбора пространств DS#, PS#, IS# предоставляют возможность обращаться к физически разделенным областям.

Входной сигнал готовности внешнего интерфейса и программно-сгенерированные состояния ожидания позволяют процессору обращаться к низкоскоростным внешней памяти или устройствам ввода/вывода.

Режим хранения позволяет внешнему устройству взять под свой контроль внешние шины 1867ВЦ4Т. Чтобы обратиться к портам ввода/вывода используются две специальные команды: PORTR и PORTW.

5.7 Память

Устройство 1867ВЦ4Т имеет пространство памяти общим объемом (192К×16) бит. Это пространство разделяется на три основных блока: (64К×16) бит программы, 64К для данных и 64К для операций ввода/вывода.

5.7.1 Пространство памяти

Пространство памяти 1867ВЦ4Т организовано в виде трех индивидуально выбираемых блоков: память программ, память данных и память ввода/вывода.

В области памяти программ содержатся исполняемые команды, а так же исполняемые таблицы. Пространство памяти данных хранит данные, используемые командами. Пространство памяти ввода/вывода с помощью интерфейса связывается с внешней, картируемой в памяти, периферией, а также может служить как дополнительное пространство для хранения данных. Имеются два бита регистра состояния режима процессора (PMST), которые управляют конфигурацией памяти: MP/MC# и OVLY.

На рисунке 3 – показана карта памяти процессора 1867ВЦ4Т.

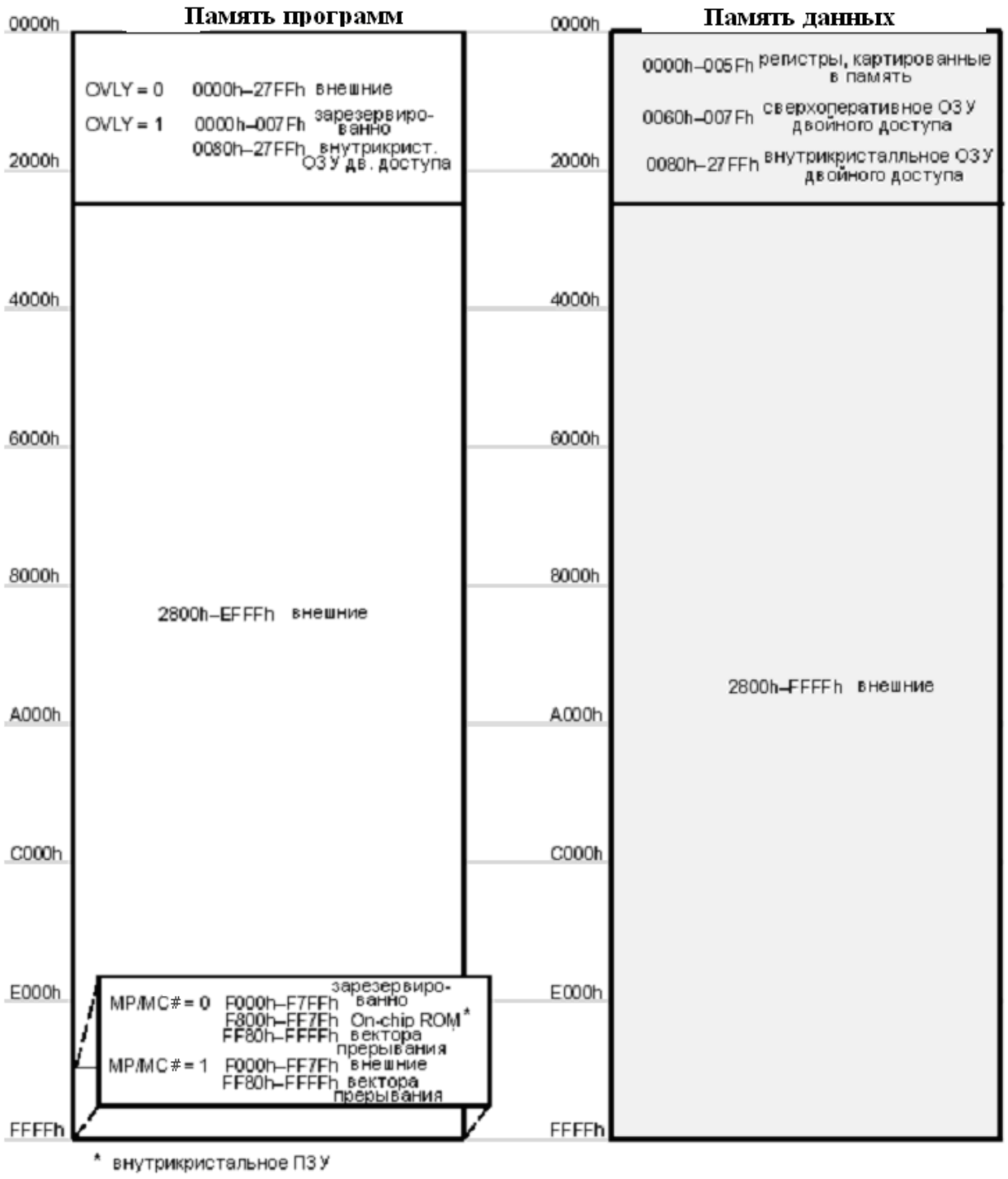


Рисунок 3 – Карта памяти

5.7.2 Программная память

Внешняя память программ процессора 1867ВЦ4Т позволяет обращаться к (64К×16) бит. Устройство 1867ВЦ4Т имеет внутрикристальное ПЗУ (2К) и ОЗУ двойного доступа DARAM (10К).

Когда ячейки памяти картированы в области программ, устройство 1867ВЦ4Т автоматически разрешает доступ к ячейкам памяти, когда адреса попадают на границу внутрикристальной памяти. Когда устройство генерации адресов программ (PAGEN) генерирует адрес, находящийся за границей внутрикристальной памяти, устройство автоматически разрешает обращение к внешней области.

5.7.2.1 Конфигурирование памяти программы

Биты MP/MC# и OVLY определяют, какая внутрикристальная память доступна в области программы.

При сбросе логический уровень, установленный на MP/MC#, передается биту MP/MC# в регистре PMST. Бит MP/MC# определяет доступность внутрикристального ПЗУ. Если MP/MC# = 1, устройство будет конфигурировано как микропроцессор и внутрикристальное ПЗУ будет недоступно. Если MP/MC# = 0, то устройство конфигурировано как микрокомпьютер и внутрикристальное ПЗУ доступно. Состояние MP/MC# может быть изменено только путем сброса, однако, с помощью программных средств можно отключить возможность выбора доступности внутрикристального ПЗУ, установив необходимое значение непосредственно в регистре PMST.

5.7.2.2 Организация внутрикристального ПЗУ

Объем внутрикристального ПЗУ (ROM) процессора 1867ВЦ4Т составляет (2К×16) бит с адресным пространством F7FF-FFFF.

5.7.2.3 Карта адресов памяти программ и содержание внутрикристального ПЗУ

При сбросе устройства векторы сброса, прерываний и системного прерываний отображаются на 128-словной странице, начинающейся с адреса FF80 в области памяти программ. Однако при сбросе устройства эти векторы могут быть повторно отображены к началу любой 128-словной страницы в области программ. Это особенность облегчает перемещение таблицы векторов из загрузочного ПЗУ и удаление ПЗУ из карты памяти.

Примечание – Во внутрикристальном ROM (128×16) бит с адресами FF00h-FF7Fh в области программ зарезервированы для целей проверки устройства. Код прикладной программы не должен использовать эти адреса.

5.7.2.4 Содержимое кодов внутрикристального ПЗУ и картирование

Устройство 1867ВЦ4Т имеет внутрикристальное загрузочное ПЗУ объемом (2К×16) бит (адреса с F800 по FFFF), которое содержит следующую информацию:

– программа начальной загрузки, которая загружается через последовательный порт, порты ввода/вывода, из внешней памяти или через host-интерфейс;
– таблица расширения μ-функции из 256 слов;
– таблица расширения А-функции из 256 слов;
– таблица векторов прерывания.

Диапазон адресов F800-FFFF для программ картирован во внутрикристальное ПЗУ, если бит MP/MC# = 0.

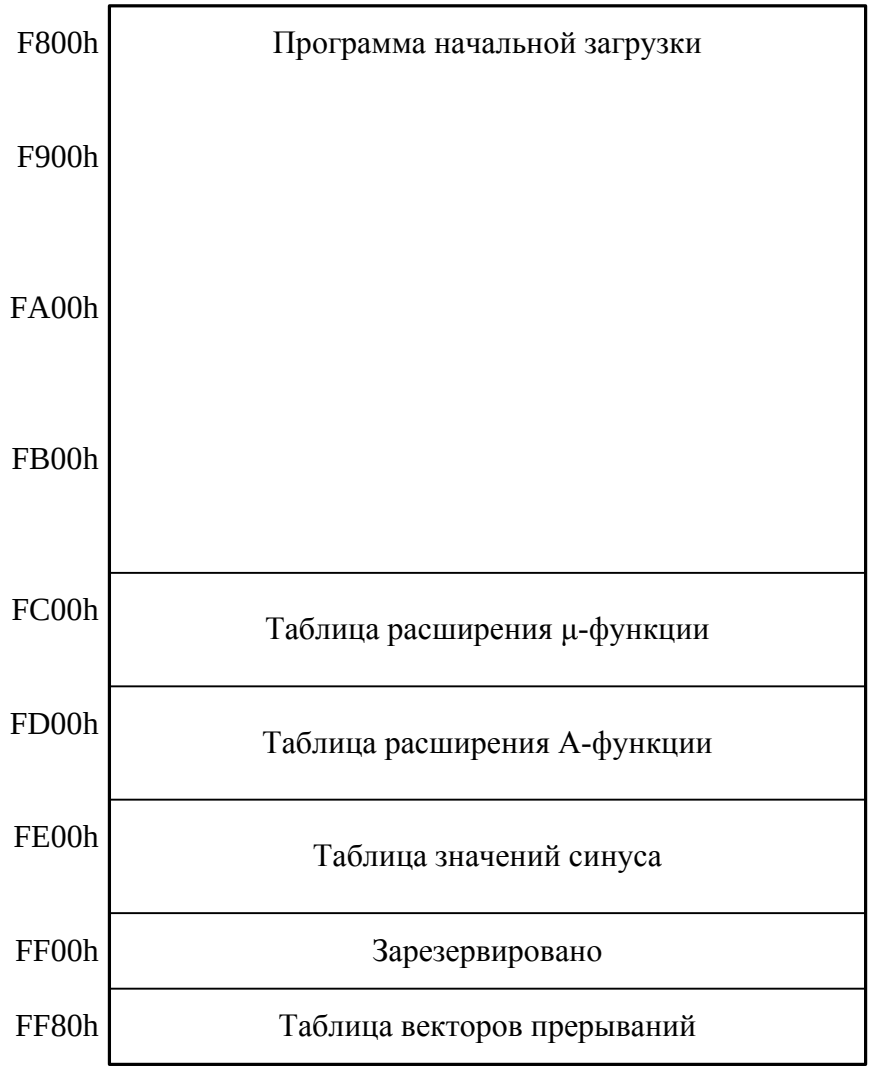


Рисунок 4 – Распределение адресов внутрикристальной ROM (верхние адреса)

5.7.3 Память данных

Память данных процессора1867ВЦ4Т содержит (64К×16) бит, из них (10К×16) бит – это внутрикристальное двойного доступа ОЗУ (DARAM). Доступ к ОЗУ (если разрешен) возможен тогда, когда адреса находятся в пределах границ, соответствующих внутрикристальным блокам памяти. Когда логика генерации адресов данных (DAGEN) генерирует адрес вне границ внутрикристальной памяти, устройство автоматически генерирует внешний адрес.

5.7.3.1 Конфигурация и организация памяти данных

Память данных может находиться как на кристалле, так и вне кристалла. Внутрикристальная DARAM размещается в пространстве памяти данных.

Для увеличения производительности внутрикристальное ОЗУ разделено на блоки. Например, такая организация ОЗУ позволяет выбирать два операнда из одного блока DARAM и записывать в другой блок DARAM.

На рисунке 5 показана организация блоков DARAM.

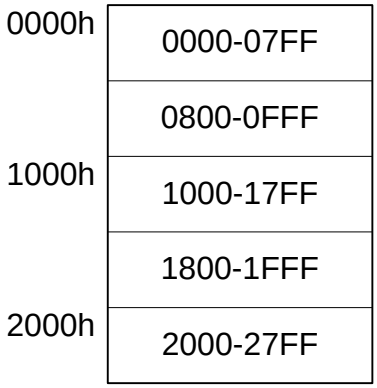


Рисунок 5 – Блочная организация внутрикристального ОЗУ

Первые 1К ОЗУ двойного доступа включают в себя картированные в памяти регистры CPU и периферийных устройств, (32×16) бит сверхоперативной памяти и (896×16) бит DARAM. Весь объем памяти разделен на блоки по 2К.

5.7.3.2 Картируемые в памяти регистры

В (64К×16) бит пространства памяти данных содержатся регистры, отображаемые в памяти, которые размещены на нулевой странице данных (адреса 0000h-007Fh). Страница данных 0 содержит следующее:

– Регистры центрального процессора (общее количество 26) доступны без состояний ожидания.

– Периферийные регистры используются как регистры управления или данных для периферийных устройств. Эти регистры содержатся в адресах 0020h-005F и подключены к специальной внутренней периферийной шине.

– Блок сверхоперативной RAM (адреса 60h-7Fh в памяти данных) включает 32 слова типа DARAM для временного хранения переменных, неорганизованных в большие массивы.

5.7.3.3 Картируемые в памяти регистры CPU

В таблице 3 представлены регистры CPU, отображаемые в памяти.

Таблица 3 – Регистры CPU, отображаемые в памяти

Имя	Адрес		Описание
	DEC	HEX	
IMR	0	0	Регистр маски прерывания
IFR	1	1	Регистр флага прерывания
-	2-5	2- 5	Зарезервировано для тестирования
ST0	6	6	Регистр состояний 0
ST1	7	7	Регистр состояний 1
AL	8	8	Младшее слово аккумулятора A (15-0)
AH	9	9	Старшее слово аккумулятора A (31-16)
AG	10	A	Биты безопасности аккумулятора A (39-32)
BL	11	B	Младшее слово аккумулятора B (15-0)
BH	12	C	Старшее слово аккумулятора B (31-16)
BG	13	D	Биты безопасности аккумулятора B (39-32)
T	14	E	Регистр временного хранения
TRN	15	F	Регистр перехода
AR0	16	10	Дополнительный регистр 0
AR1	17	11	Дополнительный регистр 1
AR2	18	12	Дополнительный регистр 2
AR3	19	13	Дополнительный регистр 3
AR4	20	14	Дополнительный регистр 4
AR5	21	15	Дополнительный регистр 5
AR6	22	16	Дополнительный регистр 6
AR7	23	17	Дополнительный регистр 7
SP	24	18	Регистр указатель вершины стека
BK	25	19	Регистр размера циклического буфера
BRC	26	1A	Счетчик повторения блоков
RSA	27	1B	Начальный адрес повтора блока
REA	28	1C	Конечный адрес повтора блока
PMST	29	1D	Регистр состояния режимов процессора
-	30-31	1E-1F	Зарезервировано

5.7.3.3.1 Регистры прерываний (IMR, IFR)

Регистр маски прерываний (IMR) индивидуально маскирует отдельные прерывания в требуемое время. Регистр флага прерываний (IFR) показывает текущее состояние прерываний.

5.7.3.3.2 Регистры состояния (ST0, ST1)

Регистры состояния содержат информацию о состоянии различных условий и режиме работы процессора. Регистр ST0 содержит флаги (OVA, OVB, C и TC) и производит арифметические операции и манипуляции с битами в дополнение к полям DP и ARP. ST1 отражает состояние режимов и команд, выполняемых процессором.

5.7.3.3.3 Аккумуляторы (A, B)

1867ВЦ4Т имеет два 40-разрядных аккумулятора: аккумулятор A и аккумулятор B. Каждый аккумулятор отображен в памяти данных и разделен на младшее слово (AL, BL), старшее слово (AH, BH) и биты безопасности (AG, BG).

5.7.3.3.4 Регистр временного хранения (T)

Регистр временного хранения (T) имеет много применений. Например, он может хранить:

– Один из множителей при операциях умножения или умножения/накопления.

– Счет динамического сдвига для команд с операциями сдвига, такими как ADD, LD и SUB.

– Динамический адрес бита для команды BITT.

– Атрибут переноса, используемый командами DADST и DSADT для ACS-операций декодирования по Виттерби.

Кроме того, команда EXP сохраняет вычисленную величину порядка в T регистре, после чего команда NORM использует содержимое T регистра для нормализации результата.

5.7.3.3.5 Регистр перехода (TRN)

16-разрядный регистр перехода (TRN) задерживает решение перехода для пути к новой метрике, чтобы выполнить алгоритм Виттерби. Команда CMPS (сравнение, выбор максимума и сохранение) обновляет содержимое регистра TRN на основании сравнения старшего и младшего слова аккумулятора.

5.7.3.3.6 Вспомогательные регистры (AR0-AR7)

1867ВЦ4Т может обращаться к восьми вспомогательным регистрам (AR0-AR7) и изменять их значение с помощью устройства ARAU (арифметическое устройство вспомогательных регистров). Основная функция этих регистров заключается в генерации 16-разрядных адресов для адресации данных. Однако вспомогательные регистры могут использоваться и как регистры общего назначения, и как счетчики.

5.7.3.3.7 Регистр указателя вершины стека (SP)

16-разрядный регистр указателя стека (SP) содержит адрес вершины стека системы. SP всегда указывает на последний элемент, помещенный в стек. Стек управляется прерываниями, внутренними прерываниями, вызовами, возвратами и командами PSHD, PSHM, POPD и POPM. Соответственно, 16-разрядное значение в стек помещается/выталкивается с предварительным уменьшением/последующим увеличением.

5.7.3.3.8 Регистр размера циклического буфера (BK)

Для определения размера блока данных при циклической адресации ARAU использует регистр размера циклического буфера (BK).

5.7.3.3.9 Регистры повторения блока (BRC, RSA, REA*)*

16-разрядный счетчик повтора блоков (BRC) определяет какое число раз должен быть повторен блок кода. 16-разрядный регистр стартового адреса блока повтора (RSA) содержит начальный адрес повторяемого блока памяти программ. 16-разрядный регистр адреса конца блока повтора (REA) содержит информацию о конечном адресе повторяемого блока памяти программ.

5.7.3.3.10 Регистр состояния режима процессора (PMST)

Регистр состояния режима процессора (PMST) управляет конфигурацией памяти устройства 1867ВЦ4Т.

5.7.4 Память ввода/вывода

1867ВЦ4Т в дополнение к областям памяти программ и данных имеет пространство памяти ввода/вывода. Пространство памяти ввода/вывода содержит (64К×16) бит, размещается в адресах 0000h-FFFFh и находится только вне процессора. Для обращения к этой памяти используются две команды – PORTR и PORTW.

5.7.5 Защита программ и данных

В устройстве 1867ВЦ4Т существуют две опции защиты: внутрикристальная защита ПЗУ и защита ПЗУ/ОЗУ. В таблице 4 представлен обзор характеристик защиты информации в ПЗУ/ОЗУ.

Таблица 4 – Защита информации в ПЗУ/ОЗУ

ПЗУ	ПЗУ/ОЗУ
Эмулятор не запускается.	Эмулятор не запускается.
Команды из внутрикристального ПЗУ могут читать данные из внутрикристального ПЗУ	Команды из внутрикристального ПЗУ могут читать данные из внутрикристального ПЗУ
Команды из внутрикристальной ОЗУ или внешние программы не могут читать данные из внутрикристальной ПЗУ и будут читать FFFFh	Команды из внутрикристальной ОЗУ или внешние программы не могут читать данные из внутрикристальной ПЗУ и будут читать FFFFh
Команды из внутрикристального ОЗУ или внешних программ могут выполнять переход во внутрикристальное ПЗУ	Команды из внутрикристального ОЗУ или внешних программ могут выполнять переход во внутрикристальное ПЗУ
Может стартовать или в режиме микропроцессора (MP/MC# = 1), или в режиме микрокомпьютера (MP/MC# = 0) в зависимости от логического уровня на выводе MP/MC#	Может стартовать только в режиме микрокомпьютера (MP/MC# = 0) вне зависимости от логического уровня на выводе MP/MC#
Может программно изменять бит MP/MC# в регистре PMST	Может программно изменять бит MP/MC# в регистре PMST

5.8 Центральное процессорное устройство (CPU)

CPU 1867ВЦ4Т состоит из следующих компонентов:
- 40-разрядное арифметико-логическое устройство (ALU).
- Два 40-разрядных регистра аккумуляторов.
- Циклический сдвигатель с поддержкой диапазона сдвига от 16 до 31.
- Блок умножения/суммирования.
- 16-разрядный временный регистр (T).
- 16-разрядный регистр перехода (TRN).
- Устройство сравнения, выборки и хранения (CSSU).
- Экспоненциальный кодировщик.

Регистры ЦП картированы в памяти.

5.8.1 Статусные и управляющие регистры CPU

В состав CPU входят три статусных и управляющих регистра:
- Регистр состояния (ST0).
- Регистр состояния (ST1).
- Регистр состояния режима процессора (PMST).

5.8.1.1 Регистры состояния (ST0 и ST1)

Индивидуальные биты регистров ST0 и ST1 могут быть установлены или очищены с помощью команд SSBX и RSBX. Например, режим расширения знака установлен командами SSBX 1, SXM или сброшен командами RSBX 1, SXM. Поля ARP, DP и ASM могут быть загружены с использованием команды LD с коротким непосредственным операндом. Поля ASM и DP могут также загружаться значениями памяти данных с использованием команды LD.

Биты регистра ST0 показаны на рисунке 6, в таблице 5 приведено их описание. На рисунке 7 – биты регистра ST1, их описание в таблице 6.

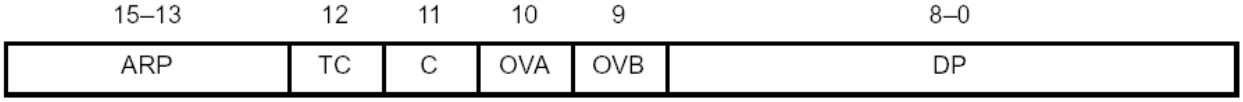


Рисунок 6 – Диаграмма регистра состояния ST0

Таблица 5 – Описание битов регистра состояния ST0

Бит	Имя	Уровень при сбросе	Функции
15-13	ARP	0	Указатель вспомогательного регистра. Это 3-разрядное поле выбирает вспомогательный регистр, для использования в режиме совместимости при косвенной адресации с одним операндом. ARP должен быть всегда установлен в ноль, когда процессор находится в стандартном режиме (CMPT = 0)

Окончание таблицы 5

Бит	Имя	Уровень при сбросе	Функции
12	TC	1	Флаг теста/управления. TC хранит результаты проверки поразрядных операций в арифметико-логическом устройстве ALU. TC можно изменить командами BIT, BITT, CMPM, CMPR и SFTC. Состояние TC (установлен или очищен) определяется выполнением условного перехода, вызова или команды возврата. TC = 1, если выполняются следующие условия: – Бит тестируется командами BIT или BITT как 1. – Условие сравнения, проверяемое командами CMPM, CMPR или CMPS, между значением памяти данных и непосредственным операндом, содержимым AR0 или другого вспомогательного регистра, старшим словом и младшим словом аккумулятора, выполнены. – Биты 31 и 30 аккумулятора, тестируемые командой SFTC, не совпадают
11	C	1	Перенос устанавливается в 1, если результат сложения генерирует перенос; устанавливается в 0, если в результате вычитания генерируется заем. В других случаях сбрасывается после сложения и устанавливается после вычитания, за исключением команд ADD и SUB с 16-разрядным сдвигом. В этих случаях операция добавления (ADD) может только устанавливать, а операция вычитания (SUB) только сбрасывать бит переноса. Перенос и заем определяются 32 разрядом. Команды сдвига и вращения (ROR, ROL, SFTA и SFTL), а также команды MIN, MAX, ABS и NEG также влияют на этот бит
10	OVA	0	Флаг переполнения аккумулятора A. OVA устанавливается в 1, когда происходит переполнение в ALU или в сумматоре умножителя, адресатом результата является аккумулятор A. Как только происходит переполнение, OVA остается установленным в 1 до любого сброса или выполнения любой из команд BC[D], CC[D], RC[D] или команды XC при использовании условий AOV или ANOV. Команда RSBX также может очистить этот бит
9	OVB	0	Флаг переполнения аккумулятора B. OVB устанавливается в 1, когда происходит переполнение в ALU или в сумматоре умножителя, адресатом результата является аккумулятор B. Как только происходит переполнение, OVB остается установленным в 1 до любого сброса или выполнения любой из команд BC[D], CC[D], RC[D] или команды XC при использовании условий BOV или BNOV. Команда RSBX также может очистить этот бит
8-0	DP	0	Указатель страницы памяти данных. Это 9-разрядное поле конкатенируется с семью младшими битами (LSB) слова команды при формировании прямого 16-разрядного адреса для адресации одиночного операнда памяти данных. Эта операция выполняется, если бит CPL статусного регистра ST1 равен 0. Поле DP может быть загружено командой LD коротким непосредственным операндом или из памяти

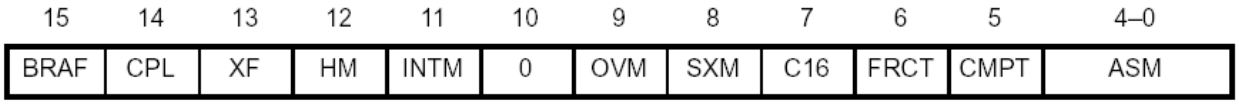


Рисунок 7 – Диаграмма регистра состояния ST1

Таблица 6 – Описание битов регистра состояния ST1

Бит	Имя	Уровень при сбросе	Функции
15	BRAF	0	Флаг активности блока-повтора. BRAF указывает – активно ли в настоящий момент повторение блока. BRAF = 0 – повтор блока неактивен. BRAF сбрасывается, когда счетчик блока повтора (BRC) становится меньше нуля. BRAF = 1 – повтор блока активен. BRAF автоматически устанавливается, когда выполняется команда RPTB
14	CPL	0	Режим компиляции. CPL показывает, какой способ используется при относительной прямой адресации: CPL = 0 – выбран режим прямой адресации с использованием указателя страниц данных (DP). CPL = 1 – выбран режим прямой адресации с использованием указателя вершины стека
13	XF	1	Состояние XF. XF отображает состояние вывода внешнего флага (XF), который является выходом общего назначения. Команда SSBX устанавливает XF, команда RSBX сбрасывает XF
12	HM	0	Режим удержания. Этот бит показывает – продолжает ли процессор выполнение внутренней команды, когда активен сигнал HOLD#: HM = 0 – процессор продолжает выполнение команд из внутренней памяти программ, но его внешний интерфейс находится в третьем состоянии. HM = 1 – процессор прекращает выполнение внутренних команд
11	INTM	1	Режим прерывания. INMT указывает на глобальное маскирование или на разрешение всех прерываний: INTM = 0 – все немаскируемые прерывания разрешены. INTM = 1 – все маскируемые прерывания запрещены. Установка бита INTM производится командой SSBX, а его сброс командой RSBX. INTM устанавливается на 1 в случае сброса или при получении маскируемого прерывания (INTR или внешние прерывания). INTM очищается (устанавливается в 0) при выполнении команд RETE и RETF (возврат из прерывания). Состояние INTM не оказывает влияния на немаскируемые прерывания (RS# и NMI#). Бит INTM не может быть установлен операциями записи в память
10		0	Всегда читается как 0

Окончание таблицы 6

Бит	Имя	Уровень при сбросе	Функции
9	OVM	0	Режим переполнения. OVM определяет каким образом будет обработано переполнение: OVM = 0 – переполненный результат из АЛУ или из умножителя обычным образом записывается приемный аккумулятор. OVM = 1 – при возникновении переполнения в приемный аккумулятор записывается максимально возможное положительное значение (00 7FFF FFFF) или минимально возможное отрицательное значение (FF 8000 0000). Бит OVM устанавливается и сбрасывается командами SSBX и RSBX, соответственно
8	SXM	1	Режим знакового расширения. Бит SXM определяет активен ли режим знакового расширения: SXM = 0 – режим знакового расширения неактивен. SXM = 1 – перед тем как использоваться в ALU, данные расширяются на знак
7	C16	0	Двойной 16-разрядный двойной точности арифметический режим. Бит C16 определяет в каком арифметическом режиме выполняет операции ALU: C16 = 0 – ALU работает в арифметическом режиме двойной точности. C16 = 1 – ALU работает в двойном 16-разрядном арифметическом режиме
6	FRCT	0	Режим дробное/целое. Когда бит FRCT установлен в 1, выходное значение умножителя сдвигается влево на 1 бит для получения дополнительного знакового разряда.
5	CMPT	0	Режим совместимости. CMPT определяет режим совместимости для ARP: CMPT = 0 – ARP не может быть использован в режиме косвенной адресации с отдельным операндом памяти данных. Когда процессор находится в таком режиме, ARP должен быть установлен в 0. CMPT = 1 – ARP обновляется в режиме косвенной адресации операндом памяти данных, за исключением случая, когда выбран вспомогательный регистр 0 (AR0)
4-0	ASM	0	Режим сдвига аккумулятора. Это 5-разрядное поле определяет значение сдвига в диапазоне от – 16 до 15 и кодировку значения двоичного дополнения. Команды с параллельным сохранением STH, STL, ADD, SUB и LD используют эту возможность. ASM может быть загружено из памяти данных или с помощью команды LD, используя короткий непосредственный операнд

5.8.1.2 Регистр состояния режима процессора (PMST)

Регистр PMST загружается командами регистров, отображаемых в памяти, такими как STM. На рисунке 8 показаны отдельные биты регистра, а в таблице 7 представлено их описание.

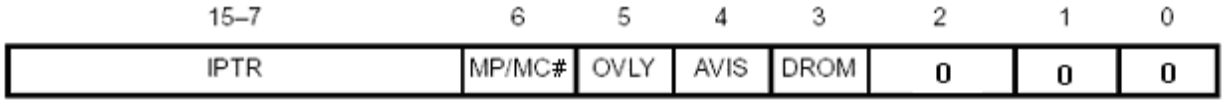


Рисунок 8 – Диаграмма регистра режима процессора (PMST)

Таблица 7 – Описание битов регистра режима процессора PMST

Бит	Имя	Уровень при сбросе	Функции
15-7	IPTR	1FF	Указатель вектора прерывания. 9-разрядное поле IPTR указывает на 128 словную страницу программ, которая хранит вектора прерывания. При сбросе все биты поля IPTR устанавливаются в 1, вектор сброса постоянно расположен по адресу FF80 в пространстве памяти программ. Команда RESET не оказывает влияния на это поле
6	MP/MC#	Значение входа MP/MC#	Режим микропроцессор/микрокомпьютер. MP/MC# включает/отключает внутрикристальную ROM, адресуемую в пространстве памяти программ. MP/MC# = 0 – внутрикристальное ПЗУ доступно для адресации. MP/MC# = 1 – внутрикристальное ПЗУ недоступно. Бит MP/MC# принимает значение, соответствующее логическому уровню, установленному при аппаратном сбросе на входе MP/MC#. До следующего сброса этот вход не опрашивается. ИНСТРУКЦИЯ "RESET" не оказывает влияния на бит MP/MC#. Бит может быть установлен или сброшен программным способом
5	OVLY	0	Перекрытие ОЗУ. Бит OVLY разрешает блокам внутрикристальной DARAM быть отображенными в пространство памяти программ. Возможные значения для бита OVLY: OVLY = 0 – внутрикристальное ОЗУ доступно в пространстве данных, но не в пространстве программ. OVLY = 1 – внутрикристальное ОЗУ доступно в пространстве данных и программ. Однако, страница 0 (адреса с 0 по 7F) не отображается в пространстве программ

Окончание таблицы 7

Бит	Имя	Уровень при сбросе	Функции
4	AVIS	0	Режим видимости адресов. Бит AVIS разрешает/запрещает выставлять внутренние программные адреса на адресных выводах. AVIS = 0 – внешняя шина адреса не отслеживает изменения адресов внутренней программы и показывает последний выведенный адрес. AVIS = 1 – этот режим позволяет отслеживать трассу внутренней программы по состоянию внешних адресных шин, в этом случае, возможно, также декодировать вектор прерывания (в сочетании с IACK#), когда векторы прерывания выбираются из внутрикристальной памяти
3	–	–	Не определен
2	–	–	Не определен
1	–	–	Не определен
0	–	–	Не определен

5.8.2 Арифметико-логическое устройство (ALU)

40-разрядное арифметико-логическое устройство, показанное на рисунке 9, реализует широкий спектр арифметических и логических операций, большинство из которых выполняется за один процессорный цикл. После выполнения операции в ALU результат обычно передается в указанный аккумулятор (A или B). Исключениями являются команды, которые выполняют операции память-в-память (ADDM, ANDM, ORM, XORM).

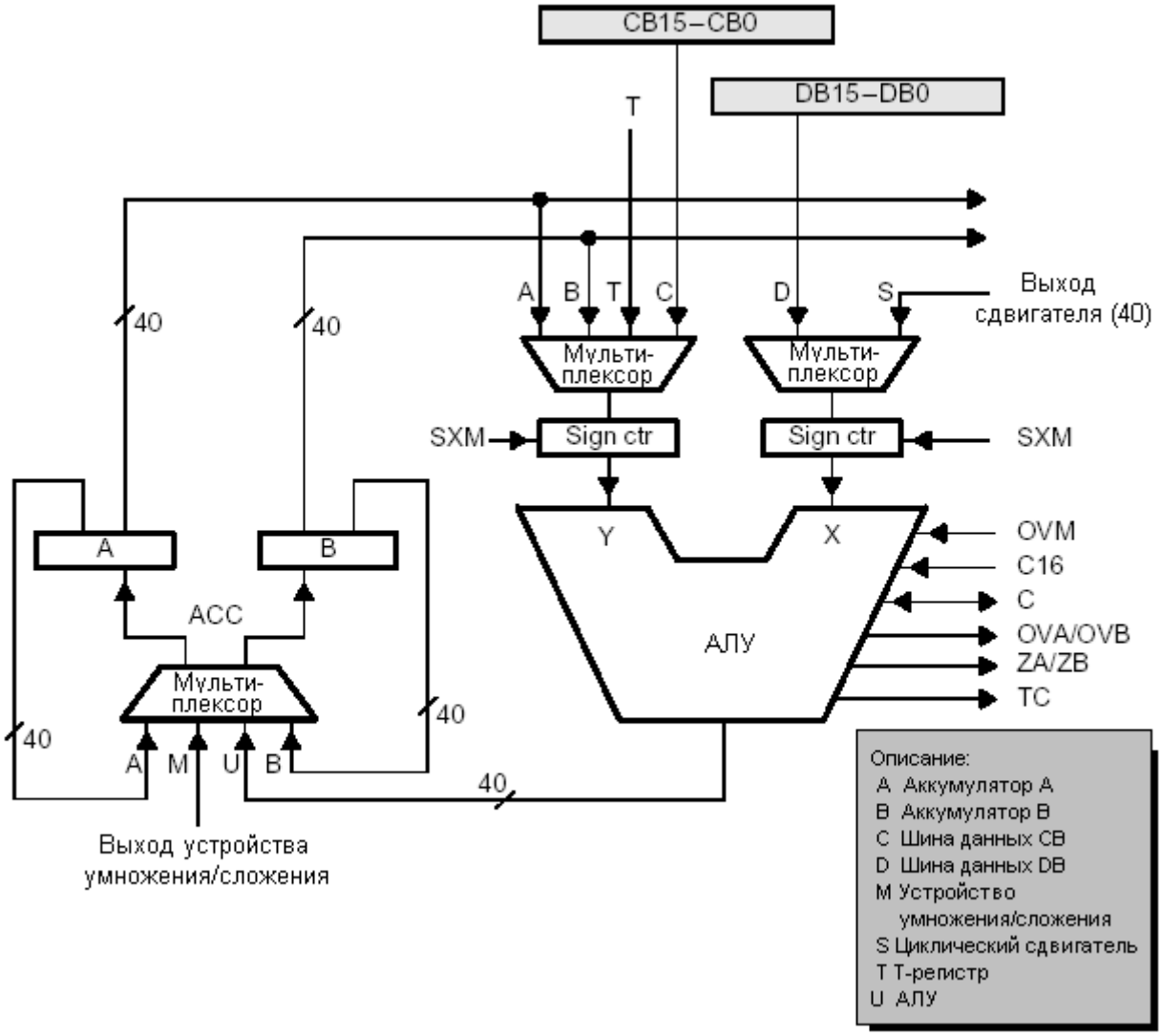


Рисунок 9 – Функциональная диаграмма ALU

5.8.2.1 Входы ALU

Арифметико-логическое устройство получает данные на вход с нескольких источников.

На вход X возможно поступление данных из двух источников:

– Выход сдвигателя (32- или 16-разрядный операнд памяти данных или значение аккумулятора со сдвигом).

– Операнд памяти данных с шины данных CB.

На вход Y ALU могут поступать следующие данные:

– Значение одного из аккумуляторов (A или B).

– Операнд памяти данных с шины CB.

– Значение T-регистра.

Когда через шину данных CB или DB передается 16-разрядный операнд памяти данных, 40-разрядный вход ALU формируется двумя путями:

– Если биты с 15 по 0 содержат операнд памяти данных, биты с 39 по 16 заполняются нулями (SXM = 0) или (SXM = 1).

– Если биты с 31 по 16 содержат операнд памяти данных, то нулями заполняются биты с 15 по 0, а биты с 39 по 32 также заполняются либо нулями (SXM = 0), либо знаковым расширением (SXM = 1).

В таблице 8 показано, что и на какой вход придет при выполнении команды ADD при использовании различных синтаксисов команды. Команда ADD выполняется за один цикл, за исключением случаев 4, 7 и 8, где используются два слова, которые исполняются за два цикла.

Таблица 8 – Выбор входа ALU при выполнении команды ADD

№ п/п	Синтаксис команды	Слов	A	B	DB	CB	Сдвиг
1	ADD *AR1,A	1	√				√
2	ADD *AR3,TS,A	1	√				√
3	ADD *AR2,16,B,A	1		√			√
4	ADD *AR1,8,B,A	2		√			√
5	ADD *AR2,8,B	1		√			√
6	ADD *AR2,*AR3,A	1			√	√	
7	ADD #1234h,6,A,B	2	√				√
8	ADD #1234h,16,A,B	2	√				√
9	ADD A,12,B	1		√			√
10	ADD B,ASM,A	1	√				√
11	DADD *AR2,A,B	1	√				√

5.8.2.2 Обработка переполнения

Логика ALU предотвращает результат от переполнения, сохраняя максимальную (или минимальную) величину. Эта особенность в полной мере применяется при вычислениях фильтров. Эта логика доступна, когда установлен бит переполнения (OVM) в регистре состояния ST1.

В случае переполнения результата:

– Если OVM = 0, результат из ALU загружается без модификации.

– Если OVM = 1, в аккумуляторы загружается максимальное положительное значение (00 7FFF FFFF) или минимальное отрицательное значение (FF 8000 0000) в зависимости от характера (направления) переполнения. Иначе говоря, производится насыщение соответствующего (приемного) аккумулятора.

– Флаг переполнения (OVA/OVB) в регистре состояния ST0 устанавливается для выбранного аккумулятора и остается неизменным, пока не произойдет одно из следующих событий:

— Произведен сброс.

— Условная команда (такая как переход, возврат, вызов или др.) выполняется в условиях переполнения.

— Флаг переполнения (OVA/OVB) очищен.

Примечание – Используя команду SAT, можно произвести насыщение аккумулятора, независимо от значения флага OVM.

5.8.2.3 Бит переноса

Арифметико-логическое устройство имеет связанный бит переноса, который оказывает действие на большинство арифметических действий ALU, включая операции вращения и сдвига. Использование бита переноса позволяет производить эффективные вычисления при арифметических операциях с увеличенной точностью. Бит переноса не затрагивает загрузку аккумулятора, выполнение логических операций или другие

неарифметические команды или команды управления, поэтому он может быть использован для управления переполнением.

Два операнда условий – C и NC – разрешают переходы, вызовы и возвраты и выполняются в соответствии с состоянием бита переноса. Для загрузки бита переноса могут использоваться команды RSBX и SSBX. Бит переноса может быть установлен путем аппаратного сброса.

5.8.2.4 Двойной 16-разрядный режим обработки данных

Для выполнения арифметических операций ALU может работать в специальном двойном 16-разрядном арифметическом режиме, при котором одновременно (в одном цикле) выполняются две 16-разрядные операции (например, два сложения или два вычитания). Этот режим выбирается путем установки поля C16 в ST1. Этот режим особенно полезен для выполнения операций Виттерби добавления/сравнения/выбора.

5.8.3 Аккумуляторы A и B

Аккумуляторы A и B могут быть сконфигурированы как приемные регистры (регистры назначения) для модуля умножения/сложения или ALU. Кроме того, они используются для команд MIN и MAX или для параллельной команды LD ‖ MAC, в которой один аккумулятор загружает данные, а другой выполняет вычисления.

Каждый аккумулятор разбит на три части, как показано на рисунках 10 и 11.

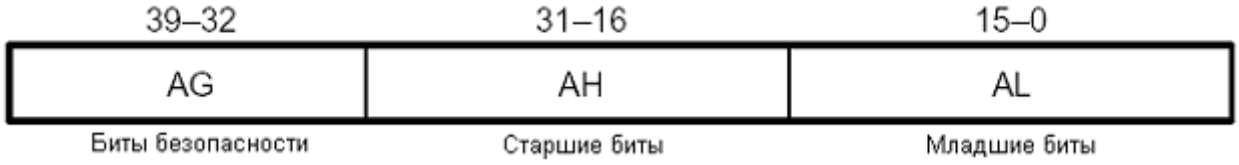


Рисунок 10 – Аккумулятор A

Рисунок 11 – Аккумулятор B

Биты безопасности могут использоваться в качестве заголовка ("верхнее поле" – headmargin) для вычислений. Заголовок позволяет в некоторых случаях избежать переполнения при итерационных вычислениях, таких как автокорреляция.

AG, BG, AH, BH, AL и BL – регистры, отображаемые в памяти, которые могут помещаться и выталкиваться из стека для контекстного сохранения и восстановления с использованием команд PSHM и POPM. Эти регистры так же могут использоваться командами работы с картированными в памяти регистрами (MMR) для адресации к нулевой странице памяти данных. Единственное различие аккумуляторов A и B в том, что биты 32-16 аккумулятора A могут использоваться как вход для множителя в устройстве умножения/сложения.

5.8.3.1 Сохранение содержимого аккумулятора

Содержимое аккумулятора можно сохранить в памяти данных с помощью команд STH, STL, STLM и SACCD или используя команды с параллельным сохранением. Для того чтобы сохранить 16 старших битов аккумулятора (MSB) в памяти со сдвигом, используются команды STH, SACCD и команды с параллельным сохранением. Для операций со сдвигом вправо биты AG и BG сдвигаются, соответственно, в AH и BH. Для операций со сдвигом влево биты AL и BL сдвигаются, соответственно, в AH и BH.

Для хранения 16 младших битов (LSB) со сдвигом в памяти используется команда STL. При сдвиге вправо биты из AH и BH сдвигаются соответственно в AL и BL. При этом младшие биты теряются. Во время выполнения сдвига влево биты AL и BL заполняются нулями. Поскольку операции сдвига выполняются в сдвигателе, содержимое аккумуляторов не изменяется. Пример показывает результат операции сохранения содержимого аккумулятора со сдвигом; аккумулятор A = 0FF 4321 1234.

Пример 1 – Сохранение аккумулятора со сдвигом

```
STH  A,8,TEMP    ; TEMP = 2112h
STH  A,-8,TEMP   ; TEMP = FF43h
STL  A,8,TEMP    ; TEMP = 3400h
STL  A,-8,TEMP   ; TEMP = 2112h
```

5.8.3.2 Сдвиг аккумулятора и операции вращения

Следующие команды используются для сдвига или вращения содержимого аккумулятора с задействованием бита переноса:

- SFTA (арифметический сдвиг).
- SFTL (логический сдвиг).
- SFTC (условный сдвиг).
- ROL (вращение аккумулятора влево).
- ROR (вращение аккумулятора вправо).
- ROLTC (вращение аккумулятора влево с TC).

В командах SFTA и SFTL счетчик сдвига может принимать следующие значения: $-16 \leq SHIFT \leq 15$. Кроме того, команда SFTA зависит от бита SXM. Когда бит SXM = 1 и SHIFT имеет отрицательное значение, команда SFTA выполняет арифметический сдвиг вправо и сохраняет знак аккумулятора. Когда бит SXM = 0, старшие биты аккумулятора MSB заполняются нулями. Команда SFTL не зависит от бита SXM, она выполняет операцию сдвига для битов 31-0, сдвигая нули в старшие или в младшие биты, в зависимости от направления сдвига.

SFTC осуществляет однобитный сдвиг влево, если 31 и 30 биты оба установлены в 1 или в 0, позволяя произвести нормализацию 32 разрядов аккумулятора, путем изъятия старших значащих незнаковых разрядов.

ROL сдвигает каждый бит аккумулятора на одну позицию влево; бит переноса сдвигается в младший бит аккумулятора; значение старшего бита аккумулятора сдвигается в бит переноса и биты безопасности аккумулятора очищаются.

ROR сдвигает каждый бит аккумулятора вправо на один бит; сдвигает значение бита переноса в старший бит аккумулятора; сдвигает значение младшего бита в бит переноса и очищает биты безопасности аккумулятора.

Команда ROLTC (вращение содержимого аккумулятора влево с TC) сдвигает аккумулятор влево и сдвигает бит теста/управления (TC) в младший бит аккумулятора.

5.8.3.3 Специальные команды

Каждый аккумулятор предназначен для выполнения специфических операций в специальных командах приложений с параллельными операциями. Такими приложениями являются: выполнение симметричной КИХ (FIR) – фильтрации с использованием команды FIRS; операции адаптивной фильтрации, использующие команды LMS; вычисление Евклидового расстояния с использованием команды SQDST и другие параллельные операции:

– Команда FIRS выполняет операции для симметричной КИХ-фильтрации с использованием умножения/накопления (MAC) параллельно со сложением.

– Команда LMS обеспечивает выполнение MAC и параллельное сложение с округлением для эффективного обновления коэффициентов в КИХ-фильтре.

– SQDST исполняет MAC и параллельное вычитание для вычисления Евклидовых расстояний.

FIRS перемножает аккумулятор A(32-16) со значением памяти программ и добавляет результат к значению в аккумуляторе B. Одновременно с этим команда складывает операнды памяти Xmem и Ymem, результат сдвигается на 16 разрядов влево и полученное значение загружается в аккумулятор A.

Во время исполнения команды LMS аккумулятор B хранит промежуточные результаты последовательности свертки и коэффициенты фильтра; аккумулятор A обновляет коэффициенты фильтра. Аккумулятор A так же используется как вход для MAC, которое выполняет команды с параллельными операциями за один цикл.

Команда SQDST вычисляет квадрат расстояния между двумя векторами. Аккумулятор A (32-16) возводится в квадрат и добавляется к значению аккумулятора B. Результат сохраняется в аккумуляторе B. В тоже время операнд Ymem вычитается из операнда Xmem и результат заносится в аккумулятор A. Искомый квадрат – значение, записанное в аккумуляторе, перед вычитанием Ymem — Xmem.

5.8.4 Циклический сдвигатель

Циклический сдвигатель используется для выполнения операций масштабирования, таких как:

– Предмасштабирование входного операнда памяти данных или значения аккумулятора перед выполнением операций ALU.

– Выполнение логического или арифметического сдвига значения аккумулятора.

– Нормализация аккумулятора.

– Постмасштабирование аккумулятора перед сохранением значения аккумулятора в памяти данных.

40-разрядный сдвигатель (рисунок 12) подключен следующим образом:

– Вход соединен с:
  — Шиной DB для 16-разрядного операнда данных
  — Шинами DB и CD для 32-разрядного операнда данных
  — Любым из двух 40-разрядных аккумуляторов
– Выход соединен с:
  — Одним из входов ALU
  — С шиной EB через устройство выбора/записи старших/младших слов.

Бит SXM управляет знаковым/беззнаковым расширением операнда данных; когда бит установлен, выполняется функция расширения знака. Некоторые команды, такие как ADDS, LDU, MAC и SUBS работают с операндами памяти без знака и не могут выполнять функцию расширения знака независимо от значения бита SXM.

Счетчик сдвига определяет количество битов, на которое надо произвести сдвиг. Положительное значение означает сдвиг влево, отрицательное – вправо.

Счетчик сдвига в некоторых случаях определяется как значение в дополнительном коде, в зависимости от типа команды. В режиме сдвига аккумулятора непосредственный операнд, поле ASM регистра ST1 или T может использоваться для указания величины сдвига:

– 4- или 5-разрядное непосредственное значение, указанное в операнде команды представляет собой значение сдвига в диапазоне от 16 до 15. Например:

```
ADD     A,-4,B      ; Добавить аккумулятор A (со сдвигом вправо на 4
                    разряда) в аккумулятор B (одно слово, один цикл)
SFTL    A,+8        ; Сдвиг (логический) аккумулятора A на восемь разрядов
                    влево (одно слово, один цикл)
```

– Значение ASM определяет величину сдвига в диапазоне от 16 до 15 и может быть загружено командой LD (с непосредственным операндом или операндом памяти данных). Например:

```
ADD     A,ASM,B     ; Добавить аккумулятор A к аккумулятору B; со сдвигом,
                    указанным в ASM
```

– Шесть младших битов регистра T указывают на значение сдвига в диапазоне от 16 до 31. Например:

```
NORM    A           ; Нормализовать аккумулятор A (T содержит значение
                    показателя степени)
```

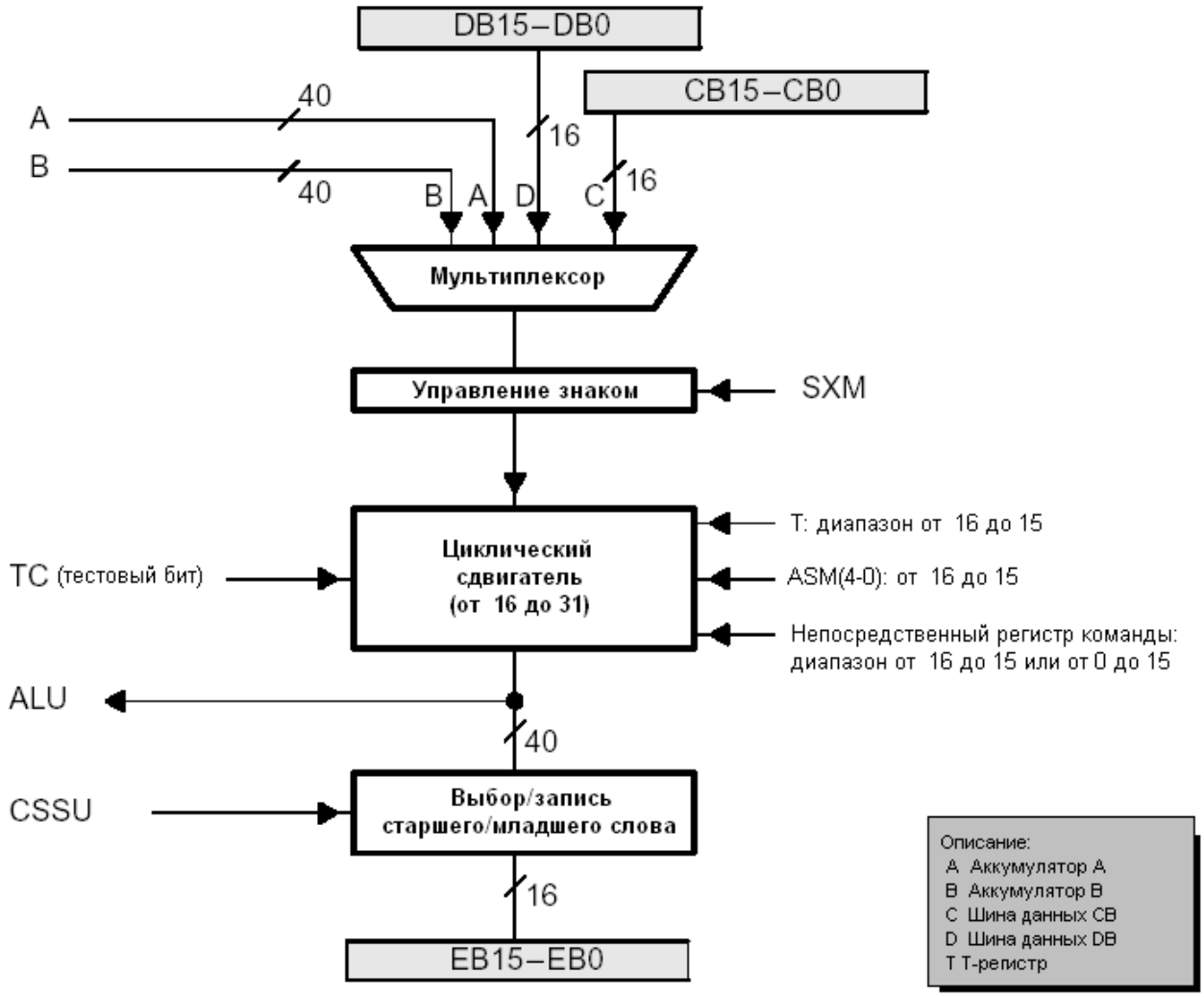


Рисунок 12 – Функциональная диаграмма циклического сдвигателя

5.8.5 Умножитель и сумматор

1867ВЦ4Т имеет 17×17-разрядный аппаратный умножитель, объединенный с 40-разрядным сумматором. Это устройство обеспечивает выполнение умножения и накопления (MAC) за один цикл. Устройство умножения/сложения показано на рисунке 13.

Умножитель может производить операции со знаком, без знака, а так же знаковое/беззнаковое умножение со следующими ограничениями:

– Для операции умножения со знаком каждый 16-разрядный операнд памяти представляется в виде 17-разрядного слова с расширением знака.

– Для умножения без знака в каждом операнде старший бит (MSB) (16-й) устанавливается в 0.

– Для умножения с/без знака один из операндов представлен со знаком, а второй расширяется путем добавления 0 в старший бит.

Значение на выходе умножителя может быть сдвинуто на один бит влево, чтобы скомпенсировать знаковый разряд, при перемножении двух 16-разрядных чисел в дополнительном коде в режиме дробное/целое. (Режим дробное/целое выбирается в случае, когда бит FRCT в ST1 равен 1).

Сумматор в устройстве умножения/сложения содержит детектор нуля, устройство округления и логику переполнения/насыщения. Округление состоит в добавлении 2^{15} к

результату и последующей очистке 16 младших разрядов в указанном аккумуляторе. Округление выполняется некоторыми командами умножения, MAC и умножения/вычитания (MAS), когда суффикс R включен в команду. Команда LMS также использует округление для минимизации ошибок квантования при вычислении коэффициентов.

Данные на входы сумматора приходят с выхода умножителя и выхода одного из аккумуляторов. Как только устройство выполнит любую операцию умножения, результат передается в аккумулятор A или B.

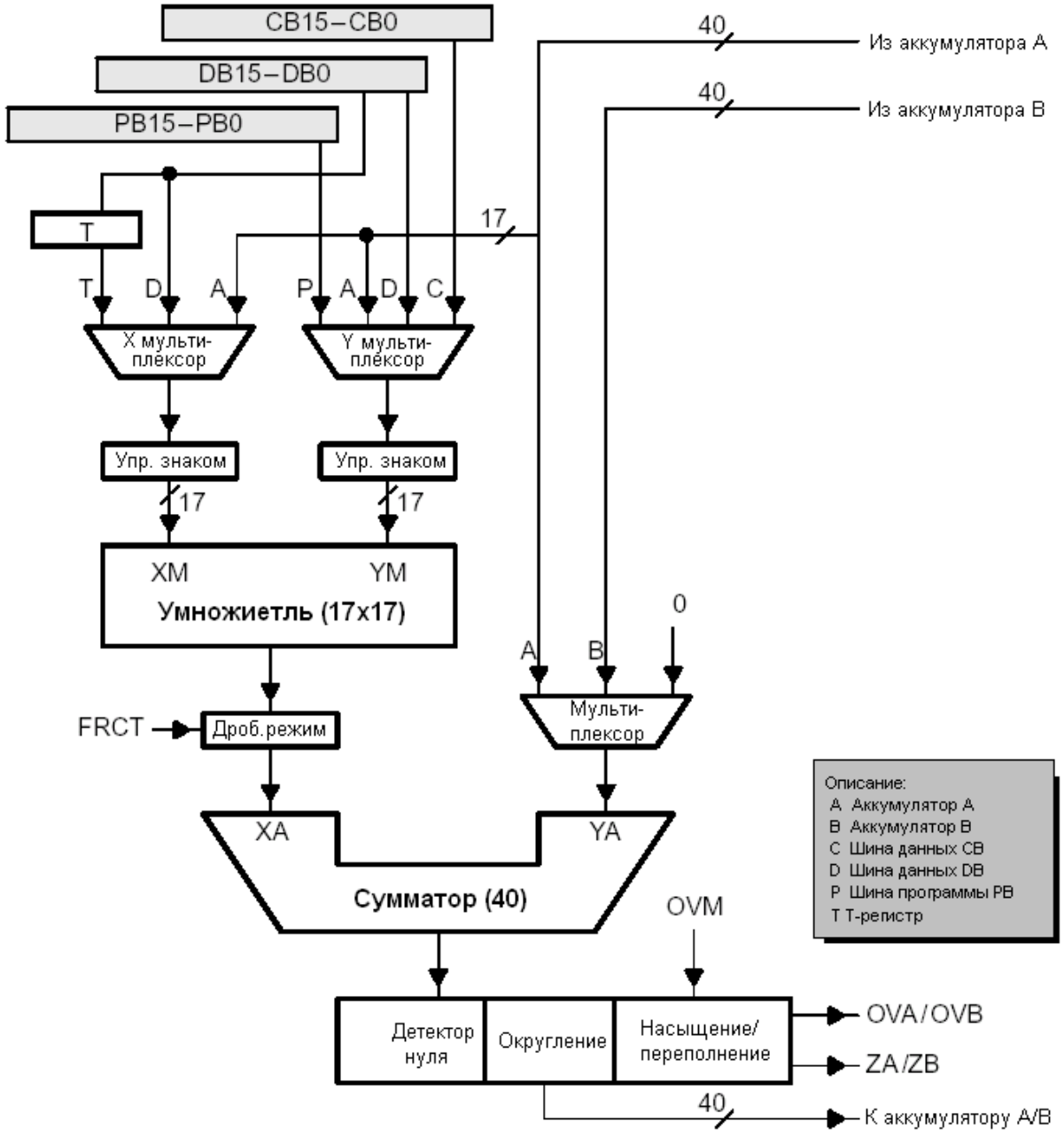


Рисунок 13 – Функциональная диаграмма умножителя/сумматора

5.8.5.1 Входы умножителя

На вход XM умножителя может приходить значение из любого следующего источника:

- Временный регистр T.
- Операнд памяти данных с шины данных DB.
- Биты 32-16 аккумулятора A.

На вход YM умножителя может приходить значение из любого следующего источника:

- Операнд памяти данных с шины DB.
- Операнд памяти данных с шины CB.
- Операнд памяти программ с шины программ PB.
- Биты 32-16 аккумулятора A.

В таблице 9 показано, какие входы используются для выполнения различных команд. Существуют девять комбинаций использования входов умножителя.

Для команд, в которых для одного входа используется содержимое регистра T, на второй вход может подаваться непосредственное значение из памяти данных через шину данных (DB) или значение из аккумулятора A.

Для команд, использующих адресацию одиночным операндом памяти данных, один операнд передается умножителю с шины DB. Второй операнд может приходить из регистра T, как непосредственное значение или из памяти программ через шину PB, или из аккумулятора A.

Для команд, использующих адресацию двойным операндом памяти данных, данные умножителю передаются через шины DB и CB.

Последние два случая используются в командах FIRS, SQUR и SQDST. Команда FIRS использует значения из входов шины PB и аккумулятора A. SQUR и SQDTS оба сомножителя получают из аккумулятора A.

Таблица 9 – Выбор входа для различных команд

№	Тип команды	Вход X умножителя			Вход Y умножителя			
		T	DB	A	PB	CB	DB	A
1	MPY #1234h, A	√					√	
2	MPY[R] *AR2, A	√					√	
3	MPYA B	√						√
4	MACP *AR2, pmad, A		√		√			
5	MPY *AR2, *AR3, B		√			√		
6	SQUR *AR2, B		√				√	
7	MPYA *AR2		√					√
8	FIRS *AR2, *AR3, pmad			√	√			
9	SQUR A, B			√				√

Регистр T обеспечивает один операнд для команд умножения и умножения/накопления; другим операндом является одиночный операнд памяти данных. Регистр T также обеспечивает операнд для команд умножения с параллельной загрузкой или параллельным сохранением, таких как LD ‖ MAC, LD ‖ MAS, ST ‖ MAC, ST ‖ MAS и ST ‖ MPY. T регистр может непосредственно загружаться командами, которые поддерживают режим MMR-адресации или неявным способом через операции умножения.

Так как биты A(32-16) могут быть входом умножителя, некоторые последовательности, которые требуют сохранения результата одного вычисления в памяти и передачи этого результата умножителю, могут быть выполнены очень быстро. Для некоторых специфических команд приложений (FIRS, SQDST, ABDST и POLY) содержимое аккумулятора A вычисляется ALU и затем передается на вход умножителя.

5.8.5.2 Команды умножения с накоплением (MAC)

Команды MAC используют весь диапазон вычислительных возможностей умножителя и одновременно работают с двумя операндами. Сложные арифметические операции могут быть выполнены модулем умножения/сложения в течение одного цикла.

В командах MAC, MAS и MACSU с адресацией с двойным операндом памяти данных, данные могут передаваться в течение каждого цикла через шины CB и DB, умножение и сложение происходит в течение цикла. Адреса данных для этих операндов генерируются ARAU0 и ARAU1 – арифметическими устройствами вспомогательных регистров.

В командах MACD и MACP данные могут передаваться к умножителю в течение каждого цикла через шины DB и PB. Шина DB извлекает данные из памяти данных, а шина PB - коэффициенты из памяти программ. Когда MACD и MACP используются в режиме повтора (RPT и RPTZ), они выполняют одноцикловые операции MAC с последовательным доступом к данным и коэффициентам. Адреса данных генерируются ARAU0 и регистром адреса программ (PAR). Адреса памяти данных модифицируются ARAU0 в соответствии с операндом памяти данных в режиме косвенной адресации; адрес памяти программ увеличивается модулем PAGEN.

Повтор команд MAC и MACP с циклической адресацией обеспечивает реализацию фильтра. Команда FIRS реализует эффективную симметричную структуру для FIR-фильтра, используя циклическую адресацию.

Команды MPYU и MACSU облегчают выполнение арифметических операций с увеличенной точностью. Команда MPYU выполняет беззнаковое умножение. Беззнаковое содержимое регистра T умножается на беззнаковое значение памяти данных и результат помещается в указанный аккумулятор. Команда MACSU выполняет знаковое/беззнаковое умножение и сложение. Беззнаковое значение, содержащееся в памяти данных, умножается на значение со знаком, так же расположенное в памяти данных и результат помещается в аккумулятор. Эта операция позволяет работать с операндами, размер которых больше 16 разрядов; такие операнды делятся на два 16-разрядных слова и обрабатываются отдельно для генерации результата более 32 разрядов.

Команды возведение в квадрат/сложение (SQURA) и возведение в квадрат/вычитание (SQURS) подают данные на оба входа умножителя для возведения в квадрат, далее результат добавляется к (при выполнении команды SQURA) или вычитается из (SQURS) аккумулятора. Команда SQUR возводит в квадрат значение из памяти данных или содержимое аккумулятора A.

5.8.6 Устройство сравнения, выборки и хранения (CSSU)

Устройство сравнения, выборки и хранения (CSSU) – специальный аппаратный модуль, предназначенный для выполнения операций сложение/сравнение/выборка (ACS) при выполнении оператора Виттерби. На рисунке 14 показано устройство сравнения, выборки и хранения, которое используется вместе с ALU для выполнения быстрых ACS-операций.

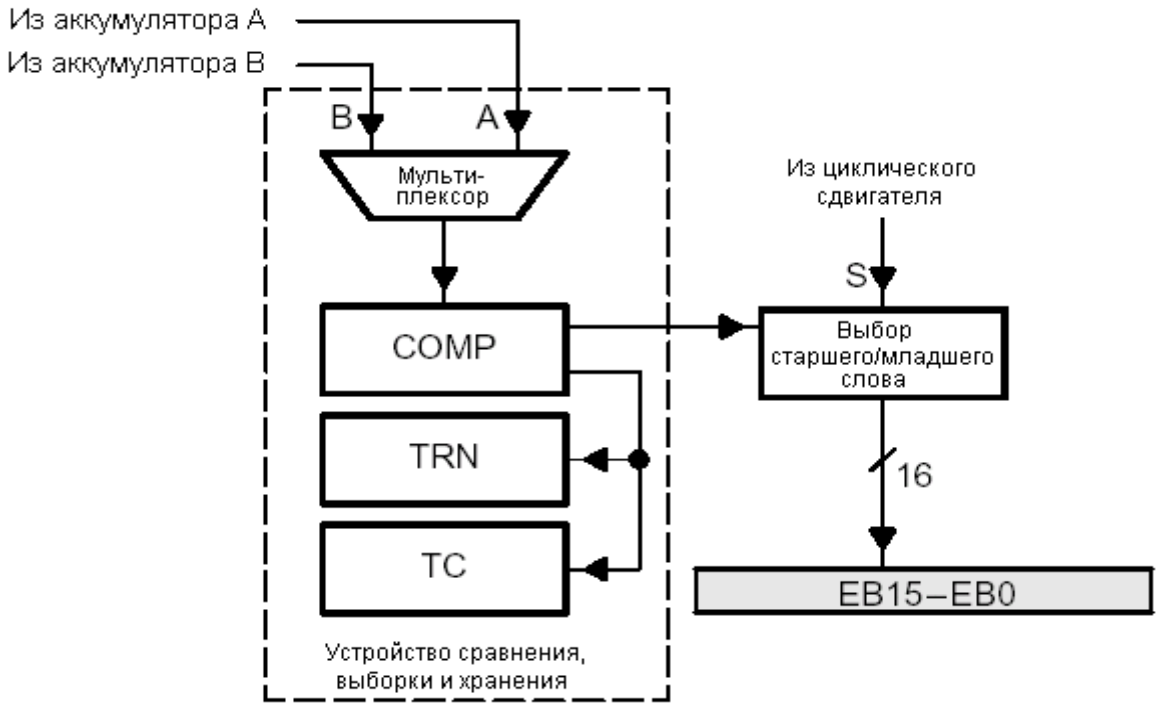


Рисунок 14 – Устройство сравнения, выборки и хранения (CSSU)

CSSU позволяет устройству 1867ВЦ4Т выполнять различные алгоритмы Виттерби ("бабочка"), используемые при декодировании и корректировке.

Функции сложения оператора Виттерби выполняет ALU (рисунок 15). Эта функция состоит из функции двойного сложения (Met1 ± D1 и Met2 ± D2). Двойное сложение выполняется в один машинный цикл, если ALU сконфигурировано для двойного 16-разрядного режима путем установки бита C16 в ST1. Если ALU сконфигурировано в этом режиме, все команды с длинными словами (32-разрядными) становятся двойными 16-разрядными командами.

Регистр T связан с входом ALU (как двойной 16-разрядный операнд) и используется как локальный регистр для хранения информации для минимизации времени доступа к памяти. В таблице 10 приведены команды, исполняющие двойные 16-разрядные действия ALU.

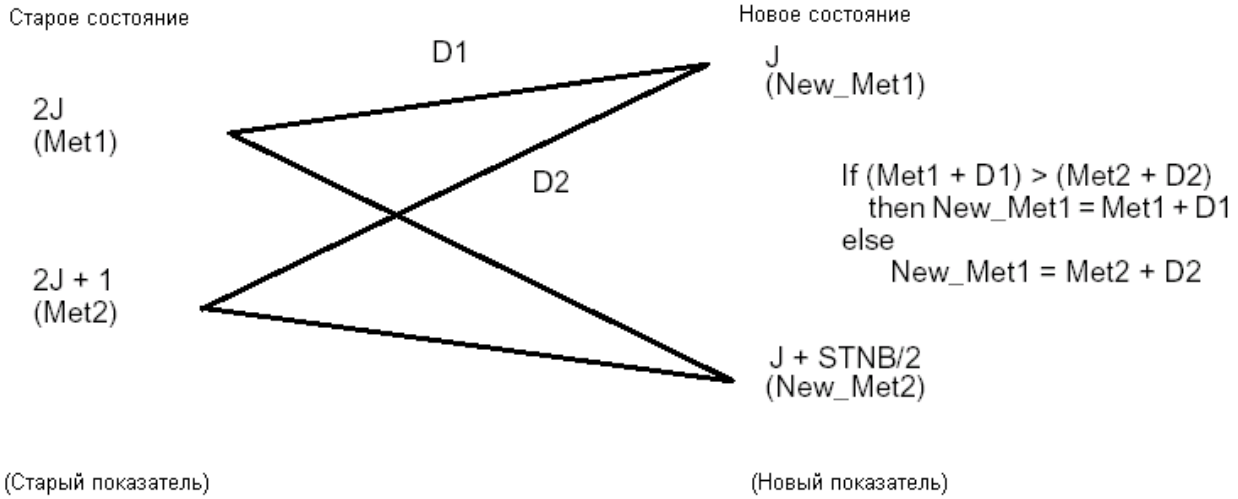


Рисунок 15 – Оператор Виттерби

Таблица 10 – Операции ALU в двойном 16-разрядном режиме

Команда		Функция (двойной 16-разрядный режим)
DADD Lmem, src [, dst]		src(31-16) + Lmem(31-16) → dst(39-16) src(15-0) + Lmem(15-0) → dst(15-0)
DADST Lmem, dst		Lmem(31-16) + T → dst(39-16) Lmem(15-0) – T → dst(15-0)
DRSUB Lmem, src		Lmem(31-16) – scr(31-16) → scr(39-16) Lmem(15-0) – scr(15-0) → scr(15-0)
DSADT Lmem, dst		Lmem(31-16) – T → dst(39-16) Lmem(15-0) + T → dst (15-0)
DSUB Lmem, src		src(31-16) – Lmem(31-16) → src(39-16) src(15-0) – Lmem(15-0) → src(15-0)
DSUBT Lmem, dst		Lmem(31-16) – T → dst(39-16) Lmem(15-0) – T → dst(15-0)
Условные обозначения:	→ Lmem src dst x(n-m)	Сохранение в; Длинное (32-разрядное) значение памяти данных; Источник – аккумулятор (A или B); Назначение – аккумулятор (A или B); Чтение x(разряды от n до m).

CSSU осуществляет операцию сравнения и выборки через команду CMPS, компаратор и 16-разрядный регистр переноса (TRN). Эта операция сравнивает две 16-разрядные части указанного аккумулятора и результат сдвигает в бит 0 в регистре TRN. Этот результат также сохраняется в бит TC в регистре ST0. Основываясь на решении, соответствующая 16-разряднная часть аккумулятора сохраняется в памяти данных. Пример 2 иллюстрирует выполнение командой CMPS операции сравнения и выборки.

Пример 2 – Выполнение команды CMPS

```
CMPS  B,*AR3        ;Если (B(31-16)>B(15-0)), то
                    ;B(31-16) → (*AR3); TRN << 1; 0 → TRN (0);
                    ;0 → TC}
                    ; иначе B(15-0) → (*AR3); TRN << 1;
                    ;1 → TRN (0); 1 → TC;
```

Регистр TRN содержит информацию о путях переноса выбранного значения в новое состояние. Эта информация может быть использована для нахождения оптимальных путей возврата подпрограммы, результатами работы которой является расшифровка кода.

5.8.7 Устройство вычисления экспоненты

Экспоненциальный кодировщик – прикладное аппаратное устройство, позволяющее за один машинный цикл выполнять команду EXP (см. рисунок 16). По команде EXP значение экспоненты в аккумуляторе может быть сохранено в регистре T в дополнительном коде в диапазоне от 8 до 31. Экспонента определяется как число избыточных битов 8, что соответствует количеству сдвигов, которые необходимо выполнить в аккумуляторе, чтобы избавиться от незначащих знаковых битов. Если значение аккумулятора превышает 32 разряда, то результат такой операции принимает отрицательное значение.

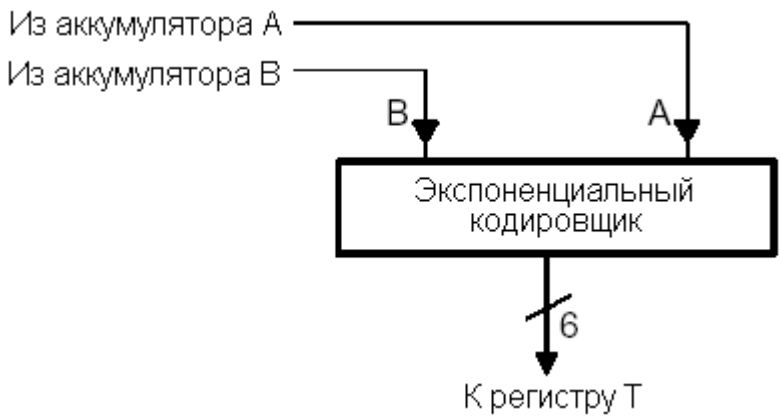


Рисунок 16 – Экспоненциальный кодировщик

Команды EXP и NORM используют кодировщик экспоненты для эффективной нормализации содержимого аккумулятора. Команда NORM обеспечивает сдвиг содержимого аккумулятора на количество битов, определяемых содержимым регистра T, за один машинный цикл. Отрицательное значение в регистре T будет означать сдвиг содержимого аккумулятора вправо, что позволяет нормализовать любое значение аккумулятора вне 32-битного диапазона. Пример 3 демонстрирует нормализацию аккумулятора A.

Пример 3 – Нормализация аккумулятора A

```
EXP   A              ;(число старших избыточных битов 8) → T
ST    T,   EXPONENT  ;Хранение экспоненты (T) в памяти данных
NORM  A              ;Нормализация аккумулятора A, (A) << (T)
```

5.9 Адресация данных

Устройство 1867ВЦ4Т поддерживает семь основных режимов адресации:

– *Непосредственная адресация* использует команды с кодировкой фиксированного значения.

– *Абсолютная адресация* использует команды с кодировкой фиксированного адреса.

– *Адресация аккумулятором* использует аккумулятор для доступа к данным в памяти программы.

– *Прямая адресация* использует в качестве адреса переменную, вычисленную с использованием адресного поля команды (семь бит) и 9-разрядного поля DP регистра состояний ST0 или указателя вершины стека.

– *Косвенная адресация* для доступа к памяти использует вспомогательные регистры.

– *Адресация к регистрам, отображенным в памяти*, модифицирует регистры, картированные в памяти, не изменяя текущее значение DP или текущее значение SP.

– *Стековая адресация* управляет добавлением и извлечением элементов в/из системного стека.

5.9.1 Непосредственная адресация

При непосредственной адресации синтаксис команды содержит специфические значения операнда. При такой адресации в командах могут кодироваться два типа значений:

– *Короткое непосредственное значение* может быть длинной 3, 5, 8 или 9 бит.

– *Длинное непосредственное значение* всегда имеет длину 16 бит.

Непосредственные значения могут использоваться в командах из одного или двух слов. 3-, 5-, 8- или 9-разрядные значения используются в однословных командах; 16-разрядное значение используется в двухсловных командах.

Длина непосредственного значения зависит от типа используемой команды. В таблице 11 приведены команды в зависимости от длины констант, используемых в них.

Таблица 11 – Команды для непосредственной адресации и используемые в них константы

3- и 5-разрядные константы	8-разрядная константа	9-разрядная константа	16-разрядная константа	
LD	FRAME	LD	ADD	ORM
	LD		ADDM	RPT
	RPT		AND	RPTZ
			ANDM	ST
			BITF	STM
			CMPM	SUB
			LD	XOR
			MAC	XORM
			OR	

В тексте программы константа помечается символом # перед своим символьным или числовым значением. Например, команда загрузки аккумулятора A шестнадцатеричной константой 80h записывается так:

```
LD   #80h,   A
```

На рисунке 17 представлено распределение бит в команде RPT, использующей короткое непосредственное значение (#K). Старшая половина команды (биты 8-15) используется для кодирования операции RPT, а младший байт отводится для кодирования самой константы.

На рисунке 18 представлена двухсловная команда RPT, использующая длинное непосредственное значение. Код операции команды занимает первое слово, а длинная константа – второе.

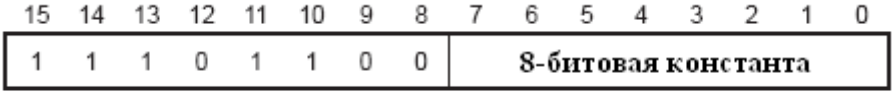


Рисунок 17 – Однословная команда RPT с адресацией, использующей короткое непосредственное значение

Рисунок 18 – Двухсловная команда RPT с адресацией, использующей 16-разрядное длинное непосредственное значение

5.9.2 Абсолютная адресация

В устройстве 1867ВЦ4Т используют четыре типа абсолютной адресации:

- Адресация к памяти данных (*dmad*):
  - MVDK *Smem, dmad.*
  - MVDM *dmad, MMR.*
  - MVKD *dmad, Smem.*
  - MVMD *MMR, dmad.*
- Адресация к памяти программ (*pmad*)
  - FIRS *Xmem, Ymem, pmad.*
  - MACD *Smem, pmad, src.*
  - MACP *Smem, pmad, src.*
  - MVDP *Smem, pmad.*
  - MVPD *pmad, Smem.*
- Адресация к портам ввода/вывода (*PA*)
  - PORTR *PA, Smem.*
  - PORTW *Smem, PA.*
- *(*lk*) адресация использует все команды, которые поддерживают использование одиночного операнда памяти данных (*Smem*).

Абсолютная адресация всегда использует 16-разрядную кодировку, поэтому все команды, которые используются для абсолютной адресации, состоят из двух слов.

5.9.2.1 dmad адресация

При *dmad* адресации используется специальное значение для точного определения адреса в пространстве данных.

В синтаксисе команды для *dmad* адресации используется символ или число, определяющее адрес в пространстве данных. Например, для копирования значения, содержащегося по адресу с меткой SAMPLE в пространстве данных, в ячейку памяти данных, адрес которой содержится в регистре AR5, необходимо задать следующую команду:

```
MVKD SAMPLE, *AR5
```

В этом примере адрес с меткой SAMPLE является *dmad* значением.

5.9.2.2 pmad адресация

При *pmad* адресации используется специальное значение для точного определения адреса в пространстве программ.

Синтаксис команды для *pmad* адресации использует символ или число, определяющее адрес в программном пространстве. Например, для копирования слова, содержащегося в области памяти программ по адресу с меткой TABLE, в ячейку памяти данных, адрес которой содержится в регистре AR7, необходимо задать следующую команду:

```
MVPD      TABLE, *AR7
```

В этом примере адрес с меткой TABLE является *pmad* значением.

5.9.2.3 PA адресация

При *PA* адресации используется специальное значение для точного определения адреса внешних портов ввода/вывода (I/O).

Синтаксис команды для *PA* адресации использует символ или число, определяющее адрес в программном пространстве. Например, для копирования значения из I/O порта с адресом FIFO в ячейку памяти данных, адрес которой содержится в регистре AR5, необходимо задать следующую команду:

```
PORTR     FIFO, *AR5
```

В этом примере FIFO относится к адресу порта.

5.9.2.4 *(lk) адресация

**(lk)* адресация использует специальное значение для точного определения адреса в пространстве данных.

В синтаксисе команды для **(lk)* адресации используется символ или число, определяющее адрес в пространстве данных. Например, для загрузки аккумулятора A значением, содержащимся по адресу BUFFER в пространстве данных, необходимо записать команду:

```
LD   *(BUFFER), A
```

Синтаксис для **(lk)* адресации разрешен для всех команд, использующих *Smem* адресацию для доступа к любой области пространства данных без изменения DP или инициализации вспомогательных регистров (ARx). Когда используется этот тип абсолютной адресации, длина команды увеличивается на одно слово. Например, 1-словная команда становится командой, состоящей из двух слов, или 2-словная команда становится 3-словной. Добавление одного слова в команду оказывает влияние на применимость команды после задержанных инструкций.

Команды, использующие абсолютную **(lk)* адресацию, не могут быть применены совместно с командами однократного повтора (RPT, RPTZ).

5.9.3 Адресация через аккумулятор

В этом типе адресации значение в аккумуляторе принимается как адрес.

Использовать содержимое аккумулятора как адрес позволяют две команды:

– READA *Smem*;

– WRITA *Smem*.

Команда READA перемещает слово из ячейки памяти программ, указываемой аккумулятором A, в ячейку памяти данных, определяемую одиночным операндом памяти данных (*Smem*).

WRITA перемещает слово из ячейки памяти данных, определяемой операндом команды *Smem,* в ячейку памяти программ, адрес которой содержится в аккумуляторе A.

В режиме повтора возможно приращение содержимого аккумулятора A.

5.9.4 Прямая адресация

При прямой адресации команда содержит семь младших битов адреса памяти данных (*dma*). 7 битов *dma* – это адрес смещения, который вместе с базовым адресом из указателя страниц данных (DP) или указателя стека (SP) формируют полный адрес памяти данных, состоящий из 16 разрядов. Используя этот тип адресации, можно получить доступ к любой из 128 ячеек текущей страницы памяти данных в произвольном порядке, не изменяя при этом значение DP или SP.

Для генерации фактического адреса может использоваться комбинация *dma* с DP или SP. Бит режима компиляции (CPL), расположенный в регистре состояния ST1, выбирает, какой из следующих методов будет использован для генерации адреса:

– Когда CPL = 0, поле *dma* конкатенируется с 9-разрядным полем указателя страниц данных (DP) для формирования 16-разрядного адреса памяти данных.

– Когда CPL = 1, 16-разрядный адрес памяти данных формируется сложением (положительное смещение) *dma* с SP.

Синтаксис команд прямой адресации использует символ или число для указания относительного значения (смещения). Например, для добавления содержимого ячейки памяти, определенной как SAMPLE в аккумулятор B, при условии корректного базового адреса в DP (CPL = 0) или SP (CPL = 1), необходимо записать:

```
ADD  SAMPLE, B
```

Семь младших битов адреса SAMPLE содержатся в слове команды.

На рисунке 19 показан формат кода операции (opcode) для команд, используемых при прямой адресации. Таблица 12 описывает биты команд прямой адресации. Рисунок 20 иллюстрирует, как формируются 16-разрядный адрес данных при прямой адресации.

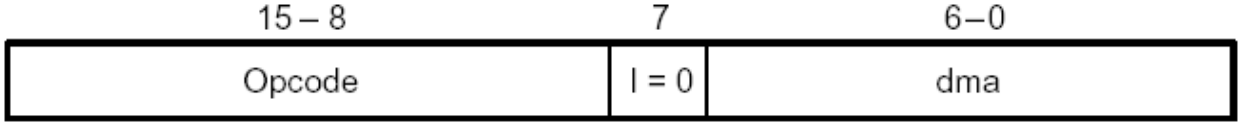


Рисунок 19 – Формат команды прямой адресации

Таблица 12 – Описание битов команд прямой адресации

Бит	Название	Выполняемая функция
15-8	Opcode	Это 8-разрядное поле содержит код операции для команды
7	I	I = 0, означает, что выбран режим прямой адресации
6-0	dma	Это 7-разрядное поле содержит относительный адрес памяти данных для команды

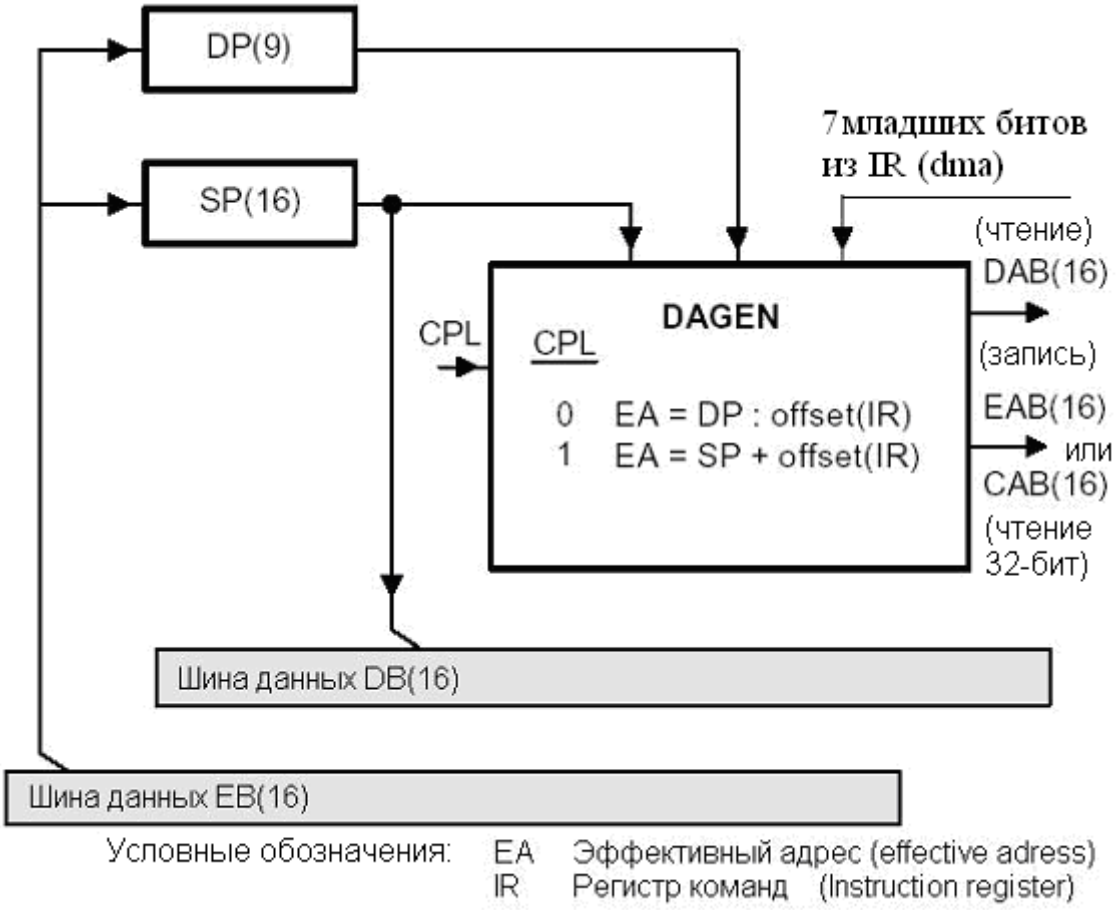


Рисунок 20 – Блок-диаграмма прямой адресации

5.9.4.1 Прямая адресация через указатель страниц (DP)

При DP прямой адресации 7-разрядное поле *dma* в регистре команд (IR) конкатенируется с 9-разрядным содержимым DP для формирования адреса. Рисунок 21 показывает, как два значения формируют результирующий адрес.

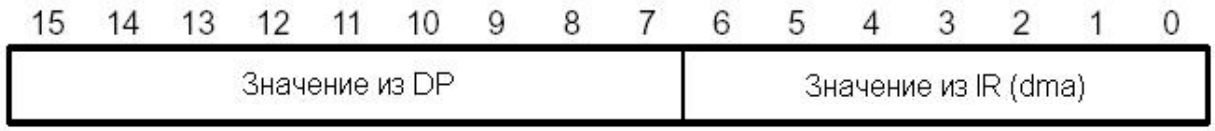


Рисунок 21 – Прямая DP-адресация

Прямая DP-адресация делит память на 512 страниц, поскольку диапазон указателя страниц данных (DP) от 0 до 511 (2^9 — 1). Каждая страница имеет 128 адресуемых слов, поскольку диапазон *dma* от 0 до 127 (2^7 — 1).

DP указывает на одну из 512 возможных страниц памяти данных, состоящую из 128 слов; а *dma* - на конкретное слово выбранной страницы. Единственное различие между указанием на позицию 0 на странице 1 и на позицию 0 на странице 2 состоит в содержимом DP. DP загружается командой LD.

5.9.4.2 Прямая адресация через указатель стека (SP)

При использовании прямой SP-адресации 7-разрядное поле *dma* в регистре команд (IR) прибавляется в виде положительного смещения к значению SP для получения результирующего 16-разрядного адреса памяти данных. На рисунок 22 показано, как эти два числа образуют конечное значение.

Рисунок 22 – Прямая SP-адресация

SP указывает на любой адрес в памяти, а *dma* - на определенное место на странице, позволяя обеспечить доступ к непрерывному 128-словному (2^7 — 1) блоку в памяти, начиная с любого базового адреса.

SP так же позволяет работать со стеком (добавлять/удалять значения).

5.9.5 Косвенная адресация

Режим косвенной адресации позволяет процессору обратиться к любой ячейке памяти в пространстве объемом (64K×16) бит, при этом адрес операнда располагается в одном из восьми 16-разрядных вспомогательных регистров (AR0-AR7), расположенных на нулевой странице памяти данных. Косвенная адресация используется главным образом для выполнения вычислений над последовательно расположенными в памяти операндами.

При использовании косвенной адресации через вспомогательный регистр адрес операнда может увеличиваться (уменьшаться) на единицу или индексироваться при каждом обращении. Специальные режимы позволяют использовать циклическую и бит-реверсную адресацию. Размер циклического буфера задается в специальном регистре размера циклического буфера BK. Регистр AR0 может быть использован для режимов индексирования и для бит-реверсной адресации.

Косвенная адресация достаточно гибка и используется не только для того, чтобы читать или записывать отдельные 16-разрядные операнды из памяти данных одной командой, но и для осуществления доступа к двум областям памяти данных с помощью одной команды. Доступ к двум областям памяти данных включает чтение двух независимых ячеек памяти, чтение и запись двух последовательных ячеек в памяти и чтение из одной ячейки памяти одновременно с записью в другую ячейку.

5.9.5.1 Косвенная адресация с одним операндом

На рисунке 23 представлен формат команды с косвенной адресацией для одного операнда памяти данных *Smem*. Таблица 13 описывает битные поля команды.

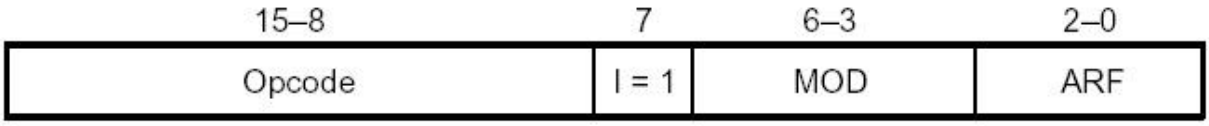


Рисунок 23 – Формат команды косвенной адресации для одиночного операнда памяти данных

Таблица 13 – Описание битов команды косвенной адресации с одиночным операндом памяти данных

Бит	Название	Выполняемая функция
15-8	Opcode	Это 8-разрядное поле содержит код операции для команды
7	I	I = 1, означает, что выбран режим косвенной адресации
6-3	MOD	Это 4-разрядное поле определяет тип косвенной адресации
2-0	ARF	Это вспомогательное 3-разрядное поле определяет вспомогательный регистр, который будет использоваться для адресации. ARF зависит от бита режима совместимости (CMPT) в регистре ST1: CMPT = 0 – стандартный режим. В стандартном режиме ARF всегда определяет вспомогательный регистр, независимо от значения ARP. ARP не обновляется. Когда процессор работает в этом режиме, ARP должен быть установлен в ноль. CMPT = 1 – режим совместимости. В режиме совместимости ARP выбирает вспомогательный регистр, если ARF = 0. В других случаях ARF выбирает вспомогательный регистр и по выполнению доступа значение ARF загружается в ARP. В команде запись *AR0 указывает на то, что этот вспомогательный регистр выбирается с помощью ARP в режиме совместимости

5.9.5.2 Арифметическое устройство вспомогательных регистров и операции генерации адреса

Два арифметических устройства вспомогательных регистров (ARAU0 и ARAU1) оперируют содержимым вспомогательных регистров. ARAUx выполняют беззнаковые 16-разрядные арифметические операции над вспомогательными регистрами.

Вспомогательные регистры могут быть:

– Загружены непосредственным значением с использованием команды STM.

– Загружены через шину данных путем записи в картированные в памяти вспомогательные регистры.

– Изменены полем косвенной адресации любой команды, поддерживающей косвенную адресацию.

– Изменены командой модификации вспомогательных регистров (MAR).

– Использованы как счетчик цикла для команды BANZ[D].

Примечание – Обычно для загрузки дополнительных регистров применяются команды STM или MVDK. Любая из них позволяет уже следующей команде использовать новое содержимое дополнительного регистра. Другие команды, которые загружают новое значение в ARx, таким свойством не обладают вследствие задержки, вносимой конвейером.

Рисунок 24 показывает использование ARAUx для генерации адреса в режиме косвенной адресации с использованием одиночного операнда памяти данных. Как видно из рисунка, основными компонентами, используемыми для генерации адреса при косвенной адресации, являются арифметические устройства вспомогательных регистров (ARAU0 и ARAU1) и вспомогательные регистры (AR0-AR7).

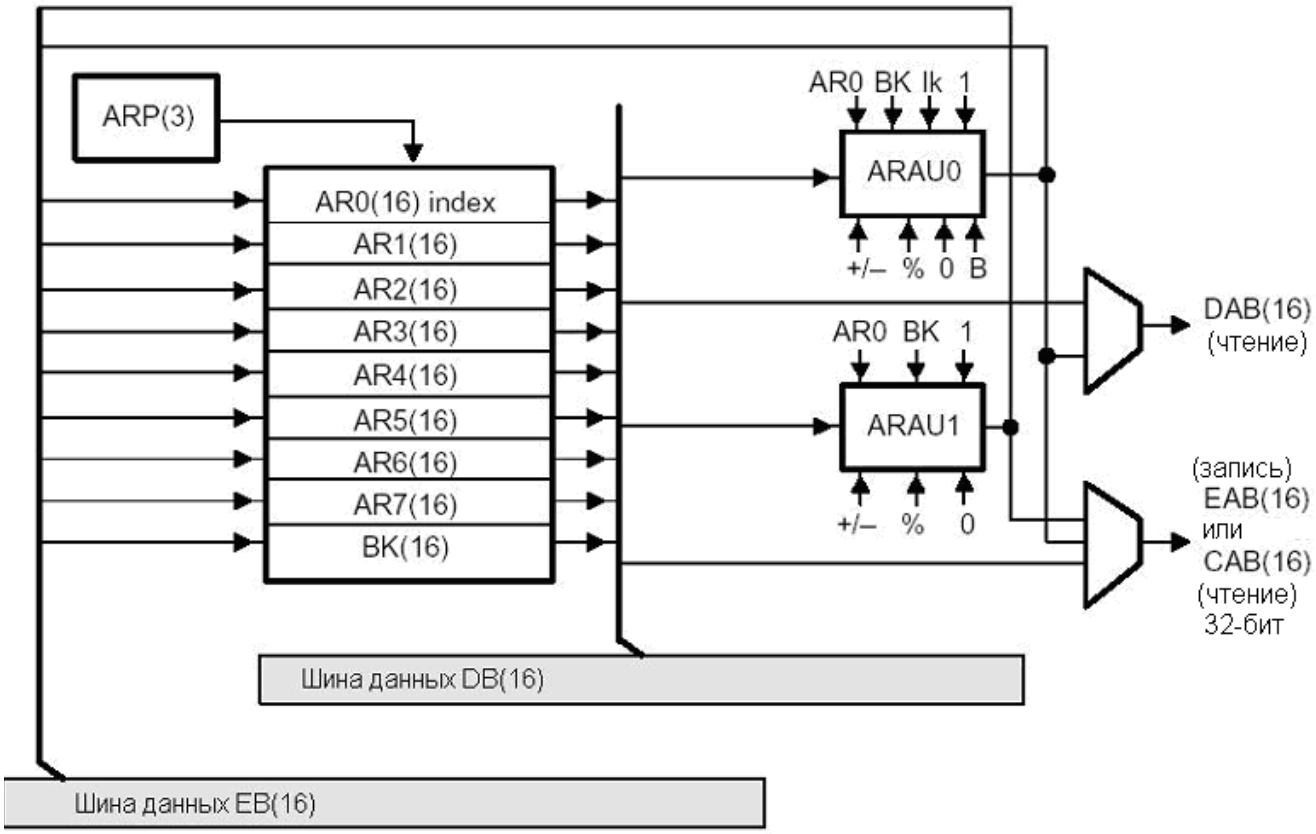


Рисунок 24 – Блок-диаграмма косвенной адресации для одиночного операнда памяти данных

5.9.5.3 Модификация адреса с одним операндом

Существует возможность модификации адресов, используемых в командах перед или после их выполнения. В частном случае адрес может и не изменяться. Можно выполнять модификацию адреса путем приращения или уменьшения на 1, добавления 16-разрядного смещения (lk) или значения индекса в AR0. Эти три типа модификации плюс возможность оставлять адрес без изменения складываются в 16 типов адресации, каждый из которых устанавливается значением 4-битного поля MOD в кодировке команды, использующей косвенную адресацию.

В таблице 14 приведены типы адресации с одним операндом памяти данных, с указанием значения поля MOD, синтаксиса ассемблера и выполняемой функцией каждой для каждого типа.

Таблица 14 – Типы косвенной адресации с одиночным операндом памяти данных

Поле MOD	Синтаксис операнда	Функция	Описание [1]
0000 (0)	*ARx	addr = ARx	ARx содержит адрес памяти данных
0001 (1)	*ARx–	addr = ARx ARx = ARx - 1	После обращения адрес в ARx уменьшается на 1 [2]
0010 (2)	*ARx+	addr = ARx ARx = ARx + 1	После обращения адрес в ARx увеличивается на 1 [2]
0011 (3)	*+ARx	addr = ARx + 1 ARx = ARx + 1	Перед использованием адрес в ARx увеличивается [2, 3]
0100 (4)	*ARx–0B	addr = ARx ARx = B(ARx–AR0)	После обращения AR0 вычитается из ARx с обратным распространением переноса
0101 (5)	*ARx–0	addr = ARx ARx = ARx – AR0	После доступа AR0 вычитается из ARx
0110 (6)	*ARx+0	addr = ARx ARx = ARx + AR0	После доступа AR0 складывается с ARx
0111 (7)	*ARx+0B	addr = ARx ARx = B(ARx+AR0)	После доступа AR0 складывается с ARx с обратным распространением переноса
1000 (8)	*ARx–%	addr = ARx ARx = circ(ARx–1)	После обращения адрес в ARx уменьшается на 1, используя циклическую адресацию [2]
1001 (9)	*ARx–0%	addr = ARx ARx=circ(ARx–AR0)	После доступа AR0 вычитается из ARx с использованием циклической адресации
1010 (10)	*ARx+%	addr = ARx ARx = circ(ARx+1)	После доступа адрес в ARx увеличивается на 1 с использованием циклической адресации [2]
1011 (11)	*ARx+0%	addr =ARx ARx=circ(ARx+AR0)	После доступа AR0 прибавляется к ARx , используя циклическую адресацию
1100 (12)	*ARx(lk)	addr = ARx + lk ARx = ARx	В качестве адреса памяти данных используется сумма ARx и 16-разрядной длинной константы (lk). ARx не обновляется [4]
1101 (13)	*+ARx(lk)	addr = ARx + lk ARx = ARx + lk	Перед использованием знаковая 16-разрядная длинная константа (lk) прибавляется к ARx, и этой суммой заменяется предыдущее значение ARx. Полученная сумма впоследствии используется для адресации операнда памяти данных [4]
1110 (14)	*+ARx(lk)%	addr = circ(ARx + lk) ARx = circ(ARx + lk)	Перед доступом знаковая 16-разрядная длинная константа (lk) добавляется к ARx с помощью циклической адресации и полученная сумма заменяет предыдущее значение ARx; эта сумма используется для адресации операнда памяти данных [4]
1111 (15)	*(lk)	addr= lk	Беззнаковая 16-разрядная длинная константа (lk) используется как абсолютный адрес памяти данных (абсолютная адресация) [4, 5]

[1] ARx используется как адрес памяти данных (если не определено другими условиями).

[2] Значение инкремента/декремента равно 1 для 16-разрядных×16-бит и 2 для 32-разрядных слов.

[3] Этот режим доступен только для записи.

[4] Этот режим не доступен для адресации к регистрам, отображенным в памяти.

[5] Этот режим подробнее обсуждается в разделе 5.9.2.4 **(lk) адресация*.

5.9.5.3.1 Модификации с индексированием (MOD = 5 или 6)

Индексная адресация – тип косвенной адресации, в котором содержимое AR0 вычитается или прибавляется к другому вспомогательному регистру (ARx). Индексная адресация отличается от адресации смещением тем, что индекс или длина шага может определяться в течение выполнения команды. Индексная адресация имеет преимущества перед адресацией смещением: она не требует дополнительного слова для команды.

Синтаксисы команд для вычитания AR0 из ARx и для добавления AR0 к ARx представлены в таблице 14 для MOD = 5 и 6 (соответственно).

5.9.5.3.2 Модификации циклической адресации (MOD = 8, 9, 10, 11 или 14)

Многие алгоритмы, такие как свертка, корреляция и КИХ-фильтрация, требуют реализации циклического буфера в памяти. В этих алгоритмах циклический буфер – подвижное окно, содержащее самые последние данные. Поступающие новые данные записываются в буфер поверх старых. Ключ к исполнению циклического буфера – выполнение циклической адресации.

Размер циклического буфера R заносится в регистр размера циклического буфера (BK) и должен начинаться на N-разрядной границе (то есть N младших бит базового адреса буфера должны быть равны 0), где N – самое малое целое число, которое удовлетворяет неравенству $2^N > R$. Значение R должно быть загружено в BK. Например, циклический буфер из 31 слова должен начинаться с адреса, в котором пять младших битов равны 0 (т. е. XXXX XXXX XXX0 0000_2) и число 31 должно быть загружено в BK. Аналогично, для 32-словного буфера в ноль должны быть установлены шесть младших битов адреса начала буфера (XXXX XXXX XX00 0000_2) и число 32 должно быть загружено в BK. Однако, в некоторых приложениях возможно использование бит-реверсной адресации для размещения 2^N буфера на 2^N границе и для использования эффекта циклической адресации.

Эффективный базовый адрес (EFB) циклического буфера определяется обнулением младших N битов, вспомогательного регистра ARx, выбираемого пользователем. *Адрес конца буфера* (EOB) определяется заменой младших N битов регистра ARx младшими N битами BK. *Индекс* (*index*) циклического буфера – это младшие N битов ARx; *шаг* (*step*) – значение, добавляемое или вычитаемое из вспомогательного регистра. При использовании циклической адресации следует помнить о трех правилах:

– Поместить первый (самый младший) адрес циклического буфера на 2^N границу, где 2^N больше чем размер циклического буфера.

– Использовать шаг меньший или равный размеру циклического буфера.

– При первом обращении к циклическому стеку вспомогательный регистр должен указывать на элемент циклической очереди.

Циклическая адресация использует следующий алгоритм:

```
If 0 ≤ index + step < BK:
     index = index + step.
Else if index + step ≥ BK:
     index = index + step – BK.
Else if index + step < 0:
     index = index + step + BK.
```

Циклическая адресация может быть использована как для одиночного, так и для двойного операнда памяти данных. Когда BK содержит ноль, циклический модификатор не приводит к модификации циклического адреса. Эта особенность полезна, когда двойной операнд должен выполнить модификацию адреса эквивалентную ARx + 0.

Рисунок 25 показывает взаимосвязь между BK, дополнительным регистром (ARx), нижней и верхней границами циклического буфера и индексом.

На рисунке 26 показано, как устроен циклический буфер и как соотносятся друг с другом сформированные значения и элементы циклического буфера.

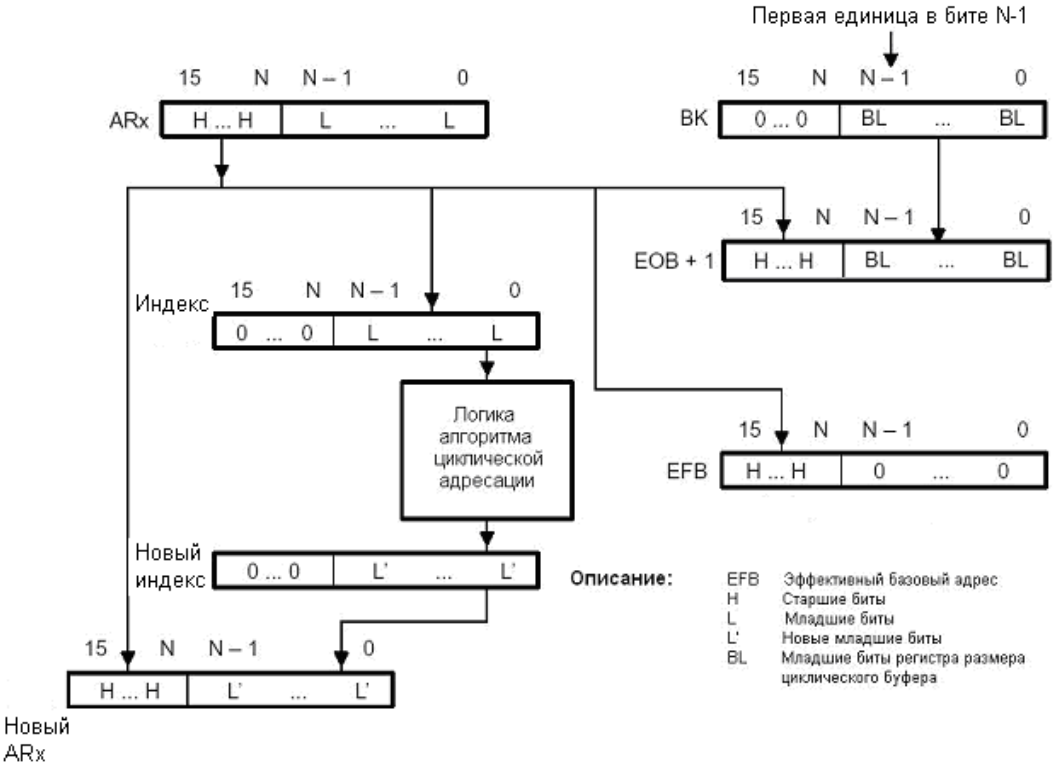


Рисунок 25 – Блок-диаграмма циклической адресации

Циклическая адресация обычно использует декремент или инкремент на 1 (MOD = 8 и 10), или декремент или инкремент на индекс (MOD = 9 и 11). Предварительная модификация 16-разрядной константой (*+ARx(lk)) требует дополнительного слова, поэтому такая команда состоит из двух или трех слов, что препятствует ее использованию в режиме повтора.

Синтаксис команды для каждого из пяти типов циклической адресации приведен в таблице 14 для MOD = 8, 9, 10, 11 и 14.

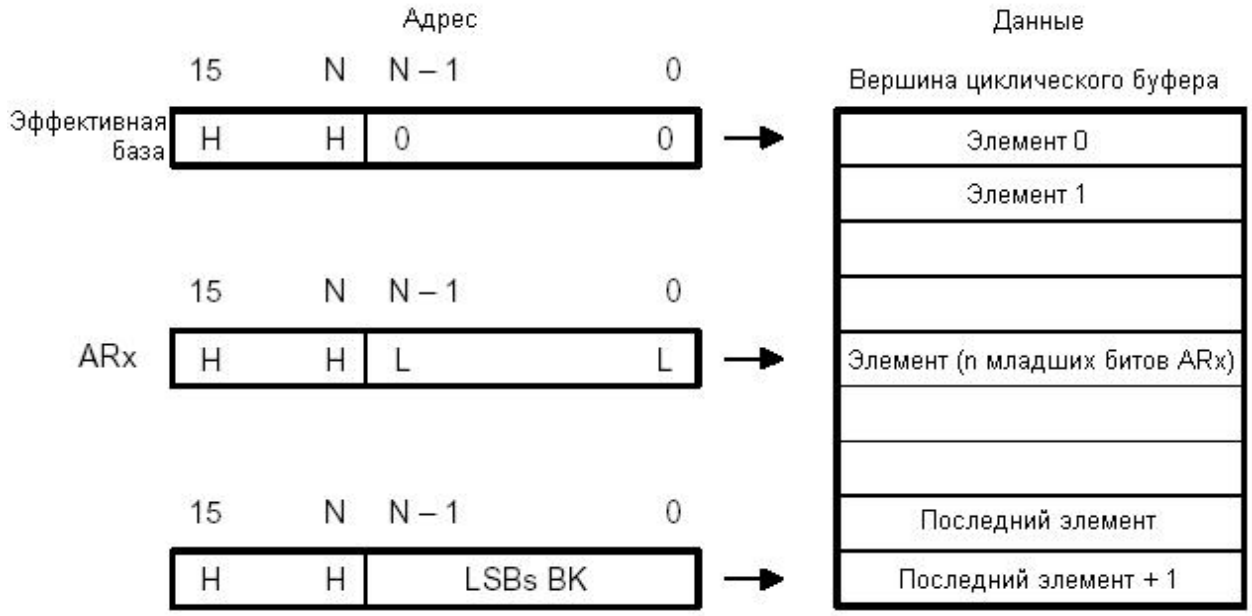


Рисунок 26 – Реализация циклического буфера

5.9.5.3.3 Модификации бит реверсной адресации (MOD = 4 или 7)

Бит-реверсная адресация позволяет увеличить скорость выполнения и расширить память программ для исполнения БПФ-алгоритмов, которые используют различные системы счисления. В этом режиме способе адресации AR0 определяет половину размера БПФ. Значение, содержащееся в AR0, должно быть равно $2N^{-1}$, где N – целое число; размер БПФ – 2N. Вспомогательные регистр указывает на физическое расположение значения данных. Когда AR0 добавляется к вспомогательному регистру с использованием бит-реверсной адресации, адрес генерируется в бит-реверсном виде с битом переноса, распространяющимся слева направо, в отличие от нормального режима, когда перенос распространяется справа налево.

Синтаксис двух режимов бит реверсной адресации показан в таблице 14 для MOD 4 и 7, соответственно.

Предположим, что вспомогательные регистры имеют длину 8 бит, регистр AR2 хранит базовый адрес данных в памяти ($0110\ 0000_2$), а регистр AR0 содержит значение $0000\ 1000_2$. Пример 4 показывает последовательность модификаций AR2 и результат значения AR2.

Пример 4 – Последовательность модификации вспомогательных регистров при бит-реверсной адресации

```
*AR2+0B   ;AR2  =   0110 0000 (0-е значение)
*AR2+0B   ;AR2  =   0110 1000 (1-е значение)
*AR2+0B   ;AR2  =   0110 0100 (2-е значение)
*AR2+0B   ;AR2  =   0110 1100 (3-е значение)
*AR2+0B   ;AR2  =   0110 0010 (4-е значение)
*AR2+0B   ;AR2  =   0110 1010 (5-е значение)
*AR2+0B   ;AR2  =   0110 0110 (6-е значение)
*AR2+0B   ;AR2  =   0110 1110 (7-е значение)
```

В таблице 15 представлено соотношение комбинации битов на каждом шаге индекса и четырех младших битов AR2, который содержит бит-реверсный адрес.

Таблица 15 – Соотношение комбинации битов на каждом шаге индекса и четырех младших битов AR2, который содержит бит-реверсный адрес

Шаг	Комбинация битов	Бит-реверсная комбинация	Бит-реверсный шаг
0	0000	0000	0
1	0001	1000	8
2	0010	0100	4
3	0011	1100	12
4	0100	0010	2
5	0101	1010	10
6	0110	0110	6
7	0111	1110	14
8	1000	0001	1
9	1001	1001	9
10	1010	0101	5
11	1011	1101	13
12	1100	0011	3
13	1101	1011	11
14	1110	0111	7
15	1111	1111	15

5.9.5.4 Модификация адреса с двойным операндом

Адресация двойным операндом памяти данных используется для команд, которые выполняют две операции чтения или одну операцию чтения и параллельно операцию сохранения (обозначается символом ‖). Все такие команды состоят только из одного слова и функционируют только в режиме косвенной адресации. Два операнда памяти данных называют соответственно Xmem и Ymem:

– Xmem – операнд чтения с доступом через шину D. Команды сохранения, например STH и STL с операцией сдвига меняют Xmem на операнд записи.

– Ymem используется как операнд чтения в командах с двойным чтением (с обращением через шину C) или как операнд записи в командах с параллельным сохранением (с обращением через шину E).

Если в командах с параллельным сохранением исходный операнд и операнд адресата указывают на одно и ту же ячейку (например, ST ‖ LD), источник считывается перед записью в адресат. Если команда с двойным операндом (например, ADD) указывает на один и тот же вспомогательный регистр с отличающимися способами адресации, то для адресации используется режим, определяемый полем Xmod.

На рисунке 27 показан формат команды косвенной адресации для двойного операнда памяти данных. В таблице 16 дано описание битов команды.

Поскольку в этом режиме для выбора вспомогательного регистра доступны только два разряда, то при таком типе адресации могут использоваться только регистры с AR2 по AR5. В таблице 17 показано, как путем выбора значений Xar или Yar выбираются вспомогательные регистры.

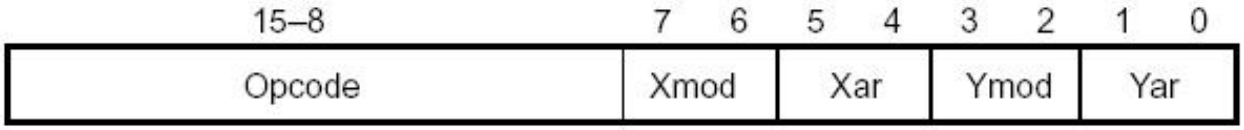


Рисунок 27 – Формат команды косвенной адресации с двойным операндом памяти данных

Таблица 16 – Описание разрядов команды косвенной адресации с двойным операндом памяти данных

Разряд	Имя	Исполняемые функции
15-8	Opcode	Это 8-разрядное поле содержит операционный код для команды
7-6	Xmod	Это 2-разрядное поле определяет тип режима косвенной адресации, используемой для доступа к операнду Xmem
5-4	Xar	Это 2-разрядное поле выбирает вспомогательный регистр, который содержит адрес Xmem
3-2	Ymod	Это 2-разрядное поле определяет тип режима косвенной адресации, используемой для доступа к операнду Ymem
1-0	Yar	Это 2-разрядное поле выбирает вспомогательный регистр, который содержит адрес Ymem

Таблица 17 – Зависимость выбранного вспомогательного регистра от значения полей Xar и Yar

Поле Xar или Yar	Вспомогательный регистр
00	AR2
01	AR3
10	AR4
11	AR5

На рисунке 28 показано, как с помощью адресации двойным операндом памяти данных, формируется адрес.

Адресация с двойным операндом памяти данных использует четыре вспомогательных регистра (AR2-AR5). ARAUx, вместе с этими регистрами, обеспечивают доступ к обоим операндам в одном цикле.

В таблице 18 приведен список типов адресации с двойным операндом памяти данных, включая значения поля модификации (Xmod или Ymod), синтаксис ассемблера и описание функций для каждого типа.

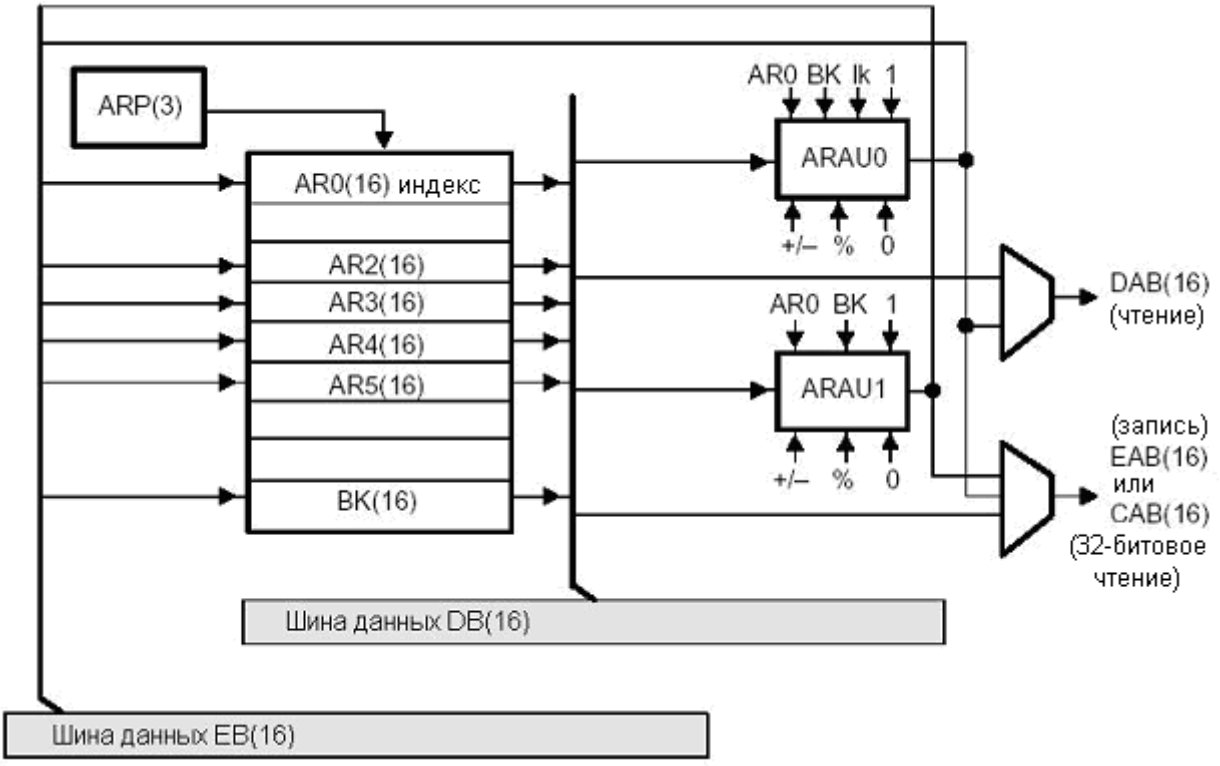


Рисунок 28 – Блок-диаграмма косвенной адресации для двойного операнда памяти данных

Таблица 18 – Типы косвенной адресации с двойным операндом памяти данных

Поле Xmod или Ymod	Синтаксис операнда	Функция	Описание [1]
00 (0)	*ARx	addr = ARx	ARx – адрес памяти данных
01 (1)	*ARx-	addr = ARx ARx = ARx - 1	После доступа, адрес в ARx уменьшается на 1
10 (2)	*ARx+	addr = ARx ARx = ARx + 1	После доступа адрес в ARx увеличивается на 1
11 (3)	*ARx+0 %	addr = ARx ARx = circ(ARx + AR0)	После доступа, AR0 добавляется к ARx с использованием циклической адресации [2]

[1] ARx используется как адрес памяти данных, если не определено другими условиями.
[2] Размер циклического буфера определяется регистром размера циклического буфера.

В каждом случае, содержимое вспомогательного регистра используется как операнд памяти данных. После использования адреса во вспомогательном регистре, ARAU 1 и 0 выполняют указанные математические операции. При запрете циклических модификаций, становится возможным выполнение индексной адресации или адресации, эквивалентной *ARx+0. Очистка BK в 0 отключает циклические модификации.

5.9.5.4.1 Одноопернадные команды, использующие формат двойного операнда

Некоторые команды, требующие только одного операнда, используют адресацию с двойным операндом, размещенным в одном слове, и требуют для выполнения один машинный цикл. В этих командах доступен только Xmem, а Xmod и Xar поля определяют только способ адресации к операнду. К этим командам можно отнести:

- BIT *Xmem, BITC.*
- SACCD *src, Xmem, cond.*
- SRCCD *Xmem, cond.*
- STRCD *Xmem, cond.*

Пять команд с опцией сдвига также относятся к этой категории:

- ADD *Xmem, SHFT, src.*
- LD *Xmem, SHFT, dst.*
- STH *src, SHFT, Xmem.*
- STL *src, SHFT, Xmem.*
- SUB *Xmem, SHFT, src.*

5.9.5.5 Режим совместимости (ARP)

В косвенной адресации может быть использован режим совместимости (ARP). При CMPT = 1 и ARF = 0 ARP указывает, какой вспомогательный регистр (AR) используется для адресации памяти. На рисунке 29 показано как ARP выбирает вспомогательные регистры.

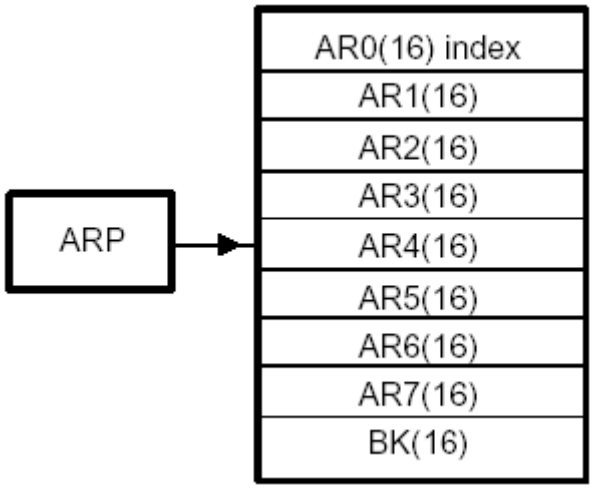


Рисунок 29 –Выбор вспомогательного регистра с помощью ARP

На рисунке 30 показан формат команды для косвенной адресации для режима ARP. В таблице 19 приведено описание битов команды в ARP-режиме.

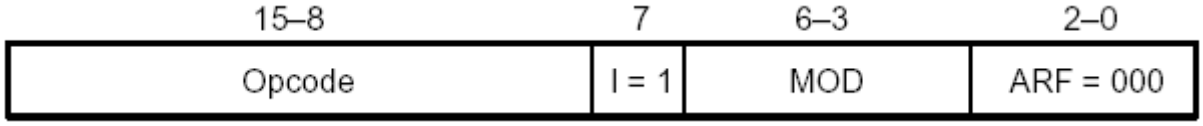


Рисунок 30 – Формат команды косвенной адресации для режима совместимости

Таблица 19 – Описание разрядов команды косвенной адресации – режим совместимости

Разряд	Имя	Исполняемые функции
15-8	Opcode	Это 8-разрядное поле содержит операционный код для команды
7	I	I = 1 означает, что командой используется косвенный режим адресации
6-3	MOD	Это 4-разрядное модифицируемое поле определяет тип косвенной адресации
2-0	ARF	Это 3-разрядное поле вспомогательного регистра определяет, какой вспомогательный регистр будет использоваться для адресации. ARF определяется битом режима совместимости (CMPT) в регистре состояния ST1: CMPT = 0 – стандартный режим. В стандартном режиме ARF всегда определяет вспомогательный регистр, независимо от значения в ARP. ARP не обновляется. Значение ARP всегда должно быть установлено в 0, когда процессор находится в стандартном режиме. CMPT = 1 – режим совместимости. В режиме совместимости ARP определяет вспомогательный регистр, если ARF = 0. Другими словами, ARF выбирает вспомогательный регистр и значение ARF загружается в ARP после доступа. *AR0 в записи команды указывает на вспомогательный регистр, выбранный ARP в режиме совместимости

Примечание – ARP всегда должен быть очищен в 0, когда процессор находится в стандартном режиме (CMPT = 0). При сбросе ARP и CMPT устанавливаются в 0 автоматически.

5.9.6 Адресация картируемых в памяти регистров

Адресация регистров, отображаемых в память, используется для модификации картированных регистров, без изменения текущего значения указателя страниц данных (DP) или текущего значения указателя стека (SP). Поскольку DP и SP не требуют модификации в этом режиме, затраты при записи регистра минимальны. При доступе к регистрам, отображаемым в памяти, используется как прямая, так и косвенная адресация.

На рисунке 31 показано как формируется адрес регистра, картируемого в памяти.

Адрес формируется двумя способами:

– Установкой девяти старших знаковых разрядов (MSB) адреса памяти данных в 0, независимо от текущего значения DP и SP при использовании прямой адресации.

– При косвенной адресации используют семь младших битов (LSB) текущего значения вспомогательного регистра.

Примечание – При косвенной адресации девять старших битов вспомогательного регистра устанавливаются в 0 после операции.

Например, если AR1 используется для указания регистра, отображенного в памяти во время рассматриваемой адресации и AR1 содержит значение FF25h, в этом случае AR1 указывает на регистр периода таймера (PRD), т. к. семь младших битов AR1 имеют значение 25h и адрес PRD – 0025h. После выполнения операции значение в AR1 остается 0025h.

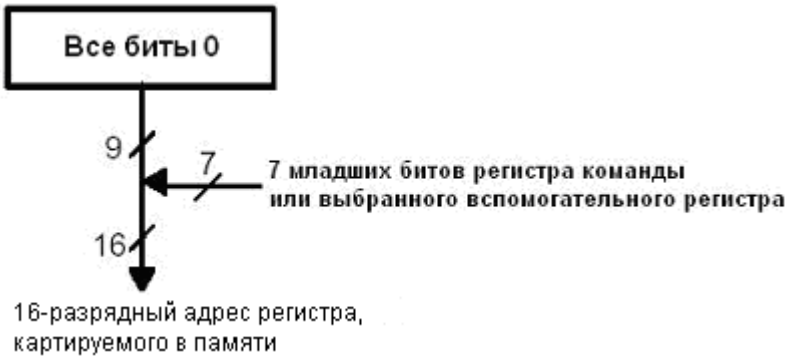


Рисунок 31 – Блок-диаграмма адресации регистров, картированных в памяти

Примечание – С помощью этого вида адресации могут быть модифицированы не только регистры, но и ячейки сверхоперативной памяти, расположенные в нулевой странице памяти данных.

Только восемь команд могут использовать адресацию картированных в памяти регистров:

– LDM *MMR, dst*;
– MVDM *dmad, MMR*;
– MVMD *MMR, dmad*;
– MVMM *MMRx, MMRy*;
– POPM *MMR*;
– PSHM *MMR*;
– STLM *src, MMR*;
– STM *#lk, MMR*.

Примечание – При адресации регистров, отображаемых в памяти, недопустимо использование следующих режимов косвенной адресации:

– *ARx(lk).
– *+ARx(lk).

– *+ARx(lk) %.
– *(lk).

В этих случаях ассемблер выдает предупреждение.

5.9.7 Адресация стеком

Системный стек используется для автоматического сохранения счетчика программ во время обработки прерываний и подпрограмм. Он так же может быть использован для сохранения дополнительных элементов контекста или для передачи значений данных. Стек заполняется от старшего к младшему адресу памяти. Процессор использует 16-разрядный картированный в памяти регистр указателя вершины стека (SP), для адресации стека. SP всегда указывает на последний элемент, помещенный в стек.

Используя адресацию стеком, четыре команды служат для обращения к стеку:

– PSHD помещает значение памяти данных в стек;
– PSHM помещает содержимое регистра, отображаемого в памяти, в стек;
– POPD записывает значение из стека в память данных;
– POPM записывает значение из стека в регистр, отображаемый в памяти.

Операция помещения в стек сопровождается предекрементом адреса в SP, операция извлечения из стека сопровождается постинкрементом адреса в SP. Рисунок 32 показывает состояние стека и регистра SP до и после помещения в стек значения X2 (PSHD X2).

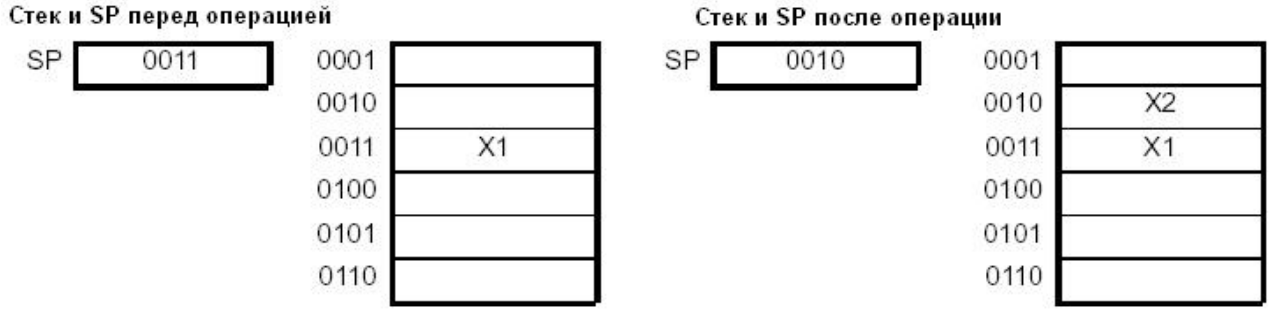


Рисунок 32 – Стек и указатель стека до и после операции добавления в стек

Другие операции так же затрагивают как сам стек, так и указатель стека. Стек используется в течение прерываний и выполнения подпрограмм для сохранения и восстановления счетчика программ. Когда вызывается подпрограмма или происходит прерывание, возвращаемый адрес автоматически сохраняется в стеке с использованием операции помещения в стек. Команды CALA[D], CALL[D], CC[D], INTR и TRAP используются для подпрограмм обработки вызовов и прерываний.

Во время выхода из подпрограммы адрес точки возврата берется из стека и загружается в счетчик программ. Для выхода из подпрограммы используются команды: RET[D], RETE[D], RETEF[D] и RC[D].

Команда FRAME добавляет короткое непосредственное смещение к указателю стека. Стек так же используется при прямой SP-адресации.

5.9.8 Типы данных

Для получения доступа к памяти в устройстве 1867ВЦ4Т используются два основных типа памяти: 16- и 32-разрядные. Большинству команд доступны 16-разрядные данные. Доступ к 32-разрядным данным требует применения специальных команд, перечисленных в таблице 20.

Таблица 20 – Команды с 32-разрядными операндами

Команда	Описание
DADD	Сложение с двойной точностью/двойное 16-разрядное прибавление к аккумулятору
DADST	Сложение с T с двойной точностью/двойное 16-разрядное сложение/вычитание с T
DLD	Загрузка в аккумулятор длинного слова
DRSUB	Вычитание с двойной точностью/двойное 16-разрядное вычитание из длинного слова
DSADT	Загрузка длинного слова со сложением T/двойная 16-разрядная загрузка со сложением/вычитанием T
DST	Сохранение содержимого аккумулятора в длинном слове
DSUB	Вычитание с двойной точностью/двойное 16-разрядное вычитание из аккумулятора
DSUBT	Загрузка длинным словом с вычитанием T/двойная 16-разрядная загрузка с вычитанием T

При работе с 16-разрядными операндами 16-разрядные слова читаются из памяти данных через шину данных D, а записываются через шину E. При доступе к 32-разрядным операндам, для чтения используются две шины: C (для старшего значащего слова) и D (для младшего значащего слова). Поскольку для записи используется только одна шина – E, операция записи (команда DST) выполняется в два цикла.

При 32-разрядном доступе, первое слово считается старшим значащим словом (MSW), второе, соответственно, – младшим значащим словом (LSW). Если первое слово размещено в четном адресе, то подразумевается, что второе слово находится в следующем нечетном адресе. Если же первое слово получает нечетный адрес, то второе слово, соответственно, получает предыдущий (младший) адрес. Такое распределение адресов показано на рисунке 33.

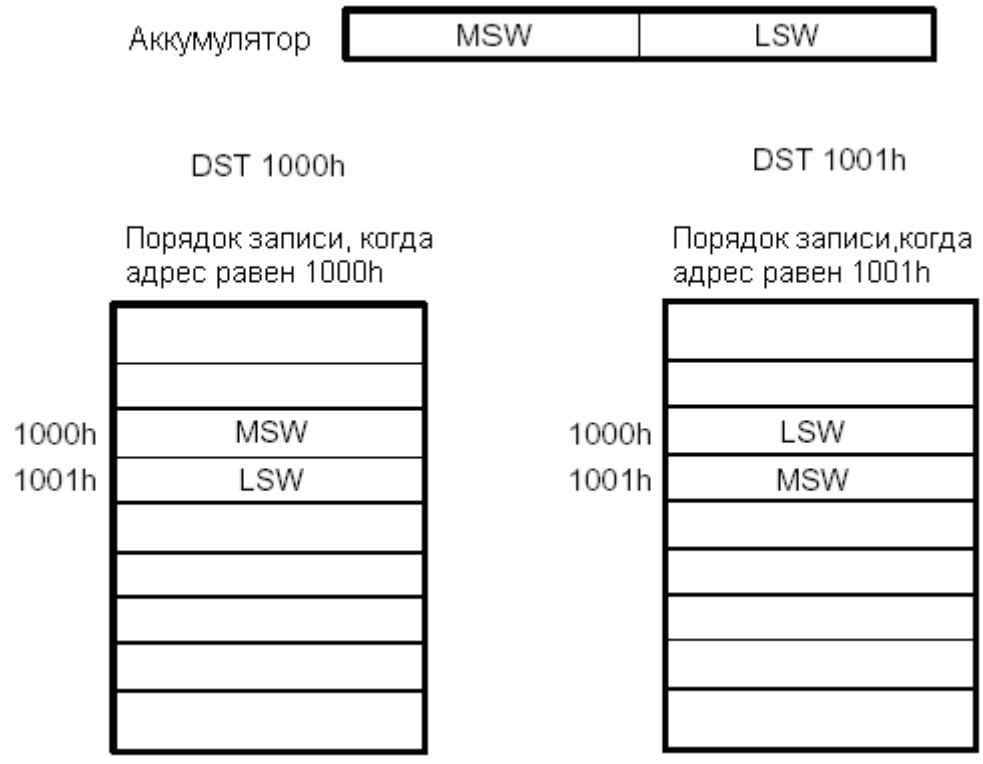


Рисунок 33 – Порядок записи слов в память

5.10 Адресация программной памяти

В этом разделе описывается генерация адресов памяти программ и способы загрузки этих адресов в счетчик команд (PC), а так же следующие операции управления программой, оказывающие влияние на значение, загружаемое в PC:

- Переходы (условный/безусловный).
- Вызовы (call).
- Возвраты (return).
- Условные операции.
- Повтор команды или блока команд.
- Аппаратный сброс.
- Прерывания.

5.10.1 Генерация адреса программной памяти

Память программ содержит исполняемый код прикладных программ, таблицы коэффициентов и непосредственные операнды. Процессор 1867ВЦ4Т может адресовать 64К слов памяти программ по шине программного адреса (PAB).

Логика генерации программного адреса (PAGEN) генерирует адрес, используемый для получения доступа к командам, таблицам коэффициентов, 16-разрядным непосредственным операндам или другой информации, хранящейся в памяти программ, и помещает этот адрес на шину PAB.

PAGEN состоит из пяти регистров (см. рисунок 34):

- Счетчик программ (PC).
- Счетчик повторов (RC).
- Счетчик повтора блоков (BRC).
- Регистр начального адреса блока повтора (RSA).
- Регистр конечного адреса блока повтора (REA).

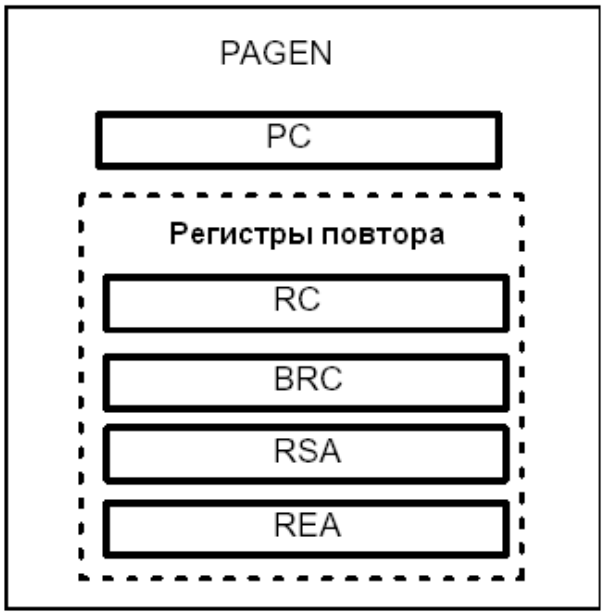


Рисунок 34 – Регистры модуля генерации адреса программ (PAGEN)

Адрес очередной команды, содержащийся в счетчике PC, помещается на шину PAB, после чего из памяти программ считывается следующая команда, а содержимое счетчика команд увеличивается на единицу. Если естественный порядок исполнения программы нарушается (например, выполняется команда перехода, вызова подпрограммы, возврат из подпрограммы, прерывание программы или команда повторения блока), соответствующий адрес загружается в PC. Команда, адресуемая через шину адреса программ PAB, помещается в регистр команд (IR).

Для повышения эффективности некоторых команд устройство генерации адреса программ используется при выборке операндов из памяти программ в процессе чтения/записи таблиц коэффициентов или пересылки данных между областями памяти программ и данных. Некоторые команды, такие как FIRS, MACD и MACP используют программную шину для выборки второго сомножителя.

5.10.2 Программный счетчик (PC)

Счетчик программы (PC) – это 16-разрядный регистр, который содержит внутренний или внешний адрес программной памяти, используемый при выборке команды или, при доступе к 16-разрядным непосредственным операндам или таблицам коэффициентов в памяти программ. Для доступа к памяти программ адрес из PC помещается на PAB.

PC может быть загружен несколькими путями. Таблица 21 показывает, что загружается в счетчик программ в соответствии с исполняемым кодом операции.

Таблица 21 – Загрузка адресов в PC

Код операции	Загрузка адреса в PC
Сброс (reset)	В PC загружается число FF80h
Последовательное выполнение	В PC загружается значение PC + 1
Ветвление (branch)	В PC загружается 16-разрядное непосредственное значение, расположенное за первым словом команды перехода
Ветвление по аккумулятору	PC загружается младшим 16-разрядным словом аккумулятора A или B
Цикл повтора блока	PC загружается стартовым адресом повтора (RSA), когда PC + 1 равен (REA) + 1 при условии, что BRAF = 1
Вызов подпрограммы	В стек помещается значение PC + 2, сам PC загружается 16-разрядным непосредственным значением, расположенным за первым словом команды. Команда возврата помещает вершину стека обратно в PC для возобновления последовательного исполнения кода
Вызов подпрограммы из аккумулятора	В стек помещается значение PC + 1, PC загружается 16-разрядным младшим словом аккумулятора A или B. Команда возврата помещает вершину стека обратно в PC для возобновления последовательного исполнения кода
Аппаратное прерывание, программное прерывание или внутреннее прерывание	Значение PC помещается в стек и в PC загружается адрес соответствующего вектора прерывания. Команда возврата помещает вершину стека обратно в PC для возобновления последовательного исполнения прерванной программы

5.11 Переходы

Переходы нарушают последовательность выполнения потока команд, передавая управление другому участку программной памяти. Поэтому, переходы оказывают влияние на адрес программы, формирующийся и хранящийся в PC. Процессор выполняет как условные, так и безусловные переходы; оба типа переходов могут выполняться как с задержкой, так и без.

5.11.1 Безусловные переходы

Безусловный переход выполняется всегда, когда встречается. Во время выполнения PC загружается указанным адресом перехода в памяти программ, чтобы выполнение программы начиналось с этого адреса. В PC загружается содержимое второго слова команды перехода или 16-разрядное значение аккумулятора (A или B).

К тому времени, когда команда перехода достигает фазы выполнения в конвейере, следующие две команды готовы к выборке. То, как именно будут обработаны эти команды, частично будет зависеть от того, выполняется ли переход с задержкой или нет:

– Без задержки: Два слова команды извлекаются из конвейера, таким образом, оставаясь неисполненными, и выполнение программы продолжается с адреса перехода.

– С задержкой: После команды перехода выполняется одна двухсловная команда или две однословные команды. Это позволяет избежать очистки конвейера, которая требует дополнительных циклов.

Примечание – Два слова после команды задержки не могут быть командой, нарушающей непрерывность счетчика команд (переход, вызов, возврат или программное прерывание).

В таблице 22 представлены команды безусловного перехода процессора 1867ВЦ4Т и число циклов, необходимых для выполнения этих команд (с задержкой и без нее).

Таблица 22 – Команды безусловного перехода

Команда	Описание	Количество циклов (без задержки/ с задержкой)
B[D]	PC загружается адресом, определяемым командой	4/2
BACC[D]	PC загружается адресом, определяемым младшими 16 разрядами указанного аккумулятора	6/4

5.11.2 Условные переходы

Условные переходы осуществляются так же, как и безусловные, однако выполняются только тогда, когда соблюдены одно или несколько поставленных условий. Возможные условия для выполнения перехода приведены в таблице 28. Если все условия выполнены, PC загружается вторым словом команды перехода, содержащим адрес перехода, с которого продолжится исполнение программы. К тому времени, как проверены поставленные условия, два слова команды, следующей за командой перехода, уже находятся в конвейере. То, как будут обработаны эти два слова, частично зависит от того, как выполняется переход – с задержкой или без:

– Без задержки: Если все условия выполнены, то эти два слова извлекаются из конвейера, таким образом, команда не исполняется; после чего выполнение программы продолжается с адреса перехода. Если условия не исполняются, то эти два слова исполняются вместо команды перехода.

– С задержкой: После команды перехода выполняется одна двухсловная команда или две однословные команды. Такая последовательность работы позволяет избегать

дополнительных циклов, связанных с очисткой конвейера. Проверяемые условия не затрагиваются командами после отложенного перехода.

В таблице 23 показаны команды условного перехода и количество циклов, необходимых для их выполнения. Поскольку выполнение условных переходов зависит от условий, в свою очередь определяемыми выполняемыми и предыдущими командами, команда условного перехода BC[D] требует на один цикл больше, чем безусловный переход.

Таблица 23 – Команды безусловного перехода

Команда	Описание	Количество циклов (условия выполнены/ не выполнены)	
		с задержкой	без задержки
BC[D]	PC загружается адресом, определяемым командой, если условие выполнено	5/3	3/3
BANZ[D]	PC загружается адресом, определяемым командой, если выбранный вспомогательный регистр не равен 0 (используется для построения цикла)	4/2	2/2

5.12 Вызовы подпрограмм

Подобно переходам, вызовы нарушают последовательный поток команд, передавая управление программой другому месту в области памяти программ. Однако, в отличие от переходов, вызов – это временная передача управления программой. При вызове подпрограммы или функции адрес следующей команды заносится в стек. Этот адрес используется для возврата из программы вызова и возобновления выполнения команд. Вызовы могут быть условные и безусловные, с задержкой или без задержки.

5.12.1 Безусловные вызовы

Безусловный вызов выполняется всегда, когда встречается. Когда выполняется вызов подпрограммы, счетчик программ (PC) загружается указанным адресом памяти программ и исполнение вызываемой подпрограммы начинается с этого адреса. Адрес, загружаемый в PC, может содержаться во втором слове команды вызова или в младших 16 разрядах аккумулятора (A или B). Перед загрузкой PC адрес возврата загружается в стек. После выполнения подпрограммы или функции команда возврата загружает в PC адрес возврата из стека и исполнение программы, следующей за вызовом, продолжается.

К тому времени, когда команда безусловного вызова достигает фазы выполнения в конвейере, следующие два слова кода уже выбираются. То, как будут обработаны эти два слова, частично зависит от того, произведен вызов с задержкой или без нее:

– Без задержки: два слова кода извлекаются из конвейера, поэтому они не исполняются, адрес возврата сохраняется в стеке и выполнение продолжается с начала вызываемой функции.

– С задержкой: Вслед за командой вызова выполняются одна двухсловная команда или две однословные команды. Это позволяет избежать появления дополнительных циклов, требуемых для выполнения операции.

Примечание – Два слова после команды задержки не могут быть командой, которая может быть причиной разрыва в PC (переход, вызов, возврат или программное прерывание).

В таблице 24 перечислены команды безусловного вызова подпрограммы (с задержкой или без задержки) и количество циклов, необходимых для их выполнения.

Таблица 24 – Команды безусловного вызова

Команда	Описание	Количество циклов (без задержки/ с задержкой)
CALL[D]	Помещает адрес возврата в стек и загружает в счетчик программ адрес, определяемый командой	4/2
CALA[D]	Помещает адрес возврата в стек и загружает в счетчик программы адрес, определяемый выбранным аккумулятором	6/4

5.12.2 Условные вызовы

Условные переходы осуществляются так же, как и безусловные, однако выполняются только тогда, когда соблюдены одно или несколько поставленных условий. Возможные условия приведены в таблице 28. Если все поставленные условия выполнены, то в счетчик программ загружается второе слово команды-вызова, которое содержит стартовый адрес вызова. Перед переходом к вызываемой функции, процессор сохраняет в стеке адрес команды, следующей за командой вызова. Вызванная функция должна оканчиваться командой возврата, которая берет адрес возврата из стека и загружает его в PC, позволяя процессору завершить программу вызова.

К тому времени, как проверены условия команды условного вызова, два слова команды, следующей за командой вызова, загружаются в конвейере. То, как будут обработаны эти два слова, частично зависит от того – произведен вызов с задержкой или без нее:

– Без задержки: Если все условия выполнены, то слова извлекаются из конвейера, поэтому они не исполняются и выполнение продолжается с начала вызываемой функции. Если условия не исполняются, то эти два слова выполняются вместо команды вызова.

– С задержкой: Вслед за командой вызова выполняется одна двухсловная команда или две однословные команды. Проверяемые условия не зависят от команд, следующих за отложенным вызовом. Если условия не подтверждаются, процессор выполняет эти два слова вместо команды вызова.

Примечание – Два слова после команды с задержкой не могут быть командой, которая может быть причиной разрыва в PC (переход, вызов, возврат или программное прерывание).

В таблице 25 перечислены команды условного вызова подпрограммы (с задержкой или без задержки) и количество циклов, необходимых для их выполнения

Таблица 25 – Команда условного вызова подпрограммы

Команда	Описание	Количество циклов (условия выполнены/ не выполнены)	
		с задержкой	без задержки
CC[D]	Сохраняет адрес возврата в стеке и загружает PC адресом, определенным командой, если условие выполняется.	5/3	3/3

5.13 Возвраты

Команды возврата из подпрограмм позволяют продолжить процесс выполнения последовательности команд, который был прерван функциями вызова или подпрограммами обработки прерываний. Возвраты могут быть условные и безусловные, с задержкой или без задержки.

5.13.1 Безусловные возвраты

Безусловный возврат выполняется всегда, когда встречается. Когда выполняется возврат, счетчик программ (PC) загружается адресом возврата из стека, после чего управление программой передается команде, следующей за командой вызова или за командой, после которой произошло прерывание.

К тому времени, когда команда безусловного возврата достигает фазы выполнения в конвейере, следующие два слова команды уже выбираются. То, как будут обработаны эти два слова, частично зависит от того, произведен возврат с задержкой или без нее:

– Без задержки: Два слова команды извлекаются из конвейера, поэтому они не исполняются и адрес возврата выбирается из стека или из регистра RTN, затем выполнение продолжается с этого адреса в функции вызова.

– С задержкой: Выполняются одна двухсловная команда или две однословные команды, следующие за командой возврата. Это позволяет избежать появления дополнительных циклов. Адрес возврата выбирается из стека или регистра RTN.

Примечание – Два слова после команды с задержкой не могут быть командой, которая может быть причиной разрыва в PC (переход, вызов, возврат или программное прерывание).

В таблице 26 перечислены команды безусловного возврата (с задержкой или без задержки) и количество циклов, необходимых для их выполнения.

Таблица 26 – Команды безусловного возврата

Команда	Описание	Количество циклов (без задержки/ с задержкой)
RET[D]	Загружает PC адресом возврата из вершины стека	5/3
RETE[D]	Загружает PC адресом возврата из вершины стека и разрешает маскируемые прерывания	5/3
RETF[D]	Загружает PC адресом возврата из регистра RTN и разрешает маскируемые прерывания	3/1

Механизм разрешения прерываний в последних командах таблицы гарантирует, что возврат произойдет прежде, чем начнется обработка следующего прерывания. При использовании команды RETF[D] загрузка PC из регистра RTN предпочтительнее, чем из стека. Это уменьшает общее количество циклов, используемых для подпрограммы обработки прерываний, и ускоряет возврат из подпрограммы.

Примечание – Регистр RTN – это внутренний, программно недоступный регистр CPU.

5.13.2 Условные возвраты

Использование аппарата условных возвратов позволяет программисту организовать несколько точек выхода из программы обслуживания прерываний (ISR) или из функции в зависимости от результата обработки данных. Кроме этого, использование условного возврата позволяет избежать условного перехода в команде возврата в конце функции или подпрограммы ISR.

Условные возвраты при выполнении условий, перечисленных в команде (допустимые условия приводятся в таблице 28), выполняются точно так же, как и безусловные. Если все условия выполнены, процессор загружает адрес возврата из стека в PC и продолжает выполнение основной программы.

Команда условного возврата – однословная; однако, в силу своей условности, времени для исполнения требует столько же, сколько и команды условного перехода или вызова.

К моменту завершения проверки условий в команде условного возврата два командных слова, следующих за командой возврата, загружаются в конвейер. То, как будут обработаны эти два слова, частично зависит от того – произведен вызов с задержкой или без нее:

– Без задержки: Если все условия выполнены, то эти два слова извлекаются из конвейера, поэтому они не исполняются, и происходит выполнение возврата. Если условия не исполняются, то эти два слова выполняются вместо команды возврата.

– С задержкой: Процессор выполняет две однословные команды, следующие за командой возврата. Это позволяет избежать очистки конвейера, требующей дополнительных циклов. Проверяемые условия не зависят от команд, следующих за возвратом с задержкой.

Примечание – Два слова после команды с задержкой не могут быть командой, которая может стать причиной разрыва в PC (переход, вызов, возврат или программное прерывание).

Таблица 27 содержит описание условной команды возврата и количество циклов, необходимых для ее выполнения.

Таблица 27 – Команды условного возврата

Команда	Описание	Количество циклов (условия выполнены/ не выполнены)	
		с задержкой	без задержки
RC[D]	Загружает PC адресом из вершины стека, если условия, указанные в команде, выполняются	5/3	3/3

5.14 Условные операции

Некоторые команды процессора 1867ВЦ4Т выполняются только тогда, когда удовлетворяются одно или несколько условий, список которых приводится в таблице 28.

Таблица 28 – Список условий для условных команд

Условие	Описание	Операнд
A = 0	Аккумулятор A равен нулю	AEQ
B = 0	Аккумулятор B равен нулю	BEQ
A ≠ 0	Аккумулятор A не равен нулю	ANEQ
B ≠ 0	Аккумулятор B не равен нулю	BNEQ
A < 0	Аккумулятор A меньше нуля	ALT
B < 0	Аккумулятор B меньше нуля	BLT
A ≤ 0	Аккумулятор A меньше или равен нулю	ALEQ
B ≤ 0	Аккумулятор B меньше или равен нулю	BLEQ
A > 0	Аккумулятор A больше нуля	AGT
B > 0	Аккумулятор B больше нуля	BGT
A ≥ 0	Аккумулятор A больше или равен нулю	AGEQ
B ≥ 0	Аккумулятор B больше или равен нулю	BGEQ
AOV = 1	Аккумулятор A переполнен	AOV
BOV = 1	Аккумулятор B переполнен	BOV
AOV = 0	Аккумулятор A не переполнен	ANOV
BOV = 0	Аккумулятор B не переполнен	BNOV

Окончание таблицы 28

Условие	Описание	Операнд
C = 1	Бит переноса ALU установлен в единицу	C
C = 0	Бит переноса ALU сброшен в ноль	NC
TC = 1	Флаг теста/управления установлен в единицу	TC
TC = 0	Флаг теста/управления сброшен в ноль	NTC
BIO = 0	Сигнал BIO находится на низком уровне	BIO
BIO = 1	Сигнал BIO находится в высоком уровне	NBIO
Условий нет	Безусловная операция	UNC

5.14.1 Использование комбинации условий

В качестве операнда условных команд могут использоваться комбинации условий. При этом для успешного завершения команды должны выполниться все условия. В таких командах возможно лишь ограниченное количество комбинаций условий (см. таблицу 29). Для каждой комбинации условия должны быть выбраны либо из группы 1, либо из группы 2:

– Группа 1: Разрешается выбирать одно условие из категории A и одно условие из категории B. Два условия не могут быть выбраны из одной категории. Например, разрешается проверять сразу оба условия и EQ и OV; и в тоже время, нельзя проверить GT и NEQ. Для обоих условий должен быть один и тот же аккумулятор. Например, нельзя одновременно тестировать AGT и BOV.

– Группа 2: Для проверки можно выбирать по одному из условий в каждой из трех категорий (A, B и C), но не по два и не по три. Например, выбор одновременно TC, C и BIO разрешается, но нет возможности выбрать NTC, C и NC.

Таблица 29 – Группы условий для команд с несколькими условиями

Группа 1		Группа 2		
Категория A	Категория B	Категория A	Категория B	Категория C
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	NBIO
LT				
LEQ				
GT				
GEQ				

5.14.2 Команды условного выполнения (XC)

Когда код команды должен перейти по условию через 1- или 2–словный кодовый сегмент, можно установить переход с помощью одноцикловой команды условного выполнения XC. Имеются два варианта команды XC: однословная команда (XC 1, cond) и двухсловная команда (XC 2, cond). Условия для команды XC такие же, как и для команд условного перехода, вызова и возврата из подпрограмм (см. таблицу 28).

Условие должно быть устойчиво в течение двух полных циклов перед исполнением команды XC, что гарантирует правильность ее отработки. Запрещается изменять условие в двух однословных командах, расположенных перед XC. В отсутствие прерываний это требование не обязательно, однако, если прерывание происходит между командой и XC, условие перехода может измениться.

5.14.3 Команды условного сохранения

Некоторые из регистров CPU могут сохраняться в памяти данных при выполнении определенных условий с помощью команд условного сохранения, перечисленных в таблице 30. Условия, используемые этими командами, перечисляются в таблице 31.

В команде условного сохранения модификация адреса и чтение операнда из памяти осуществляется независимо от условия. Если условие выполнено, соответствующий регистр сохраняется в памяти данных. Если условие не выполнено, операнд записывается в ту же самую ячейку памяти, из которой до этого читался.

Команды условного сохранения являются одноoперандными командами, но они могут использоваться в режиме косвенной адресации с двумя операндами с одним 16-разрядным словом, поэтому эти команды выполняются за один цикл.

Условное сохранение счетчика повторений блока (BRC) позволяет сохранять индекс в цикле блока повторения.

Таблица 30 – Команды условного сохранения

Команда	Регистр CPU
SACCD	Аккумулятор A или B
STRCD	Регистр временного хранения (T)
SRCCD	Счетчик блока повторения (BRC)

Таблица 31 – Перечень условий для команд условного сохранения

Синтаксис	Условие	Описание
AEQ	A = 0	Аккумулятор A равен 0
BEQ	B = 0	Аккумулятор B равен 0
ANEQ	A ≠ 0	Аккумулятор A не равен 0
BNEQ	B ≠ 0	Аккумулятор B не равен 0
ALT	A < 0	Аккумулятор A меньше 0
BLT	B < 0	Аккумулятор B меньше 0
ALEQ	A ≤ 0	Аккумулятор A меньше или равно 0
BLEQ	B ≤ 0	Аккумулятор B меньше или равно 0
AGT	A > 0	Аккумулятор A больше 0
BGT	B > 0	Аккумулятор B больше 0
AGEQ	A ≥ 0	Аккумулятор A больше или равен 0
BGEQ	B ≥ 0	Аккумулятор B больше или равен 0

5.15 Повторение одной команды

1867ВЦ4Т имеет две команды: RPT и RPTZ, которые повторяют следующую за ними команду. Число повторений определяется значением операнда команды + 1. Результат сохраняется в программно недоступном счетчике повторений RC. Максимально возможное число повторений равно 65536. Абсолютные адреса программ или данных автоматически увеличиваются на единицу при каждом повторении команды.

После декодирования команды повторения все прерывания, включая и NMI# (кроме RS#), запрещаются до завершения цикла повторения. Однако в зависимости от состояния бита HM регистра состояний ST1, устройство 1867ВЦ4Т может отвечать на запрос HOLD# при выполнение цикла повторений.

Функция повторения может быть использована с некоторыми командами, такими как умножение с накоплением и перемещение блоков, для увеличения быстродействия этих команд (таблица 32).

Однако не все команды могут быть повторены (например, команды, использующие адресацию *ARn (lk), * + ARn (lk), * + ARn (lk)% и * (lk)). Команды, перечисленные в таблице 33 , не могут быть повторены командами RPT.

Таблица 32 – Многоцикловые команды, которые становятся одноцикловыми командами при повторении

Команда	Описание	Количество циклов [1)]
FIRS	Симметричный FIR фильтр	3
MACD	Умножение с накоплением и перемещение результата в аккумулятор с задержкой	3
MACP	Умножение с накоплением и перемещение результата в аккумулятор	3
MVDK	Пересылка данных в данные	2
MVDM	Пересылка данных в MMR регистр	2
MVDP	Пересылка данных в программы	4
MVKD	Пересылка данных в данные	2
MVMD	Пересылка MMR регистров в данные	2
MVPD	Пересылка программ в данные	3
READA	Пересылка программ в данные	5
WRITA	Пересылка данных в программы	5
[1)] Количество циклов, когда команда не повторяется.		

Таблица 33 – Неповторяемые команды

Команда	Описание
ADDM	Суммирование содержимого памяти данных с длинной константой
ANDM	Логическое произведение содержимого памяти данных с длинной константой
B[D]	Безусловный переход
BACC[D]	Безусловный переход по содержимому аккумулятора
BANZ[D]	Переход по неравенству содержимого дополнительного регистра нулю
BC[D]	Условный переход
CALA[D]	Безусловный вызов подпрограммы по адресу из аккумулятора
CALL[D]	Безусловный вызов подпрограммы
CC[D]	Условный вызов подпрограммы
CMPR	Сравнение содержимого дополнительных регистров
DST	Сохранение длинного слова (32-бита) в аккумулятор
IDLE	IDLE команды
INTR	Программное прерывание
LD ARP	Загрузка указателя дополнительного регистра
LD DP	Загрузка указателя номера страницы
MVMM	Обмен между картированными в памяти регистрами
ORM	Логическая сумма содержимого памяти данных с длинной константой
RC[D]	Условный возврат
RESET	Программный сброс
RET[D]	Безусловный возврат из подпрограммы
RETE[D]	Безусловный возврат из подпрограммы с включением прерываний
RETF[D]	Безусловный возврат из подпрограммы по дальнему адресу
RND	Округление содержимого аккумулятора
RPT	Команда повторения
RPTB[D]	Команда повторения блока

Окончание таблицы 33

Команда	Описание
RPTZ	Повторение следующей команды с обнулением аккумулятора
RSBX	Сброс бита регистра состояния
SSBX	Установка бита регистра состояния
TRAP	Программное прерывание
XC	Условное выполнение
XORM	XOR содержимого памяти данных с длинной константой

5.16 Повторение блока команд

Команды повторения блока используются для организации программных циклов N + 1, где переменная N загружается в регистр счетчика повторения блока (BRC). Этот блок может содержать одну или несколько команд. В отличие от команды повторения одиночной операции, которая запрещает все маскируемые прерывания, команда повторения блока может быть прервана.

Команды, используемые для этой операции, – RPTB и RPTBD. Команда RPTB требует четыре цикла машинного времени. Отложенная команда RPTBD разрешает исполнение одной 2–словной команды или двух однословных команд, следующих за ней, чтобы избежать очистки конвейера. Таким образом, команда RPTBD выполняется за два машинных цикла. В этом случае отложенные команды не могут располагаться в двух словах после команды RPTBD.

Процесс выполнения команды повторения блока управляется флагом BRAF в регистре состояний ST1 и регистрами, отображенными в памяти:

– регистр BRC содержит число N, на единицу меньшее количества повторений блока;

– регистр начального адреса (RSA) содержит адрес первой команды программного блока для повторения;

– регистр конечного адреса (REA) содержит адрес последнего слова команды блока повторения.

Бит BRAF устанавливается в 1, когда повторение блока активизировано и число повторений больше нуля.

Инициализация блока повторения осуществляется в несколько этапов:

Этап 1: загружается счетчик блока повторений BRC числом в диапазоне от 0 до 65 535.

Этап 2: команда загружает адрес первой команды блока повторения в регистр RSA, если используется команда RPTB, или адрес второй команды после команды RPTBD, если используется последняя.

Этап 3: регистр REA загружается адресом команды, следующей за последним словом последней команды блока повторения, уменьшенным на единицу. На этом шаге устанавливается бит BRAF.

После выполнения каждой команды блока содержимое PC сравнивается с содержимым REA и при их равенстве значение BRC уменьшается на единицу. Если значение BRC больше или равно нулю, содержимое RSA загружается в PC и цикл перезапускается. В противном случае бит BRAF обнуляется и процессор завершает выполнение цикла.

Поскольку набор регистров повторения блока имеется лишь в единственном числе, вложенные блоки повторения процессором не поддерживаются. При необходимости вложенные циклы можно организовать, если команду RPTB[D] использовать для внутренних циклов, а команду BANZ[D] для организации всех внешних циклов.

5.17 Операция сброса

Сброс (RS#) – немаскируемое внешнее прерывание, которое может быть использовано всегда, когда нужно перевести процессор в исходное состояние. После включения питания RS# необходимо удерживать в течение нескольких машинных тактов в состоянии с низким уровнем для инициализации внутренних регистров процессора, а также шин данных, адреса и управления. Приблизительно через пять циклов после перехода вывода RS# в состояние с высоким уровнем процессор выбирает команду, размещенную по адресу FF80h, и начинает выполнение кода.

Сброс приводит к следующим установкам в процессоре:

- IPTR устанавливается в 1FFh;
- состояние вывода MP/MC# заносится в бит MP/MC# регистра PMST;
- в PC заносится адрес FF80h;
- константа FF80h помещается на шину адреса, независимо от состояния MP/MC#;
- шина данных переводится в высокоимпедансное состояние;
- линии управления переводятся в неактивное состояние;
- генерируется сигнал IACK#;
- бита INTM устанавливается в 1 для запрещения всех маскируемых прерываний;
- сбрасываются все флаги прерываний очисткой регистра IFR;
- счетчик повторений (RC) очищается;
- для инициализации периферийных устройств генерируется сигнал синхронного сброса (SRESET#);
- поля и биты регистров состояния инициализируются следующим образом:

ARP = 0	CLKOFF = 0	HM = 0	SXM = 1
ASM = 0	CMPT = 0	INTM = 1	TC = 1
AVIS = 0	CPL = 0	OVA = 0	XF = 1
BRAF = 0	DP = 0	OVB = 0	
C = 1	DROM = 0	OVLY = 0	
C16 = 0	FRST = 0	OVM = 0	

Примечания

1 Остальные биты состояния не инициализируются, поэтому необходима их программная установка.

2 Это же относится и к указателю стека (SP), который также не инициализируется при сбросе.

3 Если вывод MP/MC# = 0, устройство начинает выполнять программу из внутрикристальной ROM. В противном случае – из внешней памяти.

5.18 Прерывания

Прерывания вызываются специальными сигналами, порождаемыми аппаратными или программными средствами, которые вынуждают процессор приостанавливать основную программу и выполнять другую функцию, называемую подпрограммой обслуживания прерывания (ISR). Обычно прерывания генерируются аппаратными средствами, которым необходим обмен данными с 1867ВЦ4Т (например, АЦП, ЦАП или другие процессоры). Прерывания могут также использоваться для информирования процессора о некотором происшедшем событии (например, окончание счета таймера).

1867ВЦ4Т поддерживает следующие программные и аппаратные прерывания:

– программные прерывания запрашиваются программными командами INTR, TRAP или RESET;

– аппаратные прерывания вызываются сигналами из физических устройств. Существует два типа аппаратных прерываний:

— внешние аппаратные прерывания инициируются сигналами от входов внешних прерываний;

— внутренние аппаратные прерывания инициируются сигналами из внутрикристальных периферийных устройств.

Когда несколько аппаратных прерываний происходят одновременно, конфликт между ними разрешается с помощью системы приоритетов 1867ВЦ4Т.

Все прерывания, аппаратные или программные, можно отнести к одной из двух категорий:

– Маскируемые прерывания. Это аппаратные или программные прерывания, которые могут быть программно блокированы или разблокированы.

– Немаскируемые прерывания. Эти прерывания не могут быть блокированы. 1867ВЦ4Т всегда обслуживает этот тип прерываний и выполняет переход из основной программы к ISR. Немаскируемые прерывания включают в себя все программные прерывания и два внешних аппаратных прерывания: RS# (сброс) и NMI# (RS# и NMI# могут также вызываться программно). Прерывание RS# в отличие от NMI# всегда оказывает влияние на все действующие режимы процессора.

Обработка прерываний процессором 1867ВЦ4Т осуществляется в трех фазах:

1 Прием запроса на программное или аппаратное прерывание выполняемой программы. Если это прерывание маскируемое, устанавливается соответствующий бит в регистре флага прерывания (IFR).

2 Процессор подтверждает прием запроса на прерывание. Если прерывание маскируемое, то должен быть установлен в 1 соответствующий ему бит в регистре маски прерываний (IMR). Для немаскируемых аппаратных и программных прерываний это не требуется.

3 После подтверждения прерывания процессор переходит по вектору прерывания к подпрограмме обработки прерывания (ISR).

5.18.1 Регистр флага прерываний (IFR)

IFR – картируемый в памяти регистр CPU, который идентифицирует и очищает активные прерывания (его формат приведен на рисунке 35).

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Резерв	Резерв	Резерв	HPINT	INT3	TXINT	TRINT	BXINT	BRINT	TINT	INT2	INT1	INT0

Рисунок 35 – Диаграмма регистра флага прерываний IFR

Прерывание устанавливает соответствующий флаг прерывания в IFR, пока оно распознается процессором. Любое из следующих событий очищает флаг прерывания:

– при сбросе (уровень RS# – низкий);
– пришло командное прерывание TRAP;
– запись 1 в соответствующий бит IFR;
– выполняется команда INTR с соответствующим номером прерывания.

Единица в любом бите IFR указывает на отложенное прерывание. Чтобы очистить прерывание следует записать 1 в соответствующий бит регистра IFR. Все отложенные прерывания могут быть очищены записью текущего содержимого IFR обратно в IFR.

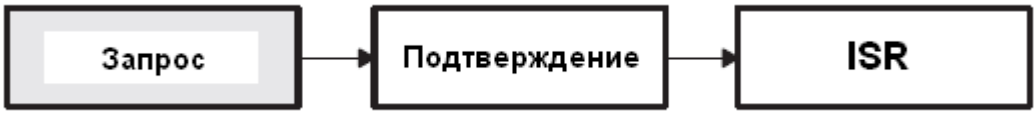
5.18.2 Регистр маски прерывания (IMR)

Формат регистра маски прерываний с точностью до бита соответствует формату регистра флагов прерываний, приведенному выше. Если бит INTM = 0 в регистре состояний ST1, то 1 в любом из битов IMR разрешает соответствующее прерывание. Ни NMI#, ни RS# не включены в IMR, т. к. IMR не влияет на эти прерывания. IMR доступен для записи и чтения (см. рисунок 36).

15–12	11	10	9	8	7	6	5	4	3	2	1	0
Резерв	Резерв	Резерв	HPINT	INT3	TXINT	TRINT	BXINT	BRINT	TINT	INT2	INT1	INT0

Рисунок 36 – Диаграмма регистра маски прерывания IMR

5.18.3 Фаза 1: Получение запроса на прерывание



Прерывание запрашивается внешним устройством или программной командой. Когда появляется запрос на прерывание, соответствующий флаг в IFR активизируется вне зависимости от того, будет ли в последствии это прерывание подтверждено процессором или нет. Флаг автоматически очищается, когда процессор переходит к обработке этого прерывания.

– *Запрос аппаратного прерывания*. Внешние аппаратные прерывания запрашиваются сигналами от внешнего источника; внутренние аппаратные прерывания - от внутрикристальных периферийных устройств.

– *Запрос программного прерывания*. Программные прерывания могут запрашиваться одной из следующих команд процессора:

— INTR посылает запрос на прерывание, вектор которого определяется константой K. При переходе к обслуживанию этого прерывания все остальные маскируемые прерывания запрещаются установкой бита INTM в единицу;

— TRAP аналогична команде INTR без установки бита INTM;

— RESET выполняет немаскируемый программный сброс, который используется для установки процессора в исходное состояние. Следует иметь в виду, что команда RESET не влияет на содержимое регистра PMST. При обслуживании этого прерывания все маскируемые прерывания запрещены. Инициализация IPTR и периферийных регистров отличается от их инициализации, выполненной аппаратным сбросом.

5.18.4 Фаза 2: Подтверждение прерывания

После того, как прерывание было запрошено, CPU должен решить, подтвердить или нет запрос. Программные и немаскируемые аппаратные прерывания подтверждаются

немедленно. Маскируемые аппаратные прерывания подтверждаются только при наличии выполнения некоторых условий:

– Уровень приоритета. Когда одновременно приходит несколько запросов от разных аппаратных прерываний, процессор обслуживает тот из них, у которого приоритет выше.

– INTM = 0. В этом случае все незамаскированные прерывания разрешены. INTM переходит в единицу автоматически в случае подтверждения прерывания. При выходе из подпрограммы обслуживания прерывания (ISR) через команду RETE бит INTM сбрасывается. Бит INTM может быть изменен соответствующими командами процессора RSBX и SSBX, а также аппаратным сбросом.

– Соответствующий бит в регистре маски прерываний IMR должен быть установлен в 1 для разрешения этого прерывания.

CPU подтверждает маскируемое аппаратное прерывание и выставляет на шину команд INTR. Эта команда загружает в PC соответствующий адрес и выбирает программный вектор. При этом CPU генерирует сигнал IACK#, который очищает соответствующий бит в регистре флага прерывания и помещает номер прерывания на шину адреса в биты A6-A2 на переднем фронте сигнала CLKOUT. Если таблица векторов прерывания расположена во внутрикристальной памяти и при этом необходимо контролировать адреса, то для этого процессор должен функционировать в режиме видимости адреса (AVIS = 1). Если прерывание приходит в то время, когда 1867ВЦ4Т находится в режиме HOLD и HM = 0, адрес не подается на шину, когда IACK# становится активным.

5.18.5 Фаза 3: Выполнение подпрограммы обработки прерывания (ISR)

После подтверждения прерывания CPU производит следующие действия:

– сохраняет значение счетчика программ PC (адрес возврата) в вершине стека в памяти данных;

– загружает PC адресом вектора прерывания;

– выбирает команду, размещенную по адресу вектора прерывания;

– выполняет переход на начало ISR, причем в случае перехода с задержкой дополнительная(ые) команда(ы) выполняется(ются) перед переходом;

– выполняет ISR, пока не встретит команду возврата;

– извлекает из вершины стека адрес возврата и загружает его PC;

– продолжает выполнять основную программу.

Каждый вектор прерывания находится в четырех последовательно расположенных позициях таблицы векторов и содержит отсроченную команду перехода и две 1-словные команды или одну 2-словную команду.

5.18.6 Сохранение контекста

В начале подпрограммы обработки прерывания, некоторых регистров должны быть сохранены в стеке. При выходе из ISR (RC[D], RETE[D]) необходимо восстановить их содержимое. Стек также используется для вызова подпрограмм, в том числе и внутри ISR. Так как регистры CPU и периферийные регистры отображены в памяти, команды PSHM и POPM могут передавать содержимое этих регистров, а также значения памяти данных в и из стека.

Учитывая особенности стека, необходимо помнить, что восстановление сохраненных в стеке данных должно идти в обратном порядке по отношению к сохранению. Кроме того, регистр BRC должен быть восстановлен до восстановления бита BRAF в ST1. В противном случае бит BRAF будет очищен, если BRC = 0 перед восстановлением BRC.

5.18.7 Время ожидания прерывания

Перед подтверждением прерывания процессор завершает выполнение всех команд, находящихся в конвейере, за исключением команд в стадиях предварительной выборки и выборки перед выполнением прерывания, так что максимальное время ожидания прерывания зависит от содержимого конвейера. Команды, требующие дополнительных циклов ожидания для доступа к медленной памяти, и команды повторения требуют дополнительного времени ожидания для обработки прерывания.

Команды повторения RPT и RPTZ требуют, чтобы все повторения последующей команды были завершены перед разрешением прерывания для защиты контекста повторенных команд. Эта защита необходима, т. к. контекст этих операций не может быть сохранен в ISR.

Так как функция состояния удержания имеет приоритет над прерываниями, она также может задерживать прерывание. Если прерывание произошло, когда CPU находится в режиме удержания (HOLD# активизирован) и вектор прерывания находится во внешней памяти, прерывание не захватывается, пока HOLDA# не перейдет в неактивное состояние и режим удержания не завершится. Однако, если процессор находится в конкурирующем режиме удержания (HM = 0) и таблица векторов прерывания размещена во внутренней памяти, CPU захватывает прерывание, независимо от состояния HOLD#.

Прерывания не могут быть обработаны между командой RSBX INTM и следующей командой в командной последовательности. Если прерывание происходит в течение декодирующейся фазы команды RSBX INTM, CPU всегда завершит выполнение этой команды, а также следующей за нею прежде, чем перейдет на подпрограмму обработки прерывания. Этот алгоритм гарантирует защиту от незапланированных прерываний при выполнении подпрограммы ISR. Если подпрограмма ISR завершается командой RETE, то команда RSBX INTM в конце ISR не нужна. Противоположная команде RSBX INTM команда SSBX INTM = 1 и команда, следующая за ней, не может прерываться.

Примечание – Сброс (RS#) не задерживается многоцикловыми командами. NMI# может быть задержан многоцикловыми командами и сигналом HOLD#.

5.18.8 Операция прерывания – общее описание

Как только прерывание CPU затребовано, процессор поступает следующим образом (рисунок 38):

– если запрошено маскируемое прерывание:

1) устанавливается соответствующий бит в IFR;

2) проверяются условия подтверждения прерывания (INTM = 0 и соответствующий бит IMR = 1). Если условия удовлетворены, CPU подтверждает прерывание генерацией сигнала IACK#, иначе прерывание игнорируется и процессор продолжает выполнять основную программу;

3) после подтверждения прерывания соответствующий флаг в IFR очищается и бит INTM устанавливается в 1 для блокировки других маскируемых прерываний;

4) содержимое PC сохраняется в стеке;

5) CPU переходит к выполнению подпрограммы обслуживания прерывания (ISR);

6) ISR завершается командой возврата, которая выталкивает адрес возврата из стека;

7) CPU продолжает выполнять основную программу.

– если затребовано немаскируемое прерывание:

1) CPU немедленно подтверждает прерывание, генерируя сигнал IACK#;

2) если прерывание было запрошено RS#, NMI# или командой INTR, бит INTM устанавливается в единицу для блокировки маскируемых аппаратных прерываний;

3) если команда INTR запросила одно из маскируемых прерываний, соответствующий флаг сбрасывается в нуль;

4) содержимое PC сохраняется в стеке;

5) CPU переходит к выполнению ISR;

6) ISR завершается командой возврата, которая выталкивает адрес возврата из стека;

7) CPU продолжает выполнение основной программы.

Примечание – Команда INTR запрещает маскируемые прерывания установкой бита режима прерываний INTM, а команда TRAP не воздействует на INTM.

5.18.9 Перемещение адресов векторов прерывания

Векторы прерывания могут быть перемещены в начало любой 128-словной страницы памяти программ, за исключением зарезервированных областей. Адрес вектора прерывания генерируется с помощью конкатенации поля указателя страницы векторов прерываний IPTR регистра PMST с номером вектора прерывания (0-31), сдвинутым на два разряда влево.

Например, если сигнал INT0# переходит в состояние с низким уровнем и IPTR = 0001h, вектор прерывания выбирается из ячейки 00C0h. Номер вектора прерывания для INT0 – 16 или 10h (рисунок 37).

При сбросе биты IPTR устанавливаются в единицу (IPTR = 1FFh); т. е. при старте процессора векторы должны находиться в странице 511 программной памяти, поэтому вектор аппаратного сброса всегда находится по адресу 0FF80h. Векторы прерывания могут отображаться в другую страницу перезагрузкой IPTR другим значением. Например, векторы прерывания могут перемещаться в начало программной памяти по адресу 0080h после загрузки IPTR значением 0001h.

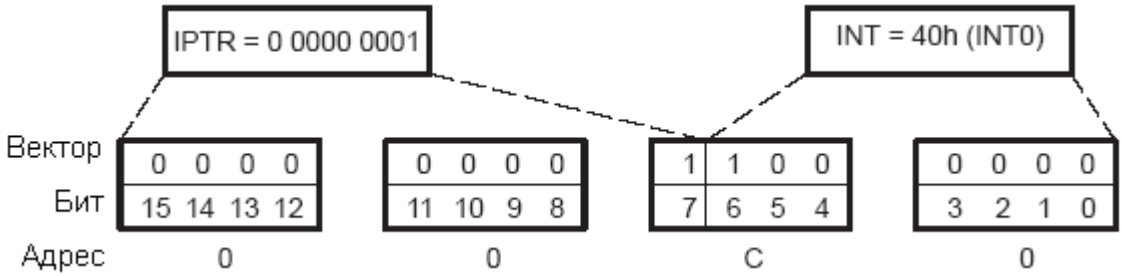


Рисунок 37 – Генерация адреса вектора прерывания

5.18.10 Таблица прерываний

В таблице 34 представлена таблица векторов прерываний и приоритетов процессора 1867ВЦ4Т.

Таблица 34 – Таблица векторов прерываний и приоритетов процессора 1867ВЦ4Т

Значение К	Приоритет	Имя	Размещение (HEX)	Функция
0	1	RS#/SINTR	0	Аппаратный и программный сброс
1	2	NMI#/SINT16	4	Немаскируемое прерывание
2	-	SINT17	8	Программное прерывание #17
3	-	SINT18	C	Программное прерывание #18
4	-	SINT19	10	Программное прерывание #19
5	-	SINT20	14	Программное прерывание #20
6	-	SINT21	18	Программное прерывание #21
7	-	SINT22	1C	Программное прерывание #22
8	-	SINT23	20	Программное прерывание #23
9	-	SINT24	24	Программное прерывание #24
10	-	SINT25	28	Программное прерывание #25
11	-	SINT26	2C	Программное прерывание #26
12	-	SINT27	30	Программное прерывание #27
13	-	SINT28	34	Программное прерывание #28
14	-	SINT29	38	Программное прерывание #29, резерв
15	-	SINT30	3C	Программное прерывание #30, резерв
16	3	INT0#/SINT0	40	Внешнее пользовательское прерывание #0
17	4	INT1#/SINT1	44	Внешнее пользовательское прерывание #1
18	5	INT2#/SINT2	48	Внешнее пользовательское прерывание #2
19	6	TINT/ SINT3	4C	Внутреннее прерывание по таймеру
20	7	BRINT/ SINT4	50	Прерывание по приему в буферированный последовательный порт
21	8	BXINT/ SINT5	54	Прерывание по передаче в буферированный последовательный порт
22	9	TRINT/ SINT6	58	Прерывание по приему в TDM последовательный порт
23	10	TXINT/ SINT7	5C	Прерывание по передаче в TDM последовательный порт
24	11	INT3#/SINT8	60	Внешнее пользовательское прерывание #3
25	12	HPINT/SINT9	64	Прерывание от host-порта
26-31	-	-	68-7F	Резерв

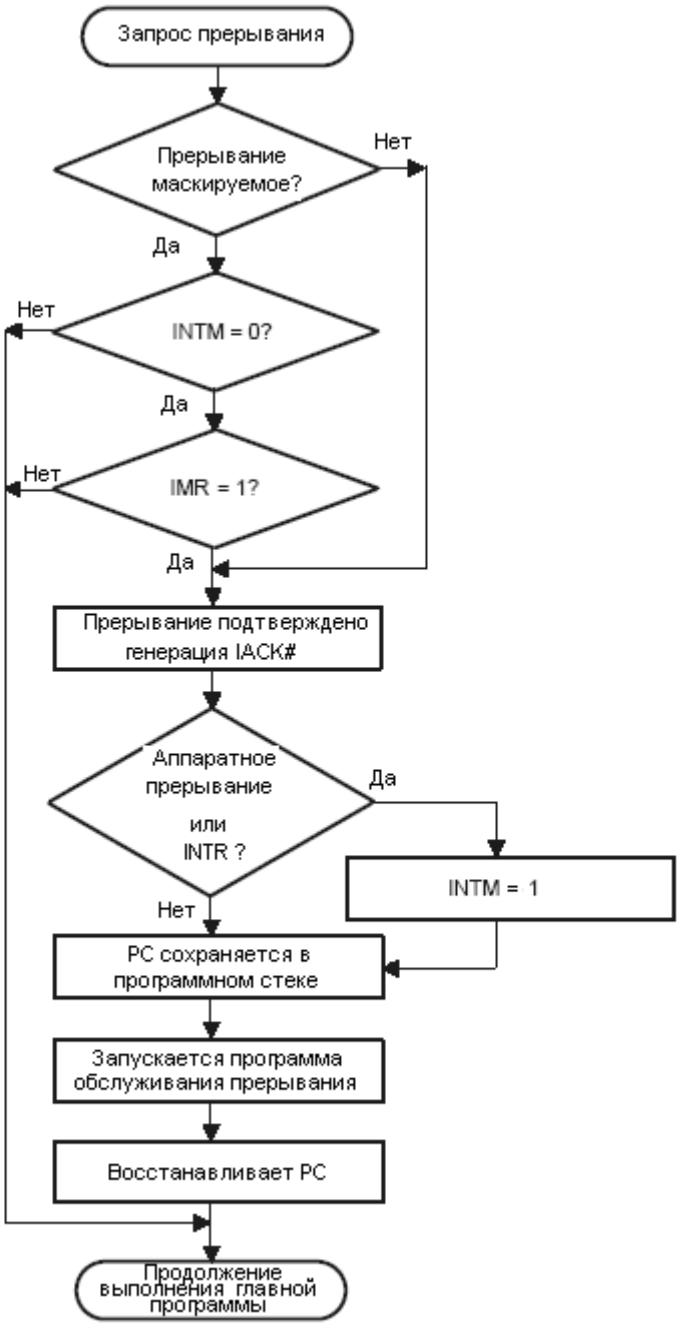


Рисунок 38 – Блок-схема алгоритма обработки прерывания процессора

5.19 Режимы уменьшения энергопотребления

Процессор 1867ВЦ4Т можно переключить в режим пониженной мощности потребления, когда отключены некоторые или все внутренние ресурсы, кроме следящих за внешними событиями, инициализирующими возвращение его в нормальный режим работы. Такое переключение осуществляется командами idle(1), idle(2) или idle(3), а также сигналом HOLD# при условии HM = 1. Режимы пониженной мощности сведены в таблицу 35 и подробно описаны ниже.

Таблица 35 – Четыре режима пониженной мощности

Операция/Особенности	IDLE1	IDLE2	IDLE3	HOLD#
CPU остановлен	Да	Да	Да	Да [1)]
Остановлена синхронизация CPU	Да	Да	Да	Нет
Остановлена периферийная синхронизация	Нет	Да	Да	Нет
Остановлено устройство PLL	Нет	Нет	Да	Нет
Внешние линии адреса помещены в высокоимпедансное состояние	Нет	Нет	Нет	Да
Внешние линии данных помещены в высокоимпедансное состояние	Нет	Нет	Нет	Да
Внешние сигналы управления помещены в высокоимпедансное состояние	Нет	Нет	Нет	Да
Режим пониженной мощности завершен, когда:				
HOLD# переходит в состояние с высоким уровнем	Нет	Нет	Нет	Да
Внутреннее немаскируемое аппаратное прерывание	Да	Нет	Нет	Нет
Немаскируемое внешнее аппаратное прерывание	Да	Да	Да	Нет
NMI#	Да	Да	Да	Нет
RS#	Да	Да	Да	Нет
[1)] В зависимости от состояния бита HM операции в CPU продолжают выполняться, если не требуется доступа к внешней памяти.				

5.19.1 Режим IDLE1

В режиме IDLE1 все ресурсы процессора, за исключением системной синхронизации и внутренних периферийных устройств, отключаются. Вывод CLKOUT также остается активным. Таким образом, периферийные устройства такие, как последовательные порты и таймер, могут выводить CPU из состояния пониженного энергопотребления.

В режим IDLE1 процессор переводится командой IDLE1, а возвращается из него требованием прерывания от одного из периферийных устройств. При этом, если в этот момент INTM = 0, процессор переходит на подпрограмму обработки этого прерывания, когда завершается режим IDLE1. Если же INTM = 1, начинает выполняться следующая за IDLE1 команда. Все пробуждающие прерывания должны быть разрешены установкой соответствующего бита в регистре IMR независимо от значения INTM. Единственным исключением являются немаскируемые прерывания RS# и NMI#.

5.19.2 Режим IDLE2

Режим IDLE2 останавливает не только CPU, но и внутрикристальные периферийные устройства, поэтому они не могут использоваться для пробуждения процессора, как это имеет место в режиме IDLE1. Однако, т. к. процессор почти полностью отключен, мощность потребления существенно уменьшается.

Для перехода в режим IDLE2 используется команда IDLE 2. Завершается режим IDLE2 активизацией любого из выводов внешних прерываний (RS#, NMI# и INTx) импульсом, длительность которого не должна быть меньше 10 нс. Если в этот момент INTM = 0, процессор переходит на подпрограмму обработки прерывания и режим IDLE2 завершается, в противном случае выполняется команда, следующая за IDLE 2. Все пробуждающие прерывания должны быть разрешены в регистре IMR независимо от значения бита INTM. После выхода процессора из режима IDLE2 необходимо заново

инициализировать все периферийные устройства, особенно, если они используют внешние синхросигналы. Это не обязательно делать, если в качестве пробуждающего прерывания используется сброс RS# с импульсом не менее 10 нс.

5.19.3 Режим IDLE3

В режиме IDLE3 выключаются все системы процессора, в том числе и PLL, и, в силу этого, потребляемая мощность в этом режиме минимальна. Кроме этого, режим IDLE3 позволяет переконфигурировать PLL извне, если требуется в целях экономии потребляемой мощности перейти на более низкую тактовую частоту.

Чтобы перевести процессор в режим IDLE3, используется команда IDLE 3. Завершается режим IDLE3 активизацией любого из выводов внешних прерывания (RS#, NMI# и INTx#) импульсом, ширина которого не должна быть меньше 10 нс. Если в этот момент INTM = 0, процессор переходит на подпрограмму обработки прерывания, в противном случае выполняется команда, следующая за IDLE3. Все пробуждающие прерывания должны быть разрешены в регистре IMR независимо от значения бита INTM. После выхода процессора из режима IDLE3 необходимо заново инициализировать все периферийные устройства, особенно, если они используют внешние синхросигналы. Это не обязательно делать, если в качестве пробуждающего прерывания используется сброс RS#, т. к. программа, запускаемая сначала, сделает все необходимые установки.

Если в процессе пробуждения процессора требования прерывания поступят от нескольких источников, первым обработается прерывание с самым высоким приоритетом

При выходе процессора из режима IDLE3 от сброса сигнал RS# должен быть активным в течение 50 мкс для обеспечения стабильности системной синхронизации во внутренней логике.

5.19.4 Режим запроса внешней шины HOLD

Режим HOLD позволяет переводить процессор в режим пониженного энергопотребления. В этом режиме линии адреса, данных и управления переводятся в высокоимпедансное состояние и, в зависимости от значения бита HM, может происходить полная остановка CPU.

Режим HOLD инициируется сигналом HOLD#. Действие сигнала HOLD# зависит от состояния бита HM. Если HM = 1, CPU останавливает выполнение программ, а линии адреса, данных и сигналов управления переводятся в высокоимпедансное состояние для понижения мощности. Если HM = 0, линии адреса, данных и сигналов управления переходят в высокоимпедансное состояние, но CPU продолжает выполнять внутреннюю программу. Использовать режим с HM = 0 можно тогда, когда системе не требуется доступ к внешней памяти.

Режим HOLD не останавливает операции во внутрикристальных периферийных устройствах (таймер и последовательные порты); они продолжают функционировать независимо от уровня сигнала HOLD# или состояния бита HM.

Этот режим завершается, когда сигнал HOLD# становится неактивным.

5.19.5 Другие возможности уменьшения энергопотребления

Частично снизить потребляемую процессором мощность можно отключением внешней шины и выхода тактового генератора CLKOUT.

– Отключение внешней шины позволяет запретить внутреннее тактирование внешних интерфейсов, устанавливая, таким образом, интерфейс в режим пониженного потребления энергии.

Отключение внешней шины происходит при установке в единицу младшего бита регистра управления переключением банков (BSCR). При сбросе процессора этот бит очищается, и доступ к внешней шине разрешен.

– Выход тактового генератора CLKOUT управляется программно модификацией бита CLKOFF регистра PMST, который определяет, когда CLKOUT запрещен или разрешен. После сброса выход CLKOUT включен.

5.20 Конвейер

5.20.1 Работа конвейера

1867ВЦ4Т имеет шестиуровневый конвейер команд. Шесть фаз конвейера независимы друг от друга, что дает возможность для перекрытия выполнения команд. В течение любого данного цикла, от одного до шести, различные команды могут быть активны, каждая в разных фазах завершения.

Шесть уровней и функций структуры конвейера:

– *Предварительная выборка*. Шина адреса программ (PAB) загружается адресом следующей команды для выборки.

– *Выборка команды*. Слово команды выбирается из шины программ (PB) и загружается в регистр команд (IR). На этом последовательность выборки команды завершается.

– *Декодирование команды*. Содержимое регистра команд (IR) декодируется с целью определения типа операции доступа к памяти и последовательности действий в устройстве генерации адреса данных (DAGEN) и CPU.

– *Доступ*. DAGEN выводит на шину адреса данных DAB адрес операнда для чтения. Если требуется второй операнд, другая шина адреса данных CAB также загружается соответствующим адресом. На этом же уровне модифицируются дополнительные регистры, использующиеся для косвенного способа адресации, и указатель стека (SP).

– *Чтение*. Операнды читаются с шин данных DB и CB. На этом последовательность чтения операндов завершается. В этой же фазе начинается последовательность записи операндов. Адрес операнда для записи загружается на шину адреса EAB. Для картируемых в памяти регистров операнд читается из памяти и записывается в указанные MMR регистры, используя шину DB.

– *Выполнение*. Последовательность записи операнда завершается записью данных, используя шину EB. В этой фазе происходит выполнение команды.

Первые две фазы конвейера, предвыборка и выборка, являются последовательностью выборки команд. В первой фазе загружается адрес новой команды. Во второй фазе читается слово команды. В случае многословных команд необходимо несколько таких последовательностей выборки команд.

В третьей фазе выбранная команда декодируется.

Две следующие фазы – доступ и чтение. Если требуется, адрес данных одного или двух операндов загружается в фазе доступа и операнд или операнды читаются в фазе чтения.

В следующей фазе команда выполняется, и результат операции по шине EB сохраняется в памяти.

На рисунке 39 представлено шесть уровней конвейера.

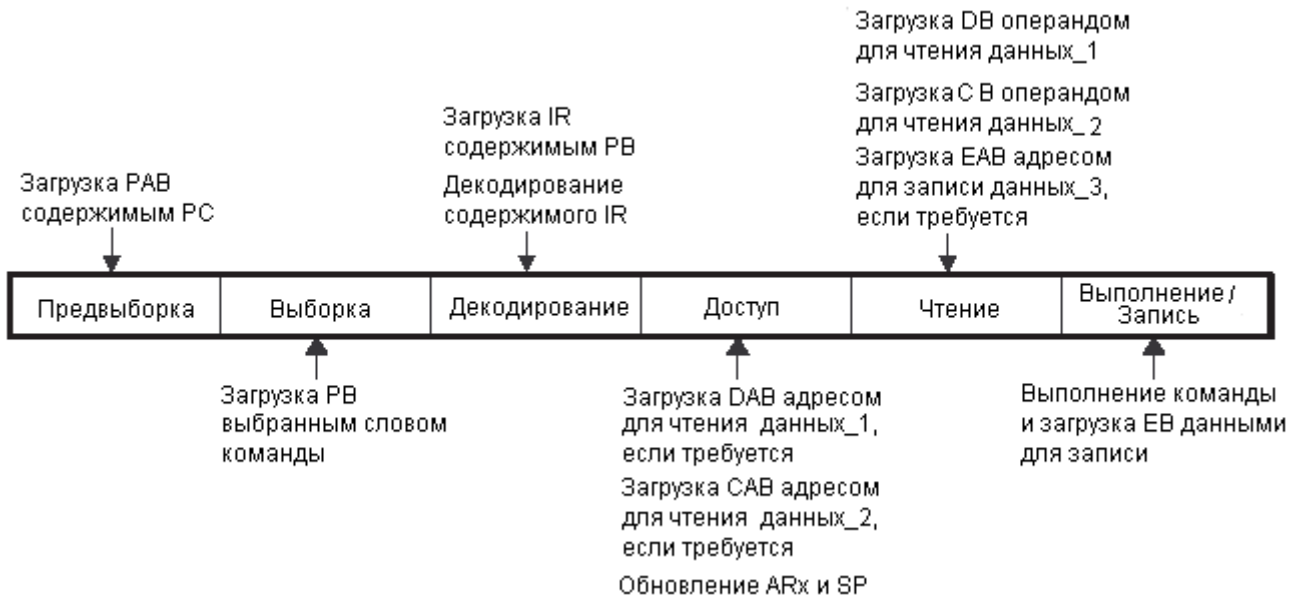


Рисунок 39 – Фазы Конвейера

Каждый доступ к памяти выполняется в двух фазах конвейера. В первой фазе шина адреса загружается адресом памяти, а во второй фазе соответствующая шина данных читает из или записывает в нее адрес памяти. Рисунок 40 иллюстрирует различные варианты доступа к памяти, реализуемые конвейером 1867ВЦ4Т. Предполагается, что все операции доступа выполняются одноцикловыми однословными командами во внутрикристальной памяти с двойным доступом (DARAM), позволяющей поддерживать два доступа в одном цикле конвейера.

(а) Выборка слова команды (одноцикловая)

Предвыборка	Выборка	Декодирование	Доступ	Чтение	Выполнение команды/запись
Загрузка PAB	Чтение из PB				

(б) Команда, выполняющая чтение одного операнда (например, LD *AR1,A, одноцикловая)

Предвыборка	Выборка	Декодирование	Доступ	Чтение	Выполнение команды/запись
			Загрузка DAB	Чтение из DB	

(в) Команда, выполняющая чтение двойного операнда (например, MAC *AR2+, *AR3+,A или DLD *AR2,A, одноцикловая)

Предвыборка	Выборка	Декодирование	Доступ	Чтение	Выполнение команды/запись
			Загрузка DAB и CAB	Чтение из DB и CB	

(г) Команда, выполняющая запись одного операнда (например, STH A, *AR1, одноцикловая)

Предвыборка	Выборка	Декодирование	Доступ	Чтение	Выполнение команды/запись
				Загрузка EAB	Запись в EB

(д) Команда, выполняющая запись двойного операнда (например, DST A, *AR1, два цикла)

Выборка	Декодиро-вание	Доступ	Чтение	Выполнение команды/запись
			Загрузка EAB	Запись в EAB

Предвыборка	Выборка	Декодиро-вание	Доступ	Чтение	Выполнение команды/запись
				Загрузка EAB	Запись в EB

(е) Команда, выполняющая чтение операнда и запись операнда (например, ST A, *AR2 ‖ LD *AR3, B, один цикл)

Предвыборка	Выборка	Декодирование	Доступ	Чтение	Выполнение команды/запись
			Загрузка DAB	Чтение из DB Загрузка EAB	Запись в EB

Рисунок 40 – Доступ к памяти через конвейер

5.20.2 Примеры выполнения команд в конвейере

5.20.2.1 Выполнение в конвейере команд перехода

Рисунки 41, 42 показывают состояние конвейера в течение выполнения команды перехода (B) и команды перехода с задержкой (BD), соответственно.

Так как команда перехода содержит два слова, она требует как минимум два цикла для полного выполнения. Тем не менее, для выполнения стандартной команды перехода необходимо четыре цикла (на рисунке 41 они выделены серым цветом). Это иллюстрируется на рисунке 42.

Адрес	Команда	
a1,a2	B b1	четырех цикловая двухсловная команда безусловного перехода
a3	i3	одноцикловая однословная команда
a4	i4	одноцикловая однословная команда
….	….	
b1	j1	

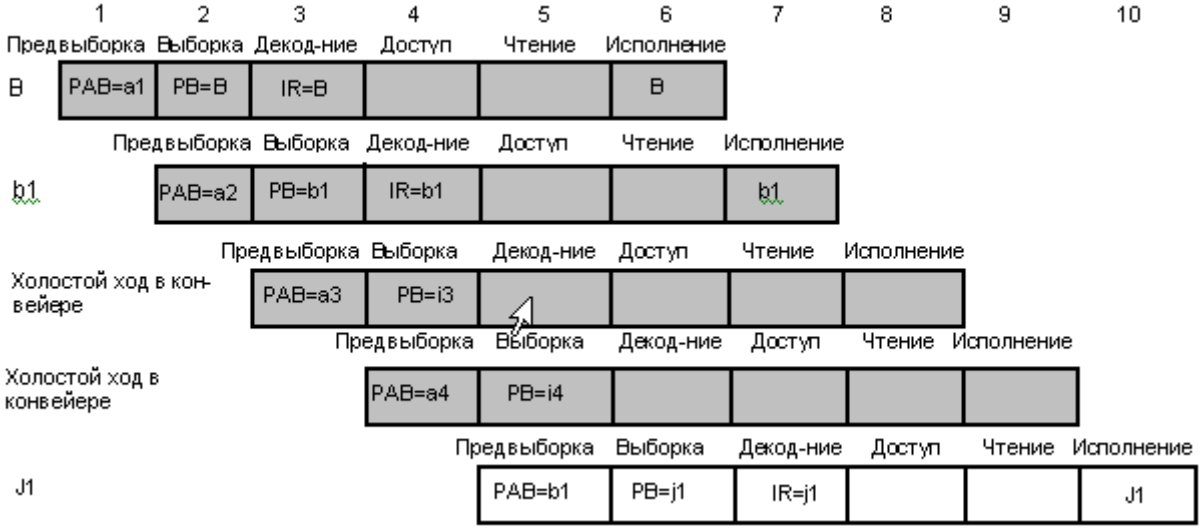


Рисунок 41 – Выполнение команды перехода (B) в конвейере

Адрес	Команда	
a1,a2	B b1	4-цикловая двухсловная команда безусловного перехода
a3	i3	одноцикловая однословная команда
a4	i4	одноцикловая однословная команда
….	….	
b1	j1	

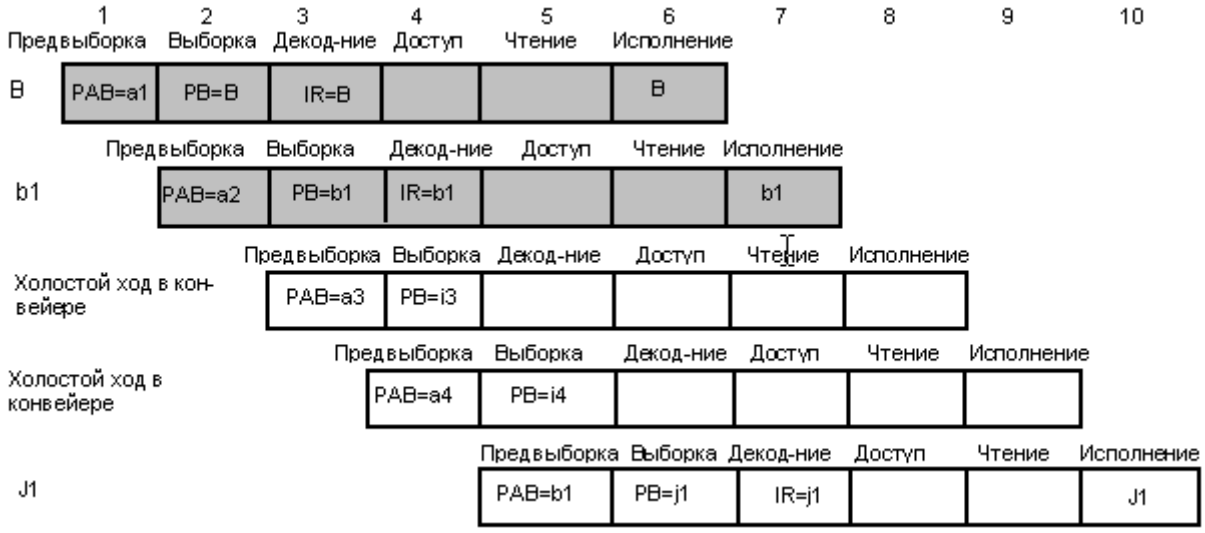


Рисунок 42 – Выполнение команды перехода с задержкой (BD) в конвейере

Для выполнения этой команды требуется два машинных цикла (на рисунке выделены серым цветом).

6 Внутрикристальная периферия

В устройствах 1867ВЦ4Т доступны следующие внутрикристальные периферийные устройства:

- внешние выводы ввода/вывода общего назначения: XF и BIO#;
- таймер;
- генератор тактовых синхроимпульсов;
- 8-разрядный стандартный интерфейс host-порта (HPI);
- буферизированный последовательный порт (BSP);
- мультиплексированный последовательный порт с разделением времени (TDM);
- программируемый генератор состояний ожидания;
- программируемый модуль переключения банков памяти.

6.1 Картируемые в памяти (MMR) периферийные регистры

Периферийные устройства доступны для обмена данными и управления через соответствующие MMR регистры, отображенные в памяти данных. Устанавливая или очищая биты в управляющих регистрах, можно разрешить, запретить, инициализировать и динамически реконфигурировать периферию. Операции в последовательных портах и таймере синхронизируются с CPU через прерывания или последовательным опросом прерываний. Когда периферийные устройства не используются, внутренние тактовые синхроимпульсы отключаются.

Периферийные MMR регистры отображаются на нулевой странице памяти данных. Список регистров периферийных устройств процессора 1867ВЦ4Т приведен в таблице 36.

Таблица 36 – Периферийные MMR регистры процессора1867ВЦ4Т

Адрес	Имя	Описание
20	BDRR	Регистр приема буферизированного последовательного порта
21	BDXR	Регистр передачи буферизированного последовательного порта
22	BSPC	Регистр управления буферизированным последовательным портом
23	BSPCE	Расширенный регистр управления буферизированным последовательным портом
24	TIM	Регистр таймера
25	PRD	Регистр периода таймера
26	TCR	Регистр управления таймером
27	-	Резерв
28	SWWSR	Программируемый регистр состояния ожидания
29	BSCR	Регистр управления переключением банков памяти
2A-2B	-	Резерв
2C	HPIC	Регистр управления интерфейсом ведущего порта
2D- 2F	-	Резерв
30	TRCV	Регистр приема TDM последовательного порта
31	TDXR	Регистр передачи TDM последовательного порта
32	TSPC	Регистр управления TDM последовательным портом
33	TCSR	Регистр выбора канала TDM последовательного порта
34	TRTA	Регистр приема/передачи TDM последовательного порта
35	TRAD	Регистр адреса приема TDM последовательного порта
36-37	-	Резерв
38	AXR	Регистр адреса передачи ABU
39	BKX	Регистр размера буфера передачи ABU
3A	ARR	Регистр адреса приемаABU
3B	BKR	Регистр размера буфера приема ABU
3C-5F	-	Резерв

6.2 Входы/выходы общего назначения

Устройство 1867ВЦ4Т имеет два вывода общего назначения I/O:
- вход управления переходом BIO#;
- выход внешнего флага XF.

6.2.1 Вывод управления переходом BIO#

BIO# предназначен для контроля состояния периферийных устройств. Он используется как альтернатива прерываниям. Состояние BIO# влияет на выполнение некоторых команд условного перехода. В командах условного выполнения оценка состояния BIO# осуществляется в течение декодирующей фазы конвейера; остальные условные команды (переход, вызов подпрограммы и возврат из нее) производят выборку BIO# в течение фазы чтения конвейера.

6.2.2 Вывод внешнего флага XF

Вывод XF управляется программным способом – установкой или сбросом соответствующего бита XF регистра состояния ST1 командами SSBX и RSBX. После сброса процессора бит XF также устанавливается в 1. На рисунке 43 представлена временная диаграмма переключения бита XF в последовательности одноцикловых команд.

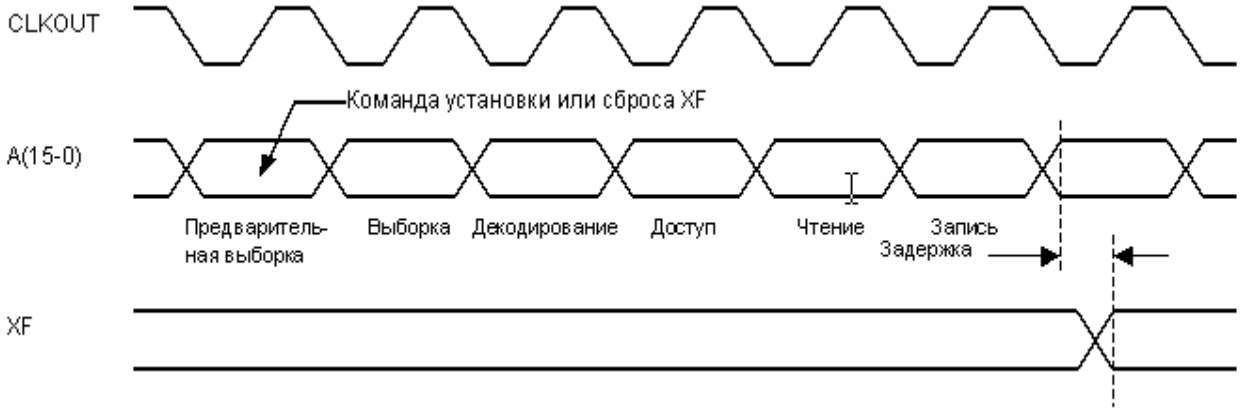


Рисунок 43 – Временная диаграмма переключения сигнала XF

6.3 Таймер

Таймер – это внутрикристальный счетчик, который состоит из трех регистров и может использоваться для генерирования периодических прерываний.

6.3.1 Регистры таймера

В таблице 37 представлены три регистра таймера:

Таблица 37 – Регистры таймера

Адрес	Регистр	Описание
0024h	TIM	Таймерный регистр
0025h	PRD	Регистр периода таймера
0026h	TCR	Регистр управления таймером

– Регистр таймера (TIM). 16-разрядный регистр таймера загружается значением регистра периода PRD, уменьшается на единицу.

– Регистр периода таймера (PRD). 16-разрядный регистр периода таймера используется для перезагрузки регистра таймера (TIM).

– Регистр управления таймером (TCR). 16-разрядный регистр управления таймером содержит биты управления и состояния таймера. Биты TCR показаны на рисунке 44 и описаны в таблице 38.

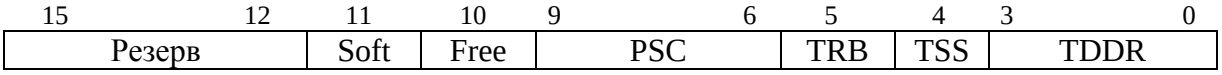


Рисунок 44 – Диаграмма регистра управления таймером

Таблица 38 – Описание битов управляющего регистра таймера

Бит	Имя	Значение после сброса	Функция
15-12	Резерв	-	Резервные; читаются всегда как 0
11	Soft	0	Используется совместно с битом Free и определяет состояние таймера в точке останова программы при отладке. Когда бит Free очищен, бит Soft выбирает режим таймера. Soft = 0 – таймер останавливается немедленно. Soft = 1 – таймер останавливается, когда счетчик переходит в 0
10	Free	0	Используется совместно с битом Soft и определяет состояние таймера в точке останова программы при отладке. Когда бит Free очищен, бит Soft выбирает режим таймера. Free = 0 – бит Soft выбирает режим таймера. Free = 1 – таймер продолжает функционировать независимо от бита Soft
9-6	PSC	-	Предварительный счетчик таймера. Определяет счет для основного счетчика таймера. Когда содержимое PSC уменьшается до нуля или после сброса таймера, PSC загружается содержимым TDDR, а содержимое TIM уменьшается на единицу
5	TRB	-	Перезагрузка таймера. Инициализирует внутрикристальный таймер. Когда TRB установлен, TIM загружается значением PRD, а PSC загружается содержимым TDDR. TRB всегда читается как 0.
4	TSS	0	Останов таймера. Запрещает или разрешает функционирование внутрикристального таймера. После сброса TSS очищается, и таймер немедленно начинает считать. TSS = 0 – таймер запускается. TSS = 1 – таймер останавливается
3-0	TDDR	0000	Коэффициент деления для поступающих импульсов. Когда PSC уменьшается до нуля, он перезагружается содержимым TDDR

6.3.2 Операции таймера

Таймер – это внутрикристальный счетчик, который используется для генерации периодических прерываний CPU. Каждый раз, когда содержимое таймера уменьшается до нуля, генерируется прерывание по таймеру (TINT) и счетчик перезагружается содержимым PRD.

Рисунок 45 представляет логическую блок-схему таймера. Она состоит из двух основных блоков: блока основного таймера, содержащего PRD и TIM, блока предварительного счетчика, содержащего биты TDDR и PCS. Таймер тактируется синхронизацией CPU.

Содержимое регистра PRD загружается в счетчик таймера TIM, когда последний уменьшается до нуля. Содержимое регистра PRD также загружается при сбросе устройства (SRESET#) или при индивидуальном сбросе таймера (TRB установлен). Каждый выходной тактовый импульс из блока предварительного счетчика уменьшает содержимое TIM на единицу. Выход блока основного счетчика – сигнал прерывания по таймеру (TINT), который пересылается в CPU или на внешний вывод TOUT. Длительность импульса TOUT эквивалентна периоду CLKOUT.

Частоту прерываний таймера (TINT) можно выразить формулой:

$$TINT = \frac{1}{tc(C) \cdot U \cdot V} = \frac{1}{tc(C) \cdot (TDDR+1) \cdot (PRD+1)},$$

где $tc(C)$ – период CLKOUT1, U – сумма содержимого TDDR+1, V – сумма содержимого TDDR+1.

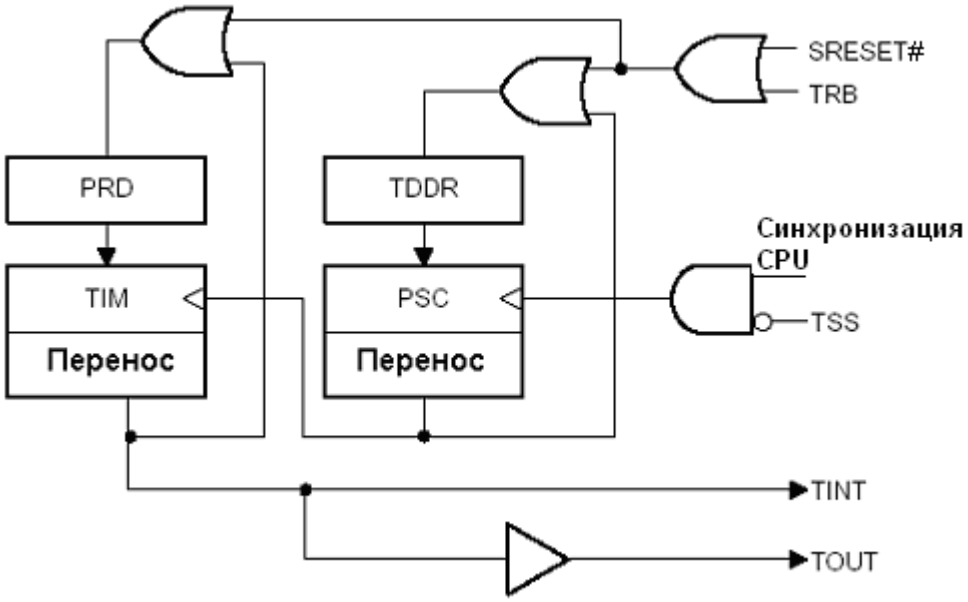


Рисунок 45 – Блок-схема таймера

Счетчик PSC загружается значением из TDDR, когда PSC уменьшается до нуля.

Содержимое TDDR также загружается в PSC при сбросе процессора или установкой бита TRB. Содержимое PSC можно считать, обратившись к регистру TCR, но нельзя загрузить непосредственно.

Таймер может быть остановлен, используя бит TSS для отключения входа синхронизации в таймер. Это позволяет сократить уровень потребляемой процессором мощности.

Таймер обеспечивает удобный и эффективный способ генерации системного времени для аналогового интерфейса.

Инициализация таймера:

– Остановить таймер, записав 1 в бит TSS регистра TCR.
– Загрузить PRD.
– Установить бит TSS в 0 и бит TRB в 1, для перезагрузки периода таймера.

Прерывание таймера может быть разрешено (если INTM = 1):

– Очищением любого задержанного прерывания по таймеру установкой бита TINT регистра IFR в 1.
– Разрешением прерывания по таймеру установкой бита TINT регистра IMR в 1.
– Разрешением всех прерываний очисткой бита INTM в 0.

При сбросе TIM и PRD устанавливаются в максимальное значение FFFFh. Поле TDDR регистра TCR очищается в ноль и таймер запускается.

6.4 Генератор

В качестве источника тактовой синхронизации могут быть использованы следующие устройства:

– внутренний генератор с подключенным к его выводам X1 и X2/CLKIN внешним кварцевым резонатором;
– внешний генератор, подключенный непосредственно к выводу X2/CLKIN, при этом вывод X1 остается не задействованным.

Задающий генератор на устройстве 1867ВЦ4Т состоит из внутреннего генератора и схемы с фазовой автоподстройкой частоты входного сигнала (Phase-Locked Loop – PLL).

6.4.1 Аппаратно конфигурируемый генератор

Схема PLL может запускаться от источника внешних колебаний, период которых больше машинного цикла CPU. Этот вариант позволяет уменьшить высокочастотный шум, возникающий в результате работы высокоскоростных схем. Сигнал с выхода внутреннего или внешнего генератора подается в PLL, который умножает его частоту на константу N (PLL X N). При использовании внутреннего генератора тактовая частота CPU равна частоте генератора поделенной на два. При использовании внешнего генератора тактовая частота CPU равна частоте входного сигнала умноженной на N.

Максимальная частота сигнала, действующего на входе PLL не должна превышать 40 МГц для 25 нс 1867ВЦ4Т устройств. После сброса или восстановления из режима IDLE3 схема PLL требует блокировки в течение 50 мкс. Режим PLL определяется состоянием выводов CLKMD1, CLKMD2 и CLKMD3. В таблице 39 представлены все режимы PLL в функции состояния выводов CLKMD1, CLKMD2 и CLKMD3. В том случае, когда PLL не используется, тактовая частота CPU равна половине частоты кварца или частоты внешнего генератора.

Переконфигурирование PLL изменением кода на внешних выводах возможно лишь режиме IDLE3 после того, как CLKOUT перейдет в состояние с высоким уровнем.

Таблица 39 – Режимы задающего генератора

Выбор режима конфигурацией выводов			Режим задающего генератора
CLKMD1	CLKMD2	CLKMD3	
0	0	0	PLLx3 от внешнего генератора
1	1	0	PLLx2 от внешнего генератора
1	0	0	PLLx3 от внутреннего генератора
0	1	0	PLLx1,5 от внешнего генератора

Окончание таблицы 39

Выбор режима конфигурацией выводов			Режим задающего генератора
CLKMD1	CLKMD2	CLKMD3	
0	0	1	От внешнего генератора, деленная пополам
1	1	1	От внутреннего генератора, деленная пополам
1	0	1	PLLx1 от внешнего генератора
0	1	1	Режим останова[1)]
[1)] PLL заблокирован. Сигнал тактовой частоты не подается на CPU и внешние устройства. Этот режим эквивалентен режиму пониженного потребления IDLE3, однако режим IDLE 3 предпочтительнее, т. к. он управляется прерываниями.			

6.5 Host-интерфейс (HPI)

HPI подключается к host-устройству как периферия, для облегчения доступа к устройству 1867ВЦ4Т. Host-устройство взаимодействует с HPI через регистры адреса и данных, к которым процессор не имеет прямого доступа, используя внешние данные и сигналы управления интерфейсом (рисунок 46). Как host-устройство, так и процессор имеют доступ к управляющему регистру HPI.

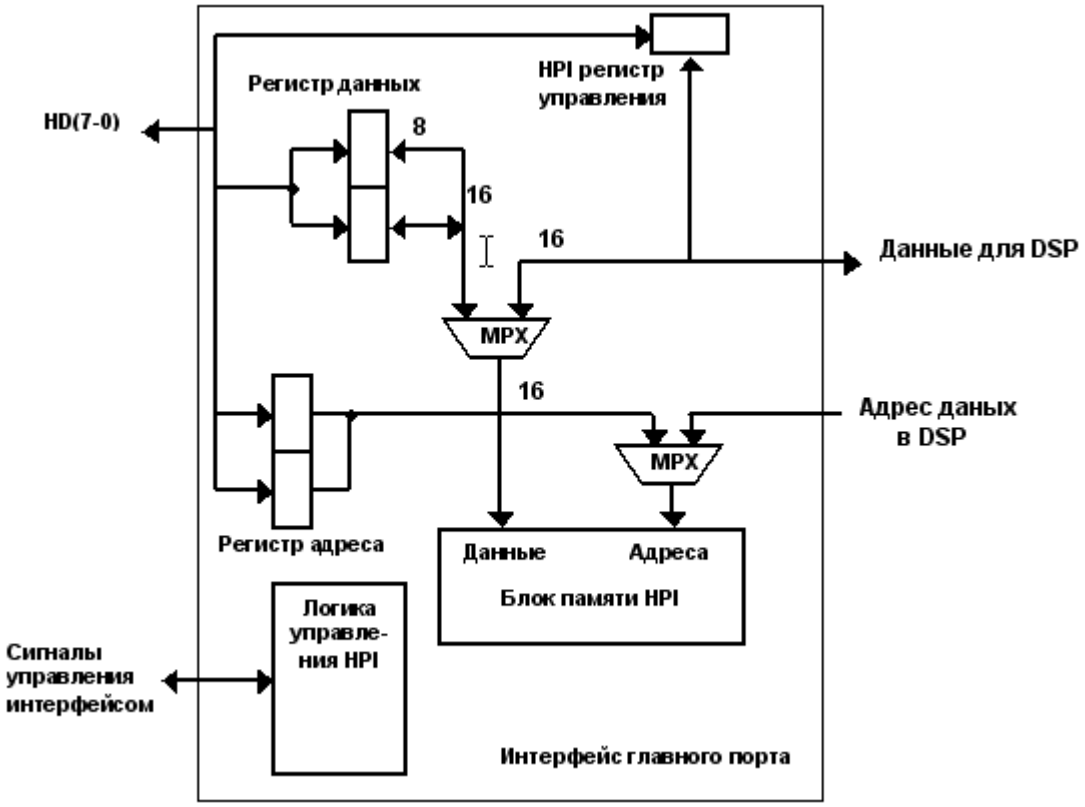


Рисунок 46 – Функциональная схема интерфейса host-порта

HPI обеспечивает передачу 16-разрядных данных в 1867ВЦ4Т при поддержке экономичного 8-разрядного внешнего интерфейса, автоматически объединяя перемещенные последовательные байты в 16-разрядные слова. Когда host-устройство выполняет передачу данных в регистры HPI, логика управления HPI автоматически обеспечивает доступ к специальному 2K-словному блоку внутренней памяти DARAM. В свою очередь процессор также может обращаться к этим данным.

HPI имеет два режима работы: режим распределенного доступа (SAM) и режим только для host-устройства (HOM). В режиме распределенного доступа (нормальный режим работы) и процессор и host-устройство могут обращаться к HPI памяти. В этом режиме доступ к памяти от обоих устройств синхронизирован внутренней схемой процессора и, в случае конфликта между 1867ВЦ4Т и host-устройством, host-устройство имеет больший приоритет доступа, и процессор ожидает один цикл. В режиме HOM только host-устройство может обращаться к HPI памяти, в то время как процессор находится в сбросе или в режиме IDLE2 с остановленными внутренними и внешними тактовыми импульсами. Следовательно, host-устройство может обращаться к HPI RAM, в то время как процессор находится в режиме минимального потребления энергии.

HPI поддерживает высокую скорость обмена в обоих направлениях. В режиме распределенного доступа HPI может передать один байт каждые пять циклов CLKOUT (то есть 64 Мбит/с) при частоте тактового генератора 40 МГц. HPI позволяет host-устройству обмениваться данными на частотах до (Fd*n)/5, где Fd – частота CLKOUT, а n – число циклов host-а для внешнего доступа. Следовательно, при 40 МГц и с типовыми значениями для n равными 4 или 3, host-устройство может обмениваться данными на скоростях до 32 или 24 МГц без состояний ожидания. В режиме HOM HPI поддерживает более высокую скорость обмена в обе стороны порядка одного байта на каждые 50 нс (т. е. 160 Мбит/с), независимо от тактовой частоты процессора.

6.5.1 Функциональное описание host-интерфейса

Внешний интерфейс HPI состоит из 8-разрядной HPI шины данных и сигналов управления, которые конфигурируют интерфейс и управляют им. Интерфейс может соединяться с рядом host-устройств с минимальной дополнительной логикой или без нее. Рисунок 47 показывает упрощенную схему подключения HPI к host-устройству.

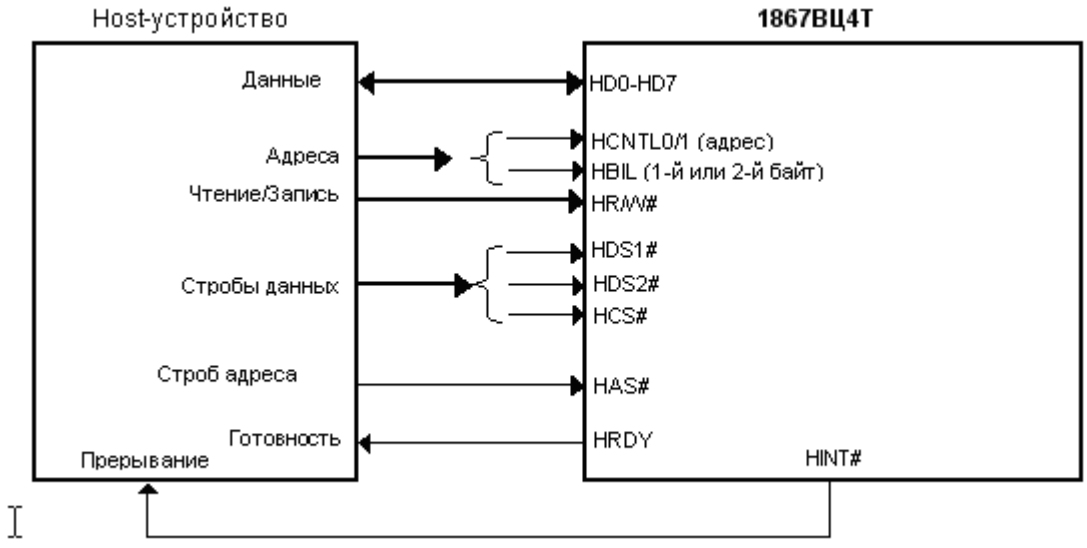


Рисунок 47 – Общая функциональная схема системы

8-разрядная шина данных (HD0-HD7) предназначена для обмена данными с host-устройством. Поскольку структура данных в процессоре 16-разрядная, каждое слово должно перемещаться в виде двух последовательных байтов. Специальный вывод HBIL указывает, какой из байт (старший или младший) перемещается в данный момент. Внутреннее управление в соответствии с этим битом расставляет байты в 16-разрядном слове. Host-устройство не должно разрывать последовательность первый/второй байт

(HBIL низкий/высокий) при обмене через HPI. Если эта последовательность будет нарушена, данные окажутся потерянными, что приведет к непредсказуемому результату.

Два входа управления (HCNTL0 и HCNTL1) указывают, к какому из внутренних регистров HPI идет обращение и тип доступа к регистру. Эти входы, наряду с HBIL, обычно управляются значащими битами шины адреса или функцией этих битов. При использовании входов HCNTL0/1, host может выбрать один из регистров HPI: регистр управления (HPIC), регистр адреса (HPIA), используемый как указатель на ячейку памяти данных или регистр данных (HPID). К регистру HPID можно также обращаться с автоматическим приращением адреса.

Автоинкремент является удобным средством при обращении к последовательно расположенным ячейкам памяти. В режиме с автоинкрементом адрес ячейки увеличивается после чтения каждого слова и перед записью каждого слова. Записывая данные в HPIC, host может прерывать процессор, а выход HINT# может использоваться процессором 1867ВЦ4Т для прерывания host-а. Host-устройство может также подтверждать прерывание и очищать HINT#, записывая соответствующие данные в HPIC.

В таблице 40 представлено описание трех регистров, которые HPI использует для связи между host-ом и процессором.

Таблица 40 – Описание регистров HPI

Имя	Адрес	Описание
HPIA	-	Регистр адреса HPI. Непосредственно доступен только для host-а. Содержит адрес ячейки памяти HPI, с которой производится текущий обмен
HPIC	002ch	Регистр управления HPI. Доступен как для host-устройства, так и для процессора. Содержит биты управления и состояния для операций HPI
HPID	-	Регистр данных HPI. Непосредственно доступен только для host-а. Содержит данные, которые читаются из памяти HPI, если текущая операция – чтение, или данные, которые будут записаны в память HPI, если текущая операция – запись

Два строба данных (HDS1# и HDS2#), строб для чтения/записи (HR/W#) и строб адреса (HAS#) позволяют связать HPI с различными host-устройствами, соответствующими промышленному стандарту, с использованием минимального количества дополнительной логики или без нее. HPI соединяется с host-устройствами через мультиплексированную шину адреса/данных, стробы данных и чтения/записи, или два отдельных строба для чтения и записи.

Сигнал готовности (HRDY) извещает host-устройство о завершении операции обмена в HPI и позволяет вставлять циклы состояния ожидания в случае, если быстродействие host-а выше, чем процессора. В этом случае host приостанавливает выполнение своей программы до момента появления квитанции HRDY. Таким образом, вывод HRDY автоматически обеспечивает аппаратную регулировку скорости обмена между host-устройством и процессором, что позволяет HPI обеспечивать гибкую и эффективную связь с широким рядом устройств промышленного стандарта.

6.5.2 Детальное описание host-интерфейса

Внешние сигналы интерфейса HPI обеспечивают гибкий интерфейс с рядом типовых host-устройств. Устройства с одним или несколькими стробами данных, адреса могут быть легко подключены к HPI.

Таблица 41 дает подробное описание функции каждого вывода HPI.

Таблица 41 – Названия и функции сигналов HPI

Вывод HPI	Host-вывод	Состо-яние	Функция сигнала
HAS#	Строб защелки адреса (ALE) или строб адреса	I	Вход строба адреса. Host-устройства с мультиплексированной шиной адреса и данных подключают к HAS# свой выход ALE или эквивалент его. Сигналы HBIL, HCNTL0/1 и HR/W# защелкиваются падающим фронтом HAS#. Сигнал HAS# должен быть задержан относительно сигналов HCS#, HDS1# или HDS2#
HBIL	Линии адреса или управления	I	Вход идентификации байта. Идентифицирует первый или второй передаваемый байт (но не старший или младший – эти байты определяются битом BOB в регистре HPIC). HBIL равен 0 для первого байта и 1 для второго
HCNTL0 HCNTL1	Линии адреса или управления	I	Выбор кристалла. Служит в качестве разрешающего входа для HPI и должен находиться в нуле в процессе обмена, но может оставаться в нуле между доступами. HCS# обычно опережает HDS1# и HDS2#. Если строб HAS# не используется и выводы HDS1# или HDS2# находятся в состоянии с низким уровнем, этот сигнал стробирует HCNTL0/1, HR/W# и HBIL. На рисунке 48 представлена эквивалентная схема входов HCS#, HDS1# и HDS2#
HCS#	Линии адреса или управления	I	Параллельная двунаправленная шина данных с 3-мя состояниями. Шина устанавливается в высокоимпедансное состояние, когда (HDSx# | HCS# = 1) или когда EMU1/OFF# активен (равен нулю)
HD0-HD7	Шина данных	I/O/Z	Параллельная двунаправленная шина данных с 3-мя состояниями. Шина устанавливается в высокоимпедансное состояние, когда (HDSx# | HCS# = 1) или когда EMU1/OFF# активен (равен нулю)
HDS1# HDS2#	Строб чтения/за-писи или строб данных	I	Входы строба данных. Управляют обменом данными в процессе доступа host-а. Когда HAS# не используется и HCS# находится в нуле, они производят выборку HBIL, HCNTL0/1 и HR/W#. Host-устройства с раздельными стробами чтения и записи подключаются к входам HDS1# и HDS2#. Host-ы с одним стробом данных подключают или к HDS1#, или к HDS2#. Неиспользуемый вывод привязывают к высокому уровню. Вход HR/W# определяет направление передачи. Так как HDS1# и HDS2# объединены внутренней схемой исключающего ИЛИ с инверсией по выходу, host с положительным стробом данных можно соединить с одним из входов HDS#. При этом другой вход HDS# должен находиться в состоянии с низким уровнем (рисунок 48)

Окончание таблицы 41

Вывод HPI	Host-вывод	Состо-яние	Функция сигнала
HINT#	Выход прерывания host-a	O/Z	Выход прерывания host-a. Управляется битом HINT регистра HPIC. Устанавливается в единицу, когда процессор сбрасывается. Переходит в третье состояние, когда EMU1/OFF# = 0
HRDY	Сигнал асинхрон-ной готовности	O/Z	Выход готовности HPI. Когда HRDY = 1, HPI готов к обмену, в противном случае следует подождать завершения обмена. Когда EMU1/OFF# активен, переводится в третье состояние. HRDY = 1 всегда, когда HCS# = 1
HR/W#	Строб чтения/записи, линия адреса, или адрес/ данные	I	Вход чтения/записи. Host-устройства должны устанавливать HR/W# в 1 для чтения HPI и в 0 для записи в HPI. Host-устройства без строба чтения/записи могут для этой цели использовать линию адреса
Условные обозначения: I – вход, O-выход, Z-третье состояние.			

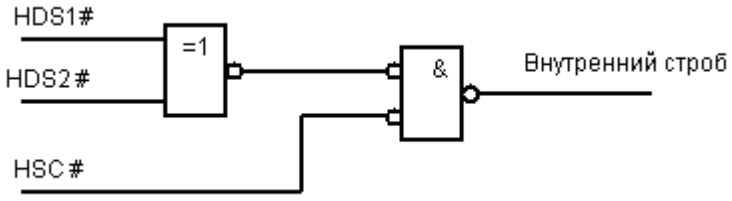


Рисунок 48 – Логика входов выбора

Три управляющих сигнала HAS# (сигнал выбора порта HPI), HDS1# и HDS2# (стробы данных HPI) формируют внутренний строб HPI (рисунок 48, где представлена логика управляющих сигналов HAS#, HDS1# и HDS2#). Сигнал готовности порта HRDY = 1 всегда, когда HAS# = 1 (случай, когда HAS# используется для управления циклами доступа к HPI вместо HDS1# и HDS2#). Поскольку HDS1# и HDS2# объединены вентилем, исполняющим функцию исключающего ИЛИ-НЕ, оба эти входа должны быть в 0.

Использование синхросигнала HAS# позволяет раньше выставлять сигналы HCNTL0/1, HBIL и HR/W# в HPI, что упрощает связь с мультиплексированными шинами адреса/данных вследствие предоставления большего промежутка времени для подключения шин. В системе такого типа часто предусматривается сигнал ALE, который обычно замыкается на вход HAS#.

Два управляющих входа (HCNTL0 и HCNTL1) указывают, к какому из внутренних регистров HPI идет обращение и тип обращения. Возможен один из четырех вариантов: обращение к управляющему регистру (HPIC), к регистру адреса (HPIA) с инкрементом и без него и к регистру данных (HPID). Регистр HPIA служит в качестве указателя ячейки памяти HPI, регистр HPIC содержит биты управления и флаги обмена и регистр HPID содержит перемещаемые данные. Адрес ячейки памяти HPI может задаваться предварительной записью в регистр HPIA или генерироваться с помощью механизма приращения предыдущего содержимого регистра HPIA (автоинкремент). Таблица 42 описывает функции битов HCNTL0/1.

Таблица 42 – Соответствие управляющих кодов на входах HCNT0/1 к выбираемому регистру

HCNT0	HCNT1	Описание
0	0	Host может читать или записывать в управляющий регистр HPIC
0	1	Host может читать или записывать в регистр HPID. Адрес автоматически увеличивается после операции чтения или перед операцией записи
1	0	Host может читать или записывать в регистр адреса HPIA. Регистр HPIA является указателем на ячейку памяти HPI
1	1	Host может читать или записывать в регистр данных HPID. Содержимое регистра адреса не изменяется

В процессоре 1867ВЦ4Т память HPI занимает пространство в (2K×16) бит области DARAM данных или программ (в зависимости от состояния бита OVLY) и расположена в диапазоне адресов с 1000h до 17FFh.

Host-устройству этот блок доступен по адресам 0-7FFh, т. к. регистр адреса HPIA подключен только к 11-ти младшим разрядам шины адреса кристалла. Например, обращение к первому слову HPI блока памяти, имеющему адрес 1000h в пространстве памяти данных процессора, возможно, если в регистре адреса HPIA содержится любой из следующих адресов: 0000h, 0800h,1000h,1800h... F800h.

Режим автоматического приращения адреса в HPIA обеспечивает удобный доступ к последовательно расположенным ячейкам в памяти HPI. В режиме автоприращения данные читаются после увеличения HPIA, а запись данных происходит перед увеличением HPIA. Следовательно, если запись должна быть сделана в первое слово памяти HPI с опцией приращения, HPIA должно предварительно загружаться любым из следующих адресов: 07FFh, 0FFFh, 17FFh,... FFFFh. HPIA – 16-битный регистр и все 16 бит могут быть записаны или считаны, хотя для адресации в памяти HPI требуется только 11 младших бит HPIA. В режиме с автоинкрементом участвуют все 16 бит этого регистра.

Режимом работы HPI управляют четыре бита. Эти биты располагаются в управляющем регистре HPIC и выполняют следующие функции:

– BOB определяет первый или второй байт как старший;

– SMOD выбирает режим главный или коллективного доступа;

– DSPINT и HINT разрешают прерывания, генерируемые процессором и host-устройством соответственно.

Подробное описание битных функций HPIC представлено в таблице 43.

Таблица 43 – Описание битов управляющего регистра HPIC

Бит	Host-доступ	Доступ процессора	Описание
BOB	Чтение\ Запись	-	Если BOB = 1, первый байт - младший значащий. Если BOB = 0, первый байт - значащий старший. BOB влияет как на данные, так и на адреса. Модифицировать этот бит может только host-устройство, для процессора он не видим. BOB должен быть проинициализирован перед первым доступом к регистру данных или адреса

Окончание таблицы 43

Бит	Host-доступ	Доступ процессора	Описание
SMOD	Чтение	Чтение\ Запись	Если SMOD = 1, установлен режим коллективного доступа (SAM). В этом режиме память HPI может быть доступна процессору. Если SMOD = 0, установлен режим HOM и процессор не имеет доступ ко всему блоку RAM HPI. SMOD = 0 в момент сброса; SMOD = 1 после сброса. SMOD модифицируется только процессором, но может быть прочитан как процессором, так и host-устройством
DSPINT	Запись	-	Прерывание процессора от host-а. Этот бит может быть записан только host-ом и не доступен для чтения ни host-у, ни процессору. Когда host устанавливает в 1, этот бит генерирует прерывание для процессора. Когда host записывает в HPIC, оба бита должны записывать одно и тоже значение
HINT	Чтение\ Запись	Чтение\ Запись	Этот бит определяет состояние внешнего вывода HINT# процессора, который может использоваться для генерации прерывания для host-устройства. При сбросе HINT# = 0, в результате чего внешний вывод HINT# переводится в состояние с высоким уровнем. Бит HINT может быть установлен только процессором, а сброшен только host-ом. Когда процессор записывает 1 в HINT, внешний вывод HINT# переходит в состояние с низким уровнем. Бит HINT читается host-ом или процессором как 0, когда внешний вывод HINT# находится в состоянии с высоким уровнем, и как 1, когда вывод HINT# находится в состоянии с низким уровнем. Чтобы сбросить прерывание host должен записать 1 в HINT

Поскольку host-интерфейс обменивается всегда байтами, а управляющий регистр – первый регистр, доступный для установки битов конфигурации и инициализации интерфейса, со стороны host-а HPIC организовывается как 16-битный регистр с одинаковыми старшим и младшим байтами (хотя доступ к определенным битам ограничивается, как указано выше), причем со стороны процессора старший байт не используется. Биты управления/состояния располагаются в четырех младших разрядах. Host выбирает HPIC регистр соответствующей установкой бит HCNTL0/1, как указано выше. Первый и второй байты, предназначенные для записи в HPIC должны быть одинаковые. Процессор имеет доступ к HPIC по адресу 002Ch в пространстве памяти данных.

Формат регистра HPIC представлен на рисунках 49-52. Обозначения на рисунках для операций чтения следующие:

0 – только для чтения;

X – биты не определены.

Для операций записи:

X – может быть записаны любые значения.

При записи из host-а оба байта должны быть одинаковыми. Биты 4-7 и 12-15 со стороны host-а, биты 4-15 и со стороны процессора резервируются для будущего расширения.

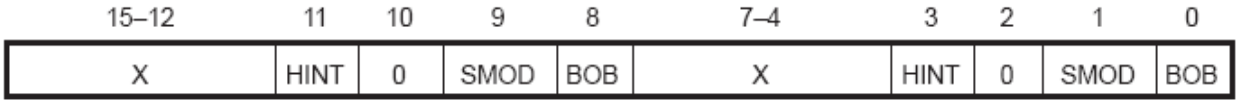


Рисунок 49 – Регистр HPIC для чтения со стороны host-а

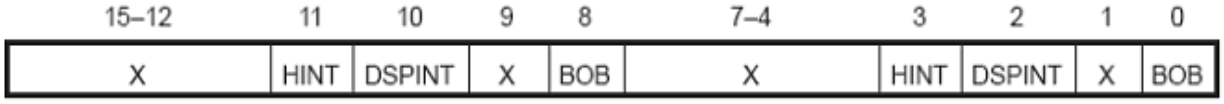


Рисунок 50 – Регистр HPIC для записи со стороны host-а

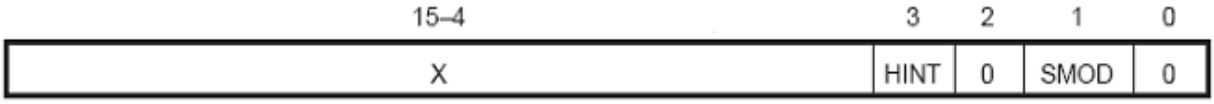


Рисунок 51 – Регистр HPIC для чтения со стороны процессора

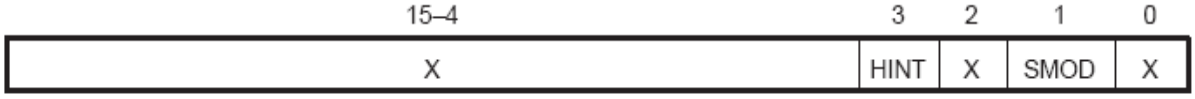


Рисунок 52 – Регистр HPIC для записи со стороны процессора

Поскольку процессор может записывать значения в биты SMOD и HINT и эти биты читаются host-интерфейсом дважды, чтение первого и второго байта для host-а может дать разные результаты, если процессор модифицирует один или оба эти бита между двумя операциями чтения. Характеристики циклов чтения/записи регистра HPIC представлены в таблице 43.

Таблица 43 – Характеристики чтения/записи регистра HPIC host-ом и процессором

Устройство	Чтение	Запись
Host	Два байта	Два одинаковых байта
Процессор	16 бит	16 бит

6.5.3 Чтение/запись из host-а через HPI

Host-устройство начинает доступ к HPI с инициализации регистра HPIC, затем регистра HPIA, после чего осуществляет запись или чтение данных через регистр HPID. Запись в HPIA или HPID инициирует внутренний цикл обмена, который обеспечивает передачу данных между HPID и внутренней памятью HPI. Поскольку для этого процесса требуется несколько циклов процессора, каждый раз при осуществлении доступа к HPI данные, поступающие в HPID, могут быть записаны в память HPI только по завершении текущего цикла доступа, а данные, читаемые из HPID – это данные из предыдущего цикла. Следовательно, при чтении полученные данные – это данные из ячейки, выбранной при предыдущем доступе, в то время как текущий доступ инициализирует следующий цикл обмена. Таким образом, если операция чтения из HPID следует непосредственно за операцией записи в HPID, считываются последние записанные в HPID данные.

Использование автоприращения для HPIA обеспечивает эффективный доступ host-а к последовательно расположенным данным в памяти HPI. При произвольных обращениях или при последовательном доступе через большие интервалы времени процессор может изменить содержимое ячеек памяти, доступных между чтением host-а и предыдущей

операцией чтения/записи данных или доступом для записи HPIA. В таких случаях следует выполнять два чтения из одного и того же предварительно модифицированного адреса.

Возможны два различных варианта обращения host-а к HPI: с использованием генерации состояний ожидания (сигнал готовности HRDY активен) и без состояний ожидания. Обычно сигнал HRDY используется в режиме коллективного доступа (SAM), а в режиме (HOM) сигнал HRDY устанавливается в состояние с высоким уровнем. Однако есть исключения, которые будут рассмотрены ниже.

Если HRDY используется, то на время выполнения внутренней передачи по чтению или записи HRDY переводится в состояние с низким уровнем, что указывает на запрет следующей передачи. После того, как внутренний цикл завершен, сигнал HRDY переводится в состояние с высоким уровнем, разрешая host-у начать следующую передачу. Это происходит через фиксированный промежуток времени после определения цикла обращения. В процессе передачи первого байта сигнал HRDY находится в состоянии с высоким уровнем. Внешний цикл HPI при использовании HRDY показан на рисунке 53.

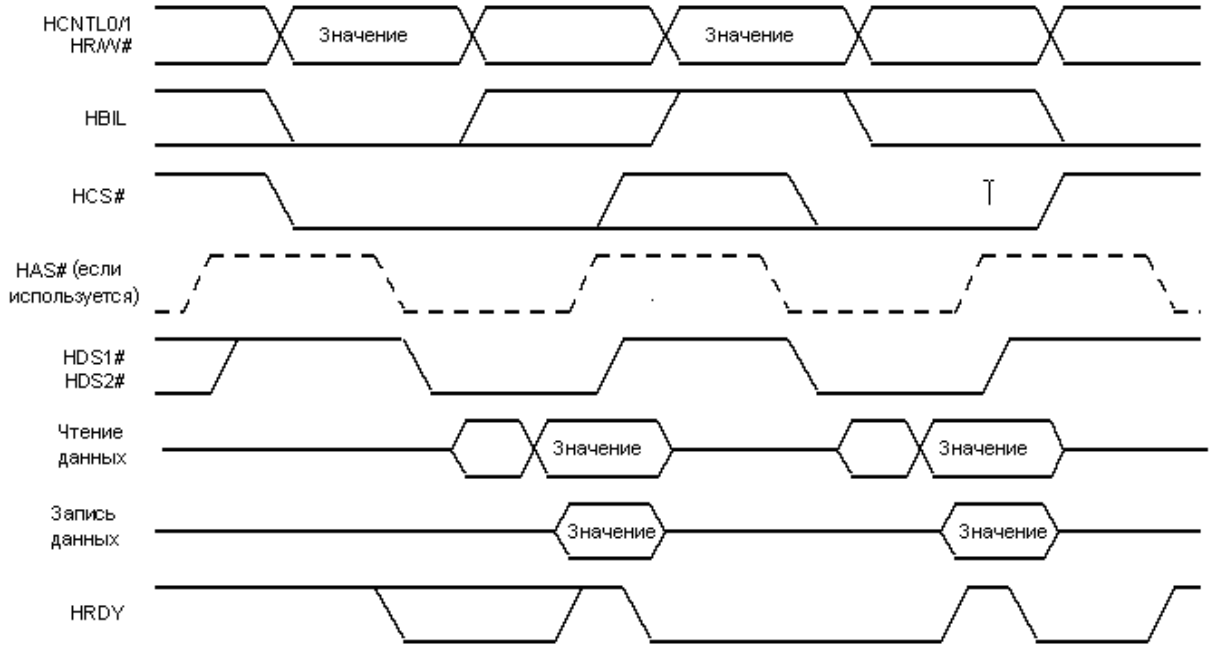


Рисунок 53 – Временная диаграмма HPI

При обычном внешнем обращении к порту HPI (см. рисунок 53) host начинает генерировать сигналы HCNTL0/1, HR/W#, HBIL и HAS#, указывая какой тип передачи должен произойти и в каком направлении: чтение или запись. Затем host генерирует сигнал HAS# (если он нужен), за которым следует один из сигналов строба данных. Если сигнал HRDY до этого находился на нижнем уровне, он переводится в единицу после завершения предыдущего внутреннего цикла. Сигналы управления при этом игнорируются. После завершения внешнего цикла HPI сигнал HRDY переходит на низкий уровень и остается таким в течение примерно пяти циклов CLKOUT, необходимых для завершения процессором внутреннего доступа к памяти HPI, после чего уровень HRDY становится снова высоким. Отметим, что сигнал HRDY всегда в единице, когда HCS# в единице.

Как говорилось выше, при обменах типа SAM обычно используют сигнал HRDY. Исключением являются ситуации при чтении HPIC, HPIA или при записи в HPIC (за исключением записи 1 в DSPINT или HINT). В этих случаях HRDY остается в состоянии с высоким уровнем.

Временные соотношения циклов доступа host-а в режиме HOM отличаются от временных соотношений в режиме SAM, описанных прежде. В режиме HOM CPU не участвует в пересылках через порт HPI, и обмен может быть завершен за достаточно короткий промежуток времени. Исключением является случай, когда в биты DSPINT или HINT регистра HPIC записывается единица. В этом случае доступ от host-а занимает несколько периодов тактовых импульсов CPU и временные соотношения в этом случае те же, что и в режиме SAM. В целом, исключая HRDY, временные соотношения при обращениях к HPI в режиме HOM не отличаются от временных соотношений в режиме SAM. Условия, при которых сигнал HRDY активен (когда используется временные соотношения режима SAM) при обращениях host-а, представлены в таблице 44. Когда HRDY неактивен (HRDY остается в состоянии с высоким уровнем), используются временные соотношения режима HOM.

Таблица 44 – Условия формирования состояний ожидания

Регистр	Состояния ожидания	
	Для чтения	Для записи
HPIC	Нет	1 в DSPINT/HINT – Есть, Все остальные циклы – Нет
HPIA	Нет	HOM – Нет, SAM – Есть
HPID	HOM – Нет, SAM – Есть	HOM – Нет, SAM – Есть

6.5.3.1 Последовательность доступа

Полный цикл доступа host-а включает в себя два байта, первый – при низком уровне HBIL, второй – при высоком. Эта 2-байтная последовательность соблюдается независимо от типа обращения host-а (HPIA, HPIC или HPID), и host не должен нарушать ее в течение HPI доступа во избежание потери данных.

Перед выборкой данных host должен сначала инициализировать HPIC, в частности бит BOB, а затем HPIA (именно в таком порядке, поскольку BOB влияет на доступ к HPIA). При доступе к памяти HPI по чтению, после завершения записи в HPIA необходимого адреса, производится чтение памяти HPI по этому адресу и считанное слово передается в два 8-битных регистра данных. Таблица 45 иллюстрирует последовательность загрузки BOB и HPIA для чтения из памяти HPI. В этом примере BOB сбрасывается в 0, запрашивается доступ по чтению к первой ячейке памяти HPI (в нашем случае по адресу 1000h), которая содержит число FFFEh.

В процессе обмена, представленном в таблице 45, сначала инициализируется BOB и HPIA, затем в HPIA загружается адрес ячейки внутренней памяти HPI. Последняя строка таблицы 45 показывает состояние порта HPI по завершении чтения из внутренней RAM, т. е. спустя некоторое время после записи второго байта адреса в регистр HPIA. По завершении чтения данные размещаются в регистрах старшего и младшего байтов. Для извлечения этих данных host должен выполнить дополнительное чтение HPID. Во время доступа к HPID для чтения содержимое регистра первого байта данных появляется на выводах HD, когда HBIL переходит на низкий уровень, а содержимое регистра второго байта данных появляется на выводах HD, когда HBIL перейдет в единицу. Затем адрес увеличивается, если выбран режим с автоинкрементом адреса, и данные из памяти вновь передаются в регистры данных. Последовательность, используемая при таком доступе, показана в таблице 46.

Таблица 45 – Инициализация BOB и HPIA

Событие	HD	HR/W	HCNTL0/1	HBIL	HPIC	HPIA	Регистр 1	Регистр 2
Запись host-a в HPIC, 1-й байт	00	0	00	0	00xx	xxxx	xxxx	xxxx
Запись host-a в HPIC, 2-й байт	00	0	00	1	0000	xxxx	xxxx	xxxx
Запись host-a в HPIA, 1-й байт	10	0	10	0	0000	10xx	xxxx	xxxx
Запись host-a в HPIA, 2-й байт	00	0	10	1		1000	xxxx	xxxx
Завершение внутреннего чтения HPI RAM						1000	FF	FE

Таблица 46 – Доступ по чтению к HPI с автоинкрементом адреса

Событие	HD	HR/W	HCNTL0/1	HBIL	HPIC	HPIA	Регистр 1	Регистр 2
Чтение данных, 1-й байт	FF	1	01	0	0000	1000	FF	FE
Чтение данных, 2-й байт	FE	1	01	1	0000	1000	FF	FE
Завершение внутреннего чтения HPI RAM						1001	6A	BC

При выборе режима автоинкремента приращение адреса происходит после передачи всех 16-бит слова, а не байта. При этом HPIA увеличивается на единицу. Последняя строка таблицы 46 показывает, что после завершения второго чтения из внутреннего RAM, содержимое ячейки 1001h (6ABCh) считывается и размещается в регистрах старшего и младшего байтов данных.

В процессе доступа к HPI по записи регистр первого байта загружается данными из host-a, когда вывод HBIL находится в состоянии с низким уровнем, а регистр второго байта – когда вывод HBIL находится в состоянии с высоким уровнем. По завершении доступа для записи данные из двух регистров передаются как 16-разрядное слово в память HPI по адресу, определяемому регистром HPIA. Если выбран режим с автоматическим приращением адреса, модификация его происходит до записи данных в память.

Доступ к HPI по записи иллюстрируется в таблице 47. В этом примере, после завершения внутренней записи, ячейка 1002h HPI RAM содержит число 1234h. Если непосредственно за операцией записи следует операция чтения по тому же самому адресу, то данные читаются непосредственно из регистров данных (1234h).

Таблица 47 – Доступ по записи в HPI с автоинкрементом адреса

Событие	HD	HR/W	HCNTL0/1	HBIL	HPIC	HPIA	Регистр 1	Регистр 2
Запись данных, 1-й байт	12	0	01	0	0000	1002	12	FE
Запись данных, 2-й байт	34	0	01	1	0000	1002	12	34
Запись во внутрен-нюю память HPI RAM завершена						1002	12	34

6.5.4 Функциональная операция DSPINT и HINT

Host-устройство и процессор могут прерывать друг друга, используя соответствующие биты в регистре HPIC. Ниже подробно описывается этот процесс.

6.5.4.1 Host-устройство использует бит DSPINT для прерывания процессора

Прерывание генерируется, когда host записывает 1 в бит DSPINT регистра HPIC. Это прерывание может использоваться для вывода процессора из состояния ожидания IDLE. Host и процессор всегда читают этот бит как 0. Запись из процессора в этот бит не возможна. Host не должен устанавливать бит DSPINT во время модификации бит BOB или HINT, т. к. это может привести к нежелательному прерыванию процессора.

Адрес вектора прерывания host-процессор – xx64h, а флаг и маска прерывания размещены в бите 9 регистров IFR и IMR соответственно. Так как таблица векторов прерываний процессора может быть размещена в памяти HPI, host может управлять обработкой прерываний, предварительно записав адрес подпрограммы обработки прерываний по адресу xx65h в памяти HPI, если в ячейке, расположенной по адресу xx64h, запрограммирована команда перехода. Если прерывания перераспределены в HPI RAM, необходимо использовать режим SAM и host не должен производить запись в ячейки с адресами xx00h-xx7Fh, за исключением xx65h.

6.5.4.2 Интерфейс host-порта процессора при использовании бита HINT для прерывания host-устройства

Когда процессор записывает 1 в бит HINT регистра HPIC, внешний выход HINT# процессора переходит в состояние с низким уровнем. Бит HINT читается как 1 и для процессора, и для host-а. Сигнал HINT# может использоваться для прерывания host-устройства. Host-устройство после обнаружения прерывания HINT может подтвердить и очистить его и бит HINT, записав 1 в бит HINT. Бит HINT сбросится и будет читаться как 0 и процессором, и host-ом, при этом вывод HINT# перейдет в состояние с высоким уровнем. Запись нуля в бит HINT как со стороны процессора, так и со стороны host-а невозможна. Процессор не должен одновременно модифицировать биты SMOD и HINT, если только не предстоит прерывание host-а.

6.5.5 Соглашения в изменении режима доступа к памяти host-порта (SMOD/HOM) и использование режимов IDLE2/3

Режим (HOM) обеспечивает host-у доступ к HPI RAM, когда процессор находится в состоянии IDLE2/3 (полностью остановлен). Кроме того, может быть остановлен вход внешних тактовых импульсов для процессора с целью уменьшения потребляемой мощности. Внешние тактовые синхроимпульсы необходимы только перед выводом процессора из состояния IDLE2/3.

Host не имеет доступа к HPI RAM в режиме SAM, когда процессор находится в режиме IDLE2/3, поскольку для доступа в этом режиме требуются тактовые импульсы CPU. Следовательно, если host-у требуется доступ к HPI RAM в то время как процессор находится в состоянии IDLE2/3, необходимо перед переходом в режим IDLE2/3 изменить режим доступа к HPI на HOM. В режиме HOM CPU имеет возможность модифицировать регистр HPIC с целью изменения состояния бита SMOD или для генерации прерывания host-у, но не имеет доступа к блоку RAM HPI. Для того чтобы процессор вновь получил доступ к RAM, следует изменить режим HPI на SAM после выхода из состояния IDLE2/3.

Для выбора режима HOM следует записать 0 в бит SMOD регистра HPIC. Для выбора режима SAM следует записать 1 в SMOD. При переходе из одного режима в другой следует учитывать две особенности:

– не следует использовать команды IDLE 2 или IDLE 3 непосредственно за командой перехода из режима SAM в режим HOM. Это объясняется тем, что вследствие работы конвейера процессора и задержек при переключении из режима SAM в HOM, команда IDLE2/3 выполняется раньше, чем происходит переключение режимов, в результате чего HPI останется в режиме SAM и host не сможет обмениваться с HPI;

– не следует использовать команды чтения из блока RAM HPI непосредственно за командой перехода из режима HOM в SAM, т. к. вследствие задержки переключения режимов команда чтения будет игнорироваться. На операции записи в RAM эти ограничения не распространяются, т. к. конвейер задерживает их исполнение.

Для host-а нет специальных ограничений, связанных с изменением режима. Например, третье устройство может вывести процессор из состояния IDLE2/3 и процессор после этого переходит в режим SAM без программного подтверждения со стороны host-а. Host может продолжать доступ в то время, когда происходит изменение режима. Однако, если host обращается к HPI RAM в то время, когда изменяется режим, фактическое изменение режима задержится до завершения доступа host-ом. В этом случае доступ CPU к HPI памяти также задерживается.

Таблица 48 иллюстрирует последовательность событий при входе и выходе процессора из состояния IDLE2/3 при использовании HPI. В это время HPI доступен host-устройству.

Таблица 48 – Последовательность входа и выхода процессора из состояния IDLE2 и IDLE3

Host или другое устройство	CPU	Режим	Частота CPU
	Переключение в режим HOM	HOM	Используется
	Выполняет команду NOP	HOM	Используется
	Выполняет команду IDLE2 или IDLE3	HOM	Используется
Может остановить тактовые импульсы DSP	В состоянии IDLE2/3	HOM	Остановлена или используется
Запускает тактовые импульсы DSP, если они были остановлены [1)]	В состоянии IDLE2/3	HOM	Используется
Посылает требование прерывания на DSP	В состоянии IDLE2/3	HOM	Используется
	CPU выходит из состояния IDLE2/3	HOM	Используется
	CPU переключается в режим SAM	SAM	Используется

[1)] При использовании внутрикристального PLL должно гарантироваться необходимое время запуска.

6.5.6 Доступ к памяти host-интерфейса (HPI RAM) во время сброса

Процессор при сбросе не функционирует, но host может обмениваться с HPI, что позволяет загружать программы или данные в HPI память. Использование этой возможности позволяет host-у управлять начальной инициализацией процессора. Последовательность событий при сбросе процессора и загрузке программы в HPI память, когда процессор находится в сбросе, показана в таблице 49.

Сначала host прекращает доступ к HPI и выжидает, по крайней мере, шесть периодов тактовой частоты процессора. Затем host переводит вывод сброса процессора в состояние с низким уровнем и после четырех тактов синхросигнала CPU начинает обмен с HPI. Во время сброса режим автоматически устанавливается на HOM, что обеспечивает высокоскоростную загрузку программы. Тактовые сигналы CPU в это время могут отсутствовать.

По окончании загрузки в HPI память, host прекращает обмен с HPI и переводит линию сброса процессора в состояние с высоким уровнем. Host может возобновить обмен с HPI не ранее, чем через 20 периодов тактовой частоты CPU после сброса. По окончании режима сброса происходит автоматическое переключение HPI в режим SAM.

Если host затребует прерывание в то время, как процессор находится в сбросе, оно будет потеряно. Таким образом, HPI память может быть использована для “теплой” загрузки процессора, и программа в этом случае начнет выполняться с самого младшего адреса памяти HPI.

Таблица 49 – Функционирование HPI в течение сброса

Host	CPU	Режим	Такты CPU
Ожидает 6 периодов частоты CPU	Запущен	X	Запущены
Переводит провод RESET в состояние с низким уровнем и ждет 4 такта	Переходит в режим сброса	HOM	Запущены
Может остановить тактовый сигнал CPU	Сброс	HOM	Остановлены или запущены
Пересылает программу и\или данные в HPI память	Сброс	HOM	Остановлены или запущены
Запускает тактовую частоту DSP, если она была остановлена [1)]	Сброс	HOM	Запущены
Переводит RESET в состояние с высоким уровнем	Сброс	HOM	Запущены
Ожидает 8 периодов тактовой частоты CPU	Выход из сброса	SAM	Запущены
Может иметь доступ к HPI	Запущен	SAM	Запущены

[1)] При использовании внутрикристального PLL должно гарантироваться необходимое время запуска.

6.6 Режим стандартного последовательного порта (SP) процессора 1867ВЦ4Т

Последовательный порт обеспечивает прямую двунаправленную связь с устройствами, такими как кодеки, последовательные A/D конвертеры и другими последовательными системами. Сигналы интерфейса последовательного порта совместимы со многими стандартными кодеками и другими последовательными устройствами. Последовательный порт может также использоваться для микропроцессорных соединений между процессорами в мультипроцессорных системах. Операции приема и передачи осуществляются с двойным буферированием, позволяющим осуществлять непрерывный обмен информацией в байтовом или словном формате. Непрерывный режим не требует наличия кадровых синхроимпульсов при обмене с максимальной частотой пакетов.

В процессоре 1867ВЦ4Т предусмотрено 2 типа последовательного порта:

– буферизированный последовательный порт (BSP);

– мультиплексированный последовательный порт с разделением по времени (TDM).

Оба типа последовательных портов, кроме своих основных режимом – режим автобуферизации для BSP и мультиплексирование по времени для TDM порта, для передачи данных могут использовать режим стандартного последовательного порта. В этом режиме TDM порт поддерживает формат 8- и 16-разрядных данных, а BSP – формат 8-, 10-, 12- и 16-разрядных данных. Перевод в режим стандартного последовательного порта осуществляется установкой нулевого бита управляющих регистров TSPCB и BSPC в 0.

Для облегчения понимания работы BSP и TDM портов в режиме стандартного последовательного порта необходимо внести некоторые обобщения, которые представлены в таблице 50.

Таблица 50 – Регистры BSP и TDM портов

Регистр			Описание
Стандартный SP	TDM	BSP	
DDR	TRCV [1)]	BDDR [1)]	Регистр приема данных
DXR	TDXR [1)]	BDXR [1)]	Регистр передачи данных
SPC	TSPC [1)]	BSPC [1)], BSPCE(9:0) [1)]	Управляющий регистр
RSR	TRSR	BRSR	Сдвиговый регистр приема данных
XSR	TXSR	BXSR	Сдвиговый регистр передачи данных
_____ [1)] MMR-регистры.			

Например: DDR – это регистр приема данных в BSP или TDM порта и т. д.

6.6.1 Операции интерфейса последовательного порта

В таблице 51 представлены выводы, используемые для операций последовательного порта. На рисунке 54 показано соединение последовательных портов двух процессоров 1867ВЦ4Т для передачи данных от устройства 0 к устройству 1.

Таблица 51 – Выводы последовательного порта

Выводы	Описание
CLKX	Тактовая частота передачи данных
CLKR	Тактовая частота приема данных
DX	Сигнал последовательной передачи данных
DR	Сигнал последовательного приема данных
FSX	Сигнал синхронизации фрейма передачи данных
FSR	Сигнал синхронизации фрейма приема данных

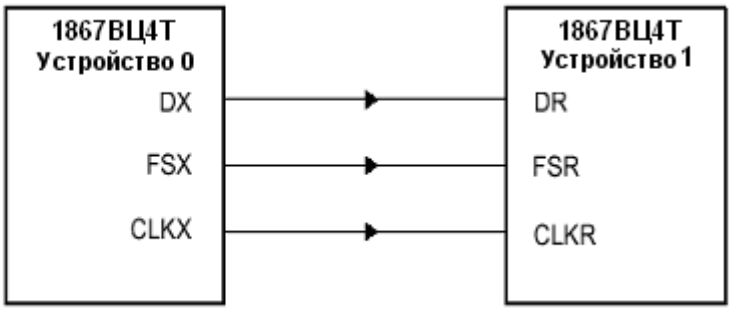


Рисунок 54 – Передача последовательных данных между двумя устройствами 1867ВЦ4Т

На рисунке 55 показана функциональная схема последовательного порта.

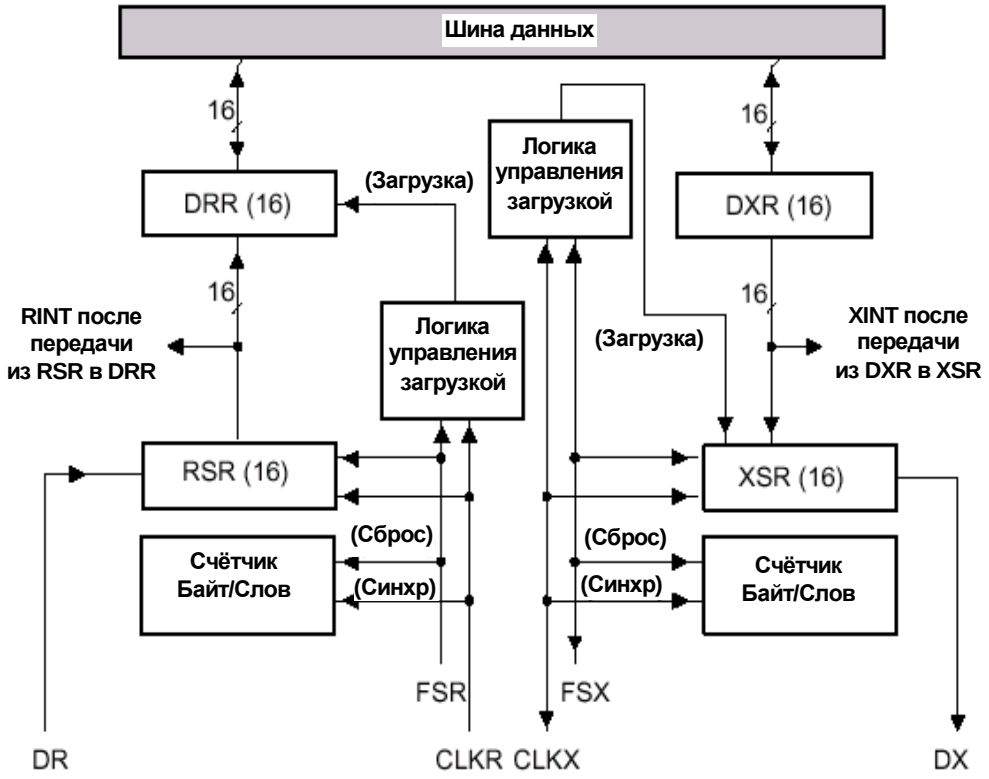


Рисунок 55 – Функциональная схема последовательного порта

Данные для передачи записываются в DXR, а принятые данные читаются из DRR. Начало передачи обозначается записью данных в DXR, данные копируются в XSR, когда он пуст (когда последнее слово последовательно передалось через вывод DX). XSR управляет сдвигом данных на DX, таким образом разрешая следующую запись в DXR, как только завершится копирование из DXR в XSR.

В процессе передачи, по завершению копирования из DXR в XSR, значение бита готовности передачи (XRDY) в SPC меняется из 0 в 1. Как только происходит это изменение, вырабатывается прерывание передачи последовательного порта (xXINT), которое сигнализирует процессору о том, что DXR готов снова загрузиться.

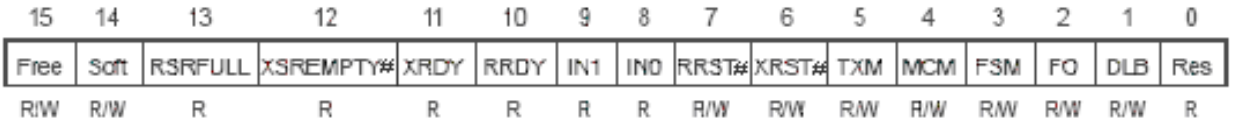
Также происходит и с принимающим процессором. Данные с вывода DR сдвигаются в RSR, затем копируются в DRR, из которого могут быть считаны. По завершении копирования RSR в DRR значение бита готовности приема (RRDY в SPC) меняется из 0 в 1. Как только происходит это изменение, вырабатывается прерывание по приему последовательного порта (xRINT). Порт обладает двойной буферизацией, т. к. он

может передавать или принимать из или в DXR или DRR в то время, как уже происходит передача или прием. Передача и прием данных синхронизируется при помощи FSX и FSR соответственно в пакетном режиме.

6.6.2 Конфигурация интерфейса стандартного последовательного порта

Регистр SPC содержит биты управления последовательным портом. На рисунке 56 показано распределение бит управляющего регистра. Некоторые биты этого регистра доступны только для чтения, другие – для чтения и записи.

В таблице 52 представлено описание битов регистра управления последовательным портом. В таблице 53 представлена конфигурация тактирования последовательного порта.



Обозначения: R – доступен только для чтения; R/W – доступен для чтения/записи

Рисунок 56 – Распределение бит в регистре управления последовательным портом (SPC)

Таблица 52 – Описание битов регистра управления последовательным портом

Бит	Имя	Значение при сбросе	Описание функционирования
15	FREE	0	Используется совместно с битом SOFT для определения состояния тактирования последовательного порта в точке останова программы при отладке. FREE = 0 – бит SOFT выбирает режим эмуляции. FREE = 1 – тактирование продолжает работать независимо от бита SOFT
14	SOFT	0	Используется совместно с битом FREE для определения состояния тактирования последовательного порта в точке останова программы при отладке. Когда бит FREE очищен, то SOFT выбирает режим эмуляции SOFT = 0 – тактирование останавливается, прерывая всякую передачу. SOFT = 1 – тактирование прекращается по завершении текущей передачи

Продолжение таблицы 52

Бит	Имя	Значение при сбросе	Описание функционирования
13	RSRFULL	0	Бит заполнения приемного сдвигового регистра. Используется, когда происходит переполнение приемного сдвигового регистра RSR. Переполнение происходит, когда RSR полный, а DRR не был считан после последней загрузки из RSR. В последовательном порту при FSM = 1, RSRFULL может стать 1, только если сформировался импульс FSR. Если FSM = 0, RSRFULL не ждет импульса FSR, а становится 1, когда происходит переполнение. RSRFULL = 0 – любое из следующих трех событий очищает RSRFULL до 0: – чтение DRR; – сброс приемника (= 0); – сброс всего устройства
12	XRESEMPTY#	0	Бит, показывающий, что передающий сдвиговый регистр пуст. Показывает, что не произошла своевременная загрузка XSR. Это происходит, когда XSR пуст и DXR не был загружен после последнего чтения из DXR в XSR. XRESEMPTY# = 0 – любое из следующих трех событий очищает XRESEMPTY# в 0: не произошла своевременная загрузка; сброс передатчика (XRST# = 0); сброс всего устройства. XRESEMPTY# = 1 – в последовательном порту XRESEMPTY# деактивируется (XRESEMPTY# = 1) сразу после записи в DXR
11	XRDY	1	Бит готовности передачи. Переход из 0 в 1 бита XRDY показывает, что содержимое DXR было скопировано в XSR и DXR готов к новой загрузке. Прерывание по передаче вырабатывается после этого перехода. При сбросе процессора или при сбросе передатчика последовательного порта (XRST# = 0), XRDY становится в 1
10	RRDY	0	Бит готовности приема данных. Переход из 0 в 1 бита RRDY показывает, что содержимое RSR было скопировано в DRR и данные могут считываться из него. Прерывание по приему вырабатывается после этого перехода. Этот бит может быть опрошен программой, не зависимо от использования прерываний последовательного порта. При сбросе процессора или при сбросе приемника последовательного порта (RRST# = 0), XRDY становится в 0

Продолжение таблицы 52

Бит	Имя	Значение при сбросе	Описание функционирования
9	IN1	X	Вход 1. Позволяет использовать внешний вывод CLKX как битный вход. IN1 отражает текущий уровень вывода CLKX. При переключении уровней CLKX существует задержка 0,5-1,5 цикла CLKOUT1, пока новое значение CLKX будет представлено в SPC
8	IN0	X	Вход 0. Позволяет использовать внешний вывод CLKR как битный вход. IN0 отражает текущий уровень вывода CLKR. При переключении уровней CLKX существует задержка 0,5-1,5 цикла CLKOUT1, пока новое значение CLKR будет представлено в SPC
7	RRST#	0	Бит сброса приемника. Разрешает и прекращает работу приемника последовательного порта. RRST# = 0 – сброс приемного последовательного порта. Запись 0 в RRST# очищает биты RSRFULL и RRDY в 0. RRST# = 1 – приемный последовательный порт задействован
6	XRST#	0	Бит сброса передатчика. Разрешает и прекращает работу передатчика последовательного порта. XRST# = 0 – сброс передающего последовательного порта. Если бит XRDY был 0, запись 0 в XRST# вырабатывает прерывание по передаче. Запись 0 в XRST# очищает бит XRESEMPTY# в 0 и устанавливает бит XRDY в 1. XRST# = 1 – передающей последовательный порт задействован
5	TXM	0	Бит режима передачи. Конфигурирует вывод FSX как вход (TXM = 0) или как выход (TXM = 1). TXM = 0 – внешняя синхронизация фрейма. Передающий порт ждет внешнего импульса FSX TXM = 1 – внутренняя синхронизация фрейма. FSX вырабатывается внутренне, когда данные передаются из DRX в XSR для обозначения передачи данных. Импульсы FSX синхронизированы с CLKX
4	MCM	0	Бит режима тактирования. Определяет источник тактов для CLKX. MCM = 0 – CLKX берется с внешнего вывода CLKX; MCM = 1 – CLKX вырабатывается внутри процессора с частотой, равной ¼ частоты CLKOUT1. Если MCM = 1 и DLB = 1, CLKR тоже вырабатывается внутри процессора

Окончание таблицы 52

Бит	Имя	Значение при сбросе	Описание функционирования
3	FSM	0	Бит режима синхронизации фреймов. Определяет, нужны ли импульсы FSX, FSR после первого импульса инициализации. FSM = 0 – непрерывный режим. Синхроимпульсы не нужны за исключением первого, но они не игнорируются, поэтому не во время поданные импульсы могут стать причиной ошибок в последовательной передаче/приеме данных; FSM = 1 – прерывный режим. Для передачи/приема каждого слова требуется импульс FSX/FSR
2	FO	0	Бит формата данных. Определяет длину передаваемых/принимаемых слов. FO = 0 – данные передаются/принимаются как 16-разрядные слова; FO = 1 – данные передаются по 8-разрядов. Первыми передаются старшие разряды (MSB)
1	DLB	0	Бит режима внутренней петли. Используется для установки работы порта в режиме внутренней петли. DLB = 0 – режим внутренней петли отключен. DR, FSR, CLKR берутся с соответствующих внешних выводов; DLB = 1 – задействован режим внутренней петли. DR и FSR подаются на DX и FSX соответственно. CLKR управляется CLKX, если MCM = 1. Если DLB = 1 и MCM = 0, сигнал CLKR берется с внешнего вывода CLKR. Эта конфигурация позволяет CLKR и CLKX тактироваться внешним источником. В режиме DLB сигналы FSX и DX появляются на внешних выводах, а FSR и DR нет. В режиме DLB могут использоваться как внутренние, так и внешние сигналы FSX, как определено битом TXM (рисунок 57)
0	RES	0	Установка в 0 разрешает использования режима стандартного порта

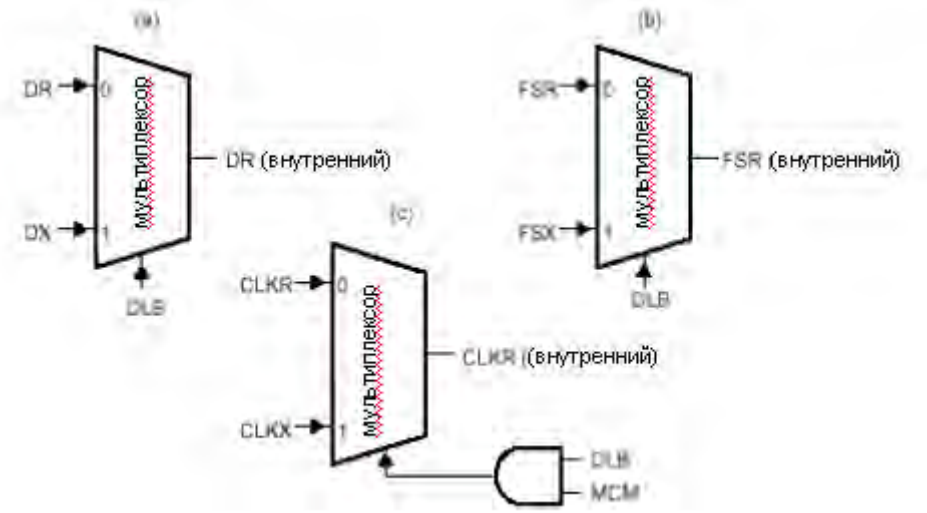


Рисунок 57 – Мультиплексоры сигналов приемника

Таблица 53 – Конфигурация тактирования последовательного порта

FREE	SOFT	Конфигурация тактирования последовательного порта.
0	0	Немедленная остановка тактирования
0	1	Передающий порт останавливается после завершения текущего слова. На приемный порт не влияет
1	X	Последовательный порт продолжает работать

6.6.3 Операции приема и передачи в пакетном режиме

При работе последовательного порта в пакетном режиме между кадрами данных возможны паузы. Каждый кадр сопровождается импульсом кадровой синхронизации. Передача из порта инициируется записью в регистр DXR. Содержимое регистра DXR передается в регистр XSR при появлении импульса кадровой синхронизации на выводе FSX (внутреннего или внешнего в зависимости от состояния бита TXM). Содержимое XSR выводится на вывод DX без сохранения выводимой информации. Если регистр DXR загружается до копирования старого содержимого в XSR, старое содержимое теряется. Регистр DXR копируется в XSR только в том случае, если XSR пустой, а DXR обновился после последнего копирования. Запись в регистр DXR может осуществляться лишь при условии XRDY = 1, что гарантируется в случае наличия запроса на прерывание от передачи через последовательный порт или вследствие программного опроса бита XRDY. Временная диаграмма передачи через последовательный порт представлена на рисунке 58.

Следует учитывать различия в работе порта при внутренней (TXM = 1, FSX является выходом) и внешней (TXM = 0, FSX является входом) кадровой синхронизации. Эти различия обусловлены в первом случае тем, что импульс кадровой синхронизации генерируется самим портом и является прямым результатом записи в регистр DXR. Во втором случае такое прямое влияние отсутствует, т. е. программа должна осуществлять запись в регистр DXR, после чего порт ждет внешний импульс кадровой синхронизации.

При использовании внутреннего генератора (TXM = 1) кадровый синхроимпульс генерируется сразу после записи данных в регистр DXR на нарастающем фронте сигнала CLKX. При использовании внешнего импульса запуск последовательного порта на передачу происходит после появления импульса кадровой синхронизации (при условии предварительного обновления содержимого регистра DXR) на нарастающем фронте сигнала CLKX. Далее, на следующем падающем фронте сигнала CLKX регистр XSR

загружается значением из DXR, а бит XRDY переходит в единицу, генерируя прерывание от передачи XINT. При последующем нарастающем фронте сигнала CLKX первый бит данных (старший бит) выводится на вывод DX. Остальные биты данных выдвигаются из регистра XSR после спада импульса кадровой синхронизации (сдвиг в регистре XSR не циклический, что приводит к потере данных в нем). Так как импульс кадровой синхронизации не синхронен с тактами CLKX, длительности первого и последующих бит могут различаться. В случае внутреннего кадрового генератора этот эффект отсутствует.

После передачи всех бит элемента данных вывод DX переходит в состояние с высоким импедансом. Следует отметить, что если регистр DXR не был загружен по прерыванию XINT, флаг XRESEMPTY# активизируется (перейдет в состояние с низким уровнем состояние), указывая на отсутствие данных для передачи. Таким образом, в случае внутренней генерации кадровых синхроимпульсов (TXM = 1), данные на линии появляются примерно через два такта сигнала CLKX после загрузки регистра DXR. При использовании внешнего синхроимпульса эта задержка отсутствует, и требование к синхронизации становится менее жестким. В последнем случае, если флаг XRESEMPTY# активен в момент появления импульса кадровой синхронизации, передается содержимое DXR.

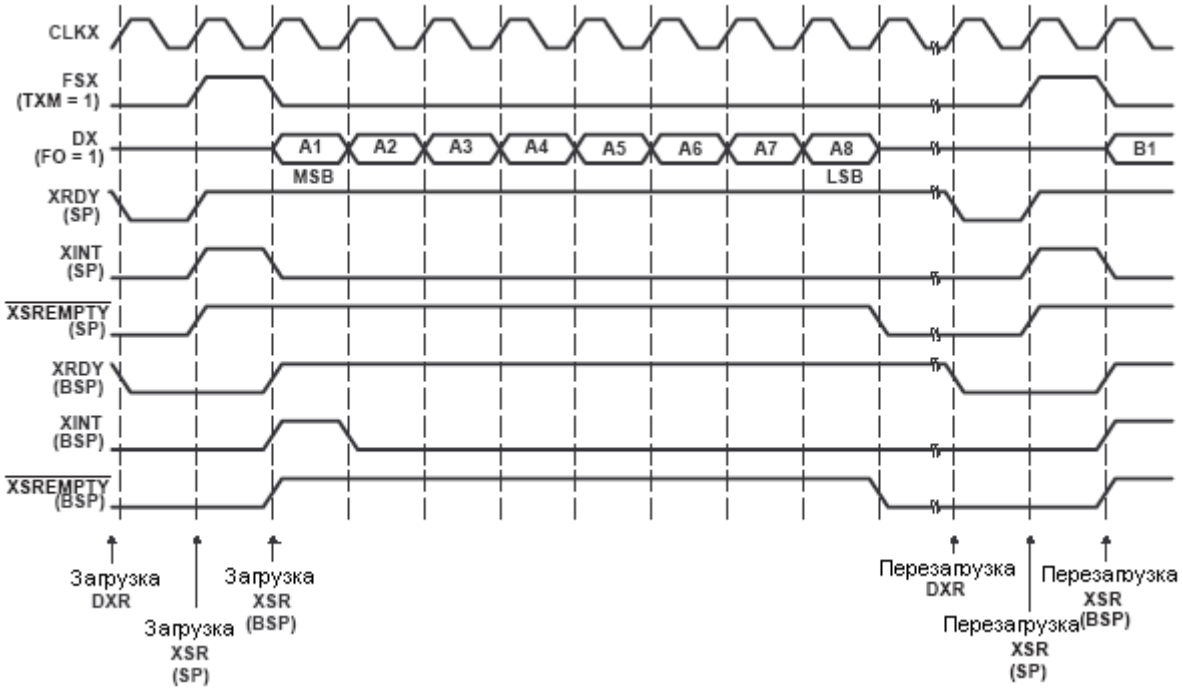


Рисунок 58 – Операция передачи из последовательного порта в пакетном режиме

Запись в RSR начинается на ниспадающем фронте сигнала CLKR после того, как импульс кадровой синхронизации перешел в состояние с низким уровнем. После приема всех бит содержимое RSR передается в DRR на ниспадающем фронте CLKR, а бит RRDY переходит в единицу, генерируя прерывание по приему RINT, как показано на рисунке 59. Следует принять во внимание, что если DRR после предыдущего приема не был считан и появится импульс кадровой синхронизации, флаг RSRFULL перейдет в состояние с высоким уровнем (рисунок 60). Эта ситуация фактически является ошибочной и требует соответствующей обработки. Ошибочной ситуацией является и появление импульса кадровой синхронизации во время текущего приема. Различные ошибки при приеме в последовательный порт обсуждаются ниже.

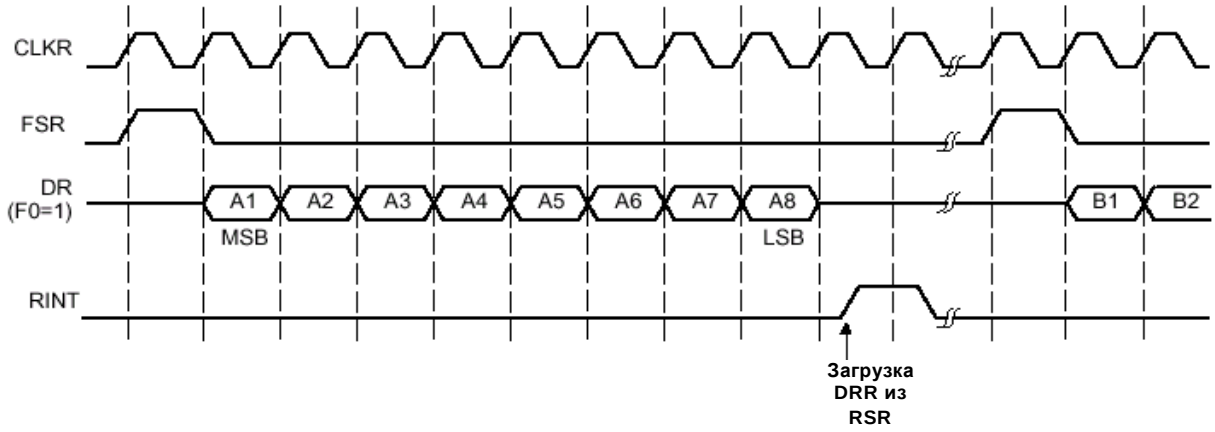


Рисунок 59 – Операция приема в последовательный порт в пакетном режиме

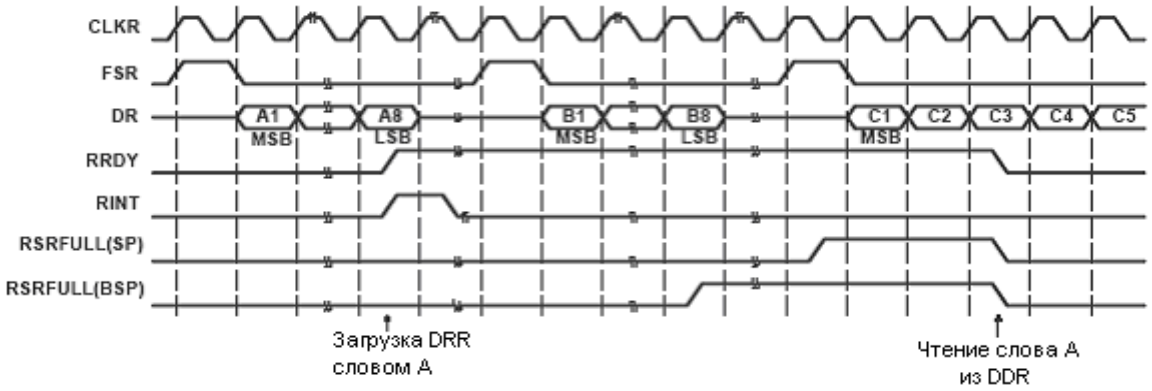


Рисунок 60 – Переполнение при приеме по последовательному порту в пакетном режиме

Следует заметить, что если кадровая частота возрастает, пауза между соседними сеансами передачи пакетов уменьшается в пределе до нуля. Это соответствует минимальному периоду между импульсами кадровой синхронизации (эквивалентному 8 или 16 тактам CLKX/CLKR в зависимости от F0), что соответствует максимальной кадровой частоте, при которой может работать последовательный порт. При максимальной кадровой частоте (см. рисунок 61) временная диаграмма является сжатой версией временной диаграммы на рисунке 58.

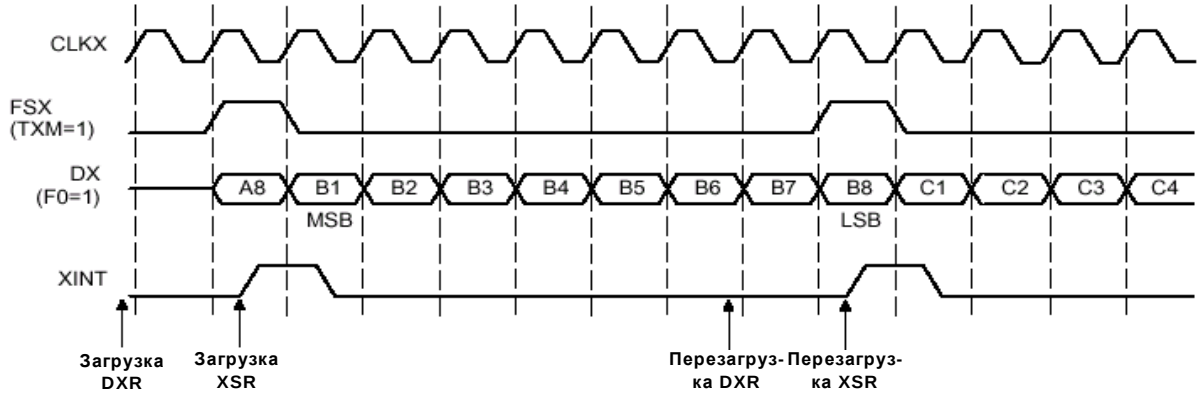


Рисунок 61 – Передача через последовательный порт в пакетном режиме с максимальной частотой кадров

Биты данных в последующих кадрах передаются непрерывно без пауз между ними. Импульс кадровой синхронизации накрывает последний бит, передаваемый в предыдущем кадре. На приемной стороне процесс выглядит аналогично (см. рисунок 62).

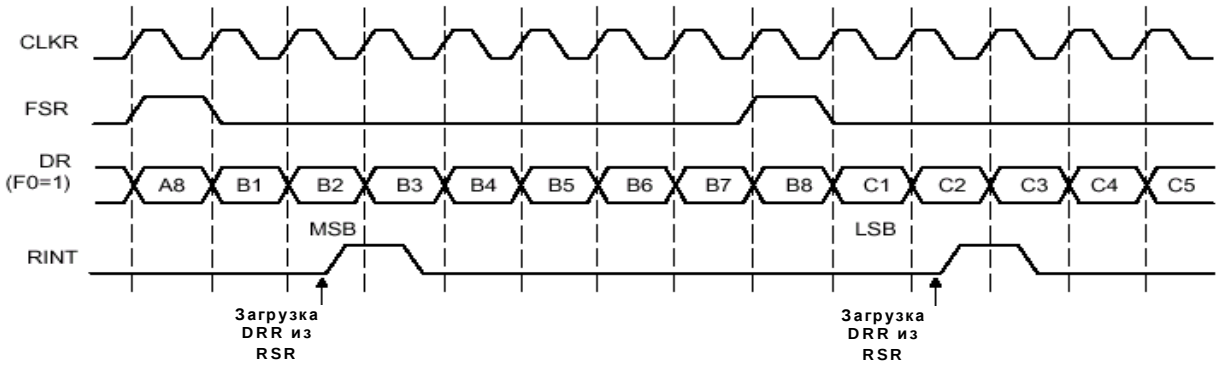


Рисунок 62 – Прием в последовательный порт в пакетном режиме с максимальной частотой кадров

На рисунках 63 и 64 представлены временные диаграммы передачи нескольких пакетов данных с максимальной кадровой частотой. Импульс кадровой синхронизации в этом случае не является необходимым. Так как пакеты данных передаются с постоянной скоростью без промежутков, сигнал тактовой синхронизации CLK обеспечивает синхронность данных и последовательного порта. Теоретически для инициализации непрерывной передачи данных необходим только начальный импульс кадровой синхронизации. Такой режим непрерывной передачи доступен последовательному порту и обсуждается ниже.

Работа последовательного порта с внешним импульсом кадровой синхронизации аналогична работе с внутренним кадровым синхроимпульсом. Передача запускается при появлении внешнего импульса кадровой синхронизации. Временная диаграмма сигналов управления последовательным портом с внешними импульсами кадровой синхронизации представлена на рисунках 63 и 64. Пусть в начальный момент времени регистр DXR загружен байтом A. На следующем такте сигнала CLKX содержимое DXR копируется в сдвиговый регистр XSR, после чего генерируется требование прерывания по передаче в последовательный порт XINT. Подпрограмма обработки этого прерывания должна поместить в регистр DXR байт B, предназначенный для следующей передачи. При появлении внешнего кадрового синхроимпульса байт A начинает выдвигаться на вывод DX. По завершении передачи байта A байт B копируется из регистра DXR в XSR и генерируется прерывание XINT. Передача байта B наружу начинается только после появления следующего импульса кадровой синхронизации, Следует отметить, что когда происходит загрузка байта B в регистр DXR, копирование регистра DXR в XSR, не происходит и прерывание XINT не генерируется, т. к. еще не закончилась передача байта A. Любая последующая запись в регистр DXR после появления импульса кадровой синхронизации обновляет содержимое регистра DXR.

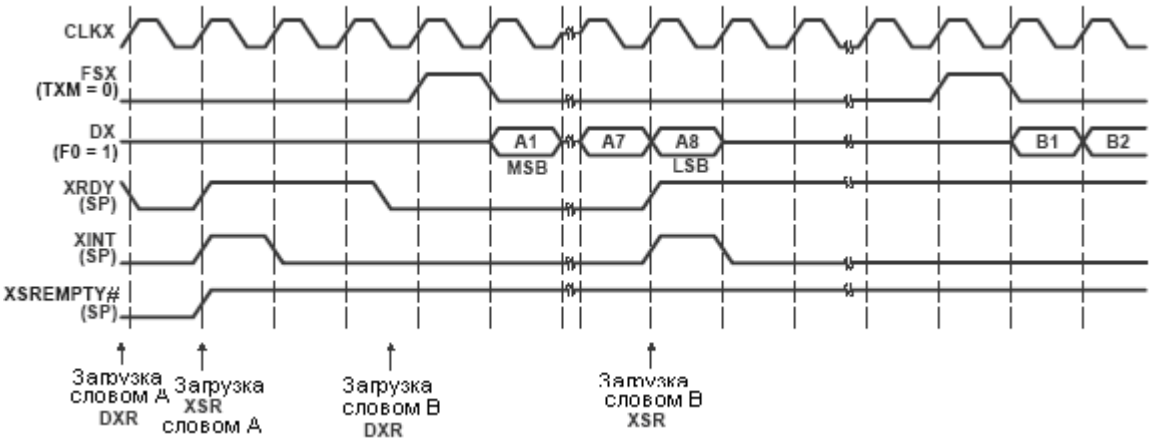


Рисунок 63 – Операция передачи из последовательного порта в пакетном режиме с внешней кадровой синхронизацией (SP)

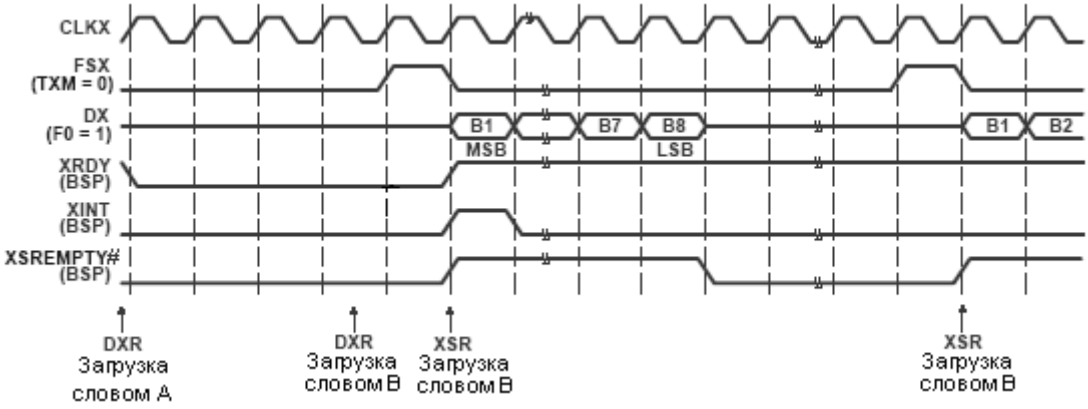


Рисунок 64 – Операция передачи из последовательного порта в пакетном режиме с внешней кадровой синхронизацией (BSP)

6.6.4 Операции приема и передачи в непрерывном режиме

В непрерывном режиме передачи пакетов с максимальной частотой кадровые синхроимпульсы FSX/FSR не нужны, за исключением первого импульса. Непрерывный режим выбирается установкой FSM = 0. При использовании внутреннего кадрового генератора кадровый синхроимпульс генерируется только после первой загрузки регистра DXR. Непрерывный режим длится, пока модифицируется регистр DXR для каждой передачи. Ошибочная запись в регистр DXR вызывает останов последовательного порта, как в случае пакетного режима (флаг XRESEMPTY# устанавливается в состояние 1 и т. д.). Если запись в DXR осуществляется после останова порта, процессор перезапускает передачу в непрерывном режиме и генерирует FSX. В этом проявляется различие между режимами с внутренней и внешней синхронизацией.

Если импульсы кадровой синхронизации генерируются внешней схемой (TXM = 0), DXR должен быть загружен до прихода внешнего синхроимпульса на вывод FSX, ибо именно он инициирует новую передачу в не непрерывном режиме. Если регистр DXR не был модифицирован синхронно с внешним импульсом кадровой синхронизации, вывод DX остается в состоянии высокого импеданса, что отличает этот режим от работы порта в пакетном режиме. Непрерывный режим можно выключить, т. е. перейти в пакетный режим, только при сбросе в исходное состояние последовательного порта или процессора. Изменение бита FSM во время передачи или останова не гарантирует перехода в пакетный режим. Тактирование передачи в непрерывном режиме показано на рисунке 65.

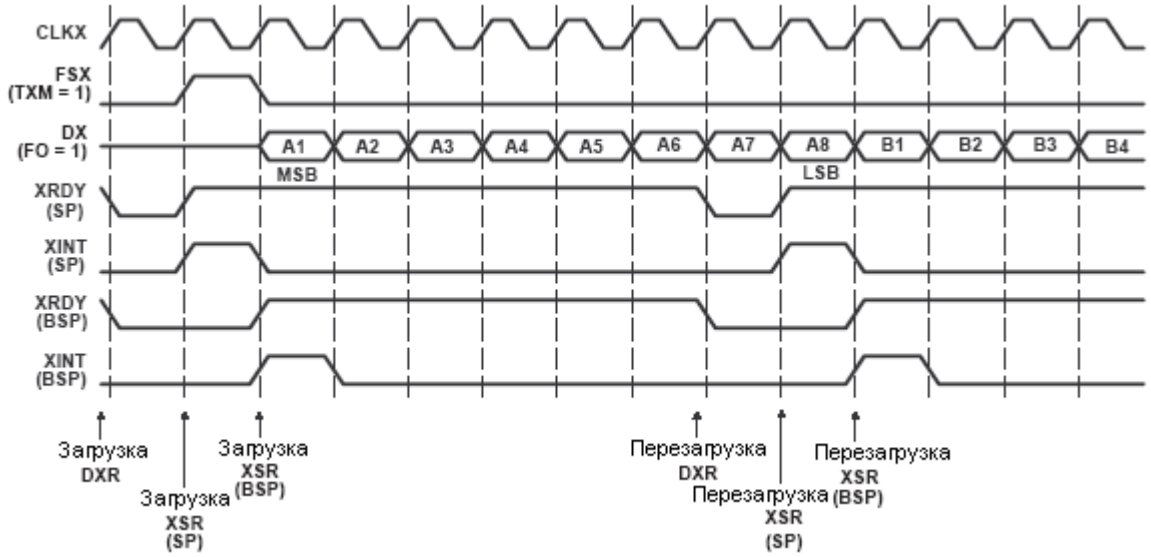


Рисунок 65 – Передача через последовательный порт в непрерывном режиме

Главное отличие от пакетного режима состоит в отсутствии импульсов кадровой синхронизации после первого синхроимпульса. До тех пор, пока DXR модифицируется перед каждой передачей, непрерывный режим будет продолжаться. Перезапись в DXR осуществляется так же, как и в пакетном режиме. Передаются всегда данные, записанные последними. Работа XSR не нарушается. Внешний импульс FSX на линии вызывает аварийное прекращение текущей передачи, потерю одного элемента данных и инициализацию новой передачи в непрерывном режиме. Более подробно это объясняется ниже.

Работа на приеме подобна работе на передаче. После начального импульса кадровой синхронизации на FSR дальнейшая синхронизация не нужна. Этот режим будет продолжаться до тех пор, пока DRR будет считываться подпрограммой обработки прерывания. Если DRR не считывается, прием в последовательный порт останавливается (флаг RSRFULL перейдет в состояние 1). Считывание DRR перезапустит непрерывный режим после прихода импульса кадровой синхронизации. Непрерывный режим можно выключить сбросом последовательного порта или процессора в исходное состояние. Тактирование на приеме показано на рисунке 66.

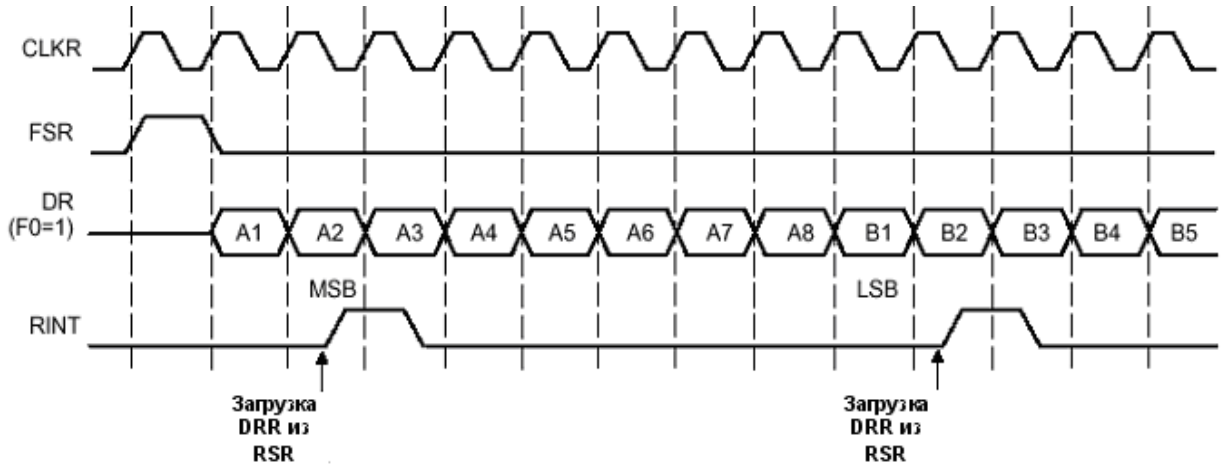


Рисунок 66 – Прием последовательного порта в непрерывном режиме

6.6.5 Ошибки в интерфейсе последовательного порта

Ошибочные ситуации при передаче данных через последовательный порт являются, как правило, результатом нестыковки программной и аппаратной составляющих процесса обмена и возникают из-за некорректного обращения к буферным регистрам DXR или DRR, или из-за несвоевременного появления импульса кадровой синхронизации. Необходимо понять, как последовательный порт обрабатывает эти ошибки, и в каком состоянии он окажется в результате их возникновения. Поскольку ошибочные ситуации в пакетном режиме отличаются от таковых в режиме непрерывном, они рассматриваются отдельно.

Начнем с рассмотрения влияния флага RSRFULL в пакетном режиме. Этот флаг появляется, когда процессор не успевает обработать все кадры, поступающие от внешнего устройства. В этом случае последовательный порт приостанавливает прием до тех пор, пока процессор не считает регистр DRR. При этом все данные, поступающие в регистр RSR, теряются. Если при появлении следующего кадрового синхроимпульса ошибочная ситуация повторяется (т. е. данные продвигаются в регистр RSR с вывода DR), текущий прием прекращается и начинается новый прием. Таким образом, данные, загружаемые в регистр RSR, теряются, но данные в DRR сохраняются, т. е. копирования из RSR в DRR не происходит. Избежать потери данных можно путем программного опроса бита RSRFULL, и, если он установлен, чтения регистра DRR. На рисунке 67 представлена блок-схема алгоритма обработки такой ошибки в последовательном порте.

Обработка ошибок передачи в пакетном режиме зависит от момента появления импульса тактовой синхронизации. Ситуация, при которой программе нечего передавать и потому содержимое регистра DXR не обновляется, не рассматривается последовательным портом как ошибка. Если в процессе передачи (т. е. данные XSR выдвигаются в штырек DX) появляется импульс кадровой синхронизации, то происходит аварийное прекращение текущей передачи, и данные в регистре XSR теряются. Затем данные, находящиеся в регистре DXR, пересылаются в XSR (копирование DXR в XSR) и передаются во внешнее устройство. Требование прерывания по передаче вырабатываются только в том случае, если обновление регистра DXR произошло после предыдущего кадрового синхроимпульса. Это может служить причиной зависания программы. Если в момент появления импульса кадровой синхронизации бит XRESEMPTY# активен, данные для передачи берутся старые. На рисунке 68 приведена блок-схема алгоритма обработки последовательным портом ошибки на передаче.

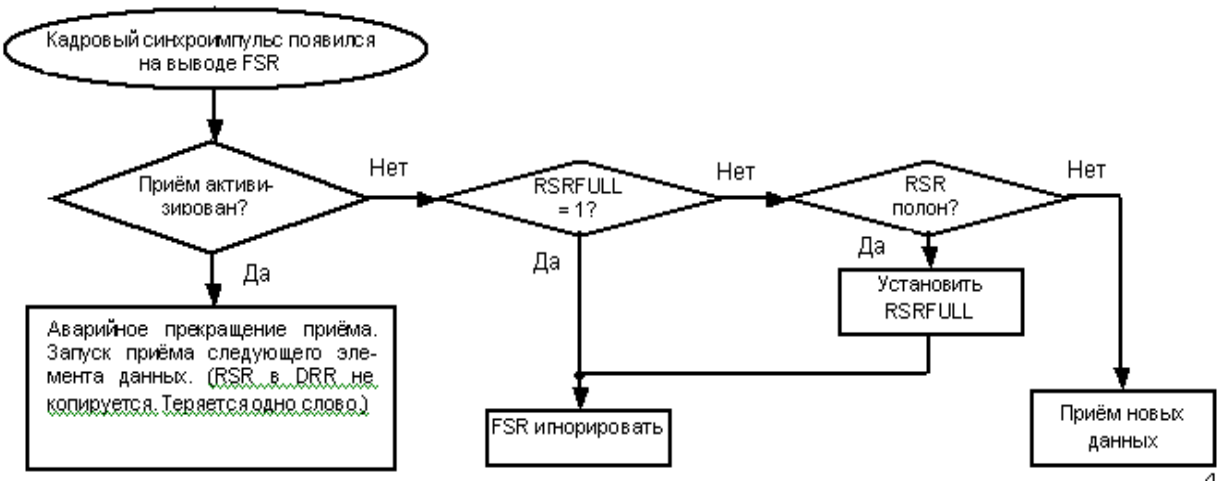


Рисунок 67 – Ошибки приема в пакетном режиме

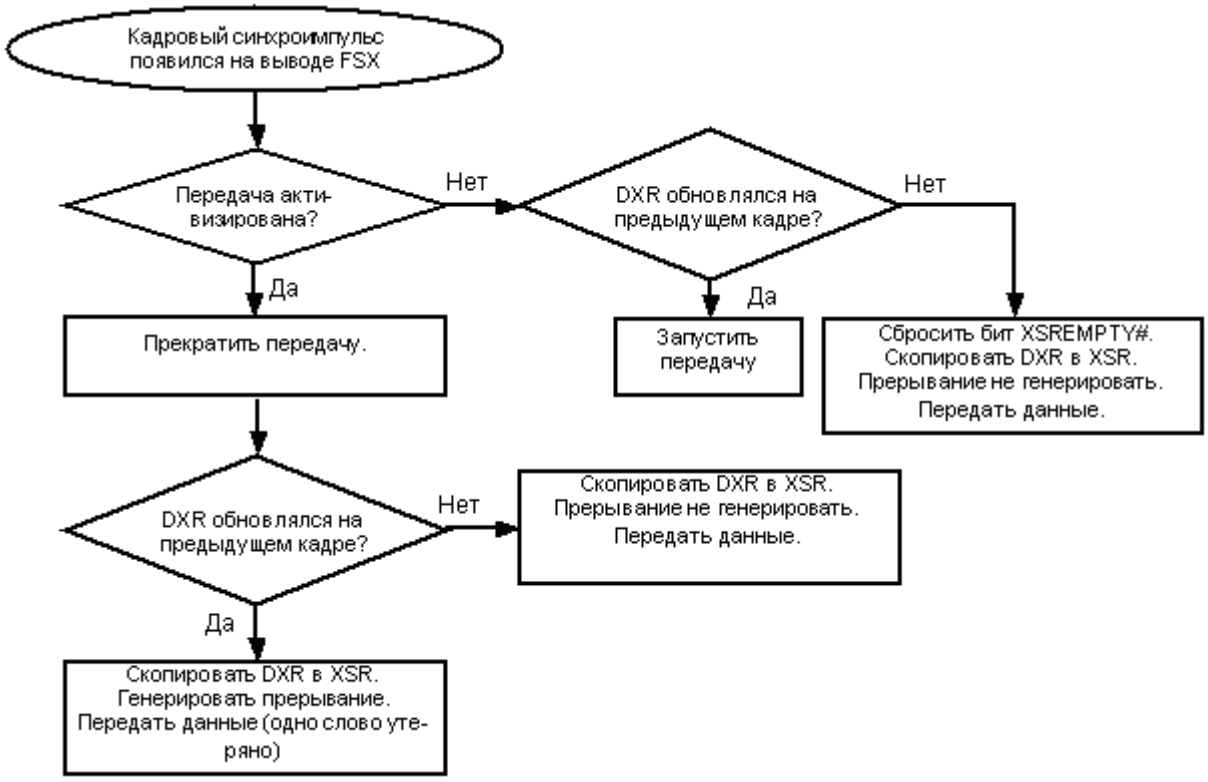


Рисунок 68 – Ошибки передачи в пакетном режиме

В непрерывном режиме ошибки приобретают большее значение. В этом режиме потеря значащих разрядов на передаче (XRESEMPTY# = 0) рассматривается как ошибка, поскольку данные не будут переданы. Как и в пакетном режиме, переполнение буферов на приеме также приводит к ошибке. Оба этих случая приводят к останову приемного и передающего трактов последовательного порта. Реакция порта на эти ошибки рассматривается в разделе описания флагов {XRESEMPTY# и RSRFULL. Ошибки, связанные с потерей значащих разрядов и переполнением, не являются неисправимыми; они могут быть скорректированы посредством считывания регистра DRR или записи в регистр DXR. Для обнуления бита XRESEMPTY# нужно обновить содержимое регистра DXR перед приходом кадрового синхроимпульса, т. е. в непрерывном режиме (FSM = 0) он будет сгенерирован автоматически в режиме внутреннего кадрового генератора или его необходимо сгенерировать в режиме внешнего генератора. На приемной же стороне для обнуления бита RSRFULL кадровый синхроимпульс не требуется. Достаточно просто считать регистр DRR. Приемник последовательного порта сохраняет границы элемента данных (8 или 16 разрядов) даже тогда, когда он не принимает данные.

Другой причиной ошибки является появление кадровых синхроимпульсов во время передачи. В непрерывном режиме после начального кадрового синхроимпульса не должно появляться никаких других синхроимпульсов. Если все же во время передачи кадровый синхроимпульс появится, передача аварийно прервется, и данные в XSR потеряются. Новый цикл передачи инициируется, как только модифицируется регистр DXR для следующей передачи. Для приема в непрерывном режиме ситуация аналогична: при появлении кадрового синхроимпульса один пакет данных (8-разрядный байт или 16-разрядное слово, в зависимости от состояния бита FO) теряется. Счетчик принятых разрядов в RSR сбрасывается в исходное состояние, поэтому данные, поступившие в регистр RSR со входа DR, теряются. Данные, поступающие от DR после появления

кадрового синхроимпульса, записываются в RSR, таким образом, появление лишнего кадрового синхроимпульса в непрерывном режиме эквивалентно перезапуску процесса обмена по последовательному порту.

На рисунках 69 и 70 показаны блок-схемы алгоритмов обработки ошибочных ситуаций на приеме и передаче для непрерывного режима. Следует отметить, что если кадровый синхроимпульс поступает после обнуления флага RSRFULL посредством считывания DRR, но до начала границы следующего слова (8 или 16 бит), имеет место аварийное прекращение приема. Кроме того, следует учитывать разницу между ошибками непрерывного и пакетного режимов на передаче. Если XSREMPTY# активен в непрерывном режиме и появляется внешний кадровый синхроимпульс, старые данные не передаются. Эта ситуация квалифицируется как ошибка, кадровый синхроимпульс игнорируется и вывод DX остается в состоянии высокого импеданса.

Рисунок 69 – Ошибки приема в непрерывном режиме

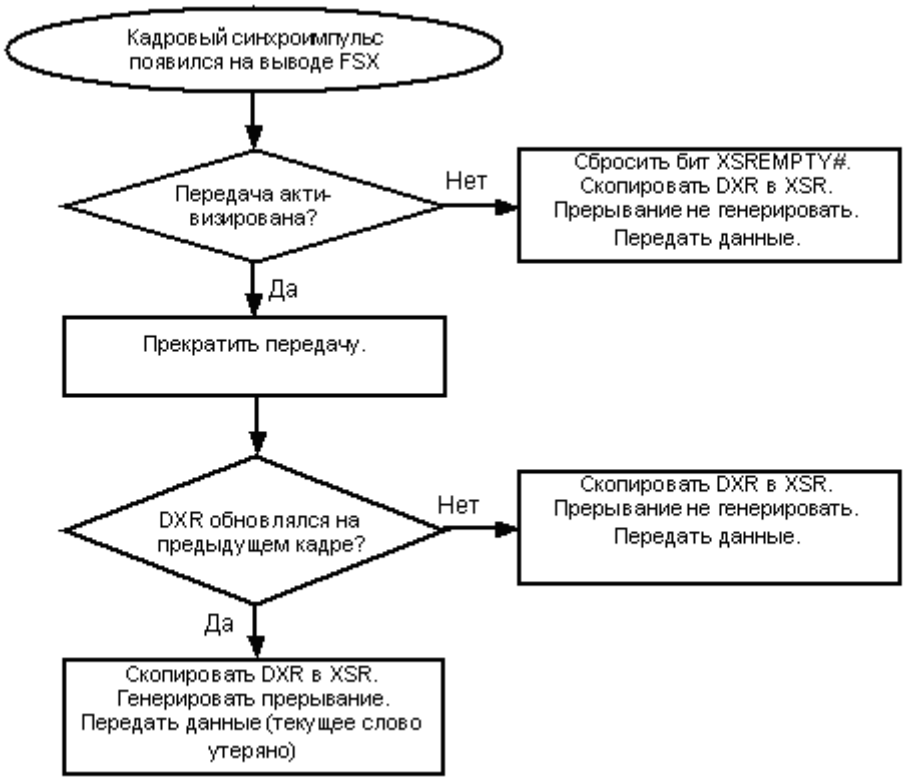


Рисунок 70 – Ошибки передачи в непрерывном режиме

6.7 Буферизированный последовательный порт

Буферизированный последовательный порт (BSP) является полнодуплексным с двойной буферизацией данных последовательным портом, представляющим собой расширенную версию стандартного последовательного порта и с блоком автоматической буферизации (ABU) (рисунок 71). Логика блока ABU позволяет порту BSP обмениваться с внешними устройствами массивами данных, размещенных в определенном внутреннем блоке памяти кристалла, на аппаратном уровне без участия процессора.

Полнодуплексный порт BSP обеспечивает удобную связь с последовательными устройствами типа кодер-декодеров, последовательных АЦП/ЦАП преобразователями и другими последовательными устройствами с привлечением минимума внешних аппаратных средств. Двойная буферизация данных в порте BSP позволяет обмениваться непрерывным потоком данных длиной в 8, 10, 12 или 16 бит. Потоки данных могут сопровождаться импульсами кадровой синхронизации с программируемой частотой как на передаче, так и на приеме. Полярность сигналов кадровой и тактовой синхронизации также программируется. Максимальная скорость обмена через порт BSP определяется частотой CLKOUT (40 Мбит/с для 25 нс и 50 Мбит/с для 30 нс). Порт BSP может работать в режиме кодоимпульсной модуляции, который позволяет использовать простой интерфейс с линией типа PCM. Режим работы порта BSP в стандартном (небуферизированном) режиме детально описан в разделе, посвященном стандартному последовательному порту.

ABU имеет собственный набор регистров круговой циклической адресации отдельно для приема и передачи с соответствующими блоками генерации адреса. Буферная память для передачи и приема размещается внутри специального блока внутренней памяти данных процессора объемом (2К×16) бит. Этот блок универсальный в том смысле, что он может использоваться процессором для своих собственных нужд, однако, это – единственный блок памяти, в котором можно создать автоматический буфер для порта BSP.

При использовании режима автобуферизации обмен данными происходит аппаратно между последовательным портом и внутренней памятью 1867ВЦ4Т, адреса ячеек которой вычисляются внутренними генераторами адресов блока ABU. Адрес начала буфера и его длина программируемы, при опустошении/переполнении буфера генерируется запрос на соответствующее прерывание процессора. При необходимости программист может легко отключить буферы. Управление блоком ABU детально рассмотрено ниже.

Режимы автобуферизации для приемника и передатчика включаются независимо. При отключенной автобуферизации порт работает в стандартном режиме и требованиями на прерывания BXINT и BRINT с передачей или приемом каждого слова. В режиме автобуферизации эти прерывания вырабатываются по опустошению или заполнению половины каждого из буферов.

Так как стандартные операции BSP режима аналогичны стандартным операциям простого последовательного порта, с них и следует начать.

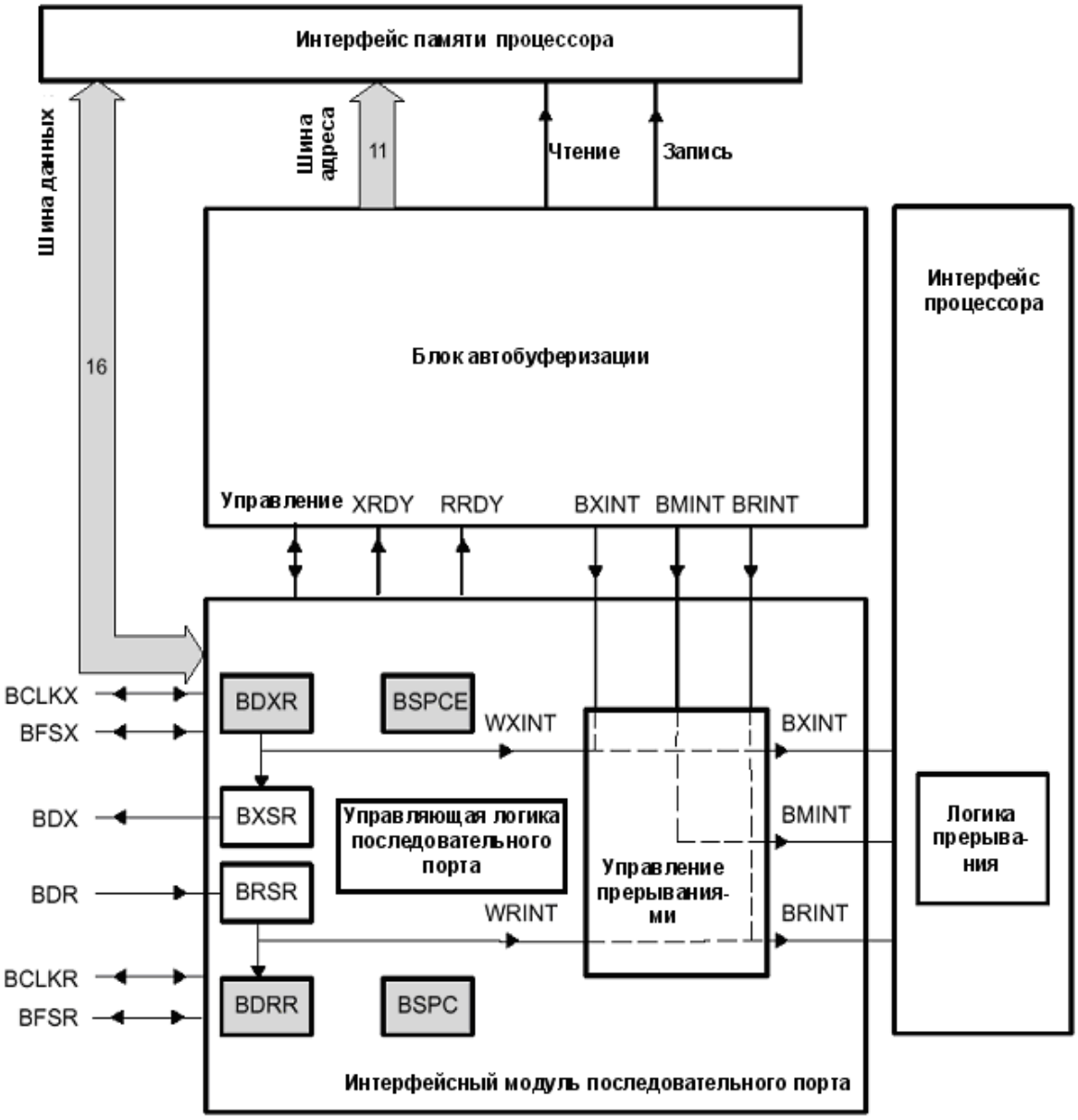


Рисунок 71 – Функциональная схема порта BSP

6.7.1 Операции BSP в стандартном режиме

Порт BSP использует собственные, отображенные в памяти, регистры приема, передачи и управления (BDXR, BDRR, и BSPC). При выполнении операций в режиме автобуферизации порт BSP использует также дополнительный регистр управления расширением возможностей порта BSPCE. Регистры BDRR, BDXR и BSPC предназначены для тех же целей, что и регистры DRR, DXR и SPC в простом последовательном порте. Как и в простом последовательном порте, порт BSP передает и получает данные через регистры сдвига BXSR и BRSR, обеспечивающие двойное буферирование и не доступные из программы. Если последовательный порт не используется, регистры BDXR и BDRR могут быть задействованы программным обеспечением как универсальные. В этом случае вывод BFSR не должен использоваться во избежание ложного обмена информацией.

Примечание – При использовании автобуферизации доступ к регистрам BDXR или BDRR ограничен, т. е. регистр BDRR может только читаться, а BDXR доступен только для записи. При этом блок ABU должен быть заблокирован. Когда порт BSP находится в

сбросе, регистр BDRR доступен только для записи. Регистр BDXR может читаться в любое время.

Все регистры буферизированного последовательного порта сведены в таблицу 53. Блок ABU использует несколько дополнительных регистров, которые обсуждены в следующем подразделе.

Таблица 53 – Регистры буферизированного последовательного порта

Регистр	Описание
BDRR	16-битный буферный регистр приема
BDXR	16-битный буферный регистр передачи
BSPC	16-битный управляющий регистр
BSPCE	16-битный расширенный регистр управления
BRSR	16-битный сдвиговый регистр приема
BXSR	16-битный сдвиговый регистр передачи

6.7.1.1 Операции BSP в стандартном режиме

Различия между портами, когда BSP порт находится в стандартном режиме, сведены в таблицу 54.

Таблица 54 – Различия между простым последовательным портом и портом BSP в стандартном режиме

Условия	Простой последовательный порт	Порт BSP
Бит RSRFULL устанавливается	Бит RSRFULL устанавливается, когда регистр RSR не скопирован до прихода следующего синхроимпульса FSR в пакетном режиме. В непрерывном режиме бит RSRFULL устанавливается при первом заполнении регистра RSR	Бит RSRFULL устанавливается после заполнения регистра BRSR
Сохранение данных в регистре RSR при переполнениях	Регистр RSR сохраняет содержимое при переполнениях	Регистр BRSR не сохраняет содержимое при переполнениях
В непрерывном режиме прием перезапускается после переполнения	Прием перезапускается после чтения DRR	Прием не перезапускается, регистр BDRR читается после появления сигнала BFSR
Расширение знака при передаче в регистр DRR 8, 10 или 12 разрядных данных	Нет	Да
Генерация сигналов XRDY/XINT после загрузки регистра XSR при сброшенном бите XRESEMPTY#	Имеет место, когда регистр DXR загружен	

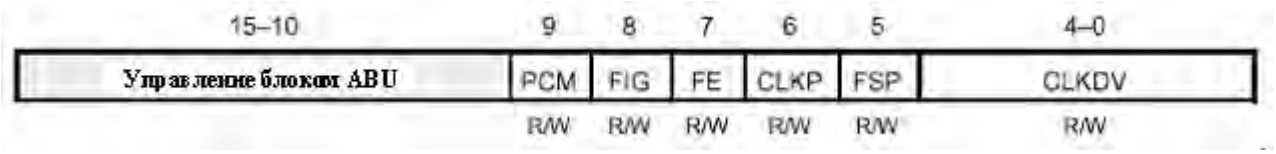
Окончание таблицы 54

Условия	Простой последовательный порт	Порт BSP
Программная доступность регистров DXR и DRR	Регистры DRR и DXR программно доступны в любое время. Имейте в виду, что информация в DRR обновляется после приема очередной посылки, а перезапись DXR раньше времени приводит к потере информации	
Максимальная частота сдвига данных из последовательного порта	CLKOUT/2	
Требования к синхронизации в процессе инициализации	Последовательный порт может быть выведен из сброса в любое время без оглядки на сигналы FSX/FSR в том случае, если биты XRST#/RRST# находятся в состоянии 1. Синхроимпульсы просто игнорируются	
Функционирование в режиме IDLE2/3	Нет	

6.7.1.2 Расширенные возможности порта BSP

Расширенные свойства BSP включают в себя возможность генерации сдвиговой частоты последовательного порта, выбора полярности для импульса тактовой синхронизации и ограничения длины передаваемых сообщений до 10- или 12-разрядных слов в дополнение к 8- и 16-разрядным передачам, предлагаемым простым последовательным портом. Дополнительно порт BSP предоставляет возможность игнорирования кадровых синхроимпульсов, пока не запрограммирован другой режим, и обеспечивает специальный режим обмена, который облегчает сопряжение с интерфейсами типа PCM.

Регистр BSPCE содержит биты состояния, которые используются в реализации расширенного режима BSP и управления блоком ABU. Десять младших разрядов BSPCE выделены для управления расширенным режимом, в то время как 6 старших разрядов используются для управления блоком ABU и будут обсуждены ниже. Рисунок 72 демонстрирует распределение бит регистра BSPCE, а в таблице 54 описаны их функции. Содержимое регистра BSPCE после сброса равно 3. Это обеспечивает режим совместимости с простым последовательным портом. В таблице 55 представлена зависимость длины слова от состояния бит FO и FE.



R – чтение, W – запись

Рисунок 72 – Распределение бит в регистре BSPCE

Таблица 54 – Функциональное назначение полей регистра BSPCE

Бит	Имя	Значение после сброса	Функция
15-10	Управление блоком ABU	-	Зарезервированы для управления модулем ABU (см. следующий подраздел)
9	PCM	0	Режим кодово-импульсной модуляции. Этот бит разрешает/запрещает режим совместимости с интерфейсом типа PCM в режиме передачи. На режим приема бит PCM не влияет. PCM = 0 – режим кодово-импульсной модуляции заблокирован. PCM = 1 – режим кодово-импульсной модуляции разрешен. Содержимое регистра BDXR в этом режиме передается тогда, когда его старший бит сброшен в ноль. Если этот бит установлен в единицу, содержимое регистра BDXR не передается, а линия BDX находится в высокоимпедансном состоянии в течение цикла передачи
8	FIG	0	Игнорирование кадра. Этот бит функционирует только в непрерывном режиме обмена с внешним кадровым синхроимпульсом. FIG = 0 – следующий за первым, инициализирующим передачу, кадровый синхроимпульс перезапускает ее. FIG = 1 – следующий за первым, инициализирующим передачу, кадровый синхроимпульс игнорируется
7	FE	0	Расширение формата. FE бит вместе с битом FO в регистре BSPC определяет длину передаваемого слова. Когда FO FE = 00, передаются 16-разрядные слова; когда FO FE = 01 - 10-разрядные слова; когда FO FE = 10 – 8-разрядные слова и, когда FO FE = 11 – 12-разрядные слова. Обратите внимание, что для 8, 10 и 12 разрядных слов происходит выравнивание справа, а знаковый разряд расширяется до полного 16-разрядного слова
6	CLKP	0	Полярность сдвиговых синхроимпульсов. Этот управляющий бит определяет момент стробирования принимаемых и передаваемых данных. CLKP = 0 – данные стробируются приемником на падающем фронте сигнала BCLKR, а выдвигаются передатчиком на возрастающем фронте сигнала BCLKX. CLKP = 1 – данные стробируются приемником на возрастающем фронте сигнала BCLKR и выдвигаются передатчиком на падающем фронте сигнала BCLKX
5	FSP	0	Полярность кадрового синхроимпульса. Этот служебный бит определяет активный уровень кадровых синхроимпульсов (BFSX и BFSR) FSP = 0 – активным является высокий уровень импульсов (BFSX и BFSR). FSP = 1 – активным является низкий уровень импульсов (BFSX и BFSR)

Окончание таблицы 54

Бит	Имя	Значение после сброса	Функция
4-0	CLKDV	00011	Управление коэффициентом деления, определяющим частоту сдвиговых синхроимпульсов на передаче. Когда MCM бит в регистре BSPC установлен в 1, сигнал CLKX генерируется внутренним генератором имеющим частоту, равную 1/(CLKDV + 1) CLKOUT. CLKDV принимает значения в диапазоне 0-31. Когда значение CLKDV нечетно или равно 0, скважность импульсов CLKX равна 2. Когда CLKDV принимает четное значение (CLKDV = 2p), скважность меняется в зависимости от состояния разряда CLKP. Когда CLKP = 0, высокий уровень на CLKX держится p + 1 циклов сигнала CLKOUT, а низкий уровень - p циклов, а когда CLKP = 1, наоборот

Таблица 55 – Зависимость длины слова от состояния бит FO и FE

F0	FE	Конфигурация порта BSP по длине слова
0	0	Обмен 16-разрядными словами. (Устанавливается после сброса)
0	1	Обмен 10-разрядными словами
1	0	Обмен 8-разрядными словами
1	1	Обмен 12-разрядными словами

Расширенные возможности позволяют получить большую гибкость интерфейса последовательного порта в ряде применений. В частности игнорирование кадровых синхроимпульсов позволяет сжимать данные на приеме, если они передаются в непрерывном режиме и не в 16-разрядном формате с внешними кадровыми синхроимпульсами. Когда бит FIG = 0 каждый следующий кадровый синхроимпульс перезапускает передачу, когда бит FIG = 1, все кадровые синхроимпульсы, кроме первого, игнорируются. Установка бита FIG в единицу позволяет порту на приеме рассматривать поступающие данные как непрерывный поток 16-разрядных слов, не смотря на то, что передатчик каждые 8, 10 или 12 бит сопровождает кадровым синхроимпульсом. Если не использовать бит FIG, каждый элемент данных размером меньшим чем 16 бит затребует для своего хранения целого слова памяти и, в случае байтовых передач, двух операций передачи и двух операций приема. Таким образом, использование бита FIG значительно уменьшит размер приемного буфера в автобуферизированном и стандартном режимах и количество процессорных циклов, необходимых для обработки передач последовательного порта в стандартном режиме. Рисунок 73 демонстрирует прием в порт BSP конфигурированный в 16-разрядном формате байтового потока.

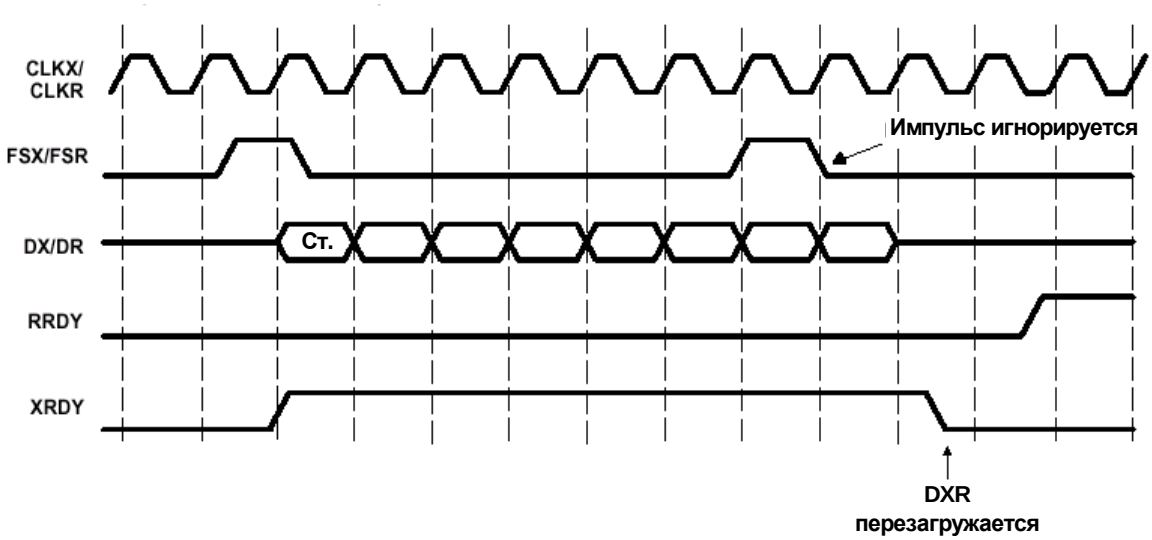


Рисунок 73 – Обмен в непрерывном режиме с внешними кадровыми синхроимпульсами и FIG = 1 (формат – 16 битов)

6.7.2 Операции автобуферизации

Операции с модулем ABU увеличивают функциональные возможности последовательного порта. Обратите внимание, что в режиме автобуферизации последовательный порт управляется теми же битами состояния в регистрах BSPC и BSPCE, что и в стандартном режиме.

Блок ABU позволяет обмениваться информацией между портом и внутрикристальной памятью без вмешательства процессора.

Блок ABU использует пять отображенных в памяти регистров: регистр адреса передаваемого массива (AXR), регистр размера передаваемого блока (BKX), регистр адреса принимаемого массива (ARR), регистр размера принимаемого блока (BKR) и регистр BSPCE. Эти регистры перечислены в таблице 56.

Таблица 56 – Регистры, используемые блоком ABU

Регистр	Описание
BSPCE	16-разрядный расширенный регистр управления
AXR	11-разрядный адресный регистр передаваемых данных
BKX	11-разрядный регистр размера буфера передаваемых данных
ARR	11-разрядный адресный регистр принимаемых данных
BKR	11-разрядный регистр размера буфера принимаемых данных

На рисунке 74 представлена функциональная схема блока ABU. Регистр BSPCE содержит биты управления блоком ABU, действие которых будет обсуждено ниже. Регистры AXR, BKX, ARR, BKR и связанные с ними круговые буферы позволяют перемещать массивы информации между внутренней памятью процессора и передающим или приемным регистрами (BDXR или BDRR) BSP порта в режиме автобуферизации. Все эти операции будут обсуждены в этом разделе ниже.

Следует обратить внимание, что регистры AXR, BKX, ARR и BKR – 11-разрядные, но читаются они как 16-разрядные слова с обнуленными пятью старшими битами. Если режим автобуферизации не используется, эти регистры можно задействовать как универсальные 11-разрядные ячейки памяти.

Передающий и принимающий буферы блока ABU независимы. Когда какой-либо буфер подключен, доступ к соответствующему регистру данных последовательного порта

(BDXR или BDRR) из программы ограничен: регистр BDRR доступен только по чтению, а регистр BDXR – только по записи при заблокированном блоке ABU. Когда порт BSP находится в сбросе, регистр BDRR доступен только по записи. Регистр BDXR может читаться в любое время. Когда при передаче или приеме автобуферизация заблокирована, соответствующий буфер будет функционировать в стандартном режиме.

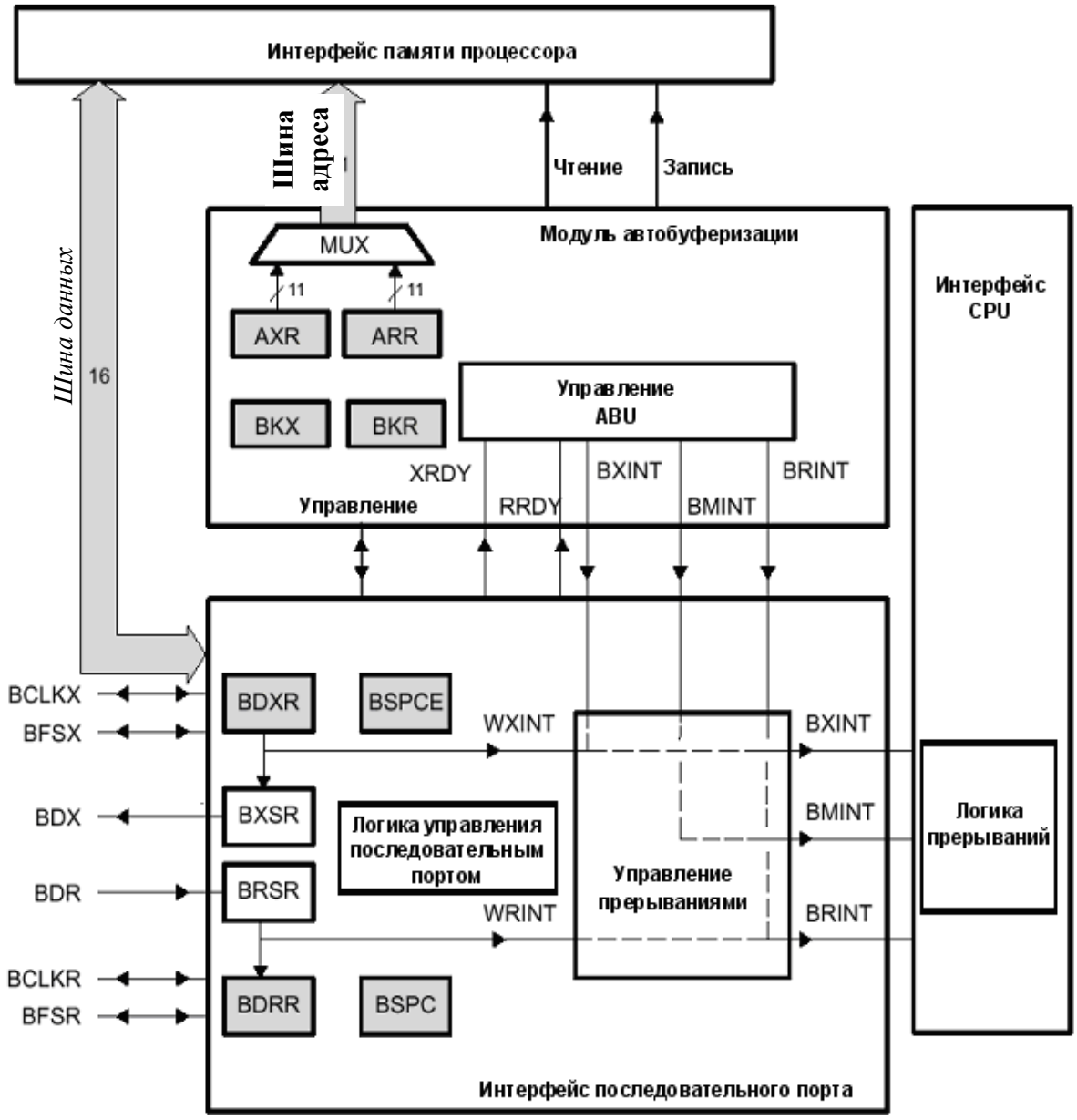


Рисунок 74 – Блок-схема устройства автобуферизации BSP порта

Блок ABU имеет возможность генерировать прерывания в CPU, когда передающий буфер освободится наполовину или полностью, или приемный буфер заполнится наполовину или полностью. Прерывания по приему или передаче одного элемента данных в режиме автобуферизации не генерируются. Этот механизм позволяет автоматически завершить накопление данных в буфере по завершении заполнения верхней или нижней половины. Пакетный или непрерывный режимы могут использовать автоматический буфер. Следует обратить внимание, что применение режима автобуферизации гораздо

эффективнее при непрерывных передачах с импульсом FSX, сгенерированным BSP в начале каждой передачи, чем при пакетных с внутренним кадровым синхроимпульсом.

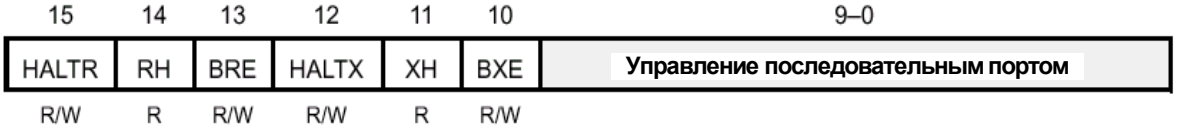
Автоматический буфер размещается во вполне определенном блоке памяти типа DARAM, объемом в (2K×16) бит, который может быть конфигурирован как память данных, память программ или как то и другое. Этот блок – единственный, в котором автобуферизация возможна. Базовый адрес блока памяти, предназначенного для автоматического буфера, 0800h.

Со стороны процессора блок автоматического буфера доступен как область программ и/или данных и тогда, когда блок ABU включен. Возможно возникновение конфликтов между процессором и блоком ABU при одновременном обращении к этому блоку. В этой ситуации предпочтение отдается блоку ABU, а обращение от процессора задерживается на один такт. Хуже всего обстоит дело в случае конфликтов при однословных обменах. Прямой доступ к памяти от внешних устройств отрабатывается в автоматическом буфере только тогда, когда блок ABU заблокирован. Этот блок в силу свойств защиты ROM не может быть отображен в область памяти программ.

Когда блок ABU разрешен для передачи и для приема и запросы из последовательного порта, сгенерированные обоими направлениями обмена приходят одновременно, запрос на передачу имеет приоритет перед приемным запросом. В этом случае сначала обслуживается запрос на передачу, а запрос на прием блокируется генератором состояния ожидания. По завершении цикла передачи память предоставляется приему данных.

6.7.2.1 Регистры управления блоком автобуферизации

Шесть старших бит в регистре BSPCE образуют регистр управления блоком ABU (ABUC). Некоторые из этих бит доступны только для чтения, в то время как другие – для чтения и записи. На рисунке 75 представлено распределение бит в ABUC, а в таблице 57 приводится описание функции каждого бита ABUC в регистре BSPCE. После сброса регистр BSPCE содержит число 3.



R – чтение, W – запись

Рисунок 75 – Распределение бит в регистре управления блоком ABU, являющимся частью регистра BSPCE

Таблица 57 – Описание назначения бит регистра управления блоком ABU

Бит	Имя	Значение после сброса	Функция
15	HALTR	0	Останов автобуферизации на приеме. Этот служебный бит управляет остановкой автобуферизации на приеме после заполнения половины текущего буфера. HALTR = 0 – автобуферизация продолжает функционировать после заполнения текущей половины буфера. HALTR = 1 – автобуферизация остановлена после заполнения текущей половины буфера. Когда это происходит, бит BRE сбрасывается в 0, а последовательный порт продолжает функционировать в стандартном режиме
14	RH	0	Бит готовности, указывающий на факт заполнения половины буфера принятой информацией. Этот бит предназначен только для чтения и указывает на то, какая половина приемного буфера заполнена. Бит RH опрашивается в подпрограмме обработки прерывания RINT (или по флагу IFR) . RH = 0 – первая половина буфера заполнена и прием в настоящее время ведется во вторую половину буфера. RH = 1 – вторая половина буфера заполнена и прием в настоящее время ведется в первую половину буфера
13	BRE	0	Разрешение автобуферизации на приеме. BRE = 0 – автобуферизация заблокирована, и последовательный порт функционирует в стандартном режиме. BRE = 1 – автобуферизация на приеме разрешена
12	HALTX	0	Останов автобуферизации на передаче. Этот служебный бит управляет остановкой автобуферизации на передаче после заполнения текущей половины буфера. HALTX = 0 – автобуферизация продолжает функционировать после передачи текущей половины буфера. HALTX = 1 – автобуферизация остановлена после передачи текущей половины буфера. Когда это происходит, бит BXE очищается и последовательный порт продолжает функционировать в стандартном режиме
11	XH	0	Бит готовности, указывающий на факт окончания передачи из одной половины буфера. Этот бит предназначен только для чтения и указывает на то, какая половина буфера была передана. XH опрашивается в подпрограмме обработки прерывания XINT (или по флагу IFR). XH = 0 – первая половина буфера была передана и передача в настоящее время осуществляется из второй половины буфера. XH = 1 – вторая половина буфера была передана и передача в настоящее время осуществляется из первой половины буфера

Окончание таблицы 57

Бит	Имя	Значение после сброса	Функция
10	BXE	0	Бит разрешения автобуферизации на передаче. BXE = 0 – автобуферизация заблокирована и последовательный порт функционирует в стандартном режиме. BXE = 1 – автобуферизация на передаче разрешена
9-0	Управ-ление последова тельным портом	-	Это поле описано выше

6.7.2.2 Процесс автобуферизации

Каждый раз, когда происходит обмен через последовательный порт в автобуферизированном режиме, данные автоматически перемещаются в или из автоматического буфера под управлением блока ABU. При этом прерывания вырабатываются не от каждого принятого или переданного слова, а только по заполнении или очищении половины соответствующего автоматического буфера. Таким образом, процессор освобождается от обработки каждого перемещаемого слова и обрабатывает содержимое половины автоматического буфера.

Размещение автоматического буфера внутри выделенного блока памяти и его размер программируются с использованием 11-разрядных регистров адреса (AXR и ARR) и 11-разрядных регистров размера блока (BKX и BKR). Передающий и приемный буферы могут размещаться в различных областях памяти (но внутри выделенного блока), могут накладываться друг на друга или занимать одну область, что позволяет осуществлять передачу и прием из одного и того же буфера.

Автобуферизация использует круговой механизм адресации для обращения к ячейкам памяти внутри автоматического блока. Этот механизм функционирует одинаково для приема и передачи. Для каждого направления обмена имеются по два регистра, предназначенных для хранения размера буфера и текущего адреса ячейки в буфере. Эти регистры имеют обозначения BKX, BKR, ARX и ARR. Каждый из пары регистров (размера блока и адреса) полностью определяют верхнюю и нижнюю границы адресов буфера для передачи и приема. Обратите внимание, что этот механизм адресации работает только внутри выделенного (2K×16)-битного блока ABU. Процессор обращается к этой области с помощью выбранного программистом способа адресации

Круговой механизм адресации позволяет каждый раз, когда адрес ячейки внутри буфера достигает его нижней границы, автоматически возвращаться к адресу верхней границы буфера. Круговой механизм адресации инициализируется посредством загрузки регистра BKX (BKR) точным размером используемого буфера (но не величиной, равной размеру без единицы) и регистра ARX (ARR) значением адреса ячейки, расположенной внутри этого буфера. Часто стартовый адрес внутри буфера берут равным 0, указывая тем самым на начало буфера, но в общем случае он может принимать любое значение внутри заданного буферного диапазона.

Регистр BKX (BKR) может рассматриваться как бы состоящим из двух частей: старшей (BKH), содержащей нули после последней, значащей, единицы размера буфера, и младшей части (BKL) длиной N бит, содержащей собственно размер буфера без расширяющих нулей. Таким же образом можно разбить и адресный регистр ARX (ARR) и

его младшую часть ARL длиной в N бит заменить нулями, при этом измененное содержимое регистра будет указывать на нижнюю границу буфера TBA. Тогда верхнюю границу буфера BBA можно определить как конкатенацию регистра ARX (ARR) с регистром BKX (BKR). Буфер состоит из двух половин: адресный интервал для первой половины - TBA =>(BKL/2) - 1 и для второй половины BKL/2 =>(BKL - 1). Рисунок 76 иллюстрирует все вышесказанное.

Минимальный размер автоматического буфера – два, максимальный размер – 2047, и любые буферы размером от 2047 до (1024×16) бит должны начинаться с адреса 0x0000 относительно базового адреса блока ABU. Если регистр адреса (AXR или ARR) загружен значением, выходящим за пределы выбранного буфера, операция ABU может быть выполнена неверно. В этом случае адресная арифметика блока ABU будет наращивать адрес ячейки после каждого доступа до тех пор, пока он не достигнет границы реального буфера, определенной содержимым регистра BKX (или BKR), после чего адрес принудительно сбросится в начало текущего буфера. В дальнейшем операции обмена через последовательный порт будут использовать именно этот буфер. Следует заметить, что любые операции, выполняемые с неправильно загруженным ARX/R могут неожиданно разрушить некоторые области памяти.

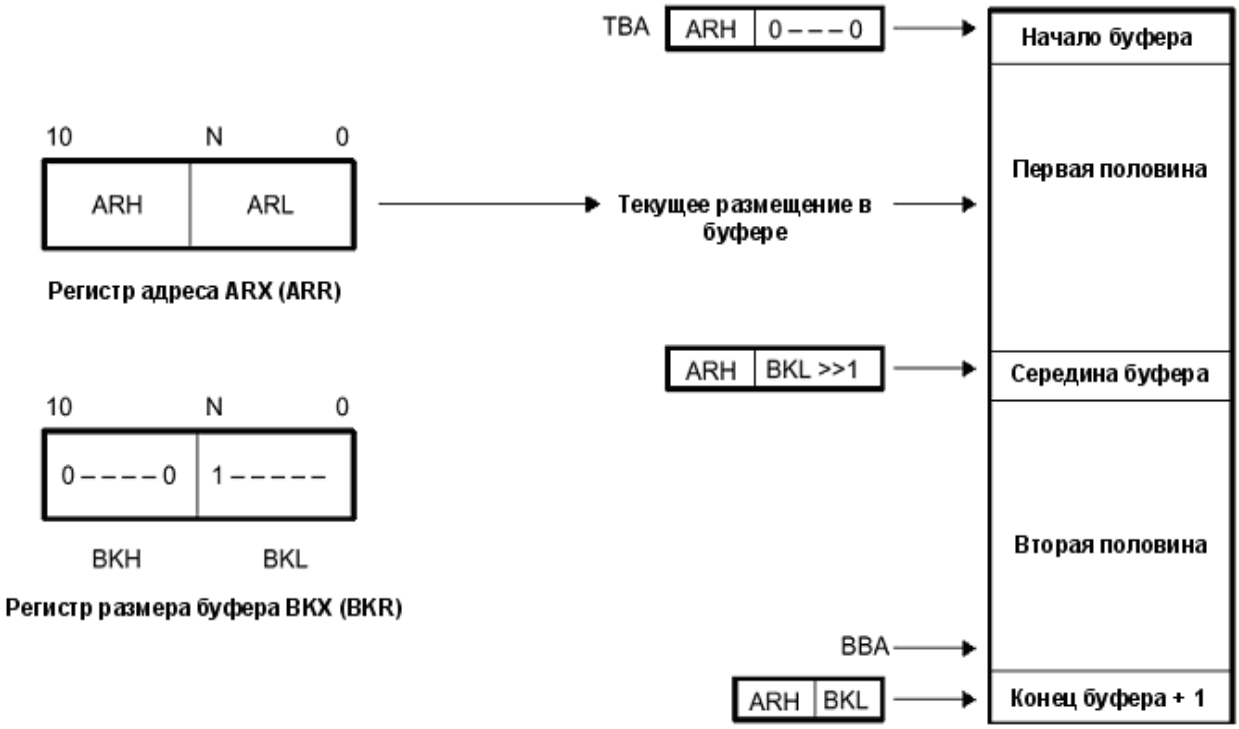


Рисунок 76 – Регистр циклического адреса

Следующий пример иллюстрирует некоторые из аспектов процесса автобуферизации. Рассмотрим буфер передачи размером 5 (BKX = 5) и буфера приема размером 8 (BKR = 8) как показано на рисунке 77. Передающий буфер может начинаться с любого относительного адреса, который является кратным 8 (т. е. 0x0000, 0x0008, 0x0010, 0x0018, ..., 0x07F8), а приемный буфер может начинаться с любого относительного адреса, кратного 16 (0x0000, 0x0010, 0x0020, ..., 0x07F0).

В этом примере, буфер передачи начинается в относительном адресе 0x0008, а буфер приема в относительном адресе 0x0010. AXR может, следовательно, содержать любое значение в диапазоне 0x0008-0x000C, а ARR может содержать любое значение в диапазоне 0x0010-0x0017. Если AXR в этом примере был загружен значением 0x000D (неприемлемый буфер для размера 5), адрес очередной ячейки будет наращиваться до

значения 0x0010, приемлемого для размера буфера, равного 5. Если это произойдет, то передающий и приемный буферы пересекутся, что может привести к непоправимой ошибке.

Процесс автобуферизации активизируется после запроса из последовательного порта, когда XRDY или RRDY устанавливаются в 1, извещая, что слово было получено. Затем происходит обращение к памяти, после чего генерируется прерывание в том случае, если текущая половина буфера (первая или вторая) была обработана. На обработанную часть буфера указывают флажки RH и XH в регистре BSPCE.

Если выбран режим автоматического отключения ABU (HALTX или HALTR бит установлен), то по достижении границы текущей половины (первой или второй) бит BXE или BRE в BSPCE очищается и автобуферизация блокируется, что запрещает генерацию дальнейших запросов к ABU. Когда блокируется автобуферизация на передаче, передача текущего содержимого XSR и последнего значения, загруженного в DXR, осуществляется так, как эти передачи уже были инициализированы. Следовательно, при использовании функции HALTX, будет иметь место некоторая задержка между моментом пересечения буферной границы и фактической остановкой передачи. Если этот факт необходимо идентифицировать, программа должна опросить на предмет выполнения условия биты XRDY = 1 и XSREMPTY# = 0 после того, как последний бит покинет передатчик. На рисунке 77 представлен пример организации автоматических буферов на передаче и приеме.

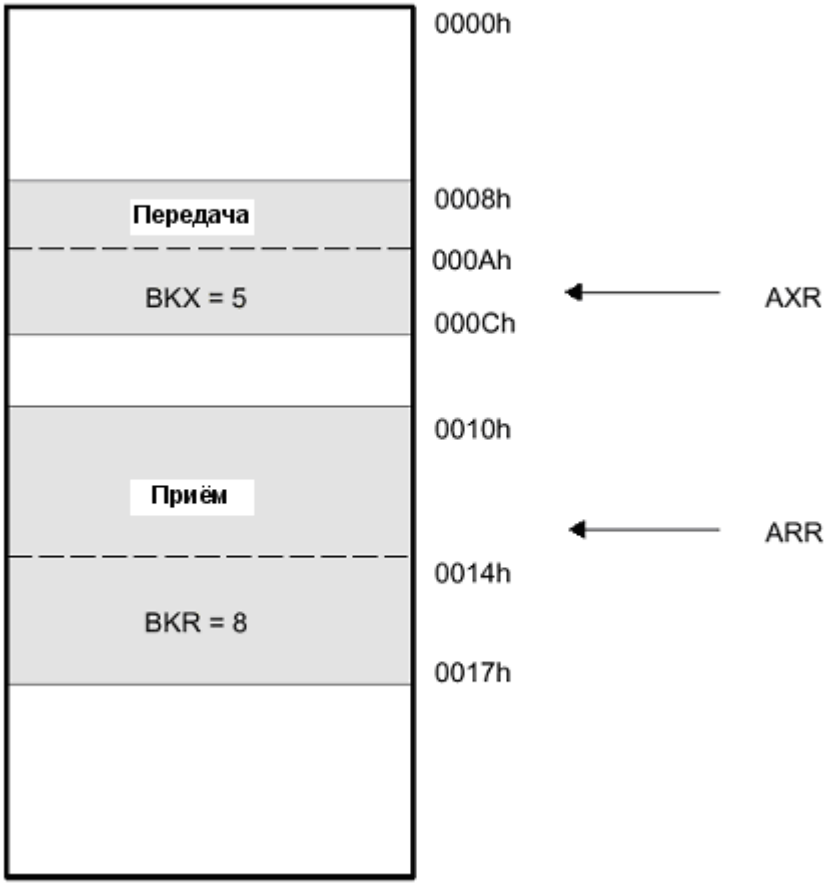


Рисунок 77 – Пример организации автоматических буферов на передаче и приеме

В приемнике, при использовании HALTR и, следовательно, отключении авто-буферизации, когда достигается граница буфера, следующие приемы могут быть потеряны, если программное обеспечение не обрабатывает прерывание по приему, т. к. BDRR больше не читается, а ABU отключен.

Последовательность событий, происходящих в процессе автобуферизации, выглядит следующим образом:

1) ABU обращается к буферной памяти.

2) Содержимое соответствующего регистра адреса увеличивается, если верхняя граница буфера не была достигнута.

3) Генерируется BXINT или BRINT и модифицируется XH/RH, если достигнута граница половины буфера.

4) Отключение ABU, если эта функция была выбрана, и произошло пересечение середины буферов.

6.7.2.3 Системные соглашения для BSP операций

Устройство 1867ВЦ4Т построено по статическому принципу, вследствие чего импульсы кадровой синхронизации могут влиять на процессы передачи по последовательному порту или на инициализацию его, то есть соответствующая операция должна закончиться до появления очередного FSX/FSR даже тогда, когда кадровые синхроимпульсы генерируются внешней по отношению к процессору схемой и, вследствие этого, не привязаны к процессу инициализации последовательного порта.

Временные соотношения при инициализации последовательного порта и BSP различаются. Последовательный порт может быть выведен из сброса в любое время относительно FSX/FSR, однако, если XRST#/RRST# переходят в 1 в течение или после того, как появится кадровый синхроимпульс, последний может игнорироваться. В стандартном режиме BSP с внешним кадровым синхроимпульсом на приеме и на передаче (TXM = 0), BSP должен быть выведен из сброса, по крайней мере, за один полный цикл CLKOUT плюс 1/2 периода тактовой синхронизации последовательного порта до момента стробирования кадрового синхроимпульса (были ли такты предварительно или нет).

Режим передачи с внутренними тактовыми и кадровыми синхроимпульсами не требует выполнения этого условия, т. к. кадровый синхроимпульс генерируется внутренними схемами автоматически (после того, как XRST# устанавливается в 1 после загрузки BDXR).

Однако, при использовании режима внешних тактовых синхроимпульсов с внутренними кадровыми синхроимпульсами, последний может быть пропущен в зависимости от момента очистки XRST# относительно тактов.

Рисунок 78 иллюстрирует процесс инициализации передатчика в стандартном режиме BSP с внешними кадровыми синхроимпульсами (TXM = 0) с высоким активным уровнем (FSP = 0) и внешними тактовыми синхроимпульсами(MCM = 0) с активным нарастающим фронтом (CLKP = 0). В этом примере, если импульс BFSX приходит в течение первого BCLKX, передача инициализируется правильно.

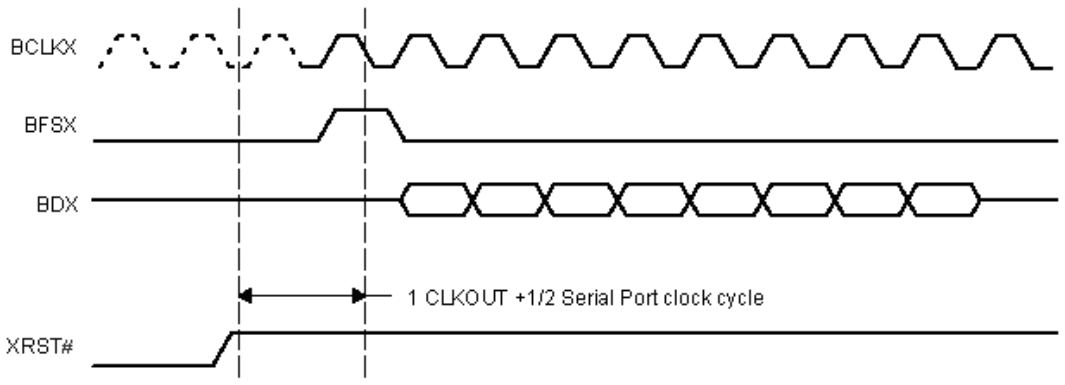


Рисунок 78 – Синхронизация инициализации BSP в стандартном режиме

В режиме автобуферизации на приеме и на передаче с внешними кадровыми синхроимпульсами (TXM = 0) BSP должен быть выведен из сброса, по крайней мере, за шесть периодов CLKOUT плюс 1/2 периода тактовых синхроимпульсов до момента стробирования кадрового синхроимпульса (были ли такты предварительно или нет). Эта задержка необходима для нормального функционирования логики ABU (рисунок 79).

Режим передачи с внутренними тактовыми и кадровыми синхроимпульсами этого не требует, т. к. все происходит автоматически.

Однако если используются внешние тактовые импульсы с внутренними кадровыми и если такты отсутствуют, когда XRST# очищен, кадровый синхроимпульс может быть пропущен в зависимости от момента очистки XRST# относительно тактов.

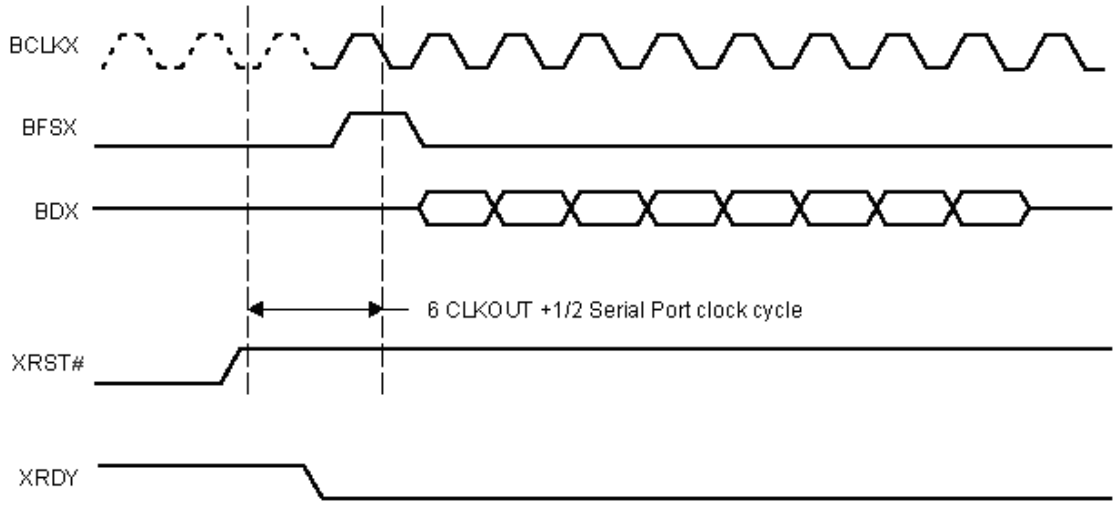


Рисунок 79 – Инициализация BSP порта в режиме автобуферизации

Рисунок 79 иллюстрирует процесс инициализации передач в режиме автобуферизации с внешними тактовыми (MCM = 0, данные стробируются нарастающим фронтом (CLKP = 0)) и кадровыми (TXM = 0) с высоким активным уровнем (FSP = 0) синхроимпульсами.

Операция запуска и перезапуска BSP в стандартном режиме требует от программы выполнения тех же самых шагов, что и инициализация последовательного порта, плюс – регистр BSPCE должен быть инициализирован для того, чтобы конфигурировать любое из желательных расширенных свойств. Чтобы запустить или перезапустить BSP в режиме автобуферизации следует выполнить те же операции и инициализировать регистр автобуферизации.

В качестве иллюстрации соответствующей операции буферизированного последовательного порта ниже приведены примеры, в которых определена последовательность действий. Эта иллюстрация основана на использовании прерываний с тем, чтобы обработать нормальный ввод/вывод между последовательным портом и CPU. Примеры иллюстрируют инициализацию буферизированного последовательного порта в режиме автобуферизации. В обоих случаях прерывания от приема и передачи используются для обслуживания ABU, однако можно обойтись и опросом регистра флагов прерывания (IFR).Секции приема и передачи могут быть инициализированы одновременно или отдельно в зависимости от требований системы.

В первом примере инициализируется последовательный порт для операций передачи в пакетном режиме с внешними тактовыми и кадровыми синхроимпульсами. Выбранный формат данных – 16 битов с высоким активным уровнем кадровых синхроимпульсов. Автобуферизация на передаче разрешается установкой бита BXE в

поле ABUC регистра BSPCE, и бита HALTX, который останавливает передачу после пересылки половины определенного буфера.

Во втором примере инициализируется последовательный порт для операции приема в непрерывном режиме. Активный уровень кадровых синхроимпульсов – низкий, формат данных – 16 битов, и выбран режим игнорирования кадровых синхроимпульсов для того, чтобы два полученных байта данных были упакованы в одно слово. Автобуферизация на приеме разрешается установкой бита BRE в поле ABUC регистра BSPCE.

6.7.2.3.1 Подпрограмма инициализации BSP порта при передаче

В примере 5 описывается подпрограмма инициализации BSP порта при передаче.

Пример 5 – Инициализации BSP порта при передаче

Действие	Описание
1) Сбросить и инициализировать последовательный порт, записав 0008 в BSPC	Эта операция сбрасывает передающую и приемную части последовательного порта таким образом, чтобы в дальнейшем последовательный порт мог функционировать с внешним сигналом FSX в режиме передачи и приема 16-разрядного слова
2) Очистить флаг прерывания последовательного порта, записав 0020h в IFR	Устраните последствия любых прерываний, которые, возможно, произошли перед инициализацией
3) Разрешить прерывания от последовательного порта посредством операции OR #0020h с IMR	Разрешаются прерывания от передачи
4) Разрешить глобальные прерывания (если это необходимо), очищая бит INTM в ST1	Глобальные прерывания должны быть разрешены для ответной реакции центрального процессора
5) Инициализировать ABU передачи, записывая 1400h в BSPCE	Заставляет BSP останавливать передачу в конце буфера, пока другой FSX не получен
6) Записать начальный адрес в AXR	Идентифицируйте адрес первого буфера для ABU
7) Записать размер буфера в BKX	Идентифицируйте размер буфера для ABU
8) Запустить последовательный порт, записав 0048h в BSPC	Передающая часть последовательного порта выводится из сброса и начинает операции с условиями, определенными в шагах 1 и 5

6.7.2.3.1 Подпрограмма инициализации BSP порта при приеме

В примере 6 описывается подпрограмма инициализации BSP порта при приеме

Пример 6 – Инициализации BSP порта при приеме

Действие	Описание
1) Сбросить и инициализировать последовательный порт, записав 0000 в BSPC	Эта операция обнуляет приемную часть последовательного порта и устанавливает последовательный порт так, чтобы функционировать впоследствии в режиме непрерывного приема 16-разрядных слов
2) Очистить любые ждущие прерывания последовательного порта, записав 0010h в IFR	Устраните последствия любых прерываний, которые, возможно, произошли перед инициализацией

Действие	Описание
3) Разрешить прерывания от последовательного порта посредством операции OR #0010h с IMR	Разрешаются прерывания на приеме
4) Разрешить глобальные прерывания (если это необходимо), очищая INTM бит в ST1	Глобальные прерывания необходимо разрешить для ответных действий CPU
5) Инициализировать ABU приема, записав 2160h в BSPCE	Этот шаг заставляет BSP получать непрерывный поток информации и не перезапускает его при получении нового FSR
6) Записать начальный адрес буфера в ARR	Идентифицируйте адрес первого буфера для ABU
7) Записать размер буфера в BKR	Идентифицируйте размер буфера для ABU
8) Запустить последовательный порт, записав 0080h в BSPC	Этот шаг выводит приемную часть последовательного порта из сброса и начинает операции с условиями, определенными в шагах 1 и 5

6.7.4 Операции BSP в режиме пониженного энергопотребления

В процессоре 1867ВЦ4Т имеется возможность переводить некоторые или все устройства в бездействующее состояние, что приводит к значительно меньшему потреблению мощности, чем в обычном режиме. Режим пониженной мощности может вызываться одним из двух способов: при выполнении команды IDLE или в режиме HOLD, когда на выводе HM присутствует низкий уровень. BSP, подобно другим периферийным устройствам (таймер, стандартный последовательный порт), может быть выведен из этого режима прерыванием от передачи (BXINT) или от приема (BRINT).

Когда включен режим IDLE или HOLD, BSP продолжает функционировать, как это имеет место с последовательным портом. Когда включен режим IDLE2/3, в отличие от последовательного порта и других on-chip периферийных устройств, которые им остановлены, BSP может все еще функционировать.

В стандартном режиме, если BSP использует внешние тактовые и кадровые синхроимпульсы, в то время как устройство находится в IDLE2/3, порт продолжает работать, и прерывание от передачи (BXINT) или от приема (BRINT) будет выводить устройство из IDLE2/3 режима, если INTM = 0 перед выполнением команды IDLE2 или IDLE3. С внутренними тактовыми или кадровыми синхроимпульсами BSP остается выключенным в IDLE2/3, пока центральный процессор не продолжит выполнение программы.

В режиме автобуферизации, если BSP использует внешние тактовые и кадровые синхроимпульсы в то время как устройство находится в IDLE2/3, передача или прием одной посылки включит внутренние часы BSP для генерации циклов, необходимых для обмена DXR (или DRR) с памятью. Как только обмен закончится, внутренние часы BSP автоматически выключатся, так что порт останется в IDLE2/3. Порт выводится из режима IDLE2/3 по прерыванию от передачи (BXINT) или от приема (BRINT), сгенерированного, когда соответствующий буфер ABU был на половину или полностью освобожден или заполнен. При этом бит INTM должен быть сброшен перед выполнением команды IDLE 2 или IDLE 3.

6.8 Мультиплексированный последовательный порт с разделением по времени (TDM)

Мультиплексируемый последовательный порт с разделением по времени (TDM) позволяет последовательно подключать к процессору 1867ВЦ4Т до семи других устройств. TDM порт, кроме того, обеспечивает простой и эффективный интерфейс для мультипроцессорных приложений.

Посредством бита TDM в управляющем регистре последовательного TDM порта TSPC, порт может быть сконфигурирован в мультипроцессорном режиме (TDM = 1) или в автономном режиме (TDM = 0). В этой главе описаны операции TDM порта, когда он сконфигурирован в мультипроцессорном режиме. Порт может быть закрыт для пониженного потребления мощности через XRST# и RRST# биты.

6.8.1 Мультиплексирование по времени

Основной режим мультиплексирования с разделением по времени – это разделение временных интервалов на несколько подинтервалов, каждый из которых представляет собой коммуникационный канал в соответствии с предварительно определенным порядком. Рисунок 80 показывает 4-канальную схему TDM. Первый временной слот обозначается как канал 1 (chan 1), второй – как канал 2 (chan 2) и т. д. Канал 1 активен в течение первого периода коммуникации и затем в течение каждого следующего четвертого периода. Оставшиеся три канала чередуются по времени за каналом 1.

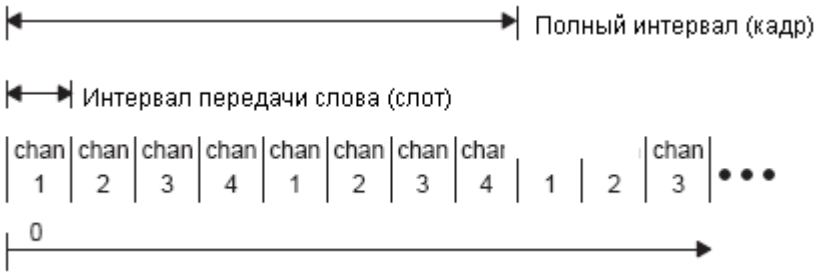


Рисунок 80 – Мультиплексирование с разделением по времени

TDM порт процессора 1867ВЦ4Т использует 8 TDM-каналов. Все элементы, передаваемые или получаемые по каждому из каналов, могут быть заданы независимо. Это позволяет достигать высокой степени гибкости в межпроцессорных связях.

6.8.2 Регистры интерфейса последовательного TDM порта

Последовательный TDM порт доступен через 6 картированных в памяти регистров и два других регистра (TRSR и TXSR), к которым нет прямого доступа в программе, но которые используются в реализации возможности двойной буферизации. Эти 8 регистров представлены в таблице 58.

Таблица 58 – Регистры последовательного TDM порта

Адрес	Регистр	Описание
0030h	TRCV	TDM регистр приема данных
0031h	TDXR	TDM регистр передачи данных
0032h	TSPC	TDM регистр управления последовательного порта
0033h	TCSR	TDM регистр выбора канала
0034h	TRTA	TDM регистр адреса приема/передачи
0035h	TRAD	TDM регистр адреса приема
-	TRSR	TDM сдвиговый регистр приема данных
-	TXSR	TDM сдвиговый регистр передачи данных

− TDM – регистр приема данных (TRCV). 16-битный TDM – регистр приема данных содержит входящие данные TDM – порта.

− TDM – регистр передачи данных (TDXR). 16-битный TDM – регистр передачи данных содержит передаваемые данные TDM – порта.

− Контрольный регистр последовательного TDM порта (TSPC). 16-битный TSPC содержит биты управления и состояния интерфейса последовательного TDM порта. Бит TDM конфигурирует порт в режиме TDM (TDM = 1) или в автономном режиме (TDM = 0). В автономном режиме порт работает как стандартный последовательный порт(SP).

− TDM – регистр выбора канала (TCSR). 16-битный TCSR определяет, в какой(какие) временной(ые) слот(ы), должен передавать каждый из процессоров 1867ВЦ4Т.

− TDM – регистр приема/передачи адреса (TRTA) определяет в восьми младших битах (RA0-RA7) адрес приема процессора 1867ВЦ4Т, а в восьми старших битах (TA0-TA7) – адрес передачи.

− TDM – регистр адреса приема данных (TRAD). 16-битный TRAD содержит различную информацию относительно состояния TDM адресной линии (TADD).

− Сдвиговый регистр приема данных TDM (TRSR). 16-битный TRSR позволяет хранение данных, поступающих со входа приема данных (TDR), и передачи этих данных в TRCV в последовательном порту (SP).

− Сдвиговый регистр передаваемых данных TDM (TXSR). 16-битный TXSR управляет перемещением выходных данных из TDXR и хранит данные, для передачи их на вывод TDX.

6.8.3 Операции последовательного TDM порта

Рисунок 81 показывает архитектуру TDM порта применительно к процессору 1867ВЦ4Т. На 4-разрядной последовательной шине могут располагаться до восьми устройств. Эта 4-разрядная последовательная шина состоит из трех шин последовательного порта: тактовых сигналов, кадровой синхронизации и данных (TCLK, TFRM, TDAT) плюс дополнительная шина (TADD), которая содержит адресную информацию устройства. Следует отметить, что TDAT и TADD – двунаправленные шины и часто управляются разными устройствами на шине в течение различных промежутков времени внутри данного кадра операции.

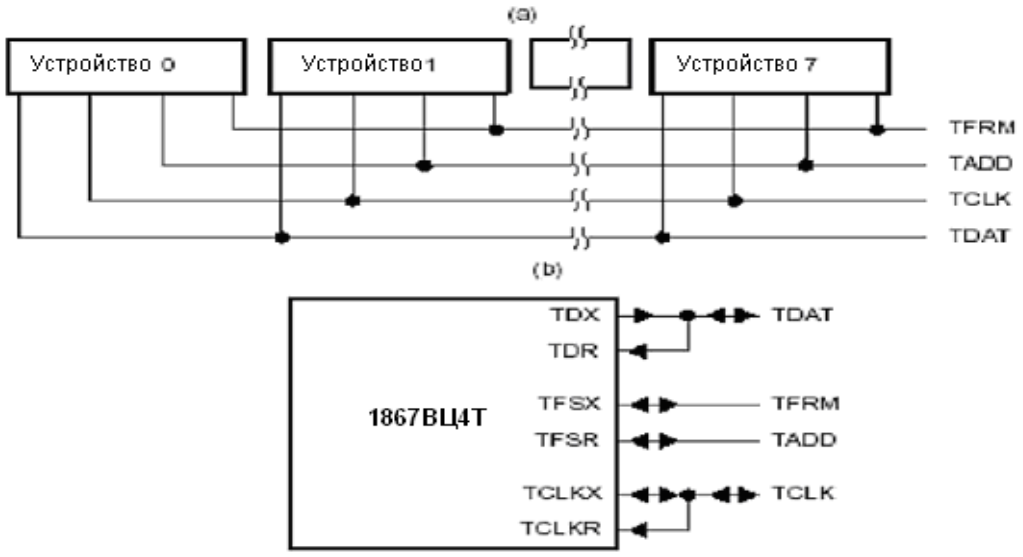


Рисунок 81 – 4-разрядная TDM шина

Линия TADD, которая управляется конкретным устройством, в конкретный момент времени определяет, какое устройство TDM конфигурации должно выполнять TDM-прием в течение этого промежутка (слота) времени. Это подобно допустимой операции чтения последовательного порта за исключением того, что некоторые соответствующие TDM регистры называются по-другому. TDM-регистр приема данных – это TRCV, сдвиговый TDM-регистр приема данных – это TRSR. Данные передаются через двунаправленную линию TDAT.

Обратите внимание, что на рисунке 81(b) выводы TDX и TDR объединены вместе для формирования линии TDAT. Кроме этого, только одно устройство может управлять линией данных и адреса (TDAT и TADD) в конкретном слоте. Выводы TDAT и TADD всех остальных устройств должны находиться в третьем состоянии в течение этого срока, что достигается соответствующим программированием регистра управления TDM порта. Между тем, в этом конкретном слоте все устройства (включая и то, которое управляет этим слотом) производят выборку линий TDAT и TADD, чтобы определить, представляет ли текущая передача достоверные данные, которые будут прочитаны одним из устройств, подключенным к шине. Когда устройство распознает адрес, который оно считает соответствующим допустимым, тогда и происходит процесс TDM – чтения и данные передаются из TRSR в TRCV регистр. Генерируется прерывание по приему (TRNT), которое показывает, что TRCV содержит достоверные принятые данные, и может быть прочитан.

Все операции TDM порта синхронизируются сигналами TCLK и TFRM. Каждый из них генерируется только одним устройством, называемым TCLK и TFRM источником. Следовательно, все другие устройства в TDM конфигурации используют эти сигналы как входы. На рисунке 81(b) выводы TCLKX и TCLKR внешне объединены вместе для формирования линии TCLK. TFRM и TADD образуются из выводов TFSX и TFSR соответственно. Это сделано для того, чтобы облегчить использование TDM порта в стандартном режиме.

Операции TDM порта управляются шестью картированными в памяти регистрами. Расположение этих регистров показано на рисунке 82. Регистры TRCV и TDXR выполняют такие же функции, что и регистры DRR и DXR соответственно. Регистр TSPC идентичен регистру SPC за исключением того, что нулевой бит служит для управления TDM режимом. Этот бит конфигурирует порт в TDM режим (TDM = 1) или в автономный режим (TDM = 0). В автономном режиме порт работает как стандартный последовательный порт.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRCV	Прием данных															
TDXR	Передача данных															
TSPC	Free	Soft	X	X	XRDY	RRDY	IN1	IN0	RRST#	XRST#	TXM	MCM	X	0	0	TDM
TCSR	X	X	X	X	X	X	X	X	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
TRTA	TA7	TA6	TA5	TA4	TA3	TA2	TA1	TA0	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
TRAD	X	X	X2	X1	X0	S2	S1	S0	A7	A6	A5	A4	A3	A2	A1	A0

Рисунок 82 – Диаграмма регистров последовательного TDM порта

При выборе TDM режима биты DLB и FO в TSPC аппаратно конфигурируются в 0, в результате чего становится недоступным режим цифровой обратной связи и фиксируется длина слова в 16 бит. Также значения FSM бита не влияют на работу порта, когда TDM = 1 и состояние флагов (биты RSRFULL, XRESEMPTY# и т. п.) не определены. Если TDM = 1, изменения, сделанные в содержимом TSPC, становятся эффективными при завершении

канала 7 текущего кадра. Таким образом, данные TSPC не могут быть изменены для текущего кадра, изменения вступят в силу в следующем кадре.

Устройство источника для синхронных сигналов TCLK и TFRM устанавливается посредством MCM и TXM битов соответственно. Устройство источника TCLK определяется установкой бита MCM TSPC регистра в 1. Вывод TCLKX конфигурируется как вход, если MCM = 0 и как выход, если MCM = 1. В последнем случае (внутренний тактовый сигнал процессора) устройство, у которого MCM = 1, обеспечивает тактовый сигнал (частота TCLK = ¼ частоты CLKOUT1) для всех устройств на шине TDM. Тактовый сигнал может быть подведен внешним источником, если MCM = 0 для всех устройств. TFRM может также быть внешним, если TXM = 0. Внешний TFRM должен соответствовать требованиям TDM синхронизации по отношению к TCLK для соответствующей операции. Не более чем одно устройство должно иметь MCM или TXM, установленными в 1 в произвольном периоде времени. Выбор устройства, которое будет являться источником тактовых сигналов и кадровой синхронизации, выполняется только один раз, во время системной инициализации.

TDM регистр выбора канала (TCSR) данного устройства определяет в каком временном слоте это устройство передает данные. 1 в одном или более битах 0-7 TCSR активизирует передатчик в течение соответствующего временного слота. Кроме того существует ограничение системного уровня: не более, чем одно устройство может совершать передачу во время одного и того же временного слота; устройства не проверяют, есть ли конфликтная ситуация при обращении к шине, слоты должны быть последовательно заданы. Как и в операции TSPC, запись в TCSR в течение текущего фрейма будет эффективна только в следующем фрейме. Данное устройство может совершать передачу более чем в один слот.

TDM регистр приема/передачи (TRTA) данного устройства определяет две ключевые части информации. Младшие 8 бит (RA7-RA0) – адрес приема устройства, старшие 8 бит (TA7-TA0) TRTA – адрес передачи (рисунок 82). Адрес приема имеет 8-битное значение, которое устройство сравнивает с 8-битным значением на линии TADD в каждом слоте. Адрес приема устанавливает слоты, в которых это устройство может принимать в зависимости от адреса, представленного в этих слотах, как определено передающими устройствами. Этот процесс происходит в каждом устройстве в течение каждого слота.

Адрес передачи (TA7-TA0) – это адрес, посредством которого устройство управляет линией TADD во время операции передачи в указанный слот. Адрес передачи указывает, какие приемные устройства могут осуществлять TDM прием данных.

Только одно устройство одновременно может управлять адресом передачи на TADD. Каждый процессор проводит побитную операцию логического суммирования своего адреса приема (RA7-RA0) со значением на линии TADD. Если эта операция приводит к значению, отличному от нуля, то выполняется достоверный TDM прием на процессоре или процессорах, чьи адреса совпадают с адресом передачи. Таким образом, при передаче одним устройством другим, один бит в верхней половине TRTA передающего устройства, имеющий значение "1", должен совпадать со значением бита в нижней половине TRTA (RA7-RA0) приемного устройства. Этот метод конфигурации TRTA позволяет одному устройству передавать данные другому устройству или устройствам, и любому одному устройству получать данные из одного или более устройств.

TDM регистр адреса приема (TRAD) содержит различную информацию о состоянии TADD линии, которая может опрашиваться для контроля предыдущих значений сигнала и взаимодействий между циклами команд и синхронизацией TDM порта.

Разряды 13-11 (X2-X0) содержат значение номера текущего слота, независимо от того был ли выполнен прием данных в этом слоте или нет. Эти данные фиксируются в начале слота и сохраняются до конца этого слота.

Разряды 8-10 (S2-S0) содержат номер последнего слота плюс 1 (по модулю), в котором данные были получены, например, если последнее считывание данных имело место в пятом слоте предыдущего фрейма, эти биты будут содержать число 6. Эти значения фиксируются в течение прерывания по TDM приему в конце слота, в котором были получены последние достоверные данные, и сохраняются до окончания следующего слота, в котором происходит достоверный прием данных.

Разряды A7-A0 хранят последний адрес, выбранный на TADD линии, независимо от того, выполнена ли операция приема достоверных данных или нет.

6.8.4 Операции передачи и приема данных в TDM режиме

На рисунке 83 представлена временная диаграмма TDM порта при передаче. Сигналы TCLK и TFRM формируются источником синхронизации. Частота TCLK равна ¼ частоты CLKOUT1, если генерируется процессором 1867ВЦ4Т. TFRM импульс формируется через каждые 128 циклов TCLK и синхронизирован для совпадения с 0 битом 7 слота, который является последним битом предыдущего фрейма. Взаимосвязь TFRM и TCLK позволяет управлять 16 битами данных для каждого из 8 временных слотов на TDAT линии, которая также дает возможность процессору выполнять максимум из 64 инструкций в течение каждого слота, предполагая, что используется внутренняя синхронизация процессора 1867ВЦ4Т. Начиная с 0 слота и считая MSB первым, передающее устройство управляет 16 битами данных для каждого слота, где каждый разряд имеет длительность, равную циклу TCLK, исключая первый бит каждого слота, длительность которого составляет половину продолжительности одного бита. Следует заметить, что данные на TDAT линии синхронизируются передающим устройством и выбираются из TDAT линии приемным устройством на переднем фронте TCLK.

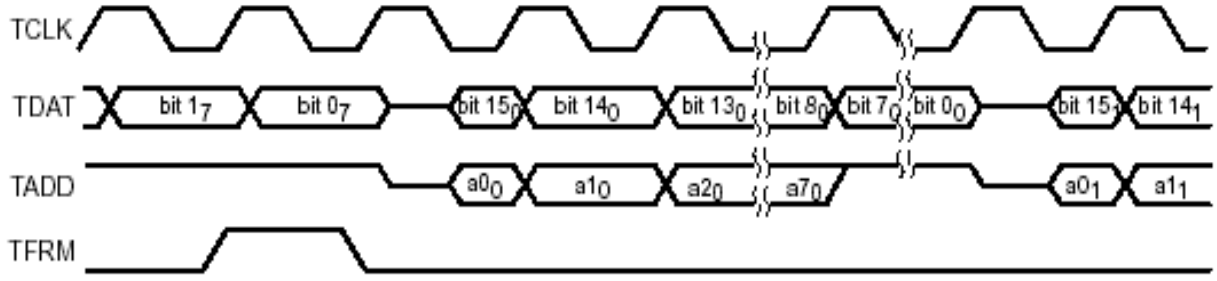


Рисунок 83 – Синхронизация последовательного порта (состояние TDM)

Одновременно с передачей данных передающее устройство так же управляет TADD линией с адресом передачи для каждого слота. Эта информация, в отличие от TDAT, имеет размер только в один байт и передается, считая LSB первым, для первой половины каждого слота. В течение второй половины слота (последние восемь периодов TCLK) TADD линия устанавливается в 1. Логика TDM приема выбирает TADD линию только в течении первых восьми периодов TCLK, игнорируя ее в течение второй половины слота. Поэтому передающее устройство (если это не 1867ВЦ4Т) может установить TADD в 1 или 0 в течении этого периода времени. Следует заметить, что, также как и в TDAT, первый бит TADD передается последним только для одной половины одного TCLK цикла.

Если нет устройства на шине TDM, сконфигурированного для передачи в слот (т. е. ни одно из устройств не имеет 1 для соответствующего слота в регистре TCSR), то слот считается пустым. В пустом слоте линии TADD и TDAT имеют третье состояние. Для исключения ложного чтения устройством 1 кОм резистор с подкачкой 0 должен быть связан с TADD линией. Это является причиной того, что TADD линия при чтении находится в 0 в пустом слоте. В противном случая, любой случайный сигнал на линии TADD, который возникает при сравнении конкретного адреса приема, будет вызывать

ложное чтение. Если рассеяние мощности учитывается и резистор не желателен, то произвольный процессор с адресом передачи, равным 0h, может управлять пустыми слотами записью в TDXR в этих слотах. Резистор в 1 кОм на TDAT линии не нужен.

Пустой слот TDM может образовываться в следующих случаях: первый случай, как уже упомянуто выше, имеет место, когда ни одно из устройств не имеет сконфигурированного TCSR для передачи в этом слоте. Второй случай: TDXR не загружен перед слотом передачи в отдельном фрейме. Это также может произойти, когда содержимое TCSR изменяется, но фактическое содержимое TCSR не обновляется, пока не появится следующий импульс TFRM. Поэтому любые последующие изменения будут эффективны только в следующем фрейме. Это справедливо для адреса приема (младшие 8 разрядов TRTA). Адрес передачи (старшие 8 разрядов TRTA) и TDXR могут быть изменены внутри текущего фрейма для отдельного слота, предполагая, что слот еще не был достигнут когда выполняется инструкция для загрузки TRTA или TDXR. Следует заметить, что для загрузки адреса передачи нет необходимости каждый раз загружать TDXR; когда произошла запись в TDXR и началась передача, текущий адрес передачи передается на TADD линию.

Номер текущего слота может быть получен при чтении X2-X0 битов регистра TRAD. Это позволяет сделать гибкой конфигурацию TDM порта в базисе "слот за слотом" и даже возможно совместное использование слотов. Ключом для использования этих возможностей является понимание временных соотношений между выполнением инструкций и фреймом/слотами TDM порта.

Если TDM порт будет переконфигурироваться от "слота к слоту", то изменения в соответствующих регистрах должны делаться достаточно быстро, чтобы желаемый эффект достигался в желаемое время.

Необходимо принять во внимание, что изменение регистра TCSR и адреса приема (младшая половина TRTA) произойдет только после начала нового фрейма. В то время, как изменение адреса передачи (старшая половина TRTA) и TDXR (передаваемые данные) пройдут в начале следующего слота, как было сказано выше.

6.8.5 Условия исключения интерфейса последовательного TDM порта

В связи с особенностями TDM порта, когда один процессор может передавать в различных слотах, понятия переполнение и недозагрузка становятся неактуальными. Поэтому флаги переполнения и недозагрузки не активны в TDM режиме. В приемном TDM порту, если TRCV не был прочитан и порт готовится принять еще одно слово (когда адрес, передаваемый на линии TADD, совпал с адресом приема в регистре TRTA), то в этом случае перезаписывается то значение, которое в нем было; приемный порт не останавливается. С другой стороны, если TDXR не был обновлен перед передачей, то линии TADD или TDAT не управляются и эти выводы остаются в третьем состоянии. Этот режим предотвращает возможность ложной передачи.

Если импульс TFRM возникает не вовремя в течение фрейма, то TDM порт не способен продолжать правильно работать и номера битов и слов становятся неопределенными, когда это случается. Только один импульс TFRM должен возникнуть каждые 128 тактов TCLK. В отличие от последовательного порта, TDM порт не может переинициализироваться импульсом синхронизации фрейма в процессе передачи. Для корректировки не правильно поданного импульса TFRM TDM порт должен быть сброшен.

6.8.6 Примеры работы интерфейса TDM порта

В этом примере восемь устройств соединены между собой через TDM порты, как показано на рисунке 84.

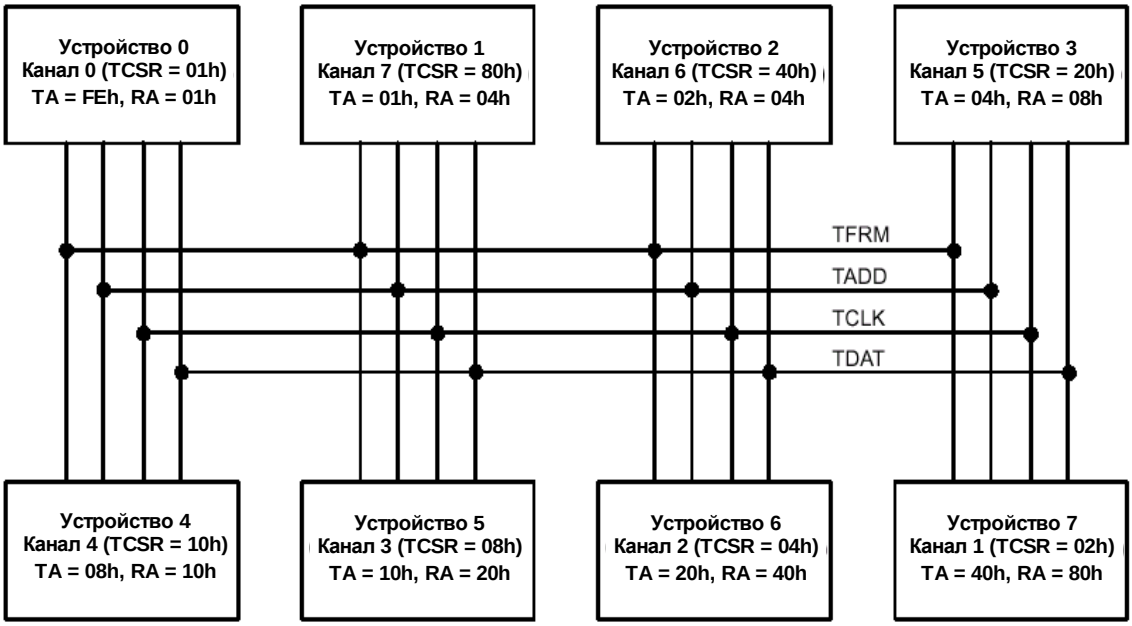


Рисунок 84 – Функциональная схема сети из восьми устройств, связанных с помощью TDM порта

Таблица 60 показывает значение TADD для каждого из восьми каналов и обозначение передающих и приемных устройств. Таблица 61 показывает конфигурацию соединения устройств друг с другом. В этом примере устройство 0 передает другим семи устройствам адреса в течение слота 0. В последующих фреймах каждое из устройств 1-7 соединяются с другим устройством.

Таблица 60 – Сценарий межпроцессорных связей

Каналы	Данные TADD	Передающее устройство	Приемные устройства
0	0FFh	0	1-7
1	40h	7	6
2	20h	6	5
3	10h	5	4
4	08h	4	3
1	02h	3	4
5	04h	3	2
6	02h	2	1
7	01h	1	0

Таблица 61 – Содержание регистров TDM порта

Устройства	TSPC	TRTA	TCSR
0	xxF9h	0FE01h	xx01h
1	xxC9h	0102h	xx80h
2	xxC9h	0204h	xx40h
3	xxC9h	0408h	xx20h
4	xxC9h	0810h	xx10h
5	xxC9h	1020h	xx08h
6	xxC9h	2040h	xx04h
7	xxC9h	4080h	xx02h

В этом примере адреса передачи передающих устройств (старшие биты TRTA) сравниваются с адресами приема (младшие биты TRTA) приемных устройств. Следует заметить, что не обязательно полное совпадение адресов приема и передачи; операция сравнения в приемном порту представлена как побитная операция AND. Поэтому необходимо, чтобы совпал только один бит из сравниваемых адресов.

Устройство также может передавать сама себя, т. к. обе операции приема и передачи происходят по переднему фронту TCLK. Для того чтобы обеспечить этот режим необходимо использовать стандартное соединение интерфейса TDM порта, как показано на рисунке 81. Если в регистре TRTA устройства 0 записано 0101h, то оно будет передавать только само на себя.

7 Операции внешней шины

7.1 Интерфейс внешней шины

Внешний интерфейс 1867ВЦ4Т состоит из шин данных, адреса и набора сигналов управления для доступа к внешней памяти и портам ввода/вывода. В таблице 62 перечислены сигналы внешнего интерфейса.

Таблица 63 – Сигналы внешнего интерфейса

Наименование вывода	Описание
A0-A15	Шина адреса
D0-D15	Шина данных
MSTRB#	Строб доступа к внешней памяти
PS#	Сигнал выбора области программ
DS#	Сигнал выбора области данных
IOSTRB#	Строб доступа к устройствам ввода/вывода
IS#	Сигнал выбора области ввода/вывода
R/W#	Сигнал чтение/запись
READY	Сигнал готовности данных
HOLD#	Сигнал состояния удержания
HOLDA#	Подтверждение состояния удержания
MSC#	Микросостояние завершено
IAQ#	Выполнение команды
IACK#	Подтверждение прерывания

Параллельный интерфейс состоит из двух взаимно - исключающих интерфейсов, управляемых сигналами IOSTRB# и MSTRB#. Сигнал MSTRB# используется для доступа к памяти программ или данных, а IOSTRB# используется для обращения к портам ввода/вывода. Сигнал R/W# управляет направлением доступа.

Внешний входной сигнал готовности (READY) и программируемые состояния ожидания позволяют процессору обращаться к внешней памяти или устройствам ввода-вывода с различными скоростями обмена. При взаимодействии с низкоскоростными устройствами процессор ждет, пока внешнее устройство не выставит на контакт READY сигнал готовности, свидетельствующий об окончании цикла обмена.

Иногда увеличение времени обращения необходимо, когда обмен осуществляется между устройствами внешней памяти. В этом случае дополнительные циклы ожидания автоматически вставляются программируемой логикой переключения банков.

Режим удержания (hold) позволяет внешнему устройству взять под свой контроль внешние шины 1867ВЦ4Т для доступа к ресурсам процессора. Возможны два режима хранения: нормальный режим и режим прямого доступа к памяти. Когда процессор обращается к внутренней памяти, шина данных устанавливается в высокоимпедансное состояние, а шина адреса и сигналы выбора областей памяти (выбор программ (PS#), выбор данных (DS#) и выбор ввода-вывода (IS#)) сохраняют предыдущее состояние. Сигналы MSTRB#, IOSTRB#, R/W#, IAQ# и MSC# находятся в пассивном состоянии. Если включен режим видимости адреса (бит AVIS, размещенный в регистре PMST, установлен в 1), внутренний адрес программы выставляется на шину адреса в сопровождении сигнала IAQ#.

7.2 Приоритет внешней шины

1867ВЦ4Т имеет одну шину программы (PB), три шины данных (CB, DB и EB), и четыре шины адреса (PAB, CAB, DAB и EAB). Такая архитектура в сочетании с конвейером позволяет одновременный доступ к этим шинам. Конфликт в конвейере происходит тогда, когда в одном цикле CPU обращается к внешней памяти дважды при выборке команды, операнда памяти данных или к внешним устройствам ввода/вывода. Этот конфликт в конвейере автоматически решается заданным приоритетом, определяющим фазы конвейера.

Рисунок 85 показывает множественный доступ CPU для выборки команды, операндов для записи и чтения данных за один цикл. Доступ к данным имеет более высокий приоритет, чем выборка памяти программ: выборка памяти программ не может начинаться, пока не завершатся все обмены данными.

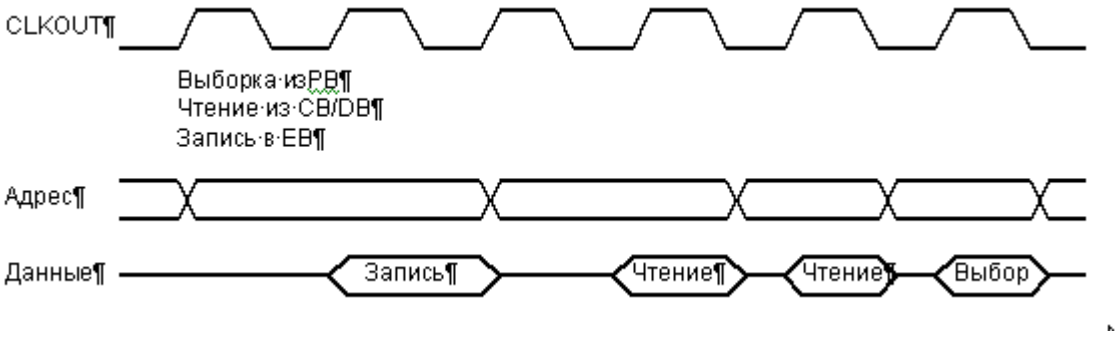


Рисунок 85 – Приоритет интерфейса внешней шины

Конфликт в конвейере происходит, когда программа и данные находятся во внешней памяти, и когда команда записи одного операнда следует за чтением двух операндов или чтением 32-разрядного операнда. Следующая последовательность команд вызывает такой конфликт в конвейере.

```
ST      T,*AR6          ;операция записи Smem
LD      *AR4+,A         ;операция чтения Xmem и Ymem
MAC     *AR5+,B         ;
```

7.3 Управление внешней шины

Два устройства 1867ВЦ4Т управляют внешней шиной: генератор состояний ожидания и логика, переключения банков внешней памяти. Эти устройства управляются двумя регистрами: программируемым регистром состояний ожидания (SWWSR) и управляющим регистром переключения банков (BSCR).

7.3.1 Генератор циклов ожидания

Программируемый генератор состояний ожидания может удлинять циклы внешней шины до семи машинных тактов, что позволяет 1867ВЦ4Т обмениваться с медленными внешними устройствами. Устройства, требующие больше семи циклов, должны использовать аппаратную задержку с помощью аппаратной линии READY. Когда запрограммировано нулевое время ожидания, внутренний генератор состояний ожидания отключен, что позволяет процессору работать с более низкой потребляемой мощностью.

Программируемый генератор состояний ожидания управляется 16-разрядным регистром состояния ожидания (SWWSR), отображенным в памяти данных по адресу 0028h.

Области программы и данных состоят каждая из двух 32K-словных блоков; область ввода/вывода состоит из одного 64K-словного блока. Каждый из этих блоков представлен соответствующим 3-разрядным полем в регистре SWWSR. Организация этого регистра представлена на рисунке 86, а битные поля описаны в таблице 64.

Значение 3-разрядного поля в SWWSR определяет число состояний ожидания, которые будут вставлены для каждого обмена с соответствующей областью в указанном адресном интервале. Минимальное значение вставляемого интервала равно нулю (000b). Значение 7 (111b) обеспечивает максимальное число состояний ожидания.

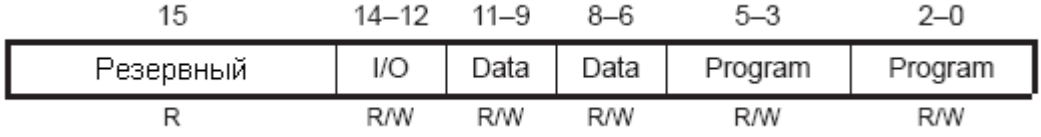


Рисунок 86 – Организация программируемого регистра состояний ожидания SWWSR

Таблица 64 – Описание бит в программируемом регистре состояний ожидания (SWWSR)

Бит	Имя	Значение после сброса	Функция
15	Резервный	0	Резервный.
14-12	Ввод/вывод	1	Пространство ввода/вывода. Значение этого поля (0-7) определяет количество циклов ожидания при обращении к пространству ввода/вывода в диапазоне адресов 0000-FFFFh
11-9	Данные	1	Пространство данных. Значение этого поля (0-7) определяет количество циклов ожидания при обращении к пространству данных в диапазоне адресов 8000-FFFFh
8-6	Данные	1	Пространство данных. Значение этого поля (0-7) определяет количество циклов ожидания при обращении к пространству данных в диапазоне адресов 0000-7FFFh

Окончание таблицы 64

Бит	Имя	Значение после сброса	Функция
5-3	Программы	1	Пространство программы. Значение этого поля (0-7) определяет количество циклов ожидания при обращении к пространству программ в диапазоне адресов 8000-FFFFh
2-0	Программы	1	Пространство программы. Значение этого поля (0-7) определяет количество циклов ожидания при обращении к пространству программ в диапазоне адресов 0000-7FFFh

На рисунке 87 представлена блок-схема генератора состояний ожидания, когда декодируется команда с внешним доступом к программе, соответствующее поле SWWSR загружается в счетчик. Если значение поля отлично от нуля, CPU посылается сигнал "не готов" и включается счетчик состояний ожидания. Этот сигнал поддерживается до тех пор, пока счетчик не обнулится и внешняя линия READY не установится в состояние с высоким уровнем. Внешний сигнал READY и READY состояний ожидания совместно через аппаратное "ИЛИ" генерируют сигнал CPU WAIT#. Линия READY выбирается по заднему фронту CLKOUT. Процессор определяет READY только тогда, когда программируется не менее двух состояний ожидания. Внешняя линия READY не анализируется до начала последнего цикла состояния ожидания.

При сбросе все поля в SWWSR устанавливаются в 111b (SWWSR = 7FFFh) – максимальное число состояний ожидания для внешнего доступа. Это гарантирует надежность обмена CPU с медленными внешними устройствами в процессе инициализации процессора.

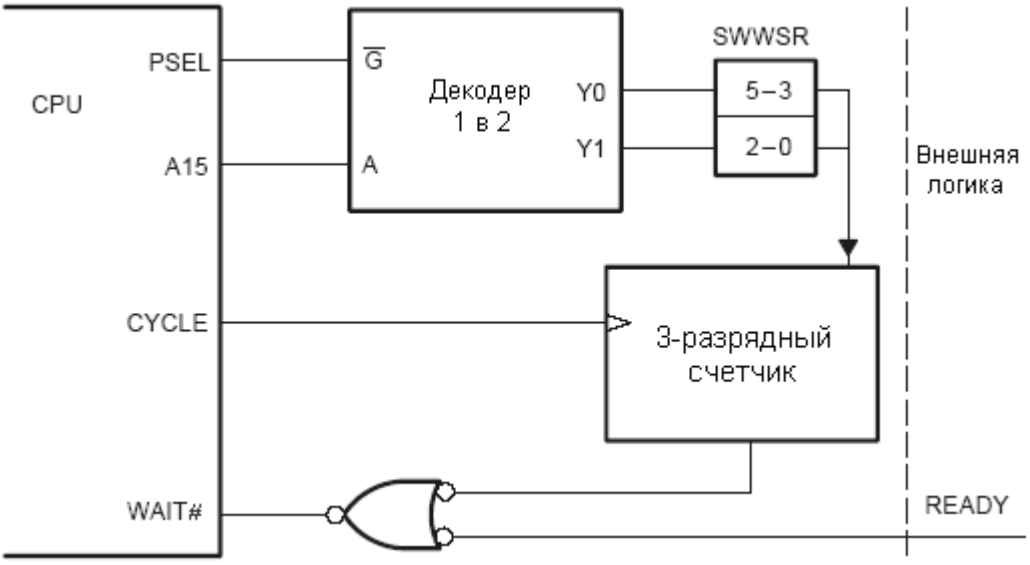


Рисунок 87 – Блок-схема генератора состояний ожидания

7.3.2 Логика переключения банков

Программируемая логика переключения банков позволяет 1867ВЦ4Т переключаться между банками внешней памяти без внешних состояний ожидания. Логика переключения банков автоматически вставляет один цикл при пересечении границ банков памяти внутри областей памяти программ или данных.

Управление переключением банков осуществляется с помощью регистра BSCR, отображенного в памяти данных по адресу 0029h. На рисунке 88 представлена организация регистра BSCR, а его поля описаны в таблице 65.

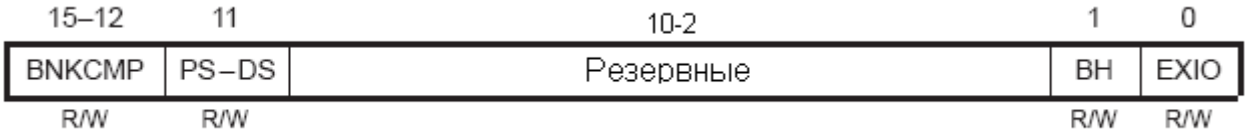


Рисунок 88 – Организация регистра переключения банков BSCR

Таблица 65 – Распределение бит в регистре переключения банков (BSCR)

Бит	Имя	Значение после сброса	Функция
15-12	BNKCMP	-	Размер банка. Определяет размер внешнего банка памяти. Поле BNKCMP маскирует четыре старших разряда адреса. Например, значение BNKCMP = 1111b определяет банк объемом в (4К×16) бит. Допускаются размеры от (4К×16) бит до (64К×16) бит. В таблице 66 показана связь между значением BNKCMP и диапазоном адресов
11	PS–DS	-	Доступ чтение программ – чтение данных. Вставляет дополнительный цикл ожидания между последовательными доступами чтения программ и чтения данных или чтения данных и чтения программы. PS-DS = 0 – дополнительные циклы ожидания не вставляются, пока границы банков не пересекаются. PS-DS = 1 – вставляется один дополнительный цикл между последовательными доступами к областям программы и данных или данных и программы
10-2	Резерв	-	Резервные биты
1	BH	0	Удержание шины: BH = 0 – удержание шины запрещено. BH = 1 – удержание шины разрешено. Шина данных D (15-0) удерживается в предыдущем состоянии
0	EXIO	0	Управление интерфейсом внешней шины. EXIO = 0 – функция отключения внешней шины запрещена. EXIO = 1 – функция отключения внешней шины разрешена. Шины адреса, данных и сигналы управления переходят в неактивное состояние по завершению текущего цикла шины. Таблица 67 содержит список сигналов и их состояние в режиме блокировки шины. Биты DROM MP/MC# и OVLY в регистре PMST, а также бит HM в регистре ST1 не могут изменяться

Таблица 66 – Соотношение между значениями BNKCMP и размерами банков

BNKCMP				Сравниваемые старшие разряды	Размер банка (16-разрядных слов)
Бит 15	Бит 14	Бит 13	Бит 12		
0	0	0	0	Нет	64K
1	0	0	0	15	32K
1	1	0	0	15-14	16K
1	1	1	0	15-13	8K
1	1	1	1	15-12	4K

Таблица 67 – Состояние сигналов при EXIO = 1

Сигналы	Состояние
A(15- 0)	Предыдущее состояние
D(15-0)	3-е состояние
DS#, PS#, IS#	Высокий уровень
MSTRB#, IOSTRB#	Высокий уровень
R/W#	Высокий уровень
MSC#	Высокий уровень
IAQ#	Высокий уровень

Биты EXIO и BH управляют доступом к внешним шинам адреса и данных. В нормальном режиме эти биты должны быть установлены в 0. Однако, для того, чтобы понизить потребляемую мощность, или в случае редкого использования внешней памяти, биты EXIO и BH могут быть установлены в 1.

Если выполняется повторный доступ по чтению к тому же самому банку памяти, дополнительные циклы не вставляются. Когда чтение выполняется из различных банков памяти, конфликтов памяти избегают вставкой дополнительного цикла. Дополнительный цикл вставляется только тогда, когда память читается два раза подряд. Отменить дополнительные циклы можно очисткой поля BNKCMP.

Механизм переключения банков автоматически вставляет один дополнительный цикл в следующих случаях:

– Чтение памяти данных из различных банков памяти.

– Чтение памяти программ из различных банков памяти.

– Чтение памяти программ после чтения памяти данных, когда бит PS-DS установлен в 1.

– Чтение памяти данных после чтения памяти программ, когда бит PS-DS установлен в 1.

На рисунке 89 представлена диаграмма переключения банков между циклами чтения памяти.

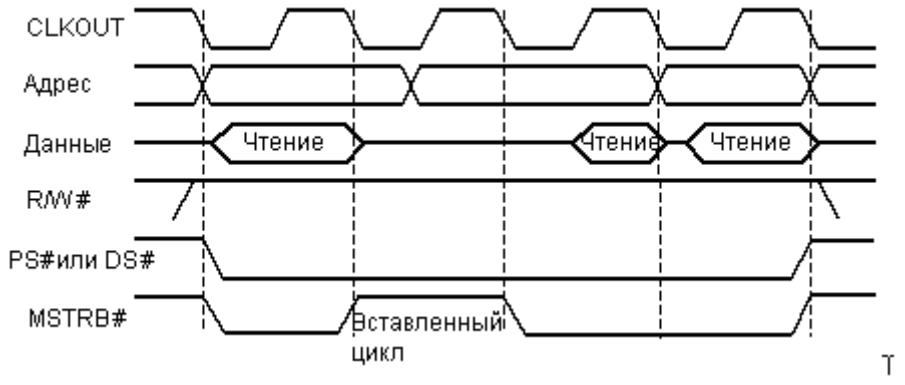


Рисунок 89 – Переключение банков между циклами чтения памяти

На рисунке 90 представлена диаграмма переключения банков между областью памяти программ и областью памяти данных.

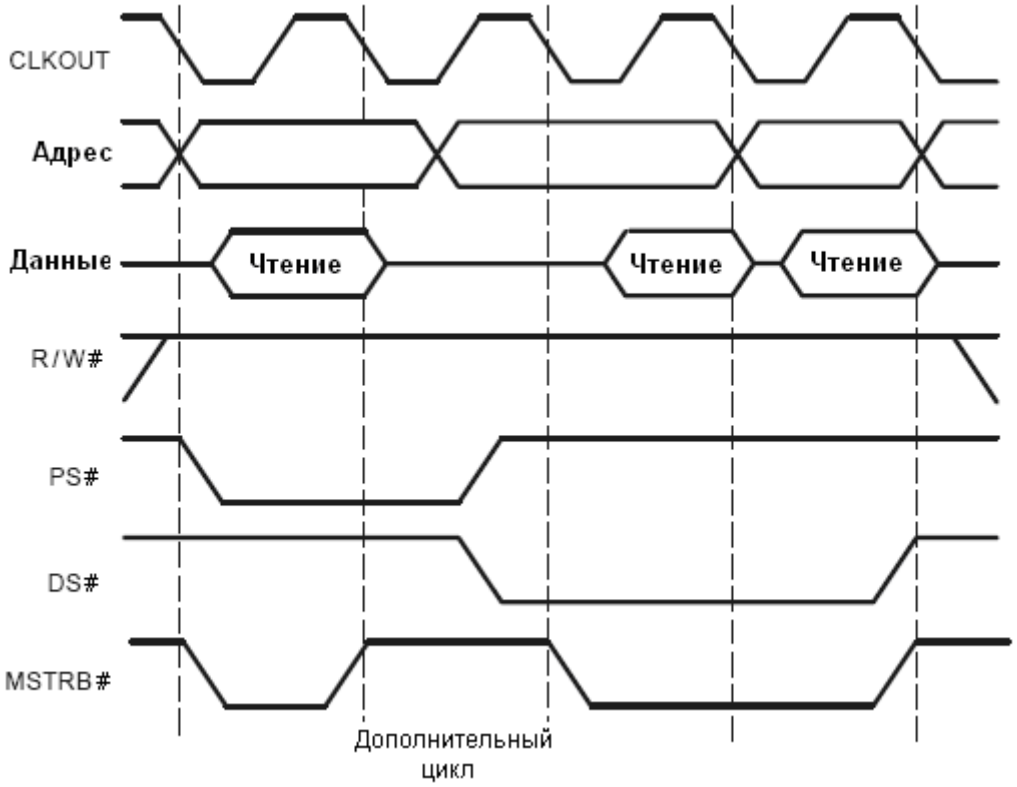


Рисунок 90 – Переключение банков между областью памяти программ и областью памяти данных

7.4 Временные характеристики внешней шины

Временной интервал обмена по внешней шине кратен целому числу циклов CLKOUT. Один цикл CLKOUT определен как период от одного заднего фронта до следующего заднего фронта сигнала CLKOUT. Некоторые внешние обращения к шине без состояний ожидания, например, запись в память или чтение и запись в порты ввода/вывода занимают два цикла. В случае, когда чтение памяти следует непосредственно за записью в нее или наоборот, чтение памяти занимает дополнительно половину цикла. Следующие подразделы обсуждают обращения с нулевым состоянием ожидания, если иначе не оговорено.

7.4.1 Временные характеристики доступа к памяти

В операциях чтения и записи сигнал MSTRB# сбрасывается в 0 для активной фазы обмена и удерживается в этом состоянии, по крайней мере, один такт сигнала CLKOUT. Изменения цикла CLKOUT происходят до и после активной части цикла записи.

В течение этого изменения:

– MSTRB# высокий.

– R/W# изменяется по переднему фронту CLKOUT, когда требуется.

– Адрес изменяется по переднему фронту CLKOUT в случаях, перечисленных ниже. Во всех других случаях адрес изменяется по заднему фронту CLKOUT:

— в предыдущем цикле CLKOUT была активизирована запись в память;

— чтение из памяти следует за записью в память;

— чтение из памяти следует за записью в порт ввода/вывода;
— чтение из памяти следует за чтением из порта ввода/вывода.

– DS#, PS#, IS# изменяются по необходимости синхронно с изменение адреса.

Рисунок 91 представляет последовательность чтение-чтение-запись без состояний ожидания. Данные читаются с шины в конце цикла чтения, когда достоверность их максимальна. Хотя процесс записи занимает два цикла, процессору нужен только один такт, если не требуется новых обращений к внешнему интерфейсу. Это помогает поддерживать максимальную производительность обработки.

Временная диаграмма иллюстрирует эти концепции:

– Взаимное чтение из одного банка при одноцикловом доступе.

– MSTRB# пребывает в состоянии с низким уровнем в течение взаимного чтения.

– MSTRB# переходит в состояние с высоким уровнем для одного цикла в течение процедуры чтение-запись вместе с изменением адреса и сигнала R/W#.

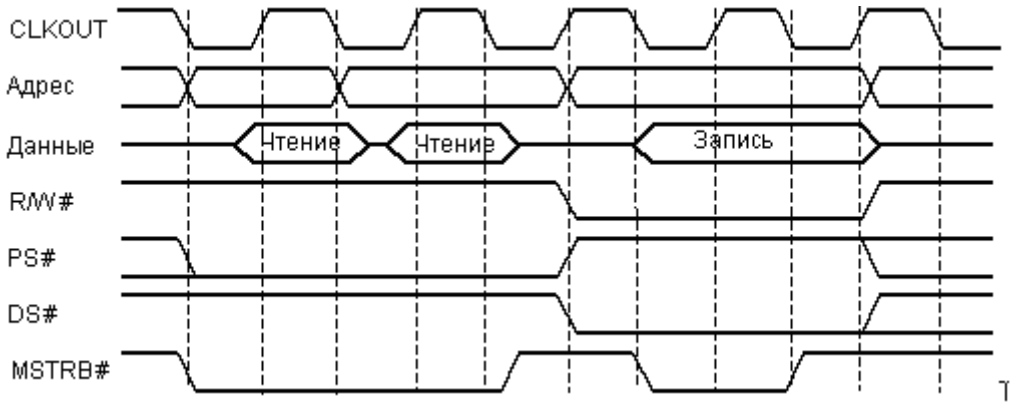


Рисунок 91 – Временные диаграммы операции чтение-чтение-запись

Рисунок 92 демонстрирует последовательность запись-запись-чтение с активным сигналом MSTRB# и с нулевым временем ожидания. Адрес и сохраняемые данные задержаны приблизительно на полцикла относительно переключения сигнала MSTRB#.

Диаграмма синхронизации иллюстрирует эту концепцию:

– MSTRB# переключается в состояние с высоким уровнем в конце каждого цикла записи для запрета доступа к памяти, в то время как адрес и/или R/W# изменяются.

– Каждая запись занимает два цикла.

– Чтение после записи длится два цикла.

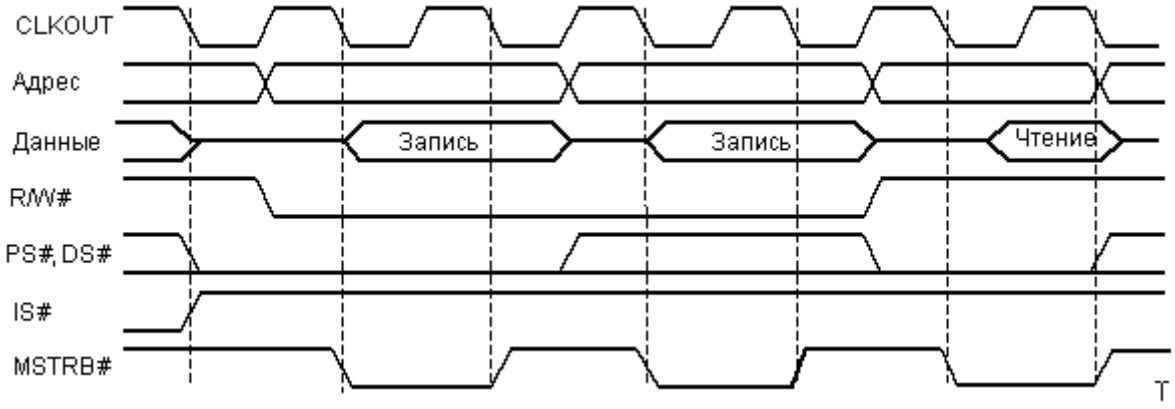


Рисунок 92 – Временные диаграммы операции запись-запись-чтение

Рисунок 93 демонстрирует последовательность чтение-чтение-запись, использующую MSTRB# и одно состояние ожидания. Процесс чтения расширен на один дополнительный цикл в соответствии с режимом состояния ожидания. Однако, процесс записи, занимает два цикла, расширен на три цикла.

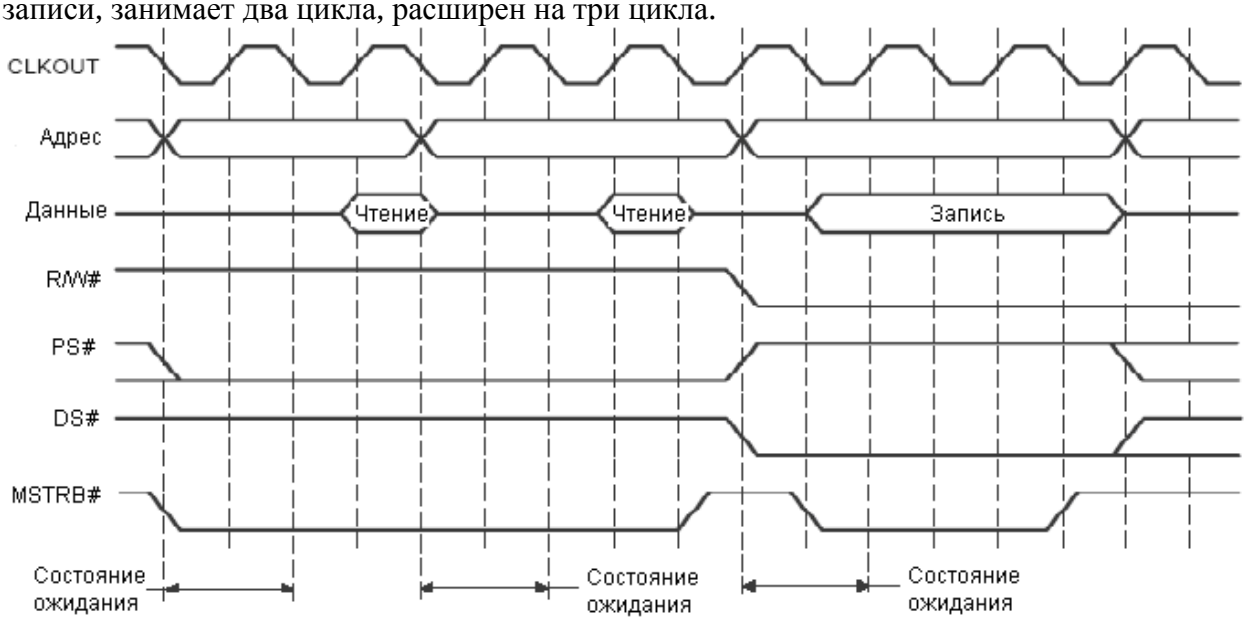


Рисунок 93 – Временные диаграммы внешней операции чтение-чтение-запись (для области программы включено состояние ожидания)

7.4.2 Временные характеристики доступа к пространству ввода/вывода (I/O)

При доступе в область ввода/вывода (I/O) активная часть чтения и записи занимает два цикла (без состояний ожидания) и временные характеристики для этих доступов такие же, как и для доступа к памяти. В течение этих циклов адрес изменяется по заднему фронту CLKOUT за исключением последовательности доступ к памяти – доступ к I/O. Сигнал IOSTRB# переходит на низкий уровень от одного положительного фронта до следующего положительного фронта CLKOUT.

На рисунке 94 показана последовательность операций чтение-запись-чтение для IOSTRB# без состояний ожидания. Для доступа IOSTRB# в режимах чтения и записи требуется минимум два цикла. Для чтения и записи с активным IOSTRB# последний должен быть полностью синхронизирован с адресом, чтобы удовлетворить этому требованию.

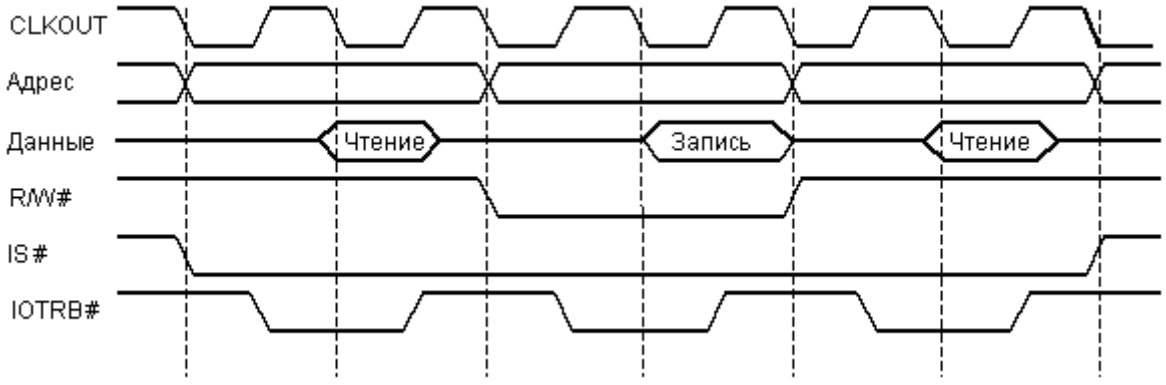


Рисунок 94 – Временная диаграмма операции для чтения-записи-чтения для интерфейса I/O

Рисунок 95 демонстрирует тот же самый доступ к области ввода/вывода, но с одним состоянием ожидания. Каждый доступ чтения и записи расширены на дополнительный цикл.

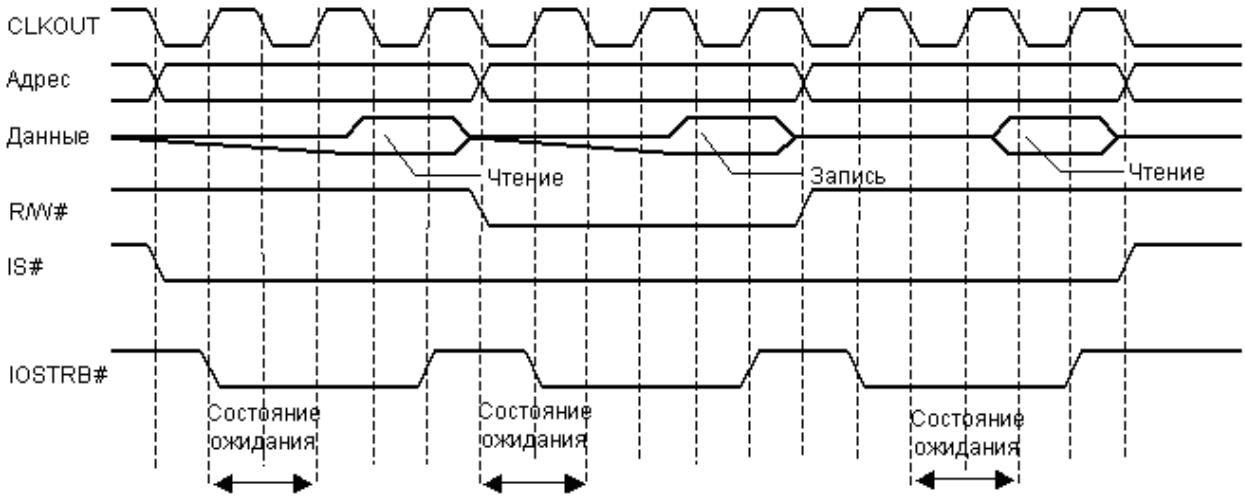


Рисунок 94 – Временная диаграмма операции для чтения-записи-чтения для интерфейса I/O (одно состояние ожидания)

7.4.3 Временные характеристики доступа к памяти и пространству ввода/вывода (I/O)

Рисунки с 95 по 102 демонстрируют различные варианты обмена между внешней памятью и областью ввода/вывода. Особенности такого обмена приведены ниже:

– Чтение и запись в области ввода/вывода занимают, по крайней мере, три такта, когда они следуют за доступом к внешней памяти.

– Чтение памяти занимает два такта, когда оно следует за доступом к области ввода/вывода.

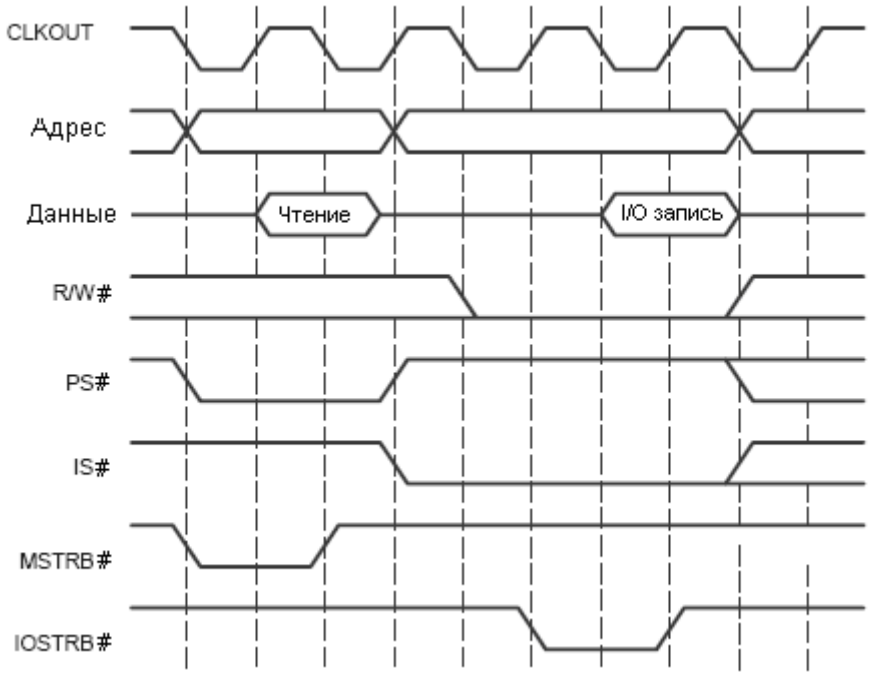


Рисунок 95 – Чтение из памяти и запись в порт I/O

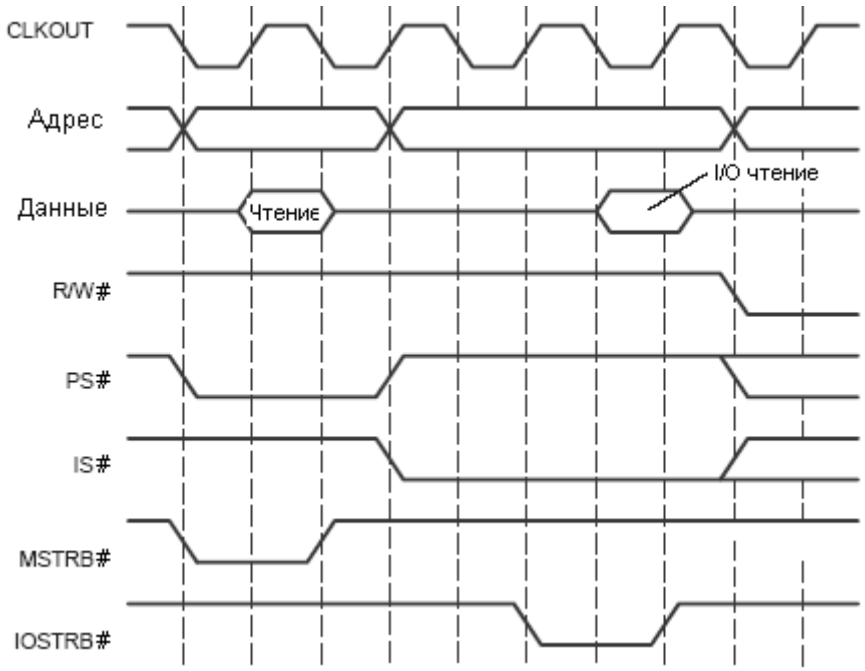


Рисунок 96 – Чтение из памяти и чтение из порта I/O

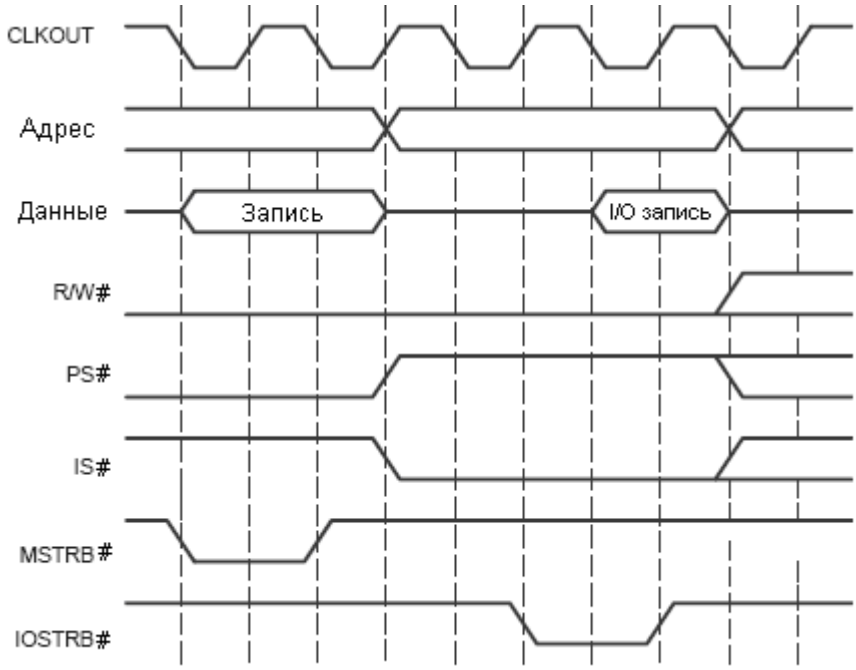


Рисунок 97 – Запись в память и запись в порт I/O

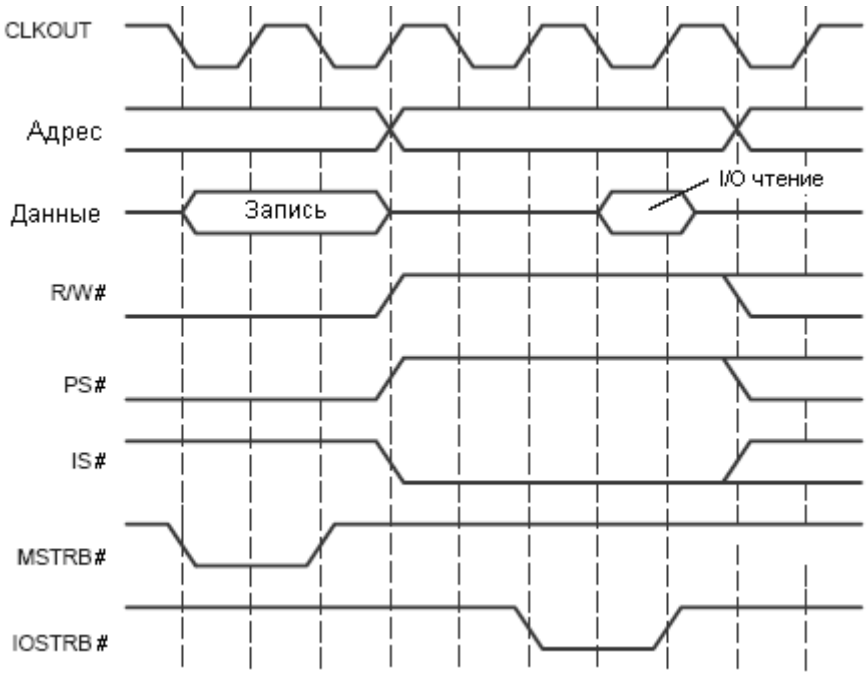


Рисунок 98 – Запись в память и чтение из порта I/O

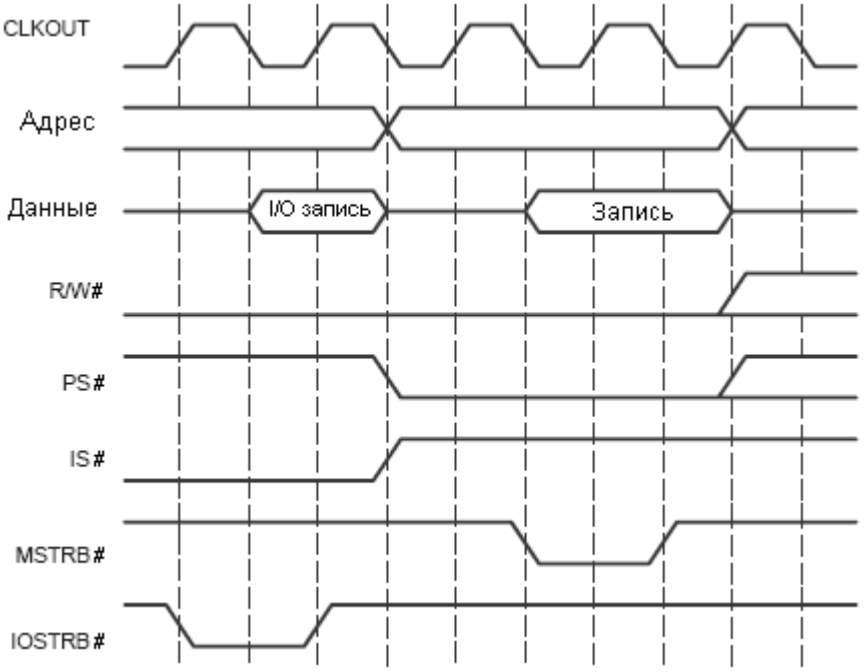


Рисунок 99 – Запись в порт I/O и запись в память

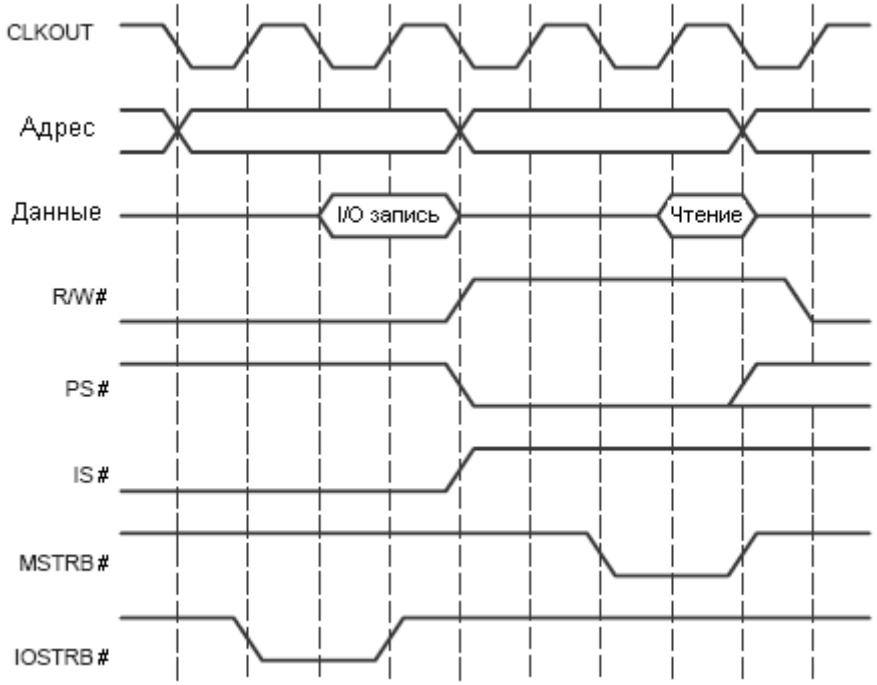


Рисунок 100 – Запись в порт I/O и чтение из памяти

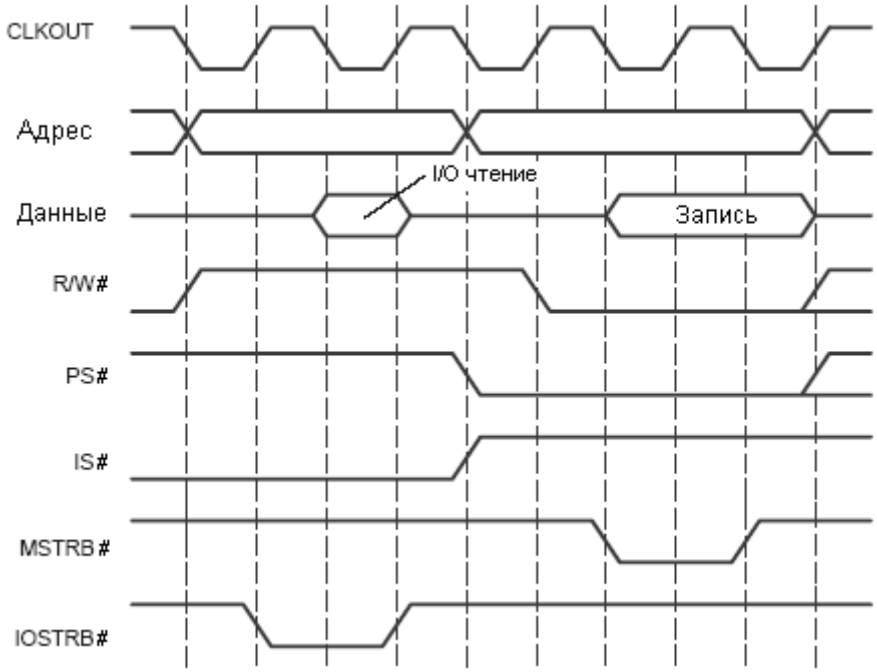


Рисунок 101 – Чтение из порта I/O и запись в память

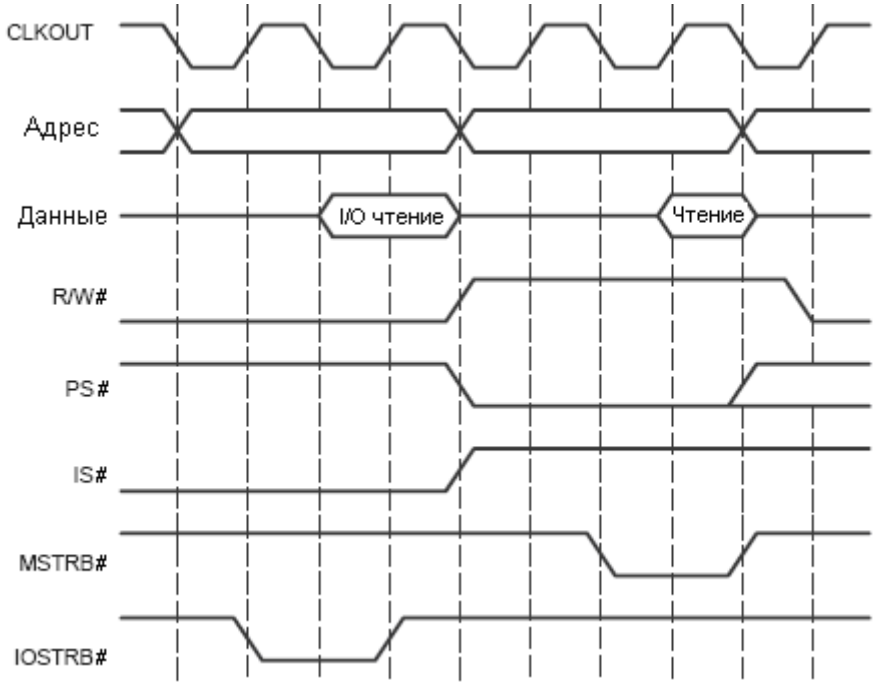


Рисунок 102 – Чтение из порта I/O и чтение из памяти

7.5 Последовательность доступа во время старта

Вход и выход из двух режимов пониженного энергопотребления IDLE1 или IDLE2 не требуют специального анализа, т. к. тактирование CPU и внутрикристальной периферии остается активным. Однако, специальный анализ необходим при двух режимах: сброс и IDLE3.

7.5.1 Сброс

Рисунок 103 иллюстрирует последовательность сброса для внешней шины. Для корректной операции сброса сигнал RS# должен быть активен, по крайней мере, два цикла CLKOUT. Но при включении питания и в режиме IDLE3 требуются, чтобы сигнал сброса был активен больше, чем два цикла CLKOUT.

Когда процессор подтверждает сброс, CPU останавливает выполнение программ и устанавливает программный счетчик в FF80h. На адресной шине выставляется FF80h, пока RS# находится в 0.

При входе в состояние сброса:

1) Через четыре цикла (после того, как сигнал RS# стал в 0) уровни сигналов PS#, IAQ# и MSTRB# переходят в низкое состояние.

2) Через пять циклов (RS# = 0), R/W# становится в высокий уровень, шина данных переходит в высокоимпедансное состояние и на адресной шине выбирается FF80h.

3) Устройство переходит в состояние сброса.

Когда сброс становится неактивным, выполнение программ начинается с адреса FF80h. Сигналы IACK# и IAQ# становятся активными (рисунок 103) независимо от состояния сигнала MP/MC#.

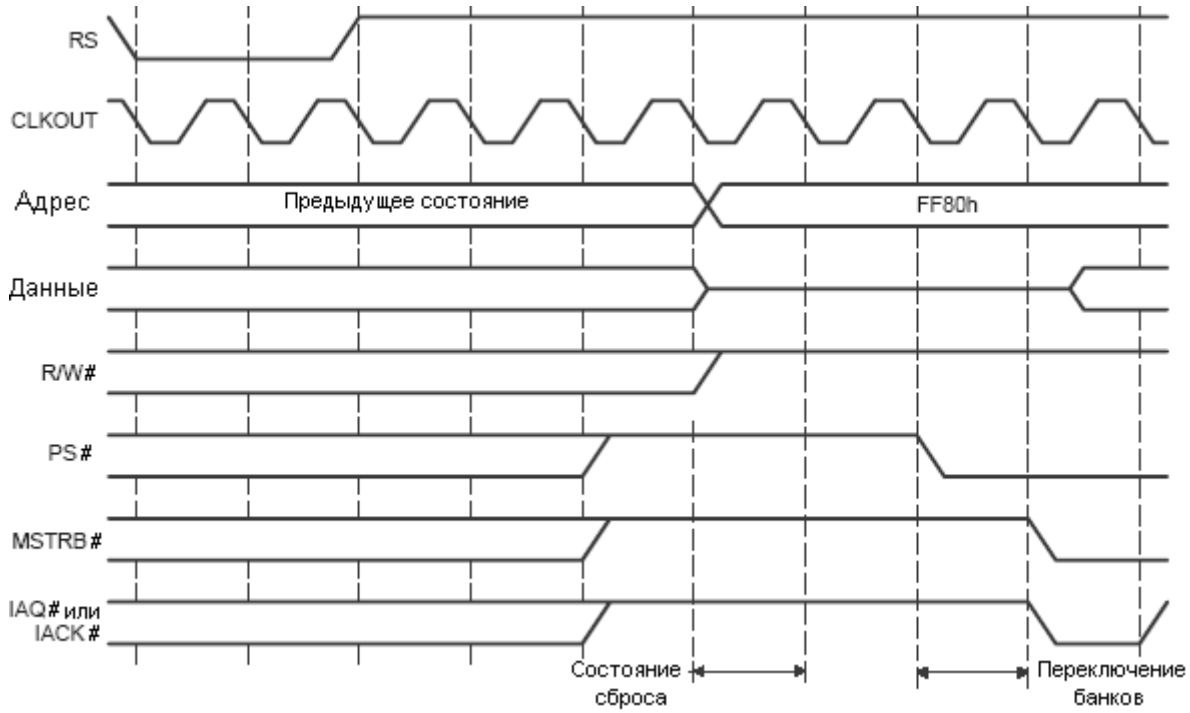


Рисунок 103 – Последовательность сброса внешней шины

7.5.2 IDLE3

Выполнение инструкции IDLE3 запускает режим выключения питания. В этом режиме PLL полностью останавливается, чтобы уменьшить потребляемую мощность. В IDLE режиме на вход может поступать тактовый сигнал без дополнительного питания, потому что передающий элемент микросхемы 1867ВЦ4Т выделяет тактовый сигнал из внутренней логики. PLL должен быть перезапущен и блокирован перед продолжением обработки данных, при включенном IDLE3. Этот режим выключения питания отключается при активации внешних сигналов прерываний INTn#, NMI# и RS#.

В таблице 68 показано время запуска режима IDLE3 с сигналами INTn# и NMI#. Это время описано для аппаратно-конфигурируемого PLL. Когда сигнал прерывания перестает действовать, внутренний счетчик считает циклы входного тактового импульса. Начальное значение, загружаемое в счетчик, зависит от коэффициента умножения PLL, чтобы гарантировать время выключения счетчика больше, чем 50 мс.

Таблица 68 – Время выключения счетчика при различных коэффициентах умножения PLL при операциях на частоте 40 МГц

Начальное значение счетчика	Коэффициент умножения PLL	Циклы CLKOUT	Время выключения счетчика, мс
2048	1	2048	51,2
2048	1,5	3072	76,8
1024	2	2048	51,2
1024	3	3072	76,8

Время выключения счетчика в таблице 68 действительно, когда частота входного тактового импульса равна 40 МГц при блокированном PLL. После того как счетчик досчитает до 0, выход закрытого PLL питает внутреннюю логику.

Низкий сигнал (минимальная длительность 10 нс) внешнего прерывания приводит к запуску 1867ВЦ4Т от IDLE (см. рисунок 104). Тактовый сигнал блокированного PLL

соединяется с CPU после n циклов. Дополнительные три цикла нужны для выхода 1867ВЦ4Т из режима IDLE3. Так же 1867ВЦ4Т не требует дополнительных двух циклов для синхронизации прерываний, потому что сигнал прерывания задает начальные условия синхронизации прерываний, которые позволяют определять прерывание сразу после запуска.

Когда для запуска из IDLE3 используется сигнал сброса, счетчик не используется; выход PLL сразу подключается к внутренней логике и сигнал CLKOUT утвержден. Время блокирования PLL – 50 нс, в этом случае CLKOUT устойчив. Поэтому необходимо удерживать сигнал сброса в низком состоянии на протяжении 50 нс, что не позволит начинать обработку данных при нестабильном тактовом сигнале.

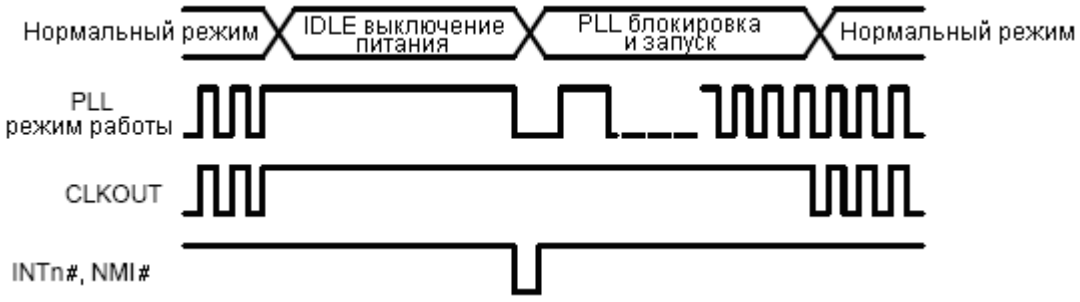


Рисунок 104 – Порядок запуска в режиме IDLE3

7.6 Режим HOLD

Два сигнала HOLD# и HOLDA# позволяют внешнему устройству принимать на себя управление внешней шиной. Процессор подтверждает получение сигнала HOLD# от внешнего устройства сигналом HOLDA#. Процессор входит в режим удержания (hold) и устанавливает шины адреса, данных и управления в высокоимпедансное состояние.

Статусный бит режима удержания (HM) регистра ST1 определяет следующие режимы для функций удержания:

– Нормальный режим приостанавливает выполнение программ при низком уровне сигнала HOLD# (HM = 1).

– Параллельный режим разрешает продолжить выполнение программ из внутренней памяти (ROM или RAM) HM = 0.

Процессор переходит в параллельный режим, если программа выполняется из внешней памяти или если выбирается операнд из внешней памяти. Однако, если программа выполняется из внутренней памяти, то процессор переходит в hold режим по внешним линиям, а выполнение программ внутри процессора продолжается. Таким образом, программа может продолжать выполнение, пока исполняются внешние операции. Это делает системные операции более эффективными.

При сбросе HM очищается в 0. HM устанавливается и сбрасывается командами SSBX и RSBX соответственно.

HOLD# не обрабатывается как прерывание. Если HOLD# принимается, CPU продолжает выполнение команд IDLE, не смотря на то, что внешние шины и управляющие сигналы устанавливаются в высокоимпедансное состояние.

Рисунок 105 показывает временные характеристики сигналов HOLD# и HOLDA#. HOLD# – это внешний асинхронный вход, который не защелкивается. Внешнее устройство должно удерживать HOLD# на низком уровне.

7.6.1 Прерывания во время режима HOLD

Все прерывания запрещены, пока HOLD# активен при HM = 1. Если прерывание принимается в течение этого периода, то оно защелкивается и остается в режиме ожидания; кроме того, HOLD# не влияет на флаги прерывания или регистры. Когда HM = 0, прерывания функционируют нормально.

7.6.2 Режим HOLD и сброс

Если HOLD# подтверждается во время активного сброса – RS# = 0, нормальная операция сброса происходит внутри, но все шины и управляющие линии остаются или становятся в высокоимпедансное состояние и подтверждается HOLDA#, как показано на рисунке 106 (a) и (b). Однако, если RS# переходит в 0 при активных HOLD# и HOLDA#, CPU сбрасывается и шины адреса, данных и управляющие сигналы остаются высокоимпедансном состоянии, как показано на рисунке 106 (c) и (d).

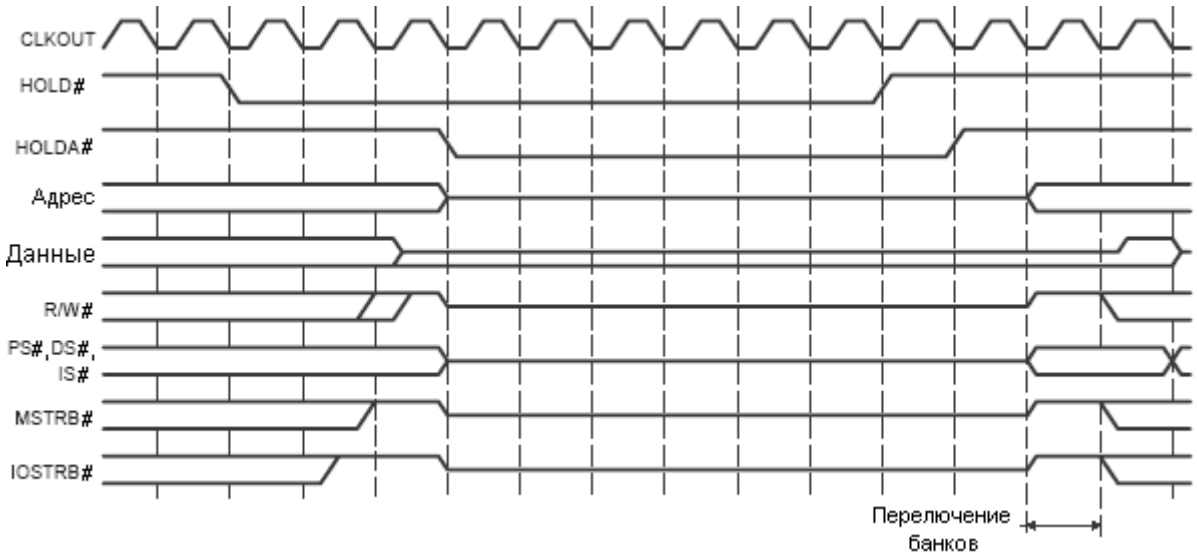


Рисунок 105 – Временные диаграммы HOLD# и HOLDA# при HM = 0

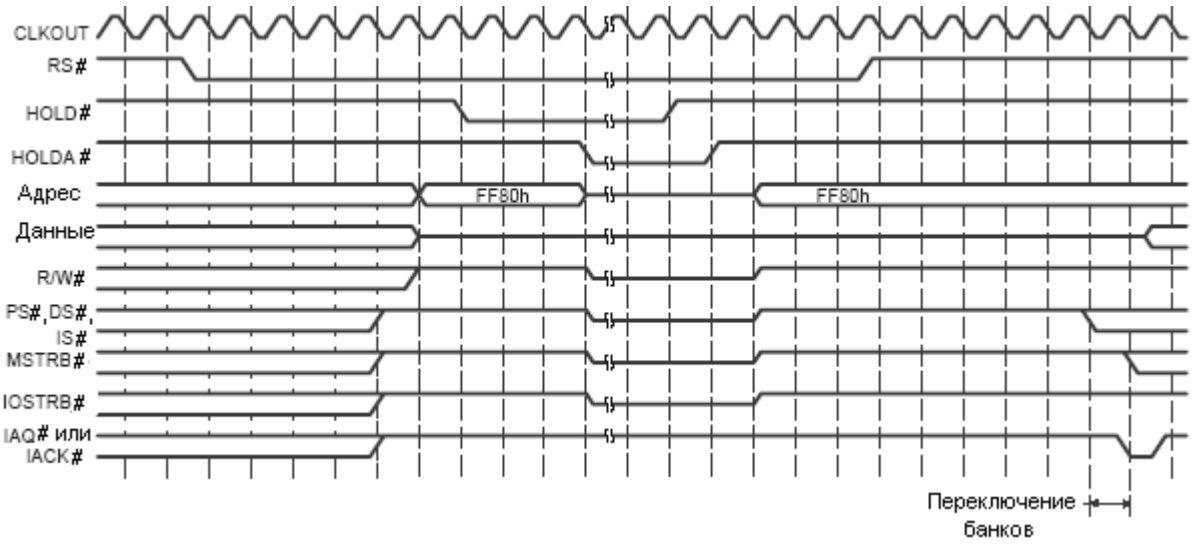


Рисунок 106(a) – Взаимодействия между HOLD# и RS#

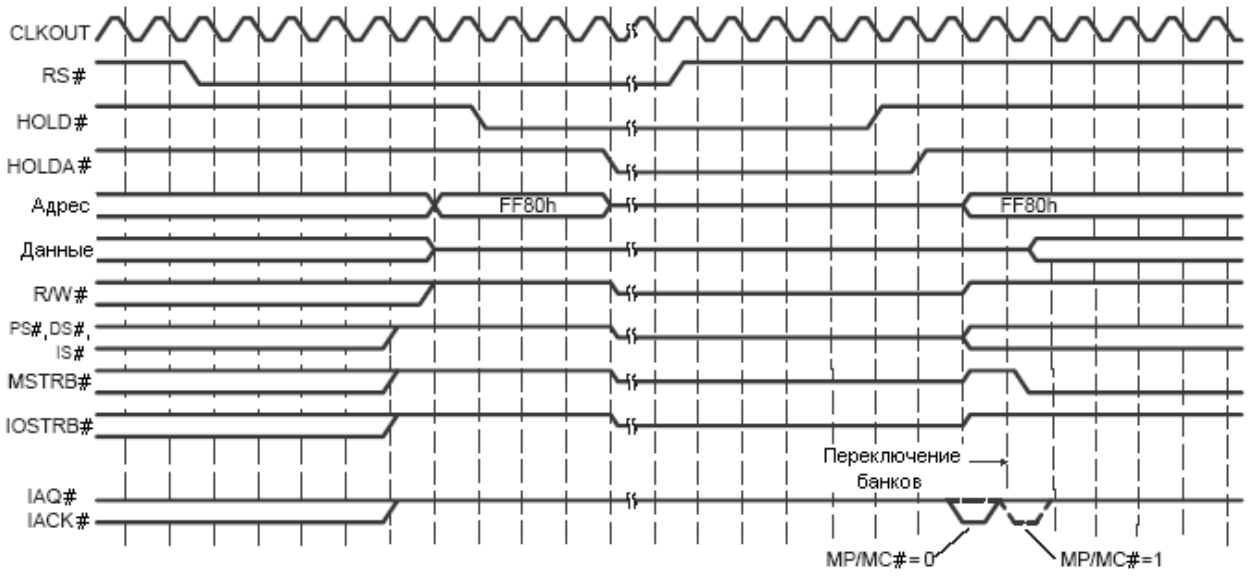


Рисунок 106(b) – Взаимодействия между HOLD# и RS#

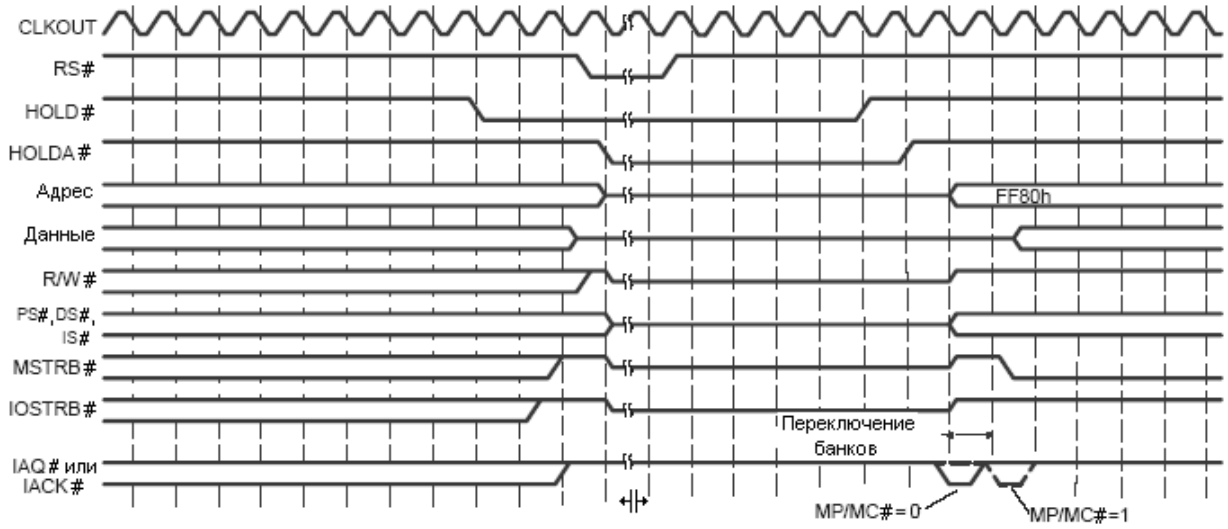


Рисунок 106(c) – Взаимодействия между HOLD# и RS#

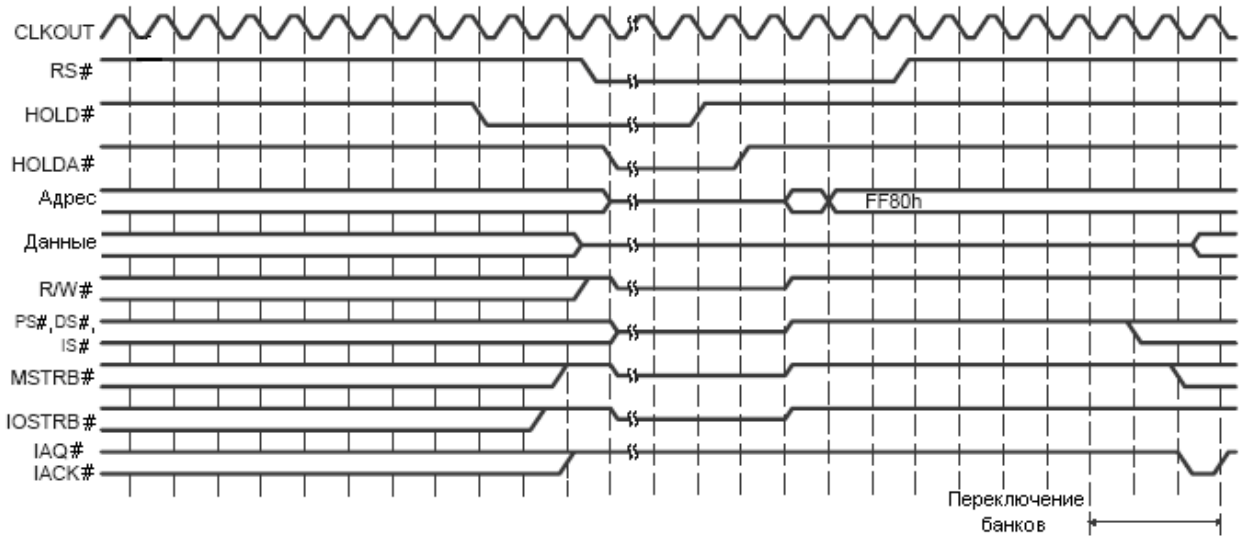


Рисунок 106(d) – Взаимодействия между HOLD# и RS#

Лист регистрации изменений								
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ докум.	Подп.	Дата
	изме-ненных	заме-ненных	новых	анну-лиро-ванных				