Архитектура

NeuroMatrix® NM6403
Руководство пользователя

Версия 1.0

АКЦИОНЕРНОЕ ОБЩЕСТВО
НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

Оглавление

Предисловие

В предисловии описывается назначение и состав документа, приводится краткий обзор разделов и глав, определяется стиль и символьные нотации, используемые в документе.

Данное руководство имеет целью ознакомить пользователя с архитектурой и системой команд процессора NeuroMatrix® NM6403, особенностям его работы с внешней памятью, принципами организации обмена по коммуникационным портам, дать рекомендации, как использовать этот процессор в реальной вычислительной системе.

Примечание: *В данном руководстве не содержится информация об ассемблере и SDK процессора NM6403. Для получения более детальной информации обращайтесь к документам "Базовое программное обеспечение процессора NM6403. Описание языка ассемблера" и "Базовое программное обеспечение процессора NM6403. Справочное руководство".*

О справочном руководстве

Данное руководство содержит следующую информацию:

- общую структуру процессора NM6403 на уровне основных блоков и шин;
- описания структуры и функционирования каждого блока в отдельности;
- форматы обрабатываемых данных и форматы команд процессора NM6403;
- организацию конвейера выполнения команд, а также системы прерываний;
- рекомендации по использованию NM6403 в составе реальных вычислительных устройств.

Как организован данный документ

Данный документ разбит на тринадцать глав и приложение. Разбиение на части осуществляется следующим образом:

Глава 1 **Введение**

Данная глава содержит сведения общего характера о назначении процессора NeuroMatrix® NM6403 и его основных характеристиках.

Глава 2 **Архитектура процессора NM6403**

В данной главе излагаются основные архитектурные особенности процессора NM6403, даётся краткое описание его основных блоков и шин, адресного пространства, программно доступных регистров и системы прерываний.

Глава 3 **RISC-ядро процессора NM6403**

Данная глава содержит детальное описание основного узла процессора NM6403 - RISC-ядра: его структуры и режимов работы.

Глава 4 **Векторный узел**

В данной главе приводится детальное описание блока обработки векторных данных процессора NM6403 - векторного узла, а также содержатся сведения о формате этих векторных данных.

Глава 5 **Методы адресации памяти**

Данная глава содержит описание методов адресации команд, скалярных и векторных данных, а также особенностей работы с системным стеком и стеками пользователя.

Глава 6 **Управление потоком команд**

В данной главе приводятся сведения об особенностях работы векторных команд и команд управления, детально описывается система прерываний, а также поведение процессора NM6403 при системном сбросе.

Глава 7 **Введение в систему команд**

Данная глава содержит краткое описание системы команд, более полно она изложена в приложении А.

Глава 8 **Конвейерная организация выполнения команд**

В данной главе описываются особенности выполнения скалярных, векторных команд и команд управления в конвейере, а также возможные при этом конфликты.

Глава 9 **Интерфейс с памятью**

Данная глава содержит сведения об организации и работе интерфейса с внешней памятью, его регистрах управления и внешних выводах.

Глава 10 **Коммуникационные порты ввода/вывода**

Данная глава содержит описание организации и работы коммуникационных портов ввода/вывода процессора NM6403, аппаратно совместимых с коммуникационными портами ЦПС TMS320C4x, его регистров и внешних выводов.

Глава 11 **Программные приложения**

В данной главе приводятся примеры написания программ на языке ассемблера для процессора NM6403: обработка данных различного типа, управление коммуникационными портами и таймерами.

Глава 12 **Аппаратные приложения**

Данная глава содержит сведения о работе процессора NM6403 в составе реальных устройств, а также даются рекомендации о его использовании в качестве базового элемента при построении многопроцессорных вычислительных систем.

Глава 13 **Основные конструктивные, электрические и динамические характеристики микросхемы процессора NM6403**

В данной главе приводятся сведения о составе и расположение внешних выводов микросхемы, её корпусе, электрических характеристиках и временных параметрах.

Приложение А **Система команд NeuroMatrix® NM6403**

Данное приложение содержит детальное описание системы команд, форматов обрабатываемых данных, программно доступных регистров.

Соглашения о нотациях

В данном справочном руководстве используются следующие типографические нотации:

courier	так помечается текст, который может быть набран пользователем с клавиатуры: исходные тексты на языках Си++ и ассемблера.
`Courier`	отмечает текст, который должен быть заменен пользовательской информацией, например, реальным значением константы.
Текст или Текст	Так помечается текст, на который необходимо обратить особое внимание.
//Текст	Так помечаются комментарии к программам.

Примечание: *Данное примечание представляет собой пример того, как оформлены все важные замечания и комментарии, возникающие по ходу описания.*

1 Введение

1.1 Общие характеристики и функциональные особенности

NM6403 представляет собой высокопроизводительный микропроцессор с элементами VLIW и SIMD архитектур. В его состав входят устройства управления, вычисления адреса и обработки скаляров, а также узел для поддержки операций над векторами с элементами переменной разрядности. Кроме того, имеются два идентичных программируемых интерфейса для работы с внешней памятью различного типа, а также два коммуникационных порта, аппаратно совместимых с портами ЦПС TMS320C4x, для возможности построения многопроцессорных систем.

Внешний интерфейс процессора NM6403 изображен на Рис. 1-1. Всего процессор NM6403 имеет 256 выводов, их обозначение, тип и функциональное назначение приведены в Табл. 1-1.

Рис. 1-1. Внешний интерфейс процессора NeuroMatrix® NM6403

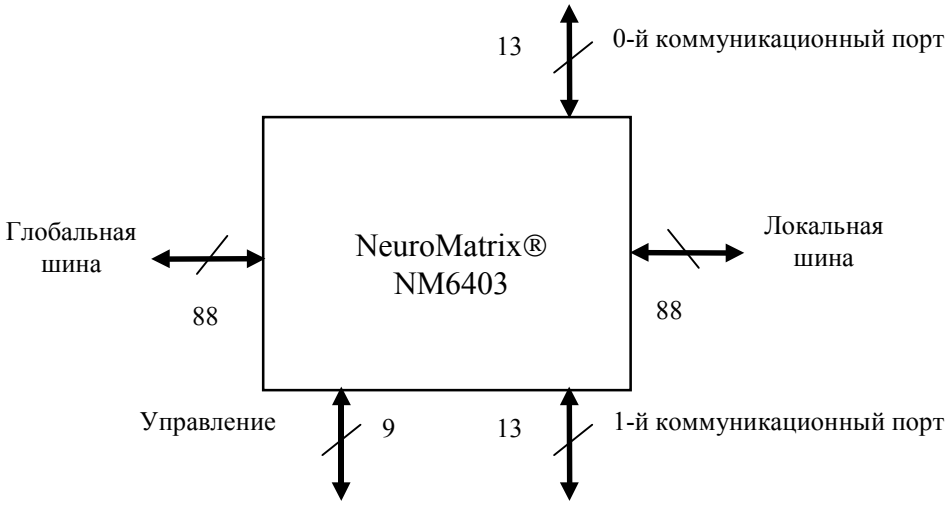


Табл. 1-1. Выводы процессора NM6403 и их функциональное назначение

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
Интерфейс с глобальной шиной (88 выводов)			
D63 - D0	64	I/O(Z)	64-разрядная шина данных
A15 - A1	15	O(Z)	15-разрядная шина адреса
$\overline{RAS0}$ / $\overline{CS0}$	1	O(Z)	строб адреса строки 0-го банка динамического ОЗУ /строб адреса страницы 0-го банка статического ОЗУ
$\overline{RAS1}$ / $\overline{CS1}$	1	O(Z)	строб адреса строки 1-го банка динамического ОЗУ /строб адреса страницы 1-го банка статического ОЗУ

Табл.1-1. Выводы процессора NM6403 и их функциональное назначение (Продолжение)

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
$\overline{\text{CASL}}$ / $\overline{\text{WEL}}$	1	O(Z)	строб адреса столбца 32 младших разрядов динамического ОЗУ/ разрешение записи в 32 младших разряда статического ОЗУ
$\overline{\text{CASH}}$ / $\overline{\text{WEH}}$	1	O(Z)	строб адреса столбца 32 старших разрядов динамического ОЗУ/ разрешение записи в 32 старших разряда статического ОЗУ
$\overline{\text{OE}}$	1	O(Z)	разрешение выдачи данных из ОЗУ
$\overline{\text{WE}}$ /A16	1	O(Z)	признак цикла (запись/чтение) для динамического ОЗУ/16-й разряд шины адреса
$\overline{\text{HOLDO}}$/A17[3)]	1	O(Z)	запрос на владение шиной процессором NM6403 /17-й разряд шины адреса
$\overline{\text{HOLDI}}$ /A18[3)]	1	I/O(Z)	запрос на захват шины извне/18-й разряд шины адреса
$\overline{\text{RDY}}$ /A19[3)]	1	I/O(Z)	сигнал готовности/19-й разряд шины адреса
Интерфейс с локальной шиной (88 выводов)			
LD63 - LD0	64	I/O(Z)	64-разрядная шина данных
LA15 - LA1	15	O(Z)	15-разрядная шина адреса
$\overline{\text{LRAS0}}$ / $\overline{\text{LCS0}}$	1	O(Z)	строб адреса строки 0-го банка динамического ОЗУ /строб адреса страницы 0-го банка статического ОЗУ
$\overline{\text{LRAS1}}$ / $\overline{\text{LCS1}}$	1	O(Z)	строб адреса строки 1-го банка динамического ОЗУ /строб адреса страницы 1-го банка статического ОЗУ
$\overline{\text{LCASL}}$ / $\overline{\text{LWEL}}$	1	O(Z)	строб адреса столбца 32 младших разрядов динамического ОЗУ/ разрешение записи в 32 младших разряда статического ОЗУ
$\overline{\text{LCASH}}$ / $\overline{\text{LWEH}}$	1	O(Z)	строб адреса столбца 32 старших разрядов динамического ОЗУ/ разрешение записи в 32 старших разряда статического ОЗУ
$\overline{\text{LOE}}$	1	O(Z)	разрешение выдачи данных из ОЗУ
$\overline{\text{LWE}}$ /LA16	1	O(Z)	признак цикла (запись/чтение) для динамического ОЗУ/16-й разряд шины адреса

Табл.1-1. Выводы процессора NM6403 и их функциональное назначение (Продолжение)

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
$\overline{\text{LHOLDO}}$ /LA17[3)]	1	O(Z)	запрос на владение шиной процессором NM6403/ 17-й разряд шины адреса
$\overline{\text{LHOLDI}}$ /LA18[3)]	1	I/O(Z)	запрос на захват шины извне/18-й разряд шины адреса
$\overline{\text{LRDY}}$ /LA19[3)]	1	I/O(Z)	сигнал готовности/19-й разряд шины адреса
Коммуникационный порт 0 (13 выводов)			
C0D7 - C0D0	8	I/O(Z)	шина данных
$\overline{\text{CREQ0}}$	1	I/O(Z)	запрос шины
$\overline{\text{CACK0}}$	1	I/O(Z)	подтверждение запроса шины
$\overline{\text{CSTRB0}}$	1	I/O(Z)	строб данных
$\overline{\text{CRDY0}}$	1	I/O(Z)	сигнал готовности данных
$\overline{\text{CDIR0}}$	1	O	направление передачи данных
Коммуникационный порт 1 (13 выводов)			
C1D7 - C1D0	8	I/O(Z)	шина данных
$\overline{\text{CREQ1}}$	1	I/O(Z)	запрос шины
$\overline{\text{CACK1}}$	1	I/O(Z)	подтверждение запроса шины
$\overline{\text{CSTRB1}}$	1	I/O(Z)	строб данных
$\overline{\text{CRDY1}}$	1	I/O(Z)	сигнал готовности данных
$\overline{\text{CDIR1}}$	1	O	направление передачи данных
Общее управление (9 выводов)			
CLK	1	I	тактовый сигнал
$\overline{\text{RESET}}$	1	I	сброс
TIMER	1	I/O(Z)	выход таймера
$\overline{\text{INT}}$	1	I	внешнее прерывание
$\overline{\text{INTA}}$	1	O	подтверждение внешнего прерывания
BOOTM	1	I	режим начальной загрузки
MVOE	1	I	разрешение выдачи данных на внешнюю шину при внутренних пересылках из регистра в регистр в процессоре NM6403

Табл.1-1. Выводы процессора NM6403 и их функциональное назначение (Продолжение)

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
TEST_OUT1	1	O	тестовый вывод (используется изготовителем микросхемы)
TEST_OUT2	1	O	тестовый вывод (используется изготовителем микросхемы)
Питание (27 выводов)			
VSS	13	I	общий
VDD	12	I	питание (3.3 В)
VBB	2	I	напряжение смещения для толерантных выводов (3.3 В или 5 В)
Неиспользуемые выводы (18 выводов)			
NC	18	-	неиспользуемый вывод

Примечание:

1) Для выводов со знаком инверсии активным является низкий уровень сигнала;

2) I- входы;

O - выходы;

O(Z) - выходы с высокоимпедансным состоянием;

I/O(Z) - двунаправленные выводы.

3) Данные выводы при многопоцессорном режиме служат при необходимости доступа к общей памяти для арбитража, а при однопоцессорном режиме в качестве дополнительных разрядов адреса.

Основные характеристики процессора NeuroMatrix® NM6403

- тактовая частота - 40 МГц (машинный такт - 25 нс)
- число эквивалентных вентилей - 115.000
- технология 0,5 мкм
- корпус 256BGA
- малое напряжение питания, от 2.7В до 3.6В
- адресное пространство - 16 Гбайт

- формат скалярных и векторных данных:
  - 32-разрядные скаляры
  - вектора с элементами переменной разрядности от 1 до 64, упакованные в 64-разрядные блоки данных
- аппаратная поддержка операций умножения вектора на матрицу или матрицы на матрицу
- аппаратная реализация функции насыщения
- два устройства генерации адреса
- регистры:
  - 8 32-разрядных регистров общего назначения
  - 8 32-разрядных адресных регистров
  - 3 внутренних памяти по 32*64 бит
  - специальные регистры управления и состояния
- команды процессора NM6403 32- и 64-разрядные (одна команда обычно задаёт две операции)
- два 64-разрядных программируемых интерфейса для работы с любым типом внешней памяти. Каждый интерфейс поддерживает обмен с двумя банками памяти разного типа (статическая или динамическая память)
- два скоростных байтовых коммуникационных порта ввода/вывода, аппаратно совместимых с портами TMS320C4x.

Применение

- обработка видеоизображений:
  - телекоммуникации
  - сигнальная обработка
  - апаратная поддержка векторно-матричных операций
- основной блок для построения больших суперпараллельных вычислительных систем и реализации нейросетевых технологий

Производительность

- скалярные операции:
  - 40MIPS

- 120 MOPS для 32-разрядных данных

- векторные операции:
  - 960.000.000 умножений и сложений в секунду (при перемножении матриц с 8-разрядными элементами)

- интерфейс с памятью и перефирия:
  - суммарная пропускная способность по 64-разрядным интерфейсам с внешней памятью - до 800 Мбайт/сек.
  - темп обмена по каждому коммуникационному порту ввода/вывода - до 20 Мбайт/сек.

Базовое программное обеспечение (БПО)

Базовое программное обеспечение процессора NM6403 обеспечивает полный цикл разработки и отладки прикладных программ. БПО позволяет разрабатывать прикладные программы на языке Си++ и на языке ассемблера процессора NM6403.

В БПО состав входят:

- компилятор Си++,
- ассемблер,
- редактор связей,
- программный эмулятор,
- символьный отладчик,
- библиотекарь объектных файлов,
- набор системных и прикладных библиотек.

Пользователь имеет возможность транслировать программы на языке Си++ в программы на языке ассемблера, транслировать ассемблерные программы в объектные файлы, собирать из объектных файлов библиотеки объектных файлов, создавать исполнимые файлы для NM6403 путем объединения нескольких объектных файлов.

Разработанный язык ассемблера имеет интуитивный синтаксис, приближающийся к языкам высокого уровня, что упрощает разработку программ и их чтение.

2 Архитектура процессора NM6403

2.1 Общая структура процессора NM6403: основные блоки и шины

Общая структура процессора NM6403 на уровне внутренних шин и блоков представлена на Рис. 2-1. NM6403 содержит следующие основные блоки и шины:

- **Внешние шины (глобальная и локальная)**, подключенные через программируемые интерфейсы: **GMI** - интерфейс глобальной шины; **LMI** - интерфейс локальной шины.

Процессор NM6403 может обращаться через глобальную и локальную шину к двум внешним памятям, каждая из которых содержит до двух банков, различающихся типом и временными параметрами.

Внутри NM6403 каждый из интерфейсов соединен соответственно с шиной глобального адреса (GLOBAL ADDRESS BUS) и шиной локального адреса (LOCAL ADDRESS BUS), а также с двумя шинами для ввода данных и команд (INPUT WEIGHT BUS и INPUT INSTRUCTION BUS), шиной вывода данных (OUTPUT DATA BUS).

Специфика функционального использования шин команд INPUT WEIGHT BUS и INPUT INSTRUCTION BUS для передачи данных и команд рассматривается далее.

Шины команд INPUT WEIGHT BUS, INPUT INSTRUCTION BUS, OUTPUT DATA BUS, LOCAL ADDRESS BUS и GLOBAL ADDRESS BUS образуют группу шин, называемую внутренними шинами процессора NM6403.

- **RISC-ядро (RISC-core)** содержит 8 32-х разрядных адресных регистра , 8 32-х разрядных регистров общего назначения, счетчик адреса программы и слово состояния программы, два функциональных устройства адресных вычислений, функциональное устройство для выполнения операций над скалярами, два таймера, а также регистры управления и состояния.

RISC-ядро производит вычисления со скалярами и адресные вычисления, которые могут задаваться явно соответствующими командами адресной арифметики, а также неявно, в командах обработки векторов.

- **Векторный узел (Vector unit)** включает 3 блока внутренней памяти, каждый из которых содержит 32 64-разрядных слова, набор специальных регистров управления, а также функциональное устройство с настраиваемой на разрядность операндов структурой для выполнения матричных операций.

- **Сопроцессор прямого доступа к памяти (DMA coprocessor)** с двумя коммуникационными портами (**Port0** и **Port1**) для обмена по двунаправленным байтовым линкам (**Link0** и **Link1**). Прием и выдачу информации по линкам можно выполнить только через внешнюю память по схеме “внешняя память” ->Link или Link->“внешняя память”.

- **Устройство управления (Control unit)** задаёт и контролирует правильность конвейерного выполнения команд, осуществляет арбитраж использования внутренних и внешних шин, обслуживает внутренние и внешние прерывания. Всего имеется 9 прерываний - 1 внешнее и 8 внутренних.

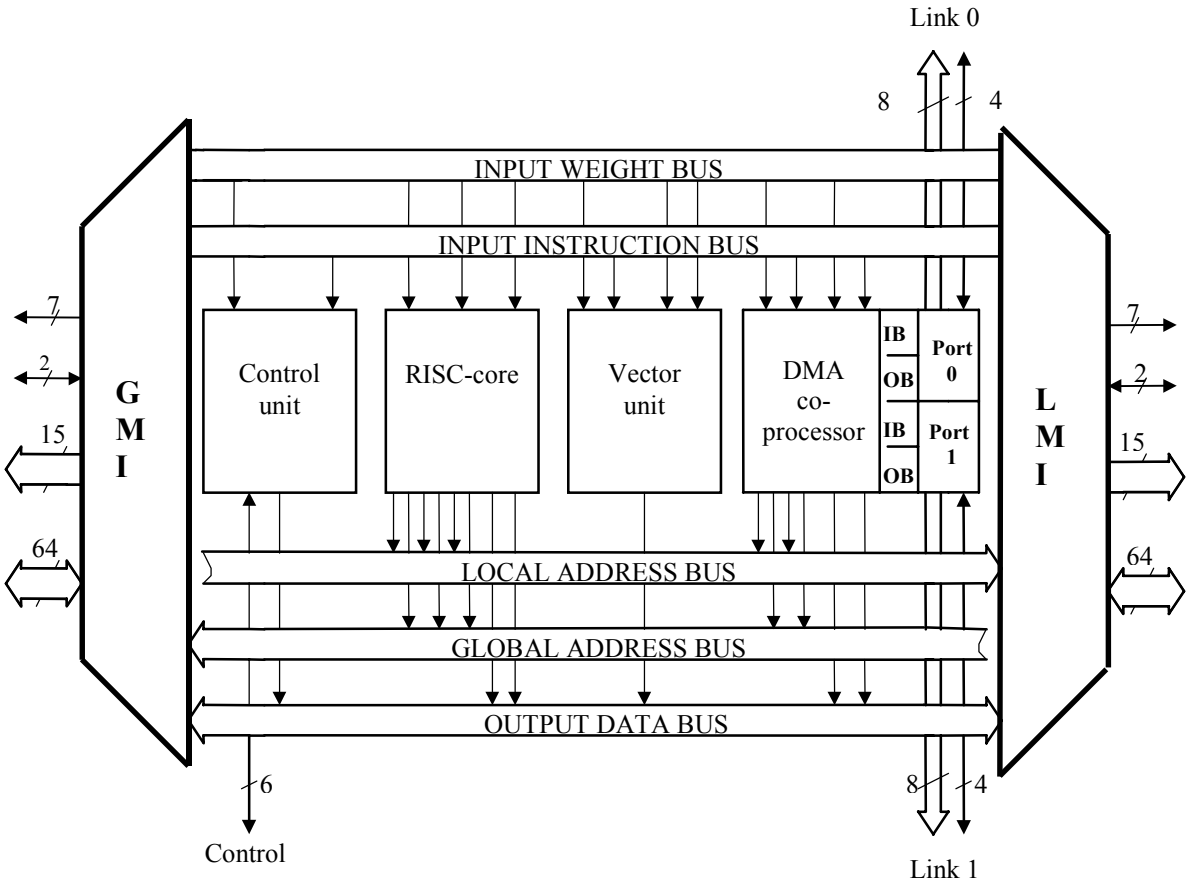
  Специфика использования шин INPUT WEIGHT BUS и INPUT INSTRUCTION BUS состоит в том, что они обе используются как для передачи данных, так и для передачи команд. В RISC-ядре и векторном узле, сопроцессоре прямого доступа к памяти эти шины используются следующим образом:

  RISC-ядро - INPUT WEIGHT BUS - передача данных и команд, INPUT INSTRUCTION BUS- преимущественно передача команд;

  Векторный узел- INPUT WEIGHT BUS - передача данных-весовых коэффициентов и команд, INPUT INSTRUCTION BUS - передача данных и команд;

  Процессор прямого доступа к памяти - INPUT WEIGHT BUS передача данных и команд, INPUT INSTRUCTION BUS - передача данных и команд.

Рис. 2-1. Общая структура процессора NeuroMatrix® NM6403



GMI - интерфейс глобальной шины
LMI - интерфейс локальной шины
Link0 - 0-й коммуникационный порт
Link1 - 1-й коммуникационный порт

2.2 Программно доступные регистры

В процессоре NM6403 программно доступны следующие регистры:

- адресные регистры, регистры общего назначения и счётчик команд;
- специальные регистры:
  - ♦ слово состояния процессора и регистр запросов на прерывание и ПДП;
  - ♦ регистры управления векторного процессора;
  - ♦ регистры управления интерфейсом локальной и

глобальной шины;

- регистры каналов ввода/вывода;
- таймеры.

Перечень всех регистров с указанием их разрядности и назначения дан в Табл. 2-1.

Табл. 2-1. Программно доступные регистры процессора NM6403

Обозначение регистров и их назначение	Разряд-ность
GRi - регистр общего назначения i (i=0,…,7)	32
ARj - адресный регистр j (j=0,…,6)	32
SP(AR7) - указатель стека адресов возврата	32
PC - программный счетчик	32
PSWR - регистр слова состояния процессора	32
Ti - таймер i (i=0,1)	32
GMICR - регистр управления интерфейсом с глобальной шиной	32
LMICR - регистр управления интерфейсом с локальной шиной	32
INTR - регистр запросов на прерывание и ПДП	32
OCAi - регистр адреса канала вывода i (i=0,1)	32
ICAi - регистр адреса канала ввода i (i=0,1)	32
OCCi - счетчик канала вывода i (i=0,1)	32
ICCi - счетчик канала ввода i (i=0,1)	32
DORi - регистр данных канала вывода i (i=0,1)	64
DIRi - регистр данных канала ввода i (i=0,1)	64
FiCR (H,L) - регистр управления функцией активации i (i=1,2) (старшая, младшая часть)	64(32)
VR(H,L) - регистр порога (старшая, младшая часть)	64(32)
NB(H,L) - регистр границ нейронов (старшая, младшая часть)	64(32)
SB(H,L) - регистр границ синапсов (старшая, младшая часть)	64(32)

2.3 Организация памяти

Процессор NM6403 использует 32-разрядный вычисляемый адрес при обращении во внешнюю память, причём обмен происходит по 32 или 64 разряда. Таким образом, доступное адресное пространство равно 16 Гбайт. Оно делится на две равные части - локальное и

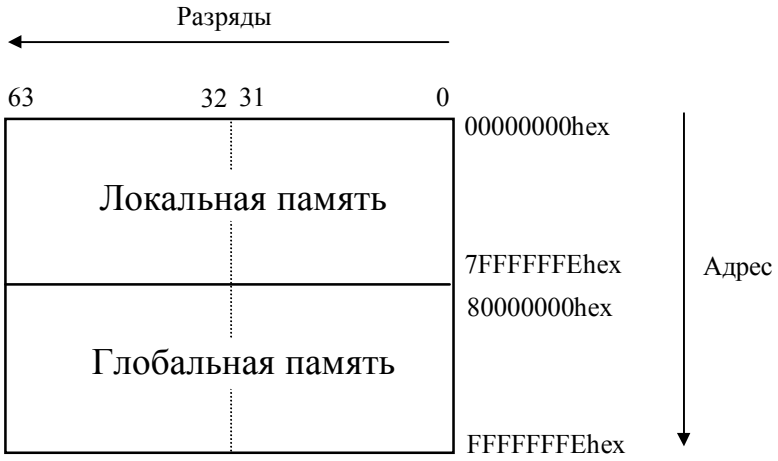
глобальное (см. Рис. 2-2). Если старший разряд адреса равен нулю, идёт обращение к локальной памяти, если единица - к глобальной. Младший разряд вычисляемого адреса используется при доступе к 32-разрядным данным: если он равен нулю, используется младшая часть памяти (разряды 31 - 0), если единица - старшая (разряды 63 - 32). При обращении за 64-разрядными данными или при выборке команд он игнорируется.

Обмен 32-разрядными данными с внешней памятью производится только скалярными командами, если в качестве источника или приёмника в них указан 32-разрядный регистр. В случае, когда это 64-разрядный регистр или когда обмен задаётся векторной командой, используются соответственно 64-разрядные данные.

Выборка команд из памяти всегда осуществляется по 64 разряда, хотя процессор NM6403 работает как с 32-, так и 64-разрядными командами. Это накладывает определённые ограничения на расположение команд в памяти: все 64-разрядные команды, а также адреса, по которым осуществляется переход, должны быть выровнены по чётному адресу. В случае выборки одновременно двух 32-разрядных команд первой будет выполняться команда, находящаяся в младшей половине 64-разрядного слова.

Отличие глобальной и локальной памяти связано кроме адресации с состоянием шин процессора NM6403, через которые эти памяти подключены, после системного сброса. Шина локальной памяти после сигнала сброса принадлежит процессору NM6403, а шина глобальной памяти - не принадлежит. Такая несимметричность сделана для упрощения мультипроцессорной работы.

Рис. 2-2. Адресное пространство процессора NM6403



2.4 Способы адресации памяти

Внешняя память процессора NM6403 адресуется при обращениях за командами, а также при чтении и записи операндов скалярных или

векторных команд.

Способы адресации команд

- адресация по содержимому счетчика адреса команд;
- непосредственная адресация по абсолютному адресу, заданному в команде;
- адресация по содержимому AR-регистра или GR-регистра;
- адресация по сумме содержимого AR-регистра и GR-регистра
- адресация относительно содержимого AR-регистра или счетчика адреса команд с непосредственно заданным смещением.

Способы адресации данных

- непосредственная адресация по абсолютному адресу, заданному в команде;
- адресация по содержимому AR-регистра или GR-регистра;
- адресация по сумме содержимого AR-регистра и GR-регистра;
- адресация относительно содержимого AR-регистра с непосредственно заданным смещением.

Более подробно о методах адресации, используемых процессором NM6403, см в главах 5 и 7 .

2.5 Система прерываний процессора NM6403

Процессор NM6403 поддерживает одно внешнее прерывание и девять внутренних:

- два прерывания от таймеров;
- прерывание по переполнению при выполнении арифметической операции, заданной скалярной командой;
- прерывание по запрещенной векторной команде;
- четыре прерывания от каналов ввода-вывода по завершению обмена через коммуникационные порты;
- пошаговое прерывание в режиме отладки.

Более подробно о прерываниях процессора NM6403 см. в разделе 6.3.

3 RISC-ядро процессора NM6403

3.1 Структура RISC-ядра

RISC-ядро является одним из основных узлов процессора NM6403. Оно предназначено для вычисления адресов команд и управления их выборкой, вычисления адресов операндов и весовых коэффициентов при работе процессора NM6403 с памятью, а также для поддержки арифметических и логических операций над 32-разрядными данными в дополнительном коде, когда использование векторного узла неэффективно. Структурная схема RISC-ядра представлена на Рис. 3-1. В состав его входят:

- PROGRAMM SEQUENCER - устройство выборки команд;
- DAG1 - генератор адреса данных 1;
- DAG2 - генератор адреса данных 2;
- RALU - блок регистрового АЛУ;
- TIMERS BLOCK - блок таймеров;
- CONTROL AND STATE REGISTERS BLOCK - блок регистров управления и состояния.

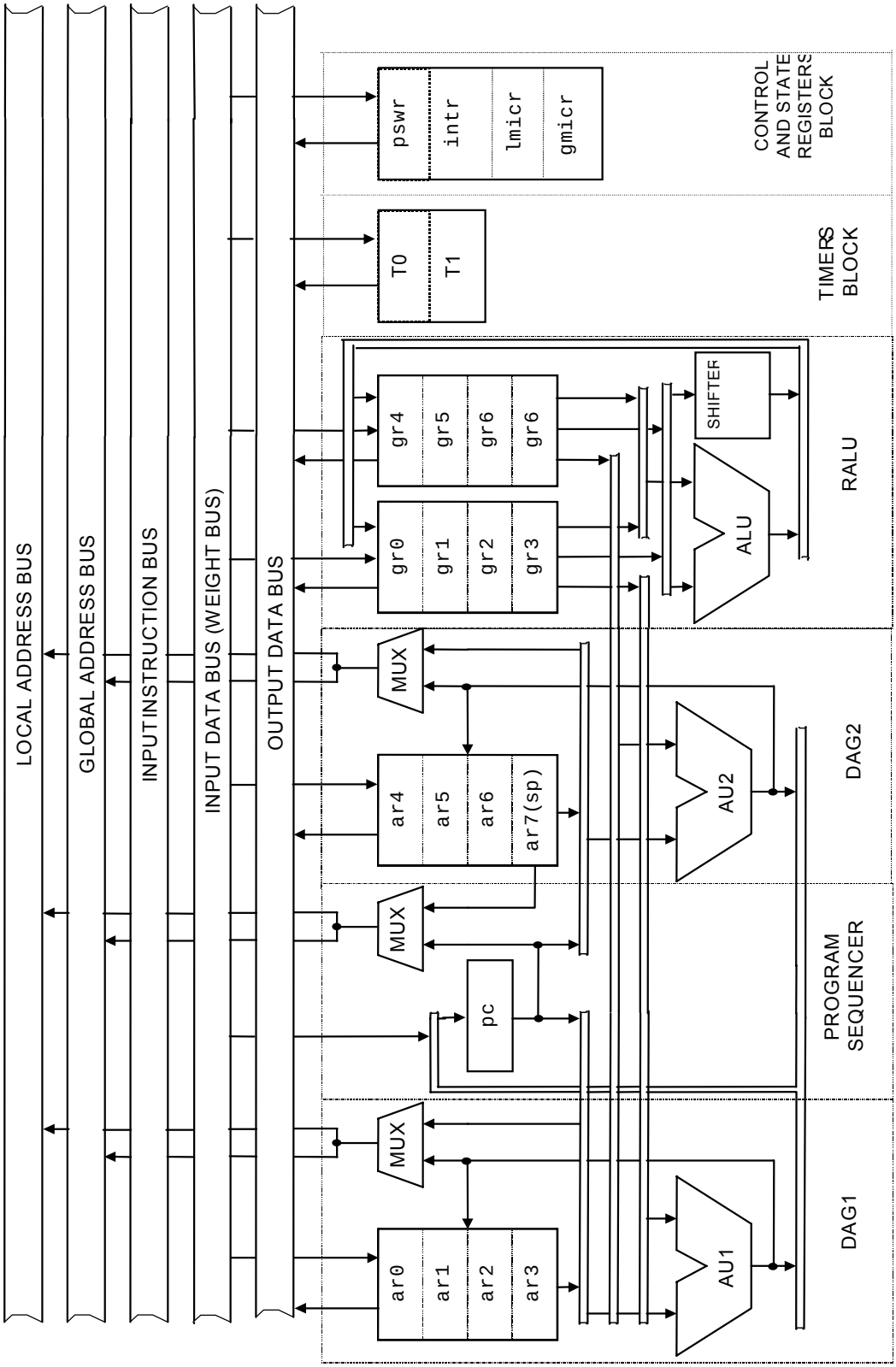
Устройство выборки команд PROGRAMM SEQUENCER

Устройство выборки команд на линейных участках программы служит для вычисления адреса новой команды в режиме инкремента и формирования запроса блоку арбитража на её выборку из внешней памяти. В случае, когда требуется осуществить переход, его адрес вычисляется на генераторе адреса данных 1 или 2, а данное устройство используется при отработке прерываний или команд перехода к подпрограмме для создания запроса блоку арбитража на запись во внешнюю память адреса возврата и слова состояния процессора по адресу, хранящемуся в системном указателе стека адресов возврата - AR7(SP). Если запрос блоком арбитража удовлетворён, то указатель стека автоматически увеличивается на 2. PROGRAMM SEQUENCER включает в себя следующие узлы:

PC - счетчик команд для вычисления адреса следующей выбираемой команды;

MUX - мультиплексор для формирования адреса запроса на работу с внешней памятью от PROGRAMM SEQUENCER.

Рис. 3-1. Структурная схема RISC–ядра процессора NM6403



Генератор адреса данных DAG1

Генератор адреса данных DAG1 служит для вычисления исполнительного адреса при обращении к памяти за данными и адресов перехода при ветвлениях, вызовах подпрограмм и отработке прерываний, а также для создания запроса блоку арбитража на работу с внешней памятью по этим адресам. DAG1 включает в себя следующие узлы:

AR0, … , AR3 - адресные регистры 0, … , 3;

AU1 - арифметическое устройство 1 для вычисления адреса, одним из операндов которого может быть любой из адресных регистров AR0 - AR3 или PC, а вторым - любой из регистров общего назначения GR0 - GR3 или 32-разрядная константа из кода команды;

MUX - мультиплексор для формирования адреса запроса на работу с внешней памятью от DAG1.

Генератор адреса данных DAG2

Генератор адреса данных DAG2 аналогичен DAG1 за исключением того, что он использует свои регистры, а также что на нём формируется запрос на чтение адреса возврата по командам возврата из подпрограммы или прерывания (для возврата из прерывания читается также старое слово состояния процессора) по адресу, хранящемуся в системном указателе стека адресов возврата - AR7(SP), уменьшенному на 2 с помощью арифметического устройства 2. Если данный запрос блоком арбитража удовлетворён, то указатель стека автоматически уменьшается на 2. DAG1 включает в себя следующие узлы:

AR4, … , AR7 - адресные регистры 4, … , 7, причём AR7 может использоваться как обычный адресный регистр, так и в качестве системного указателя стека адресов возврата - SP;

AU2 - арифметическое устройство 2 для вычисления адреса, одним из операндов которого может быть любой из адресных регистров AR4 - AR7 или PC, а вторым - любой из регистров общего назначения GR3 - GR7 или 32-разрядная константа из кода команды;

MUX - мультиплексор для формирования адреса запроса на работу с внешней памятью от DAG2.

Блок регистрового АЛУ RALU

Блок регистрового АЛУ служит для поддержки арифметических операций над скалярными 32-разрядными данными в дополнительном коде. RALU включает в себя следующие узлы:

GR0, … , GR7 - регистры общего назначения 0, … , 7;

ALU - арифметико-логическое устройство для выполнения арифметических или логических операций над операндами, которыми могут быть любые два регистра общего назначения;

SHIFTER - устройство сдвига для выполнения циклических, логических или арифметических сдвигов операнда, в качестве которого может использоваться любой из регистров общего назначения.

Блок таймеров

Блок таймеров предназначен для отсчёта заданных временных интервалов. Он состоит из двух независимых 32-разрядных таймеров - T0 и T1, каждый из которых может работать как в непрерывном, так и однократном режимах. Необходимый интервал работы таймера задаётся записью в него нужного числа машинных тактов, умноженного на -1 в дополнительном коде. Наименьшее число считаемых тактов - 1 (записываемый код - FFFFFFFF hex), наибольшее - 2^{32} (записываемый код - 00000000 hex). После того, как таймер досчитает до нуля в режиме инкремента, формируется соответствующее прерывание. Если был задан непрерывный режим работы, восстанавливается исходное содержимое таймера и продолжается счёт. Если нет, счёт прекращается и содержимое таймера остаётся равным нулю. В случае повторного запуска таймера опять восстановится его старое содержимое. Когда процессор NM6403 работает с динамической памятью, таймер T0 используется для формирования запросов на регенерацию этой памяти и должен быть соответствующим образом программно настроен.

Управление таймерами T0 и T1 осуществляется с помощью установки или сброса соответствующих битов в регистре PSWR.

Блок регистров управления и состояния

Блок регистров управления и состояния содержит следующие 32-разрядные регистры:

PSWR - слово состояния процессора;

INTR - регистр запросов на прерывание и ПДП;

LMICR - регистр управления интерфейсом с локальной шиной;

GMICR - регистр управления интерфейсом с глобальной шиной.

Более подробно о данных регистрах будет рассказано в разделе 3.3.

3.2 Основные режимы работы RISC-ядра

Структура RISC-ядра выбрана таким образом, чтобы он мог

работать одновременно в двух режимах:

1) поддержка операций векторного узла, когда RISC-ядро вычисляет адрес следующей команды, а также до двух исполнительных адресов для операндов векторного узла и для загрузки весовых коэффициентов;

2) поддержка команд управления (организация циклов, ветвлений и т.д.) и скалярных команд обработки данных, когда RISC-ядро вычисляет адрес следующей команды, исполнительный адрес для скалярных операндов, а также выполняет арифметические и логические операции над данными в GR0-GR7.

Адрес следующей команды может быть определен либо с помощью инкрементации PC, либо путем его задания константой в коде команды, либо сложением содержимого одного из адресных регистров - AR0, …, AR7 или PC со смещением, заданным константой в коде команды или содержимым одного из регистров общего назначения - GR0, ..., GR7. Тем самым в скалярном процессоре поддерживаются следующие команды управления (как условные, так и безусловные): переход/переход со смещением (JUMP/SKIP), переход к подпрограмме (CALL), возврат из подпрограммы/прерывания (RET/RETI).

Методы адресации операндов

Исполнительный адрес операндов может вычисляться с использованием следующих методов адресации: по содержимому адресного регистра без его изменения, с его инкрементацией, декрементацией, а также по сумме содержимого адресного регистра и соответствующего регистра общего назначения. Для скалярных команд кроме этого возможно определение исполнительного адреса с помощью константы, задаваемой в поле команды. Более подробно о командах управления и методах адресации можно узнать в главах 5 и 7.

3.3 Регистры управления и состояния

RISC-ядро содержит четыре регистра управления и состояния:

PSWR - слово состояния процессора;

INTR - регистр запросов на прерывание и ПДП;

LMICR - регистр управления интерфейсом с локальной шиной;

GMICR - регистр управления интерфейсом с глобальной шиной.

Ниже приводится их детальное описание.

Рис. 3-2. Регистр слова состояния процессора NM6403 PSWR (32 разряда)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BC		TIMER pin control				CP1 cont.			CP0 cont.			T0C		T1C		FCL		INTERRUPT MASKS												FLAGS			
GBRE	LBRE	TEN	TM2	TM1	TM0	CPI1	CHI1	COI1	CPI0	CH10	CO10	TME0		TME1		WFCL	AFCL	TS0M	SPMM	VPMM	INTM	IC1M	IC0M	OC1M	OC0M	TM1M	SM			NZVC			

Все разряды регистра слова состояния процессора NM6403 программно доступны по записи и чтению, после системного сброса они содержат нули. Функциональное назначение данных разрядов приведено ниже.

Поле BC(31 - 30р.) задает режим работы по внешним шинам при работе с общей памятью:

31-й р. (GBRE) - разрешение передачи контроля над глобальной шиной внешнему устройству при запросе от него:

0 - контроль не передаётся;

1 - контроль передаётся.

30-й р. (LBRE) - разрешение передачи контроля над локальной шиной внешнему устройству при запросе от него:

0 - контроль не передаётся;

1 - контроль передаётся.

Поле TIMER pin control (29 - 26р.) определяет режим работы внешнего вывода TIMER:

29-й р. (TEN) - разрешение выдачи на внешний вывод TIMER:

0 - вывод находится в высокоэпидансном состоянии;

1 - разрешается выдача.

28-26 рр. (TM2 - TM0) - определение того, что выдаётся на внешний вывод TIMER:

TM2	TM1	TM0	Состояние вывода TIMER
0	0	0	логическая единица
0	0	1	логический ноль
0	1	0	$\overline{INT\_T0 \oplus TIMER}$
0	1	1	$\overline{INT\_T1 \oplus TIMER}$
1	0	0	$\overline{INT\_T0}$
1	0	1	$\overline{INT\_T1}$
1	1	0	INT_T0
1	1	1	INT_T1

Примечание: *INT_T0 и INT_T1 - сигналы запроса на прерывание при обнулении таймеров T0 и T1.*

Поле CP1 cont.(25 - 23 р.) определяет работу коммуникационного порта 1:

25-й р. (CP1I) -бит внешней инициализации канала ввода порта 1 (устанавливается/сбрасывается только командой формата 2.3):

0 - инициализация канала ввода порта 1 (сколько 64-разрядных слов принять и по какому адресу записать в память) осуществляется программно;

1 - инициализация канала ввода порта 1 осуществляется по первому принятому слову;.

24-й р. (ICH1) -бит останова канала ввода порта 1:

0 - разрешение принимать данные по этому порту;

1 - останов канала ввода порта.

23-й р. (OCH1) - бит останова канала вывода порта 1:

0 - разрешение выдавать данные по этому порту;

1 - останов канала вывода, при этом разрешается захват

шины данных портом другого процессора NM6403.

Поле CP0 cont.(22 - 20 р.) определяет работу коммуникационного порта 0:

22-й р. (CP1I) -бит внешней инициализации канала ввода порта 0 (устанавливается/сбрасывается только командой формата 2.3):

0 - инициализация канала ввода порта 0 (сколько 64-разрядных слов принять и по какому адресу записать в память) осуществляется программно;

1 - инициализация канала ввода порта 0 осуществляется по первому принятому слову;.

21-й р. (ICH0) -бит останова канала ввода порта 0:

0 - разрешение принимать данные по этому порту;

1 - останов канала ввода порта.

20-й р. (OCH0) - бит останова канала вывода порта 0:

0 - разрешение выдавать данные по этому порту;

1 - останов канала вывода, при этом разрешается

захват шины данных портом другого процессора NM6403.

Поле T0C (19 - 18 р.) управляет таймером 0:

19-й р. (TM0) - разрешение работы таймера T0 в непрерывном режиме:

0 - при обнулении таймера возникает запрос на прерывание, больше никаких действий не производится;

1 - при обнулении таймера возникает запрос на прерывание, восстанавливается содержимое T0 и продолжается его работа.

18-й р. (TE0) - разрешение работы таймера T0 (устанавливается/сбрасывается только командой формата 2.3):

0 - нет счета, если бит TM0 равен нулю;

1 - есть счет независимо от бита TM0.

Поле T1C (17 - 16 р.) управляет таймером 1:

17-й р. (TM1) - разрешение работы таймера T1 в непрерывном режиме:

0 - при обнулении таймера возникает запрос на прерывание, больше никаких действий не производится;

1 - при обнулении таймера возникает запрос на прерывание, восстанавливается содержимое T1 и

продолжается его работа.

16-й р. (TE1) - разрешение работы таймера T1 (устанавливается/сбрасывается только командой формата 2.3):

0 - нет счета, если бит TM1 равен нулю;

1 - есть счет независимо от бита TM1.

Поле FCL (15 - 14 р.) управляет очисткой WFIFO или AFIFO:

15-й р. (WFCL) - очистка WFIFO:

0 - нет очистки;

1 - есть очистка.

14-й р. (AFCL) - очистка AFIFO:

0 - нет очистки;

1 - есть очистка.

Поле INTERRUPT MASKS (13 - 4 р.) содержит маски на каждое прерывание: если какой либо бит маски равен единице, то соответствующее ему прерывание разрешено, если равен нулю - запрещено.

13-й р. (T0M) - маска прерывания по таймеру T0.

12-й р. (SPM) - маска прерывания по переполнению при выполнении арифметической операции, заданной скалярной командой.

11-й р. (VPM) - маска прерывания по запрещённой векторной команде.

10-й р. (INTM) - маска прерывания по внешнему прерыванию.

9-й р. (IC1M) - маска прерывания по завершению ввода по коммуникационному порту 1.

8-й р. (IC0M) - маска прерывания по завершению ввода по коммуникационному порту 0.

7-й р. (OC1M) - маска прерывания по завершению вывода по коммуникационному порту 1.

6-й р. (OC0M) - маска прерывания по завершению вывода по коммуникационному порту 0.

5-й р. (T1M) - маска прерывания по таймеру T1.

4-й р. (ST) - маска пошагового прерывания.

Поле FLAGS (3 - 0 р.) содержит признаки процессора NM6403:

3-й р. (N) - признак знака;

2-й р. (Z) - признак нуля;

1-й р. (V) - признак переполнения;

0-й р. (C) - признак переноса.

Рис. 3-3. Регистр запросов на прерывание и ПДП INTR (32 разряда)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS		DMAR				PS		AFIFO_VAL								RAM_VAL						VPF				INTREQ							
GBS	LBS	IC1DR	IC0DR	OC1DR	IC0DR	CP1S	CP0S	EMPTY	[4:0]							[4:0]						EMPTA	FULLA	EMPTW	FULLW	T0R	SPR	VPR	INR	IC1R	IC0R	OC1R	OC0R	T1R
0	1	0	0	1	1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	1	0	1	0	0	0	0	0	0	0	0	0

Регистр INTR программно доступен только на чтение. Функциональное назначение разрядов регистра INTR приведено ниже. Внизу на Рис. 3-3 указано их состояние после системного сброса.

Поле BS (31 - 30 р.) содержит информацию о том, принадлежат ли процессору NM6403 нулевые банки памяти на локальной и глобальной шинах в данный момент:

31-й р. (GBS) - состояние глобальной шины:

0 - данная шина принадлежит процессору NM6403 при необходимости обращения по ней в нулевой банк памяти;

1 - данная шина процессору NM6403 не принадлежит.

30-й р. (LBS) - состояние локальной шины:

0 - данная шина принадлежит NM6403 при необходимости обращения по ней в нулевой банк памяти;

1 - данная шина процессору NM6403 не принадлежит.

Поле DMAR (29 - 26 р.) содержит запросы на ПДП от коммуникационных портов:

29-й р. (IC1DR) - запрос на ПДП от канала ввода порта 1.

28-й р. (IC0DR) - запрос на ПДП от канала ввода порта 0.

27-й р. (OC1DR) - запрос на ПДП от канала вывода порта 1.

26-й р. (OC0DR) - запрос на ПДП от канала вывода порта 0.

При возникновении запроса на ПДП от каналов ввода в соответствующих разрядах 29 и 28 регистра INTR устанавливается единица, которая сбрасывается, когда данное ПДП разрешается.

При возникновении запроса на ПДП от каналов вывода в соответствующих разрядах 27 и 26 регистра INTR устанавливается ноль, который меняется на единицу, когда данное ПДП разрешается.

Поле PS (25 - 24 р.) содержит информацию о состоянии коммуникационных портов 0 и 1:

25-й р. (CP1S) - направление передачи ком. порта 1:

0 - порт в режиме выдачи;

1 - порт в режиме приема.

23-й р. (CP0S) - направление передачи ком. порта 0:

0 - порт в режиме выдачи;

1 - порт в режиме приема.

Поле AFIFO_VAL (23 - 18 р.) говорит о том, сколько 64-разрядных слов должно быть в AFIFO после того, как закончатся все векторные команды, на данный момент попавшие в конвейер процессора NM6403:

23-й р. (EMPTY) - признак пустоты AFIFO:

0 - AFIFO пусто;

1 - AFIFO не пусто, и число записанных в нем слов определяется разрядами 22 - 18.

22-18 рр. ([4:0]) - число записанных в AFIFO слов:

00000 - одно 64-разрядное число;

00001 - два 64-разрядных числа;

. . . . . .

11111 - тридцать два 64-разрядных числа.

Поле RAM_VAL (17 - 13 р.) говорит о том, сколько 64-разрядных слов должно быть в ОЗУ данных RAM после того, как закончатся все векторные команды, на данный момент попавшие в конвейер процессора NM6403:

00000 - одно 64-разрядное число;

00001 - два 64-разрядных числа;

. . . . . .

11111 - тридцать два 64-разрядных числа.

Поле VPF (12 - 9 р.) содержит флаги WFIFO и AFIFO векторного процессора:

12-й р. (EMPTA) - флаг очистки AFIFO:

0 - AFIFO содержит информацию;

1 - AFIFO пусто.

11-й р. (FULLA) - флаг заполнения AFIFO:

0 - AFIFO не заполнено полностью;

1 - AFIFO полностью заполнено.

10-й р. (EMPTW) - флаг очистки WFIFO:

0 - WFIFO содержит информацию;

1 - WFIFO пусто.

9-й р. (FULLW) - флаг заполнения WFIFO:

0 - WFIFO не заполнено полностью;

1 - WFIFO полностью заполнено.

Поле INTREQ (8 - 0 р.) содержит запросы на прерывание:

8-й р. (T0R) - запрос на прерывание от таймера T0.

7-й р. (SPR) - запрос на прерывание по переполнению при выполнении арифметической операции, заданной скалярной командой.

6-й р. (VPR) - запрос на прерывание по запрещённой векторной команде.

5-й р. (INR) - запрос на прерывание внешнее.

4-й р. (IC1R) - запрос на прерывание по завершению ввода по коммуникационному порту 1.

3-й р. (IC0R) - запрос на прерывание по завершению ввода по коммуникационному порту 0.

2-й р. (OC1R) - запрос на прерывание по завершению вывода по коммуникационному порту 1.

1-й р. (OC0R) - запрос на прерывание по завершению вывода по коммуникационному порту 0.

0-й р. (T1R) - запрос на прерывание по таймеру T1.

При возникновении запроса на прерывание в соответствующем бите INTR устанавливается единица, которая сбрасывается, когда данный запрос начинает обслуживаться.

Регистры управления интерфейсом LMICR и GMICR (32 разряда)

Ниже представлен формат регистров управления интерфейсом с глобальной и локальной шиной GMICR и LMICR, а в Табл. 3-1 приведено функциональное описание отдельных полей. Данный формат GMICR и LMICR позволяет подключать к любой из внешних шин до двух банков памяти, различающихся типом, страничной организацией и динамическими параметрами. Регистры GMICR и LMICR программно доступны на запись и чтение, после системного сброса все их разряды сброшены в ноль. Для более полной информации см. главу 9.

Рис. 3-4. Формат регистра управления интерфейсом с глобальной (локальной) шиной GMICR (LMICR)

31 30 29 28	27 26 25 24	23 22 21 20	19 18	17 16	15 14 13 12 11	10 9 8 7 6	5 4	3	2	1 0
BOUND	PAGE1	PAGE0	TYPE1	TYPE0	TIME1	TIME0	TRAS	EXRDY1	EXRDY0	SHMEM

Примечание: *На Рис. 3-4 приведена мнемоника, соответствующая регистру управления глобальной шиной. Для регистра управления локальной шиной к имени поля добавляется префикс “L”..*

Табл. 3-1. Описание полей регистра управления интерфейсом с глобальной (локальной) шиной GMICR (LMICR)

Номера разрядов	Обозначение[1)]	Функциональное назначение
31 - 28	BOUND	Разбиение адресного пространства шины на банки 0 и 1 (см. Табл. 9-3 для глобальной шины и Табл. 9-4 для локальной шины).
27 - 24	PAGE1	Размер страницы памяти банка 1 (см. Табл. 9-5)[2)].
23 - 20	PAGE0	Размер страницы памяти банка 0 (см. Табл. 9-5).
19 - 18	TYPE1	Тип памяти банка 1[3)]: 00 - SRAM; 01 - DRAM 1-го типа (сигнал $\overline{RAS1}$ переводится в пассивное состояние при обращении к новой странице памяти); 10 - DRAM 2-го типа (сигнал $\overline{RAS1}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот, при циклах Idle); 11 - DRAM 3-го типа (сигнал $\overline{RAS1}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот);
17 - 16	TYPE0	Тип памяти банка 0: 00 - SRAM; 01 - DRAM 1-го типа (сигнал $\overline{RAS0}$ переводится в пассивное состояние при обращении к новой странице памяти); 10 - DRAM 2-го типа (сигнал $\overline{RAS0}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот, при циклах Idle); 11 - DRAM 3-го типа (сигнал $\overline{RAS0}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот);
15 - 11	TIME1	Временные параметры циклов обращения к памяти банка 1[2)].
10 - 6	TIME0	Временные параметры циклов обращения к памяти банка 0.

Таблица 3-1. Описание полей регистра управления интерфейсом с глобальной (локальной) шиной GMICR (LMICR)(Продолжение)

Номера разрядов	Обозначение [1)]	Функциональное назначение
5 – 4	TRAS	Длительность активного уровня сигнала $\overline{RAS}$ при регенерации DRAM[4)].
3	RDY1	Условие окончания цикла обращения к памяти банка 1[2)]: 0 – только по внутреннему счётчику; 1 – по внутреннему счётчику и с учётом внешнего сигнала готовности[5)].
2	RDY0	Условие окончания цикла обращения к памяти банка 0: 0 – только по внутреннему счётчику; 1 – по внутреннему счётчику и с учётом внешнего сигнала готовности[5)].
1 – 0	SHMEM	Конфигурация внешней шины: 00 – многопроцессорная конфигурация 1-го типа (банк 0 – “общий”, банк 1 – “общий”); 01 – многопроцессорная конфигурация 2-го типа (банк 0 – “свой”, банк 1 – “общий”); 10 – многопроцессорная конфигурация 3-го типа (банк 0 – “свой”, банк 1 – “чужой”); 11 – однопроцессорная конфигурация.

Примечание:

1) Приведена мнемоника, соответствующая регистру управления интерфейсом с глобальной шиной. Для регистра управления локальной шиной к имени поля добавляется префикс “L”.

2) Если банк 1 отсутствует, то значения полей PAGE1, TIME1 и RDY1 безразличны.

3) Если банк 1 отсутствует и в поле TYPE0 указан тип DRAM, то значение поля TYPE1 безразлично. Если же банк 1 отсутствует и в поле TYPE0 указан тип SRAM, то в поле TYPE1 следует так же указывать тип SRAM. В противном случае будут выполняться циклы регенерации несуществующей DRAM.

4) Если в обоих полях TYPE0 и TYPE1 указан тип SRAM, то значение поля TRAS безразлично.

5) Если в поле SHMEM указана одна из многопроцессорных конфигураций, то значения полей RDY0 и RDY1 безразличны.

4 Векторный узел

В данной главе приводится описание векторного узла процессора NM6403, который используется при обработке векторных данных.

4.1 Структура векторного узла

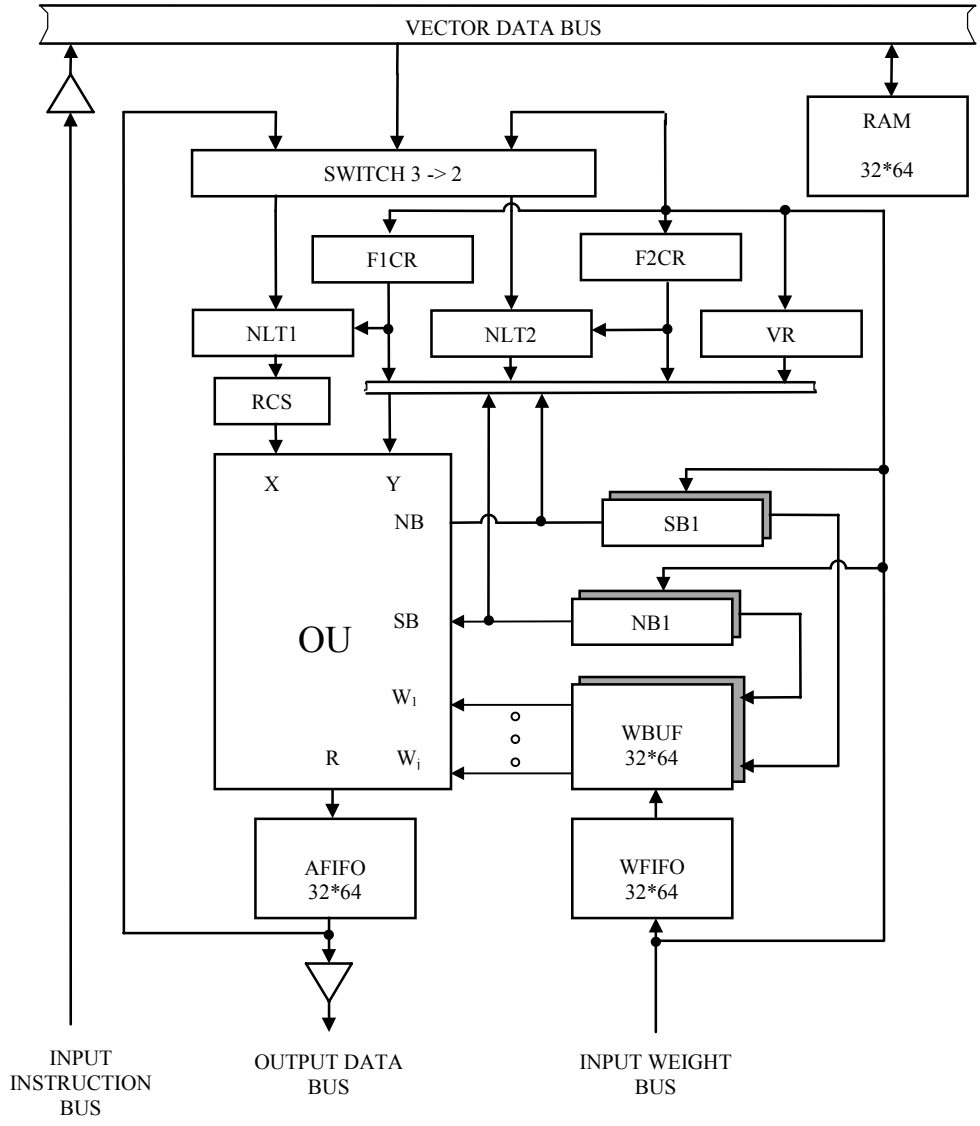
Векторный узел (VU) является основным вычислительным узлом процессора NM6403. VU ориентирован на обработку данных произвольной разрядности от 1 до 64 разрядов, упакованных в 64-разрядные слова.

Структурная схема VU представлена на Рис. 4-1. Основными узлами VU являются:

OU	Операционное устройство
RCS	Циклический сдвигатель вправо
NLT1, NLT2	Нелинейные преобразователи
SWITCH 3→2	Коммутатор 3 в 2
WBUF и WOPER	Память весовых коэффициентов и операционная память весовых коэффициентов
WFIFO	FIFO весовых коэффициентов
AFIFO	Накопительное FIFO
RAM	Векторный регистр
VR	Регистр порогов
F1CR, F2CR	Регистры управления нелинейными преобразователями
SB1 и SB2	Регистр границ синапсов и операционный регистр границ синапсов
NB1 и NB2	Регистр границ нейронов и операционный регистр границ нейронов

Работа всех перечисленных выше узлов будет описана в следующих разделах.

Рис. 4-1. Структурная схема векторного узла



4.2 Форматы векторных данных

Данные

VU предназначен для обработки целочисленных данных, разрядность которых может быть произвольной и в общем случае лежит в диапазоне от 1 до 64 разрядов. Однако, схемотехническая реализация ряда исполнительных узлов VU накладывает дополнительные ограничения на разрядность данных:

- множители, поступающие на вход X OU и используемые в операциях взвешенного суммирования, должны иметь четную разрядность;

- переменные, для которых вычисляется нелинейная функция

активации, должны иметь не менее двух разрядов.

Целочисленные данные, используемые в арифметических операциях, должны быть представлены в дополнительном коде. Результаты арифметических операций формируются так же в дополнительном коде.

64-разрядные слова упакованных данных

VU ориентирован на обработку массивов данных. Разрядность всех узлов VU и всех внутренних шин процессора NM6403, к которым подключен VU, равна 64. Поэтому с целью повышения производительности NM6403 путем эффективного использования аппаратных ресурсов VU осуществляет обработку целочисленных данных, которые упакованы в 64-разрядные слова с помощью простой конкатенации (см. Рис. 4-2). В общем случае слово упакованных данных представляет собой вектор **D**={D_I … D_1}, содержащий I данных, суммарная разрядность которых равна 64. Причем, в одном слове **D** могут быть упакованы данные, имеющие разную разрядность. Количество данных I, упакованных в одном слове, зависит от их разрядностей и может принимать целочисленное значение в диапазоне от 1 до 64.

В каждом такте VU осуществляет обработку всех данных слова, поступающего на любой из исполнительных узлов VU. При этом исполнительные узлы определяют границы данных в слове по содержимому соответствующих конфигурационных регистров. Разряды NB2, в которых записана 1, соответствуют старшим разрядам данных слова, поступающего на вход Y, а при выполнении арифметических операций и на вход X OU. Разряды SB2, в которых записана 1, соответствуют младшим разрядам данных слова, поступающего на вход X OU при выполнении операции взвешенного суммирования. Если в i-м разряде F1CR записана 1, а в (i+1)-м - 0, то i-й разряд слова, поступающего на вход NLT1, будет являться старшим разрядом одного из данных, упакованных в этом слове. Аналогично F2CR определяет границы данных в слове, поступающем на вход NLT2. В качестве примера (см. Рис. 4-3) приведено содержимое конфигурационных регистров для слова, состоящего из двухразрядного операнда D_1, четырехразрядного операнда D_2, шестиразрядного операнда D_3 и т.д. Здесь ‘x’ в NLTCR1 и NLTCR2 означает, что значение данного разряда не определяет границ данных в слове.

В регистры NB2 и SB2 информация поступает соответственно из регистров NB1 и SB1 при выполнении команды LOAD. Регистры NB1, SB1, F1CR F2CR и VR подключены к шине данных процессора NM6403 - INPUT WEIGHT BUS - и программно доступны по записи.

Рис. 4-2. Формат слова упакованных векторных данных

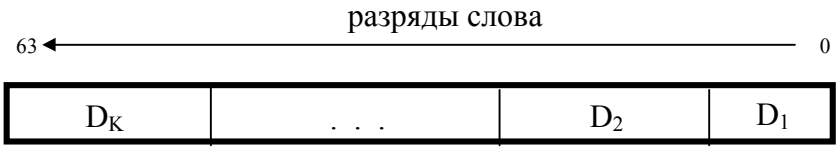
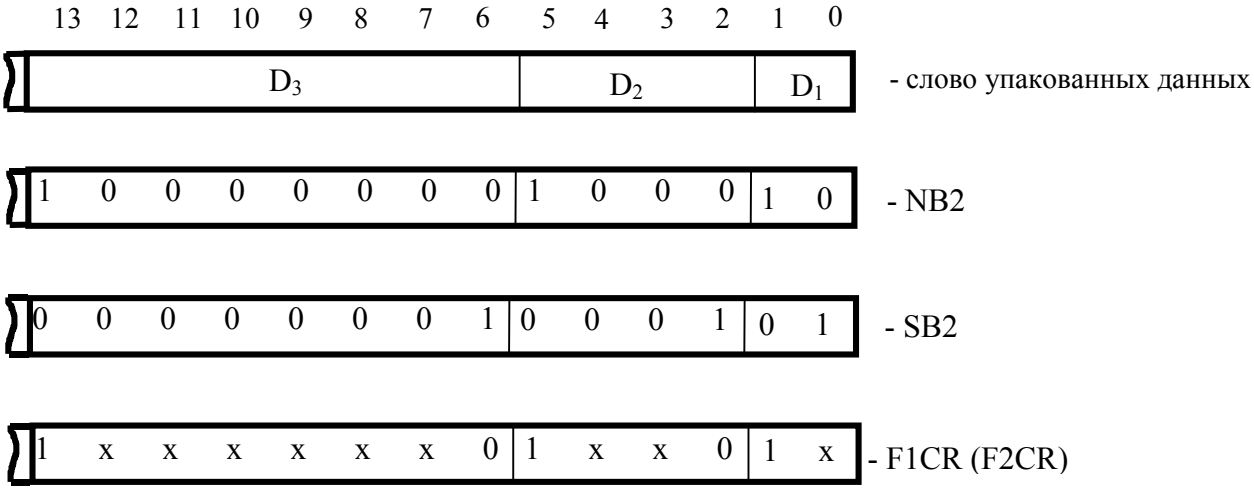


Рис. 4-3. Форматы конфигурационных регистров



Матрицы весовых коэффициентов и вектора слов упакованных данных

Все исполнительные узлы VU позволяют за один такт выполнять операции над 64-разрядными словами упакованных данных. Базовой операцией является операция взвешенного суммирования, при выполнении которой в каждом такте используется матрица весовых коэффициентов, хранящаяся в WOPER. Размерность матрицы весовых коэффициентов может достигать 32 64-разрядных слов упакованных весовых коэффициентов. С целью эффективного использования аппаратных ресурсов VU в процессоре NM6403 предусмотрен механизм подкачки новой матрицы весовых коэффициентов из внешней памяти на фоне выполнения операций с текущим содержимым WOPER. 64-разрядные слова упакованных весовых коэффициентов поступают с одной из внешних шин процессора NM6403 - INPUT WEIGHT BUS - через WFIFO в WM. При этом процесс загрузки WBUF длится 32 такта и управляется содержимым регистров SB1 и NB1, которые задают соответственно количество слов упакованных весовых коэффициентов в матрице весов и границы весовых коэффициентов в этих 64-разрядных словах. Затем по команде LOAD за один такт содержимое WBUF переписывается в WOPER, а содержимое регистров NB1 и SB1 - в регистры NB2 и SB2 соответственно. Одновременно с этим OU может выполнять операции с использованием прежнего содержимого WOPER, NB2 и SB2. Так как процесс загрузки

матрицы весов длится 32 такта, то каждая векторная команда позволяет выполнять за L тактов одни и те же операции над векторами из L 64-разрядных слов упакованных данных. Переменная L задается в одном из полей команды и может принимать целочисленное значение в диапазоне от 1 до 32.

4.3 Операционное устройство OU

OU служит для выполнения арифметических и логических операций над 64-разрядными словами упакованных данных $\mathbf{X}=\{X_K \dots X_1\}$ и $\mathbf{Y}=\{Y_I \dots Y_1\}$, поступающими соответственно на входы X и Y OU, и матрицей весов **W**, которая подается на входы $W_1,\dots,W_J$ в виде J 64-разрядных слов упакованных весовых коэффициентов $\mathbf{W_1}=\{W_{1I} \dots W_{11}\},\dots,\mathbf{W_J}=\{W_{JI} \dots W_{J1}\}$. Результат каждой операции формируется на выходе R в виде 64-разрядного слова упакованных данных $\mathbf{R}=\{R_I \dots R_1\}$. Переменная I может принимать любое целочисленное значение от 1 до 64 в зависимости от содержимого NB2 (I равно количеству единиц в NB2), а переменная J - от 1 до 32 в зависимости от содержимого SB2 (J равно количеству единиц в SB2). Значения, принимаемые переменной K, зависят от типа выполняемой операции: K=I - для арифметических операций, K=J - для операции взвешенного суммирования.

Тип выполняемой операции задается кодом команды. Операции выполняются в конвейере с темпом одна операция за такт. Никаких признаков результатов операций OU не формирует.

OU выполняет три следующих набора операций, каждому из которых соответствует своя программная модель OU (см. Рис. 4-4):

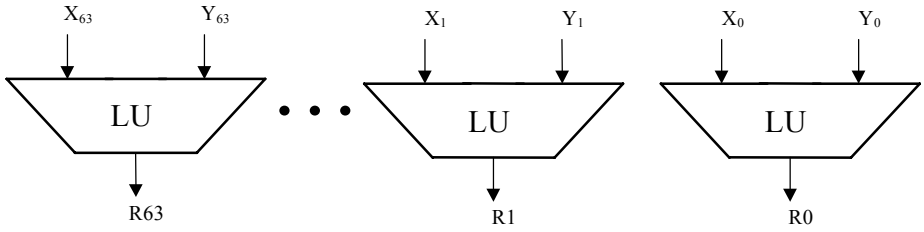
- 16 всевозможных поразрядных логических операций над 64-разрядными операндами, поступающими на входы X и Y. В данном режиме OU представляется в виде 64 параллельно работающих одноразрядных логических устройств LU, выполняющих одну и ту же логическую операцию над каждой парой разрядов входных операндов (см. Рис. 4-4.а). Содержимое регистров SB2 и NB2 не влияет на результаты логических операций.

- 4 арифметические операции над словами **X** и **Y**: $X_i + Y_i$; $X_i - Y_i$; $X_i + 1$; $X_i - 1$, где i=1,...,I. При выполнении данных арифметических операций OU можно представить в виде I параллельно включенных арифметических устройств AU, выполняющих одну и ту же арифметическую операцию над соответствующими данными слов **X** и **Y** (см. Рис. 4-4.б). Количество и разрядности AU и соответствующих данных в словах **X**, **Y** и **R** совпадают и определяются

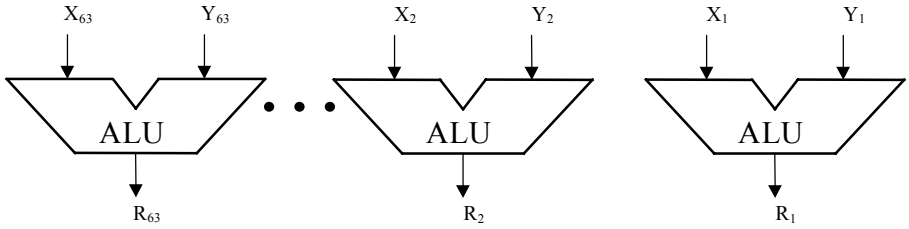
содержимым регистра NB2. Содержимое регистра SB2 не влияет на результаты данных арифметических операций.

- Сложение вектора **Y** с произведением матрицы весов **W :** $R_i = Y_i + \Sigma W_{ij} * X_j$, где i=1,...,I, j=1,...,J. При выполнении данной операции, получившей также название взвешенного суммирования, OU можно представить в виде I параллельно работающих схем, каждая из которых содержит J умножителей и один (J+1)-операндный сумматор (см. Рис. 4-4.c). Количество I и разрядности соответствующих операндов в словах **Y**, $\mathbf{W_J}$,...,$\mathbf{W_1}$ и **R** совпадают и определяются содержимым регистра NB2, а количество J и разрядности операндов в векторе **X** определяются содержимым регистра SB2. Умножители в OU реализованы по модифицированному алгоритму Бута, поэтому разрядность каждого операнда, входящего в состав слова **X,** должна быть четной.

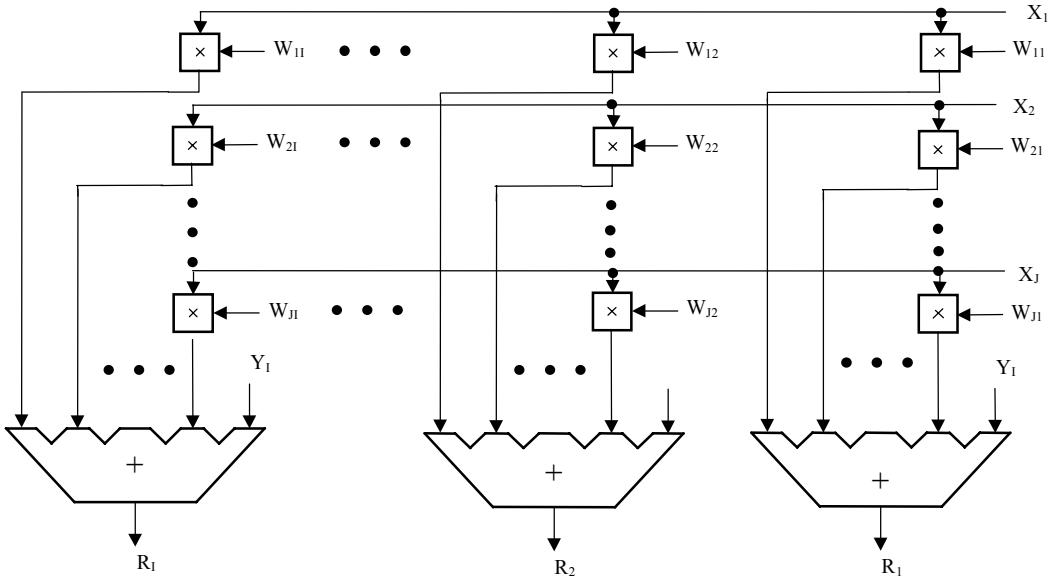
Рис. 4-4. Программные модели OU в различных режимах его работы



а) Программная модель OU при выполнении логических операций



б) Программная модель OU при выполнении арифметических операций



c) Программная модель OU при выполнении взвешенного суммирования

4.4 Циклический сдвигатель вправо RCS

В зависимости от кода команды 64-разрядные слова, поступающие на вход X OU, проходят через RCS без изменений или циклически сдвигаются вправо на один разряд. За один такт выполняется сдвиг одного слова как единого операнда, не зависимо от количества данных в слове.

4.5 Нелинейные преобразователи NLT1, NLT2 и их регистры управления F1CR, F2CR

NLT1 и NLT2 служат для вычисления нелинейных функций активации над 64-разрядными словами упакованных данных (см. Рис. 4-5). Для каждого нелинейного преобразователя в VU предусмотрен свой программно доступный регистр управления (F1CR для NLT1 и F2CR для NLT2), содержимое которого в зависимости от кода выполняемой команды может определять количество и разрядности данных, составляющих обрабатываемое слово, а также максимальное абсолютное значение результата вычисления функции насыщения H (см. Рис. 4-5.б) для каждого составляющего слова.

NLT1 и NLT2 абсолютно идентичны, поэтому для обозначения нелинейного преобразователя далее будет использоваться мнемоника NLTx. Все сказанное о NLTx будет относиться как к NLT1, так и к NLT2. Аналогично, мнемоника FxCR используется ниже для обозначения одного из регистров F1CR или F2CR.

В зависимости от кода команды NLTx выполняет за один такт одну из трех операций над 64-разрядным словом упакованных данных:

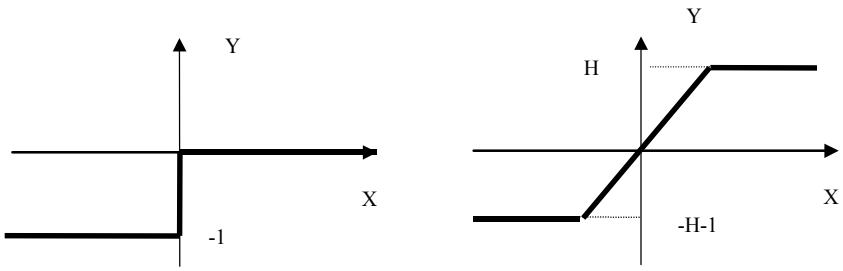
- пропускает на выход входное слово без изменений независимо от содержимого регистра FxCR;
- вычисляет пороговую функцию для каждого операнда входного слова (см. Рис. 4-5.а). При этом содержимое регистра FxCR определяет количество и разрядности данных входного и выходного слова. Формат регистра FxCR при вычислении пороговой функции представлен на Рис. 4-3;
- вычисляет функцию насыщения для каждого операнда входного слова (см. Рис. 4-5.б). При этом содержимое регистра FxCR определяет не только количество и разрядности операндов входного и выходного слов упакованных данных, но и максимальные абсолютные значения каждого операнда выходного слова. Формат регистра FxCR при вычислениии функции насыщения представлен на Рис. 4-3.

Из представленных (см. Рис. 4-3) форматов регистра FxCR следует, что минимальная разрядность операндов, составляющих обрабатываемое слово, равна двум. Поэтому за один такт NLTx может обрабатывать от 1 до 32 данных различной разрядности, суммарная разрядность которых равна 64.

Необходимо отметить, что при прохождении слова упакованных данных через NLTx независимо от выполняемой им операции

количество и разрядности операндов, составляющих 64-разрядное слово, не изменяются. Уменьшение абсолютных значений входных операндов в результате вычисления нелинейных функций активации сопровождается расширением знаковых разрядов операндов. Таким образом, NLTx служит только для уменьшения абсолютных значений входных операндов, а не для уменьшения их разрядности.

Рис. 4-5. Нелинейные функции активации, вычисляемые NLTx



а) Пороговая функция б) Функция насыщения

4.6 Памяти весовых коэффициентов WBUF и WOPER

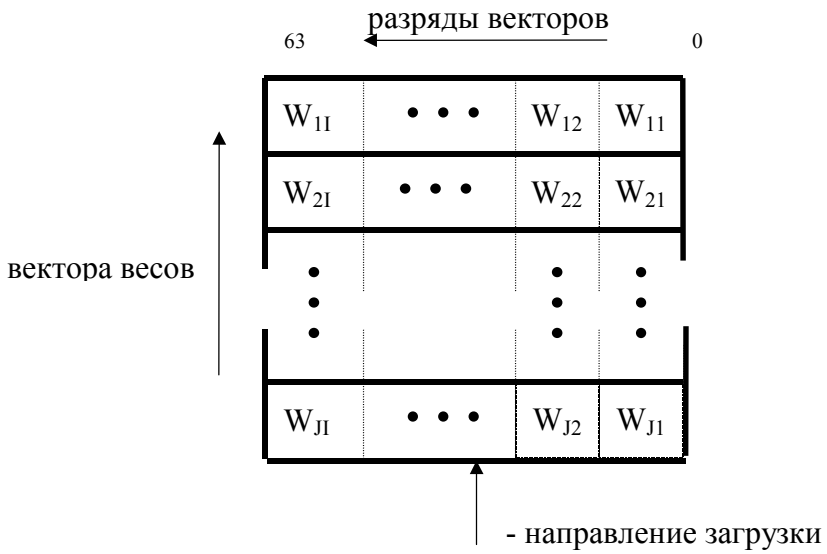
Память весовых коэффициентов состоит из двух матриц ячеек памяти WBUF и WOPER, каждая из которых имеет емкость 32x64 бита и позволяет хранить матрицу весов **W** в виде J 64-разрядных слов упакованных весовых коэффициентов:
$\mathbf{W_1}=\{W_{1I} \ldots W_{11}\},\ldots,\mathbf{W_J}=\{W_{JI} \ldots W_{J1}\}$.

WOPER служит для хранения матрицы весов, используемой в операциях взвешенного суммирования, выполняемых OU. Выходы всех ячеек WOPER соединены непосредственно с соответствующими входами OU, а их входы - с выходами соответствующих ячеек WBUF. Благодаря этому запись во все ячейки WOPER осуществляется за один такт по команде LOAD. При этом содержимое WBUF целиком копируется в WOPER. Одновременно с этим содержимое регистров SB1 и NB1 копируется в регистры SB2 и NB2.

WBUF служит для подкачки из WFIFO новой матрицы весов на фоне выполнения OU текущих операций взвешенного суммирования с использованием старой матрицы весов, хранящейся в WOPER. Загрузка матрицы весов в WBUF инициируется одной командой и осуществляется за 32 такта путем последовательной записи J 64-разрядных слов упакованных весовых коэффициентов, выбираемых из WFIFO. При этом количество загружаемых слов J определяется содержимым SB1 (J равно количеству единиц в SB1). В режиме записи WBUF работает по аналогии с памятью

магазинного (стекового) типа. Причем первым загружается слово $\mathbf{W_1}=\{W_{1I} \ldots W_{11}\}$, а последним - слово $\mathbf{W_J}=\{W_{JI} \ldots W_{J1}\}$. Во всех этих словах количество весовых коэффициентов и границы между ними совпадают и определяются содержимым регистра NB1. Последовательность загрузки и расположение весовых коэффициентов в WBUF иллюстрируется на Рис. 4-6.

Рис. 4-6. Формат матрицы весов WBUF



4.7 FIFO весовых коэффициентов (WFIFO)

Двухпортовое WFIFO имеет емкость 32x64 бит и используется в качестве накопительного буфера в процессе подкачки матрицы весов в WBUF из внешней памяти. Запись и чтение из WFIFO ведется 64-разрядными словами упакованных весовых коэффициентов. Загрузка WFIFO осуществляется через внутреннюю шину данных процессора NM6403 - INPUT WEIGHT BUS.

WFIFO введено в схему VU с целью повышения эффективности использования шин NM6403 в процессе подкачки весовых коэффициентов в WBUF из внешней памяти. Как отмечалось в предыдущем разделе, загрузка матрицы весов в WBUF выполняется за 32 такта. При этом в зависимости от содержимого теневого регистра SB1 количество загружаемых слов J может принимать целочисленное значение в диапазоне от 1 до 32. В отсутствии WFIFO непосредственная загрузка матрицы весов из внешней памяти в WBUF, проводимая при малых значениях J, приводила бы к значительным простоям в работе одной из шин. Поэтому процесс загрузки матрицы весов из внешней памяти в WBUF выполняется процессором NM6403 в два этапа. Первоначально, по команде загрузки WFIFO в него записывается массив из N 64-разрядных слов

упакованных весовых коэффициентов, выбираемых из в внешней памяти. Значение N задается кодом команды и находится в диапазоне от 1 до 32. Затем по команде загрузки матрицы весов в WBUF J 64-разрядных слов упакованных весовых коэффициентов переписываются из WFIFO в WBUF. Следует отметить, что нецелесообразно использовать значения N, меньшие, чем значения J, так как это может привести к зависанию процессора NM6403. При N>>J в WFIFO можно записать сразу несколько матриц весов, которые затем будут последовательно переписываться в WBUF, причём команды загрузки WFIFO и WBUF могут существенно отстоять друг от друга в общем потоке команд.

Таким образом, WFIFO служит для согласования асинхронных процессов загрузки матрицы весов в WBUF и чтения слов упакованных весовых коэффициентов из внешней памяти. Контроль за данными процессами может осуществляться программно путем анализа флагов “WFIFO пустое” и “WFIFO полное”, которые фиксируются в регистре INTR NM6403. WFIFO может быть программно очищено путем установки в 1 соответствующего бита PSWR.

4.8 Накопительное FIFO (AFIFO)

Двухпортовое AFIFO емкостью 32x64 бита используется в VU в качестве аккумулятора и служит для хранения одного вектора 64-разрядных слов упакованных данных, которые являются результатом выполнения последней векторной операционной команды. В зависимости от кода команды содержимое AFIFO может пересылаться в следующие приемники:

- во внешнюю память через шину выходных данных процессора NM6403;
- одновременно во внешнюю память и в RAM через выходную шину - OUTPUT DATA BUS, шину команд процессора NM6403 - INPUT INSTRUCTION BUS - и шину векторных данных VU-VECTOR DATA BUS;
- на входы исполнительных узлов VU по цепи обратной связи, проходящей через коммутатор 3 в 2;
- одновременно во внешнюю память и на входы исполнительных узлов VU по цепи обратной связи;
- одновременно во внешнюю память, в RAM и на входы исполнительных узлов VU.

Выходы AFIFO и цепь обратной связи VU отделены от шины выходных данных процессора NM6403 буфером. Поэтому при

пересылке содержимого AFIFO только на входы исполнительных узлов VU шина выходных данных NM6403 остается свободной.

Необходимо отметить, что AFIFO предназначено для хранения только одного вектора слов упакованных данных. Если в последовательности выполняемых команд между двумя векторными операционными командами не встречается ни одной команды чтения из AFIFO и если вторая векторная операционная команда также не требует чтения из AFIFO, то процессор NM6403 отработает эту команду, как команду NOP, и сформирует при этом внутреннее прерывание по неправильной последовательности команд.

Контроль за содержимым AFIFO может осуществляться программно путем анализа флагов “AFIFO пустое” и “AFIFO полное”, которые фиксируются в регистре INTR. AFIFO может быть программно очищено путем установки в 1 соответствующего бита PSWR.

4.9 Векторный регистр RAM

RAM представляет собой однопортовую память типа FIFO емкостью 32x64 бита, которая подключена к шине векторных данных VU - VECTOR DATA BUS. Основное отличие RAM от обычного FIFO заключается в том, что после чтения из RAM его содержимое не изменяется.

RAM служит для хранения одного вектора 64-разрядных слов упакованных данных, которые могут передаваться на исполнительные узлы VU через коммутатор 3 в 2. В зависимости от кода команды слова, записываемые в RAM, выбираются из внешней памяти или из AFIFO. В обоих случаях слова данных поступают на вход RAM через шину команд NM6403 - INPUT INSTRUCTION BUS - и шину векторных данных VU - VECTOR DATA BUS. Последняя отделена от шины команд однонаправленным буфером с высокоимпедансным состоянием. Это позволяет осуществлять пересылку содержимого RAM в исполнительные узлы VU, не занимая шину команд процессора NM6403.

Необходимо отметить, что RAM предназначен для хранения только одного вектора слов упакованных данных, поэтому любая запись в RAM приводит к потере ее прежнего содержимого.

4.10 Коммутатор 3 в 2

Коммутатор 3 в 2 обеспечивает выбор двух источников векторных данных, поступающих на входы исполнительных узлов VU. В зависимости от кода команды через коммутатор 3 в 2 на вход каждого нелинейного преобразователя могут поступать следующие вектора слов упакованных данных:

- вектора данных с нулевыми значениями всех разрядов;
- содержимое RAM с шины векторных данных VU - VECTOR DATA BUS;
- содержимое AFIFO по цепи обратной связи VU;
- вектора данных из внешней памяти.

В последнем случае векторные данные поступают на входы коммутатора по одной из двух шин зависимости от того, есть ли чтение из RAM операнда в тот же момент времени: по шине команд - INPUT INSTRUCTION BUS - если нет или по шине входных данных процессора NM6403 - INPUT WEIGHT BUS- если есть.

В случае, когда вектор слов упакованных данных выбирается из внешней памяти, то количество слов в нем задается кодом команды. Если же вектор выбирается из RAM или AFIFO, то количество слов в нем равно числу 64-разрядных слов, записанных в эти памяти по предыдущим командам. Последнее связано с тем, что содержимое RAM или AFIFO всегда считывается целиком.

Кроме функций коммутации пакетов векторных данных коммутатор 3 в 2 может выполнять функции поразрядного маскирования векторных данных, пропускаемых на исполнительные узлы VU. Если код векторной команды задает операцию маскирования, то на вход NLT1 поступает результат поразрядной логической операции И над пропускаемым на NLT1 словом упакованных данных и 64-разрядной маской, а на вход NLT2 - результат поразрядной логической операции И над пропускаемым на NLT2 словом упакованных данных и инверсией 64-разрядной маски. При этом векторам слов упакованных данных соответствует вектор масок, который выбирается одновременно с векторами слов упакованных данных из тех же источников и по тем же правилам. Фактически коммутатор 3 в 2 является полным коммутатором 3 в 3, у которого третьим выходом является маска.

Код команды может задавать одновременную обработку до трех векторов упакованных данных. Если при этом выбираются пакеты с разным количеством слов, то NM6403 отработает эту команду, как команду NOP, и сформирует внутреннее прерывание по запрещенной векторной команде.

4.11 Регистр порогов VR

Регистр порогов VR служит для хранения 64-разряного слова упакованных порогов или смещений. Существуют команды, по которым при выполнении операции взвешенного суммирования

содержимое VR подается на вход Y OU.

5 Методы адресации памяти

5.1 Методы адресации команд

При выборе очередной команды из внешней памяти её адрес может вычисляться следующими способами:

- по содержимому счётчика адреса команд с его икрементацией на 2 (выборка команд идёт по 64 разряда) на линейных участках программы;

- по содержимому AR-регистра или GR-регистра для команд перехода/перехода к подпрограмме с соответствующей модификацией счётчика адреса команд (PC):

1) PC <- ARi, i = 0, …, 7;
2) PC <- ARi + GRi, i = 0, …, 7;
3) PC <- GRi, i = 0, …, 7;
4) PC <- PC + GRi;

- по содержимому AR-регистра или PC с непосредственно заданным в команде смещением в виде 32-разрядной константы (Const32) для команд перехода/перехода к подпрограмме со смещением с соответствующей модификацией PC:

1) PC <- ARi + Const32;
2) PC <- Const32;
3) PC <- PC + Const32.

- по содержимому стека адресов возврата, находящемуся во внешней памяти и адресуемому по содержимому указателя стека (SP) с его декрементацией на 2 для команд возврата из подпрограммы/прерывания с соответствующей модификацией PC: PC <- [--SP].

Более подробно о методах адресации команд см. главу 7.

5.2 Методы адресации скалярных данных

Процессор NM6403 поддерживает обмен данными с внешней памятью как скалярными, так и векторными данными, задаваемый соответственно скалярными или векторными командами. Скалярные команды задают единичную пересылку регистр <- память или память <- регистр. В зависимости от кода регистра (т.е. от его разрядности) обмен с памятью осуществляется по 32 или 64 разряда, при этом исполнительный адрес памяти (Address) может вычисляться следующими способами:

- по содержимому AR-регистра или GR-регистра с соответствующей модификацией AR-регистра:

1) Address <- GRi, i = 0, …,7; нет модификации адресных регистров.
2) Address <- ARi + GRi; ARi <- ARi + GRi, i = 0, …,7.
3) Address <- GRi; ARi <- GRi, i = 0, …,7.
4) Address <- ARi, i = 0, …,7; нет модификации адресных регистров.
5) Address <- ARi; ARi <- ARi + GRi, i = 0, …,7.
6) Address <- ARi - a; ARi <- ARi - a, i = 0, …,7; a = 1 при адресации 32-разрядных данных, a = 2 при адресации 64-разрядных данных.
7) Address <- ARi; ARi <- ARi + a, i = 0, …,7; a = 1 при адресации 32-разрядных данных, a = 2 при адресации 64-разрядных данных.

- по содержимому AR-регистра с непосредственно заданным в команде смещением в виде 32-разрядной константы (Const32):

1) Address <- Const32; нет модификации адресных регистров.
2) Address <- ARi + Const32; ARi <- ARi + Const32, i = 0, …,7.
3) Address <- Const32; ARi <- Const32, i = 0, …,7.

Более подробно о методах адресации скалярных данных см. главу 7. О методах адресации векторных данных см. раздел 5.3.

5.3 Методы адресации векторных данных

Векторные команды задают обмен с внешней памятью блоками по N 64-разрядных слов, где N определяется кодом команды и может меняться в диапазоне от 1 до 32. Исполнительный адрес памяти при этом вычисляется по тем же правилам, что и для скалярных команд, но N раз и N раз будет модифицирован соответствующий адресный регистр. Отличие состоит лишь в том, что для векторных команд нельзя использовать адресацию с помощью непосредственно заданного в команде смещения в виде 32-разрядной константы. Все остальные методы адресации по содержимому AR-регистра или GR-регистра с соответствующей модификацией AR-регистра остаются доступными:

1) Address <- GRi, i = 0, …,7; нет модификации адресных регистров.
2) Address <- ARi + GRi; ARi <- ARi + GRi, i = 0, …,7.
3) Address <- GRi; ARi <- GRi, i = 0, …,7.
4) Address <- ARi, i = 0, …,7; нет модификации адресных регистров.
5) Address <- ARi; ARi <- ARi + GRi, i = 0, …,7.
6) Address <- ARi - 2; ARi <- ARi - 2, i = 0, …,7.
7) Address <- ARi; ARi <- ARi + 2, i = 0, …,7.

Более подробно о методах адресации векторных данных см. главу 7.

5.4 Особенности работы с системным стеком и стеками пользователя

Процессор NM6403 имеет специальный регистр-указатель стека SP, с помощью которого организуется стек во внешней памяти. Этот стек используется в качестве системного при вызовах подпрограмм, обработке прерываний или возврате из них. Пользователь имеет возможность использовать его для передачи параметров между процедурами или для временного хранения своих данных, а также организовать свой стек, используя в качестве указателя стека любой из адресных регистров - AR0 - AR6.

Системный стек

Указатель стека SP - это 32-разрядный регистр, который содержит адрес вершины системного стека. Стек наполняется от младших адресов к старшим. SP всегда указывает на адрес, по которому будет записан следующий элемент (см. Рис. 5-1). Запись в стек идёт в режиме постинкремента, чтение - в режиме декремента.

Запись в стек осуществляется каждый раз, когда выполняется команда перехода к подпрограмме или осуществляется вход в прерывание. В этом случае в память идёт запись 64-разрядного слова, старшая часть которого содержит 32-разрядное слово состояния процессора, а младшая - 32-разрядный счётчик адреса команд по адресу в SP, а сам SP затем увеличивается на два.

Чтение из стека происходит каждый раз, когда встречается команда возврата из подпрограммы или прерывания. В данном случае содержимое SP уменьшается на 2, по этому адресу из стека читается и восстанавливается счётчик адреса команд и старое слово состояния процессора (только для возврата из прерывания).

Вышеописанный механизм позволяет легко создавать программные прерывания, когда при вызове обработчика прерывания можно использовать обычную команду перехода к подпрограмме, а выйти из него можно командой возврата из прерывания.

Пользователь может также писать и читать из системного стека, пользуясь обычными методами адресации относительно AR7(SP), но при этом он должен строго следить за тем, чтобы содержимое SP было чётным. В противном случае при переходе к подпрограммам или отработке прерываний часть данных в стеке может быть потеряно.

После системного сброса содержимое SP не определено, поэтому необходимо позаботится о записи в него адреса начала системного стека до того, как будут разрешены прерывания или встретится хоть одна команда перехода к подпрограмме. Не существует ограничений

на расположение системного стека во внешней памяти, лишь бы адрес начала стека был выровнен по границе 64-разрядного слова. Он может находиться как в локальной, так и в глобальной памяти, но если используется режим работы с общей памятью, желательно помещать его в ту часть памяти, которая доступна только самому процессору NM6403.

Рис. 5-1. Системный стек

Стеки пользователя

Пользователь имеет возможность сам создать свой стек, используя в качестве указателя стека любой из адресных регистров - AR0 - AR6. Обмен со стеком будет осуществляться с помощью обычных методов адресации: запись в режиме постинкремента, чтение - в режиме декремента. В данном случае можно писать и читать из стека как 32-разрядные, так и 64-разрядные данные, причём в первом случае указатель стека будет меняться на +/- 1, во втором на +/- 2. Ограничение существует только на обмен 64-разрядными данными: указатель стека должен быть в этот момент выровнен на границу 64-разрядного слова. В остальном работа со стеком пользователя аналогична работе с системным стеком.

6 Управление потоком команд

6.1 Векторные команды

Одной из особенностей описываемого процессора NM6403 является наличие векторных команд, которые позволяют вести эффективную обработку массивов упакованных в 64-разрядные слова данных. Векторная команда может выполняться различное количество тактов - от 1 до 32 - в зависимости от кода команды. Если она однотактовая, то её работа в конвейере ни чем особенным не отличается от работы обычной скалярной команды. В случае, когда векторная команда выполняется несколько тактов, это можно рассматривать как аппаратную организацию цикла, состоящего из одной однотактовой векторной команды. Одновременно с работой одной векторной команды возможна работа до четырёх других векторных команд и одной скалярной. Такое решение позволяет более полно использовать аппаратные ресурсы процессора NM6403, а также две внешних памяти - локальную и глобальную. Например, можно одновременно читать из локальной памяти очередной блок данных, чтобы выполнить с ним какую-либо операцию, и писать в глобальную результат предыдущей векторной команды.

Поскольку векторные команды имеют по сравнению со скалярными более глубокий конвейер, их использование в общем случае оправдано, если обрабатываются блоки данных, содержащие не менее 6 64-разрядных слов. Когда данное условие не выполняется, возможны простои конвейера из-за взаимоизависимостей по данным.

Более подробно о векторных командах см. главы 7 и 8.

6.2 Команды управления

Процессор NM6403 имеет следующие команды управления:

- переход;
- переход со смещением, заданным в команде в виде 32-разрядной константы;
- переход к подпрограмме;
- переход к подпрограмме со смещением, заданным в команде в виде 32-разрядной константы;
- возврат из подпрограммы;
- возврат из прерывания.

Любая команда управления может быть как условной, так и

безусловной. Для избежания появления пустых тактов в конвейере желательно для каждой команды перехода/перехода к подпрограмме использовать две отложенные команды, для возврата из подпрограммы или прерывания - три.

Более подробные сведения о командах управления см. в главах 7 и 8.

6.3 Система прерываний

Типы прерываний

Процессор NM6403 поддерживает одно внешнее прерывание и девять внутренних:

- два прерывания от таймеров;
- прерывание по переполнению при выполнении арифметической операции, заданной скалярной командой;
- прерывание по запрещённой векторной команде;
- четыре прерывания от каналов ввода-вывода по завершению обмена через коммуникационные порты;
- пошаговое прерывание в режиме отладки.

Данные прерывания представлены в Табл. 6-1 и расположены в порядке уменьшения приоритета сверху вниз (при установке нескольких запросов на прерывание будет обслуживаться тот запрос, который имеет наибольший приоритет).

Табл. 6-1. Прерывания процессора NM6403

№	Тип прерывания	Адрес-вектор прерывания
1.	Обнуление системного таймера T0	00000000hex
2.	Прерывание по переполнению при выполнении арифметических операций в RISC-ядре	00000008hex
3.	Прерывание по запрещённой векторной команде	00000010hex
4.	Внешнее прерывание	00000018hex
5.	Прерывание по завершению ввода по коммуникационному порту 1	00000020hex

Табл.6-1. Прерывания процессора NM6403 (Продолжение)

№	Тип прерывания	Адрес-вектор прерывания
6.	Прерывание по завершению ввода по коммуникационному порту 0	00000028hex
7.	Прерывание по завершению вывода по коммуникационному порту 1	00000030hex
8.	Прерывание по завершению вывода по коммуникационному порту 0	00000038hex
9.	Обнуление таймера T1	00000040hex
10.	Пошаговое прерывание	00000048hex

Формирование запросов на прерывания

Запрос прерывания любого типа устанавливается только на один такт работы NM6403 и в конце этого такта фиксируется по переднему фронту синхросигнала в соответствующем разряде регистра запросов на прерывание INTR.

- Запрос прерывания по таймеру T0 или T1 возникает при переполнении соответствующего таймера, работающего в счетном режиме. Данный запрос фиксируется в INTR одновременно с записью нуля в T0 или T1 в счетном режиме.

- Запрос прерывания по переполнению при выполнении арифметической операции, заданной скалярной командой, возникает, если признак V регистра PSWR становится равным единице и соответствующая этому прерыванию маска в PSWR - также единица. Данный запрос фиксируется в INTR одновременно с установкой в единицу бита V PSWR.

- Запрос прерывания по запрещённой векторной команде возникает при попытке извлечь из буфера команд векторной команды, которая пытается сделать любое из следующих действий:

  - ♦ одновременно читать и писать в ОЗУ данных (RAM);
  - ♦ использовать данные из внешней памяти для операций в векторном процессоре, но при этом не задать операции чтения из внешней памяти;
  - ♦ читать из пустого AFIFO;
  - ♦ писать в непустое AFIFO, если нет операции чтения из него в той же команде;
  - ♦ использовать в качестве операндов для операций в векторном процессоре ОЗУ данных и AFIFO, в которых находится разное

число 64-разрядных данных;

- ♦ использовать в качестве операндов для операций в векторном процессоре ОЗУ данных и данные из внешней памяти, причём число 64-разрядных данных в ОЗУ не совпадает с кодом в команде, который задаёт количество обращений к внешней памяти;

- ♦ использовать в качестве операндов для операций в векторном процессоре AFIFO и данные из внешней памяти, причём число 64-разрядных данных в AFIFO не совпадает с кодом в команде, который задаёт количество обращений к внешней памяти;

- ♦ задать операционную команду арифметическую или взвешенного суммирования и одновременно перезапись весов из теневой матрицы в рабочую (взведён бит “LOAD”).

Эта ситуация может привести к самоблокировке конвейера процессора NM6403, ликвидировать которую можно только с помощью системного сброса. Поэтому данная векторная команда не выполняется, а вместо её провала в конвейер фиксируется запрос на данное прерывание, если соответствующая этому прерыванию маска в регистре PSWR равна единице.

- Запрос прерывания по внешнему прерыванию возникает при приходе сигнала низкого уровня на внешний вход $\overline{INT}$ и фиксируется в INTR в конце такта, в котором пришел данный запрос.

- Запросы прерывания по завершению ввода/вывода по коммуникационным портам возникают при обнулении счетчика передаваемых/принимаемых слов соответствующего порта и фиксируется в INTR после осуществления последнего ПДП для данной передачи/приема

- Запросом на пошаговое прерывание является установка в единицу соответствующей маски на данное прерывание в регистре PSWR.

После фиксации запроса на прерывание в регистре запросов INTR его отработка начнётся, если выполняются следующие условия:

- соответствующая этому прерыванию маска в PSWR равна единице;

- нет готовых к отработке других более приоритетных прерываний.

Если эти два условия соблюдаются, выбирается очередная команда, но она не выполняется, а вместо неё осуществляется переход по адрес-вектору прерывания с запоминанием адреса возврата и слова состояния процессора, и соответствующий запрос в INTR

сбрасывается. Другие прерывания будут запрещены, пока не произойдёт любая запись в регистр PSWR и не установятся новые маски прерываний. В конце отработки прерывания выбирается команда возврата из прерывания, по которой восстанавливается содержимое счётчика команд и слово состояния процессора, снова выбирается команда, которая из-за прерывания не была выполнена, и продолжается обычное выполнение программы. Следует отметить, что любая команда возврата из прерывания разрешает отработку других прерываний, поскольку по ней идёт запись в PSWR старого содержимого из стека.

Когда разрешается пошаговое прерывание (режим отладки), при возврате из данного прерывания включается специальный механизм, который позволяет выбрать и исполнить очередную 64-разрядную команду или пару 32-разрядных команд, и только потом войти в новое пошаговое прерывание. Для остальных прерываний возможно по выходу из одного сразу попасть в другое прерывание.

6.4 Системный сброс

После включения питания процессор NM6403 находится в неопределённом состоянии. Чтобы начать с ним работу, необходимо выставить низкий уровень на входе $\overline{RESET}$ и держать его по меньшей мере 12 периодов входного тактового сигнала CLK.

Системный сброс не одинаково воздействует на все внешние выводы процессора NM6403: на некоторые синхронно, на некоторые асинхронно. Синхронный сброс привязан к внутренней синхронизации, асинхронный сброс непосредственно влияет на выводы и поэтому он осуществляется быстрее. Более подробную информацию о влиянии системного сброса на выводы NM6403 можно получить в разделе 13.5.

Табл. 6-2 содержит сведения о состоянии внешних выводов после подачи низкого уровня на вход $\overline{RESET}$. Для каждого вывода имеется информация, какой для него применяется сброс - синхронный или асинхронный.

Табл. 6-2. Внешние выводы процессора NM6403 после системного сброса

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
Интерфейс с глобальной шиной (88 выводов)			
D63 - D0	64	I/O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
A15 - A1	15	O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{RAS0}}$ / $\overline{\text{CS0}}$	1	O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{RAS1}}$ / $\overline{\text{CS1}}$	1	O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{CASL}}$ / $\overline{\text{WEL}}$	1	O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{CASH}}$ / $\overline{\text{WEH}}$	1	O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{OE}}$	1	O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{WE}}$ /A16	1	O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{HOLDO}}$/A17[3)]	1	O(Z)	Синхронный сброс. Выдача высокого уровня.
$\overline{\text{HOLDI}}$/A18[3)]	1	I/O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{RDY}}$ /A19[3)]	1	I/O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
Интерфейс с локальной шиной (88 выводов)			
LD63 - LD0	64	I/O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
LA15 - LA1	15	O(Z)	Синхронный сброс. Выдача низкого уровня.
$\overline{\text{LRAS0}}$ / $\overline{\text{LCS0}}$	1	O(Z)	Синхронный сброс. Выдача высокого уровня.
$\overline{\text{LRAS1}}$ / $\overline{\text{LCS1}}$	1	O(Z)	Синхронный сброс. Выдача высокого уровня.
$\overline{\text{LCASL}}$ / $\overline{\text{LWEL}}$	1	O(Z)	Синхронный сброс. Выдача высокого уровня.
$\overline{\text{LCASH}}$ / $\overline{\text{LWEH}}$	1	O(Z)	Синхронный сброс. Выдача высокого уровня.
$\overline{\text{LOE}}$	1	O(Z)	Синхронный сброс. Выдача высокого уровня.

Табл.6-2. Внешние выводы процессора NM6403 после системного сброса (Продолжение)

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
$\overline{\text{LWE}}$/LA16	1	O(Z)	Синхронный сброс. Выдача высокого уровня.
$\overline{\text{LHOLDO}}$/LA17	1	O(Z)	Синхронный сброс. Выдача высокого уровня.
$\overline{\text{LHOLDI}}$/LA18	1	I/O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{LRDY}}$/LA19	1	I/O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
Коммуникационный порт 0 (13 выводов)			
C0D7 - C0D0	8	I/O(Z)	Асинхронный сброс. Выдача неопределённой информации.
$\overline{\text{CREQ0}}$	1	I/O(Z)	Асинхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{CACK0}}$	1	I/O(Z)	Асинхронный сброс. Выдача высокого уровня.
$\overline{\text{CSTRB0}}$	1	I/O(Z)	Асинхронный сброс. Выдача высокого уровня.
$\overline{\text{CRDY0}}$	1	I/O(Z)	Асинхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{CDIR0}}$	1	O	Синхронный сброс. Выдача низкого уровня.
Коммуникационный порт 1 (13 выводов)			
C1D7 - C1D0	8	I/O(Z)	Асинхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{CREQ1}}$	1	I/O(Z)	Асинхронный сброс. Выдача высокого уровня.
$\overline{\text{CACK1}}$	1	I/O(Z)	Асинхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{CSTRB1}}$	1	I/O(Z)	Асинхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{CRDY1}}$	1	I/O(Z)	Асинхронный сброс. Выдача высокого уровня.
$\overline{\text{CDIR1}}$	1	O	Синхронный сброс. Выдача высокого уровня.

Табл.6-2. Внешние выводы процессора NM6403 после системного сброса (Продолжение)

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
Общее управление (9 выводов)			
CLK	1	I	Системный сброс не действует на состояние данного вывода.
$\overline{\text{RESET}}$	1	I	Вход системного сброса
TIMER	1	I/O(Z)	Синхронный сброс. Установка в высокоимпедансное состояние.
$\overline{\text{INT}}$	1	I	Системный сброс не действует на состояние данного вывода.
$\overline{\text{INTA}}$	1	O	Синхронный сброс. Выдача низкого уровня.
BOOTM	1	I	Системный сброс не действует на состояние данного вывода.
MVOE	1	I	Системный сброс не действует на состояние данного вывода.
TEST_OUT1	1	O	Выход не определён.
TEST_OUT2	1	O	Выход не определён.
Питание (27 выводов)			
VSS	13	I	Системный сброс не действует на состояние данного вывода.
VDD	12	I	Системный сброс не действует на состояние данного вывода.
VBB	2	I	Системный сброс не действует на состояние данного вывода.
Неиспользуемые выводы (18 выводов)			
NC	18	-	Системный сброс не действует на состояние данного вывода.

Примечание:

1) Для выводов со знаком инверсии активным является низкий уровень сигнала;

2) I- входы;

O - выходы;

O(Z) - выходы с высокоимпедансным состоянием;

I/O(Z) - двунаправленные выводы.

Состояние регистров и вспомогательных внутрикристальных памятей процессора NM6403 после системного сброса

Состояние регистров после системного сброса следующее:

```
PC    =  1100.0000.0000.0000.0000.0000.0000.0000 bin;
PSWR  =  0000.0000.0000.0000.0000.0000.0000.xxxx bin
```

(признаки N,Z,V,C неопределены);

```
INTR  =  0100.1110.0xxx.xxxx.xxx1.0100.0000.0000 bin;
LMICR =  0000.0000.0000.0000.0000.0000.0000.0000 bin;
GMICR =  0000.0000.0000.0000.0000.0000.0000.0000 bin.
```

Все остальные регистры имеют неопределенное значение.

Вспомогательные памяти векторного узла находятся в таком состоянии:

WFIFO и AFIFO - пусты;

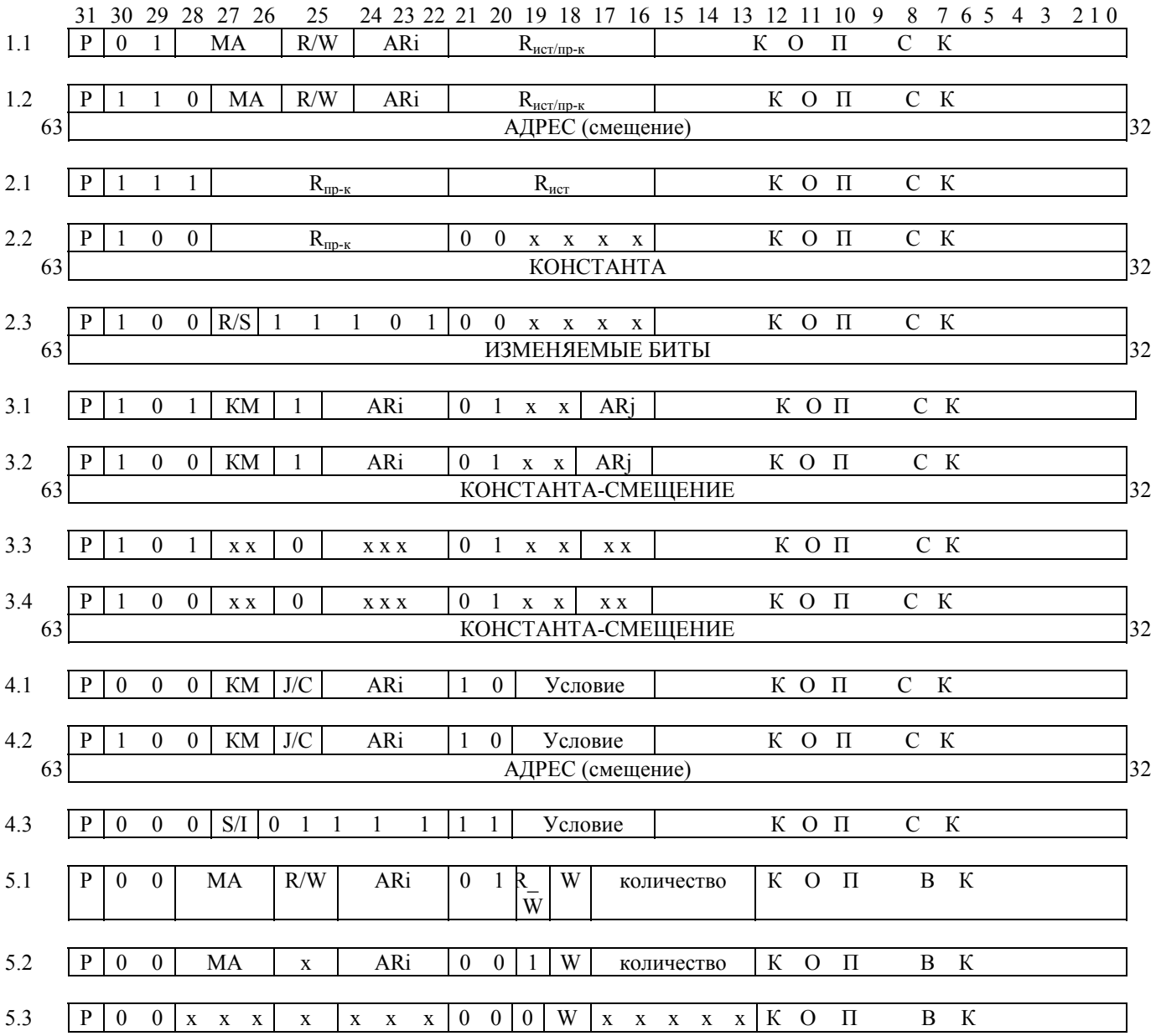
RAM - неопределенное состояние.

После системного сброса должна будет выполниться процедура инициализации процессора NM6403, информацию о которой см. в главе 12.

7 Введение в систему команд

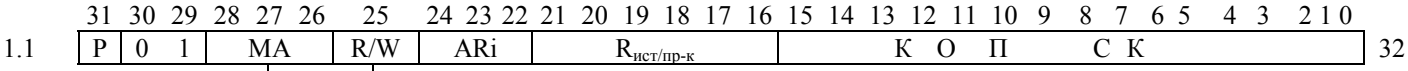
Процессор NM6403 работает с командами фиксированной длины по 32 или 64 разряда, которые можно разделить на 4 основные группы: векторные команды и команды скалярные - обработки операндов, управления и пересылки. Все команды выполняются за один такт синхронизации. Форматы команд процессора NM6403 приведены на Рис. 7-1, краткое их описание дано ниже. Более полно о системе команд можно узнать в приложении А.

Рис. 7-1. Форматы команд процессора NM6403



7.1 Форматы команд, задающих пересылку данных “регистр-память” (форматы 1.1 и 1.2)

1.1 Rист/пр-к $\leftrightarrow(f_{адр.}(AR_i, GR_i))$; $AR_i \leftarrow f_m(AR_i, GR_i)$



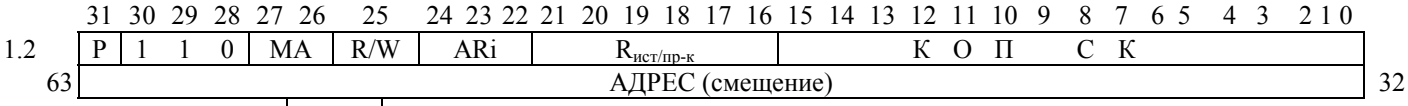
R/W	Направление пересылки данных
0	Чтение из памяти в $R_{ист/пр-к}$
1	Запись в память из $R_{ист/пр-к}$

MA	$f_{адр.}(AR_i, GR_i)$	$f_M(AR_i, GR_i)$
0 0 0	GR_i	AR_i
0 0 1	AR_i+GR_i	AR_i+GR_i
0 1 0	GR_i	GR_i
0 1 1	Резерв	Резерв
1 0 0	AR_i	AR_i
1 0 1	AR_i	AR_i+GR_i
1 1 0	AR_i-a	AR_i-a
1 1 1	AR_i	AR_i+a

Примечание: *a=1 при адресации 32-разрядных данных;*

a=2 при адресации 64-разрядных данных.

1.2 Rист/пр-к $\leftrightarrow(f_{адр.}(AR_i, C_M))$; $AR_i \leftarrow f_m(AR_i, C_M)$

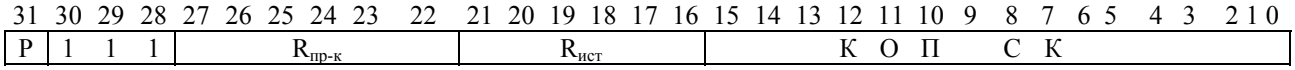


R/W	Направление пересылки данных
0	Чтение из памяти в $R_{ист/пр-к}$
1	Запись в память из $R_{ист/пр-к}$

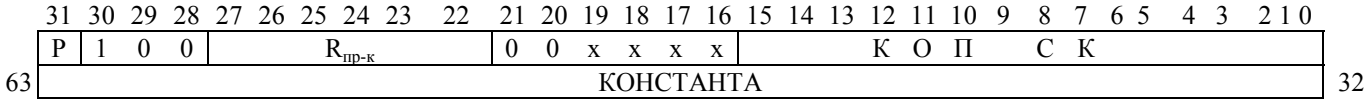
MA	$f_{адр.}(AR_i, C_M)$	$f_M(AR_i, C_M)$
0 0	Адрес	AR_i
0 1	AR_i+C_M	AR_i+C_M
1 0	Адрес	Адрес
1 1	Резерв	Резерв

7.2 Форматы команд пересылки данных типа “регистр-регистр” (форматы 2.1 - 2.3)

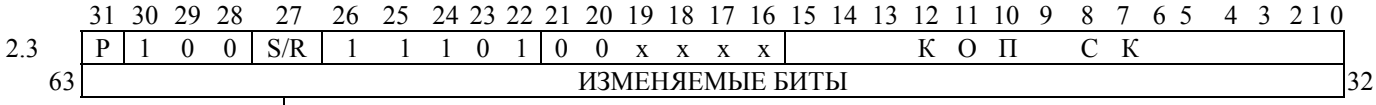
2.1 Rист→Rпр-к



2.2 Rпр-к←константа



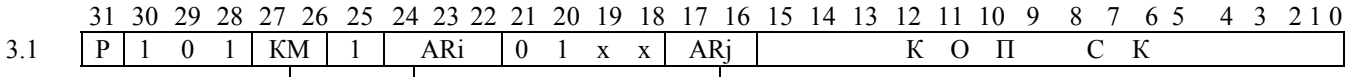
2.3 Установка/сброс битов регистра слова состояния процессора (PSWR)



R/S	Изменение разряда (ов)
0	Сброс в 0
1	Установка в 1

7.3 Форматы команд модификации адресных регистров (форматы 3.1 - 3.4)

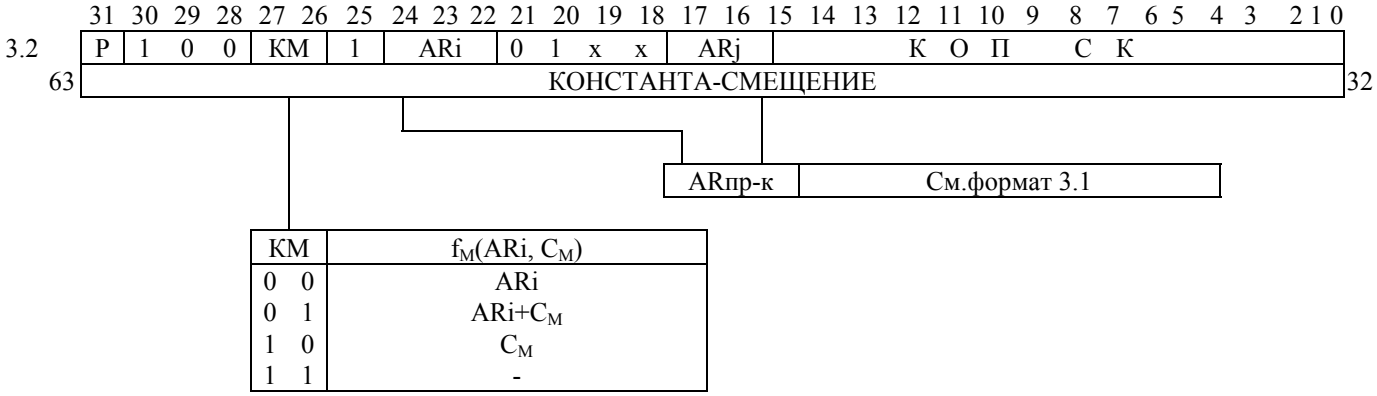
3.1 ARj←f_M(ARi,GRi)



ARпр-к	AR приёмник
0 0 0	AR0
0 0 1	AR1
0 1 0	AR2
0 1 1	AR3
1 0 0	AR4
1 0 1	AR5
1 1 0	AR6
1 1 1	SP(AR7)

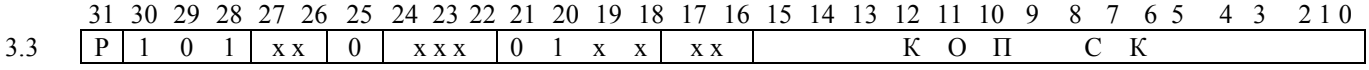
КМ	$f_M(ARi, GR_i)$
0 0	ARi
0 1	AR_i+Gr_i
1 0	Gr_i
1 1	Резерв

3.2 ARj←f_M(ARi, C_M)

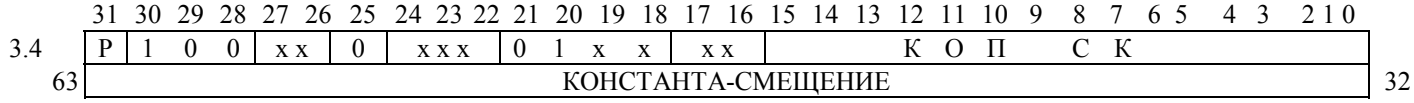


КМ	f_M(ARi, C_M)
0 0	ARi
0 1	ARi+C_M
1 0	C_M
1 1	-

3.3 Нет операций ввода/вывода и модификации адресных регистров

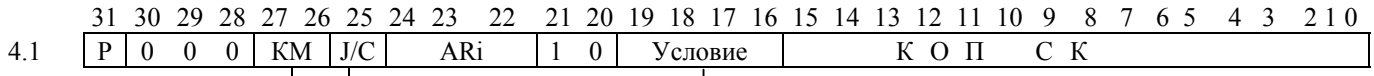


3.4 Нет операций ввода/вывода и модификации адресных регистров



7.4 Форматы команд управления (форматы 4.1 - 4.3)

4.1 Переход/переход к подпрограмме

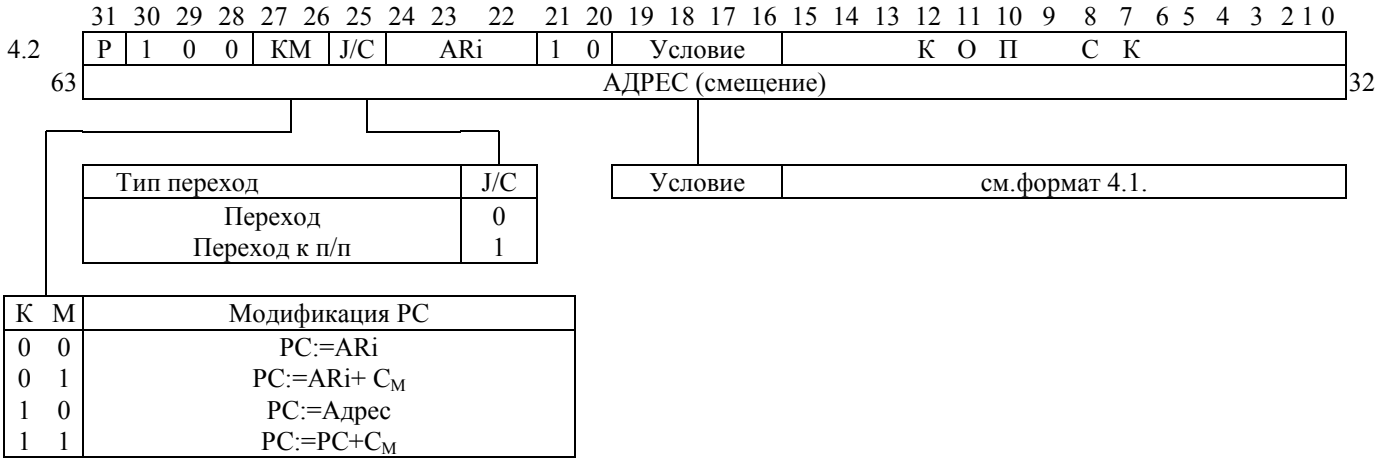


Тип переход	J/C
Переход	0
Переход к п/п	1

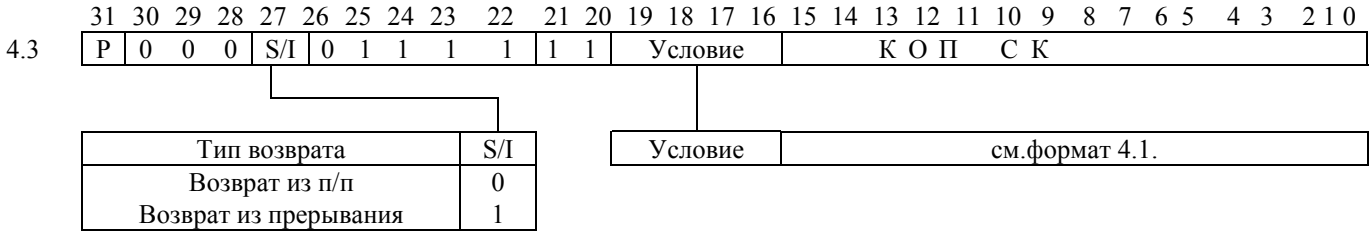
К	М	Модификация PC
0	0	PC:=ARi
0	1	PC:=ARi+GRi
1	0	PC:=GRi
1	1	PC:=PC+GRi

Условие				Анализируемое сочетание признаков
0	0	0	0	Переход, если C=0
0	0	0	1	Переход, если V=0
0	0	1	0	Переход, если N+Z=0
0	0	1	1	Переход, если N=0
0	1	0	0	Переход, если (V ⊕ N)+Z=0
0	1	0	1	Переход, если V ⊕ N=0
0	1	1	0	Переход, если Z=0
0	1	1	1	Переход безусловный
1	0	0	0	Переход, если C=1
1	0	0	1	Переход, если V=1
1	0	1	0	Переход, если N+Z=1
1	0	1	1	Переход, если N=1
1	1	0	0	Переход, если (V ⊕ N)+Z=1
1	1	0	1	Переход, если V ⊕ N=1
1	1	1	0	Переход, если Z=1
1	1	1	1	Перехода нет

4.2 Переход/переход к подпрограмме со смещением

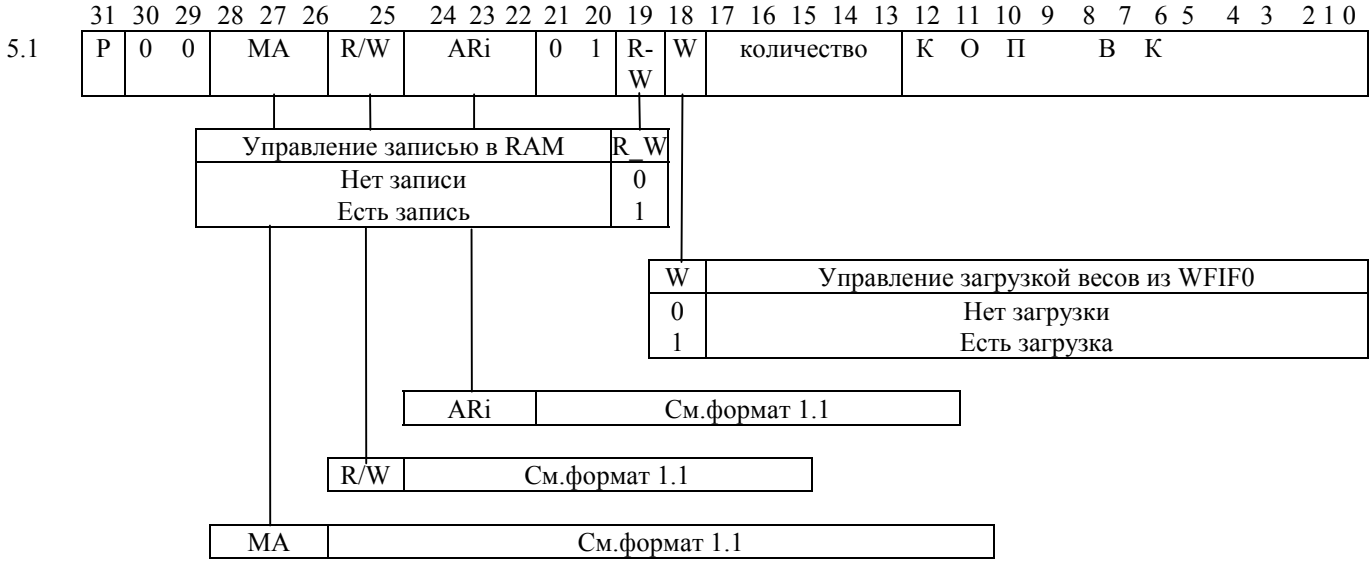


4.3 Возврат из подпрограммы/прерывания



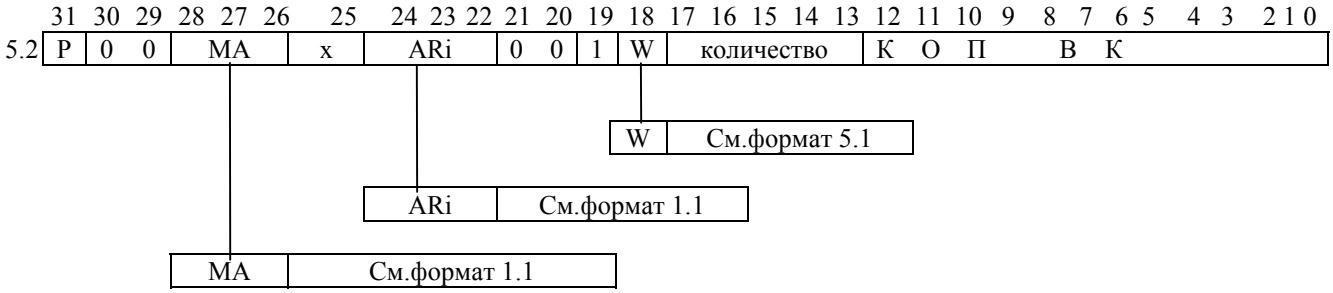
7.5 Формат векторных команд (форматы 5.1 - 5.3)

5.1 ВП↔(fадр.(ARi, GRi)); ARi←f_M(ARi, Gri)

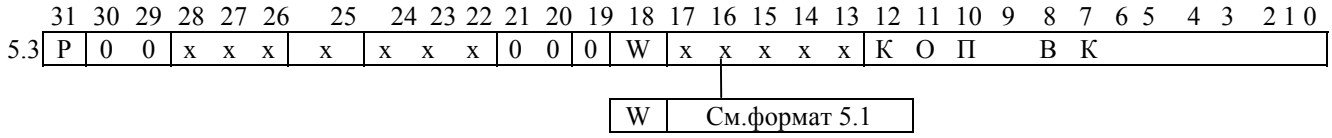


5.2 Загрузка весов в WFIFO из внешней памяти

WFIFO←(fадр.(ARi, GRi)); ARi←f_M(ARi, Gri)

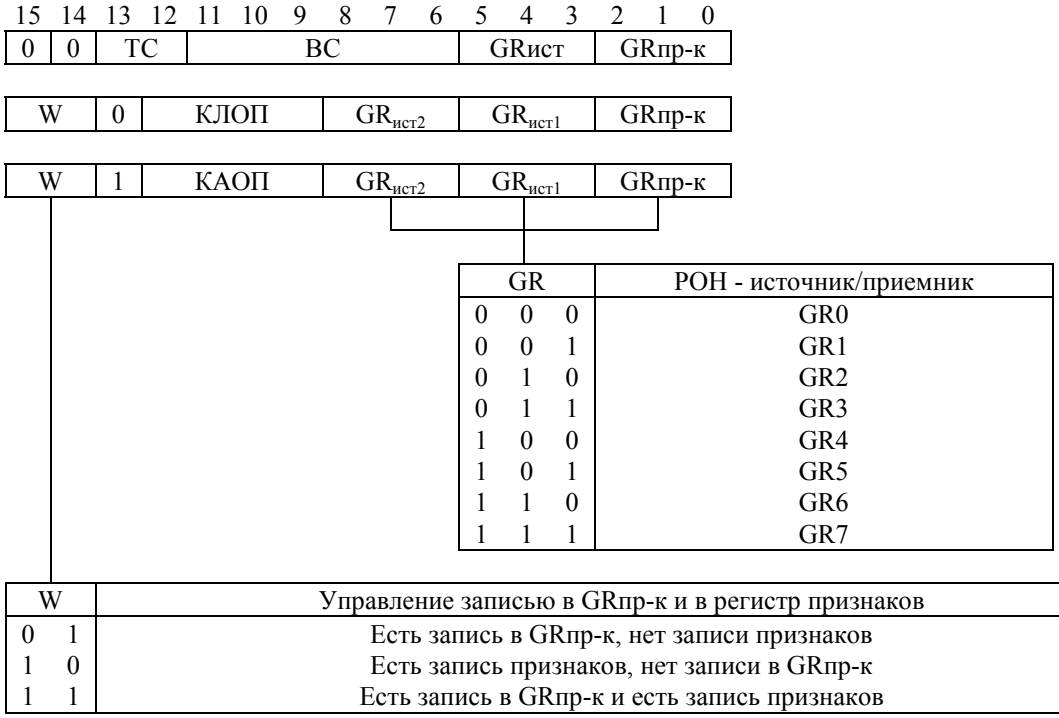


5.3 Нет операций ввода/вывода



Примечание: *для форматов 5.1и 5.2 поле "количество" задает количество 64-разрядных слов, участвующих в операциях пересылок данных*

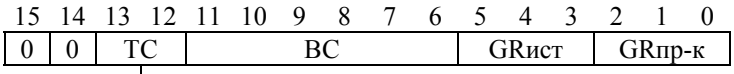
7.6 Форматы поля КОП СК, задающего арифметико-логическую операцию в скалярной команде



GR	РОН - источник/приемник
0 0 0	GR0
0 0 1	GR1
0 1 0	GR2
0 1 1	GR3
1 0 0	GR4
1 0 1	GR5
1 1 0	GR6
1 1 1	GR7

W	Управление записью в GRпр-к и в регистр признаков
0 1	Есть запись в GRпр-к, нет записи признаков
1 0	Есть запись признаков, нет записи в GRпр-к
1 1	Есть запись в GRпр-к и есть запись признаков

6.1Форматы поля КОП СК, задающего операцию сдвига



TC		Тип сдвига
0	0	Циклический сдвиг
0	1	Логический сдвиг
1	0	Арифметический сдвиг
1	1	Логический сдвиг через “C”

Примечание:

1. Поле BC задает величину сдвига в дополнительном коде:

- положительное значение - сдвиг влево (от 0 до 31 разряда),

- отрицательное значение - сдвиг вправо (от 1 до 32 разрядов).

2. “C” - признак переноса слова состояния процессора.

3. Логический сдвиг через “C” может выполняться только на один разряд влево или вправо; сдвиги остальных типов могут выполняться на любое от 1 до 31 число разрядов влево или вправо.

4. Сдвиг на 0 или 32 разряда любого типа воспринимается как код “нет операции”, и при этом не изменяется ни приёмник результата операции GRпр-к, ни признаки.

Сдвиги от 1 до 31 разряда меняют признаки слова состояния процессора по следующим правилам:

N - признак знака - равен старшему (знаковому) разряду результата;

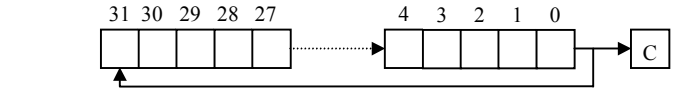
Z - признак нуля - равен единице, если все разряды GRпр-ка нулевые, или нулю в противном случае;

V - признак переполнения - устанавливается в единицу только при арифметических сдвигах влево, если получаемый признак C не равен 31 разряду GRпр-ка, во всех остальных случаях равен нулю;

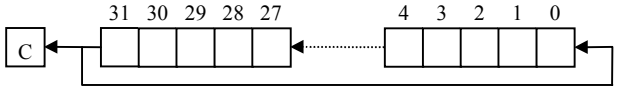
C - признак переноса - равен последнему вытолкнутому при сдвиге разряду из GRист.

Схемы различных вариантов сдвига на один разряд изображены на Рис. 7-2. Сдвиги на большее число разрядов эквивалентны многократному сдвигу на единицу, хотя и выполняются за один такт работы процессора.

Рис. 7-2. Схемы сдвигов



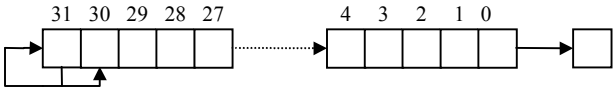
Циклический сдвиг вправо



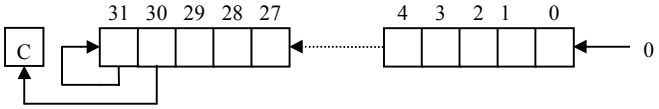
Циклический сдвиг влево

Логический сдвиг вправо

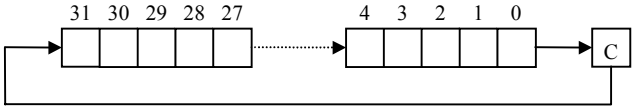
Логический сдвиг влево



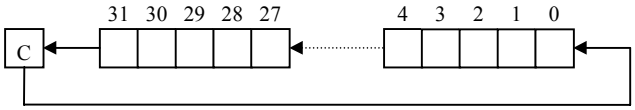
Арифметический сдвиг вправо



Арифметический сдвиг влево



Логический сдвиг вправо через “C”



Логический сдвиг влево через “C”

6.2 Формат поля КОП СК, задающего логическую операцию

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W		0	КЛОП				$GR_{ист2}$			$GR_{ист1}$			GRпр-к		

КЛОП				Код логической операции	N	Z	V	C
0	0	0	0	0	0	1	0	0
0	0	0	1	$\overline{GR_{ист2}}$ & $\overline{GR_{ист1}}$	+	+	0	0
0	0	1	0	$GR_{ист2}$ & $\overline{GR_{ист1}}$	+	+	0	0
0	0	1	1	$\overline{GR_{ист1}}$	+	+	0	0
0	1	0	0	$\overline{GR_{ист2}}$ & $GR_{ист1}$	+	+	0	0
0	1	0	1	$\overline{GR_{ист2}}$	+	+	0	0
0	1	1	0	$GR_{ист2} \oplus GR_{ист1}$	+	+	0	0
0	1	1	1	$\overline{GR_{ист2}} + \overline{GR_{ист1}}$	+	+	0	0
1	0	0	0	$GR_{ист2}$ & $GR_{ист1}$	+	+	0	0
1	0	0	1	$GR_{ист2} \oplus \overline{GR_{ист1}}$	+	+	0	0
1	0	1	0	$GR_{ист2}$	+	+	0	0
1	0	1	1	$GR_{ист2} + \overline{GR_{ист1}}$	+	+	0	0
1	1	0	0	$GR_{ист1}$	+	+	0	0
1	1	0	1	$\overline{GR_{ист2}} + GR_{ист1}$	+	+	0	0
1	1	1	0	$GR_{ист2} + GR_{ист1}$	+	+	0	0
1	1	1	1	-1	1	0	0	0

6.3 Формат поля КОП СК, задающего арифметическую операцию

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W		1	КАОП				$GR_{ист2}$			$GR_{ист1}$			GRпр-к		

КАОП				Код арифметической операции	N	Z	V	C
0	0	0	0	$GR_{ист2}$-$GR_{ист1}$	+	+	+	+
0	0	0	1	$GR_{ист2}$-$GR_{ист1}$-1+"C	+	+	+	+
0	0	1	0	$GR_{ист2}$+1	+	+	+	+
0	0	1	1	$GR_{ист2}$+"C"	+	+	+	+
0	1	0	0	$GR_{ист2}$-1	+	+	+	+
0	1	0	1	$GR_{ист2}$-1+"C"	+	+	+	+
0	1	1	0	$GR_{ист2}$+ $GR_{ист1}$	+	+	+	+
0	1	1	1	$GR_{ист2}$+ $GR_{ист1}$+"C"	+	+	+	+
1	0	0	0	Первый шаг умножения	?	?	0	+
1	0	0	1	Шаг умножения	?	?	0	+
1	0	1	X	Резерв				
1	1	0	0	-$GR_{ист2}$	+	+	+	+
1	1	X	1	Резерв				
1	1	1	X	Резерв				

Примечание: *C - признак переноса из слова состояния процессора.*

Если в поле W задаёт запись признаков, то они устанавливаются соответственно столбцам N, Z,V,C форматов 6.2 и 6.3., иначе они сохраняют своё значение. В форматах используются следующие обозначения:

“+”	-	признак устанавливается по результату операции;
“0”	-	признак обнуляется;
“1”	-	признак устанавливается в единицу;
“?”	-	значение признака не определено.
N		31 (знаковому) разряду результата;
Z	равен	единице, если все разряды результата нулевые, иначе - нулю;
V	равен	нулю для операций шага умножения, для остальных операций он формируется как результат операции “исключающее или” переноса из 31 разряда и переноса в 31 разряд;
C	равен	переносу из 31 разряда для всех операций, за исключением шага умножения, когда он становится равен предпоследнему по значимости (1-му) разряду множителя (GR7).

7.7 Формат поля КОП ВК, задающего арифметико-логическую операцию в векторной команде

12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	КЛОП				FPX	FPY	X		Y		L	Логическая операция над операндами X и Y
1	1	КАОП				FAX	FAY	X		Y		L	Арифметическая операция над операндами X и Y
0	1	M		0	SH	FPX	FPY	X		Y		L	Операция маскирования $X*M+Y*\overline{M}$
0	1	0 0		1	X	X	X	0 0		0 0		L	Запись в AFIFO содержимого регистров F2CR, F1CR, NB2, SB2, VR
0	0	0 0		0	X	X	X	0 0		0 0		L	Нет операции
0	0	M		VR	SH	FAX	FAY	X		Y		L	Операция взвешенного суммирования типа W*X+Y

L	Управление перезаписью весов из теневой матрицы весов в рабочую
0	Нет перезаписи
1	Есть перезапись

OP		Выбор операнда
0	0	Операнд равен нулю
0	1	Операнд из RAM
1	0	Операнд из AFIF0
1	1	Операнд выбирается из внешней памяти

FP(FA)	Управление функцией пороговой (функцией активации)
0	Функция не используется
1	Функции используется

Формат поля КОП ВК, задающего логическую операцию

12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	КЛОП				FPX	FPY	X		Y		L

КЛОП				Код логической операции
0	0	0	0	0
0	0	0	1	$\overline{X}$ & $\overline{Y}$
0	0	1	0	X& $\overline{Y}$
0	0	1	1	$\overline{Y}$
0	1	0	0	$\overline{X}$ &Y
0	1	0	1	$\overline{X}$
0	1	1	0	X ⊕ Y
0	1	1	1	$\overline{X} + \overline{Y}$
1	0	0	0	X&Y
1	0	0	1	$\overline{X}$ ⊕ Y
1	0	1	0	X
1	0	1	1	X+ $\overline{Y}$
1	1	0	0	Y
1	1	0	1	$\overline{X}$ +Y
1	1	1	0	X+Y
1	1	1	1	-1

Формат поля команд КОП ВК, задающего арифметическую операцию

12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	КАОП				FAX	FAY	X		Y		L

КАОП				Код арифметической операции
X	0	0	X	X- Y
X	0	1	X	X+1
X	1	0	X	X-1
X	1	1	X	X+Y

Операции маскирования и взвешенного суммирования

12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	M		0	SH	FPX	FPY	X		Y		L

Операция маскирования $X*M + Y*\overline{M}$

0	0	M		VR	SH	FAX	FAY	X		Y		L

Операция взвешенного суммирования типа W*X+Y

SH	Управление циклическим сдвигом операнда X на 1 разряд вправо
0	Нет сдвига
1	Есть сдвиг

VR	Управление выборкой регистра VR в качестве операнда Y
0	VR не является операндом Y
1	VR является операндом Y

M		Выбор операнда - маски
0	0	Маскирование отсутствует
0	1	Маска выбирается из RAM
1	0	Маска выбирается из AFIF0
1	1	Маска выбирается из внешней памяти

7.8 Поле управления одновременным выполнением нескольких команд

31

P	Управление выполнением данной команды на фоне выполнения векторной команды
0	Выполнение команды откладывается до окончания векторной команды
1	Команда выполняется даже если еще не закончилась векторная команда

7.9 Поле выбора адресного регистра

25 24 23

ARi			Номер AR или GR
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Примечание: *AR7 (SP) может использоваться в качестве системного указателя стека адресов возврата при входе/выходе из подпрограммы (прерывания), причём изменяется он в этом случае только на +/- 2.*

7.10 Поле $R_{ист/пр-к}$ в командах пересылки данных

$R_{ист/пр-к}$						Регистр-источник	Регистр-приемник
0	0	0	0	0	0	AR0	AR0
0	0	0	0	0	1	AR1	AR1
0	0	0	0	1	0	AR2	AR2
0	0	0	0	1	1	AR3	AR3
0	0	0	1	0	0	AR4	AR4
0	0	0	1	0	1	AR5	AR5
0	0	0	1	1	0	AR6	AR6
0	0	0	1	1	1	AR7(SP)	AR7(SP)
0	0	1	0	0	0	OCA0	OCA0
0	0	1	0	0	1	ICA0	ICA0
0	0	1	0	1	0	OCA1	OCA1
0	0	1	0	1	1	ICA1	ICA1
0	0	1	1	0	0	T0	T0
0	0	1	1	0	1	LMICR	LMICR
0	0	1	1	1	0	GMICR	GMICR
0	0	1	1	1	1	PC	PC

$R_{ист/пр-к}$						Регистр-источник	Регистр-приемник
0	1	0	0	0	0	GR0	GR0
0	1	0	0	0	1	GR1	GR1
0	1	0	0	1	0	GR2	GR2
0	1	0	0	1	1	GR3	GR3
0	1	0	1	0	0	GR4	GR4
0	1	0	1	0	1	GR5	GR5
0	1	0	1	1	0	GR6	GR6
0	1	0	1	1	1	GR7	GR7
0	1	1	0	0	0	OCC0	OCC0
0	1	1	0	0	1	ICC0	ICC0
0	1	1	0	1	0	OCC1	OCC1
0	1	1	0	1	1	ICC1	ICC1
0	1	1	1	0	0	T1	T1
0	1	1	1	0	1	Резерв	PSWRreset
0	1	1	1	1	0	INTR	Резерв
0	1	1	1	1	1	PSWR	PSWR
1	0	0	0	0	0	GR0, AR0	GR0, AR0
1	0	0	0	0	1	GR1,AR1	GR1,AR1
1	0	0	0	1	0	GR2,AR2	GR2,AR2
1	0	0	0	1	1	GR3,AR3	GR3,AR3
1	0	0	1	0	0	GR4,AR4	GR4,AR4
1	0	0	1	0	1	GR5,AR5	GR5,AR5
1	0	0	1	1	0	GR6,AR6	GR6,AR6
1	0	0	1	1	1	GR7,AR7	GR7,AR7
1	0	1	0	0	0	OCC0, OCA0	OCC0, OCA0
1	0	1	0	0	1	ICC0, ICA0	ICC0, ICA0
1	0	1	0	1	0	OCC1, OCA1	OCC1, OCA1
1	0	1	0	1	1	ICC1, ICA1	ICC1, ICA1
1	0	1	1	0	0	T1, T0	T1, T0
1	0	1	1	0	1	DIR0	DOR0
1	0	1	1	1	0	DIR1	DOR1
1	0	1	1	1	1	PSWR,PC	PSWR,PC
1	1	0	0	0	0	Резерв	NBL
1	1	0	0	0	1	Резерв	SBL
1	1	0	0	1	0	Резерв	F1CRL
1	1	0	0	1	1	Резерв	F2CRL
1	1	0	1	0	0	Резерв	NBH
1	1	0	1	0	1	Резерв	SBH
1	1	0	1	1	0	Резерв	F1CRH
1	1	0	1	1	1	Резерв	F2CRH
1	1	1	0	0	0	Резерв	NB
1	1	1	0	0	1	Резерв	SB
1	1	1	0	1	0	Резерв	F1CR
1	1	1	0	1	1	Резерв	F2CR
1	1	1	1	0	0	Резерв	VR
1	1	1	1	0	1	Резерв	PSWRset
1	1	1	1	1	0	Резерв	VRL
1	1	1	1	1	1	Резерв	VRH

Табл. 7-1.Обозначение регистров, назначение, разрядность

Обозначение регистров и их назначение	Разрядность
GRi - регистр общего назначения i (i=0,…,7)	32
ARj - адресный регистр j (j=0,…,6)	32
SP(AR7) - указатель стека адресов возврата	32
PC - программный счетчик	32
PSWR - регистр слова состояния процессора	32
PSWRset - код для побитовой установки PSWR в единицу (псевдорегистр)	-
PSWRreset - код для побитового сброса PSWR в ноль (псевдорегистр)	-
Ti - таймер i (i=0,1)	32
GMICR - регистр управления интерфейсом с глобальной шиной	32
LMICR - регистр управления интерфейсом с локальной шиной	32
INTR - регистр запросов на прерывание и ПДП	32
OCAi - регистр адреса канала вывода i (i=0,1)	32
ICAi - регистр адреса канала ввода i (i=0,1)	32
OCCi - счетчик канала вывода i (i=0,1)	32
ICCi - счетчик канала ввода i (i=0,1)	32
DORi - регистр данных канала вывода i (i=0,1)	64
DIRi - регистр данных канала ввода i (i=0,1)	64
FiCR (H,L) - регистр управления функцией активации i (i=1,2) (старшая, младшая часть)	64(32)
VR(H,L) - регистр порога (старшая, младшая часть)	64(32)
NB(H,L) - i-й регистр границ нейронов (старшая, младшая часть)	64(32)
SB(H,L) - регистр границ синапсов (старшая, младшая часть)	64(32)

Дополнительную информацию можно найти в "Описании языка Ассемблер".

8 Конвейерная организация выполнения команд

8.1 Общие сведения

Выполнение каждой команды в процессоре NM6403 разделено на несколько стадий, каждая из которых реализуется соответствующей ступенью (фазой) конвейера выполнения команд. Для скалярных команд конвейер насчитывает 5 ступеней, для векторных добавляются ещё 3 ступени.

Конвейерное выполнение основных типов команд (скалярных, векторных и команд управления) подробно рассматривается далее в разделе 8.2, соответственно в пунктах 8.2.1, 8.2.2 и 8.2.3.

Выполняемые процессором NM6403 команды в той или иной степени используют следующие вычислительные ресурсы:

- регистры;
- блоки внутренней памяти;
- функциональные устройства;
- внутренние и внешние шины.

Для правильного выполнения программы в конвейере процессора NM6403 необходимо отслеживание конфликтов по использованию этих ресурсов в командах. Если выбранная для выполнения в процессоре NM6403 команда вступает в конфликт по использованию ресурсов с уже находящимися в конвейере командами, то её запуск в конвейер следует задержать до разрешения этого конфликта. В NM6403 такое отслеживание конфликтов реализуется аппаратно в устройстве управления.

В эффективных по времени программах на языке ассемблера следует учитывать возможные задержки выполнения команд из-за конфликтов и, по возможности, не использовать в программе конфликтные сочетания команд.

В компиляторах языков высокого уровня для процессора NM6403 особенности конвейерного выполнения команд должны учитываться в специальных блоках машинно-зависимой оптимизации, автоматически планирующих оптимальное использование ресурсов и оптимальную очередность выполнения команд.

Конфликты, возникающие при конвейерном выполнении команд в процессоре NM6403, рассматриваются далее в разделе 8.3. В этом разделе выделены две группы конфликтов: конфликты при выборке команд из памяти в буфер команд (см. 8.3.1) и конфликты, связанные с использованием вычислительных ресурсов (см. 8.3.2).

Пункт 8.3.1 содержит описание организации работы со счетчиком адреса команд (PC) и дисциплины выборки команд из внешней памяти в буфер команд. Показывается, как используемый в процессоре NM6403 способ выборки команд проявляется при выполнении линейных участков программы, а также при выполнении передач управления.

Пункт 8.3.2 - основной в описании конфликтных ситуаций. Он содержит описания аппаратно реализованных алгоритмов отслеживания использования вычислительных ресурсов. Эти алгоритмы выполняют три основные функции:

- проверка условий запуска команды на выполнение в конвейере, которая состоит в определении отсутствия конфликтов по используемым в команде ресурсам;
- захват необходимых для выполнения команды вычислительных ресурсов;
- освобождение захваченных для выполнившейся команды вычислительных ресурсов.

В пункте 8.3.2 подробно рассматриваются алгоритмы проверки условий запуска команд, а также захвата и освобождения ресурсов.

Работа с ресурсами в процессоре NM6403 имеет определенную особенность. Дело в том, что для более эффективного использования аппаратуры в NM6403 все вычислительные ресурсы разделены на две группы:

- ресурсы с индивидуальным их использованием командами;
- ресурсы с совместным использованием.

Ресурсы индивидуального использования (регистры, блоки внутренней памяти и функциональные устройства) захватываются уходящей на выполнение непосредственно после проверки условий запуска командой и освобождаются после завершения этой команды. После захвата ресурс не может использоваться другой командой до тех пор, пока он не будет освобожден.

Ресурсы совместного использования (внешние и внутренние шины) захватываются условно непосредственно после проверки условий запуска. В процессе выполнения команды реальный захват такого устройства производится посредством обслуживания заявок на захват. Захват по такой заявке имеет малую продолжительность, обычно это передача одного слова по шине. После обслуживания заявки происходит освобождение ресурса. Такой процесс может повторяться многократно, пока не прекратится выполнение команды. Аппаратно реализованный в блоке приоритетов процессора NM6403 алгоритм обслуживания таких заявок

рассмотрен в пункте 8.3.2.

8.2 Конвейерное выполнение основных типов команд

Процессор NM6403 имеет восьмиуровневый конвейер, причём все восемь ступеней занимаются только векторными командами, скалярные используют лишь первые пять. На каждой ступени конвейера может находиться только одна скалярная команда. Специальный механизм блокировок гарантирует, что скалярные команды заканчиваются в том же порядке, в каком были выбраны из памяти. Векторные команды также поступают в конвейер в том же порядке, что и выбирались. Однако, поскольку они занимают конвейер на несколько тактов (от 1 до 32 в зависимости от кода команды), заканчиваться они могут уже в другом порядке. Наряду со скалярной командой на той же стадии конвейера может присутствовать одна или несколько векторных команд (до пяти). Ограничением в этом случае служит наличие необходимых ресурсов, а также условие, что все эти команды не претендуют на один и тот же ресурс. Невыполнение этих требований приводит к тому, что позже выбранная команда остаётся на предыдущей стадии конвейера и ждёт, пока не освободится нужный ресурс.

Рассмотрим на примерах работу конвейера. На Рис. 8-1 показано выполнение векторной и скалярной команды. Каждая из них проходит стадию выборки (FETCH) и дешифрации (IR0). На уровень конвейера IR1 попадает в данный момент времени только одна команда, раньше других выбранная из памяти. В случае, когда имеются две 32-разрядные команды в одном 64-разрядном слове, первой в конвейер запускается та, что находится в младшей части (выборка из памяти команд производится по 64-разряда). Пусть это будет векторная команда. В следующем такте векторная команда проходит на IR2, а скалярная может попасть на IR1,если отсутствует конфликт по ресурсам несмотря на то, что векторная продолжает занимать данную стадию конвейера столько тактов, сколько определено в коде команды. Ещё через такт векторная команда пройдёт на уровень IR3, продолжая занимать IR1 и IR2, а скалярная - на IR2 и т.д. Таким образом, возможно выполнение одновременно векторной и скалярной команды.

На Рис. 8-2 показан случай выполнения одновременно двух векторных команд. Как и в предыдущем случае, на IR1 сначала попадёт только одна команда, раньше других выбранная из памяти. В следующем такте она пройдёт на IR2, продолжая занимать на IR1. Вторая команда может попасть на IR1 в случае отсутствия конфликта по ресурсам. Первая команда является опрерационной, поэтому она последовательно проходит по всем стадиям конвейера вплоть до IR6, причём занимает каждую стадию столько тактов, сколько определено командой. В этом случае второй командой,

одновременно с ней выполняющейся, может быть только команда ввода/вывода, которой не требуются стадии IR4 - IR6.

Более подробно о ковейерном выполнении команд основных типов говорится в следующих подразделах.

Рис. 8-1. Общий вид конвейера процессора NM6403 и выполнение в нём скалярных и векторных команд

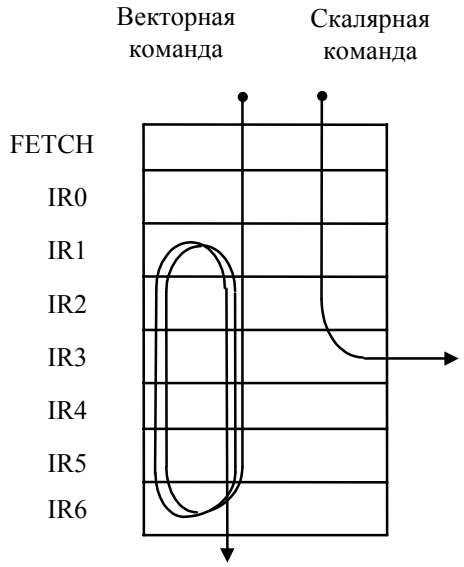
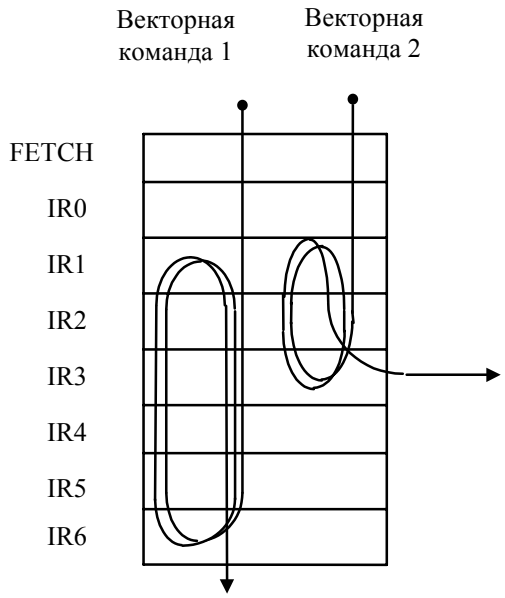


Рис. 8-2. Одновременное выполнение двух векторных команд в конвейере



8.2.1 Конвейерное выполнение скалярных команд

Конвейер для скалярных команд представлен на Рис. 8-3. Пример конвейерного выполнения данных команд можно видеть на Рис. 8-4.

В упрощенном виде основной конвейер выполнения скалярных команд процессора NM6403 состоит из следующих стадий:

F - фаза выборки команды из внешней памяти в буфер команд;

D - фаза выборки команды из буфера команд, дешифрация команды, проверка возможности начала ее выполнения, захват необходимых ресурсов;

A - фаза адресных вычислений с установкой значений адресных регистров (AR-регистров), формирование запросов на работу с памятью, увеличение счетчика адреса программы и установка его на шину для выборки комад;

E - фаза скалярных вычислений с получением значений регистров общего назначения (GR-регистров) и выполнения обмена с внешней памятью данными;

W - фаза записи в регистры из внешней памяти и завершения команды.

Рис. 8-3. Конвейер выполнения скалярных команд

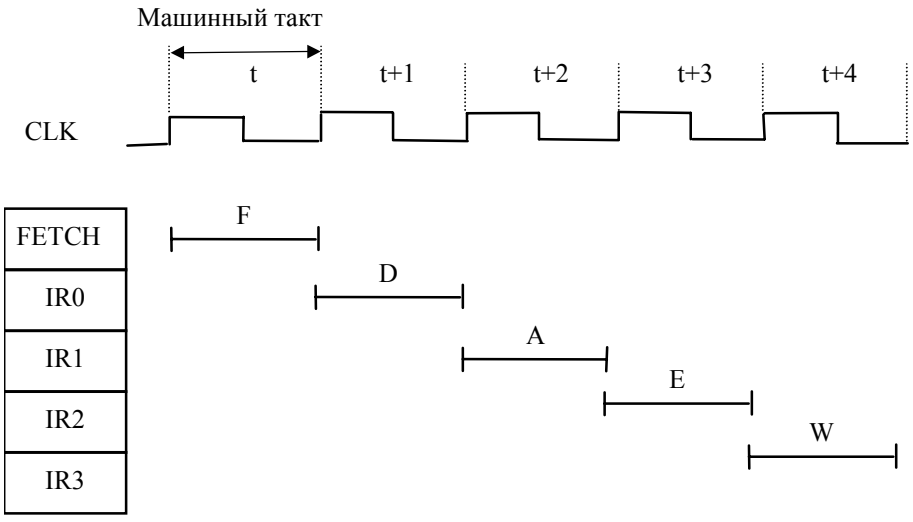


Рис. 8-4. Пример конвейерного выполнения скалярных команд

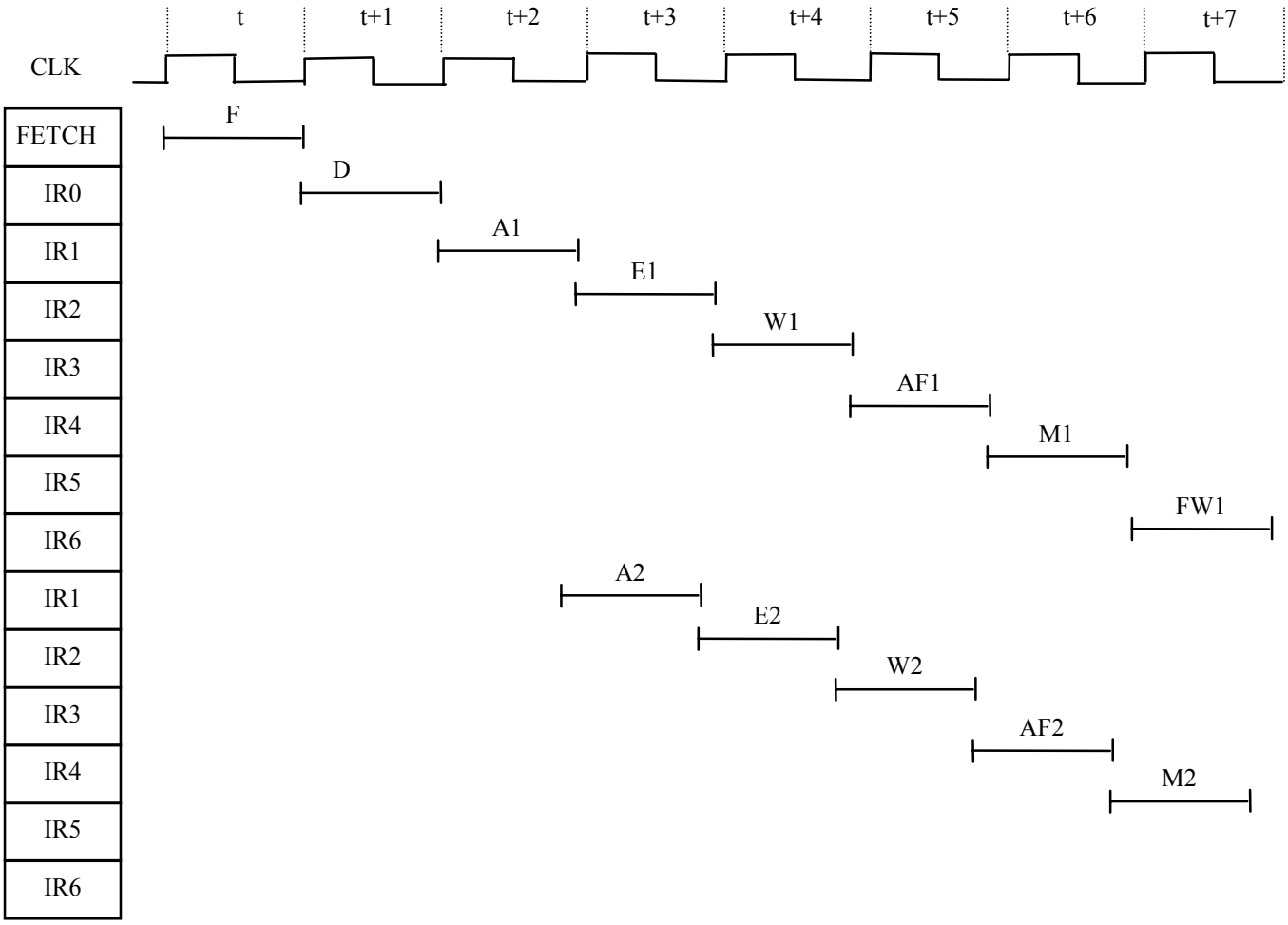
```
r0 <- r1 + r2          FDAEW
r1 <- r2 + r3           FDAEW
r2 <- r3 + r4            FDAEW
```

8.2.2 Конвейерное выполнение векторных команд

Ковейер для основных типов векторных команд приводится на Рис. 8-5, Рис. 8-6, Рис. 8-7. В зависимости от типа требуется от 4 до 8 фаз конвейера. В общем случае основной конвейер выполнения векторных команд процессора NM6403 состоит из следующих стадий:

- F - фаза выборки команды из внешней памяти в буфер команд;
- D - фаза выборки команды из буфера команд, дешифрация команды, проверка возможности начала ее выполнения, захват необходимых ресурсов;
- Ai - фаза адресных вычислений с установкой значений адресных регистров (AR-регистров), формирование запросов на работу с памятью, чтение из AFIFO;
- Ei - фаза выполнения обмена с внешней памятью данными;
- Wi - фаза записи в регистры векторных операндов данных из внешней памяти, RAM или AFIFO с возможностью их маскирования, чтения весов из WFIFO в память весовых коэффициентов WBUF, записи в RAM или WFIFO из внешней памяти;
- AFi - фаза выполнения функций активации над входными операндами;
- Mi - фаза взвешенного суммирования с формированием промежуточного результата (двухрядного кода), перезаписи весов из WBUF в операционную память весовых коэффициентов WOPER;
- FWi - фаза получения окончательного результата взвешенного суммирования (преобразования двухрядного кода в однорядный) или обычной арифметической операции, записи его в AFIFO и завершения команды.

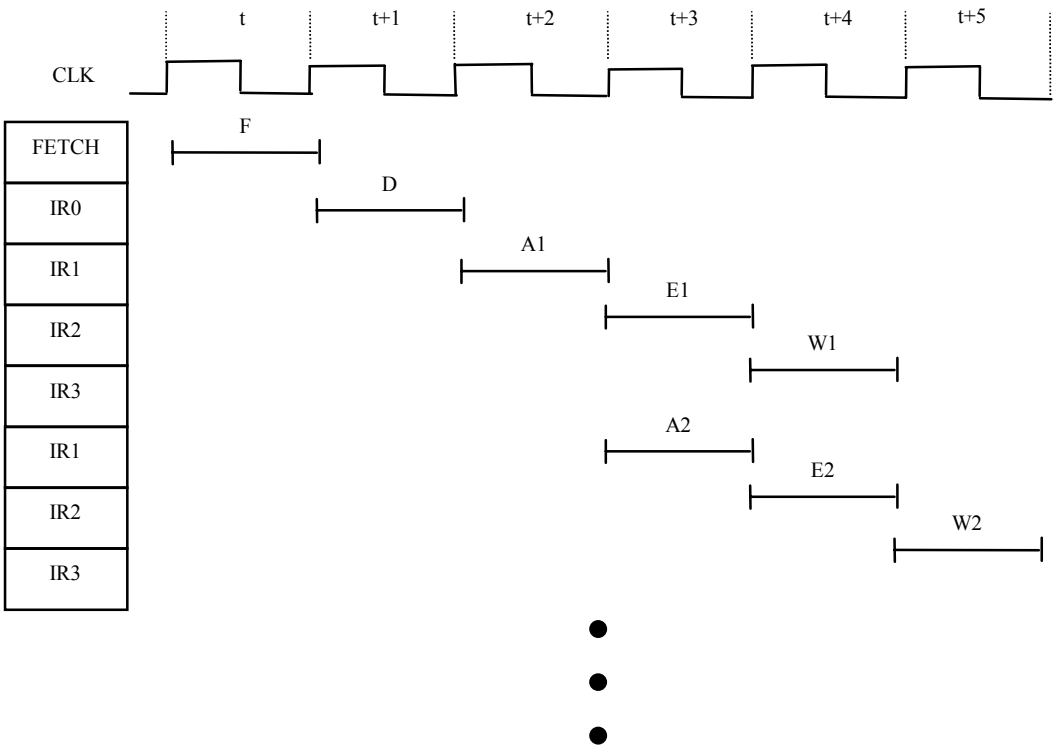
Рис. 8-5. Конвейер выполнения обрабатывающей векторной команды и его условное обозначение на временных диаграммах



Условное обозначение на временных диаграммах:

F	D	A1	E1	W1	AF1	M1	FW1						
			A2	E2	W2	AF2	M2	FW2					
•	•	•	•	•	•	•	•	•	•	•			
							An	En	Wn	AFn	Mn	FWn	

Рис. 8-6. Конвейер выполнения векторной команды загрузки из внешней памяти в WFIFO или RAM



Условное обозначение на временных диаграммах:

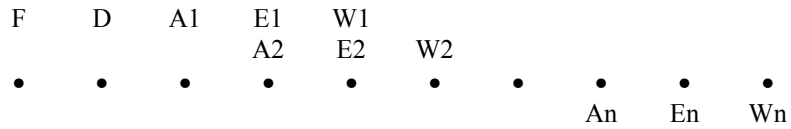
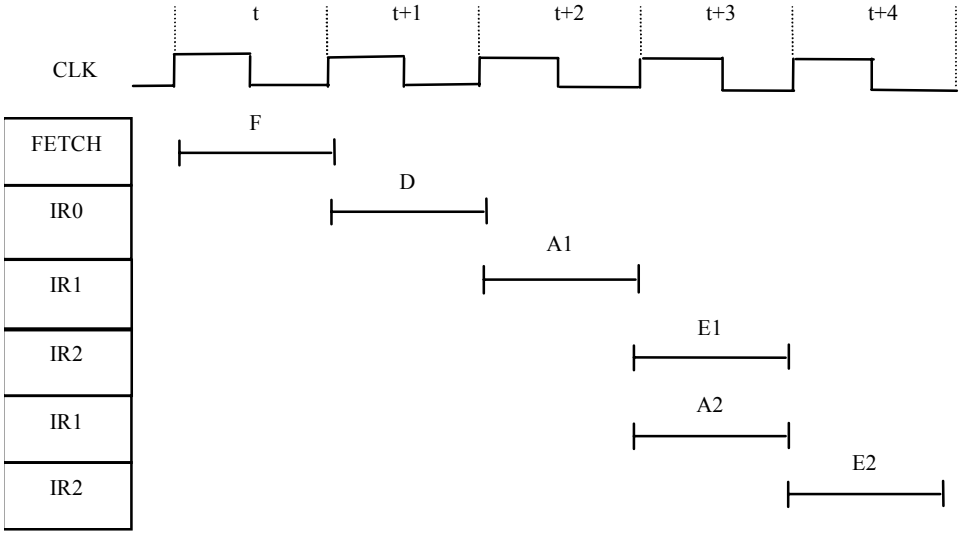
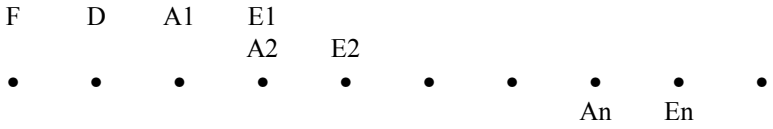


Рис. 8-7. Конвейер выполнения векторной команды записи во внешнюю память из AFIFO



Условное обозначение на временных диаграммах:



8.2.3 Конвейерное выполнение команд управления

К командам управления процессора NM6403 относятся:

- переход условный или безусловный;
- переход к подпрограмме условный или безусловный;
- возврат из подпрограммы условный или безусловный;
- возврат из прерывания условный или безусловный.

Примеры конвейерного выполнения перечисленных выше команд представлены на Рис. 8-8, Рис. 8-9, Рис. 8-10, Рис. 8-11. Основными стадиями конвейера для команд управления являются:

F - фаза выборки команды из внешней памяти в буфер команд,

D - фаза выборки команды из буфера команд, дешифрация команды, проверка возможности начала ее выполнения, захват необходимых ресурсов;

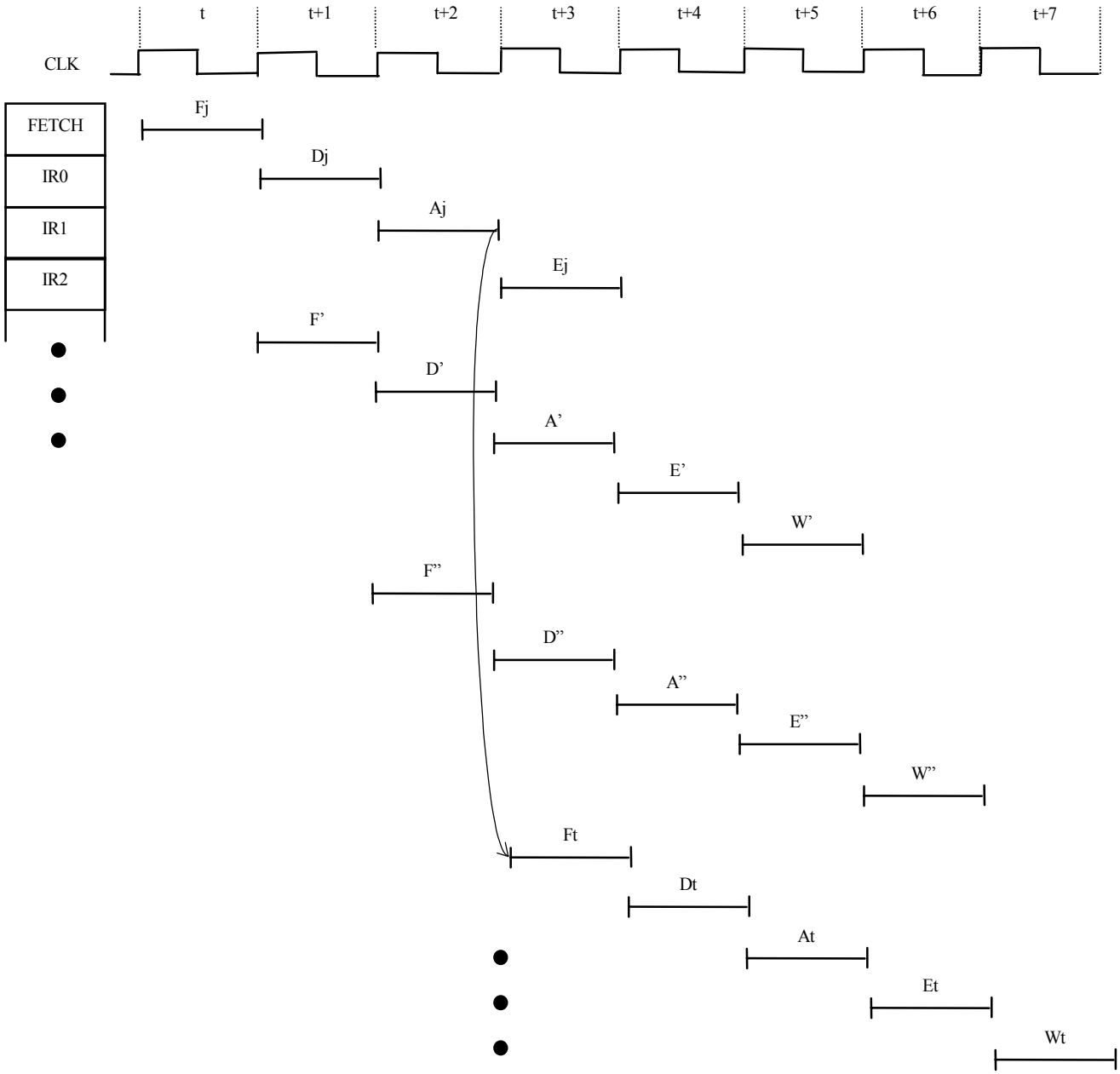
A - фаза проверки условий для условных команд. Для них, когда условие выполнено и для безусловных команд происходит также следующее: вычисление адреса перехода для команд групп 1 и 2 и формирование запроса на чтение по нему команды; установка запроса на запись в память адреса возврата и слова состояния процессора по указателю стека с его постинкрементацией для группы 2 или на их чтение по указателю стека с его декрементацией для групп 3 и 4. Если условие не выполняется, происходит только инкрементация старого содержимого счётчика команд и формирование по нему запроса на чтение очередной команды,

E - фаза скалярных вычислений с получением значений регистров общего назначения (GR-регистров) и выполнения обмена с внешней памятью, если он был задан на предыдущей фазе. Заданные командой скалярные вычисления выполняются не зависимо от результата проверки условий,

W - - фаза записи в счётчик команд для команд групп 3 и 4, а также в слово состояния процессора только для группы 4 нового содержимого из стека и формирование запроса на выборку команды, если данные команды прошли проверку выполнения условий на фазе *A*.

Как видно из рисунков, при выполнении команд управления в конвейере возникает задержка при вычислении нового содержимого счётчика команд: для групп 1 и 2 - 2 такта, для групп 3 и 4 - 3 такта. Чтобы не терять эти такты, используются отложенные команды.

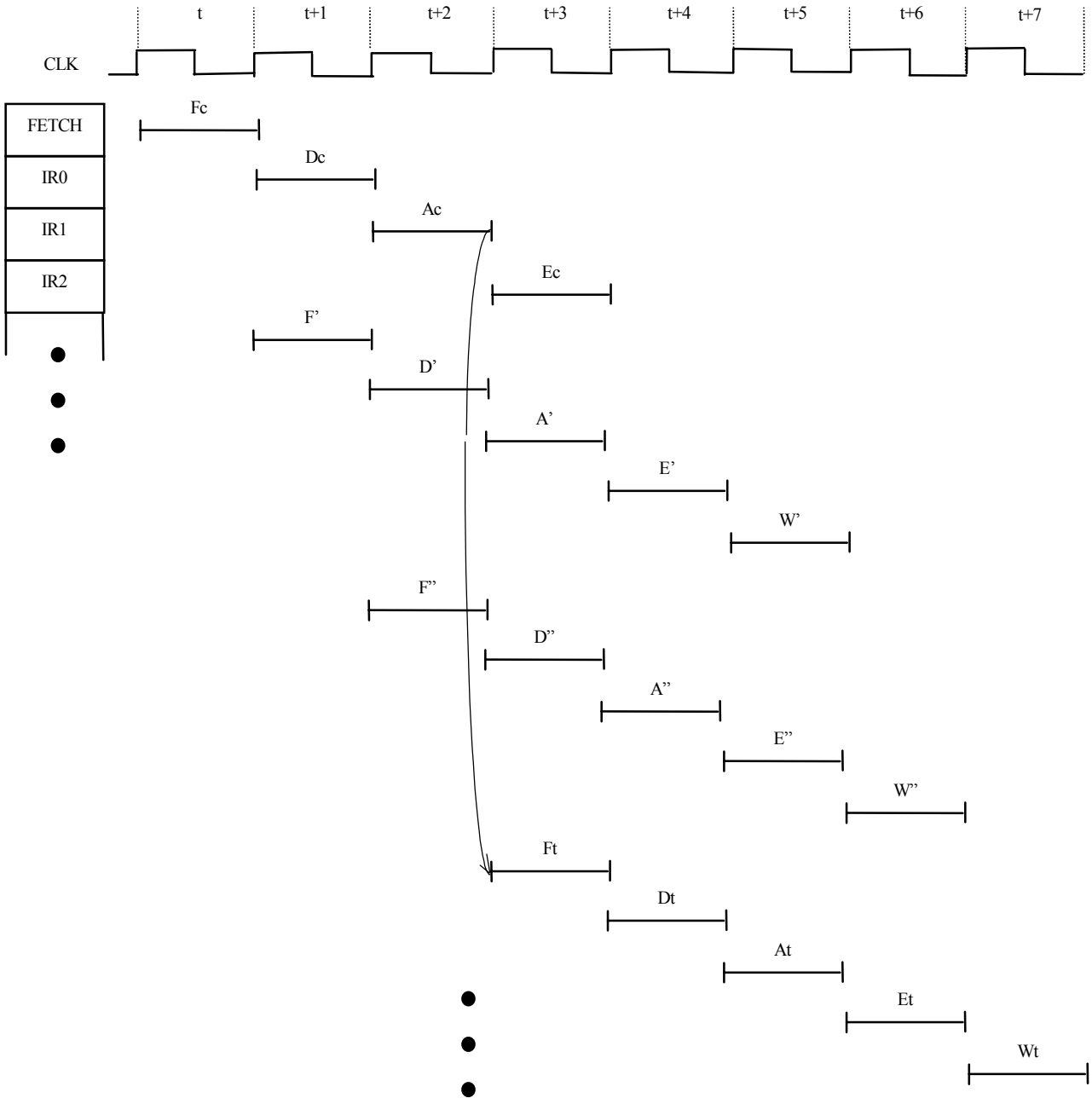
Рис. 8-8. Конвейер выполнения команд безусловного и условного перехода, если условие выполняется



Условное обозначение на временных диаграммах:

Fj	Dj	Aj	Ej				-	команда перехода;
'	F	D'	A'	E'	W'		-	первая отложенная команда;
		F"	D"	A"	E"	W"	-	вторая отложенная команда;
			Ft	Dt	At	Et	Wt	- команда, выбранная по адресу перехода.

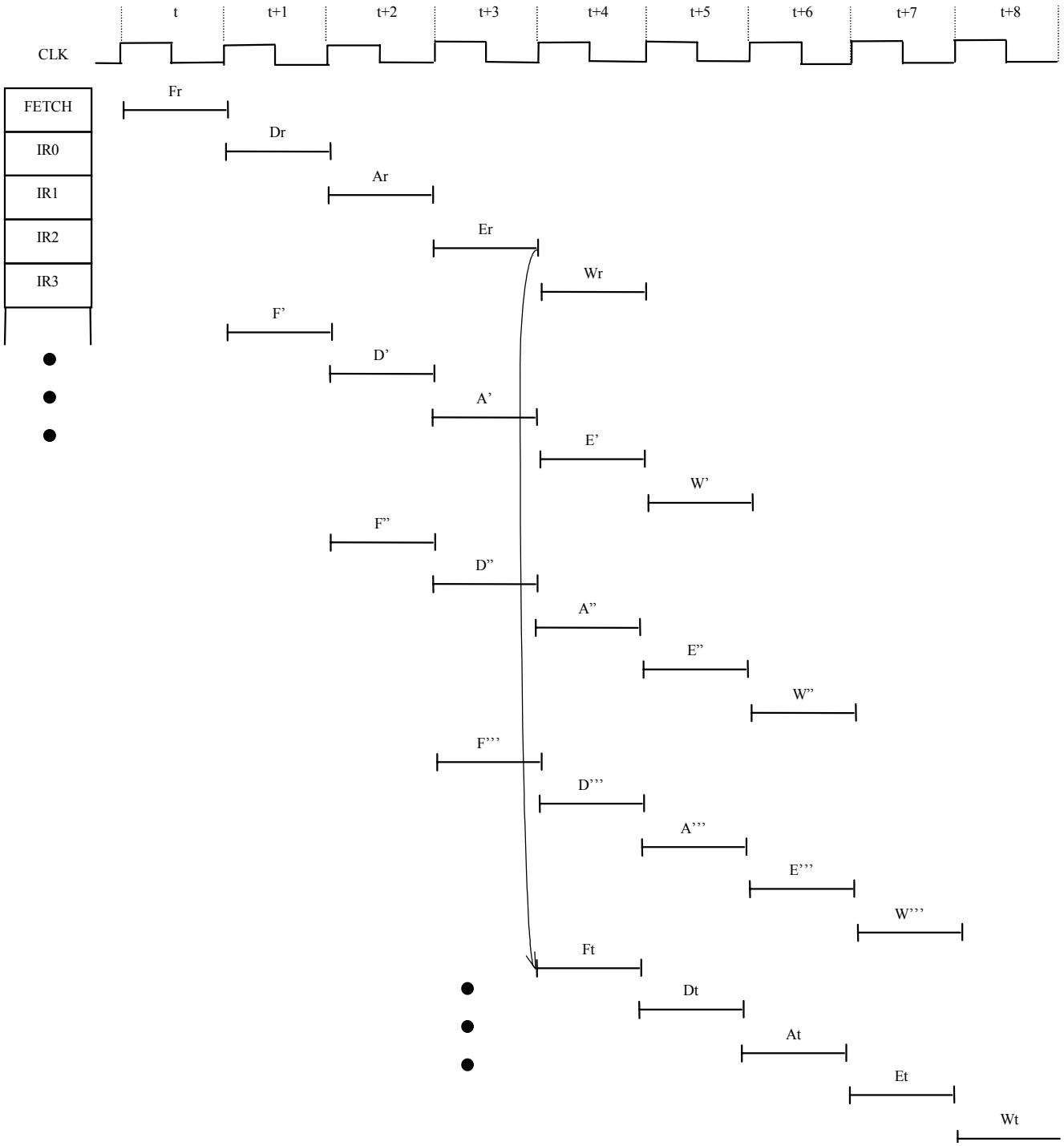
Рис. 8-9. Конвейер выполнения команд безусловного и условного перехода к подпрограмме, если условие выполняется



Условное обозначение на временных диаграммах:

Fc	Dc	Ac	Ec				-	команда перехода к подпрограмме;
	F'	D'	A'	E'	W'		-	первая отложенная команда;
		F"	D"	A"	E"	W"	-	вторая отложенная команда;
			Ft	Dt	At	Et	Wt	- первая команда подпрограммы.

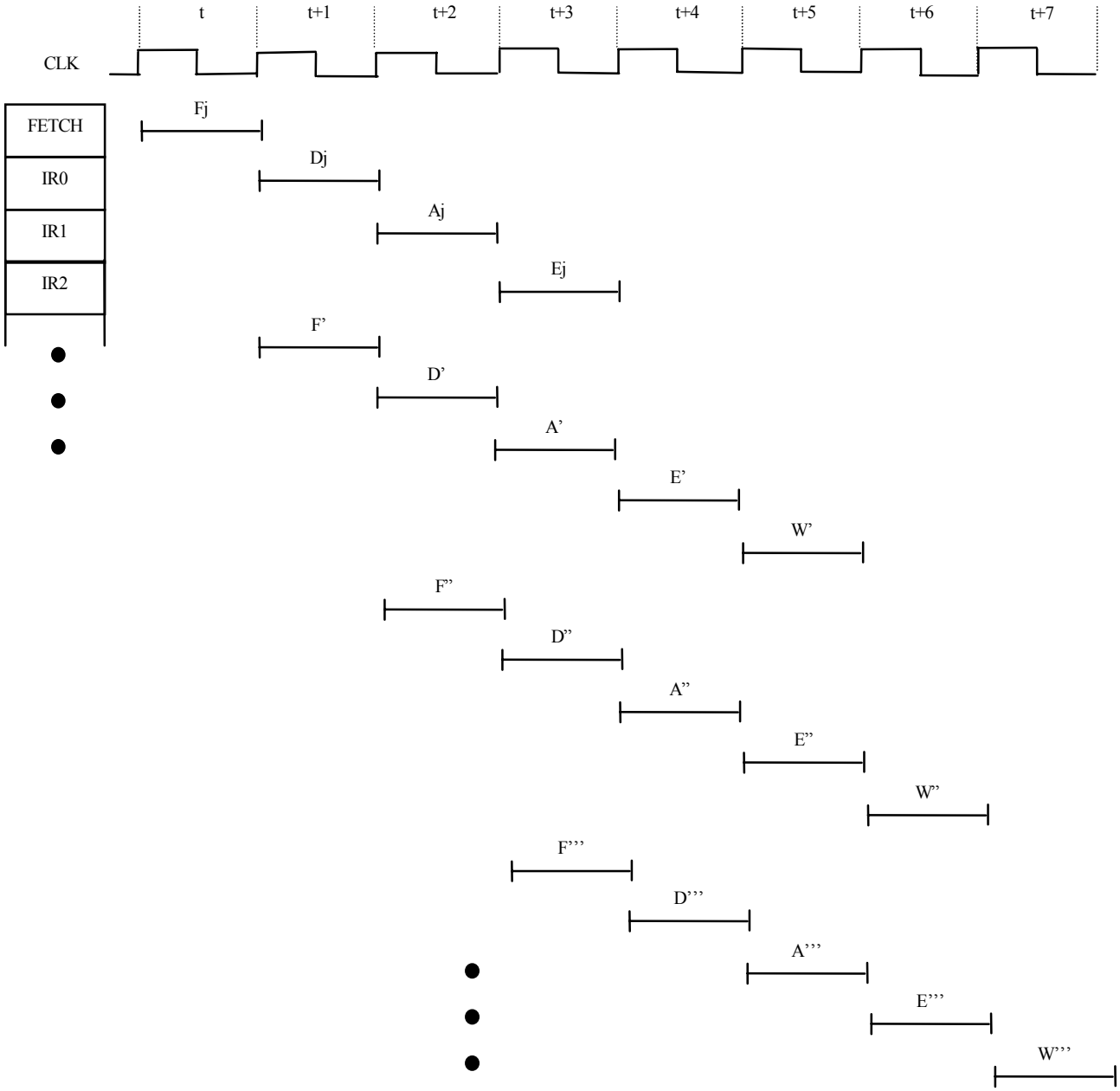
Рис. 8-10. Конвейер выполнения команд безусловного и условного возврата из подпрограммы или прерывания, если условие выполняется



Условное обозначение на временных диаграммах:

Fr	Dr	Ar	Er	Wr			-	команда возврата из подпрограммы/прерывания;
	F'	D'	A'	E'	W'		-	первая отложенная команда;
		F"	D"	A"	E"	W"	-	вторая отложенная команда;
		F’"	D"'	A"'	E"'	W"'	-	третья отложенная команда;
			Ft	Dt	At	Et	Wt	- команда, выбранная по адресу возврата.

Рис. 8-11. Конвейер выполнения условных команд управления, если условие не выполняется



Условное обозначение на временных диаграммах:

Fj	Dj	Aj	Ej				- команда условного перехода;
	F'	D'	A'	E'	W'		- первая отложенная команда;
		F"	D"	A"	E"	W"	- вторая отложенная команда;
		F’"	D'"	A'"	E'"	W'"	- следующая команда в программе.

8.3 Конфликты, возникающие при конвейерном выполнении команд

Данный раздел посвящён описанию конфликтов при работе конвейера, связанным с особенностью работы процессора NM6403 при подкачке команд в буфер команд и запуске их оттуда в конвейер на исполнение. Для программистов он представляет особый интерес, поскольку знание причин возникновения данных конфликтов помогает в большом количестве случаев оптимизировать программный код таким образом, что удаётся их избежать и тем самым заметно повысить производительность. Особенно это актуально для векторных команд, так как их правильное использование может увеличить скорость вычислений в 2-4 раза.

8.3.1 Конфликты при выборке команд из памяти в буфер команд

Буфер команд процессора NM6403 имеет размер 2*64 разряда и может хранить максимум либо четыре 32-разрядные команды, либо две
64-разрядные, либо одну 64-разрядную и две 32-разрядные. Выбор команд из памяти идёт всегда по 64 разряда (отсюда требование располагать 64-разрядные команды и осуществлять переходы только по чётному адресу). Если при этом выбрались две 32-разрядные команды, на исполнение пойдёт команда, находящаяся в младшей части, затем та, что в старшей части, и только после этого возможна выборка следующих команд на освободившееся место. Запуск в конвейер 64-разрядной команды даёт возможность сразу же произвести новую выборку команд. Запрос на чтение из памяти очередной команды формируется каждый раз, если свободна хоть одна 64-разрядная ячейка в буфере. Если он удовлетворяется, счётчик адреса команд автоматически инкрементируется. Исключением являются только два случая:

- после системного сброса не происходит выборки команд, хотя буфер пустой, пока не произойдёт инициализация процессора;
- если в буфере находится любая команда управления, она не пойдёт на исполнение, пока буфер команд не станет полным. Этот механизм гарантирует выборку всех необходимых отложенных команд прежде, чем будет изменён счётчик адреса команд.

Ёмкость буфера команд была выбрана с таким расчётом, чтобы иметь оптимальное количество отложенных команд при передаче управления - минимум одну 64-разрядную, максимум - три 32-разрядные. Как было отмечено в пункте 8.2.3, для работы конвейера без пустых тактов каждой команде перехода/перехода к подпрограмме желательно иметь 2 отложенные, а возврату из подпрограммы/прерывания - 3 отложенные команды. От

того, как расположить команды управления в памяти, и зависит число отложенных команд. Если команда управления 64-разрядная (содержит 32-разрядный адрес перехода или смещение), отложенных команд может быть: одна 64-разрядная или две 32-разрядные. Если она 32-разрядная и находится в старшей половине 64-разрядного слова, количество отложенных команд определяется так же, как и в первом случае. Если она 32-разрядная и находится в старшей половине, возможны следующие варианты количества отложенных команд: либо одна 32-разрядная и одна 64-разрядная, либо три 32-разрядных. Таким образом, путём простой перестановки команд можно добиться оптимального количества отложенных команд. Слишком большое их число трудно эффективно использовать, малое их число может привести к простоям конвейера.

Следующая проблема опять связана с ёмкостью буфера команд. При выборке 64-разрядных команд возможен пустой такт в конвейере. Это связано с тем, что, произойдёт ли запуск команды в на исполнение и освободится ли буфер команд, узнаётся только на второй стадии конвейера (дешифрация). Только на третьей стадии формируется запрос на выборку новой команды, а на четвёртой произойдёт сама выборка. Таким образом, если бы буфер команд был объёмом 1*64 разряда, между двумя выборками команд в лучшем случае был бы пропуск в два такта. Так как у нас буфер имеет объём 2*64 разряда, то мы имеем в общем случае пропуск в один такт. Если используются команды только 32-разрядные, то это не имеет значения, поскольку они выбираются из памяти по две за раз. Проблема существует только тогда, когда выбираются 64-разрядные команды из памяти с темпом в один такт. При этом в конвейере будут иметься пустые такты - по одному пустому такту на одну 64-разрядную команду.

8.3.2 Конфликты, связанные с использованием вычислительных ресурсов

При дешифрации команды, выбранной из буфера команд, проверяется одновременно ее корректность и условия, при которых эта команда может быть запущена на выполнение в конвейер процессора NM6403. Эти действия производятся на фазе конвейера, называемой IR0.

Корректность команды на фазе IR0 контролируется только для векторных команд и состоит в проверке наиболее опасных для функционирования ситуаций, приводящих к самоблокировке процессора NM6403. Такие запрещенные ситуации разделяются на две группы:

- статические запрещенные ситуации, они связаны только с неправильным заданием операндов команды, не зависят от состояния процессора NM6403 в данный момент времени;

- динамические запрещенные ситуации, их возникновение зависит от состояния процессора NM6403 в данный момент времени.

Статические запрещенные ситуации:

- одновременное чтение и запись во внутреннее ОЗУ данных (RAM);
- использование данных из внешней памяти для операций в векторном процессоре, но при этом не задана операция чтения из внешней памяти.

Динамические запрещенные ситуации:

- чтение из пустого AFIFO;
- запись в непустое AFIFO, если не задана операция любого чтения из AFIFO в той же команде;
- использование в качестве операндов для операций в векторном процессоре RAM и AFIFO, в которых находится разное число 64-разрядных данных;
- использование в качестве операндов для операций в векторном процессоре RAM и данные из внешней памяти, причём число 64-разрядных данных в RAM не совпадает с кодом в команде, который задаёт количество обращений к внешней памяти;
- использование в качестве операндов для операций в векторном процессоре AFIFO и данные из внешней памяти, причём число 64-разрядных данных в AFIFO не совпадает с кодом в команде, который задаёт количество обращений к внешней памяти;
- задание одновременно векторной операционной команды арифметической или взвешенного суммирования и перезаписи весов из теневой матрицы в рабочую.

Эти ситуации могут привести к самоблокировке конвейера процессора NM6403, ликвидировать которую можно только с помощью системного сброса. В случае обнаружения такой векторной команды вместо неё в конвейер попадает команда “нет операции” и фиксируется запрос на прерывание по ошибке при выполнении операций в векторном процессоре, если соответствующая этому прерыванию маска в регистре PSWR равна единице.

Условия запуска команд на выполнение, захват и освобождение ресурсов

Команда может попасть в конвейер только в том случае, когда соблюдаются все перечисленные ниже условия:

- отсутствует блокировка от фазы IR1 конвейера;
- если в команде сброшен бит параллельной работы (31-й разряд команды равен нулю), то на момент запуска данной команды в конвейер должны закончиться все векторные операции на уровне IR1;
- на момент запуска данной команды в конвейер отсутствуют конфликты по ресурсам с выбранными ранее командами.

Если эти условия не выполняются, команда остаётся в буфере, и следующая выбранная команда не сможет попасть в конвейер в обход неё.

Ниже приведена Табл. 8-1, в которой для каждого типа команд перечисляются те требуемые командой ресурсы, по которым может произойти конфликт в конвейере.

Табл. 8-1. Возможные конфликты по ресурсам при запуске команды в конвейер

КОМАНДА	INSTR. BUS	INP. DATA BUS	VECT. DATA BUS	OUTP. DATA BUS	AR источник	GR источник	AR приёмник	AGU ½	R источник	R приёмник	GR7	GR приёмник	BIT C	FLAGS	SP	WFIFO	OU	AFIFO
Формат 1.1 (чтение из памяти)	-	+	-	-	+	+	+	+ / +	-	+	+	+	+	-	-	-	-	-
Формат 1.1 (запись в память)	-	-	-	+	+	+	+	+ / +	+	-	+	+	+	-	-	-	-	-
Формат 1.2 (чтение из памяти)	-	+	-	-	+	-	+	+ / +	-	+	+	+	+	-	-	-	-	-
Формат 1.2 (запись в память)	-	-	-	+	+	-	+	+ / +	+	-	+	+	+	-	-	-	-	-
Формат 2.1	-	+	-	+	-	-	-	- / -	+	+	+	+	+	-	-	-	-	-
Формат 2.2	-	+	-	+	-	-	-	- / -	-	+	+	+	+	-	-	-	-	-

Табл.8-1. Возможные конфликты по ресурсам при запуске команды в конвейер (Продолжение)

КОМАНДА	INSTR. BUS	INP. DATA BUS	VECT. DATA BUS	OUTP. DATA BUS	AR источник	GR источник	AR приёмник	AGU ½	R источник	R приёмник	GR7	GR приёмник	BIT C	FLAGS	SP	WFIFO	OU	AFIFO
Формат 2.3	-	+	-	+	-	-	-	- / -	-	-	+	+	+	-	-	-	-	-
Формат 3.1	-	-	-	-	+	+	+	+ / +	-	-	+	+	+	-	-	-	-	-
Формат 3.2	-	-	-	-	+	-	+	+ / +	-	-	+	+	+	-	-	-	-	-
Формат 3.3	-	-	-	-	-	-	-	- -	-	-	+	+	+	-	-	-	-	-
Формат 3.4	-	-	-	-	-	-	-	- -	-	-	+	+	+	-	-	-	-	-
Формат 4.1 (переход)	-	-	-	-	+	+	-	+ / +	-	-	+	+	+	+	-	-	-	-
Формат 4.1 (переход к п/п)	-	-	-	+	+	+	-	+ / +	+	-	+	+	+	+	+	-	-	-
Формат 4.2 (переход)	-	-	-	-	+	-	-	+ / +	-	-	+	+	+	+	-	-	-	-
Формат 4.2 (переход к п/п)	-	-	-	+	+	-	-	+ / +	+	-	+	+	+	+	+	-	-	-
Формат 4.3 (возврат из п/п)	-	+	-	-	+	-	+	- +	-	-	+	+	+	+	-	-	-	-

Табл.8-1. Возможные конфликты по ресурсам при запуске команды в конвейер (Продолжение)

КОМАНДА	INSTR. BUS	INP. DATA BUS	VECT. DATA BUS	OUTP. DATA BUS	AR источник	GR источник	AR приёмник	AGU ½	R источник	R приёмник	GR7	GR приёмник	BIT C	FLAGS	SP	WFIFO	OU	AFIFO
Формат 4.3 (возврат из прерывания)	-	+	-	-	+	-	+	- / +	-	-	+	+	+	+	-	-	-	-
Формат 5.1 (чтение из памяти)	+	+	+	-	+	+	+	+ / +	-	-	-	-	-	-	-	+	+	+
Формат 5.1 (запись в память)	+	-	+	+	+	+	+	+ / +	-	-	-	-	-	-	-	+	+	+
Формат 5.2	-	+	+	-	+	+	+	+ / +	-	-	-	-	-	-	-	+	+	+
Формат 5.3	-	-	+	-	-	-	-	- / -	-	-	-	-	-	-	-	+	+	+

Далее каждый из ресурсов, способный стать источником конфликта, описывается более подробно.

Внутренние шины

- Ресурс - INSTRUCTION BUS - занимается только векторными командами :
  - ♦ формата 5.1, которые читают из внешней памяти операнд для операций в OU;
  - ♦ формата 5.1, которые пишут из внешней памяти или AFIFO в RAM.

Ресурс освобождается с последним тактом выполнения данных команд на IR1.

- Ресурс - INPUT DATA BUS (WEIGHT BUS) - занимается только

векторными командами:

- ♦ формата 5.1, которые читают из внешней памяти операнд для операций в OU и при этом требуется чтение другого операнда из RAM;
- ♦ формата 5.2, которые пишут из внешней памяти в WFIFO.

Ресурс освобождается с последним тактом выполнения данных команд на IR1.

- Ресурс - VECTOR DATA BUS - занимается только векторными командами:
  - ♦ формата 5.1, которые читают из внешней памяти операнд для операций в OU;
  - ♦ формата 5.1, которые пишут из внешней памяти или AFIFO в RAM;
  - ♦ форматов 5.1;5.2;5.3, в которых требуется чтение операнда из RAM для операций в OU.

Ресурс освобождается с последним тактом выполнения данных команд на IR1.

- Ресурс - OUTPUT DATA BUS - занимается только векторными командами формата 5.1, которые пишут во внешнюю память из AFIFO. Ресурс освобождается с последним тактом выполнения данных команд на IR1.

Адресные ресурсы

- Ресурс - AR источник - может потребоваться командами форматов 1.1; 1.2; 3.1; 3.2; 4.1; 4.2; 4.3 (для данного формата это SP); 5.1; 5.2 в качестве источника адреса при соответствующем методе адресации/модификации адресного регистра. Он может стать источником конфликта в следующих случаях:
  - ♦ данный адресный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1;
  - ♦ данный адресный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока конфликт не разрешится.

- Ресурс - GR источник - может потребоваться командами форматов 1.1; 3.1; 4.1; 5.1; 5.2 в качестве источника адреса при соответствующем методе адресации/модификации адресного

регистра. Он может стать источником конфликта в следующих случаях:

- ♦ данный РОН используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1;
- ♦ данный РОН используется в качестве R приёмника для команд форматов 1.1;1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера;
- ♦ данный РОН используется в качестве GR приёмника для команд форматов 1,2,3 или 4, задающих операцию в RALU на уровне IR1 (для команд шагов умножения в качестве GR приёмника неявно используется также GR7);
- ♦ данный РОН используется в качестве GR приёмника для команд форматов 1,2,3 или 4, задающих операцию в RALU на уровне IR2 и есть блокировка этого уровня конвейера.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока конфликт не разрешится.

- Ресурс - AR приёмник - может потребоваться командами форматов 1.1; 1.2; 3.1; 3.2; 4.3 (для данного формата это SP); 5.1; 5.2, если данный адресный регистр модифицируется при соответствующем методе адресации/модификации адресного регистра. Он может стать источником конфликта в следующих случаях:
  - ♦ данный адресный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1;
  - ♦ данный адресный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока конфликт не разрешится.

- Ресурс - AGU1 или AGU2 - занимается только векторными командами формата 5.1. или 5.2., которые осуществляют обмен с внешней памятью, причём AGU1 - если 28-й разряд команды равен 0 и AGU2 - если 28-й разряд команды равен 1. Ресурс освобождается с последним тактом выполнения данных команд на IR1.

Данный ресурс может быть необходим для команд форматов 1.1; 1.2; 3.1; 3.2; 4.1; 4.2; 4.3 (для данного формата это AGU2); 5.1; 5.2 при вычислении/модификации адреса. В случае, когда используется конкретный адресный регистр или РОН в качестве источника или конкретный адресный регистр в качестве приёмника, то требуется

конкретный AGU: AGU1 - если 28-й разряд команды равен 0 и AGU2 - если 28-й разряд команды равен 1. Если в соответствии с методом адресации адресный регистр не меняется и для вычисления адреса „не используются ни адресный регистр, ни РОН, то необходимо наличие хоть одного свободного AGU.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока ресурс не освободится.

Ресурсы, требуемые при операциях пересылок данных

- Ресурс - R источник - может потребоваться командами форматов 1.1; 1.2; 2.1; 4.1 и 4.2. (для данных форматов это PSWR) в качестве источника при операциях пересылок данных. Он может стать причиной конфликта в следующих случаях:
  - ♦ данный регистр занят в качестве AR приёмника для векторных команд формата 5.1 или 5.2 на IR1 и освобождается с последним тактом выполнения данных команд на этом уровне;
  - ♦ данный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1 (это не касается регистров T0,T1,DIR0/DOR0, DIR1/DOR1);
  - ♦ данный регистр используется в качестве R приёмника для команд форматов 1.1;1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера (это не касается регистров T0,T1,DIR0/DOR0, DIR1/DOR1);
  - ♦ данный регистр - это PSWR и на IR1 есть команда формата 1, 2, 3 или 4, меняющая признаки по результату операции в RALU;
  - ♦ данный регистр - это PSWR, на IR2 есть команда формата 1, 2, 3 или 4, меняющая признаки по результату операции в RALU, и есть блокировка этого уровня конвейера;
  - ♦ данный регистр - это PSWR и на IR1 или IR2 есть команда формата 2.3.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока конфликт по нему не разрешится или он не освободится.

- Ресурс - R приёмник - может потребоваться командами форматов 1.1; 1.2; 2.1; 2.2 в качестве приёмника при операциях пересылок данных. Он может быть занят в следующих случаях:
  - ♦ данный регистр используется в качестве AR источника/приёмника или GR источника векторными командами форматов 5.1 или 5.2 при вычислении адреса на

уровне IR1, и освобождается с последним тактом выполнения данных команд на этом уровне;

- данный регистр - это NB или SB, а в конвейере ещё не закончилась заданная векторной командой запись весовых коэффициентов из WFIFO в OU или не отработал бит “LOAD” на IR1;
- данный регистр - это VR, а в конвейере ещё не закончилась векторная команда на IR1, использующая его при операциях с OU;
- данный регистр - это F1CR, а в конвейере ещё не закончилась векторная команда на IR1, использующая его при операциях с OU;
- данный регистр - это F2CR, а в конвейере ещё не закончилась векторная команда на IR1, использующая его при операциях с OU.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока ресурс не освободится.

Ресурсы, требуемые при операциях в RALU

- Ресурс - GR7 - используется командами форматов 1,2,3 или 4, задающими в RALU операцию “шаг умножения” или “первый шаг умножения”, в качестве одного из операндов. Он может стать источником конфликта в следующих случаях:
  - данный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1;
  - данный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока конфликт не разрешится.

- Ресурс - GR приёмник - используется командами форматов 1, 2, 3 или 4, задающими операцию в RALU, в качестве приёмника результата (для команд шагов умножения в качестве GR приёмника неявно используется также GR7). Он может быть занят в том случае, если данный регистр используется в качестве GR источника векторными командами форматов 5.1 или 5.2 при вычислении адреса на уровне IR1, и освобождается с последним тактом выполнения данных команд на этом уровне.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока ресурс не освободится.

- Ресурс - BIT C (признак переноса в регистре PSWR) - требуется командами форматов 1,2,3 или 4, задающими операцию в RALU с учётом этого признака (логический сдвиг через “C”, арифметические операции с участием “C”, “шаг умножения”). Он может стать источником конфликта в следующих случаях:

  - ♦ PSWR используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1;
  - ♦ PSWR используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера.
  - ♦ на уровне IR1 находится команда формата 2.3;
  - ♦ на IR2 - команда формата 2.3 и есть блокировка этого уровня конвейера.

Ресурсы, требуемые командами управления

- Ресурс - FLAGS (признаки в регистре PSWR) - требуется командами управления формата 4, если нужно определить условие их выполнения. Он может стать источником конфликта в следующих случаях:

  - ♦ PSWR используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1;
  - ♦ PSWR используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера;
  - ♦ на уровне IR1 или IR2 находится команда формата 2.3;
  - ♦ на IR1 - команда формата 1, 2, 3 или 4, меняющая признаки по результату операции в RALU;
  - ♦ на IR2 - команда формата 1, 2, 3 или 4, меняющая признаки по результату операции в RALU, и есть блокировка этого уровня конвейера;

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока конфликт не разрешится.

- Ресурс - SP - требуется командами перехода к подпрограмме формата 4.1 или 4.2 для обращения в память по указателю стека. Он может быть занят векторной командой на IR1, использующей его в качестве адресного регистра при работе с памятью, и освобождается с последним тактом выполнения команды на этом уровне. Этот ресурс также является источником конфликта в следующих случаях:

- данный адресный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR1;
- данный адресный регистр используется в качестве R приёмника для команд форматов 1.1; 1.2; 2.1; 2.2 на уровне IR2 и есть блокировка этого уровня конвейера.

Ресурсы, требуемые при операциях в OU

- Ресурс - WFIFO - необходим векторным командам формата 5, задающим запись весовых коэффициентов из WFIFO в OU. Он может быть занят в следующих случаях:
  - на уровне IR1 ещё не закончилась предыдущая векторная команда, пишущая весовые коэффициенты из WFIFO в OU;
  - не отработал бит “LOAD” на IR1.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока ресурс не освободится.

- Ресурс - OU - необходим векторным командам формата 5, если в них задана операция над данными в векторном процессоре или взведён бит “LOAD”(0-й разряд команды равен единице). Он может быть занят в следующих случаях:
  - на уровне IR1 ещё не закончилась предыдущая векторная команда, использующая OU;
  - не отработал бит “LOAD” на IR1.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока ресурс не освободится.

- Ресурс - AFIFO - необходим векторным командам формата 5, если для операции в OU он используется в качестве одного из операндов, а также формата 5.1, если задана запись в память. Он может быть занят, если на уровне IR1 ещё не закончилась предыдущая векторная команда, читающая из AFIFO.

Команда в буфере, требующая данный ресурс, будет оставаться там и не уйдёт в конвейер до тех пор, пока ресурс не освободится.

Блокировки выполнения команд при работе с условно занятыми ресурсами

Попавшие в конвейер команды на уровне IR1 могут выставить одновременно до двух запросов на работу по локальной и глобальной шинам (либо от двух векторных, либо от одной векторной и одной скалярной команды). Кроме того, может сформироваться запрос на выборку новой команды и до четырёх запросов на ПДП (прямой доступ в память) от каналов ввода/вывода коммуникационных портов. Таким образом, встаёт задача

произвести арбитраж и выбрать из этих семи запросов те, которые возможно обслужить в данный момент времени (в лучшем случае - два, т.к. внешних шин только две). Выбор ведётся исходя из того, какие ресурсы (внутренние и внешние шины) необходимы для удовлетворения каждого из запросов и не заняты ли они, а также в зависимости от приоритета. Если запрос претендует на некоторые ресурсы и хоть один из них занят, то он не обслуживается и не влияет на остальные запросы не зависимо от его приоритета. Ниже более подробно описываются запросы в порядке убывания приоритета на обслуживание:

- Запрос на ПДП от канала ввода коммуникационного порта 1 (запись в память).

Данный запрос претендует на следующие ресурсы:

  - ♦ внутреннюю выходную шину OUTPUT DATA BUS;
  - ♦ одну из внешних шин - LOCAL BUS или GLOBAL BUS - в зависимости от старшего разряда адреса ПДП.

Если данный запрос сформирован при внешней инициализации канала ввода, то принятое коммуникационным портом 64-разрядное слово не записывается в память, а выдаётся через внешнюю шину и затем записывается в регистровую пару ICC0,ICA0. В этом случае он претендует на следующие ресурсы:

  - ♦ внутреннюю выходную шину OUTPUT DATA BUS;
  - ♦ внутреннюю входную шину INPUT WEIGHT BUS;
  - ♦ одну из внешних шин - LOCAL BUS, если она в данный момент принадлежит процессору NM6403, или GLOBAL BUS в противном случае.

- Запрос на ПДП от канала ввода коммуникационного порта 0 (запись в память). Данный запрос претендует на следующие ресурсы:

  - ♦ внутреннюю выходную шину OUTPUT DATA BUS;
  - ♦ одну из внешних шин - LOCAL BUS или GLOBAL BUS - в зависимости от старшего разряда адреса ПДП.

Если данный запрос сформирован при внешней инициализации канала ввода, то принятое коммуникационным портом 64-разрядное слово не записывается в память, а выдаётся через внешнюю шину и затем записывается в регистровую пару ICC1,ICA1. В этом случае он претендует на следующие ресурсы:

  - ♦ внутреннюю выходную шину OUTPUT DATA BUS;

- внутреннюю входную шину INPUT WEIGHT BUS;
- одну из внешних шин - LOCAL BUS, если она в данный момент принадлежит процессору NM6403, или GLOBAL BUS в противном случае.

- Запрос на ПДП от канала вывода коммуникационного порта 1 (чтение из памяти). Данный запрос претендует на следующие ресурсы:
  - одну любую из внутренних входных шин - INPUT INSTRUCTION BUS или INPUT WEIGHT BUS;
  - одну из внешних шин - LOCAL BUS или GLOBAL BUS - в зависимости от старшего разряда адреса ПДП.
- Запрос на ПДП от канала вывода коммуникационного порта 0 (чтение из памяти). Данный запрос претендует на следующие ресурсы:
  - одну любую из внутренних входных шин - INPUT INSTRUCTION BUS или INPUT WEIGHT BUS;
  - одну из внешних шин - LOCAL BUS или GLOBAL BUS - в зависимости от старшего разряда адреса ПДП.
- Запрос на выборку новой команды. Данный запрос претендует на следующие ресурсы:
  - одну любую из внутренних входных шин - INPUT INSTRUCTION BUS или INPUT WEIGHT BUS;
  - одну из внешних шин - LOCAL BUS или GLOBAL BUS - в зависимости от старшего разряда адреса команды.

До двух запросов от команд форматов 1.1; 1.2; 2.1; 2.2; 2.3; 4.1 и 4.2 (переход к подпрограмме); 4.3; 5.1; 5.2 на уровне IR1. Требуемые ими внутренние шины в зависимости от типа команды описаны в разделе о запуске команд в конвейер, причём на IR1 не может быть команд, претендующих на одни и те же внутренние ресурсы. Каждому из запросов также необходима одна из внешних шин - LOCAL BUS или GLOBAL BUS - в зависимости от старшего разряда вычисленного в соответствии с командой адреса. Исключение составляют команды форматов 2.1; 2.2; 2.3, когда не происходит обращение в память, а внешняя шина используется для выдачи содержимого регистра-источника или константы и записи его в регистр-приёмник в соответствии с командой. При этом требуется GLOBAL BUS, если она свободна на момент до начала арбитража запросов, или LOCAL BUS в противном случае.

Для описанных в п. 6 запросов используется следующее правило: наибольший приоритет среди них сначала имеет запрос на

обслуживание команды возврата из подпрограммы/прерывания (формат 4.3), затем - от той команды, которая ранее попала в конвейер.

Далее излагается алгоритм арбитража для вышеперечисленных запросов.

Шаг 1. Если есть запрос на ПДП от канала ввода коммуникационного порта 1, то он будет обслужен, если свободны все требуемые им ресурсы. После этого они становятся занятыми.

Переход на шаг 2.

Шаг 2. Если есть запрос на ПДП от канала ввода коммуникационного порта 0, то он будет обслужен, если свободны все требуемые им ресурсы. После этого они, становятся занятыми.

Переход на шаг 3.

Шаг 3. Если есть запрос на ПДП от канала вывода коммуникационного порта 1, то он будет обслужен, если свободна соответствующая внешняя шина, и не прошёл запрос на шаге 1 (может быть обслужен только один запрос от данного коммуникационного порта). В этом случае занимается вышеуказанная внешняя шина.

Переход на шаг 4.

Шаг 4. Если есть запрос на ПДП от канала вывода коммуникационного порта 0, то он будет обслужен, если свободна соответствующая внешняя шина, и не прошёл запрос на шаге 2 (может быть обслужен только один запрос от данного коммуникационного порта). В этом случае занимается вышеуказанная внешняя шина.

Переход на шаг 5.

Шаг 5. Если есть запрос на выборку новой команды, то он будет обслужен, если свободна соответствующая внешняя шина. В этом случае считается занятой соответствующая внешняя шина.

Переход на шаг 6.

Шаг 6. Если есть запрос от команды на IR1 (имеется в виду, что он либо единственный на IR1, либо более приоритетный) то он будет обслужен, если свободны все требуемые им ресурсы. В этом случае данные ресурсы становятся занятыми.

Переход на шаг 7.

Шаг 7. Если остался менее приоритетный запрос от другой команды на IR1, то он будет обслужен, если свободны все требуемые им ресурсы. Арбитраж на этом закончен.

Остаётся только добавить, что для запросов на шаге 3, 4 или 5 не важно, с какими именно внутренними входными шинами они будут работать, поэтому при арбитраже учитывается только, свободна ли необходимая им внешняя шина (если свободна хоть одна из двух внешних шин, то обязательно свободна хоть одна из двух внутренних входных шин). Поэтому для данных запросов после арбитража происходит доопределение, с какими именно внутренними входными шинами они будут работать, в следующем порядке:

если обслужен запрос 5 (выборка новой команды) и свободна если обслужен запрос 5 (выборка новой команды) и свободна INPUT INSTRUCTION BUS, то она им и занимается, иначе им занимается INPUT WEIGHT BUS;

если обслужен запрос 3 (ПДП от канала вывода коммуникационного порта 1) и свободна INPUT INSTRUCTION BUS, то она им и занимается, иначе им занимается INPUT WEIGHT BUS;

если обслужен запрос 4 (ПДП от канала вывода коммуникационного порта 0) и свободна INPUT INSTRUCTION BUS, то она им и занимается, иначе им занимается INPUT WEIGHT BUS.

9 Интерфейс с памятью

В данной главе, описывается организация и работа интерфейса с глобальной и с локальной памятью. Интерфейс с локальной памятью практически полностью идентичен интерфейсу с глобальной памятью, поэтому в ряде разделов данной главы описывается только интерфейс с глобальной памятью. Примеры подключения процессора NM6403 к внешним шинам различных конфигураций описаны в главе 12. Временные параметры внешних сигналов процессора NM6403 приведены в главе 13.

9.1 Общие сведения

Процессор NM6403 имеет два идентичных интерфейса с локальной и глобальной шинами. Каждый интерфейс имеет следующие особенности:

- суммарное число выводов 88, в том числе 64-разрядная шина данных и шина адреса до 19 разрядов с возможностью выдачи по ней 30-разрядного адреса в мультиплексном режиме;
- работа как с 64-разрядными словами данных, так и 32-разрядными полусловами данных (младший разряд 32-разрядного адреса используется для адресации 32-разрядного полуслова внутри 64-разрядного слова данных);
- возможность выполнения однотактных циклов “чтение из памяти” и “запись в память”;
- наличие сигналов для работы нескольких процессоров с общей памятью;
- программируемое пользователем разбиение адресного пространства на один или два банка памяти, которые могут различаться типом памяти (SRAM, Flash ROM, DRAM, EDO DRAM), размером страницы памяти и временными параметрами;
- поддержка страничного режима для каждого банка памяти независимо от её типа в банке;
- возможность формировать циклы ожидания (задаются как программно, так и с помощью внешнего сигнала готовности);
- аппаратная поддержка режима регенерации для DRAM и EDO DRAM;
- набор управляющих сигналов интерфейса позволяет работать с внешней памятью различного типа напрямую без использования внешнего контроллера.

Интерфейс с глобальной шиной аналогичен интерфейсу с локальной

шиной за исключением:

1) Они относятся к разным адресным пространствам, определяемым старшим разрядом вычисляемого 32-разрядного адреса.

2) При обозначении сигналов интерфейса с локальной шиной добавляется префикс “L”.

3) После системного сброса локальная шина принадлежит процессору NM6403, глобальная - нет.

Дальнейшая информация будет относиться к обеим интерфейсам, однако, в некотрых разделах будет описан только интерфейс с глобальной шиной.

9.2 Сигналы интерфейса с памятью

Функциональное описание сигналов интерфейса с памятью дано ниже, см. Табл. 9-1 Некотрые из этих сигналов могут менять выполняемую ими функцию в зависимости от содержимого регистра управления интерфейсом (см. раздел 9.3) и от текущего состояния глобальной (локальной) шины. На Рис. 9-1, Рис. 9-2 и Рис. 9-3 показаны варианты функционального состава внешней шины в момент работы с динамической (а) или статической памятью (б) в многопроцессорном и однопроцессорном режимах работы.

Табл. 9-1. Внешние выводы процессора NM6403, подключаемые к глобальной (локальной) шине.

Обозначение [1]	**Кол-во**	**Тип** [3]	**Функциональное назначение**
D63 – D0	64	I/O	64-разрядная шина данных
A15 – A1	15	O(Z)	15-разрядная шина адреса
$\overline{\text{RAS0}}$ [2] / $\overline{\text{CS0}}$ [2]	1	O(Z)	Строб адреса строки банка 0 DRAM / Выборка банка 0 SRAM
$\overline{\text{RAS1}}$ [2] / $\overline{\text{CS1}}$ [2]	1	O(Z)	Строб адреса строки банка 1 DRAM / Выборка банка 1 SRAM
$\overline{\text{CASL}}$ [2] / $\overline{\text{WEL}}$ [2]	1	O(Z)	Строб адреса столбца 32 младших разрядов DRAM / Разрешение записи в 32 младших разряда SRAM
$\overline{\text{CASH}}$ [2] / $\overline{\text{WEH}}$ [2]	1	O(Z)	Строб адреса столбца 32 старших разрядов DRAM / Разрешение записи в 32 старших разряда SRAM

Табл.9-1. Внешние выводы процессора NM6403, подключаемые к глобальной (локальной) шине. (Продолжение)

Обозначение [1]	Кол-во	Тип [3]	Функциональное назначение
$\overline{OE}$ [2]	1	O(Z)	Разрешение выдачи данных из RAM
$\overline{WE}$ [2] / A16	1	O(Z)	Признак цикла “запись в DRAM” / 16-й разряд адреса SRAM
$\overline{HOLDO}$ [2] / A17	1	O	Запрос шины процессором NM6403 / 17-й разряд адреса SRAM
$\overline{HOLDI}$ [2] / *A*18	1	I/O	Запрос шины внешним устройством / 18-й разряд адреса SRAM
$\overline{RDY}$ [2] / A19	1	I/O	Асинхронный сигнал готовности / 19-й разряд адреса SRAM

Примечание:

1) Имена выводов в таблице соответствуют глобальной шине, для локальной шины к имени вывода следует добавить префикс “L”.

2) Для выводов со знаком инверсии активным является низкий уровень сигнала.

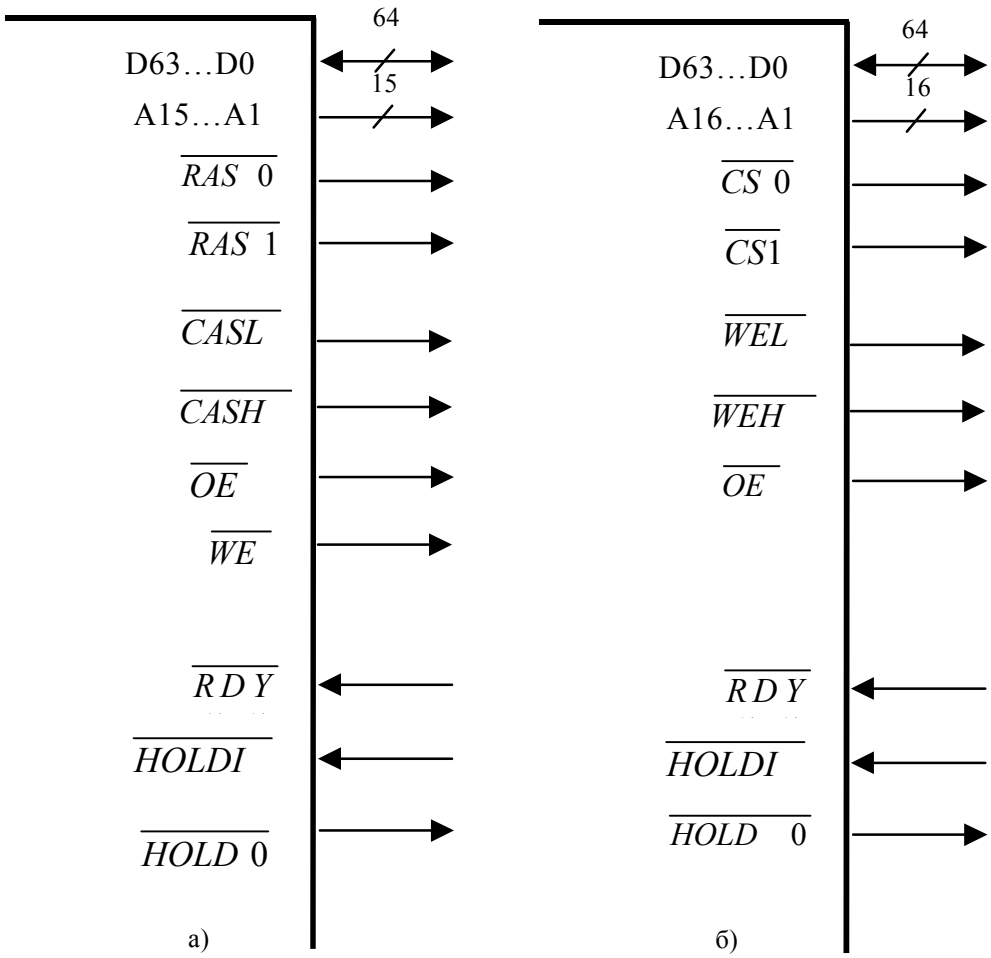
3) Использованы следующие обозначения типов выводов:

I/O - двунаправленный вывод;

O(Z) - выход с высокоимпедансным состоянием;

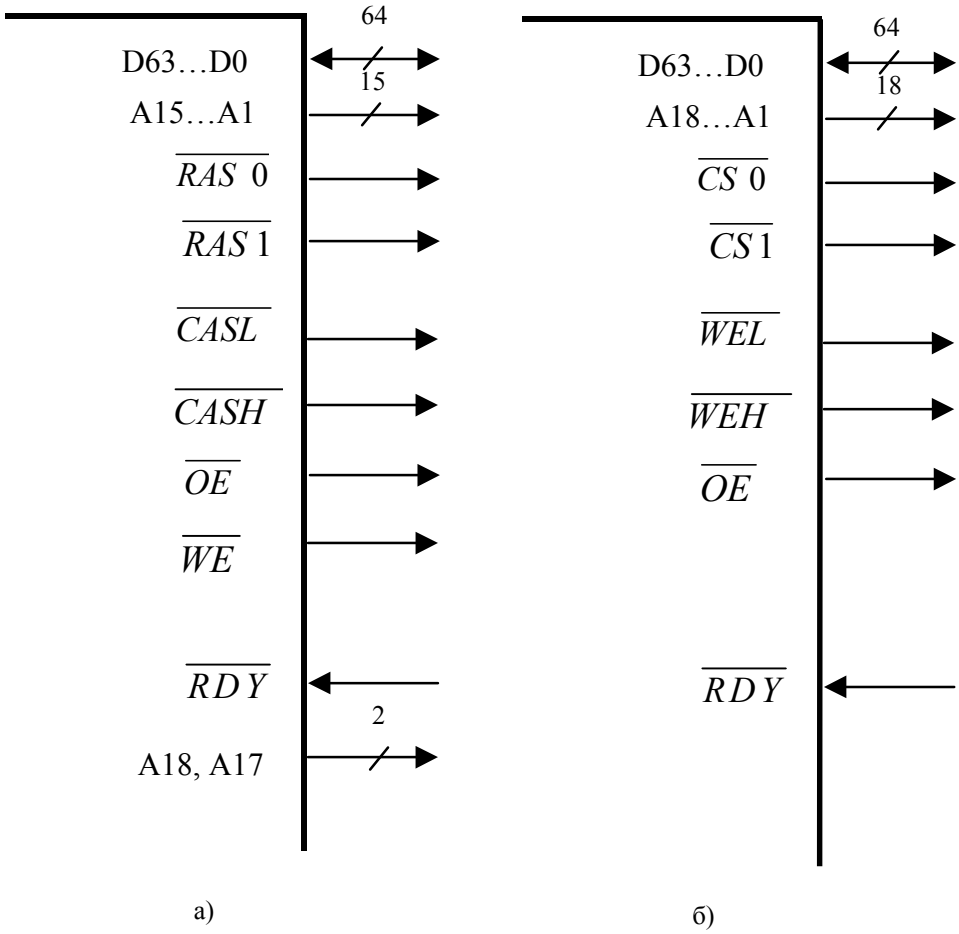
O - выход.

Рис. 9-1. Функциональный состав внешней шины при работе с DRAM или EDRAM (а) и SRAM или Flash ROM (б) в многопроцессорном режиме



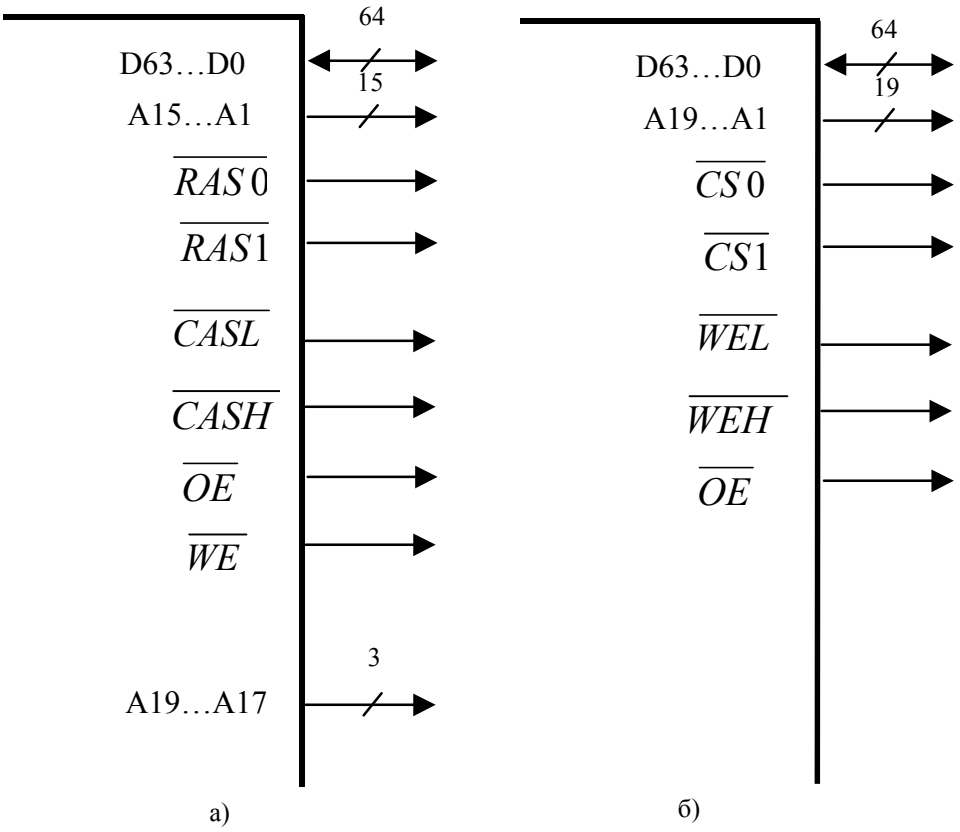
Примечание: *1) Выводы на рисунке относятся к интерфейсу с глобальной шиной. Для интерфейса с локальной шиной выводы имеют ту же конфигурацию, только к имени вывода добавляется префикс “L”.*

Рис. 9-2. Функциональный состав внешней шины при работе с DRAM или EDRAM (а) и SRAM или Flash ROM (б) в однопроцессорном режиме с использованием внешнего сигнала готовности $\overline{RDY}$



Примечание: *Выводы на рисунке относятся к интерфейсу с глобальной шиной. Для интерфейса с локальной шиной выводы имеют ту же конфигурацию, только к имени вывода добавляется префикс “L”.*

Рис. 9-3. Функциональный состав внешней шины при работе с DRAM или EDRAM (а) и SRAM или Flash ROM (б) в однопроцессорном режиме без использования внешнего сигнала готовности $\overline{RDY}$



Примечание: *Выводы на рисунке относятся к интерфейсу с глобальной шиной. Для интерфейса с локальной шиной выводы имеют ту же конфигурацию, только к имени вывода добавляется префикс "L".*

9.3 Регистры управления интерфейсом

Ниже представлен формат регистров управления глобальной и локальной шинами GMICR и LMICR, а Табл. 9-2 приведено функциональное описание отдельных полей. Данный формат GMICR и LMICR позволяет подключать к любой из внешних шин до двух банков памяти, различающихся типом, страничной организацией и динамическими параметрами. Регистры GMICR и LMICR программно доступны на запись и чтение, после системного сброса все их разряды сброшены в ноль.

Рис. 9-4. Формат регистра управления интерфейсом с глобальной (локальной) шиной GMICR (LMICR)

31 30 29 28	27 26 25 24	23 22 21 20	19 18	17 16	15 14 13 12 11	10 9 8 7 6	5 4	3	2	1 0
BOUND	PAGE1	PAGE0	TYPE1	TYPE0	TIME1	TIME0	TRAS	EXRDY1	EXRDY0	SHMEM

Примечание:

На рисунке приведены имена полей, соответствующие регистру управления интерфейсом с глобальной шиной. Для регистра управления интерфейсом с локальной шиной к имени каждого поля добавляется префикс “L”.

Табл. 9-2. Описание полей регистра управления интерфейсом с глобальной (локальной) шиной GMICR (LMICR)

Номера разрядов	Обозначение[1)]	Функциональное назначение
31 - 28	BOUND	Разбиение адресного пространства шины на банки 0 и 1 (см. Табл. 9-3 для глобальной шины и Табл. 9-4 для локальной шины).
27 - 24	PAGE1	Размер страницы памяти банка 1 (см. Табл. 9-5)[2)].
23 - 20	PAGE0	Размер страницы памяти банка 0 (см. Табл. 9-5).
19 - 18	TYPE1	Тип памяти банка 1[3)]: 00 - SRAM; 01 - DRAM 1-го типа (сигнал $\overline{\text{RAS1}}$ переводится в пассивное состояние при обращении к новой странице памяти); 10 - DRAM 2-го типа (сигнал $\overline{\text{RAS1}}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот, при циклах Idle); 11 - DRAM 3-го типа (сигнал $\overline{\text{RAS1}}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот);

Табл. 9-2. Описание полей регистра управления интерфейсом с глобальной (локальной) шиной GMICR (LMICR) (Продолжение)

Номера разрядов	Обозначение[1)]	Функциональное назначение
17 - 16	TYPE0	Тип памяти банка 0: 00 - SRAM; 01 - DRAM 1-го типа (сигнал $\overline{RAS0}$ переводится в пассивное состояние при обращении к новой странице памяти); 10 - DRAM 2-го типа (сигнал $\overline{RAS0}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот, при циклах Idle); 11 - DRAM 3-го типа (сигнал $\overline{RAS0}$ переводится в пассивное состояние при обращении к новой странице памяти, при переходе от цикла “запись” к циклу “чтение” и наоборот);
15 - 11	TIME1	Временные параметры циклов обращения к памяти банка 1[2)].
10 - 6	TIME0	Временные параметры циклов обращения к памяти банка 0.
5 - 4	TRAS	Длительность активного уровня сигнала $\overline{RAS}$ при регенерации DRAM[4)].
3	RDY1	Условие окончания цикла обращения к памяти банка 1[2)]: 0 - только по внутреннему счётчику; 1 - по внутреннему счётчику и с учётом внешнего сигнала готовности[5)].
2	RDY0	Условие окончания цикла обращения к памяти банка 0: 0 - только по внутреннему счётчику; 1 - по внутреннему счётчику и с учётом внешнего сигнала готовности[5)].
1 - 0	SHMEM	Конфигурация внешней шины: 00 - многопроцессорная конфигурация 1-го типа (банк 0 - “общий”, банк 1 - “общий”); 01 - многопроцессорная конфигурация 2-го типа (банк 0 - “свой”, банк 1 - “общий”); 10 - многопроцессорная конфигурация 3-го типа (банк 0 - “свой”, банк 1 - “чужой”); 11 - однопроцессорная конфигурация.

Примечание:

1) Приведена мнемоника, соответствующая регистру управления интерфейсом с глобальной шиной. Для регистра управления локальной шиной к имени поля добавляется префикс “L”.

2) Если банк 1 отсутствует, то значения полей PAGE1, TIME1 и RDY1 безразличны.

3) Если банк 1 отсутствует и в поле TYPE0 указан тип DRAM, то значение поля TYPE1 безразлично. Если же банк 1 отсутствует и в поле TYPE0 указан тип SRAM, то в поле TYPE1 следует так же указывать тип SRAM. В противном случае будут выполняться циклы регенерации несуществующей DRAM.

4) Если в обоих полях TYPE0 и TYPE1 указан тип SRAM, то значение поля TRAS безразлично.

5) Если в поле SHMEM указана одна из многопроцессорных конфигураций, то значения полей RDY0 и RDY1 безразличны.

9.4 Разбиение адресного пространства шины на банки памяти

Процессор NM6403 работает с 32-разрядными адресами, и в зависимости от старшего бита вычисленного адреса обращение идёт к глобальной (бит равен 1) или к локальной (бит равен 0) шине. Таким образом, всё адресное пространство делится между этими шинами поровну. Кроме того, к каждой шине можно подключать один или два банка памяти, различающихся размером памяти (от 32K до 1G слов), типом памяти (SRAM, Flash ROM, DRAM, EDO DRAM) и размером страницы памяти.

9.4.1 Границы банков памяти

Разбиение адресного пространства глобальной шины на банки памяти задается полем BOUND регистра GMICR в соответствии с Табл. 9-3, а локальной шины - полем LBOUND регистра LMICR в соответствии с Табл. 9-4.

Размер каждого банка памяти должен быть кратен размеру страницы этого банка.

Табл. 9-3. Разбиение адресного пространства глобальной шины на банки 0 и 1, задаваемое полем BOUND регистра GMICR

Поле BOUND	Количество 64-разрядн. Слов в банке 0	Адресное пространство банка 0 глобальной шины	Количество 64-разрядн. слов в банке 1	Адресное пространство банка 1 глобальной шины
0000	2^{15} = 32K	80000000 - 8000FFFF	1G - 32K	80010000 - FFFFFFFF
0001	2^{16} = 64K	80000000 - 8001FFFF	1G - 64K	80020000 - FFFFFFFF
0010	2^{17} = 128K	80000000 - 8003FFFF	1G - 128K	80040000 - FFFFFFFF

Табл. 9-3. Разбиение адресного пространства глобальной шины на банки 0 и 1, задаваемое полем BOUND регистра GMICR (Продолжение)

Поле BOUND	Количество 64-разрядн. Слов в банке 0	Адресное пространство банка 0 глобальной шины	Количество 64-разрядн. слов в банке 1	Адресное пространство банка 1 глобальной шины
0011	2^{18} = 256K	80000000 - 8007FFFF	1G - 256K	80080000 - FFFFFFFF
0100	2^{19} = 512K	80000000 - 800FFFFF	1G - 512K	80100000 - FFFFFFFF
0101	2^{20} = 1M	80000000 - 801FFFFF	1G - 1M	80200000 - FFFFFFFF
0110	2^{21} = 2M	80000000 - 803FFFFF	1G - 2M	80400000 - FFFFFFFF
0111	2^{22} = 4M	80000000 - 807FFFFF	1G - 4M	80800000 - FFFFFFFF
1000	2^{23} = 8M	80000000 - 80FFFFFF	1G - 8M	81000000 - FFFFFFFF
1001	2^{24} = 16M	80000000 - 81FFFFFF	1G - 16M	82000000 - FFFFFFFF
1010	2^{25} = 32M	80000000 - 83FFFFFF	1G - 32M	84000000 - FFFFFFFF
1011	2^{26} = 64M	80000000 - 87FFFFFF	1G - 64M	88000000 - FFFFFFFF
1100	2^{27} = 128M	80000000 - 8FFFFFFF	1G - 128M	90000000 - FFFFFFFF
1101	2^{28} = 256M	80000000 - 9FFFFFFF	1G - 256M	A0000000 - FFFFFFFF
1110	2^{29} = 512M	80000000 - BFFFFFFF	1G - 512M	C0000000 - FFFFFFFF
1111*)	2^{30} = 1G	80000000 - FFFFFFFF	0	-

Примечание: **) Если в поле BOUND задан код 1111, то банк 0 занимает всё адресное пространство глобальной шины, а банк 1 отсутствует.*

Табл. 9-4. Разбиение адресного пространства локальной шины на банки 0 и 1, задаваемое полем LBOUND регистра LMICR.

Поле LBOUND	Количество 64-разрядн. Слов в банке 0	Адресное пространство банка 0 локальной шины	Количество 64-разрядн. слов в банке 1	Адресное пространство банка 1 локальной шины
0000	2^{15} = 32K	00000000 - 0000FFFF	1G - 32K	00010000 - 7FFFFFFF
0001	2^{16} = 64K	00000000 - 0001FFFF	1G - 64K	00020000 - 7FFFFFFF
0010	2^{17} = 128K	00000000 - 0003FFFF	1G - 128K	00040000 - 7FFFFFFF
0011	2^{18} = 256K	00000000 - 0007FFFF	1G - 256K	00080000 - 7FFFFFFF
0100	2^{19} = 512K	00000000 - 000FFFFF	1G - 512K	00100000 - 7FFFFFFF
0101	2^{20} = 1M	00000000 - 001FFFFF	1G - 1M	00200000 - 7FFFFFFF
0110	2^{21} = 2M	00000000 - 003FFFFF	1G - 2M	00400000 - 7FFFFFFF
0111	2^{22} = 4M	00000000 - 007FFFFF	1G - 4M	00800000 - 7FFFFFFF
1000	2^{23} = 8M	00000000 - 00FFFFFF	1G - 8M	01000000 - 7FFFFFFF
1001	2^{24} = 16M	00000000 - 01FFFFFF	1G - 16M	02000000 - 7FFFFFFF
1010	2^{25} = 32M	00000000 - 03FFFFFF	1G - 32M	04000000 - 7FFFFFFF
1011	2^{26} = 64M	00000000 - 07FFFFFF	1G - 64M	08000000 - 7FFFFFFF
1100	2^{27} = 128M	00000000 - 0FFFFFFF	1G - 128M	10000000 - 7FFFFFFF
1101	2^{28} = 256M	00000000 - 1FFFFFFF	1G - 256M	20000000 - 7FFFFFFF
1110	2^{29} = 512M	00000000 - 3FFFFFFF	1G - 512M	40000000 - 7FFFFFFF
1111*)	2^{30} = 1G	00000000 - 7FFFFFFF	0	-

Примечание: **) Если в поле LBOUND задан код 1111, то банк 0 занимает всё адресное пространство глобальной шины, а банк 1 отсутствует.*

9.4.2 Размер страницы банка памяти

При обращении к каждому банку памяти поддерживается разбиение этого банка на страницы. Размер страницы определяется полями PAGE0 и PAGE1 регистра управления интерфейсом GMICR соответственно для банков 0 и 1 глобальной шины и полями LPAGE0 и LPAGE1 регистра управления интерфейсом LMICR для банков 0 и 1 локальной шины согласно Табл. 9-5. При работе с SRAM или Flash ROM страничный режим обеспечивает существенное увеличение объёма адресуемой памяти несмотря на ограниченное число адресных выводов. При работе с DRAM поддержка страничного режима позволяет обойтись без внешнего контроллера или значительно его упростить.

Табл. 9-5. Размеры страниц памяти, задаваемые полями PAGE0 и PAGE1 регистра GMICR и полями LPAGE0 и LPAGE1 регистра LMICR

Поле (L)PAGEx	Разряды адреса, задающие страницу памяти 2)	Разряды адреса, адресующие слова внутри страницы1)	Размер страницы памяти
0000	23 - 9	8 - 1	$2^8 = 256$
0001	24 - 10	9 - 1	$2^9 = 512$
0010	25 - 11	10 - 1	$2^{10} = 1K$
0011	26 - 12	11 - 1	$2^{11} = 2K$
0100	27 - 13	12 - 1	$2^{12} = 4K$
0101	28 - 14	13 - 1	$2^{13} = 8K$
0110	29 - 15	14 - 1	$2^{14} = 16K$
0111	30 - 16	15 - 1	$2^{15} = 32K$
1000	30 - 17 4)	16 - 1	$2^{16} = 64K$
1001	30 - 18 4)	17 - 1	$2^{17} = 128K$
1010	30 - 19 4)	18 - 1	$2^{18} = 256K$
1011	30 - 20 4)	19 - 1	$2^{19} = 512K$
11xx	Резерв	Резерв	Резерв

Примечание:

1) Число разрядов, адресующих ячейки памяти в пределах одной страницы, не должно превышать количество адресных выводов микросхемы. В противном случае будет отсутствовать страничная организация памяти.

2) Вследствие ограничения на число адресных выводов использование страниц памяти небольших размеров может привести к уменьшению используемого адресного пространства.

3) Старший разряд адреса, задающего страницу памяти, может быть в диапозоне от 23-го до 30-го.

4) Используется только при работе с SRAM с целью увеличения размера страницы банка SRAM. При этом для адресации страницы памяти на внешние адресные выводы выдаются разряды адреса с 30 по 16.

Увеличение размера страницы SRAM

С целью увеличения размера страницы SRAM некоторые внешние выводы процессора NM6403 имеют двойное функциональное назначение:

- вывод $\overline{WE}$ ($L\overline{WE}$)при работе с DRAM служит для выдачи признака выполняемой операции (чтение/запись), а при работе со SRAM или Flash ROM ((L)TYPEx = 00) - для выдачи 16-го разряда адреса;
- выводы $\overline{HOLDI}$ ($L\overline{HOLDI}$) и $\overline{HOLDO}$ ($L\overline{HOLDO}$) в отсутствии режима разделения памяти между двумя процессорами ((L)SHMEM = 11) - для выдачи 17-го и 18-го разрядов адреса;
- вывод $\overline{RDY}$ ($\overline{LRDY}$) используется для приёма внешнего сигнала готовности (при (L)EXRDYx = 1) или для выдачи 19-го разряда адреса (при (L)EXRDYx = 0 и (L)SHMEM = 11).

Таким образом, максимальный размер страницы SRAM в зависимости от содержимого полей SHMEM и EXRDYx регистра управления интерфейсом может варьироваться от 2^{16} =64К до 2^{19}=512К 64-разрядных слов.

9.5 Временные диаграммы циклов обращения к памяти

Временные диаграммы циклов обращения к памяти характеризуются функциональным составом внешних сигналов интерфейса, последовательностью их переключений и временными параметрами.

Функциональный состав и последовательность переключений внешних сигналов интерфейса в каждом выполняемом цикле обращения к памяти однозначно определяются его типом. В свою очередь тип выполняемого цикла может зависеть от следующих факторов:

- процесс, инициирующий выполнение данного цикла (текущая команда в программной последовательности, процесс ПДП при обмене данными через коммуникационные порты, прерывание по таймеру T0 для регенерации DRAM);
- текущее содержимое регистров GMICR и LMICR;
- код выполняемой команды или направление передачи данных по коммуникационным портам;
- адрес, по которому происходит текущее обращение к памяти;

- предыстория данного цикла (тип предыдущего цикла обращения к памяти и адрес последней адресованной ячейки памяти).

Рассмотреть подробно все многообразие циклов обращения к памяти в настоящем описании не представляется возможным.

Вместе с тем, каждый цикл обращения к памяти состоит из последовательно выполняемых фаз. Различаются семь типов фаз с фиксированной для каждой выполняемой операции последовательностью переключений внешних сигналов интерфейса. Все многообразие циклов обращения к памяти складывается из этих семи типов фаз. Количество фаз в различных циклах может варьироваться в диапазоне от 1 до 5. Причем в одном цикле не могут встречаться однотипные фазы. В пунктах 9.5.2 - 9.5.5 дается подробное описание всех типов фаз при выполнении различных операций и правила, по которым различные циклы обращения к памяти составляются из отдельных фаз.

Различаются два типа временных параметров циклов обращения к памяти:

1) Задержки переключения выходных сигналов и времена предустанова входных сигналов относительно фронтов синхросигнала, поступающего на вход CLK процессора NM6403. Значения данных параметров фиксированы, так как определяются физической реализацией NM6403, и приведены в разделе 13.5.

2) Длительности отдельных фаз циклов обращения к памяти. Значения этих параметров кратны периоду процессорного такта T и могут быть заданы пользователем путем загрузки определенных кодов в поля TIME0 (LTIME0), TIME1 (LTIME1) и TRAS (LTRAS) регистра GMICR (LMICR) на этапе инициализации процессора NM6403. Причем одна и та же фаза может иметь различную длительность при обращении к разным банкам памяти. Программирование длительностей отдельных фаз циклов обращения к памяти позволяет NM6403 эффективно работать с широкой номенклатурой микросхем SRAM и DRAM в синхронном режиме, что обеспечивает высокую производительность и отсутствие внешних контроллеров памяти.

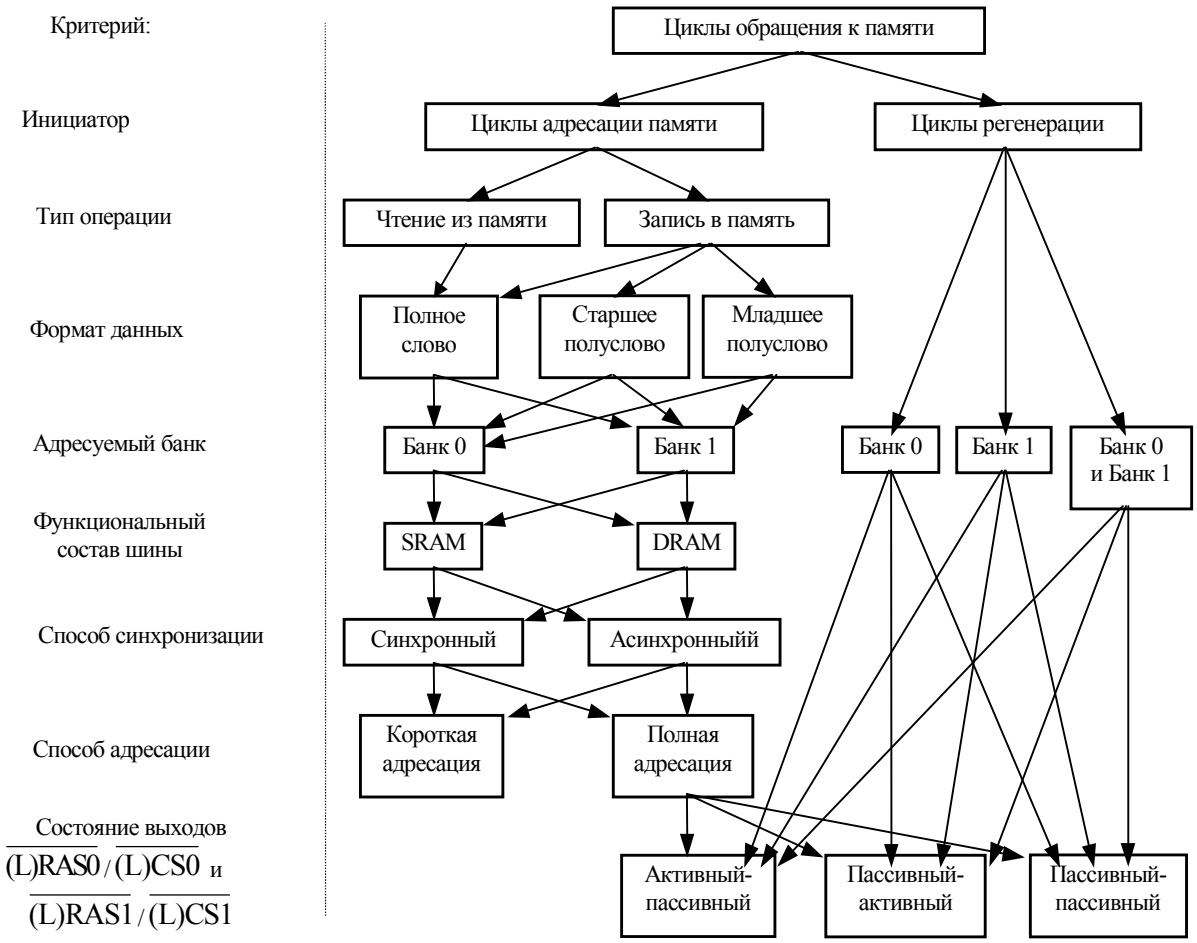
9.5.1 Типы циклов обращения к памяти

На Рис. 9-5 приведена полная классификация циклов обращения к памяти, поддерживаемых процессором NM6403. В представленной классификации каждому уровню иерархии циклов обращения к памяти соответствует определенный критерий, по которому различаются группы циклов данного уровня.

С точки зрения последовательности переключений внешних сигналов интерфейса наиболее существенно различаются циклы адресации памяти и циклы регенерации DRAM. Данные группы

циклов различаются причиной их инициализации. Циклы адресации памяти инициируются текущей командой ввода/вывода в программах пользователя и процессами ПДП при передаче информации через коммуникационные порты, а циклы регенерации DRAM - прерываниями по таймеру T0, если хотя бы в одном из полей TYPE0, TYPE1, LTYPE0 или LTYPE1 регистров GMICR и LMICR указан код DRAM.

Рис. 9-5. Классификация циклов обращения к памяти



По функциональным признакам (типу выполняемой операции и формату передаваемых данных) в циклах адресации памяти выделяются:

- циклы чтения 64-разрядного слова из памяти,
- циклы записи 64-разрядного слова в память,
- циклы записи 32-разрядного младшего полуслова в память,
- циклы записи 32-разрядного старшего полуслова в память.

Необходимо отметить, что независимо от формата данных, записываемых в регистр процессора NM6403, все команды ввода

данных инициируют цикл чтения 64-разрядного слова. Операции по выделению 32-разрядных полуслов из считанного полного слова выполняются во внутренней структуре процессора NM6403.

По выбираемому банку памяти различаются циклы обращения к банку 0, в течение которых формируется сигнал на выходе $\overline{(L)RAS0}$ / $\overline{(L)CS0}$, и циклы обращения к банку 1, в течение которых формируется сигнал на выходе $\overline{(L)RAS1}$ / $\overline{(L)CS1}$. При выполнении циклов адресации памяти банк i (i=0,1) выбирается с учетом текущего адреса и содержимого поля BOUND (LBOUND) регистра GMICR (LMICR) (см. пункт 9.4.1). При выполнении циклов регенерации банк i выбирается только в том случае, если в поле TYPEi (LTYPEi) регистра GMICR (LMICR) указан код DRAM. Как правило, для каждой внешней шины активное значение сигнала устанавливается не более, чем на одном из указанных выходов NM6403. Исключение составляют циклы одновременной регенерации строк DRAM в банках 0 и 1, во время которых сигналы формируются на обоих выходах $\overline{(L)RAS0}$ и $\overline{(L)RAS1}$.

Функциональный состав внешних сигналов интерфейса в цикле обращения к памяти определяется типом памяти (SRAM или DRAM), к которой обращается процессор NM6403 в данном цикле (см. раздел 9.2). Поэтому он зависит от номера i адресуемого банка памяти и содержимого соответствующего поля TYPEi (LTYPEi) регистра GMICR (LMICR). Функциональный состав внешних сигналов интерфейса может меняться в процессе работы NM6403, если к одной из внешних шин подключены два банка памяти различного типа. Временные диаграммы циклов адресации DRAM подробно описаны в пункте 9.5.3, циклов адресации SRAM - в пункте 9.5.4, а циклов регенерации DRAM - в пункте 9.5.5.

По способу синхронизации различаются синхронные и асинхронные циклы адресации памяти. Способ синхронизации определяет условия завершения циклов адресации памяти. Длительность каждого синхронного цикла определяется его типом и значениями полей TIME0 (LTIME0) и TIME1 (LTIME1) регистра GMICR (LMICR) (см. пункты 9.5.2 и 9.5.4) и не зависит от внешних сигналов. Асинхронный цикл отличается от однотипного синхронного цикла только дополнительным необходимым условием завершения цикла - наличием активного уровня сигнала на входе $\overline{(L)RDY}$ в конце цикла. При этом минимальная длительность асинхронного цикла не может быть меньше длительности однотипного синхронного цикла. Асинхронные циклы целесообразно использовать только для адресации медленных устройств пользователя, подключенных к внешней шине.

Асинхронные циклы могут быть реализованы только при однопроцессорной конфигурации внешней шины, так как при

любой многопроцессорной конфигурации шины вход $\overline{\text{(L)RDY}}$ используется для обеспечения межпроцессорного протокола по передаче управления общей внешней шиной (см. раздел 9.2). В случае однопроцессорной конфигурации внешней шины способ синхронизации (синхронный/ассинхронный) в цикле обращения к банку i определяется содержимым поля RDYi (LRDYi) регистра GMICR (LMICR). Таким образом, чтобы обеспечить выполнение асинхронных циклов адресации банка i, пользователю необходимо записать в поле SHMEM двоичный код 11, соответствующий однопроцессорной конфигурации, а в поле RDYi - значение 1, соответствующее асинхронным циклам. В противном случае на внешней шине будут выполняться синхронные циклы адресации банка i. Следует отметить, что для обращения к разным банкам памяти могут использоваться циклы адресации с различными способами синхронизации.

Процессор NM6403 поддерживает страничную организацию памяти. По способу адресации памяти различаются циклы с полной адресацией, во время каждого из которых на адресные выходы NM6403 последовательно выдаются адрес страницы памяти и адрес ячейки памяти в пределах этой страницы, и циклы с короткой адресацией, во время каждого из которых на адресные выходы процессора NM6403 выдается только адрес ячейки памяти в пределах страницы, адресация к которой осуществлялась в предыдущем цикле адресации памяти. Последний способ адресации поддерживается NM6403 с целью повышения пропускной способности внешних шин.

Короткая адресация в текущем цикле обращения к памяти реализуется процессором NM6403 только при выполнении всех следующих условий:

- на момент начала данного цикла адресации банка i сигнал на выходе $\overline{\text{(L)RASi}}$/$\overline{\text{(L)CSi}}$ должен находиться в активном состоянии;
- в текущем цикле должно происходить обращение к той же странице памяти, что и в предыдущем цикле адресации памяти;
- если адресуется DRAM 2-го или 3-го типа, то тип операции (чтение/запись), выполняемой в текущем цикле, должен совпадать с типом операции, выполнявшейся в предыдущем цикле адресации памяти.

Невыполнение любого из этих трех условий приводит к необходимости выполнения цикла с полной адресацией памяти.

Если очередной цикл обращения к памяти является циклом с полной адресацией или циклом регенерации DRAM, то состояние

сигналов на выходах $\overline{(L)RAS0}$/$\overline{(L)CS0}$ и $\overline{(L)RAS1}$/$\overline{(L)CS1}$ в момент начала данного цикла влияет на его длительность и на последовательность переключений сигналов на выходах $\overline{(L)RAS0}$/$\overline{(L)CS0}$ и $\overline{(L)RAS1}$/$\overline{(L)CS1}$ в первых тактах этого цикла.

В моменты времени между двумя соседними циклами возможны следующие сочетания значений сигналов на выходах $\overline{(L)RAS0}$/$\overline{(L)CS0}$ и $\overline{(L)RAS1}$/$\overline{(L)CS1}$: “активный-пассивный”, “пассивный-активный” и “пассивный-пассивный”. Причем состояние “активный-пассивный” устанавливается только по окончании любого цикла адресации банка 0, состояние “пассивный-активный” - по окончании любого цикла адресации банка 1, а состояние “пассивный-пассивный” - после системного сброса, по окончании любого цикла регенерации DRAM, по окончании цикла IDLE (пустой цикл), если он выполнялся сразу же за одним из циклов адресации DRAM 2-го типа, и в тех случаях, когда управление внешней шиной передаётся другому процессору.

Кроме циклов обращения к памяти на внешних шинах могут формироваться циклы межрегистровых пересылок и циклы IDLE. Циклы межрегистровых пересылок формируются процессором NM6403 в специальном тестовом режиме, если на вход NM6403 MVOE подаётся напряжение высокого уровня. Циклы IDLE формируются процессором NM6403 с целью заполнения временных интервалов между циклами обращения к памяти. Как правило, циклы IDLE представляют собой однотактные циклы, в течение которых на внешней шине не происходит никаких переключений. Исключение составляет цикл IDLE, выполняющийся сразу же за одним из циклов адресации DRAM 2-го типа. Данный цикл состоит только из одной фазы RP (фаза перевода сигналов $\overline{(L)RAS0}$ и $\overline{(L)RAS1}$ в пассивное состояние), которая присутствует как в циклах адресации памяти (см. пункты 9.5.2 - 9.5.4), так и в циклах регенерации DRAM (см. пункт 9.5.5).

9.5.2 Программируемые фазы циклов адресации памяти

Циклы адресации памяти включают в себя от 1 до 5 фаз. Типы этих фаз и их программируемые длительности приведены в Табл. 9-6.

Табл. 9-6. Программируемые динамические параметры интерфейса в циклах адресации памяти

Фаза цикла обращения к памяти		Длительность фазы[1)]		
Обозначение	Наименование	Обозначение	DRAM	SRAM
RP	Фаза сброса сигналов $\overline{(L)RAS0}/\overline{(L)CS0}$ и $\overline{(L)RAS1}/\overline{(L)CS1}$	T_{RP}	(1 - 2) T	1 T[3)]
PAGE	Фаза адресации страницы памяти	T_{PAGE}	(1 - 2) T	1 T[3)]
CP[2)]	Пассивная фаза адресации ячейки памяти	T_{CP}	(0 - 1) T	(0 - 3) T
CA	Активная фаза адресации ячейки памяти	T_{CA}	(1 - 2) T	(1 - 4) T
BE	Фаза перехода выходов памяти в высокоимпедансное состояние.	T_{BE}	(1 - 2) T	(1 - 2) T

Примечание:

1) T - длительность процессорного такта.

2) Фаза CP является опцией. Если ее длительность T_{CP} задана равной 0, то данная фаза не будет встречаться ни в одном из циклов адресации памяти.

3) Длительности фаз PAGE и RP в циклах адресации SRAM не программируются пользователем и всегда равны одному процессорному такту.

Значения длительностей фаз, указанных в Табл. 9-6, могут задаваться пользователем в поле TIME0 (LTIME0) для банка 0 и в поле TIME1 (LTIME1) для банка 1 регистра GMICR (LMICR). Форматы полей TIME0 (LTIME0) и TIME1 (LTIME1) зависят от типа памяти в банке 0 и банке 1 соответственно. Формат данных полей для DRAM приведен на Рис. 9-6, а для SRAM – на Рис. 9-7. Каждое из полей TIME0 (LTIME0) и TIME1 (LTIME1) разбито на несколько подполей, в каждом из которых указывается значение длительности одной из программируемых фаз. Обозначения этих подполей совпадают с обозначениями длительностей фаз, приведенными в Табл. 9-6.

Рис. 9-6. Форматы полей TIME0 (LTIME0) и TIME1 (LTIME1) регистра GMICR (LMICR) для DRAM

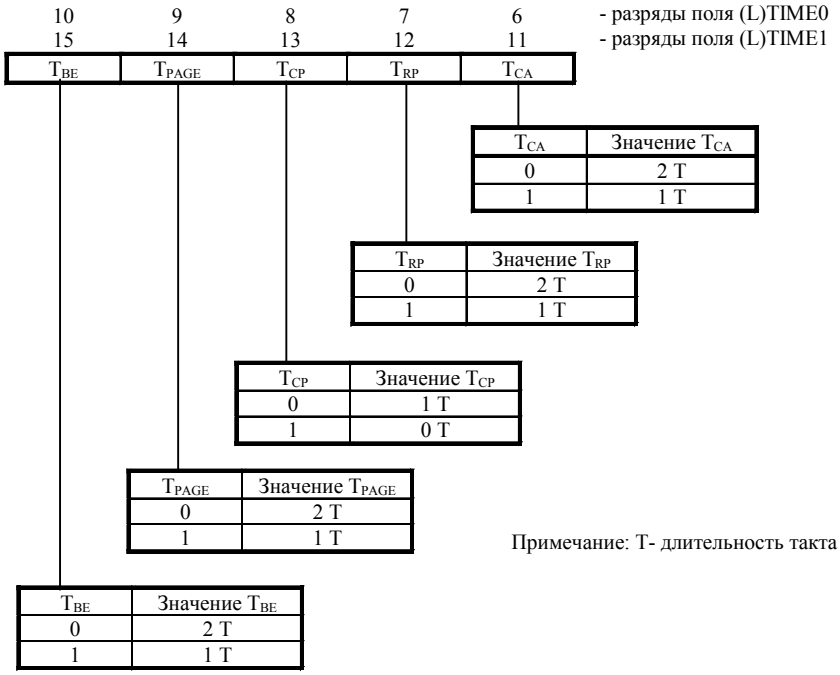
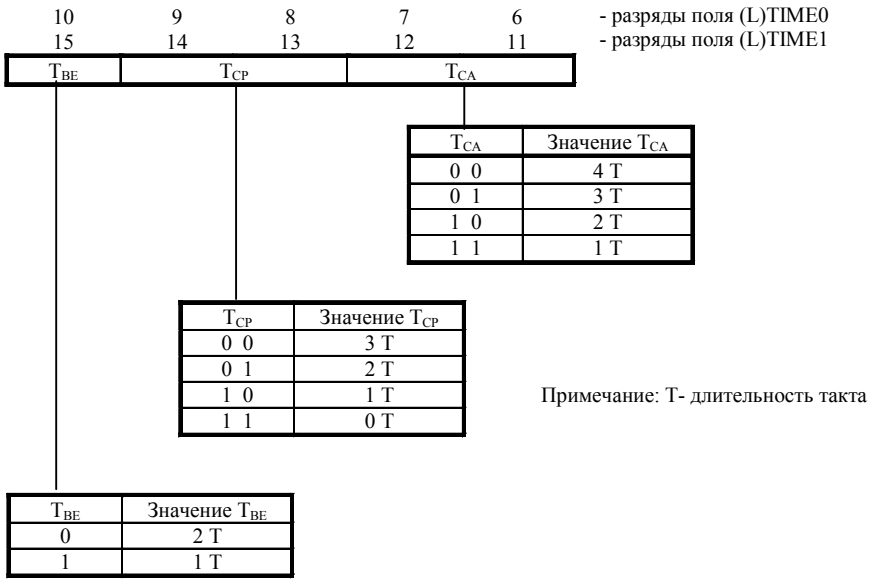
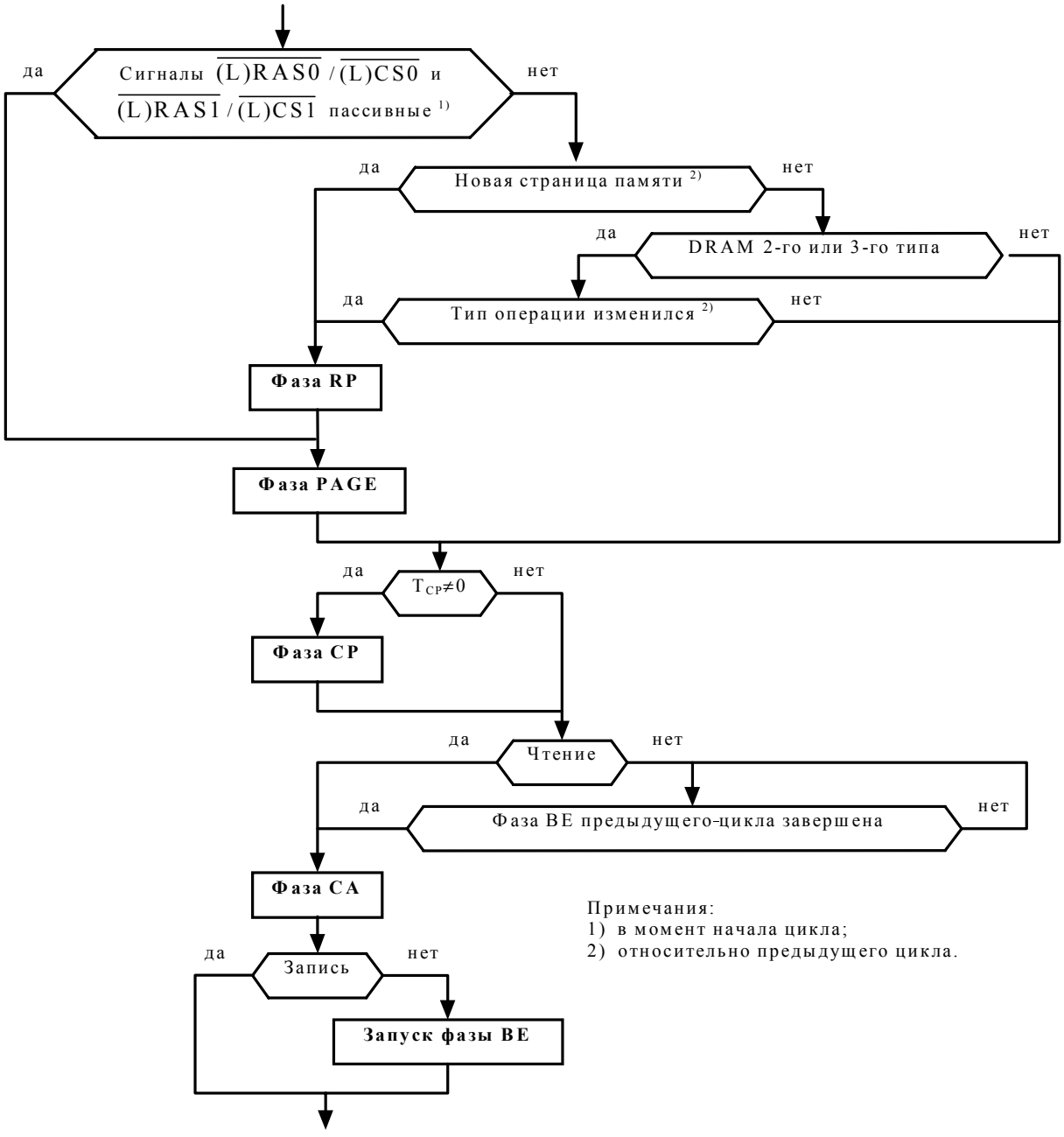


Рис. 9-7. Форматы полей TIME0 (LTIME0) и TIME1 (LTIME1) регистра GMICR (LMICR) для SRAM



Количество и последовательность фаз в каждом цикле адресации памяти определяются типом данного цикла и значением поля T_{CP}. На Рис. 9-8 приведены правила, по которым формируются последовательности фаз в различных циклах адресации памяти.

Рис. 9-8. Правила формирования последовательности фаз в циклах обращения к памяти



Ниже дается подробное описание каждой фазы, встречающейся в различных циклах адресации DRAM:

- RP - фаза сброса сигналов (L)RAS0/(L)CS0 и (L)RAS1/(L)CS1. Данная фаза присутствует в каждом цикле с полной адресацией памяти, если в момент начала цикла хотя бы один из сигналов (L)RAS0/(L)CS0 или (L)RAS1/(L)CS1 находится в активном состоянии. При этом фаза RP всегда является первой фазой цикла. Вначале фазы RP по заднему фронту сигнала CLK сигналы RAS0 и RAS1 переводятся в пассивное состояние. Это единственное действие, выполняемое NM6403 на данной фазе. Длительность фазы RP зависит от состояния выходов (L)RAS0/(L)CS0 и (L)RAS1/(L)CS1 в момент начала цикла: (T_{RP}) составляет 1 - 2 процессорных такта в зависимости от содержимого поля T_{RP} регистра GMICR.

- PAGE - фаза адресации строки (страницы) DRAM. Фаза PAGE присутствует в каждом цикле с полной адресацией памяти. Если предыдущий цикл обращения к памяти завершался фазой RP, фаза PAGE является первой фазой цикла. Перед началом данной фазы управляющие сигналы RAS0, RAS1, CASL, CASH и OE находятся в пассивном состоянии, сигнал WE сохраняет свое значение, оставшееся от предыдущего цикла обращения к памяти, шина данных D63...D0 находится в высокоимпедансном состоянии. В первом такте фазы PAGE по заднему фронту сигнала CLK процессор NM6403 выставляет на шину адреса A15...A1 адрес строки DRAM, который является адресом страницы памяти. Одновременно с этим в циклах записи формируется сигнал WE, а в циклах чтения снимается сигнал WE. В том же такте по переднему фронту сигнала CLK NM6403 формирует сигнал RASi, где i = 0 или 1 в зависимости от того, к какому банку памяти обращается процессор NM6403. Задний фронт сигнала RASi сообщает DRAM или внешнему контроллеру о том, что на шине адреса установлен адрес строки DRAM, а на выходе WE - признак цикла записи. Длительность фазы PAGE (T_{PAGE}) составляет 1 - 2 процессорных такта в зависимости от содержимого поля T_{PAGE} регистра GMICR.

- CP - пассивная фаза адресации столбца DRAM. Данная фаза является опцией - ее наличие или отсутствие в цикле адресации памяти определяется содержимым поля T_{CP} регистра GMICR. Если содержимое данного поля говорит о наличии фазы CP, то данная фаза присутствует в каждом цикле адресации памяти независимо от его типа. В циклах с полной адресацией памяти фаза CP всегда следует за фазой PAGE, а в циклах с короткой

адресацией памяти - является первой фазой цикла. Вначале фазы CP по заднему фронту сигнала CLK процессор NM6403 выставляет на шину адреса A15...A1 адрес столбца DRAM. Одновременно с этим в циклах чтения формируется сигнал $\overline{OE}$, а в циклах записи в DRAM 2-го типа на шину данных D63...D0 выдаются данные, предназначенные для записи в память. Причем в циклах записи младшего/старшего полуслова в DRAM 2-го типа записываемое полуслово выдается одновременно на 32 старших (D63...D32) и 32 младших (D31...D0) разряда шины данных. Кроме того в циклах записи с короткой адресацией памяти вначале фазы CP по заднему фронту сигнала CLK процессор NM6403 формирует сигнал $\overline{WE}$. В течение всей фазы CP сигнал $\overline{RASi}$ находится в активном состоянии, а сигналы $\overline{CASL}$ и $\overline{CASH}$ - в пассивном состоянии. В циклах адресации DRAM 1-го или 3-го типа выходы D63...D0 находятся в высокоимпедансном состоянии в течение всей фазы CP.

- CA - активная фаза адресации столбца DRAM. Это единственная фаза, которая присутствует в каждом цикле адресации памяти независимо от его типа и содержимого регистра GMICR. Если наличие фазы CP прописано в регистре GMICR, то фаза CA всегда следует за фазой CP, в противном случае фаза CA следует за фазой PAGE в циклах с полной адресацией памяти или является первой фазой цикла с короткой адресацией памяти. В циклах записи в DRAM 1-го или 3-го типа вначале фазы CA по заднему фронту сигнала CLK NM6403 выставляет на шину данных D63...D0 данные, предназначенные для записи в память. Причем в циклах записи младшего/старшего полуслова в DRAM 1-го или 3-го типа записываемое полуслово выдается одновременно на 32 старших (D63...D32) и 32 младших (D31...D0) разряда шины данных. В первом такте фазы CA по переднему фронту сигнала CLK процессор NM6403 формирует сигналы $\overline{CASL}$ и $\overline{CASH}$. Причем в циклах чтения и циклах записи слова формируются оба этих сигнала, в циклах записи младшего полуслова - только сигнал $\overline{CASL}$, а в циклах записи старшего полуслова - только сигнал $\overline{CASH}$. Если фаза CP отсутствовала, то все операции, выполняемые обычно вначале фазы CP, выполняются в первом такте фазы CA. В конце фазы CA по заднему фронту сигнала CLK процессор NM6403 снимает сигналы $\overline{CASL}$, $\overline{CASH}$ и адрес столбца DRAM. Одновременно с этим в циклах чтения информация с шины данных D63...D0 фиксируется в регистре NM6403 и, если далее не следует новый цикл чтения с короткой адресацией памяти, снимается сигнал $\overline{OE}$, а в циклах записи выходы D63...D0 переводятся в высокоимпедансное состояние, если далее не следует новый цикл записи с короткой адресацией DRAM 2-го типа или цикл записи с короткой адресацией DRAM 1-го или 3-го типа без фазы CP.

Длительность фазы CA (T_{CA}) составляет 1 - 2 процессорных такта в зависимости от содержимого поля T_{CA} регистра GMICR. Фактически фаза CA является последней фазой любого цикла адресации памяти. По окончании данной фазы

- BE - фаза перехода выходов памяти в высокоимпедансное состояние. Данная фаза следует за фазой CA в каждом цикле чтения. Единственное назначение фазы BE - это задержать выдачу процессором NM6403 данных на шину D63...D0 до полного перехода выходов памяти в высокоимпедансное состояние. Данная цель достигается тем, что начало фазы CA очередного цикла записи задерживается до полного окончания фазы BE предшествующего ему цикла чтения. Фаза BE носит вспомогательный характер и выполняется на фоне других фаз, таких как RP, PAGE и CP, следующего цикла. Длительность фазы BE (T_{BE}) составляет 1 - 2 процессорных такта в зависимости от содержимого поля T_{BE} регистра GMICR.

Длительность одного синхронного цикла адресации DRAM может составлять от одного до семи процессорных тактов.

Далее дается подробное описание каждой фазы, встречающейся в различных циклах “чтение” и “запись” при работе с SRAM:

- PAGE - фаза адресации страницы SRAM. Фаза PAGE присутствует в цикле “чтение/запись”, если предыдущий цикл завершался фазой RP. При этом фаза PAGE является первой фазой цикла. Перед началом данной фазы управляющие сигналы $\overline{CS0}$, $\overline{CS1}$, $\overline{WEL}$, $\overline{WEH}$ и $\overline{OE}$ находятся в пассивном состоянии, а шина данных D63...D0 - в высокоимпедансном состоянии. Вначале фазы PAGE по заднему фронту сигнала CLK процессор NM6403 выставляет на шину адреса A16...A1 адрес страницы SRAM. В том же такте по переднему фронту сигнала CLK NM6403 формирует сигнал $\overline{CSi}$, где i = 0 или 1 в зависимости от того, к какому банку памяти обращается процессор NM6403. Задний фронт сигнала $\overline{RASi}$ сообщает SRAM или внешнему контроллеру о том, что на шине адреса установлен адрес страницы памяти. Длительность фазы PAGE (T_{PAGE}) всегда равна одному процессорному такту.

- CP - пассивная фаза адресации ячейки SRAM. Данная фаза является опцией - ее наличие или отсутствие в цикле “чтение/запись” определяется содержимым поля T_{CP} регистра GMICR. Если содержимое данного поля говорит о наличии фазы CP, то данная фаза присутствует в каждом цикле “чтение/запись” независимо от его предистории. Фаза CP следует за фазой PAGE, если последняя присутствует в цикле, или является первой фазой цикла “чтение/запись”. Вначале фазы CP по заднему фронту сигнала CLK NM6403 выставляет на шину адреса A16...A1 адрес

страницы памяти. Одновременно с этим в циклах “чтение” формируется сигнал $\overline{OE}$, а на шину данных D63...D0 выдаются данные, предназначенные для записи в память. Причем в циклах “запись младшего/старшего полуслова” записываемое полуслово выдается одновременно на 32 старших (D63...D32) и 32 младших (D31...D0) разряда шины данных.. В течение всей фазы CP сигнал $\overline{CSi}$ находится в активном состоянии, а сигналы $\overline{WEL}$ и $\overline{WEH}$ - в пассивном состоянии. Длительность фазы CP (T_{CP}) составляет 0 - 3 процессорных такта в зависимости от содержимого поля T_{CP} регистра GMICR.

- CA - активная фаза адресации ячейки SRAM. Это единственная фаза, которая присутствует в каждом цикле “чтение/запись”. Если наличие фазы CP прописано в регистре GMICR, то фаза CA всегда следует за фазой CP, в противном случае фаза CA следует за фазой PAGE или является первой фазой цикла. В первом такте фазы CA циклов “запись” по переднему фронту сигнала CLK процессор NM6403 формирует сигналы $\overline{WEL}$ и $\overline{WEH}$. Причем в циклах “запись слова” формируются оба этих сигнала, в циклах “запись младшего полуслова” - только сигнал $\overline{WEL}$, а в циклах “запись старшего полуслова” - только сигнал $\overline{WEH}$. Если фаза CP отсутствовала, то все операции, выполняемые обычно вначале фазы CP, выполняются в первом такте фазы CA. В конце фазы CA по заднему фронту сигнала CLK NM6403 снимает адрес ячейки SRAM. Одновременно с этим в циклах “чтение” информация с шины данных D63...D0 фиксируется в регистре процессора NM6403 и, если далее не следует новый цикл “чтение из той же страницы DRAM”, снимается сигнал $\overline{OE}$, а в циклах “запись” снимаются сигналы $\overline{WEL}$ и $\overline{WEH}$ и, если далее не следует новый цикл “запись в ту же страницу”, выходы D63...D0 переводятся в высокоимпедансное состояние. Длительность фазы CA (T_{CA}) составляет 1 - 4 процессорных такта в зависимости от содержимого поля T_{CA} регистра GMICR.

- RP - фаза пассивного значения сигнала $\overline{CSx}$. Данная фаза присутствует в каждом цикле обращения к памяти, если за данным циклом следует цикл обращения к новой странице памяти или цикл регенерации DRAM другого банка шины (см. пункт 9.5.3). Фаза RP следует только за фазой CA и является последней фазой цикла обращения к памяти. Вначале фазы RP по заднему фронту сигнала CLK сигналы $\overline{CS0}$ и $\overline{CS1}$ переводятся в пассивное состояние. Это единственное действие, выполняемое NM6403 на данной фазе. Длительность фазы RP (T_{RP}) всегда равна одному процессорному такту. По окончании фазы RP все управляющие сигналы находятся в пассивном состоянии, выходы D63...D0 находятся в высокоимпедансном состоянии, а на адресную шину A15...A1 выдается неопределенное состояние.

- BE - фаза перехода выходов памяти в высокоимпедансное состояние. Данная фаза следует за фазой CA цикла “чтение”, если за этим циклом следует цикл “запись”. Единственное назначение фазы BE - это задержать выдачу процессором NM6403 данных на шину D63...D0 до полного перехода выходов памяти в высокоимпедансное состояние. Данная цель достигается тем, что начало фазы CA очередного цикла “запись” задерживается до полного окончания фазы BE предшествующего ему цикла “чтение”. Фаза BE носит вспомогательный характер и выполняется на фоне других фаз, таких как RP, PAGE и CP. Длительность фазы BE (T_{BE}) составляет 1 - 2 процессорных такта в зависимости от содержимого поля T_{BE} регистра GMICR.

9.5.3 Временные диаграммы циклов адресации DRAM

Для иллюстрации последовательности переключений внешних сигналов интерфейса в каждой фазе циклов адресации DRAM на Рис. 9-9 и Рис. 9-10. приведены временные диаграммы синхронных циклов с полной адресацией банка 0 и T_{CP}=T. Цикл с полной адресацией DRAM и T_{CP}=T выбран по той причине, что он включает в себя максимально возможное количество фаз. Обращение к банку 0 в данных циклах выбрано в качестве примера. Оно отличается от обращения к банку 1 только поведением сигналов $\overline{RAS0}$ и $\overline{RAS1}$. Приведенные временные диаграммы справедливы для всех трех типов DRAM, с которыми может работать процессор NM6403.

На Рис. 9-9 и Рис. 9-10 указаны все фазы, составляющие циклы адресации DRAM. Причем реальная длительность каждой фазы совпадает с величиной, заданной в соответствующем разряде поля TIME0 регистра GMICR. Исключение составляет фаза RP, длительность которой зависит не только от содержимого поля T_{RP}, но и от состояния выходов $\overline{RAS0}$ и $\overline{RAS1}$ в момент начала цикла.

9.5.4 Временные диаграммы циклов адресации SRAM

Для иллюстрации последовательности переключений внешних сигналов интерфейса в каждой фазе циклов обращения к SRAM на Рис. 9-11 и Рис. 9-12- приведены временные диаграммы циклов обращения к новой странице банка 0. Циклы обращения к новой странице памяти выбраны по той причине, что каждый из них включает в себя максимально возможное количество фаз. Обращение к банку 0 в данных циклах выбрано в качестве примера. Оно отличается от обращения к банку 1 только поведением сигналов $\overline{CS0}$ и $\overline{CS1}$.

Рис. 9-9. Временные диаграммы синхронного цикла чтения из банка 0 с полной адресацией DRAM.

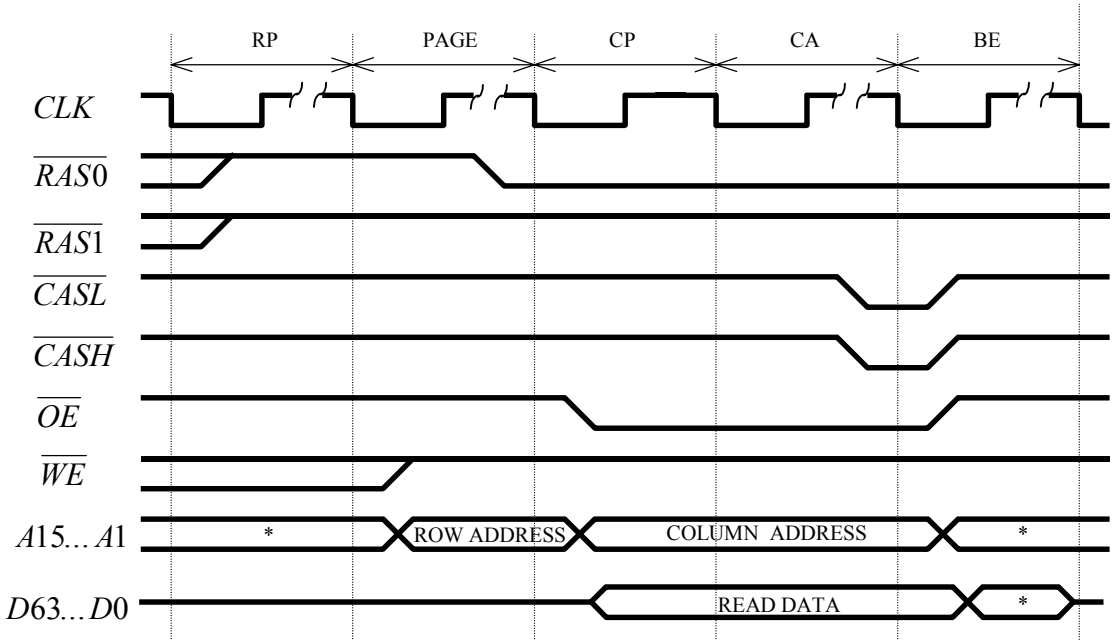


Рис. 9-10. В*ременные диаграммы синхронного цикла записи в банк 0 с полной адресацией DRAM.*

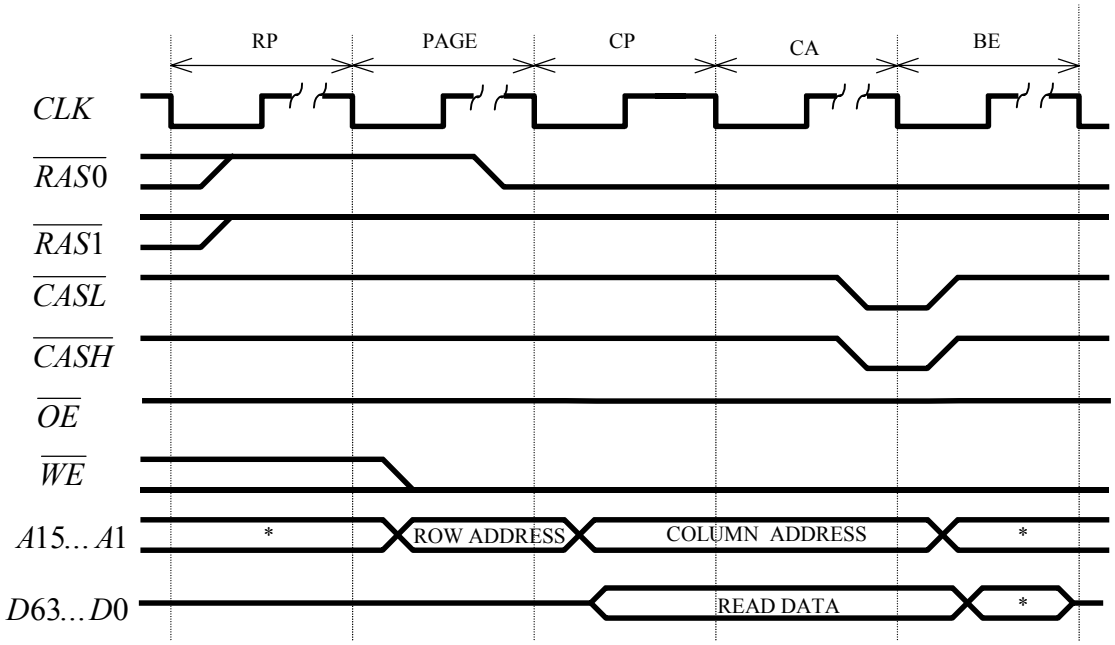


Рис. 9-11. Временные диаграммы синхронного цикла чтения из банка 0 с полной адресацией SRAM.

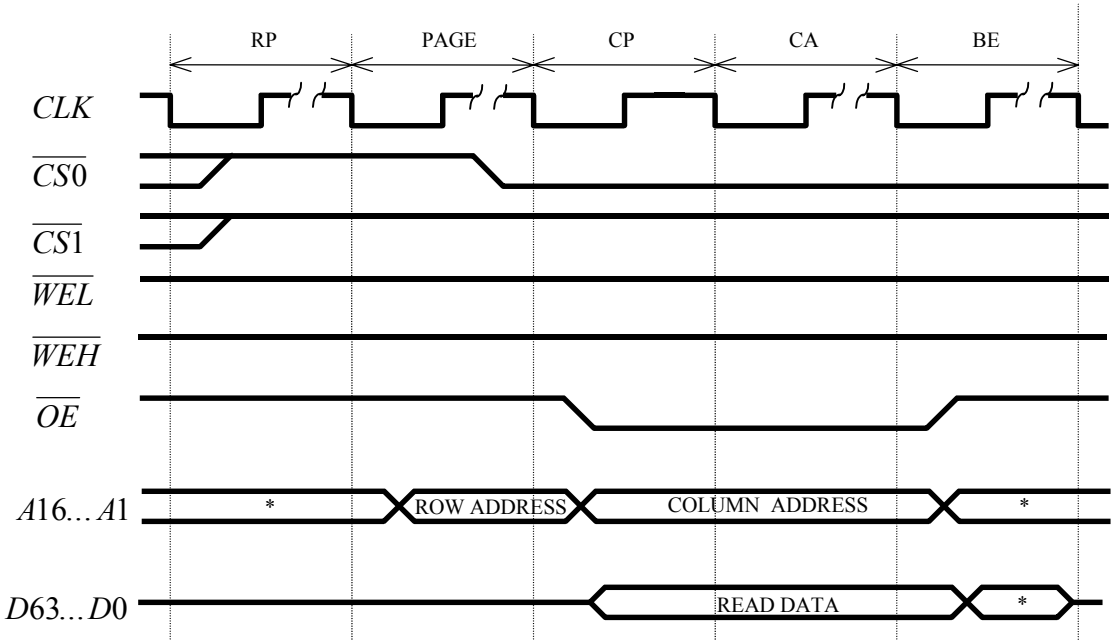
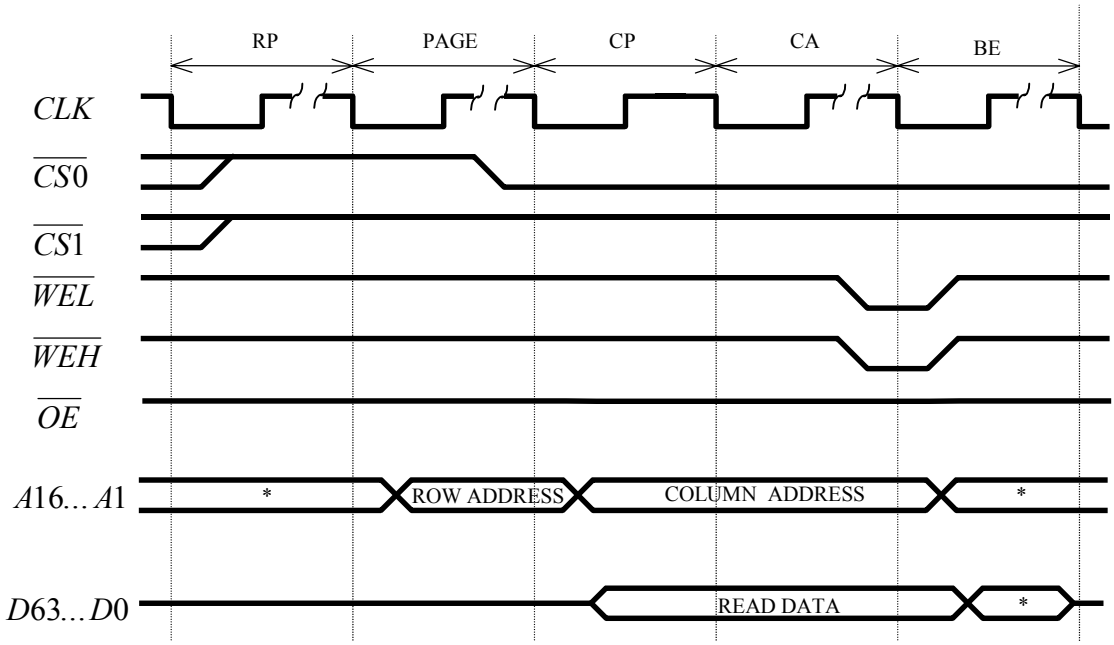


Рис. 9-12. Временные диаграммы синхронного цикла записи в банк 0 с полной адресацией SRAM.



9.5.5 Регенерация DRAM

Циклы регенерации динамической памяти включают в себя от 1 до 2 фаз. Одна из фаз при работе с DRAM - RP - была уже описана (см. Табл. 9-6). Вторая - RAS - является особой, так как встречается только при регенерации. Её длительность может программироваться от 1 до 4 процессорных тактов (см. Рис. 9-13.).

Для иллюстрации последовательности переключений внешних сигналов интерфейса в каждой фазе циклов регенерации динамической памяти приведены на Рис. 9-14 ,Рис. 9-15 и Рис. 9-16 которые отражают все возможные ситуации для данного случая.

Рис. 9-13. Формат поля (L)TRAS регистров GMICR и LMICR

5 4 - разряды поля (L) TRAS

TRAS

Примечание: T-длительность такта

T_{RAS}	Значение T_{RAS}
0 0	4 T
0 1	3 T
1 0	2 T
1 1	1 T

Рис. 9-14. Временные диаграммы регенерации двух банков DRAM.

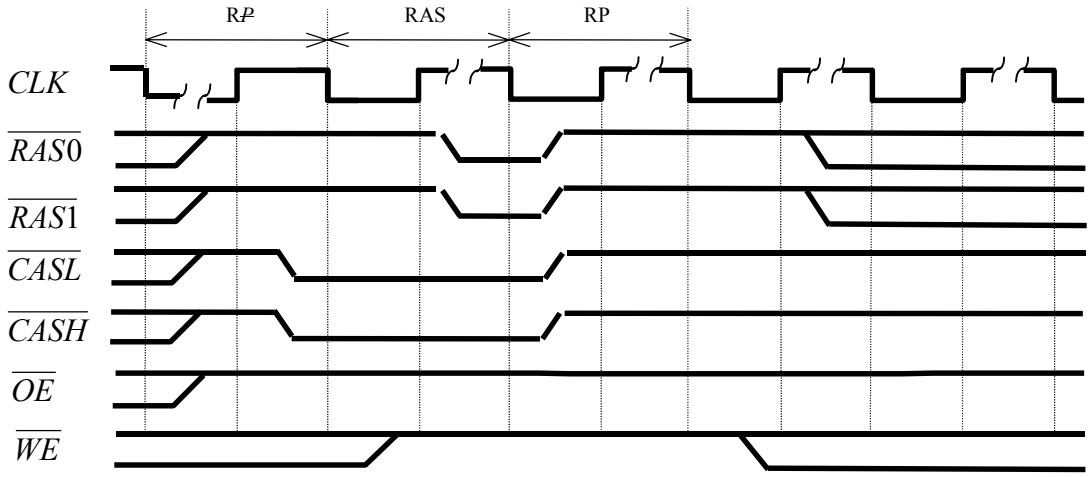


Рис. 9-15. Временные диаграммы регенерации банка 0 DRAM.

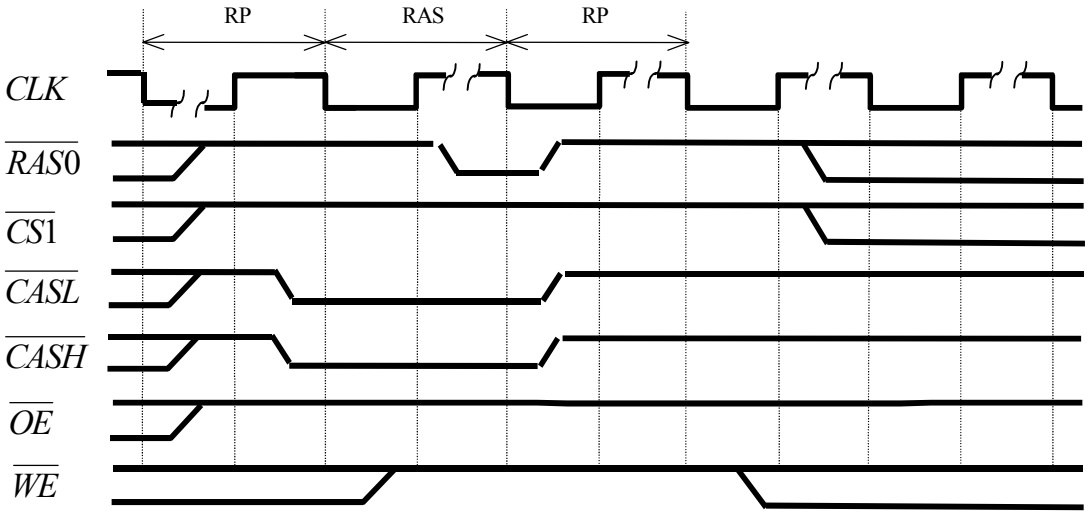
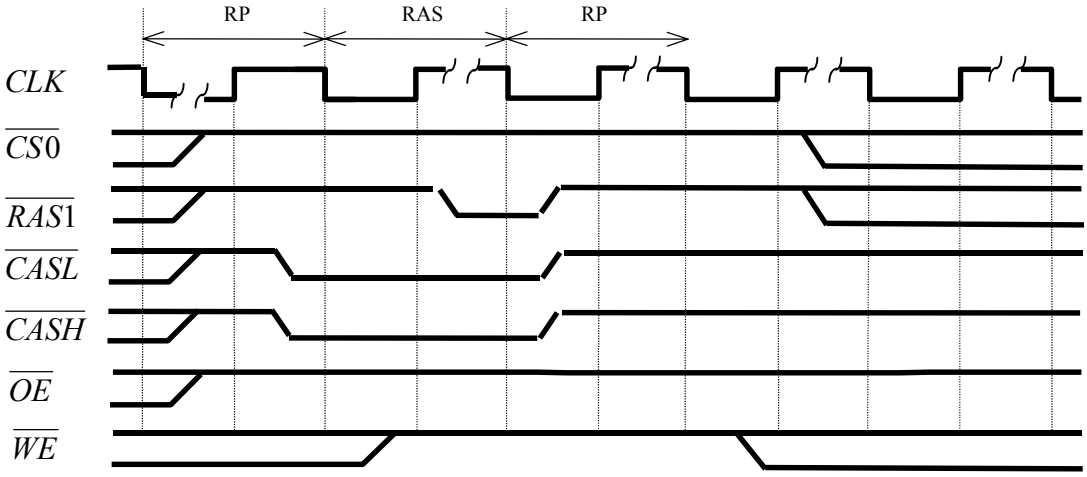


Рис. 9-16. Временные диаграммы регенерации банка 1 DRAM.



9.6 Конфигурации внешних шин, поддерживаемые интерфейсом

Процессор NM6403 поддерживает как однопроцессорный, так и многопроцессорный режим работы по любой из двух внешних шин. Если к общей памяти подключены два процессора, то арбитраж для доступа к ней осуществляется между ними без использования внешнего контроллера. При этом нужно помнить, что объединяться могут только разноимённые шины процессоров - локальная одного с глобальной другого, так как после системного сброса обладать

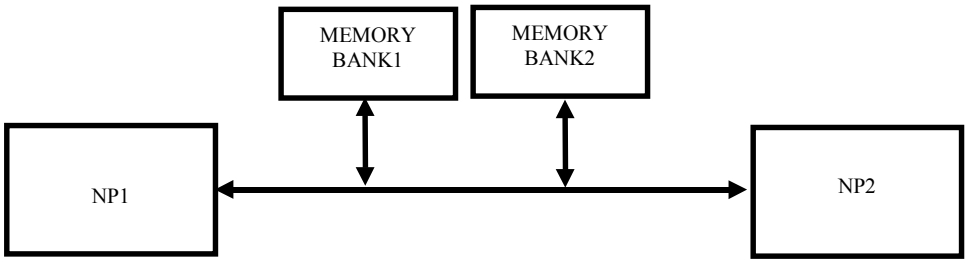
правом доступа к общей памяти может только один процессор.

Возможны три типа конфигурации внешней шины для многопроцессорного режима:

- многопроцессорная конфигурация 1-го типа (банк 0 - “общий”, банк 1 - “общий”);
- многопроцессорная конфигурация 2-го типа (банк 0 - “свой”, банк 1 - “общий”);
- многопроцессорная конфигурация 3-го типа (банк 0 - “свой”, банк 1 - “чужой”).

Пример конфигурации 1-го типа можно видеть на Рис. 9-17. Данная конфигурация характеризуется тем, что доступ в память (MEMORY BANK1, MEMORY BANK2) может осуществляться только одним процессором - NP1 или NP2 - в данный момент времени. Арбитраж для возможности работы с памятью происходит с помощью сигналов $\overline{HOLDO}$, $\overline{HOLDI}$ и $\overline{RDY}$ каждого процессора, причём выход $\overline{HOLDO}$ одного соединяется со входом $\overline{HOLDI}$ другого, сигналы $\overline{RDY}$ обоих процессоров объединены друг с другом, а также соединены с pull-up резистором. Временные диаграммы для вышеуказанных сигналов приведены на рис. 13-16 и13-19.

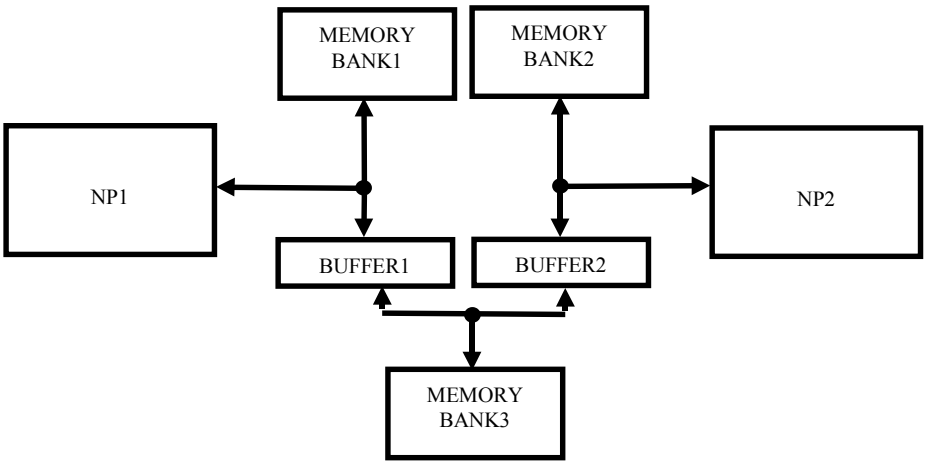
Рис. 9-17. Многопроцессорная конфигурация 1-го типа (банк 0 - “общий”, банк 1 - “общий”).



Пример конфигурации 2-го типа приведён на Рис. 9-18.

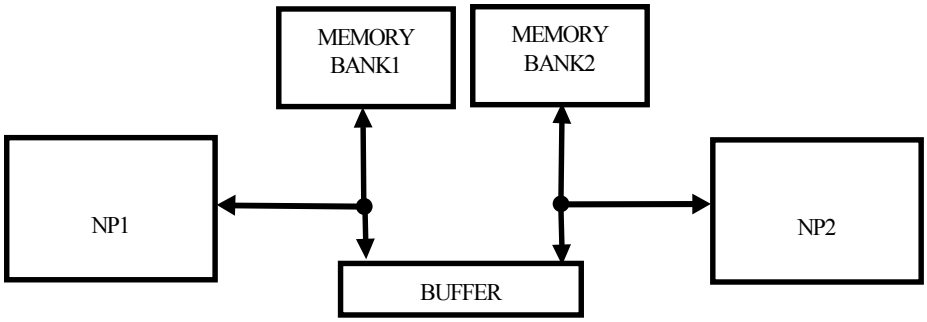
Она отличается от первой тем, что каждый процессор имеет свой банк памяти, которому другой не имеет доступа: NP1 принадлежит MEMORY BANK1, а NP2 - MEMORY BANK2. Банк MEMORY BANK3 является общим для NP1 и NP2, которые осуществляют к нему доступ поочередно через буфера BUFFER1 и BUFFER2. Арбитраж на право работать с банком MEMORY BANK3 осуществляется так же, как для конфигурации 1-го типа. Временные диаграммы для арбитража в данном режиме можно видеть на рис. 13-17 и 13-20.

Рис. 9-18. Многопроцессорная конфигурация 2-го типа (банк 0 - “свой”, банк 1 - “общий”).



Пример конфигурации 3-го типа показан на Рис. 9-19. Она отличается тем, что каждый процессор имеет свой банк памяти: NP1 принадлежит MEMORY BANK1, а NP2 - MEMORY BANK2, но существует возможность каждому процессору обратиться к чужому банку через буфер BUFFER. Временные диаграммы для арбитража в данном режиме можно видеть на рис. 13-18 и 13-21.

Рис. 9-19. Многопроцессорная конфигурация 3-го типа (банк 0 - “свой”, банк 1 - “чужой”).



10 Коммуникационные порты ввода/вывода

Данная глава содержит описание организации и работы коммуникационных портов ввода/вывода процессора NM6403, аппаратно совместимых с коммуникационными портами ЦПС TMS320C4x. Примеры построения параллельных вычислительных систем путём соединения нескольких процессоров с помощью данных портов описаны в главе 12. Временные параметры внешних сигналов коммуникационных портов процессора NM6403 приведены в главе 13.

10.1 Общие сведения

При распределении одной задачи на несколько кристаллов требуется возможность оперативного обмена информацией между этими кристаллами без заметного снижения производительности. Для быстрого обмена между процессорами предлагаются следующие возможности:

- захват шины данных и адреса извне для требуемого доступа в память (после снятия захвата процессор NM6403 способен продолжать свою работу);
- передача информации по двум высокоскоростным байтовым двунаправленным портам на фоне выполнения процессором NM6403 своих программ.

Основные характеристики каждого из портов процессора NM6403:

- передача в обе стороны с производительностью 20 Мбайт/сек (при рабочей тактовой частоте 40 Мгц);
- непосредственная коммутация процессоров с помощью 8 линий данных и 4 линий управления;
- автоматический арбитраж и асинхронный обмен для гарантии правильной работы передатчика и приемника;
- синхронизация работы процессора NM6403 и двух коммуникационных портов посредством внутренних прерываний и внутренних сигналов запроса на ПДП;
- полная аппаратная совместимость с коммуникационными портами ЦПС TMS320C4x;
- возможность работы с входными уровнями от 3 до 5 вольт;
- поддержка таких видов мультипроцессорных архитектур, как кольца и двунаправленные конвейеры.

10.2 Структура коммуникационных портов

Процессор NM6403 содержит два идентичных высокоскоростных коммуникационных порта: 0-й и 1-й, каждый из которых обеспечивает двунаправленный интерфейс для связи с внешним устройством. Рис. 10-1 показывает внутреннюю структуру одного коммуникационного порта. Каждый порт x (x = 0,1) содержит:

- CPIx - устройство управления интерфейсом порта, которое осуществляет арбитраж при передаче данных между процессором NM6403 и внешним устройством через шину данных коммуникационного порта. CPI более детально описан в разделе 10.4;
- MUX - мультиплексор для формирования адреса запроса на ПДП от коммуникационного порта;
- OCCx - счётчик канала вывода, который определяет число выводимых через коммуникационный порт 64-разрядных слов;
- OCAx - регистр адреса канала вывода, который задаёт адрес памяти, откуда будут в режиме ПДП считываться данные при выводе через порт;
- OCDR - регистр данных канала вывода;
- ICCx - счётчик канала ввода, который определяет число вводимых через коммуникационный порт 64-разрядных слов;
- ICAx - регистр адреса канала ввода, который задаёт адрес памяти, куда будут в режиме ПДП записываться данные при вводе через порт;
- ICDRx - регистр данных канала ввода.

Каждый коммуникационный порт содержит следующие двунаправленные линии управления и данных:

- $\overline{CREQ_x}$ - запрос на выдачу по шине данных коммуникационного порта;
- $\overline{CACK_x}$ - подтверждение на захват шины данных коммуникационного порта при получении сигнала $\overline{CREQ_x}$ от другого процессора;
- $\overline{CSTRB_x}$ - строб порта коммуникации. Передающий процессор устанавливает этот сигнал для индикации, что он выдал очередные данные на шину данных коммуникационного порта;
- $\overline{CRDY_x}$ - сигнал готовности коммуникационного порта. Принимающий процессор устанавливает этот сигнал для

индикации, что он принял данные через шину данных коммуникационного порта;

- $C_xD(7-0)$ - шина данных коммуникационного порта. По этой шине пересылаются данные в обе стороны между двумя процессорами или между процессором NM6403 и другим устройством.

Кроме того, имеется выход $\overline{CDIRx}$, который указывает, в каком режиме работает в данный момент коммуникационный порт - ввода (на выходе высокий уровень) или вывода (на выходе низкий уровень).

Рис. 10-1. Структурная схема коммуникационного порта ввода/вывода

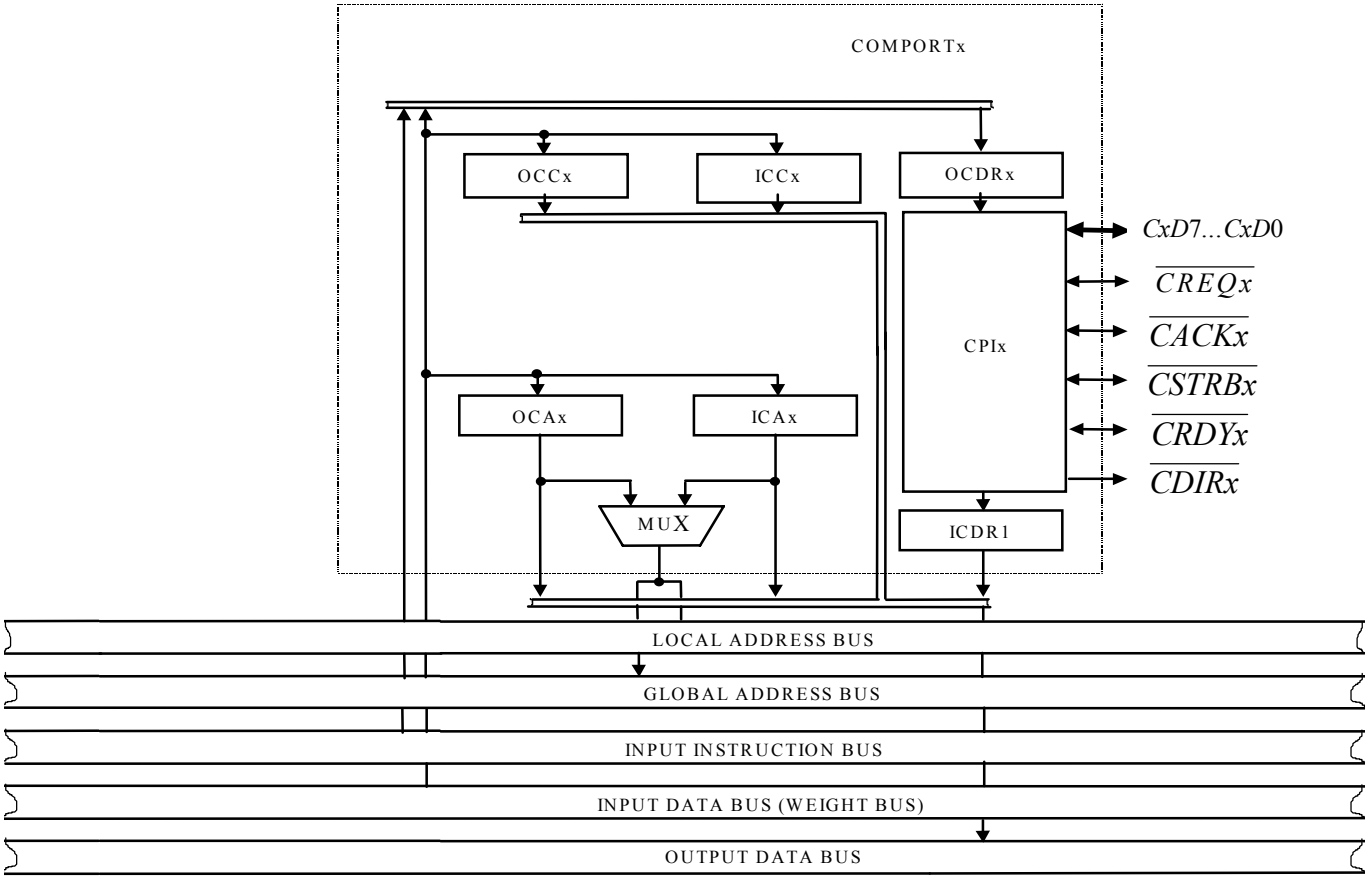
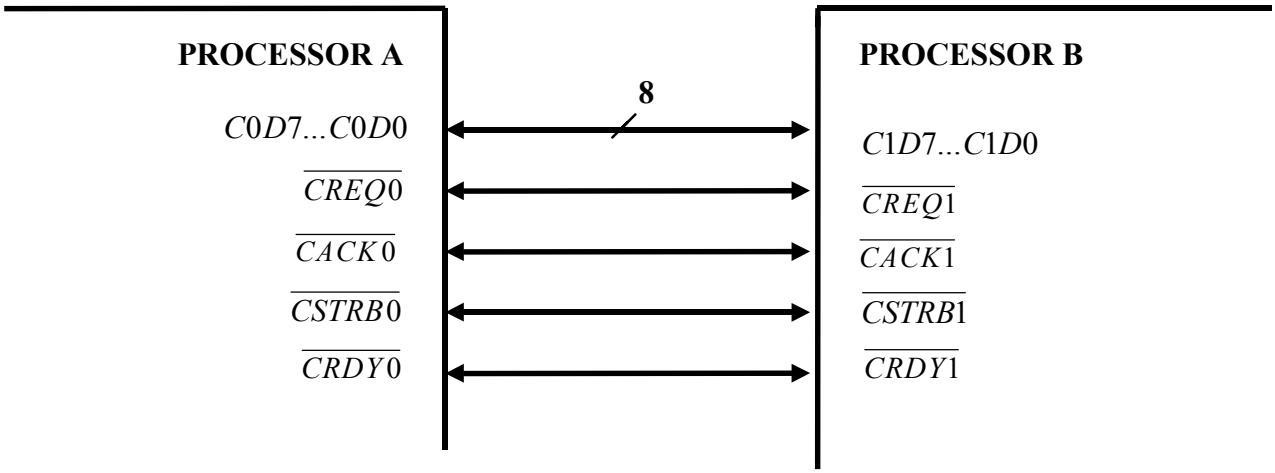


Рис. 10-2 показывает два процессора NM6403, соединенных посредством коммуникационных портов. Шина данных коммуникационного порта $C_xD(7-0)$ с помощью сигналов управления передает данные в любом направлении между процессорами A и B. CPI двух процессоров совместно генерируют последовательности сигналов управления для гарантированной передачи данных с возможно большей скоростью. Для предотвращения конфликтов по шине данных CPI производят арбитраж на захват шины, позволяя только одному процессору

выдавать на шину в данный момент времени. CPI другого процессора может потребовать захват шины после того, как сосед передаст свои данные.

Рис. 10-2. Пример соединения двух процессоров NM6403 через интерфейс портов ввода/вывода



Сигналы $\overline{CREQ_x}$ и $\overline{CACK_x}$ управляют арбитражем между двумя процессорами NM6403 таким образом:

1) CPI, который не захватил шину данных $C_xD(7-0)$ и если ему есть чего передать, устанавливает активный сигнал $\overline{CREQ_x}$ для запроса на захват этой шины;
2) CPI - владелец шины, если у него самого не инициализирован вывод, делает активным сигнал $\overline{CACK_x}$ для подтверждения запроса и отдает право на владение шиной запросившему CPI. Этим способом данные сигналы передают право на выдачу от одного CPI другому, и CPI, получивший данное право, захватывает шину.

При обмене данными происходит следующее:

1) передающий коммуникационный порт в режиме ПДП считывает из памяти 64-разрядное двойное слово в буферный регистр данных порта OCDRx;
2) вышеуказанный порт выставляет байт данных на шину $C_xD(7-0)$ и устанавливает активный сигнал $\overline{CSTRB_x}$ для сигнализации принимающему коммуникационному порту, что на шине находятся данные;
3) после приема в буферный регистр очередного байта принимающий порт делает активным сигнал $\overline{CRDY_x}$ для подтверждения, что данные получены;
4) когда будут получены все 8 байт, принимающий порт может переписать их из буферного регистра данных порта ICDRx в соответствующую ячейку памяти в режиме ПДП.

10.3 Управление коммуникационным портом

Управление коммуникационными портами осуществляется путем установки/сброса соответствующих битов регистра слова состояния процессора PSWR, а именно, для коммуникационного порта 0 это разряды 22-20, а для коммуникационного порта 1 - разряды 25-23. Ниже описывается действие каждого из этих битов.

Для коммуникационного порта 0:

22-й р. (CP0I) -бит внешней инициализации канала ввода порта 0 (может быть изменён только командой формата 2.3) :

0 - инициализация канала ввода порта 0 (сколько 64- разрядных двойных слов принять и по какому адресу записать в память) осуществляется программно;

1 - инициализация канала ввода порта 0 осуществляется по первому принятому двойному слову;.

21-й р. (ICH0) -бит останова канала ввода порта 0:

0 - разрешение принимать данные по этому порту;

1 - останов канала ввода порта.

20-й р. (OCH0) - бит останова канала вывода порта 0:

0 - разрешение выдавать данные по этому порту;

1 - останов канала вывода.

Для коммуникационного порта 1:

25-й р. (CP1I) -бит внешней инициализации канала ввода порта 1 (может быть изменён только командой формата 2.3) :

0 - инициализация канала ввода порта 1 (сколько 64- разрядных двойных слов принять и по какому адресу записать в память) осуществляется программно;

1 - инициализация канала ввода порта 1 осуществляется по первому принятому двойному слову.

24-й р. (ICH1) -бит останова канала ввода порта 1:

0 - разрешение принимать данные по этому порту;

1 - останов канала ввода порта.

23-й р. (OCH1) - бит останова канала вывода порта 1:

0 - разрешение выдавать данные по этому порту;

1 - останов канала вывода.

10.4 Принцип работы устройства управления интерфейсом порта (CPI)

CPI отвечает за арбитраж между двумя устройствами и определяет, какое из них в данный момент владеет шиной данных коммуникационного порта. Данный арбитраж позволяет каждому из устройств, соединенных посредством портов, поочередно захватывать шину данных. CPI может находиться в одном из четырех состояний, представленных в Табл. 10-1.

Табл. 10-1. Состояния CPI

Состояние CPI	Краткая характеристика	Статус CPI
00	1. CPI захватил шину. 2. Канал вывода не используется.	CPI захватил шину, но канал вывода не используется.
01	1. CPI не захватил шину. 2. Нет запроса на захват шины от CPI.	CPI не владеет шиной и не запрашивает её.
10	1. CPI захватил шину. 2. Канал вывода в работе.	CPI захватил шину и использует ее для выдачи данных.
11	1. CPI не захватил шину. 2. CPI выдает запрос на захват шины.	CPI в настоящий момент не владеет шиной, но выставляет запрос на её захват.

Рис. 10-2 описывает диаграмму состояний CPI. Для того, чтобы выдать свои данные на шину коммуникационного порта, CPI должен произвести арбитраж между собственным запросом на вывод, формируемым внутри кристалла, и внешним запросом, получаемым по линии $\overline{CREQ}$.

Этот арбитраж заканчивается захватом одним из CPI разных портов шины данных. При системном сбросе 0 канал имеет право на выдачу, а 1 канал - нет. Арбитраж осуществляется по линиям $\overline{CREQ}$ и $\overline{CACK}$.

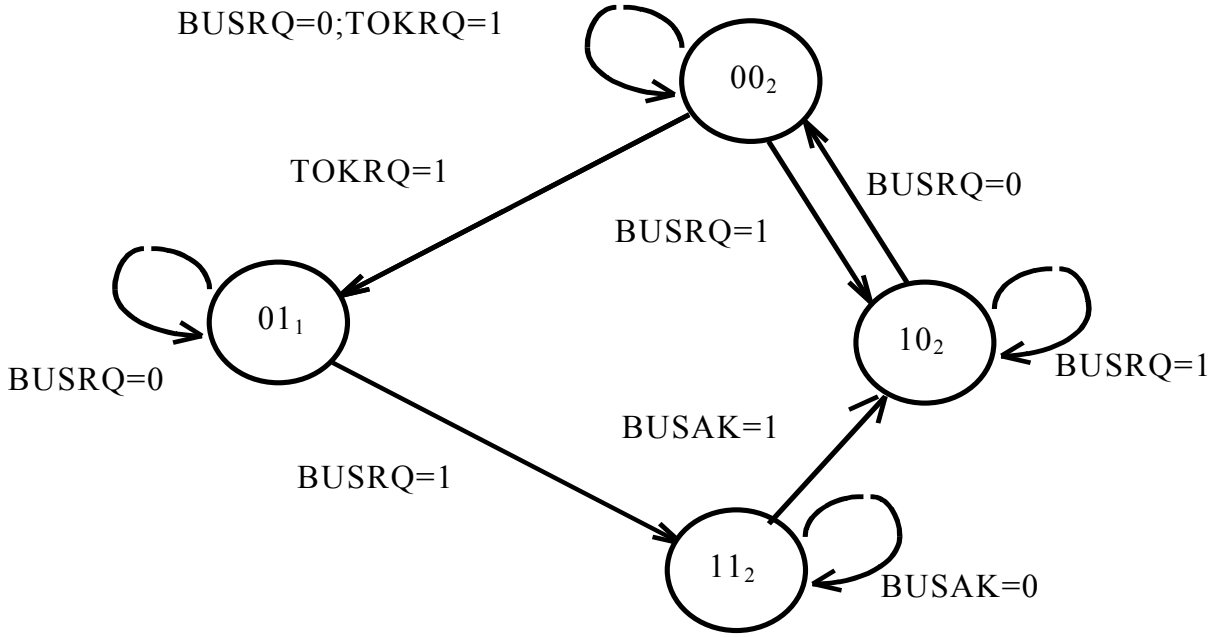
Для понимания работы схемы арбитража рассмотрим передачу данных от процессора A к B (см. рис.). В начале CPI A находится в состоянии 00_2 и B в состоянии 01_2. Если CPI процессора A получает сигнал от своего буфера вывода начать выдачу (BUSRQ=1), он разрешает сразу же передать столько двойных слов по 8 байт по шине данных коммуникационного порта, сколько требуется, и переходит в состояние 10_2. По окончании передачи BUSRQ=0 и CPI процессора A возвращается в состояние 00_2.

Если CPI процессора B получает запрос со своего буфера вывода

начать передачу, он выставляет сигнал $\overline{CREQ}$. CPI процессора A получает этот запрос (TOKRQ=1) и затем активизирует сигнал $\overline{CACK}$, позволяя B захватить шину коммуникационного порта. CPI процессора B затем генерирует внутренний сигнал BUSAK=1 и захватывает шину. CPI процессора A переходит в состояние 01_2, а CPI процессора B - в состояние 10_2.

Если входной буфер не успел освободиться и начинается передача следующего слова, приемник просто не установит сигнал $\overline{CRDY}$, пока ситуация не исправится.

Рис. 10-3. Диаграмма состояний CPI



10.5 *Останов каналов ввода/вывода.*

Останов каналов ввода/вывода осуществляется с помощью битов ICHx и OCHx (x = 0,1) в PSWR. Целью останова канала ввода является как можно быстрее остановить ввод, но без потери информации. При этом не выдается сигнал готовности при приеме первого байта очередного 32-разрядного слова. В этот момент передача данных будет заморожена, пока не будет снят останов или не случится системный сброс. Если позднее останов снимется, передача будет продолжена без потери данных.

Если канал ввода будет остановлен в момент, когда от соседа приходит запрос на захват шины, то этот запрос подтверждается до останова.

Останов канала вывода аналогичен останову канала ввода. Предположим, что процессор A имеет бит OCHx=1. Тогда останов канала вывода будет выполнен в зависимости от текущего

состояния:

- если коммуникационный порт не захватил шину, вывод блокируется и запрос на захват не осуществляется;
- если коммуникационный порт А захватил шину и в настоящее время передает слово, тогда после окончания выдачи данного слова передача далее не осуществляется;
- если коммуникационный порт А захватил шину и ICH=0, тогда он при запросе от порта В передает ему право на выдачу на шину данных;
- если коммуникационный порт А захватил шину и ICH=1, OCH=0, тогда при запросе от порта В захват им шины не разрешается;
- при снятии останова канал может продолжать выдачу, если он сохранил на это право. Если нет, он должен запросить шину данных обратно и закончить обмен.

В Табл. 10-2. представлено краткое описание вышеизложенного.

Табл. 10-2. Краткое описание действий при остановах ввода/вывода

Содержимое ICH, OCH	**При захвате портом шины**	**Порт не захватил шину**
ICH=1 OCH=0	а) не отдает монополию на шину б) выдача данных может осуществляться	а) не выдает сигнала готовности при приеме первого байта слова б) если первый байт уже принят, будет осуществлен прием остатка слова, затем останов ввода
ICH=0 OCH=1	а) не будут выдаваться данные б) если останов после передачи первого байта, заканчивается выдача первого слова, затем производится останов в) будет при запросе разрешать захват шины	а) будет принимать данные б) не будет требовать захвата шины порта
ICH=1 OCH=1	а) не будет разрешать захват другим портом шины б) не будет выдавать данные в) если останов после передачи первого байта, заканчивается выдача первого слова, затем производится останов	а) не выдает сигнала готовности при приеме первого байта слова б) если первый байт уже принят, будет осуществлен прием остатка слова, затем останов ввода в) не будет требовать захвата шины

10.6 Взаимодействие коммуникационных портов с центральным процессором

Коммуникационные порты поддерживают несколько режимов синхронизации своей работы с работой центрального процессора:

- сигналы ПДП на запись/чтение из памяти в/из регистра данных коммуникационного порта;
- прерывания по завершению обмена.

Прерывания могут быть следующих типов:

- INT_I1 (прерывание по завершению ввода по порту 1);
- INT_O1 (прерывание по завершению вывода из порта 1);
- INT_I0 (прерывание по завершению ввода по порту 0);
- INT_O0 (прерывание по завершению вывода из порта 0).

После выдачи сигнала прерывания от соответствующего порта по завершению ввода (вывода) дальнейший ввод (вывод) по данному порту осуществляться не будет, пока не будет обработано это прерывание и не произойдет следующая инициализация канала ввода (вывода).

10.7 Временные диаграммы арбитража шины данных коммуникационных портов

Арбитраж шины данных для коммуникационных портов осуществляется посредством обмена сигналами $\overline{CREQ}$ и $\overline{CACK}$. Временная диаграмма данного обмена приведена на Рис. 10-4. Для имен сигналов на этом рисунке используются суффиксы, говорящие, какому процессору они принадлежат. Например, $\overline{CREQ_B}$ обозначает сигнал $\overline{CREQ}$ процессора B. Табл. 10-4 содержит описание отраженных на рисунке событий, а временные параметры можно увидеть в разделе 13.2.

Следует отметить, что при арбитраже на линиях управления $\overline{CREQ}$, $\overline{CSTRB}$ и $\overline{CRDY}$ в некотором промежутке времени один из портов ещё не перешёл в режим приёма, а другой уже начинает выдавать. Такая ситуация диктует требование - переключение из приёма в передачу и обратно необходимо проводить через выдачу высокого уровня. Например, на Рис. 10-4 видно, что сигнал процессора A - $\overline{CSTRB_A}$ - после выдачи высокого уровня переходит в приём только после прихода положительного фронта $\overline{CREQ_A}$, при этом процессор

В по той же линии (сигнал $\overline{CSTRB_B}$) держит высокий уровень период времени, равный приблизительно половине длительности его тактового импульса. Поскольку оба процессора выдают высокий уровень, то нет сквозного тока от одного устройства к другому. По этой же причине есть ещё одно ограничение - тактовые частоты работы процессоров, соединённых через коммуникационные порты, не должны различаться более чем в два раза. В противном случае, более быстрое устройство может начать выдачу низкого уровня, в то время как более медленное ещё держит высокий уровень. Это может вызвать конфликт по линиям управления и привести к повреждению буферов ввода/вывода.

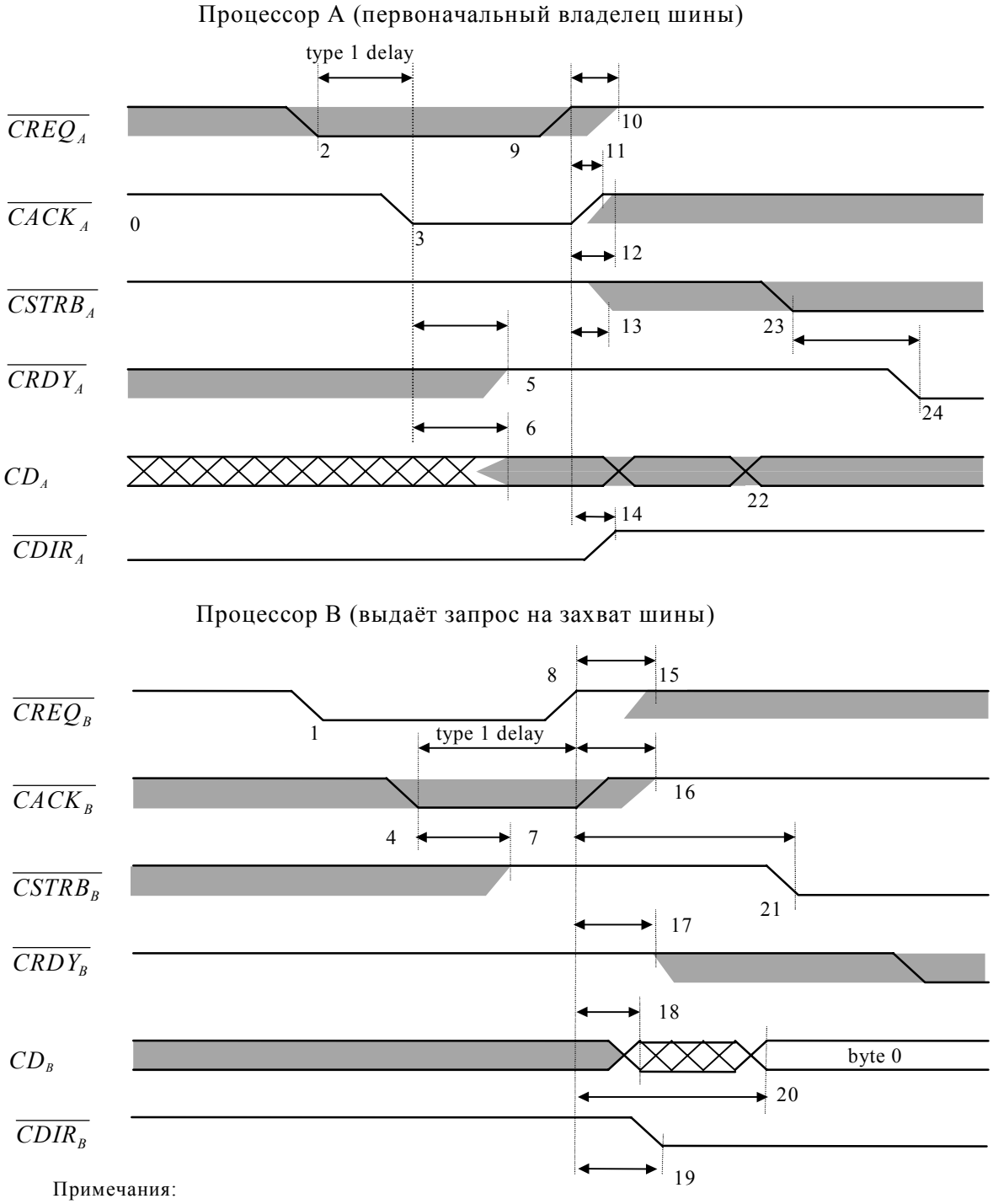
Не существует ограничений по времени при арбитраже для сигналов $\overline{CREQ}$ и $\overline{CACK}$, что позволяет процессору работать с гораздо более медленными устройствами, если не допускается конфликтов при переключении по линиям управления и данных.

Для индикации режима работы у каждого порта есть выход $\overline{CDIR}$, который имеет высокий уровень, если порт принимает, низкий - если выдаёт. Для нашего случая возможное состояние выходов $\overline{CDIR_A}$ и $\overline{CDIR_B}$ проанализировано в Табл. 10-3.

Табл. 10-3. Возможное состояние выходов $\overline{CDIR_A}$ и $\overline{CDIR_B}$

$\overline{CDIR_A}$	$\overline{CDIR_B}$	**Режим работы коммуникационных портов**
0	0	Аппаратная неисправность одного или обоих портов
0	1	Порт А находится в режиме выдачи, порт В - в режиме приёма
1	0	Порт А находится в режиме приёма, порт В - в режиме выдачи
1	1	В данный момент времени происходит изменение направления передачи. Если данная комбинация не изменяется в течение более чем одного периода синхросигнала, то это говорит о аппаратной неисправности одного или обоих портов

Рис. 10-4. Временные диаграммы арбитража шины данных коммуникационных портов



Примечания:

- вывод работает как вход (если не окрашен - работает как выход)

type 1 delay - данный тип задержки объясняется в разделе 10.9

Табл. 10-4. Описание событий, происходящих при арбитраже

Номер события	Описание события
0	Первоначально шина данных принадлежит процессору A, но ему нечего передавать процессору B
1	Процессору B необходимо передать информацию, поэтому он выставляет запрос на захват шины (сигнал $\overline{CREQ_B}$ меняет свой уровень с высокого на низкий)
2	Спустя некоторое время, связанное с задержкой прохождения сигнала, процессор A обнаруживает запрос на захват шины (сигнал $\overline{CREQ_A}$ имеет низкий уровень)
3	Процессор A соглашается на захват шины и сигнализирует об этом, меняя уровень своего сигнала $\overline{CACK_A}$ с высокого на низкий спустя определённое время (type 1 delay)
4	Спустя время, связанное с задержкой прохождения сигнала, процессор B видит разрешение на захват шины (сигнал $\overline{CACK_B}$ имеет низкий уровень)
5	Процессор A переключает вывод $\overline{CRDY_A}$ с приёма на выдачу высокого уровня
6	Процессор A переключает шину $CD_A(7-0)$ на приём
7	Процессор B переключает вывод $\overline{CSTRB_B}$ с приёма на выдачу высокого уровня
8	Процессор B снимает свой запрос на захват шины (сигнал $\overline{CREQ_B}$ меняет свой уровень с низкого на высокий) спустя определённое время (type 1 delay) после приёма низкого уровня $\overline{CACK_B}$
9	Спустя время, связанное с задержкой прохождения сигнала, процессор A видит высокий уровень $\overline{CREQ_A}$
10	Процессор A переключает вывод $\overline{CREQ_A}$ с приёма на выдачу высокого уровня
11	Процессор A меняет уровень своего сигнала $\overline{CACK_A}$ с низкого на высокий
12	Процессор A переключает вывод $\overline{CACK_A}$ на приём
13	Процессор A переключает вывод $\overline{CSTRB_A}$ на приём

Табл. 10-4. Описание событий, происходящих при арбитраже (Продолжение)

Номер события	Описание события
14	Процессор A меняет уровень сигнала $\overline{CDIR_A}$ с низкого на высокий
15	Процессор B переключает вывод $\overline{CREQ_B}$ на приём
16	Процессор B переключает вывод $\overline{CACK_B}$ с приёма на выдачу высокого уровня
17	Процессор B переключает вывод $\overline{CACK_B}$ на приём
18	Процессор B переключает шину $CD_B(7-0)$ на выдачу
19	Процессор B меняет уровень сигнала $\overline{CDIR_A}$ с высокого на низкий
20	Процессор B выдаёт на шину $CD_B(7-0)$ первый передаваемый байт по положительному фронту своего синхросигнала
21	Процессор B выставляет низкий уровень на выводе $\overline{CSTRB_B}$ по следующему положительному фронту синхросигнала
22	Спустя время, связанное с задержкой прохождения сигнала, процессор A видит первый принимаемый байт на шине $CD_A(7-0)$
23	Спустя время, связанное с задержкой прохождения сигнала, процессор A видит низкий уровень $\overline{CSTRB_A}$, подтверждающий достоверность принимаемых данных
24	Процессор A читает данные и затем выставляет низкий уровень на выводе $\overline{CRDY_A}$

10.8 Временные диаграммы передачи информации по шине данных коммуникационных портов

Передача информации по шине данных коммуникационных портов осуществляется побайтно по четыре байта за одну посылку (первым передаётся младший байт). Она осуществляется с помощью сигналов стробирования $\overline{CSTRB}$ и готовности $\overline{CRDY}$. Временная диаграмма данной передачи приведена на Рис. 10-5. Для имен сигналов на этом рисунке используются суффиксы, говорящие, какому процессору они принадлежат. Например, $\overline{CSTRB_B}$ обозначает сигнал $\overline{CSTRB}$ процессора B. Табл. 10-5 содержит описание отраженных на рисунке событий, а временные параметры можно увидеть в разделе 13.2.

Обмен байтами производится полностью ассинхронно, и скорость этого обмена может быть больше, чем один байт за период тактового сигнала *CLK*. Исключение составляет передача первого байта. Данные в этом случае со стороны передатчика появляются по переднему фронту его тактового сигнала *CLK*. Через такт по тому же фронту *CLK* сигнал строба передатчика $\overline{CSTRB}$ меняет свой уровень с высокого на низкий.

Приёмник обнаруживает низкий уровень $\overline{CSTRB}$ и, если готов к приёму, выдаёт низкий уровень сигнала $\overline{CRDY}$, что говорит передатчику: первый байт принят. После этого передатчик выдаёт следующий байт и меняет уровень сигнала $\overline{CSTRB}$ на высокий. Далее приёмник меняет уровень $\overline{CRDY}$ на высокий, а в ответ на это передатчик снова выдаёт низкий уровень сигнала $\overline{CSTRB}$ и т.д. пока не передадутся все четыре байта в посылке.

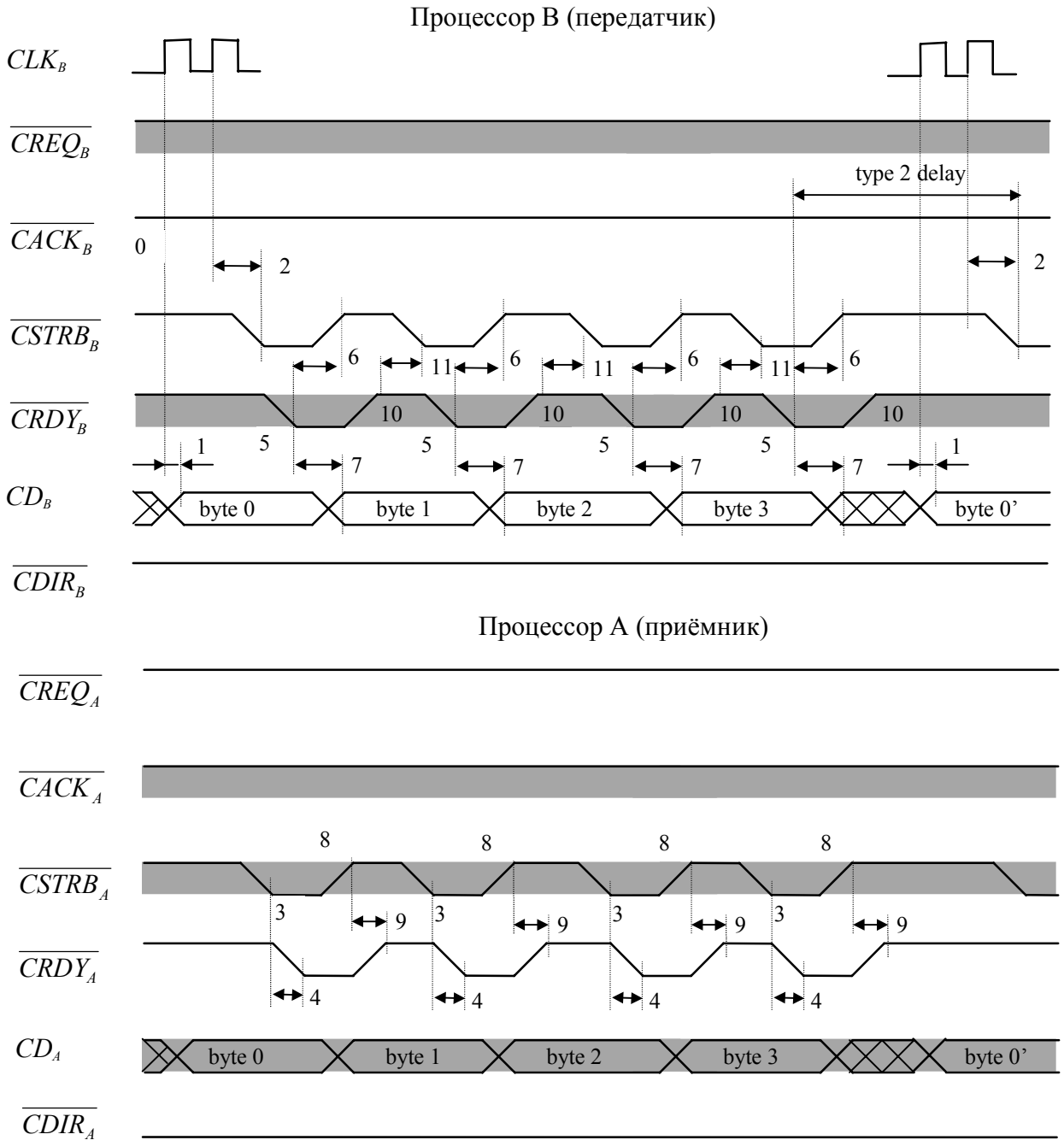
Между двумя соседними посылками вставляется определённая задержка - type 2 delay, которая объясняется в разделе 10.9. Данная задержка позволяет передатчику синхронизироваться относительно своего тактового сигнала, чтобы начать передачу первого байта в посылке, как описано выше.

Если входной буфер приёмника полон, то он просто не выдаст низкий уровень $\overline{CRDY}$, сигнализируя передатчику, что он не готов, и тем самым предотвращается потеря информации. При обратной ситуации, когда не готов передатчик, не будет низкого уровня $\overline{CSTRB}$.

Даже в случае, когда истинность данных гарантируется, ни в коем случае нельзя подключать линии управления $\overline{CSTRB}$ и $\overline{CRDY}$ к земле. В противном случае правильная работа коммуникационных портов нарушится и передачи осуществляться не будет.

Следует обратить особое внимание на то, что передача посылками по четыре байта ведётся только из-за необходимости достичь аппаратную совместимость с коммуникационными портами TMS320C4x. На самом деле, процессор NM6403 обменивается по 64-разряда (восемь байт) за две посылки по четыре байта, начиная с младших (0-й,1-й,2-й и 3-й байты в первой посылке и 4-й,5-й,6-й и 7-й байты во второй). При этом возможен останов каналов ввода или вывода после передачи только одной посылки, как и у TMS320C4x. Как только останов снимется, передача оставшейся информации будет продолжена без потерь. Единственное ограничение, которое существует для TMS320C4x, связанного с процессором NM6403 через коммуникационные порты, - передавать или принимать чётное число 32-разрядных слов.

Рис. 10-5. Временные диаграммы передачи информации по шине данных коммуникационных портов



Примечания:

- вывод работает как вход (если не окрашен - работает как выход)

type 2 delay - данный тип задержки объясняется в разделе 10.9

byte 0' - первый байт следующей посылки

Табл. 10-5. Описание событий, происходящих при обмене данными

Номер события	Описание события
0	Шина данных принадлежит процессору B, и ему есть что передать
1	Процессор B выставляет первый байт на шину $CD_B(7-0)$ по переднему фронту своего синхросигнала CLK_B
2	Процессор B меняет уровень своего сигнала $\overline{CSTRB_B}$ на низкий по переднему фронту синхросигнала CLK_B
3	Спустя время, связанное с задержкой прохождения сигнала, процессор A видит низкий уровень сигнала $\overline{CSTRB_A}$, сигнализирующий об истинности данных
4	Процессор A читает данные и затем выставляет низкий уровень на выводе $\overline{CRDY_A}$
5	Спустя время, связанное с задержкой прохождения сигнала, процессор B видит низкий уровень сигнала $\overline{CRDY_B}$, что значит - данные прочитаны
6	Процессор B меняет уровень своего сигнала $\overline{CSTRB_B}$ на высокий
7	Процессор B выставляет следующий байт на шину $CD_B(7-0)$
8	Спустя время, связанное с задержкой прохождения сигнала, процессор A видит высокий уровень сигнала $\overline{CSTRB_A}$
9	Процессор A выставляет высокий уровень на выводе $\overline{CRDY_A}$
10	Спустя время, связанное с задержкой прохождения сигнала, процессор B видит высокий уровень сигнала $\overline{CRDY_B}$
11	Процессор B меняет уровень своего сигнала $\overline{CSTRB_B}$ на низкий
События 3-11 повторяются дважды для байтов 2 и3 (асинхронный обмен)	
12	Процессор B выставляет неопределённую информацию на шину $CD_B(7-0)$

10.9 Синхронизация при работе коммуникационных портов

При передаче данных по коммуникационным портам и при арбитраже иногда требуется синхронизация относительно своего тактового сигнала *CLK*. Устройство арбитража порта использует три типа синхронизации:

- **синхронизация типа 1 (type 1 delay)** - вызывает задержку от одного до двух машинных тактов между приёмом сигнала и выдачей на него отклика. Приём осуществляется по заднему фронту тактового сигнала *CLK,* а выдача отклика - по следующему заднему фронту тактового сигнала.

Минимальная задержка в один машинный такт произойдёт в случае, если входной сигнал меняется как раз по заднему фронту тактового сигнала. Эта задержка показана на Рис. 10-6. Максимальная задержка в два машинных такта будет, если входной сигнал меняется сразу после заднего фронта тактового сигнала. Эта задержка показана на Рис. 10-7.

Рис. 10-6. Минимальная задержка при синхронизации типа 1 (type 1 delay)

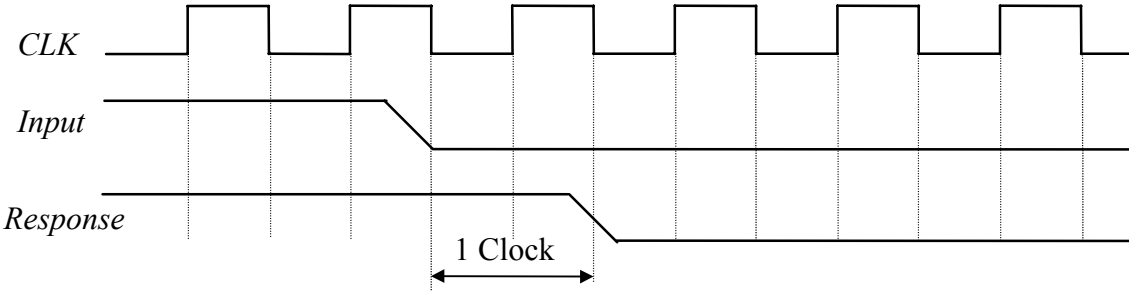
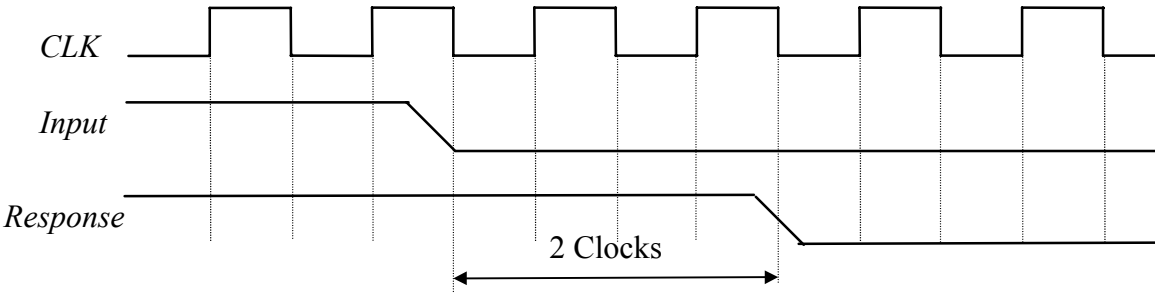


Рис. 10-7. Максимальная задержка при синхронизации типа 1 (type 1 delay).



- **синхронизация типа 2 (type 2 delay)** - вызывает задержку от 1,5 до 2,5 машинных тактов между приёмом сигнала и выдачей на него отклика. Приём осуществляется по заднему фронту тактового сигнала *CLK,* а выдача отклика - через 1,5 машинных такта по переднему фронту тактового сигнала.

Минимальная задержка в 1,5 машинных такта произойдёт в случае, если входной сигнал меняется как раз по заднему фронту тактового сигнала. Эта задержка показана на Рис. 10-8.

Максимальная задержка в 2,5 машинных такта будет, если входной сигнал меняется сразу после заднего фронта тактового сигнала. Эта

задержка показана на Рис. 10-9.

Рис. 10-8. Минимальная задержка при синхронизации типа 2 (type 2 delay).

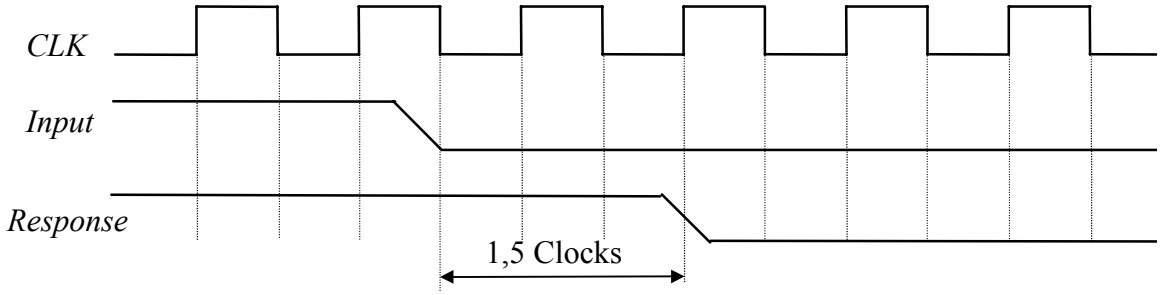
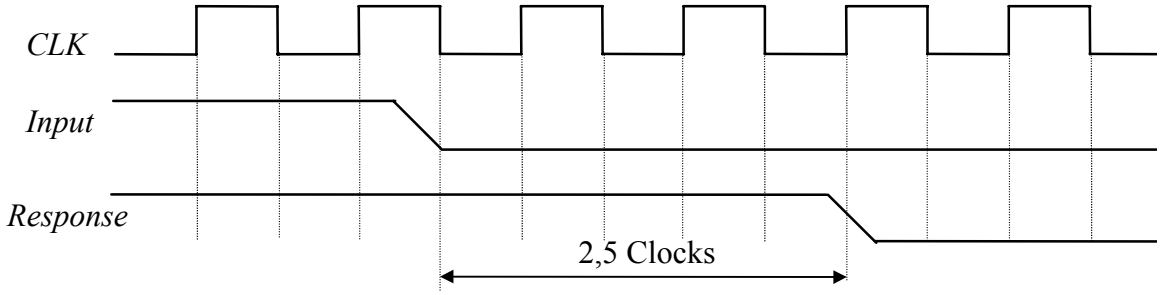


Рис. 10-9. Максимальная задержка при синхронизации типа 2 (type 2 delay).



- **синхронизация типа 3 (type 3 delay)** - вызывает задержку от 0,5 до 1,5 машинных тактов между приёмом сигнала и выдачей на него отклика. Приём осуществляется по заднему фронту тактового сигнала *CLK,* а выдача отклика - по следующему положительному фронту тактового сигнала.

Минимальная задержка в 0,5 машинных такта произойдёт в случае, если входной сигнал меняется как раз по заднему фронту тактового сигнала. Эта задержка показана на Рис. 10-10.

Максимальная задержка в 1,5 машинных такта будет, если входной сигнал меняется сразу после заднего фронта тактового сигнала. Эта задержка показана на Рис. 10-11.

Рис. 10-10. Минимальная задержка при синхронизации типа 3 (type 3 delay).

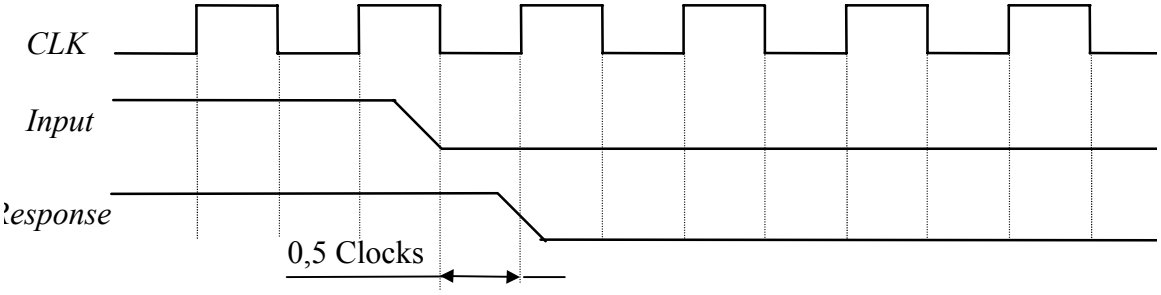
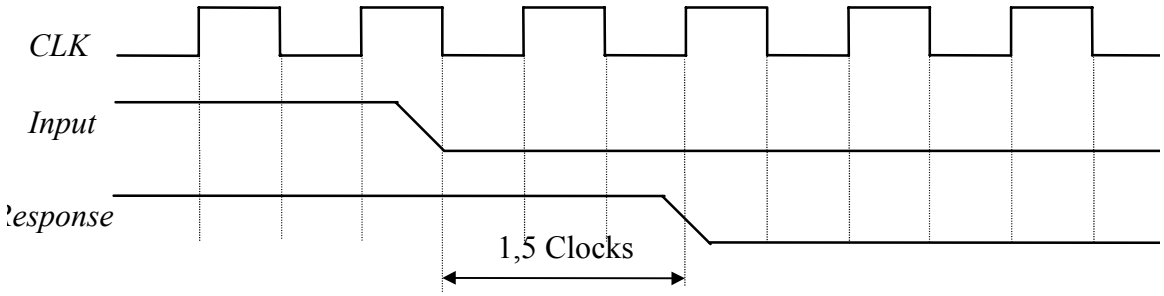


Рис. 10-11. Максимальная задержка при синхронизации типа 3 (type 3 delay).



В Табл. 10-6. представлены задержки всех трёх типов для соответствующих сигналов коммуникационных портов.

Табл. 10-6. Сигналы коммуникационных портов и задержки синхронизации

Входной сигнал и его отклик	Тип задер-жки	Мин. задержка (машинные такты)	Макс. задержка (машинные такты)
$\overline{CREQ}\downarrow$ и $\overline{CACK}\downarrow$	1	1	2
$\overline{CACK}\downarrow$ и $\overline{CREQ}\uparrow$	1	1	2
$\overline{CRDY}\downarrow$ и $\overline{CSTRB}\downarrow$ между передачей двух посылок по четыре байта	2	1,5	2,5
$\overline{CRDY}\downarrow$ и выдача первого байта данных по шине $CD_B(7-0)$ в начале посылки	3	0,5	1,5
$\overline{CACK}\downarrow$ и переключение $\overline{CSTRB}$ с приёма на выдачу высокого уровня	3	0,5	1,5

10.10 Действия программиста для активизации обмена по коммуникационным портам

Чтобы осуществить обмен по коммуникационным портам, требуется провести инициализацию соответствующих каналов ввода или вывода программно. Пока не осуществлена инициализация, ни прием, ни передача не могут быть выполнены, за исключением загрузки программы инициализации процессора NM6403 по коммуникационному порту 1 после системного сброса.

При выдаче по коммуникационному порту x (x = 0,1) осуществляются следующие действия:

1) Инициализируется канал вывода, т.е. программно осуществляется запись в регистр адреса памяти/источника OCAx и в

счетчик передаваемых 64-разрядных двойных слов OCCx, причём это можно сделать либо одной командой, либо парой, но в этом случае обязательно сначала идёт запись в OCAx, а затем в OCCx. Канал вывода начинает работу, как только произошла запись в OCCx. Если необходимо передать N двойных слов, где N может быть в диапазоне от 1 до 2^{32}, то в OCCx нужно занести число, равное -1*N в дополнительном коде. Например, для выдачи одного двойного слова требуется записать код -1 (FFFFFFFF hex), для выдачи 2^{32} двойных слов - код 0 (00000000 hex).

2) Осуществляется захват шины данных коммуникационного порта, если это не было сделано ранее.

3) В буферный регистр данных порта OCDRx в режиме ПДП записывается 64-разрядное слово из памяти по адресу, хранящемуся в регистре адреса OCAx, содержимое регистра адреса увеличивается на два.

4) 64-разрядное слово передается побайтно по шине данных порта и в конце передачи счетчик OCCx увеличивается на единицу.

5) Повторяются п.3 и п.4, пока содержимое счетчика не станет равным нулю.

6) При равенстве нулю счетчика вырабатывается прерывание по завершению выдачи и дальнейший вывод может осуществляться по данному порту только после следующей программной инициализации.

При приеме по коммуникационному порту x (x = 0,1) можно повести инициализацию канала ввода двумя способами - программно, когда сам программист определяет, сколько 64-разрядных слов будет принято и по какому адресу в память они будут записаны, либо от внешнего источника, когда это определяется по первому принятому слову.

При программной инициализации осуществляются следующие действия:

Инициализируется канал ввода, т.е. программно осуществляется запись в регистр адреса памяти/приемника ICAx и счетчик принимаемых слов ICCx, причём это можно сделать либо одной командой, либо парой, но в этом случае обязательно сначала идёт запись в ICAx, а затем в ICCx. Канал ввода начинает работу, как только произошла запись в ICCx. Если необходимо принять N 64-разрядных слов, где N может быть в диапазоне от 1 до 2^{32}, то в OCCx нужно занести число, равное -1*N в дополнительном коде. Например, для приёма одного слова требуется записать код -1 (FFFFFFFF hex) , для приёма 2^{32} двойных слов - код 0 (00000000 hex).

1) Осуществляется передача управления шиной данных коммуникационного порта внешнему устройству, если это не было сделано ранее.
2) При приеме побайтно всего 64-разрядного слова в буферный регистр коммуникационного порта ICDRx производится запись

его содержимого в память в режиме ПДП по адресу, определяемому регистром ICAx. Затем содержимое этого регистра увеличивается на два, а счетчик принимаемых слов увеличивается на единицу.

3) Повторяется п.2 и п.3, пока содержимое счетчика передаваемых слов не станет равным нулю.
4) При равенстве нулю счетчика вырабатывается прерывание по завершению приема и дальнейший прием не осуществляется и сигнал готовности не выдается, пока не произойдет следующая инициализация.

При инициализации от внешнего источника осуществляются следующие действия:

1) Активизируется нужный канал ввода x (x = 0,1) путём установки соответствующего бита CPxI в регистре PSWR командой формата 2.3.
2) Принимается первое 64-разрядное слово, которое загружается в ICCx (старшие 32 разряда) и в ICAx (младшие 32 разряда), после чего сбрасывается соответствующий бит внешней инициализации в регистре PSWR, и осуществляется обычный приём, пока ICCx не станет равным нулю.

Сразу же после конца приема/передачи по коммуникационному порту может быть осуществлена передача/прием, если соответствующий канал инициализирован.

Поясним всё вышесказанное примерами. Пусть два процессора NM6403 - NP1 и NP2 - связаны через коммуникационные порты, а именно, порт 1 процессора NP1 соединен с портом 0 процессора NP2.

Пример 1

Требуется передать массив из 100 64-разрядных чисел от NP1 к NP2.

Для NP1 программа на ассемблере может выглядеть следующим образом:

```
Out_data : long[100];          // Описание передаваемого массива
Out_data_size : word = -100;   // Описание переменной, содержащей число
                               // передаваемых двойных слов
pswr clear 008000000h;         // Очищаем бит останова вывода по порту 1 (OCH1),
                               // если он был взведён
oca1 = Out_data;               // Запись в OCA1 адреса передаваемого массива
occ1 = [Out_data_size];        // Запись в OCC1 числа передаваемых двойных
                               // слов, тем самым активизируем выдачу по порту 1
```

Для NP2 программа на ассемблере может быть такой:

```
In_data : long[100];          // Описание принимаемого массива
In_data_size : word = -100;   // Описание переменной, содержащей число
                              // принимаемых двойных слов
pswr clear 00200000h;         // Очищаем бит останова ввода по порту 0 (ICH0),
                              // если он был взведён
ica0 = In_data;               // Запись в ICA0 адреса принимаемого массива
icc0 = [In_data_size];        // Запись в ICC0 числа принимаемых двойных
                              // слов, тем самым активизируем приём по порту 0
```

Пример2

Пусть нужно передать пакет от NP2 к NP1. Данный пакет содержит заголовок, который говорит о том, что передаются 50 64-разрядных двойных слов, которые NP1 должен принять и положить по адресу 0 hex.

Для NP1 программа на ассемблере может выглядеть следующим образом:

```
pswr clear 01000000h;  // Очищаем бит останова ввода по порту 1 (ICH1), если он был взведён
pswr set 02000000h;    // Активизируем приём по порту 1 с инициализацией по первому
                       //принимаемому слову (взводим бит CP1I)
```

Для NP2 программа на ассемблере может быть такой:

```
struct Packet;                  //        Начало описания передаваемого пакета

   Np1_data_addr : word = 0h;   // Адрес, по которому NP1 положит принятый массив

   Data_size : word = -50;      // Описание переменной, содержащей число двойных слов,
                                // которые NP1 должен принять
   Out_data : long[50];         // Описание передаваемого массива
end Packet;                     //        Конец описания передаваемого пакета
   pswr clear 00100000h;        // Очищаем бит останова вывода по порту 0 (OCH0), если он
                                // был взведён
   oca0 = Packet;               // Запись в OCA0 адреса передаваемого массива
   occ0 = - sizeof(packet) / 2; // Запись в OCC0 числа передаваемых слов в пакете, тем
                                // самым активизируем выдачу по порту 1
```

10.11 Системный сброс

Данный раздел описывает состояние коммуникационных портов процессора NM6403 во время и после системного сброса. Временные диаграммы для данного случая см. в разделе 13.5.

После включения питания состояние процессора зависит от того, что подаётся на вход системного сброса $\overline{RESET}$:

Если на входе $\overline{RESET}$ стоит сигнал высокого уровня, то состояние процессора не определено, и не возможно предсказать работу буферов ввода/вывода коммуникационных портов.

Если на входе $\overline{RESET}$ сигнал низкого уровня, выводы коммуникационных портов находятся в высокоимпедансном состоянии. Все каналы ввода/вывода очищаются, и вся информация в регистрах данных каналов пропадает. Чтобы гарантировать правильную работу после системного сброса, рекомендуется линии управления портов подключить через резистор к питанию.

После системного сброса (после изменения уровня с низкого на высокий на входе $\overline{RESET}$) коммуникационный порт 0 начинает работать на выдачу, т.е. режим работы CPI - 00_2. Состояние его выводов будет следующим:

- $\overline{CREQ0}$ и $\overline{CRDY0}$ находятся в режиме приёма;
- $\overline{CACK0}$ и $\overline{CSTRB0}$ выдают высокий уровень;
- шина $C0D(7-0)$ находится в неопределенном состоянии.

Счётчики каналов OCC0 и ICC0, а также регистры адресов каналов OCA0 и ICA0 неопределенны. Биты управления портом 0 (разряды 22-20 регистра PSWR) принимают следующие значения:

- 22-й р. (CP0I) равен нулю (инициализация канала ввода порта 0 осуществляется программно);
- 21-й р. (ICH0) равен нулю (нет останова канала ввода порта 0);
- 20-й р. (OCH0) равен нулю (нет останова канала вывода порта 0).

Коммуникационный порт 1 после системного сброса начинает работать на приём, т.е. режим работы CPI - 01_2. Состояние его выводов будет следующим:

- $\overline{CREQ1}$ и $\overline{CRDY1}$ выдают высокий уровень;
- $\overline{CACK1}$ и $\overline{CSTRB1}$ находятся в режиме приёма;
- шина $C0D(7-0)$ находится в высокоимпедансном состоянии.

Счётчики каналов OCC1 и ICC1, а также регистры адресов каналов OCA1 и ICA1 неопределенны. Биты управления портом 1 (разряды 25-23 регистра PSWR) принимают следующие значения:

- 25-й р. (CP1I) равен нулю (инициализация канала ввода порта 1 осуществляется программно);
- 24-й р. (ICH1) равен нулю (нет останова канала ввода порта 1);
- 23-й р. (OCH1) равен нулю (нет останова канала вывода порта 1).

Во время системного сброса порт 0 конфигурируется как порт вывода, порт 1 - как порт ввода. Поэтому в мультипроцессорной системе необходимо объединять порт одного процессора NM6403 с разноимённым портом другого (порт 0 с портом 1 и порт 1 с портом 0). В случае работы с TMS320C4x порт 0 процессора NM6403 может быть соединён с портом 3,4 или 5 этого ЦПС, а порт 1 - с портами 0,1 или 2.

11 Програмные приложения

В данной главе приводятся примеры написания программ на языке ассемблера для процессора NM6403. Её содержание не претендует на полноту, а носит скорее ознакомительный характер. Более подробные сведения о методах программирования и языке ассемблера для процессора NM6403 можно найти в руководстве для программирования.

11.1 Примеры работы с различными типами данных

Данный раздел содержит пример написания макросов на языке ассемблера для процессора NM6403. Здесь можно найти макросы для передачи и возврата аргументов из вызываемых модулей, обработку 8-, 16-, 32- и 64-разрядных данных и т.д. Листинг на языке ассемблера приводится ниже.

Пример 11.1. Макросы для обработки различных типов данных

```
// Базовое программное обеспечение процессора NM6403
// Макросы для генератора ассемблерного кода, обеспечивающие
// работу со встроенными типами данных
//-------------------------------------------------
// Макросы конвенции о вызовах из библиотеки libasm
//-------------------------------------------------
// Функции из библиотеки libasm имеют свою собственную конвенцию
// о вызовах. Данная конвенция различается для функций с различным
// числом параметров.
// Макросы CALL_BUILTIN_xxxx задают конвенцию о вызове для функции
// в зависимости от числа аргументов.
// Каждый регистр рассматривается как отдельный аргумент.
// CALL_BUILTIN_2x1 - функция принимает два аргумента, возвращает один
// CALL_BUILTIN_4x2 - функция принимает четыре аргумента, возвращает два
// CALL_BUILTIN_2x2 - функция принимает два аргумента, возвращает два
// CALL_BUILTIN_1x2 - функция принимает один аргумент, возвращает два
//=================================================
// Макрос вызывает функцию с двумя параметрами
// возвращаемое значение помещается в один регистр
//
// CALL_BUILTIN_2x1(
//        Func,   - Имя вызываемой функции
//        Arg1,   - регистр первого аргумента
//        Arg2,   - регистр второго аргумента
//        Res     - регистр результата
//    )
//
```

Пример 11.1. Продолжение

```
        //=======================================================
        macro CALL_BUILTIN_2x1( Func, Arg1, Arg2, Res )
            // перспективный вариант соглашения о вызовах для 2 аргументов
            // параметры функции находятся выше сохраненного PC,PSW
                delayed call Func;
                push Arg1;
                push Arg2;
            // end of delayed commands of call
                Res = gr7;
        end CALL_BUILTIN_2x1;
        //=======================================================
        // Макрос вызывает функцию с четыремя параметрами
        // возвращаемое значение помещается в два регистра
        //
        // CALL_BUILTIN_4x2(
        //      Func,   - Имя вызываемой функции
        //      Arg1,   - регистр первого аргумента
        //      Arg2,   - регистр второго аргумента
        //      Arg3,   - регистр третьего аргумента
        //      Arg4,   - регистр четвертого аргумента
        //      Res1,   - первый регистр результата
        //      Res2    - второй регистр результата
        //   )
        //
        //=======================================================
macro CALL_BUILTIN_4x2( Func, Arg1, Arg2, Arg3, Arg4, Res1, Res2 )
            ar5 = sp;
            sp += 4;
            [ ar5++ ] = Arg1;
            [ ar5++ ] = Arg2;
            [ ar5++ ] = Arg3;
            [ ar5 ] = Arg4;
            call Func;
            ar5 = sp;
            Res1 = [ --ar5 ];
            Res2 = [ --ar5 ];
            sp -= 4;
end CALL_BUILTIN_4x2;
```

Пример 11.1. Продолжение

```
//=========================================================
// Макрос вызывает функцию с двумя параметрами
// возвращаемое значение помещается в два регистра
//
// CALL_BUILTIN_2x2(
//       Func,   - Имя вызываемой функции
//       Arg1,   - регистр первого аргумента
//       Arg2,   - регистр второго аргумента
//       Res1,   - первый регистр результата
//       Res2    - второй регистр результата
//   )
//
//=========================================================
macro CALL_BUILTIN_2x2( Func, Arg1, Arg2, Res1, Res2 )
             ar5 = sp;
             sp += 2;
             [ ar5++ ] = Arg1;
             [ ar5 ] = Arg2;
             call Func;
             ar5 = sp;
             Res1 = [ --ar5 ];
             Res2 = [ --ar5 ];
             sp -= 2;
end CALL_BUILTIN_2x2;
```

Пример 11.1. Продолжение

```
//=====================================================
// Макрос вызывает функцию с одним параметром
// возвращаемое значение помещается в два регистра
//
// CALL_BUILTIN_1x2(
//       Func,   - Имя вызываемой функции
//       Arg1,   - регистр первого аргумента
//       Res1,   - первый регистр результата
//       Res2    - второй регистр результата
//   )
//
//=====================================================
macro CALL_BUILTIN_1x2( Func, Arg, Res1, Res2 )
    // перспективный вариант вызова:
    // параметр передается в gr7
    // возвращаются значения на вершине стека.
                delayed call Func with gr7 = Arg;
                nul 32;
  // call Func;
                if false delayed skip 2;
                Res1 = [ --ar7 ];
                Res2 = [ --ar7 ];
  // if never delayed skip 2;
end CALL_BUILTIN_1x2;
  //-------------------------------------------------
  // Макросы для неподдерживаемых операций с
  // короткими числами - 32 бита.
  // Макросы для работы с длинными числами 64 бита
  //-------------------------------------------------
//======== Макрос MULT32 =========
// Умножение 32-битных чисел, знаковое или беззнаковое,
// с 32-битным результатом.
// На входе:   Arg1 - регистр с первым множителем
//             Arg2 - регистр со вторым множителем
// На выходе:  Res - младшие 32 бита результата
// Изменяет:   gr7
// ЗАМЕЧАНИЕ: Arg1 и Arg2 не могут совпадать.
macro MULT32 ( Res, Arg1, Arg2 )
                with gr7 = Arg2;
                with Arg2 = Arg1 *: gr7;
.repeat 15;
                     with Arg2 = Arg1 * gr7;
.endrepeat;
                with Res = gr7;
end MULT32;
```

Пример 11.1. Продолжение

```
//======= Макрос UTRUNC8 ========
// Усечение 32-разрядного значения до 8 младших бит без распространения знака.
// На входе:    Src - регистр с операндом.
// На выходе:   Dst - регистр с результатом.
macro UTRUNC8 ( Dst, Src )
        Dst = Src << 24;
        Dst >>= 24;
end UTRUNC8;
//======= Макрос ITRUNC8 ========
// Усечение 32-разрядного значения до 8 младших бит с распространением знака
// 8-го бита.
// На входе:    Src - регистр с операндом.
// На выходе:   Dst - регистр с результатом.
macro ITRUNC8 ( Dst, Src )
        Dst = Src << 24;
        Dst A>>= 24;
end ITRUNC8;
//======= Макрос UTRUNC16 ========
// Усечение 32-разрядного значения до 16 младших бит без распространения знака.
// На входе:    Src - регистр с операндом.
// На выходе:   Dst - регистр с результатом.
macro UTRUNC16 ( Dst, Src )
        Dst = Src << 16;
        Dst >>= 16;
end UTRUNC16;
//======= Макрос ITRUNC16 ========
// Усечение 32-разрядного значения до 16 младших бит с распространением знака
// 16-го бита.
// На входе:    Src - регистр с операндом.
// На выходе:   Dst - регистр с результатом.
macro ITRUNC16 ( Dst, Src )
        Dst = Src << 16;
        Dst A>>= 16;
end ITRUNC16;
```

Пример 11.1. Продолжение

```
//======= Макрос LOAD64C ========
// Загрузка регистровой пары 64-битным значением.
// На входе:    Val - значение-константа.
// На выходе:   ResLo - регистр с младшими 32 битами,
//              ResHi - регистр со старшими 32 битами.
macro LOAD64C ( ResLo, ResHi, Val )
        ResLo = loword( Val );
        ResHi = hiword( Val );
end LOAD64C;
macro ADD64 ( ResLo, ResHi, A1Lo, A1Hi, A2Lo, A2Hi )
            gr7 = A1Lo + A2Lo;
            ResLo = gr7
                      with ResHi = A1Hi + A2Hi + carry;
end ADD64;
macro SUB64 ( ResLo, ResHi, A1Lo, A1Hi, A2Lo, A2Hi )
            gr7 = A1Lo - A2Lo;
            ResLo = gr7
                       with ResHi = A1Hi - A2Hi - 1 + carry;
end SUB64;
//======= Макрос NEG64 ========
// Обращение знака 64-разрядного числа.
// На входе:    ArgLo - младшие 32 бита операнда,
//              ArgHi - старшие 32 бита операнда.
// На выходе:   ResLo - младшие 32 бита результата,
//              ResHi - старшие 32 бита результата.
// Изменяет: gr7, ArgLo.
macro NEG64 ( ResLo, ResHi, ArgLo, ArgHi )
             ResHi = false;
             ArgLo = -ArgLo;
             ResLo = ArgLo
                  with ResHi = ResHi - ArgHi - 1 + carry;
end NEG64;
```

Пример 11.1. Продолжение

```
//======= Макрос MULT64 =========
// Умножение 64-битных чисел, знаковых либо беззнаковых.
// ЗАМЕЧАНИЕ. Внешняя переменная long_zero содержит длинный нуль
// На входе:    LeftLo - младшие 32 бита 1-го операнда,
//              LeftHi - старшие 32 бита 1-го операнда,
//              RightLo - младшие 32 бита 2-го операнда,
//              RightHi - старшие 32 бита 2-го операнда,
//              стек выровнен по четному адресу.
// На выходе:   ResLo - младшие 32 бита результата,
//              ResHi - старшие 32 бита результата.
macro MULT64( ResLo, ResHi, LeftLo, LeftHi, RightLo, RightHi )

        nb1 = 0;            // длинный нуль
        sb = 0;
        ar5 = sp;
        sp += 4;
        [ ar5++ ] = LeftLo;
        [ ar5++ ] = LeftHi;
        [ ar5++ ] = RightLo;
        [ar5++ ] = RightHi;
        rep 1 wfifo = [ --ar5 ], ftw, wtw;
        rep 1 data = [ --ar5 ]
                    with vsum , data, 0;
        rep 1 [ar5] = afifo;
        ResLo = [ ar5++ ];
        ResHi = [ ar5 ];
        sp -= 4;
end MULT64;
//======= Макрос I32TO64 =========
// Преобразование 32-битного целого со знаком в 64-битное.
// На входе:    Val - регистр с операндом
// На выходе:   ResLo - регистр с младшими 32 битами результата
//              ResHi - регистр со старшими 32 битами результата
macro I32TO64 ( ResLo, ResHi, Val )
        ResLo = Val
                    with ResHi = Val A>> 31;
end I32TO64;
//======= Макрос U32TO64 =========
// Преобразование 32-битного целого без знака в 64-битное.
// На входе:    Val - регистр с операндом
// На выходе:   ResLo - регистр с младшими 32 битами результата
//              ResHi - регистр со старшими 32 битами результата
macro U32TO64 ( ResLo, ResHi, Val )
              ResLo = Val
                      with ResHi = false;
end U32TO64;
```

Пример 11.1. Продолжение

```
//======= Макрос STASG ========
// Порождает код, копирующий (небольшой) массив данных (структуру).
// На входе:    Dst - адресный регистр с адресом реципиента (не ar5),
//          Src - адресный регистр с адресом источника (не ar5),
//          Size - длина копируемой области.
// Изменяет: Dst, Src, gr6, gr7.
macro STASG ( Dst, Src, Size )
        gr6 = sp;
        sp += 2;
        [ gr6 ] = Src with gr6++;
        [ gr6 ] = Dst;
.repeat Size;
             gr7 = [ Src++ ];
             [ Dst++ ] = gr7;
.endrepeat;
        Dst = [ gr6 ] with gr6--;
        Src = [ gr6 ];
        sp -= 2;
end STASG;
//===== Макрос UGFLD =======
// Доступ к  беззнаковому битовому полю структуры
// На входе:    Addr    - регистр с адресом 32-битного слова
//          Offset  - смещение от начала слова в битах
//          Size    - размер поля в битах
// На выходе:   Res     - регистр с результатом
// Замечание: данные в структуре расположены от младших бит к старшим.
macro UGFLD ( Res, Addr, Offset, Size )
        Res = [ Addr ];
        Res <<= 32 - Offset - Size;
        Res >>= 32 - Size;
end UGFLD;
//===== Макрос IGFLD =======
// Доступ к  знаковому битовому полю структуры
// На входе:    Addr    - регистр с адресом 32-битного слова
//          Offset  - смещение от начала слова в битах
//          Size    - размер поля в битах
// На выходе:   Res     - регистр с результатом
// Замечание: данные в структуре расположены от младших бит к старшим.
macro IGFLD ( Res, Addr, Offset, Size )
        Res = [ Addr ];
        Res <<= 32 - Offset - Size;
        Res A>>= 32 - Size;
end IGFLD;
```

Пример 11.1. Продолжение

```
//===== Макрос SFLD0 =======
// Присваивание битовому полю структуры.
// На входе:    Addr    - регистр с адресом 32-битного слова
//          Offset  - смещение от начала слова в битах
//          Size    - размер поля в битах
//          Val     - присваиваемое полю значение
//                    (регистр кроме gr0 либо константа)
// Замечание: данные в структуре расположены от младших бит к старшим.
macro SFLD0 ( Addr, Offset, Size, Val )
        push ar0, gr0;
        gr0 = [ Addr ];
        gr7 = ( 0ffffffffh << ( Offset + Size ) ) or
                    ( 0ffffffffh >> ( 32 - Offset ) ); // маска
        gr0 = gr0 and gr7;  // обнуляем поле
        gr7 = Val << ( 32 - Size );
        gr7 >>= ( 32 - Offset - Size );
        gr7 = gr0 or gr7;   // устанавливаем поле
        pop ar0, gr0;
        [ Addr ] = gr7;
end SFLD0;
//===== Макрос SFLD1 =======
// Присваивание битовому полю структуры.
// На входе:    Addr    - регистр с адресом 32-битного слова
//          Offset  - смещение от начала слова в битах
//          Size    - размер поля в битах
//          Val     - присваиваемое полю значение
//                    (регистр кроме gr1 либо константа).
// Замечание: данные в структуре расположены от младших бит к старшим.
macro SFLD1 ( Addr, Offset, Size, Val )
        push ar1, gr1;
        gr1 = [ Addr ];
        gr7 = ( 0ffffffffh << ( Offset + Size ) ) or
                    ( 0ffffffffh >> ( 32 - Offset ) ); // маска
        gr1 = gr1 and gr7;  // обнуляем поле
        gr7 = Val << ( 32 - Size );
        gr7 >>= ( 32 - Offset - Size );
        gr7 = gr1 or gr7;   // устанавливаем поле
        pop ar1, gr1;
        [ Addr ] = gr7;
end SFLD1;
```

11.2 Управление работой коммуникационных портов ввода/вывода

Данный раздел содержит пример написания функций на языке ассемблера, позволяющих управлять работой коммуникационных портов 0 и 1 процессора NM6403. С помощью этих функций можно активизировать приём и передачу по портам, приостановить (замаскировать) каждое из этих действий и т.д. Листинг на языке ассемблера приводится ниже.

Пример 11.2. Функции управления коммуникационными портами 0 и 1 процессора NM6403

```
// Базовое программное обеспечение процессора NM6403
// Функции для работы с коммуникационными портами процессора
      const CP1I = 02000000h;   // бит внешней инициализации канала ввода порта 1
      const ICH1 = 01000000h;   // бит останова канала ввода порта 1
      const OCH1 = 00800000h;   // бит останова канала вывода порта 1
      const CP0I = 00400000h;   // бит внешней инициализации канала ввода порта 0
      const ICH0 = 00200000h;   // бит останова канала ввода порта 0
      const OCH0 = 00100000h;   // бит останова канала вывода порта 0
begin text
           global _CpPrgrmInit : label;
           global _CpFrstWordInit : label;
           global _CpInHalt : label;
           global _CpInRelease : label;
           global _CpOutHalt : label;
           global _CpOutRelease : label;
           global _CpSendMsg : label;
           global _CpRecieveMsg : label;
           global _GetCpInCount : label;
           global _GetCpOutCount : label;
//----------------------------------------
// size_t CpPrgrmInit ( int ch_no );
// инициализация канала ввода порта ch_no (сколько 64-разрядных слов
// принять и по какому адресу записать в память) осуществляется
// программно
    <_CpPrgrmInit>
        push ar6;
        ar6 = ar7 - 4;
       gr7 = [ar6];  // gr7 = ch_no
        pop ar6
            with gr7;
        if =0 goto _Cp0PrgrmInit;   // если порт 0, перейти на метку _Cp0PrgrmInit
     <_Cp1PrgrmInit>
           gr7 = pswr;
           pswr clear CP1I;   // обнулить бит внешней инициализации
                              // канала ввода порта 1
                    return;
```

Пример 11.2.Продолжение

```
            <_Cp0PrgrmInit>
            gr7 = pswr;
            pswr clear CP0I;   // обнулить бит внешней инициализации
                               // канала ввода порта 0
                              return;
//----------------------------------------
// size_t CpFrstWordInit ( int ch_no );
// инициализация канала ввода порта ch_no (сколько 64-разрядных слов
// принять и по какому адресу записать в память) осуществляется по
// первому принятому слову
      <_CpFrstWordInit>
                  push ar6;
                  ar6 = ar7 - 4;
                  gr7 = [ar6];  // gr7 = ch_no
                  pop ar6
                        with gr7;
if =0 goto _Cp0FrstWordInit;   // если порт 0, перейти на метку  _Cp0FrstWordInit
            <_Cp1FrstWordInit>
                        gr7 = pswr;
                        pswr set CP1I;   // установить бит внешней инициализации
                                         // канала ввода порта 1
                              return;
            <_Cp0FrstWordInit>
                        gr7 = pswr;
                        pswr set CP0I;  // установить бит внешней инициализации
                                        // канала ввода порта 0
                              return;
//----------------------------------------
// size_t CpInHalt ( int ch_no );
// останов канала ввода порта ch_no
      <_CpInHalt>
                  push ar6;
                  ar6 = ar7 - 4;
                  gr7 = [ar6];  // gr7 = ch_no
                  pop ar6
                        with gr7;
            if =0 goto _Cp0InHalt;   // если порт 0, перейти на метку _Cp0InHalt
            <_Cp1InHalt>
                  gr7 = pswr;
                  pswr set ICH1;   // установить бит останова канала ввода порта 1
                              return;
            <_Cp0InHalt>
                  gr7 = pswr;
                  pswr set ICH0;   // установить бит останова канала  ввода порта 0
                              return;
```

Пример 11.2.Продолжение

```
//----------------------------------------
// size_t CpInRelease ( int ch_no );
// разрешение принимать данные по порту ch_no

      <_CpInRelease>
            push ar6;
            ar6 = ar7 - 4;
            gr7 = [ar6];   // gr7 = ch_no
            pop ar6
                  with gr7;
         if =0 goto _Cp0InRelease;   // если порт 0, перейти на метку _Cp0InRelease
            <_Cp1InRelease>
                  gr7 = pswr;
                  pswr clear ICH1; // обнулить бит останова канала ввода порта 1
                              return;
            <_Cp0InRelease>
                  gr7 = pswr;
                  pswr clear ICH0;   // обнулить бит останова канала ввода порта 0
                        return;
//----------------------------------------
// size_t CpOutHalt ( int ch_no );
// останов канала вывода порта ch_no
      <_CpOutHalt>
                  push ar6;
                  ar6 = ar7 - 4;
                  gr7 = [ar6];   // gr7 = ch_no
                  pop ar6
                        with gr7;
            if =0 goto _Cp0OutHalt;   // если порт 0, перейти на метку _Cp0OutHalt
            <_Cp1OutHalt>
                  gr7 = pswr;
                  pswr set OCH1;   // установить бит останова канала вывода порта 1
                              return;
            <_Cp0OutHalt>
                  gr7 = pswr;
                  pswr set OCH0;   // установить бит останова канала вывода порта 0
                              return;
```

Пример 11.2.Продолжение

```
//----------------------------------------
// size_t CpOutRelease ( int ch_no );
// разрешение выдавать данные по порту ch_no
      <_CpOutRelease>
            push ar6;
            ar6 = ar7 - 4;
            gr7 = [ar6];   // gr7 = ch_no
            pop ar6
            with gr7;
        if =0 goto _Cp0OutRelease;   // если порт 0, перейти на метку _Cp0OutRelease
            <_Cp1OutRelease>
            gr7 = pswr;
            pswr clear OCH1;   // обнулить бит останова канала вывода порта 1
                        return;
            <_Cp0OutRelease>
            gr7 = pswr;
            pswr clear OCH0;   // обнулить бит останова канала вывода порта 0
                        return;
//----------------------------------------
// void CpSendMsg ( int ch_no, size_t BufSize, void * BufAddr );
// передача BufSize 64-разрядных слов с адреса BufAddr по порту ch_no
      <_CpSendMsg>
            push ar6,gr6;
            ar6 = ar7 - 5;
            gr6 = [ar6];   // gr6 = ch_no
            pop ar6,gr6
                        with gr6;
        if =0 goto _Cp0SendMsg;   // если порт 0, перейти на метку _Cp0SendMsg
            <_Cp1SendMsg>
                  push ar6;
                  ar6 = ar7 - 8;
                  oca1,occ1 = [ar6]; // Cp1OutAddr в регистр адреса канала вывода 1
                                     // Cp1OutCount в счетчик канала вывода 1
                  pop ar6;
                              return;
            <_Cp0SendMsg>
                        push ar6;
            ar6 = ar7 - 8;
            oca0,occ0 = [ar6];   // Cp0OutAddr в регистр адреса канала вывода 0
                                 // Cp0OutCount в счетчик канала вывода 0
                        pop ar6;
                              return;
```

Пример 11.2.Продолжение

```
//----------------------------------------
// void CpRecieveMsg ( int ch_no, size_t BufSize, void * BufAddr);
// прием BufSize 64-разрядных слов по адресу BufAddr по порту ch_no
<_CpRecieveMsg>
            push ar6,gr6;
            ar6 = ar7 - 5;
            gr6 = [ar6];   // gr6 = ch_no
            pop ar6,gr6
            with gr6;
          f =0 goto _Cp0RecieveMsg;  // если порт 0, перейти на метку _Cp0RecieveMsg
            <_Cp1RecieveMsg>
                  push ar6;
                  ar6 = ar7 - 8;
                  ica1,icc1 = [ar6];   // Cp1InAddr в регистр адреса канала ввода 1
                                       // Cp1InCount в счетчик канала ввода 1
                  pop ar6;
                              return;
            <_Cp0RecieveMsg>
                  push ar6;
                  ar6 = ar7 - 8;
                  ica0,icc0 = [ar6];   // Cp0InAddr в регистр адреса канала ввода 0
                                       // Cp0InCount в счетчик канала ввода 0
                  pop ar6;
                              return;
//----------------------------------------
// size_t GetCpInCount ( int ch_no );
// функция возвращает значение счетчика канала ввода порта ch_no
      <_GetCpInCount>
                  push ar6;
                  ar6 = ar7 - 4;
                  gr7 = [ar6];   // gr7 = ch_no
                  pop ar6
                  with gr7;
           if =0 goto _GetCp0InCount;  // если порт 0, перейти на метку _GetCp0InCount
          <_GetCp1InCount>
                        gr7 = icc1;
                              return;
          <_GetCp0InCount>
                        gr7 = icc0;
                              return;
```

Пример 11.2.Продолжение

```
//---------------------------------------
// size_t GetCpOutCount ( int ch_no );
// функция возвращает значение счетчика канала вывода порта ch_no
 <_GetCpOutCount>
        push ar6;
        ar6 = ar7 - 4;
        gr7 = [ar6];   // gr7 = ch_no
        pop ar6
                with gr7;
       if =0 goto _GetCp0OutCount;// если порт 0, перейти на метку _GetCp0OutCount
           <_GetCp1OutCount>
                     gr7 = occ1;
                          return;
           <_GetCp0OutCount>
                     gr7 = occ0;
                          return;
end text;
```

11.3 Управление таймерами

Данный раздел содержит пример написания функций на языке ассемблера, позволяющих управлять режимом работы таймеров 0 и 1 процессора NM6403. С помощью этих функций можно задать период работы каждого таймера, запустить его в непрерывном или однократном режимах, приостановить работу и т.д. Листинг на языке ассемблера приводится ниже.

Пример 11.3. Функции управления таймерами 0 и 1 процессора NM6403

```
// Базовое программное обеспечение процессора NM6403
// Функции для работы с таймерами процессора
      const TM0 = 080000h;   // разрешение работы таймера T0 в непрерывном режиме
      const TE0 = 020000h;   // разрешение работы таймера T0
      const TM1 = 040000h;   // разрешение работы таймера T1 в непрерывном режиме
      const TE1 = 010000h;   // разрешение работы таймера T1
      const TEN = 020000000h;      // разрешение выдачи на внешний вывод TIMER
      const TPOC = 01c000000h;  // поле определения того, что подается на
                                    // внешний вывод TIMER
```

Пример 11.3.Продолжение

```
      const tmp1 = 07h;
begin text
      global _UnIntptTimerMode : label;
      global _IntptTimerMode : label;
      global _ClearTimerCtrl : label;
      global _SetTimerCtrl : label;
      global _SetTimerCounter : label;
      global _GetTimerCounter : label;
      global _TimerPinHalt : label;
      global _TimerPinRelease : label;
      global _TimerPinOut : label;
//----------------------------------------
// size_t UnIntptTimerMode ( int timer_no );
// при обнулении таймера возникает запрос на прерывание,
// восстанавливается содержимое t0 ( t1 ) и продолжается его работа
    <_UnIntptTimerMode>
        push ar6;
        ar6 = ar7 - 4;
        gr7 = [ar6];   // gr7 = timer_no
        pop ar6
            with gr7;
        if =0 goto _UnIntptTimer0Mode;   // если таймер 0, перейти
                                         //на метку  _UnIntptTimer0Mode
           <_UnIntptTimer1Mode>
           gr7 = pswr;
           pswr set TM1;  // установить бит разрешения работы таймера 1
                          // в непрерывном режиме
                            return;
```

Пример 11.3.Продолжение

```
            <_UnIntptTimer0Mode>
                  gr7 = pswr;
                  pswr set TM0;   // установить бит разрешения работы таймера 0
                                  // в непрерывном режиме
                              return;
//----------------------------------------
// size_t IntptTimerMode ( int timer_no );
// при обнулении таймера возникает запрос на прерывание,
// больше никаких действий не производится
      <_IntptTimerMode>
                  push ar6;
                  ar6 = ar7 - 4;
                  gr7 = [ar6];  // gr7 = timer_no
                  pop ar6
                              with gr7;
    if =0 goto _IntptTimer0Mode;// если таймер 0, перейти на метку _IntptTimer0Mode
      <_IntptTimer1Mode>
            gr7 = pswr;
            pswr clear TM1;   // обнулить бит разрешения работы таймера 1
                              // в непрерывном режиме
                        return;
      <_IntptTimer0Mode>
            gr7 = pswr;
            pswr clear TM0;   // обнулить бит разрешения работы таймера 0
                              // в непрерывном режиме
                              return;
//----------------------------------------
// size_t ClearTimerCtrl ( int timer_no );
// нет счета, если 19 ( 17 ) бит PSWR равен 0
      <_ClearTimerCtrl>
                  push ar6;
                  ar6 = ar7 - 4;
                  gr7 = [ar6];   // gr7 = timer_no
                  pop ar6
                              with gr7;
                  if =0 goto _ClearTimer0Ctrl;   // если таймер 0, перейти на
                                                 //метку _ClearTimer0Ctrl
            <_ClearTimer1Ctrl>
                  gr7 = pswr;
                  pswr clear TE1;   // обнулить бит разрешения работы таймера 1
                              return;
            <_ClearTimer0Ctrl>
                  gr7 = pswr;
                  pswr clear TE0;   // обнулить бит разрешения работы таймера 0
                              return;
```

Пример 11.3.Продолжение

```
//----------------------------------------
// size_t SetTimerCtrl ( int timer_no );
// есть счет независимо от 19 ( 17 ) бита PSWR
      <_SetTimerCtrl>
                push ar6;
                ar6 = ar7 - 4;
                gr7 = [ar6];   // gr7 = timer_no
                pop ar6
                     with gr7;
        if =0 goto _SetTimer0Ctrl;  // если таймер 0, перейти на метку _SetTimer0Ctrl
            <_SetTimer1Ctrl>
                gr7 = pswr;
                pswr set TE1;   // установить бит разрешения работы таймера 1
                           return;
            <_SetTimer0Ctrl>
                gr7 = pswr;
                pswr set TE0;   // установить бит разрешения работы таймера 0
                           return;
//----------------------------------------
// size_t SetTimerCounter ( int timer_no, size_t CounterValue );
// запись значения CounterValue в регистр t0 ( t1 ),
// возвращает предыдущее значение t0 ( t1 )
      <_SetTimerCounter>
                push ar6;
                ar6 = ar7 - 4;
                gr7 = [ar6];   // gr7 = timer_no
                pop ar6
                           with gr7;
        if =0 goto _SetCounterT0;   // если таймер 0, перейти на метку _SetCounterT0
            <_SetCounterT1>
                      push ar5;
                      ar5 = ar7 - 5;
                      gr7 = t1;
                      t1 = [ar5];   // запись значения CounterValue в регистр t1
                      pop ar5;
                           return;
            <_SetCounterT0>
                      push ar5;
                      ar5 = ar7 - 5;
                      gr7 = t0;
                      t0 = [ar5];   // запись значения CounterValue в регистр t0
                      pop ar5;
                      return;
```

Пример 11.3.Продолжение

```
//----------------------------------------
// size_t GetTimerCounter ( int timer_no );
// возвращает значение счетчика таймера timer_no

      <_GetTimerCounter>
                push ar6;
                ar6 = ar7 - 4;
                gr7 = [ar6];   // gr7 = timer_no
                pop ar6
                              with gr7;
          if =0 goto _GetCounterT0;  // если таймер 0, перейти на метку _GetCounterT0

             <_GetCounterT1>
                        gr7 = t1;
                              return;

             <_GetCounterT0>
                        gr7 = t0;
                             return;

//----------------------------------------
// size_t TimerPinHalt ( void );
// вывод TIMER находится в высокоэпидансном состоянии

      <_TimerPinHalt>
                gr7 = pswr;
                pswr clear TEN;  // обнулить бит разрешения выдачи на вывод TIMER
                      return;

//----------------------------------------
// size_t TimerPinRelease ( void );
// разрешается выдача на вывод TIMER

      <_TimerPinRelease>
           gr7 = pswr;
           pswr set TEN;   // установить бит разрешения выдачи на вывод TIMER
                   return;
```

Пример 11.3.Продолжение

```
//-----------------------------------------
// size_t TimerPinOut ( int det_out );
// определение того, что выдается на внешний вывод TIMER

      <_TimerPinOut>
                  push gr5;
                  push ar6,gr6;
                  ar6 = ar7 - 6;
                  gr6 = [ar6];   // gr6 = det_out

                  gr5 = tmp1;
                  gr7 = pswr
                              with gr6 = gr6 and gr5;
                  pswr clear TPOC   // обнулить поле определения того,
                                    // что выдается на вывод TIMER
                              with gr6 = gr6 << 26;
                  gr5 = pswr;
                  pop ar6,gr6
                              with gr5 = gr5 or gr6;
                  pswr = gr5;  // установить новое поле определения того,
                               // что выдается на вывод TIMER
                  pop gr5;
                        return;

//-----------------------------------------
end text;
```

12 Аппаратные приложения

12.1 Инициализация процессора NM6403 после системного сброса

Инициализация процессора после системного сброса может проводиться двумя способами в зависимости от уровня на входе *BOOTM* :

- если на входе *BOOTM* низкий уровень, то инициализация проводится по внешнему прерыванию;
- если на входе *BOOTM* высокий уровень, то инициализация проводится по коммуникационному порту 1.

Инициализация процессора NM6403 по внешнему прерыванию

Инициализация по внешнему прерыванию проводится после того, как сигнал $\overline{RESET}$ становится пассивным, на входе *BOOTM* низкий уровень, и пришёл отрицательный импульс на вход внешнего прерывания $\overline{INT}$ (см. рис. 13.5). Данное прерывание не фиксируется, но по нему формируется запрос на выборку первой команды по адресу C0000000 hex, вывод $\overline{INTA}$ переключается с выдачи низкого на выдачу высокого уровня и начинается обычная работа процессора. При этом надо иметь в виду, что во время системного сброса обнуляются регистры PSWR, LMICR и GMICR (см. разделы 3.3 и 6.4).

Нулевое содержимое LMICR означает следующее: на локальной шине определены два банка статической памяти - 0-й объёмом 32 К 64-разрядных слов, 1-й объёмом - (1Г - 32 К) слов, размер страницы каждого -256 слов, время обращения к любому банку - 7 тактов, режим работы с общей памятью 00 (см. главу 9).

Тоже самое относится к GMICR, за одним исключением: тип 1-го банка памяти задаётся входным уровнем на выводе *TIMER* : низкий уровень означает статику, высокий - динамику с временем обращения 3 такта.

Нулевое содержимое PSWR означает следующее:

- контроль над локальной и глобальной шиной не отдается по запросу от внешнего устройства (после системного сброса локальная шина принадлежит процессору NM6403, глобальная-нет);
- вывод *TIMER* находится в режиме приёма;
- коммуникационные порты находятся в пассивном состоянии;
- таймеры находятся в пассивном состоянии;
- замаскированы все прерывания.

Из всего вышеизложенного следует, что выборка первой команды

по адресу C0000000 hex обозначает обращение к 1-му банку глобальной шины. Данное решение принято из следующих соображений: программа инициализации должна находится в том банке памяти, к которому имеет доступ помимо процессора NM6403 другое устройство, выполняющее роль хоста, т.е. должен быть реализован режим работы с общей памятью. Это справедливо только для 1-го банка для любого из трёх возможных таких режимов (см. раздел 9.6).

Поскольку процессору после системного сброса глобальная шина не принадлежит, он, чтобы произвести выборку первой команды, выставляет запрос на владение ею (на выходе $\overline{HOLDO}$ - низкий уровень). После того, как хост записал в 1-й банк этой шины всю необходимую информацию, он может отдать контроль над ней, выдав сигнал $\overline{RDY}$ (см. рис. 13.16). Процессор NM6403 после этих действий приступает к выборке команд программы инициализации. Если хост опять потребует контроля над глобальной шиной (выставит на выводе $\overline{HOLDI}$ низкий уровень), он его не получит, пока не будет записана единица в 31-й разряд PSWR.

В программе инициализации в первую очередь рекомендуется задать следующие действия:

1) Записать необходимую информацию в регистры LMICR и GMICR. Тем самым мы задаём реальную конфигурацию системы, определяем режим и темп работы с внешней памятью. После записи в GMICR вывод *TIMER* в режиме приема больше не используется и его можно задействовать в качестве управляемого выхода, поэтому советуем этот вывод просто соединить с pull-down (1-й банк глобальной шины- статика) или pull-up резистором (1-й банк глобальной шины- динамика).
2) Установить таймеры 0 и 1 в начальное состояние.
3) Записать в регистр SP адрес начала системного стека.
4) Записью в PSWR соответствующей информации запустить таймеры, снять маски на прерывания, желательно разрешить передачу контроля над локальной и глобальной шиной. Если используется динамическая память, обязательно нужно активизировать 0-й таймер, поскольку он формирует запросы на регенерацию этой памяти.

После выполнения всего вышеперечисленного можно считать, что инициализация процессора NM6403 по внешнему прерыванию полностью закончилась.

Инициализация процессора NM6403 по коммуникационному порту 1

Если на входе *BOOTM* высокий уровень, то инициализация проводится по коммуникационному порту 1, который после действия сигнала $\overline{RESET}$ начинает работать на приём. Он получает пакет 64-разрядных двойных слов, первые пять из этих слов являются заголовком пакета, а остальные - телом пакета.

Содержимое пакета можно видеть на Рис. 12-1. Заголовок пакета выглядит следующим образом:

1-е двойное слово: старшие 32 разряда не имеют значение, младшие 32 разряда определяют информацию, которая будет записана в регистр управления интерфейсом локальной шины LMICR.

2-е двойное слово: старшие 32 разряда не имеют значение, младшие 32 разряда определяют информацию, которая будет записана в регистр управления интерфейсом глобальной шины GMICR.

3-е двойное слово: старшие 32 разряда будут записаны в таймер T1, а младшие 32 - в таймер T0.

4-е двойное слово: старшие 32 разряда будут записаны в регистр слова состояния процессора (PSWR), а младшие 32 - в счётчик команд PC.

5-е двойное слово: старшие 32 разряда будут записаны в счётчик канала ввода ICC1, а младшие 32 - в регистр адреса канала ввода ICA1.

Тело пакета может содержать системные программы, программы пользователя и обработки прерываний, массивы данных и т.д.

Запись в регистры LMICR и GMICR даёт возможность задать внешним образом конфигурацию локальной и глобальной шин, т.е. режим работы по шинам, тип внешней памяти и её характеристики.

Запись в таймер T0 позволяет сразу указать период сигналов регенерации, если используется динамическая память на одной из шин.

Запись в регистр PSWR даёт возможность запустить таймеры в нужном режиме, разрешить передачу контроля над локальной и глобальной шиной и т.д. На данном шаге не рекомендуется разрешать прерывания, поскольку не определён регистр SP - указатель системного стека.

Запись в PC позволяет задать адрес, по которому после инициализации произойдёт обращение за первой командой.

Запись в регистр ICA1 определит тот начальный адрес в памяти, куда в режиме ПДП будет писаться принимаемые в пакете данные, а запись в ICC1 даёт возможность указать, сколько двойных слов необходимо принять помимо заголовка пакета. Минимальное число принимаемых слов 1 (ICC1 = FFFFFFFF hex), а максимальное - 2^{32} (ICC1 = 00000000 hex).

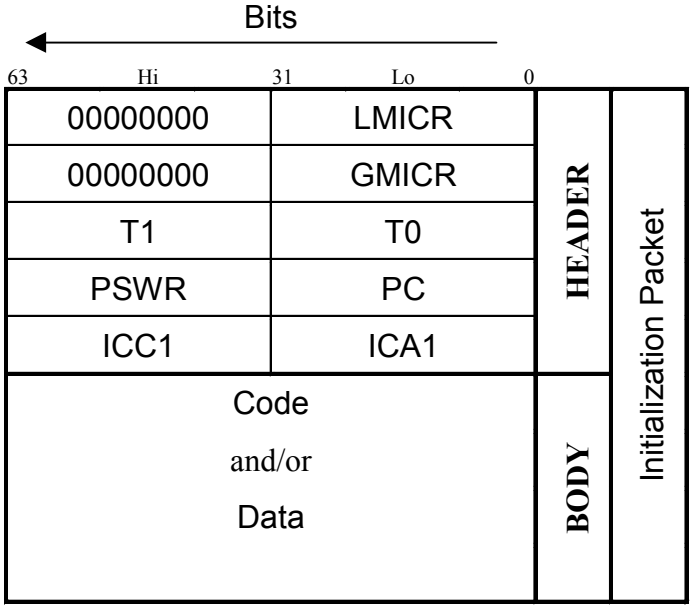
После того, как заголовок пакета принят, процессор NM6403 имеет полную информацию о своём внешнем окружении и о том, сколько информации необходимо ещё принять и куда её поместить. Далее принимаемые двойные слова (тело пакета) в режиме ПДП записываются в память. Как только пройдёт последнее ПДП, формируется запрос на прерывание по завершению ввода по порту

1. Оно не фиксируется, но по нему происходит обращение за первой командой по адресу, записанному в PC, а вывод $\overline{INTA}$ переключается с выдачи низкого на выдачу высокого уровня. Далее начинается выполнение программы инициализации, в которой в первую очередь рекомендуется задать следующие действия:

1) Записать в регистр SP адрес начала системного стека.
2) Записью в PSWR соответствующей информации разрешить прерывания.

После выполнения всего вышеперечисленного можно считать, что инициализация процессора NM6403 по коммуникационному порту 1 полностью закончилась. В заключение необходимо отметить, что для данного способа инициализации вывод *TIMER* в режиме ввода не используется, поэтому подключение его к pull-down или pull-up резистору не обязательно.

Рис. 12-1. Содержимое пакета инициализации процессора NM6403 по коммуникационному порту 1



12.2 Принципы построения многопроцессорных вычислительных систем

Основными архитектурными особенностями процессора NM6403 для построения различных многопроцессорных вычислительных систем являются наличие двух высокоскоростных двунаправленных байтовых коммуникационных портов, аппаратно совместимых с портами сигнального процессора TMS320C40, и поддержка режима работы с общей памятью. Путем объединения процессоров NM6403 различными способами можно добиться реализации большого числа высокопроизводительных параллельных систем разнообразной конфигурации. На Рис. 12-2. приведены примеры построения таких

вычислительных сетей.

Кроме вышеперечисленных возможностей можно создавать вычислительные сети практически любой конфигурации с использованием сигнального процессора TMS320C40 в качестве коммутирующего элемента. Примеры такого подхода можно увидеть на Рис. 12-3.

Используемый интерфейс с памятью позволяет строить вычислительные системы с различной архитектурой на основе процессора NM6403 (см Рис. 12-3):

- архитектура с общей памятью;
- архитектура с распределенной памятью;
- смешанная архитектура.

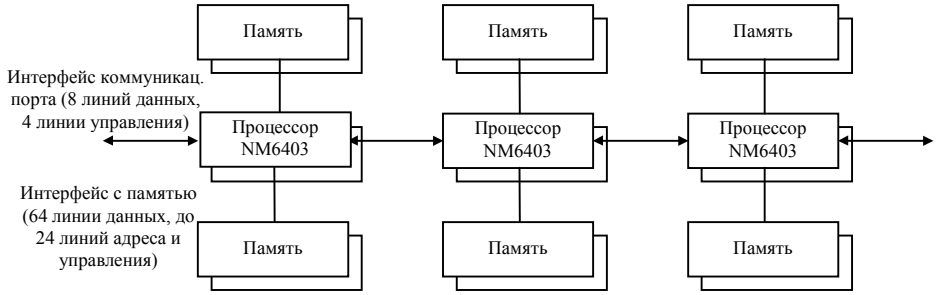
Архитектура с общей памятью характеризуется тем, что вычислительная система имеет глобальную память, которая доступна нескольким процессорам NM6403.

Архитектура с распределенной памятью подразумевает, что каждый процессор имеет свою локальную память, а взаимодействие между ними осуществляется через коммуникационные порты.

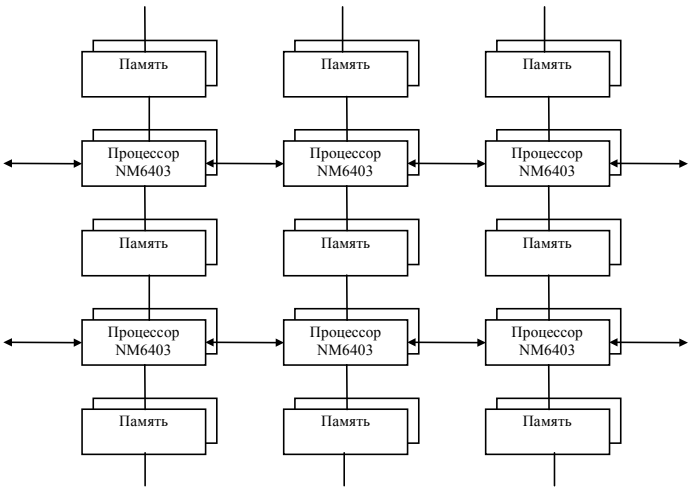
Смешанная архитектура характеризуется тем, что каждый процессор NM6403 имеет свою локальную память, а также возможность доступа к глобальной памяти вместе с другими процессорами.

При объединении небольшого количества процессоров NM6403 в вычислительную систему можно использовать архитектуру с общей глобальной памятью, но если число используемых процессоров большое, затраты на доступ в глобальную память становятся слишком велики, поэтому рекомендуется архитектура с распределенной памятью.

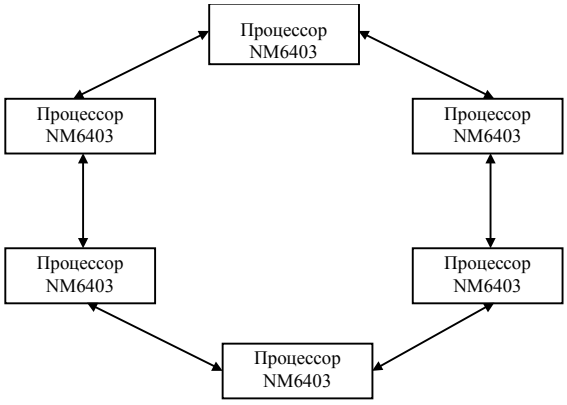
Рис. 12-2. Примеры построения вычислительных сетей на базе процессора NM6403



а) двунаправленный конвейер (для операций над матрицами, эмуляции нейросетей прямого распространения и других конвейеризированных вычислений)

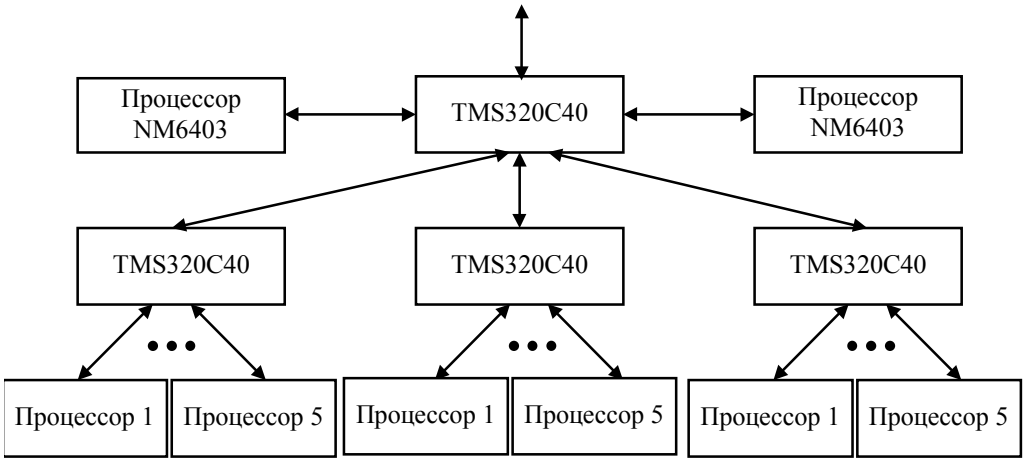


б) структура типа двумерной решетки (для операций над матрицами и эмуляции нейросетей прямого распространения

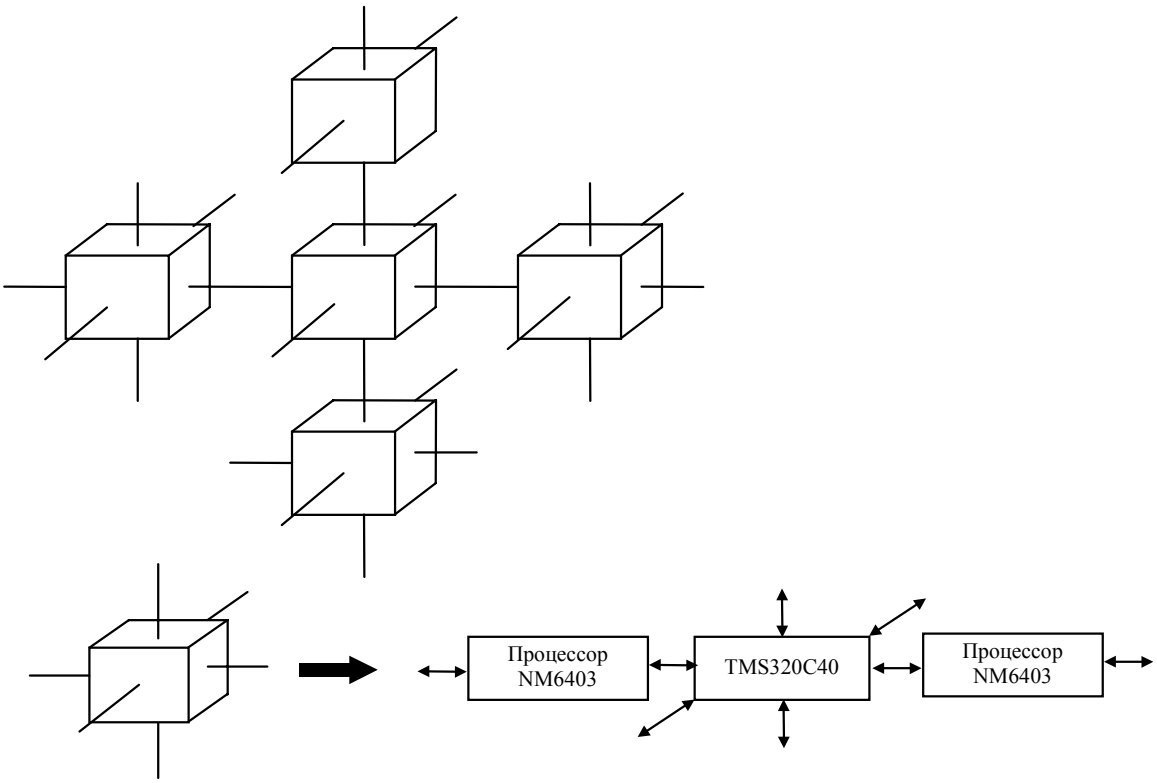


в) двунаправленное кольцо (для эмуляции различных нейросетей, в том числе с обратными связями и многослойных)

Рис. 12-3. Примеры построения вычислительных сетей на базе процессора NM6403 и TMS320C40 в качестве коммутирующего элемента

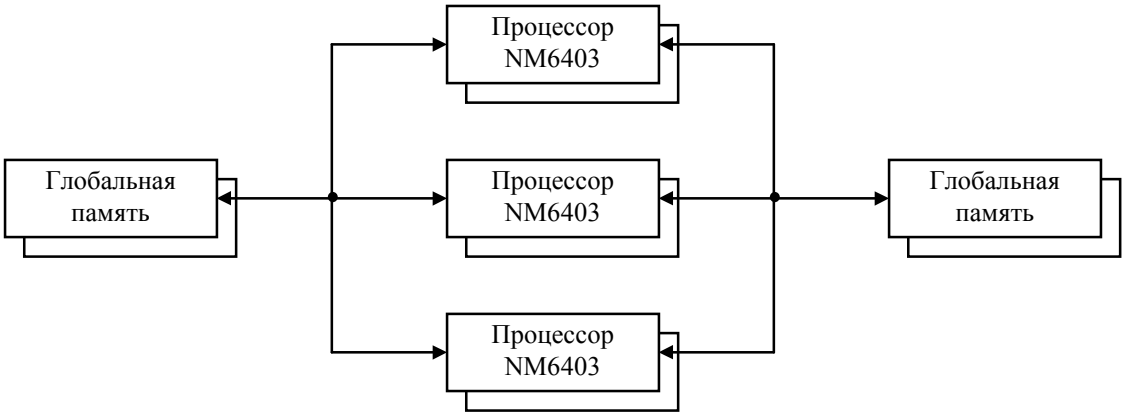


а) структура типа дерева (для эмуляции многослойных нейросетей, а также для задач распознавания образа),

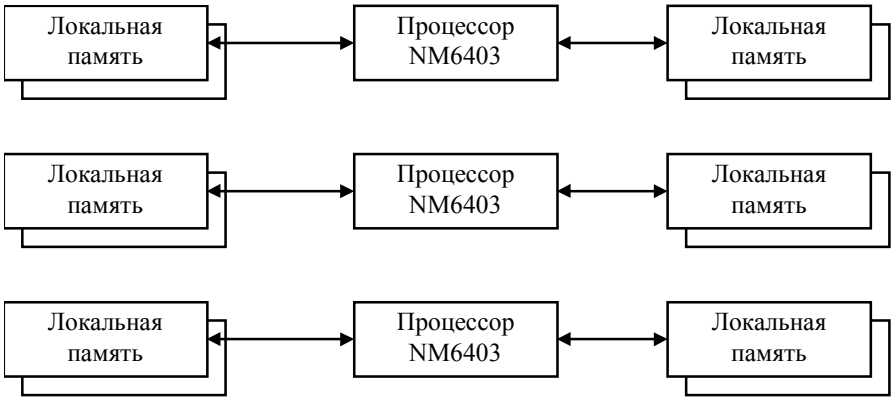


б) структура типа трехмерной решетки (для эмуляции трехмерных нейросетей, а также для задач распознавания образа).

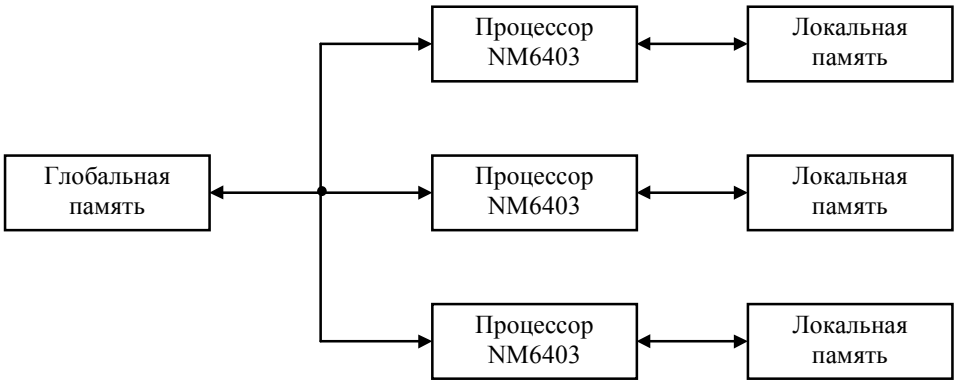
Рис. 12-4. Вычислительные системы с общей/распределенной памятью



а) архитектура с общей памятью;



б) архитектура с распределенной памятью;



в) смешанная архитектура.

13 Основные конструктивные, электрические и динамические характеристики микросхемы процессора NM6403

13.1 Состав и расположение внешних выводов микросхемы

Микросхема процессора NM6403 имеет 256-выводной корпус BGA (Ball Grid Array). Её цоколёвка показана на Рис. 13-1. Описание выводов микросхемы приведено в четырёх таблицах: Табл. 13-1, Табл. 13-2, Табл. 13-3, Табл. 13-4.

Рис. 13-1. Цоколёвка микросхемы процессора NM6403

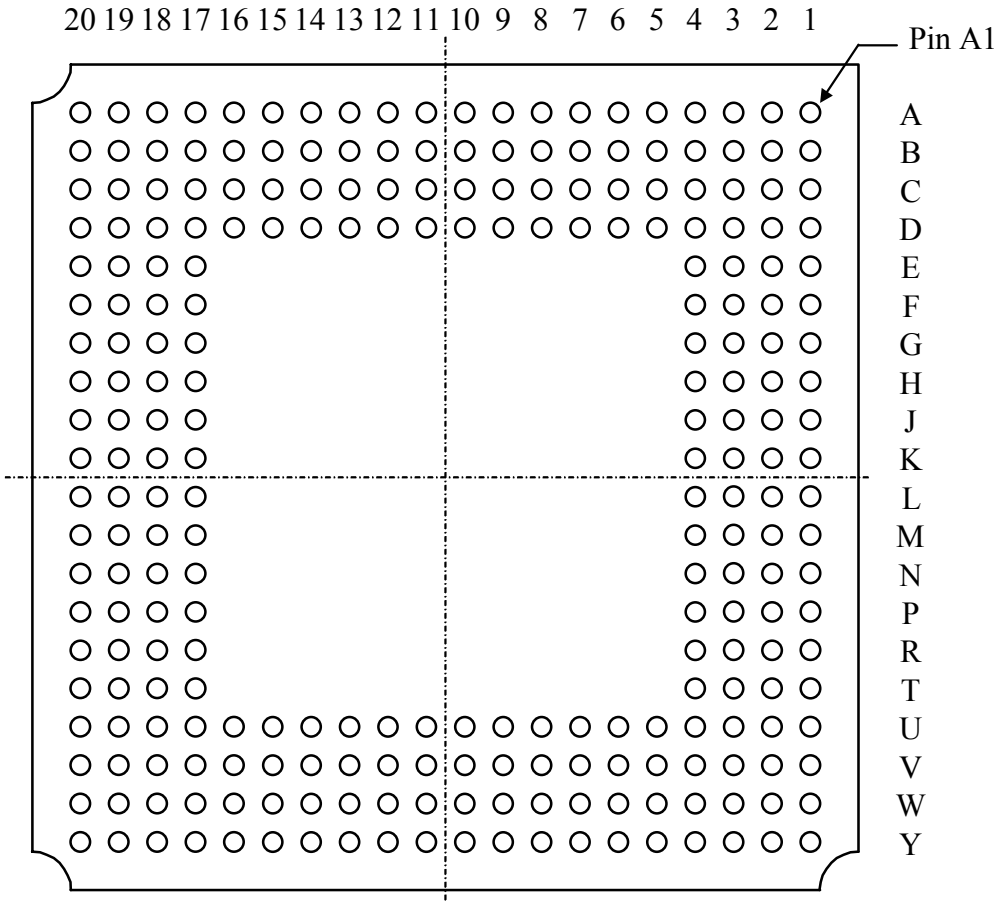


Табл. 13-1. Выводы микросхемы, отсортированные по соответствующим им именам сигналов в алфавитном порядке

Сигнал	Вывод
A1	D10
A2	C10
A3	B10
A4	A10

Сигнал	Вывод
A5	C11
A6	B11
A7	A12
A8	B12

Сигнал	Вывод
A9	C12
A10	D12
A11	A13
A12	B13

Табл. 13-1. Выводы микросхемы, отсортированные по соответствующим им именам сигналов в алфавитном порядке (Продолжение)

Сигнал	Вывод
A13	C13
A14	A14
A15	C14
BOOTM	B9
C0D0	Y6
C0D1	V7
C0D2	W8
C0D3	Y8
C0D4	U9
C0D5	V9
C0D6	W9
C0D7	Y9
C1D0	D7
C1D1	A5
C1D2	A6
C1D3	B7
C1D4	A7
C1D5	C8
C1D6	B8
C1D7	A8
$\overline{\text{CACK0}}$	U7
$\overline{\text{CACK1}}$	B5
$\overline{\text{CASH}}$ / $\overline{\text{WEH}}$	R19
$\overline{\text{CASL}}$ / $\overline{\text{WEL}}$	P17
$\overline{\text{CDIR0}}$	W5
$\overline{\text{CDIR1}}$	B4
CLK	V15
$\overline{\text{CRDY0}}$	Y5
$\overline{\text{CRDY1}}$	C5
$\overline{\text{CREQ0}}$	W6
$\overline{\text{CREQ1}}$	C6
$\overline{\text{CSTRB0}}$	V6
$\overline{\text{CSTRB1}}$	A4
D0	B16
D1	A17
D2	C17
D3	B18
D4	A20
D5	C18
D6	C19
D7	E17

Сигнал	Вывод
D8	E18
D9	E19
D10	G17
D11	F19
D12	F20
D13	G20
D14	H19
D15	J17
D16	J20
D17	K18
D18	K20
D19	L18
D20	M20
D21	M18
D22	N20
D23	N18
D24	T20
D25	T18
D26	V20
D27	U18
D28	V19
D29	Y20
D30	V18
D31	W18
D32	D5
D33	B3
D34	A2
D35	B1
D36	D2
D37	E4
D38	D1
D39	E2
D40	G4
D41	F1
D42	G2
D43	H3
D44	H1
D45	J3
D46	J1
D47	K3
D48	L2

Сигнал	Вывод
D49	L4
D50	M2
D51	M4
D52	N2
D53	P1
D54	R1
D55	R2
D56	U1
D57	U2
D58	T4
D59	V2
D60	V3
D61	Y1
D62	Y2
D63	V4
$\overline{\text{HOLDI}}$ / A18	D14
$\overline{\text{HOLDO}}$ / A17	B15
$\overline{\text{INT}}$	C9
$\overline{\text{INTA}}$	D9
LA1	W10
LA2	V10
LA3	Y10
LA4	Y11
LA5	W11
LA6	U11
LA7	Y12
LA8	W12
LA9	V12
LA10	U12
LA11	Y13
LA12	W13
LA13	V13
LA14	Y14
LA15	W14
$\overline{\text{LCASH}}$ / $\overline{\text{LWEH}}$	V16
$\overline{\text{LCASL}}$ / $\overline{\text{LWEL}}$	Y17
LD0	C16
LD1	D16
LD2	B17
LD3	A19
LD4	B19

Табл. 13-1. Выводы микросхемы, отсортированные по соответствующим им именам сигналов в алфавитном порядке (Продолжение)

Сигнал	Вывод
LD5	B20
LD6	D18
LD7	C20
LD8	D20
LD9	F18
LD10	E20
LD11	G18
LD12	G19
LD13	H18
LD14	H20
LD15	J18
LD16	K17
LD17	K19
LD18	L20
LD19	L19
LD20	M19
LD21	M17
LD22	N19
LD23	P20
LD24	T19
LD25	U20
LD26	T17
LD27	U19
LD28	W20
LD29	W19
LD30	Y19
LD31	V17
LD32	C4
LD33	B2
LD34	C3
LD35	C2
LD36	D3
LD37	C1
LD38	E3
LD39	E1
LD40	F2
LD41	G3
LD42	G1
LD43	H2
LD44	J4
LD45	J2

Сигнал	Вывод
LD46	K2
LD47	K1
LD48	L3
LD49	M1
LD50	M3
LD51	N1
LD52	N3
LD53	P2
LD54	P3
LD55	T1
LD56	T3
LD57	V1
LD58	U3
LD59	W1
LD60	W2
LD61	W3
LD62	W4
LD63	U5
$\overline{\text{LHOLDI}}$/LA18	Y16
$\overline{\text{LHOLDO}}$/LA17	W15
$\overline{\text{LOE}}$	W16
$\overline{\text{LRAS0}}$/$\overline{\text{LCS0}}$	W17
$\overline{\text{LRAS1}}$/$\overline{\text{LCS1}}$	Y18
$\overline{\text{LRDY}}$/LA19	U14
$\overline{\text{LWE}}$/LA16	V14
MVOE	A9
$\overline{\text{OE}}$	R18
NC	A3
NC	A11
NC	A16
NC	A18
NC	B6
NC	B14
NC	D19
NC	F3
NC	J19
NC	L1
NC	P19
NC	T2
NC	U16
NC	V5

Сигнал	Вывод
NC	V8
NC	V11
NC	Y7
NC	Y15
$\overline{\text{OE}}$	R18
$\overline{\text{RAS0}}$/$\overline{\text{CS0}}$	R20
$\overline{\text{RAS1}}$/$\overline{\text{CS1}}$	P18
$\overline{\text{RDY}}$/A19	C15
$\overline{\text{RESET}}$	R3
TEST_OUT1	Y3
TEST_OUT2	Y4
TIMER	P4
VBB	C7
VBB	W7
VDD	D6
VDD	D11
VDD	D15
VDD	F4
VDD	F17
VDD	K4
VDD	L17
VDD	R4
VDD	R17
VDD	U6
VDD	U10
VDD	U15
VSS	A1
VSS	D4
VSS	D8
VSS	D13
VSS	D17
VSS	H4
VSS	H17
VSS	N4
VSS	N17
VSS	U4
VSS	U8
VSS	U13
VSS	U17
$\overline{\text{WE}}$/A16	A15

Табл. 13-2. Выводы микросхемы, отсортированные по их номерам

Вывод	Сигнал
A1	VSS
A2	D34
A3	NC
A4	$\overline{\text{CSTRB1}}$
A5	C1D1
A6	C1D2
A7	C1D4
A8	C1D7
A9	MVOE
A10	A4
A11	NC
A12	A7
A13	A11
A14	A14
A15	$\overline{\text{WE}}$/A16
A16	NC
A17	D1
A18	NC
A19	LD3
A20	D4
B1	D35
B2	LD33
B3	D33
B4	$\overline{\text{CDIR1}}$
B5	$\overline{\text{CACK1}}$
B6	NC
B7	C1D3
B8	C1D6
B9	BOOTM
B10	A3
B11	A6
B12	A8
B13	A12
B14	NC
B15	$\overline{\text{HOLDO}}$/A17
B16	D0
B17	LD2
B18	D3
B19	LD4
B20	LD5

Вывод	Сигнал
C1	LD37
C2	LD35
C3	LD34
C4	LD32
C5	$\overline{\text{CRDY1}}$
C6	$\overline{\text{CREQ1}}$
C7	VBB
C8	C1D5
C9	$\overline{\text{INT}}$
C10	A2
C11	A5
C12	A9
C13	A13
C14	A15
C15	$\overline{\text{RDY}}$/A19
C16	LD0
C17	D2
C18	D5
C19	D6
C20	LD7
D1	D38
D2	D36
D3	LD36
D4	VSS
D5	D32
D6	VDD
D7	C1D0
D8	VSS
D9	$\overline{\text{INTA}}$
D10	A1
D11	VDD
D12	A10
D13	VSS
D14	$\overline{\text{HOLDI}}$/A18
D15	VDD
D16	LD1
D17	VSS
D18	LD6
D19	NC
D20	LD8

Вывод	Сигнал
E1	LD39
E2	D39
E3	LD38
E4	D37
E17	D7
E18	D8
E19	D9
E20	LD10
F1	D41
F2	LD40
F3	NC
F4	VDD
F17	VDD
F18	LD9
F19	D11
F20	D12
G1	LD42
G2	D42
G3	LD41
G4	D40
G17	D10
G18	LD11
G19	LD12
G20	D13
H1	D44
H2	LD43
H3	D43
H4	VSS
H17	VSS
H18	LD13
H19	D14
H20	LD14
J1	D46
J2	LD45
J3	D45
J4	LD44
J17	D15
J18	LD15
J19	NC
J20	D16

Табл. 13-2. Выводы микросхемы, отсортированные по их номерам(Продолжение)

Вывод	Сигнал
K1	LD47
K2	LD46
K3	D47
K4	VDD
K17	LD16
K18	D17
K19	LD17
K20	D18
L1	NC
L2	D48
L3	LD48
L4	D49
L17	VDD
L18	D19
L19	LD19
L20	LD18
M1	LD49
M2	D50
M3	LD50
M4	D51
M17	LD21
M18	D21
M19	LD20
M20	D20
N1	LD51
N2	D52
N3	LD52
N4	VSS
N17	VSS
N18	D23
N19	LD22
N20	D22
P1	D53
P2	LD53
P3	LD54
P4	TIMER
P17	$\overline{\text{CASL}}$ / $\overline{\text{WEL}}$
P18	$\overline{\text{RAS1}}$ / $\overline{\text{CS1}}$
P19	NC
P20	LD23

Вывод	Сигнал
R1	D54
R2	D55
R3	$\overline{\text{RESET}}$
R4	VDD
R17	VDD
R18	$\overline{\text{OE}}$
R19	$\overline{\text{CASH}}$ / $\overline{\text{WEH}}$
R20	$\overline{\text{RAS0}}$ / $\overline{\text{CS0}}$
T1	LD55
T2	NC
T3	LD56
T4	D58
T17	LD26
T18	D25
T19	LD24
T20	D24
U1	D56
U2	D57
U3	LD58
U4	VSS
U5	LD63
U6	VDD
U7	$\overline{\text{CACK0}}$
U8	VSS
U9	C0D4
U10	VDD
U11	LA6
U12	LA10
U13	VSS
U14	$\overline{\text{LRDY}}$ /LA19
U15	VDD
U16	NC
U17	VSS
U18	D27
U19	LD27
U20	LD25

Вывод	Сигнал
V1	LD57
V2	D59
V3	D60
V4	D63
V5	NC
V6	$\overline{\text{CSTRB0}}$
V7	C0D1
V8	NC
V9	C0D5
V10	LA2
V11	NC
V12	LA9
V13	LA13
V14	$\overline{\text{LWE}}$ /LA16
V15	CLK
V16	$\overline{\text{LCASH}}$ / $\overline{\text{LWEH}}$
V17	LD31
V18	D30
V19	D28
V20	D26
W1	LD59
W2	LD60
W3	LD61
W4	LD62
W5	$\overline{\text{CDIR0}}$
W6	$\overline{\text{CREQ0}}$
W7	VBB
W8	C0D2
W9	C0D6
W10	LA1
W11	LA5
W12	LA8
W13	LA12
W14	LA15
W15	$\overline{\text{LHOLDO}}$ /LA17
W16	$\overline{\text{LOE}}$
W17	$\overline{\text{LRAS0}}$ / $\overline{\text{LCS0}}$
W18	D31
W19	LD29
W20	LD28

Табл. 13-2. Выводы микросхемы, отсортированные по их номерам (Продолжение)

Вывод	Сигнал
Y1	D61
Y2	D62
Y3	TEST_OUT1
Y4	TEST_OUT2
Y5	$\overline{\text{CRDY0}}$
Y6	C0D0
Y7	NC
Y8	C0D3
Y9	C0D7
Y10	LA3
Y11	LA4
Y12	LA7
Y13	LA11
Y14	LA14
Y15	NC
Y16	$\overline{\text{LHOLDI}}$/LA18
Y17	$\overline{\text{LCASL}}$/$\overline{\text{LWEL}}$
Y18	$\overline{\text{LRAS1}}$/$\overline{\text{LCS1}}$
Y19	LD30
Y20	D29

13.2 Функциональное назначение выводов

Функциональное назначение внешних выводов микросхемы процессора NM6403 приведено в Табл. 13-3. Данная таблица содержит для каждого вывода или группы выводов их тип (вход, выход или двунаправленный) и количество в группе. Знак инверсии в имени сигнала обозначает, что для него активным является низкий уровень. Сигналы группируются по функциональному признаку.

Табл. 13-3. Выводы процессора NM6403 и их функциональное назначение

Сигнал[1)]	Кол-во	Тип вывода[2)]	Функциональное назначение
Интерфейс с глобальной шиной (88 выводов)			
D63 - D0	64	I/O(Z)	64-разрядная шина данных
A15 - A1	15	O(Z)	15-разрядная шина адреса
$\overline{\text{RAS0}}$/$\overline{\text{CS0}}$	1	O(Z)	строб адреса строки 0-го банка динамического ОЗУ /строб адреса страницы 0-го банка статического ОЗУ
$\overline{\text{RAS1}}$/$\overline{\text{CS1}}$	1	O(Z)	строб адреса строки 1-го банка динамического ОЗУ /строб адреса страницы 1-го банка статического ОЗУ

Табл. 13-3. Выводы процессора NM6403 и их функциональное назначение (Продолжение)

Сигнал[1]	Кол-во	Тип вывода[2]	Функциональное назначение
$\overline{\text{CASL}}$ / $\overline{\text{WEL}}$	1	O(Z)	строб адреса столбца 32 младших разрядов динамического ОЗУ / разрешение записи в 32 младших разряда статического ОЗУ
$\overline{\text{CASH}}$ / $\overline{\text{WEH}}$	1	O(Z)	строб адреса столбца 32 старших разрядов динамического ОЗУ / разрешение записи в 32 старших разряда статического ОЗУ
$\overline{\text{OE}}$	1	O(Z)	разрешение выдачи данных из ОЗУ
$\overline{\text{WE}}$ /A16	1	O(Z)	признак цикла (запись/чтение) для динамического ОЗУ /16-й разряд шины адреса
$\overline{\text{HOLDO}}$ /A17[3]	1	O(Z)	запрос на владение шиной процессором /17-й разряд шины адреса
$\overline{\text{HOLDI}}$ /A18[3]	1	I/O(Z)	запрос на захват шины извне/18-й разряд шины адреса
$\overline{\text{RDY}}$ /A19[3]	1	I/O(Z)	сигнал готовности/19-й разряд шины адреса
Интерфейс с локальной шиной (88 выводов)			
LD63 - LD0	64	I/O(Z)	64-разрядная шина данных
LA15 - LA1	15	O(Z)	15-разрядная шина адреса
$\overline{\text{LRAS0}}$ / $\overline{\text{LCS0}}$	1	O(Z)	строб адреса строки 0-го банка динамического ОЗУ /строб адреса страницы 0-го банка статического ОЗУ
$\overline{\text{LRAS1}}$ / $\overline{\text{LCS1}}$	1	O(Z)	строб адреса строки 1-го банка динамического ОЗУ /строб адреса страницы 1-го банка статического ОЗУ
$\overline{\text{LCASL}}$ / $\overline{\text{LWEL}}$	1	O(Z)	строб адреса столбца 32 младших разрядов динамического ОЗУ / разрешение записи в 32 младших разряда статического ОЗУ
$\overline{\text{LCASH}}$ / $\overline{\text{LWEH}}$	1	O(Z)	строб адреса столбца 32 старших разрядов динамического ОЗУ / разрешение записи в 32 старших разряда статического ОЗУ
$\overline{\text{LOE}}$	1	O(Z)	разрешение выдачи данных из ОЗУ
$\overline{\text{LWE}}$ /LA16	1	O(Z)	признак цикла (запись/чтение) для динамического ОЗУ /16-й разряд шины адреса
$\overline{\text{LHOLDO}}$ /LA17[3]	1	O(Z)	запрос на владение шиной процессором /17-й разряд шины адреса
$\overline{\text{LHOLDI}}$ /LA18[3]	1	I/O(Z)	запрос на захват шины извне/18-й разряд шины адреса
$\overline{\text{LRDY}}$ /LA19[3]	1	I/O(Z)	сигнал готовности/19-й разряд шины адреса
Коммуникационный порт 0 (13 выводов)			
C0D7 - C0D0	8	I/O(Z)	шина данных
$\overline{\text{CREQ0}}$	1	I/O(Z)	запрос шины
$\overline{\text{CACK0}}$	1	I/O(Z)	подтверждение запроса шины
$\overline{\text{CSTRB0}}$	1	I/O(Z)	строб данных
$\overline{\text{CRDY0}}$	1	I/O(Z)	сигнал готовности данных
$\overline{\text{CDIR0}}$	1	O	направление передачи данных

Табл. 13-3. Выводы процессора NM6403 и их функциональное назначение (Продолжение)

Сигнал[1)]	Кол-во	Тип выво-да[2)]	Функциональное назначение
Коммуникационный порт 1 (13 выводов)			
C1D7 - C1D0	8	I/O(Z)	шина данных
$\overline{\text{CREQ1}}$	1	I/O(Z)	запрос шины
$\overline{\text{CACK1}}$	1	I/O(Z)	подтверждение запроса шины
$\overline{\text{CSTRB1}}$	1	I/O(Z)	строб данных
$\overline{\text{CRDY1}}$	1	I/O(Z)	сигнал готовности данных
$\overline{\text{CDIR1}}$	1	O	направление передачи данных
Общее управление (9 выводов)			
CLK	1	I	тактовый сигнал
$\overline{\text{RESET}}$	1	I	сброс
TIMER	1	I/O(Z)	выход таймера
$\overline{\text{INT}}$	1	I	внешнее прерывание
$\overline{\text{INTA}}$	1	O	подтверждение внешнего прерывания
BOOTM	1	I	режим начальной загрузки
MVOE	1	I	разрешение выдачи данных на внешнюю шину при внутренних пересылках из регистра в регистр в процессоре
TEST_OUT1	1	O	тестовый вывод (используется изготовителем микросхемы)
TEST_OUT2	1	O	тестовый вывод (используется изготовителем микросхемы)
Питание (27 выводов)			
VSS	13	I	общий
VDD	12	I	питание (3.3 В)
VBB	2	I	напряжение смещения для толерантных выводов (3.3 В или 5 В)
Неиспользуемые выводы (18 выводов)			
NC	18	-	неиспользуемый вывод

Примечание:

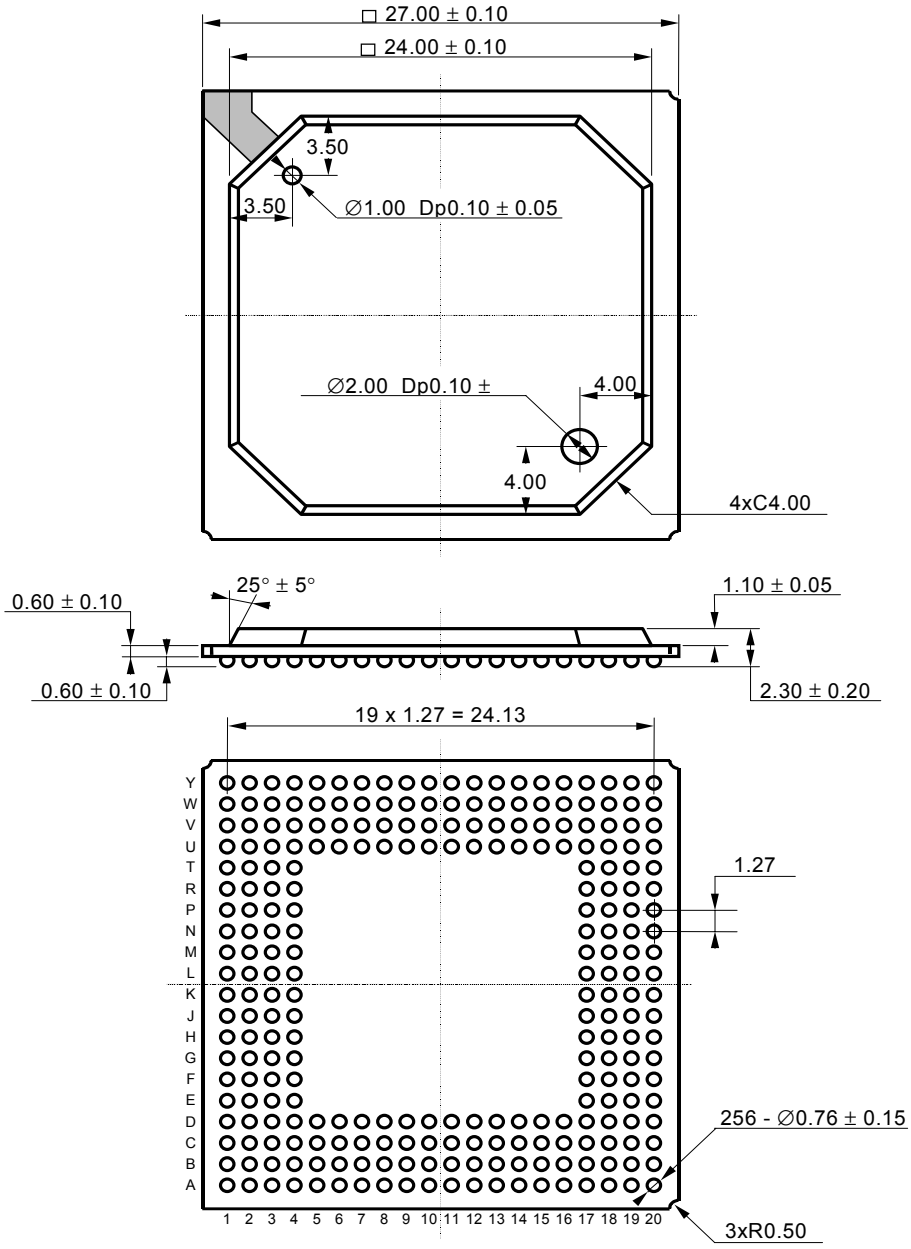
1) Для выводов со знаком инверсии активным является низкий уровень сигнала;
2) I- входы; O - выходы;O(Z) - выходы с высокоимпедансным состоянием;
I/O(Z) - двунаправленные выводы;
3) Данные выводы при многопоцессорном режиме служат при необходимости доступа к общей памяти для арбитража, а при однопоцессорном режиме в качестве дополнительных разрядов адреса.

13.3 Конструктивные характеристики микросхемы

Внешний вид корпуса микросхемы процессора NM6403(256-Pin BGA) можно увидеть на Рис. 13-2. Данный рисунок также содержит

сведения о габаритных, установочных и присоединительных размерах микросхемы.

Рис. 13-2. Внешний вид корпуса микросхемы процессора NM6403 (256-Pin BGA)



13.4 Электрические характеристики

Данный раздел содержит сведения о электрических характеристиках микросхемы процессора NM6403. Он включает в себя следующие сведения:

- характеристики буферов ввода/вывода для каждого внешнего вывода микросхемы (см. Табл. 13-4);
- предельно допустимые электрические и температурные режимы работы микросхемы (см. Табл. 13-5);

рекомендуемые условия эксплуатации микросхемы (см. Табл. 13-6);

- электрические характеристики микросхемы (см. Табл. 13-7).

Табл. 13-4. Выводы микросхемы с указанием характеристик соответствующим им буферов ввода/вывода

Сигнал	Кол-во	Тип вывода[1)]	Толерантный вход[2)]	Pull-up	Pull-down	Выходной ток, мА	Шинный выход
Интерфейс с глобальной шиной (88 выводов)							
D63 - D0	64	I/O(Z)				4	+
A15 - A1	15	O(Z)				8	+
$\overline{\text{RAS0}}$/$\overline{\text{CS0}}$	1	O(Z)				8	
$\overline{\text{RAS1}}$/$\overline{\text{CS1}}$	1	O(Z)				8	
$\overline{\text{CASL}}$/$\overline{\text{WEL}}$	1	O(Z)				8	
$\overline{\text{CASH}}$/$\overline{\text{WEH}}$	1	O(Z)				8	
$\overline{\text{OE}}$	1	O(Z)				12	
$\overline{\text{WE}}$/A16	1	O(Z)				8	+
$\overline{\text{HOLDO}}$/A17	1	O(Z)				8	+
$\overline{\text{HOLDI}}$/A18	1	I/O(Z)				8	+
$\overline{\text{RDY}}$/A19	1	I/O(Z)		+		8	+
Интерфейс с локальной шиной (88 выводов)							
LD63 - LD0	64	I/O(Z)				4	+
LA15 - LA1	15	O(Z)				8	+
$\overline{\text{LRAS0}}$/$\overline{\text{LCS0}}$	1	O(Z)				8	
$\overline{\text{LRAS1}}$/$\overline{\text{LCS1}}$	1	O(Z)				8	
$\overline{\text{LCASL}}$/$\overline{\text{LWEL}}$	1	O(Z)				8	
$\overline{\text{LCASH}}$/$\overline{\text{LWEH}}$	1	O(Z)				8	
$\overline{\text{LOE}}$	1	O(Z)				12	
$\overline{\text{LWE}}$/LA16	1	O(Z)				8	+
$\overline{\text{LHOLDO}}$/LA17	1	O(Z)				8	+
$\overline{\text{LHOLDI}}$/LA18	1	I/O(Z)				8	+
$\overline{\text{LRDY}}$/LA19	1	I/O(Z)		+		8	+
Коммуникационный порт 0 (13 выводов)							
C0D7 - C0D0	8	I/O(Z)	+			6	
$\overline{\text{CREQ0}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CACK0}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CSTRB0}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CRDY0}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CDIR0}}$	1	O				4	

Табл. 13-4. Выводы микросхемы с указанием характеристик соответствующим им буферов ввода/вывода (Продолжение)

Сигнал	Кол-во	Тип вывода[1)]	Толерантный вход[2)]	Pull-up	Pull-down	Выходной ток, мА	Шинный выход
Коммуникационный порт 1 (13 выводов)							
C1D7 – C1D0	8	I/O(Z)	+			6	
$\overline{\text{CREQ1}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CACK1}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CSTRB1}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CRDY1}}$	1	I/O(Z)	+	+		6	
$\overline{\text{CDIR1}}$	1	O				4	
Общее управление (9 выводов)							
CLK	1	I					
$\overline{\text{RESET}}$	1	I					
TIMER	1	I/O(Z)				4	
$\overline{\text{INT}}$	1	I					
$\overline{\text{INTA}}$	1	O				4	
BOOTM	1	I					
MVOE	1	I			+		
TEST_OUT1	1	O				2	
TEST_OUT2	1	O				2	
Питание (27 выводов)							
VSS	13	I					
VDD	12	I					
VBB	2	I					

Примечание:

I- входы;
O – выходы;
O(Z) – выходы с высокоимпедансным состоянием;
I/O(Z) – двунаправленные выводы;
2) Все толерантные входы могут работать в одном из двух режимов: с входными сигналами либо в диапазоне от –0,3 до 3,6 В, либо от –0,3 до 5,3 В в зависимости от напряжения смещения для этих входов – соответственно 3,3 или 5,0 В, которое подаётся на два вывода VBB (одно и то же на оба вывода).

Табл. 13-5. Предельно допустимые электрические и температурные режимы работы

Обозначение	Условие / Характеристика	Диапазон значений		Единица измерения
		не менее	не более	
V_{DD}	Напряжение питания	-0.3	4.6	В
V_{BB}	Напряжение смещения уровня логической единицы толерантных входов	-0.3	6.0	В
V_{IN}	Напряжение на стандартных входах	-0.3	V_{DD}+0.3	В
V_{INBB}	Напряжение на толерантных входах	-0.3	V_{BB}+0.3	В
I_{IN}	Входной ток	-10	10	мА
T_{STG}	Температура хранения	-40	125	°С

Примечание: *1) Внешние воздействия, значения которых выходят за пределы указанных диапазонов, могут приводить к выходу из строя микросхемы;*
2) Значения всех напряжений приведены относительно земли.

Табл. 13-6. Рекомендуемые условия эксплуатации

Обозначение	Параметр	Диапазон значений		Единица измерения
		не менее	не более	
V_{DD}	Напряжение питания	3.0	3.6	В
V_{BB5}	Напряжение 5В-смещения толерантных входов	4.75	5.25	В
V_{BB3}	Напряжение 3В-смещения толерантных входов	3.15	3.45	В
V_{IH}	Напряжение высокого уровня на стандартном входе[3)]	0.7V_{DD}	V_{DD}+0.3	В
V_{IHBB}	Напряжение высокого уровня на толерантном входе[4)]	2.1	V_{BB}+0.3	В
V_{IL}	Напряжение низкого уровня на стандартном входе[3)]	-0.3	0.3 V_{DD}	В
V_{ILBB}	Напряжение низкого уровня на толерантном входе[4)]	-0.3	0.8	В
T_{CASE}	Температура на поверхности корпуса	-40	85	°С

Примечание: *1) Внешние воздействия, значения которых выходят за пределы указанных диапазонов, могут приводить к сбоям в работе микросхемы;*
2) Значения всех напряжений приведены относительно земли;
3) Применимо к входам и двунаправленным выводам без толерантного входа;
4) Применимо к двунаправленным выводам с толерантным входом.

Табл. 13-7. Электрические характеристики при V_{DD} = 3.3В± 10 %, V_{BB}= 5.0В± 5%, T_{CASE} = 0-70°C

Обозначение	Характеристика	Условия измерения	Значения			Единица измерения
			не менее	номинал	не более	
V_{OH}	Напряжение высокого уровня на любом выходе[1)]	I_{OH} = -1 мкА	V_{DD}-0.05			В
V_{OH2}	Напряжение высокого уровня на 2 мА выходе[2)]	V_{DD} = 3.0 В, I_{OH} = -2 мА	2.4			В
V_{OH4}	Напряжение высокого уровня на 4 мА выходе[3)]	V_{DD} = 3.0 В, I_{OH} = -4 мА	2.4			В
V_{OH6}	Напряжение высокого уровня на 6 мА выходе[4)]	V_{DD} = 3.0 В, I_{OH} = -6 мА	2.4			В
V_{OH8}	Напряжение высокого уровня на 8 мА выходе[5)]	V_{DD} = 3.0 В, I_{OH} = -8 мА	2.4			В
V_{OH12}	Напряжение высокого уровня на 12 мА выходе[6)]	V_{DD} = 3.0 В, I_{OH} = -12 мА	2.4			В
V_{OL}	Напряжение низкого уровня на любом выходе[1)]	V_{DD} = 3.6 В, I_{OH} = 1 мкА			0.05	В
V_{OL2}	Напряжение низкого уровня на 2 мА выходе[2)]	V_{DD} = 3.6 В, I_{OH} = 2 мА			0.4	В
V_{OL4}	Напряжение низкого уровня на 4 мА выходе[3)]	V_{DD} = 3.6 В, I_{OH} = 4 мА			0.4	В
V_{OL6}	Напряжение низкого уровня на 6 мА выходе[4)]	V_{DD} = 3.6 В, I_{OH} = 6 мА			0.4	В
V_{OL8}	Напряжение низкого уровня на 8 мА выходе[5)]	V_{DD} = 3.6 В, I_{OH} = 8 мА			0.4	В
V_{OL12}	Напряжение низкого уровня на 12 мА выходе[6)]	V_{DD} = 3.6 В, I_{OH} = 12 мА			0.4	В
I_{IH}	Ток высокого уровня на входе[7)]	V_{IN} = V_{DD}	-10		10	мкА
I_{IHPD}	Ток высокого уровня на входе с pull-down резистором[8)]	V_{IN} = V_{DD}	10		200	мкА
I_{IH5}	Ток высокого уровня на толерантном входе[9)]	V_{IN} = 5V, V_{BB} = 5V	-10		10	мкА
I_{IL}	Ток низкого уровня на входе[10)]	V_{IN} = V_{SS}	-10		10	мкА
I_{ILPU}	Ток низкого уровня на входе с pull-up резистором[11)]	V_{IN} = V_{SS}	-200		-10	мкА

Табл. 13-7. Электрические характеристики при V_{DD} = 3.3В± 10 %, V_{BB}= 5.0В± 5%, T_{CASE} = 0-70°C (Продолжение)

Обозначение	Характеристика	Условия измерения	Значения			Единица измерения
			не менее	номинал	не более	
I_{OZ}	Ток утечки выхода, находящегося в высокоимпедансном состоянии	$V_{OUT} = V_{SS}$ или V_{DD}	-10		10	мкА
I_{DD}	Ток потребления динамический	T_{CASE} = 25°C, V_{DD} = 3.6 В, f_{CLK} =40МГц			300	мА
I_{DDI}	Ток потребления статический	T_{CASE} = 25°C, V_{DD} = 3.6 В, f_{CLK} =40МГц			95	мА
I_{DDI}	Ток потребления статический	T_{CASE} = 25°C, V_{DD} = 3.6 В, f_{CLK} = 0 МГц			60	мА
R_{PU}	Сопротивление pull-up			100		КОм
R_{PD}	Сопротивление pull-down			100		КОм
C_{IN}	Емкость входа				10	пФ

Примечание:

1) Применимо ко всем выходам и двунаправленным выводам;
2) Применимо к 2mA выходам;
3) Применимо к 4mA выходам и двунаправленным выводам;
4) Применимо к 6mA двунаправленным выводам;
5) Применимо к 8mA выходам и двунаправленным выводам;
6) Применимо к 12mA выходам;
7) Применимо ко всем входам, за исключением MVOE, и нетолерантным двунаправленным выводам;
8) Применимо ко входу с pull-down резистором – MVOE;
9) Применимо ко всем толерантным двунаправленным выводам;
10) Применимо ко всем входам и двунаправленным выводам без pull-up резистора;
11) Применимо ко всем двунаправленным выводам с pull-up резистором;
12) f_{CLK} - тактовая частота работы процессора;
13) V_{SS} - напряжение на выводах земли.

13.5 Временные характеристики

В данном разделе приведены временные диаграммы и временные параметры для сигналов процессора NM6403. Временные диаграммы представлены на Рис. 13-3 - Рис. 13-21, причём к

каждому рисунку прилагается таблица (см. Табл. 13-8 - Табл. 13-18) с соответствующими временными характеристиками. Эти характеристики получены при моделировании процессора с учётом реальной топологии для следующих условий: V_{DD}=3.0 В, T_{CASE}=85°C, $C_{нагр.}$=75 пФ.

Рис. 13-3. Временная диаграмма тактового сигнала CLK

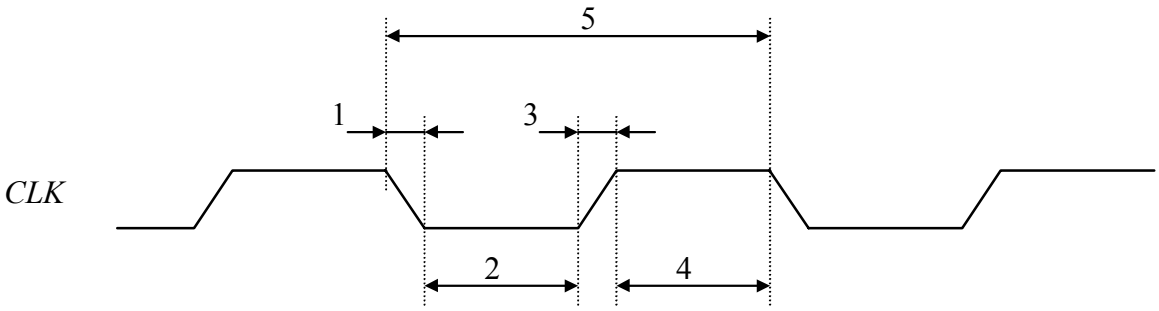
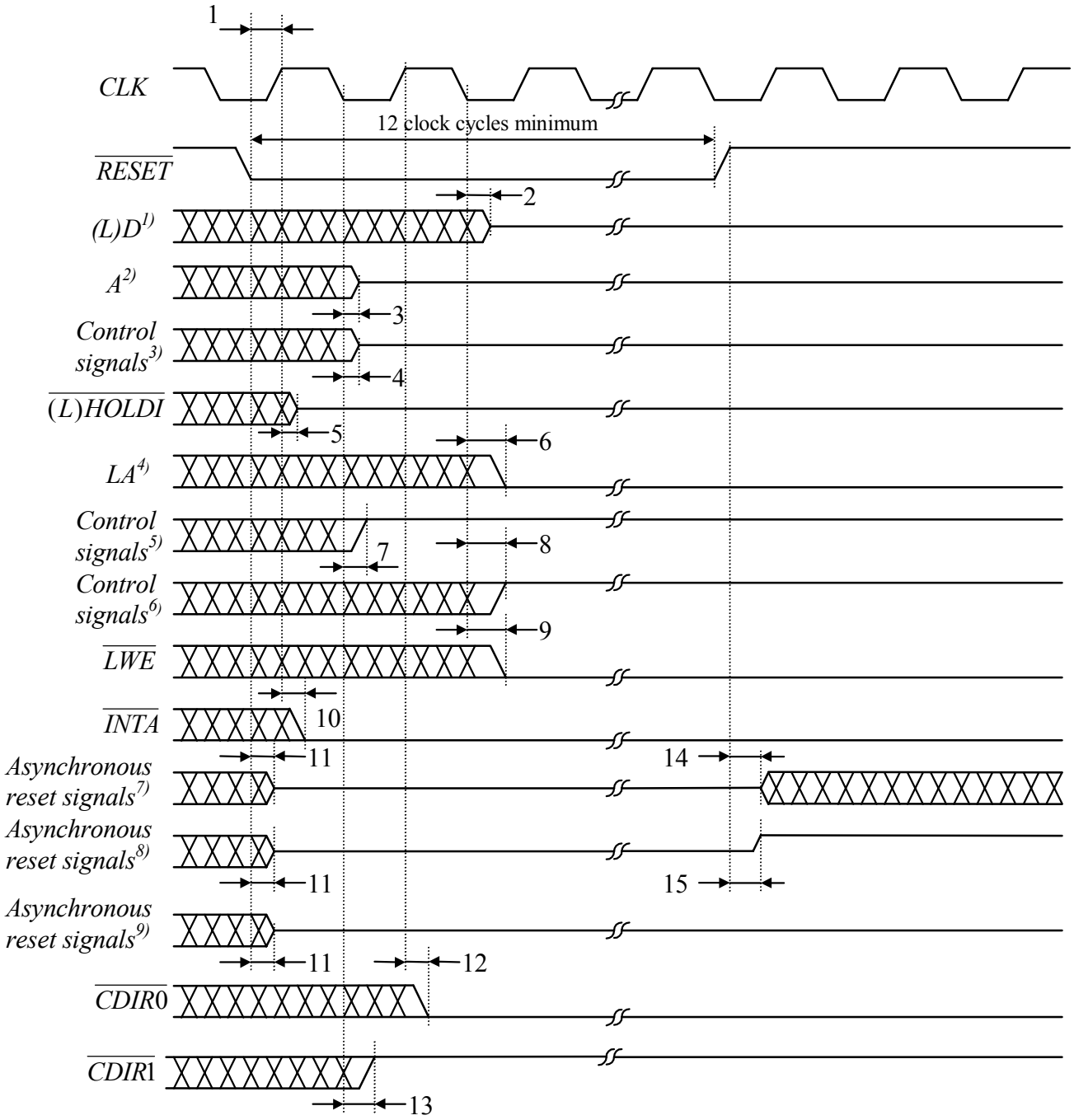


Табл. 13-8. Временные параметры для тактового сигнала CLK

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{f(CLK)}$	Длительность заднего фронта *CLK*		5
2.	$t_{w(CLKL)}$	Длительность импульса низкого уровня *CLK*	12.5	
3.	$t_{r(CLK)}$	Длительность переднего фронта *CLK*		5
4.	$t_{w(CLKH)}$	Длительность импульса высокого уровня *CLK*	12.5	
5.	$t_{c(CLK)}$	Период *CLK*	25	

Рис. 13-4. Временная диаграмма системного сброса



Примечание:

1) Обозначение (L)D относится к шинам D(63-0) и LD(63-0);
2) Обозначение A относится к адресной шине A(15-1);
3) Имеются в виду сигналы $\overline{RAS0}$, $\overline{RAS1}$, $\overline{CASL}$, $\overline{CASH}$, $\overline{WE}$, $\overline{OE}$, $\overline{(L)RDY}$ *uTIMER;*
4) Обозначение LA относится к адресной шине LA(15-1);
5) Имеются в виду сигналы $\overline{LRAS0}$, $\overline{LRAS1}$, $\overline{LCASL}$, $\overline{LCASH}$;
6) Имеются в виду сигналы $\overline{LOE}$ *и* $\overline{(L)HOLDO}$;
7) Имеется в виду шина данных коммуникационного порта 0 – C0D(7-0);

8) Подразумеваются сигналы управления коммуникационных портов 0 и 1: $\overline{CREQ1}$, $\overline{CACK0}$, $\overline{CSTRB0}$ и $\overline{CRDY1}$;
9) Имеются в виду шина данных коммуникационного порта 1 – C1D(7-0) и сигналы управления коммуникационных портов 0 и 1: $\overline{CREQ0}$, $\overline{CACK1}$, $\overline{CSTRB1}$ и $\overline{CRDY0}$.

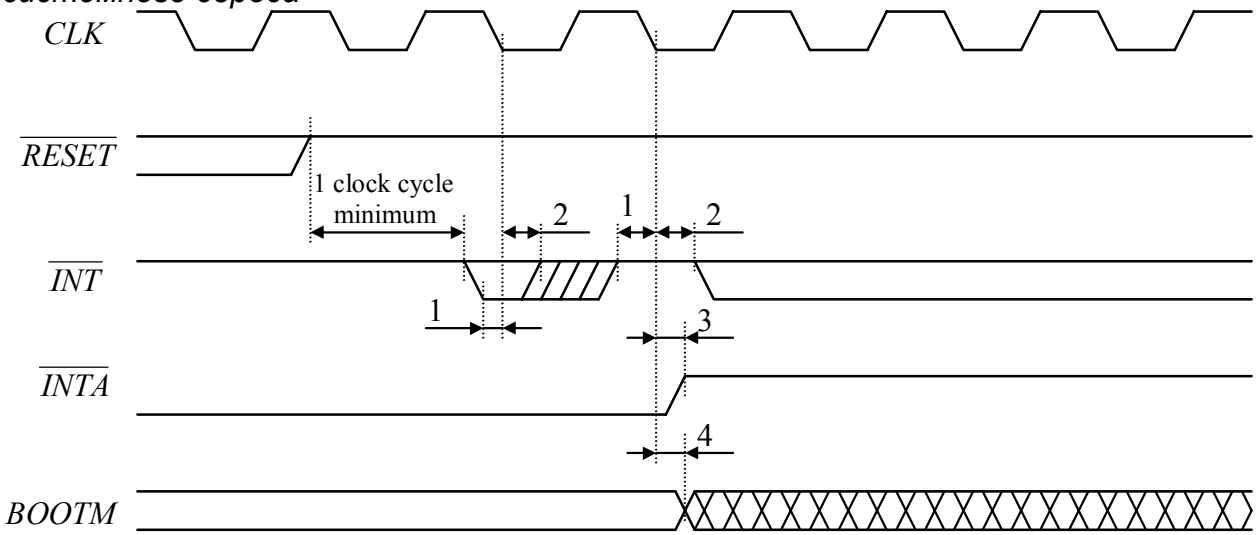
Табл. 13-9. Временные параметры для системного сброса

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{su(RESET)}$	Время предустанова сигнала $\overline{RESET}$ относительно положительного фронта тактового сигнала CLK	12	
2.	$t_{dis(CKF-DZ)}$	Задержка перехода шин $(L)D$ в высокоимпедансное состояние относительно отрицательного фронта CLK		9
3.	$t_{dis(CKF-AZ)}$	Задержка перехода шины A в высокоимпедансное состояние относительно отрицательного фронта CLK		12
4.	$t_{dis(CKF-CONZ)}$	Задержка перехода сигналов управления в высокоимпедансное состояние относительно отрицательного фронта CLK		12
5.	$t_{dis(CKR-HOLDIZ)}$	Задержка перехода сигналов $\overline{(L)HOLDI}$ в высокоимпедансное состояние относительно положительного фронта CLK		15.5
6.	$t_{d(CKF-LAL)}$	Задержка выдачи низкого уровня на шине LA относительно отрицательного фронта CLK		13
7.	$t_{d(CKF-CONH1)}$	Задержка выдачи высокого уровня на сигналах управления относительно отрицательного фронта CLK		12.5
8.	$t_{d(CKF-CONH2)}$	Задержка выдачи высокого уровня на сигналах управления относительно отрицательного фронта CLK		12

Табл. 13-9. Временные параметры для системного сброса (Продолжение)

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
9.	$t_{d(CKF-LWEL)}$	Задержка выдачи низкого уровня сигнала $\overline{LWE}$ относительно отрицательного фронта CLK		13
10.	$t_{d(CKR-INTAL)}$	Задержка выдачи низкого уровня сигнала $\overline{INTA}$ относительно положительного фронта CLK		18.5
11.	$t_{dis(RESETF-ASZ)}$	Задержка перехода сигналов с асинхронным сбросом в высокоимпедансное состояние относительно отрицательного фронта $\overline{RESET}$		5
12.	$t_{d(CKR-CDIR0L)}$	Задержка выдачи низкого уровня сигнала $\overline{CDIR0}$ относительно положительного фронта CLK		14
13.	$t_{d(CKF-CDIR1H)}$	Задержка выдачи высокого уровня сигнала $\overline{CDIR1}$ относительно отрицательного фронта CLK		16
14.	$t_{d(RESETR-ASX)}$	Задержка перехода шины $C0D(7-0)$ из высокоимпедансного состояния в режим выдачи относительно положительного фронта $\overline{RESET}$		4.5
15.	$t_{d(RESETR-ASH)}$	Задержка перехода сигналов с асинхронным сбросом из высокоимпедансного состояния в режим выдачи высокого уровня относительно положительного фронта $\overline{RESET}$		5

Рис. 13-5. Временная диаграмма инициализации процессора NM6403 после системного сброса

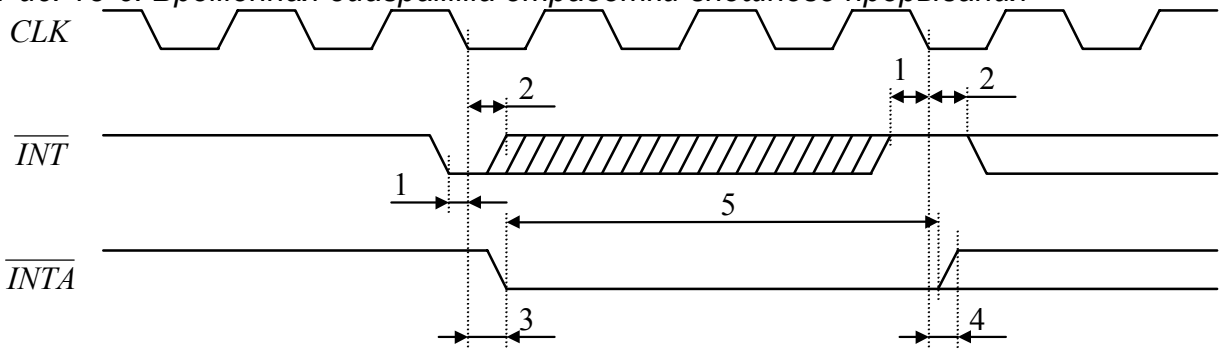


Примечание: *Инициализация процессора NM6403 после системного сброса может производиться в зависимости от уровня сигнала на входе BOOTM двумя способами:*
Если на входе BOOTM низкий уровень, то инициализация происходит по внешнему прерыванию (отрицательный импульс на входе $\overline{INT}$ *). Данное прерывание не будет отработано, но начнёт выбираться первая команда по адресу C0000000 hex. Индикацией того, что инициализация закончилась, будет наличие высокого уровня на выходе* $\overline{INTA}$*. После этого события процессор сможет нормально отслеживать внешние прерывания, а состояние входа BOOTM не будет иметь значение;*
Если на входе BOOTM высокий уровень, то инициализация происходит через коммуникационный порт 1, по которому производится приём пакета данных (см. раздел 12.1). По окончании инициализации (момент времени, когда разрешается запрос на запись последнего принятого слова в память в режиме ПДП) на выходе $\overline{INTA}$ *будет высокий уровень. После данного события процессор сможет нормально отслеживать внешние прерывания, а состояние входа BOOTM не будет иметь значение. До этого на входе* $\overline{INT}$ *необходимо поддерживать высокий уровень во избежание неоднозначной работы микросхемы.*

Табл. 13-10. Временные параметры при инициализации процессора NM6403

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{su(INT)}$	Время предустанова сигнала $\overline{INT}$ относительно отрицательного фронта тактового сигнала *CLK*	15	
2.	$t_{h(INT)}$	Время удержания сигнала $\overline{INT}$ относительно отрицательного фронта *CLK*	10	
3.	$t_{d(CKF-INTAH)}$	Задержка выдачи высокого уровня на выходе $\overline{INTA}$ относительно отрицательного фронта *CLK*		16.5
4.	$t_{d(CKF-BOOTMX)}$	Время удержания сигнала *BOOTM* относительно отрицательного фронта *CLK*		10

Рис. 13-6. Временная диаграмма отработки внешнего прерывания



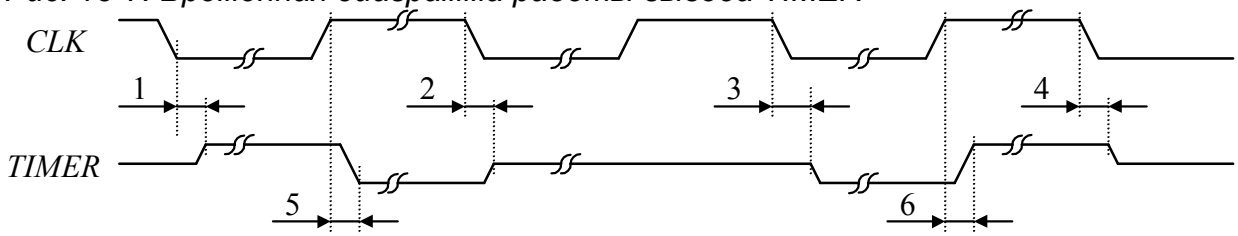
Примечание: *Вход внешнего прерывания $\overline{INT}$ может стать активным (низкий уровень) только тогда, когда на выходе $\overline{INTA}$ высокий уровень, в противном случае прерывание может быть проигнорировано. Низкий уровень на выходе $\overline{INTA}$ означает, что ещё не обработалось текущее внешнее прерывание. Изменение на данном выводе уровня с низкого на высокий произойдёт, если выполнились следующие два условия: на входе $\overline{INT}$ высокий уровень и прошёл запрос на выборку первой команды программы обработки внешнего прерывания.*

Табл. 13-11. Временные параметры при отработке внешнего прерывания

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{su(INT)}$	Время предустанова сигнала $\overline{INT}$ относительно отрицательного фронта тактового сигнала *CLK*	15	
2.	$t_{h(INT)}$	Время удержания сигнала $\overline{INT}$ относительно отрицательного фронта *CLK*	10	
3.	$t_{d(CKF-INTAL)}$	Задержка выдачи низкого уровня на выходе $\overline{INTA}$ относительно отрицательного фронта *CLK*		18.5
4.	$t_{d(CKF-INTAH)}$	Задержка выдачи высокого уровня на выходе $\overline{INTA}$ относительно отрицательного фронта *CLK*		16.5
5.	$t_{w(INTAL)}$	Длительность импульса низкого уровня на выходе $\overline{INTA}$	2P-2	

Примечание: *P - период тактового сигнала CLK.*

Рис. 13-7. Временная диаграмма работы вывода TIMER

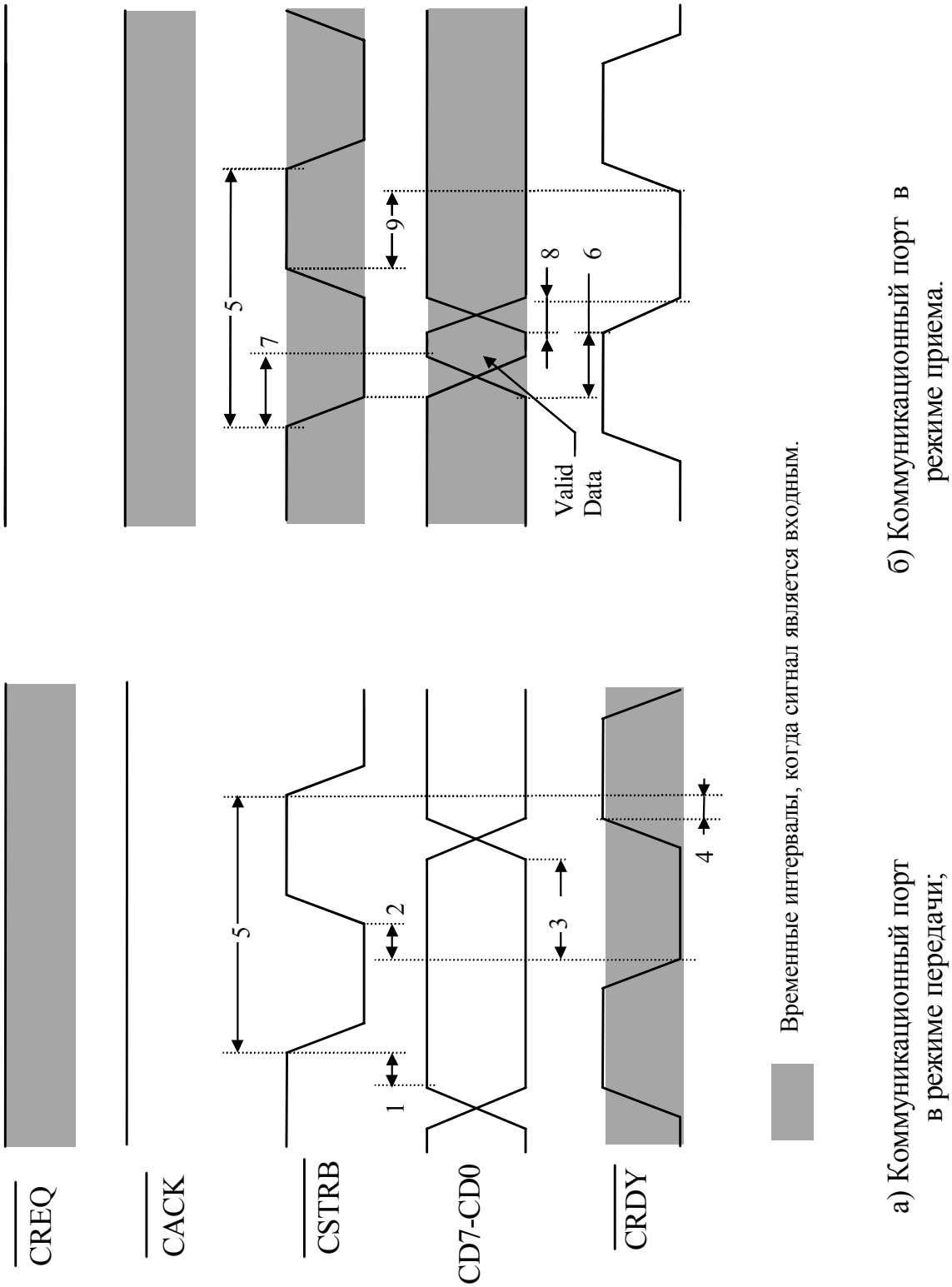


Примечание: *Вывод TIMER после системного сброса переходит в режим приёма. Уровень напряжения на нём при инициализации по внешнему прерыванию задаёт, какой тип памяти используется в банке памяти 1 глобальной шины, откуда будет выбираться первая команда: если низкий - то статическая, если высокий - динамическая. После первой записи в GMICR (регистр управления глобальным интерфейсом) данный вывод может использоваться в качестве программно управляемого выхода. Запись соответствующёй информации в 29 - 26 разряды регистра PSWR(регистр слова состояния процессора) приведёт к выдаче на нём либо высокого, либо низкого уровня, либо периодического сигнала, параметры которого определяются работой таймера 0 или 1. Имеется также возможность перевода этого вывода обратно в высокоимпедансное состояние*

Табл. 13-12. Временные параметры работы вывода TIMER

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{d(CKF-TIMERZH)}$	Задержка перехода вывода *TIMER* из высокоимпедансного состояния в режим выдачи высокого уровня относительно отрицательного фронта *CLK*		17
2.	$t_{dis(CKF-TIMERLZ)}$	Задержка перехода вывода *TIMER* из режима выдачи низкого уровня в высокоимпедансное состояние относительно отрицательного фронта *CLK*		9
3.	$t_{d(CKF-TIMERZL)}$	Задержка перехода вывода *TIMER* из высокоимпедансного состояния в режим выдачи низкого уровня относительно отрицательного фронта *CLK*		16
4.	$t_{dis(CKF-TIMERHZ)}$	Задержка перехода вывода *TIMER* из режима выдачи высокого уровня в высокоимпедансное состояние относительно отрицательного фронта *CLK*		9
5.	$t_{d(CKR-TIMERHL)}$	Задержка перехода вывода *TIMER* с выдачи высокого уровня на выдачу низкого относительно положительного фронта *CLK*		13.5
6.	$t_{d(CKR-TIMERLH)}$	Задержка перехода вывода *TIMER* с выдачи низкого уровня на выдачу высокого относительно положительного фронта *CLK*		15

Рис. 13-8. Временные диаграммы передачи байта через коммуникационный порт



а) Коммуникационный порт в режиме передачи;

б) Коммуникационный порт в режиме приема.

Табл. 13-13. Временные параметры коммуникационного порта при передаче байта данных

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{su(CD)W}$	Задержка выдачи среза $\overline{CSTRB}$ после выдачи байта данных (передатчик)	2	
2.	$t_{d(RL-SH)W}$	Задержка выдачи фронта $\overline{CSTRB}$ после приема среза $\overline{CRDY}$ (передатчик)	0	20
3.	$t_{h(CD)W}$	Время удержания данных после приема среза $\overline{CRDY}$ (передатчик)	2	
4.	$t_{d(RH-SL)W}$	Задержка выдачи среза $\overline{CSTRB}$ после приема фронта $\overline{CRDY}$ (передатчик)	0	18
5.	t_{BYTE}	Период передачи байтов		71
6.	$t_{d(SL-RL)R}$	Задержка выдачи среза $\overline{CRDY}$ после приема среза $\overline{CSTRB}$ (приемник)	0	18
7.	$t_{su(CD)R}$	Время предустанова данных перед приемом среза $\overline{CSTRB}$ (приемник)	0	
8.	$t_{h(CD)R}$	Время удержания данных после выдачи среза $\overline{CRDY}$ (приемник)	2	
9.	$t_{d(SH-RH)R}$	Задержка выдачи фронта $\overline{CRDY}$ после приема фронта $\overline{CSTRB}$ (приемник)	0	15

Табл. 13-14. Временные параметры коммуникационного порта при передаче 32-разрядного слова данных

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	t_{WORD}	Период передачи 32-разрядных слов	2P	2.5P+284
2.	$t_{d(RL-SL)W}$	Задержка выдачи фронта $\overline{CSTRB}$ первого байта нового слова после получения среза $\overline{CRDY}$ последнего байта предыдущего слова	1.5P	2.5P+20

Примечание: *P - период тактового сигнала CLK.*

Рис. 13-9. Временные диаграммы передачи 32-разрядного слова через коммуникационный порт

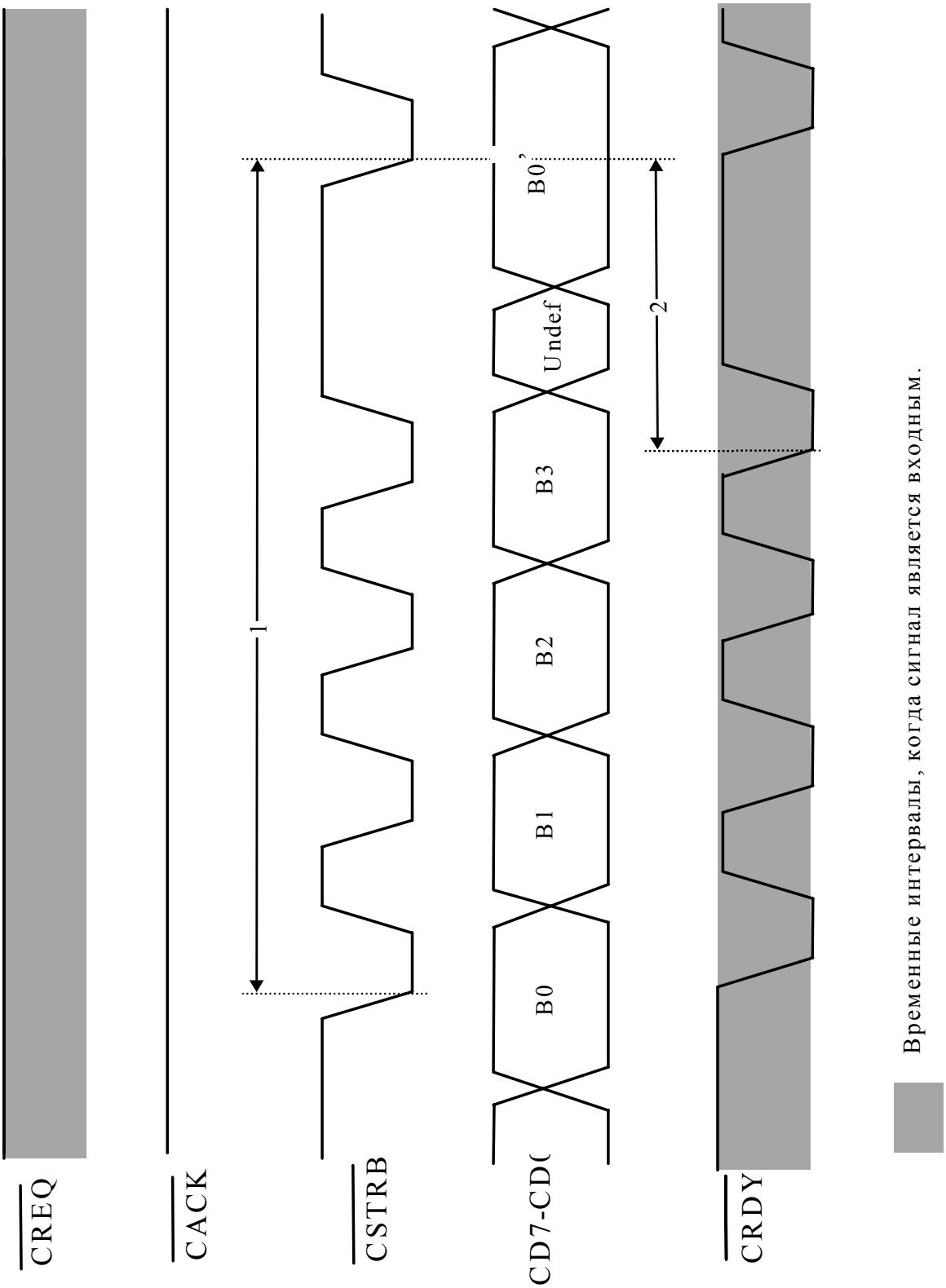


Рис. 13-10. Временные диаграммы перехода коммуникационного порта из режима приема в режим передачи

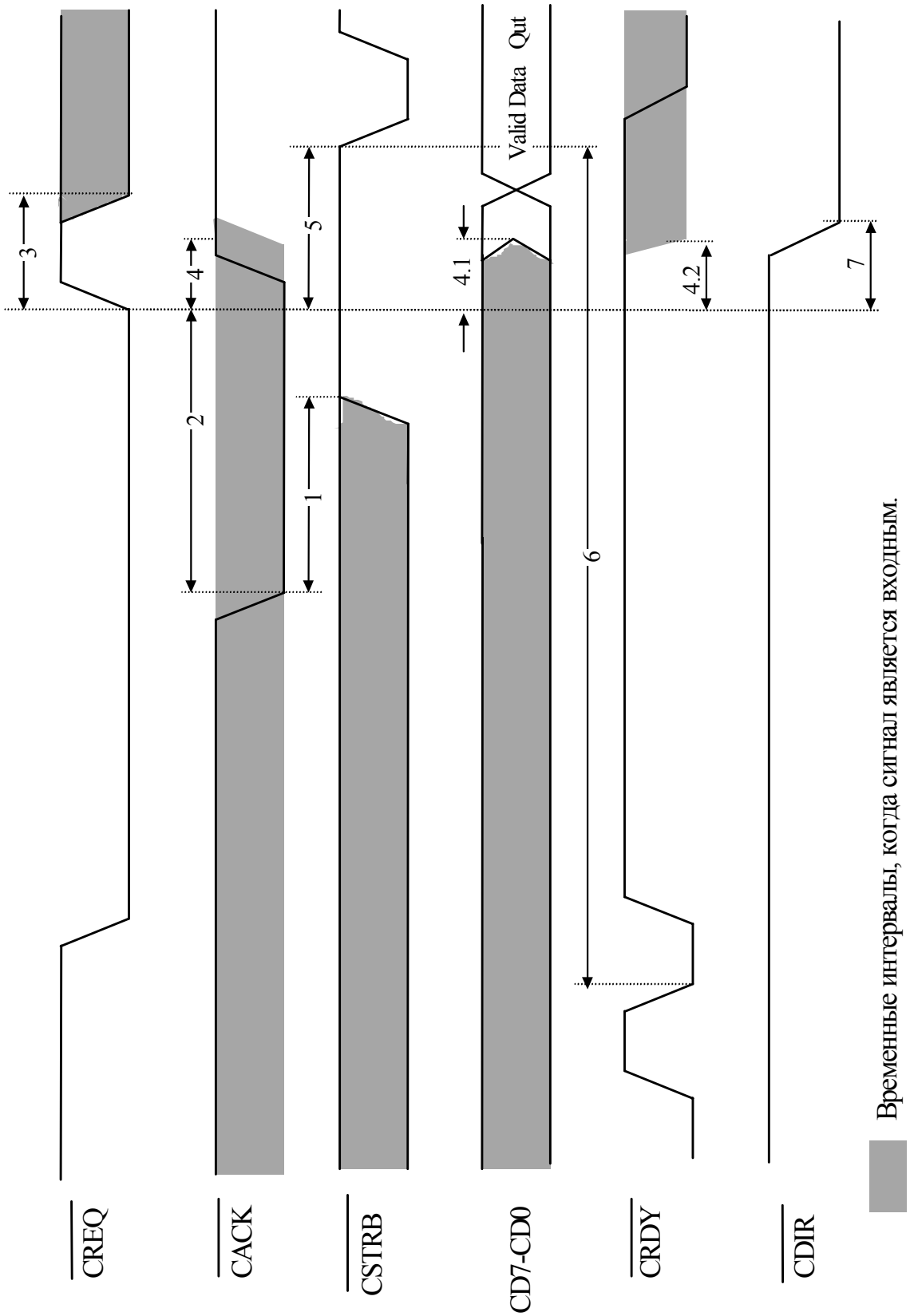


Табл. 13-15. Временные параметры коммуникационного порта при переходе из режима приема в режим передачи

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{d(AL-SO)T}$	Задержка перехода $\overline{CSTRB}$ из высокоимпедансного состояния в пассивное после получения среза $\overline{CACK}$	0.5P	1.5P+10
2.	$t_{d(AL-RQH)T}$	Задержка выдачи фронта $\overline{CREQ}$ после получения среза $\overline{CACK}$	P	2P+13.5
3.	$t_{d(RQH-RQI)T}$	Задержка перехода $\overline{CREQ}$ из пассивного состояния в высокоимпедансное после выдачи фронта $\overline{CREQ}$	0.5P	0.5P+9
4.	$t_{d(RQH-AO)T}$	Задержка перехода $\overline{CACK}$ в высокоимпедансное состояние после выдачи фронта $\overline{CREQ}$	0.5P	0.5P+9
4.1	$t_{d(RQH-DO)T}$	Задержка выхода шины данных из высокоимпедансного состояния после выдачи фронта $\overline{CREQ}$	0.5P	0.5P+8
4.2	$t_{d(RQH-RI)T}$	Задержка перехода $\overline{CRDY}$ из пассивного состояния в высокоимпедансное после выдачи фронта $\overline{CREQ}$	0.5P	0.5P+9
5.	$t_{d(RQH-SL)T}$	Задержка выдачи среза $\overline{CSTRB}$ после выдачи фронта $\overline{CREQ}$	1.5P	1.5P+20
6.	$t_{d(RL-SL)T}$	Задержка выдачи среза $\overline{CSTRB}$ после выдачи среза $\overline{CRDY}$	3.5P	5.5P+20
7.	$t_{d(RQH-DL)T}$	Задержка переключения выхода $\overline{CDIR}$ с высокого уровня на низкий после выдачи фронта $\overline{CREQ}$	0.5P	0.5P+ 14.5

Примечание: *P - период тактового сигнала CLK.*

Рис. 13-11. Временные диаграммы перехода коммуникационного порта из режима передачи в режим приема

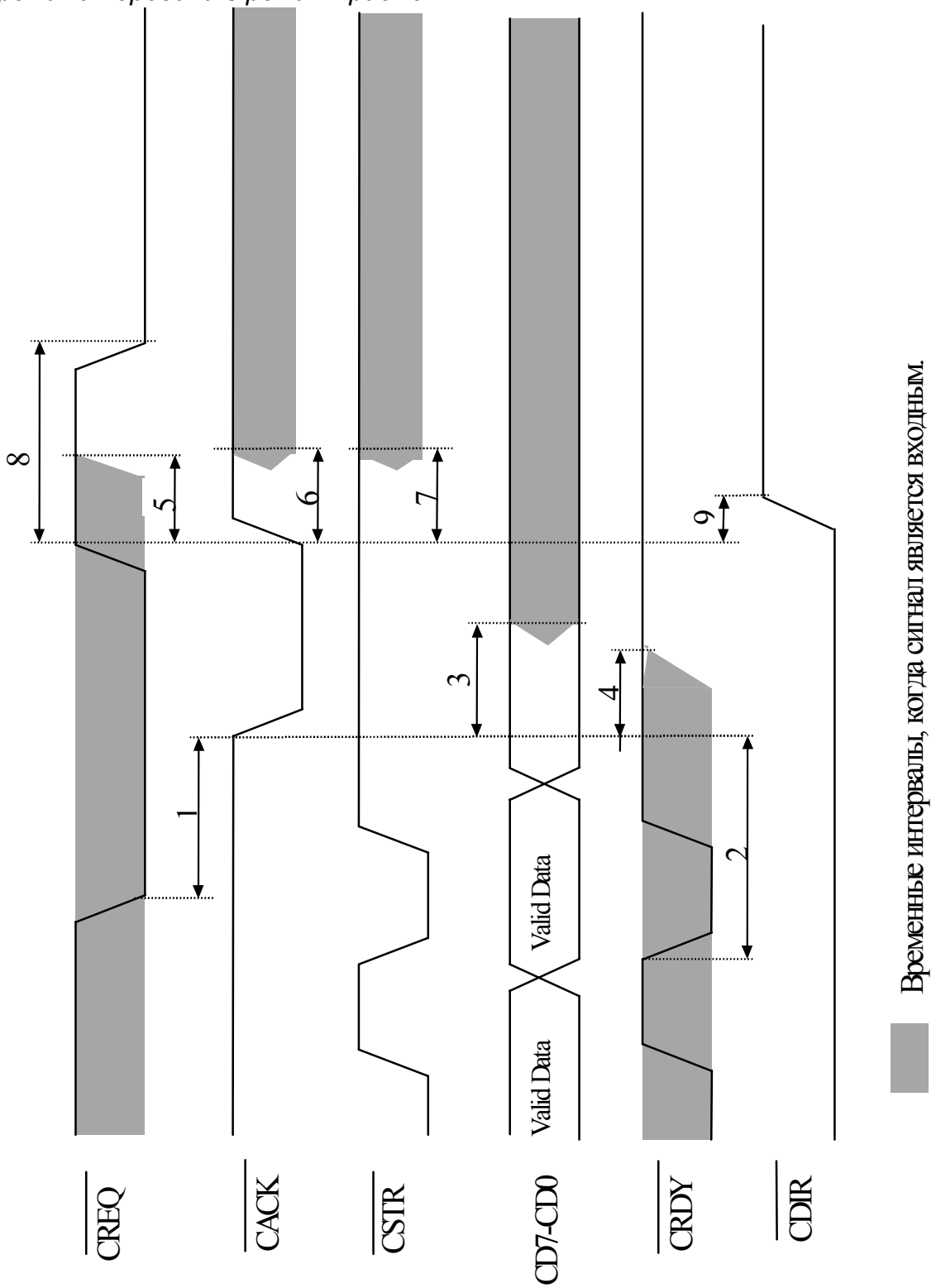
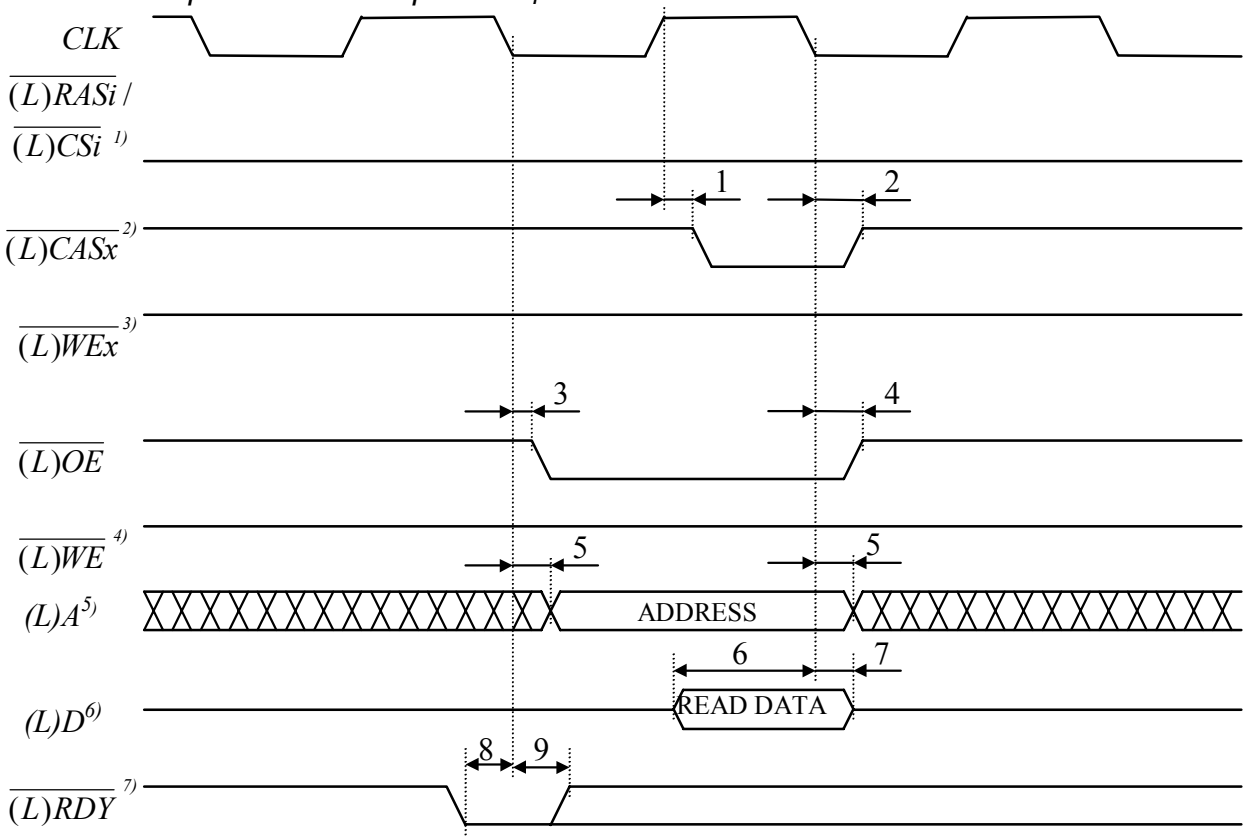


Табл. 13-16. Временные параметры коммуникационного порта при переходе из режима передачи в режим приема

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{d(RQL-AL)T}$	Задержка выдачи среза $\overline{CACK}$ после приема среза $\overline{CREQ}$	P	2P+16
2.	$t_{d(RL-AL)T}$	Задержка выдачи среза $\overline{CACK}$ после приема среза $\overline{CRDY}$	P	2P+16
3.	$t_{d(AL-CD)I}$	Задержка перехода шины данных в высокоимпедансное состояние после выдачи среза $\overline{CACK}$	0.5P	0.5P+7.5
4.	$t_{d(AL-RO)T}$	Задержка перехода $\overline{CRDY}$ из высокоимпедансного состояния в пассивное после выдачи среза $\overline{CACK}$	0.5P	0.5P+8
5.	$t_{d(RQH-RQO)T}$	Задержка перехода $\overline{CREQ}$ из высокоимпедансного состояния в пассивное после получения фронта $\overline{CREQ}$		18
6.	$t_{d(RQH-AI)T}$	Задержка перехода $\overline{CACK}$ в высокоимпедансное после получения фронта $\overline{CREQ}$		18
7.	$t_{d(RQH-SI)T}$	Задержка перехода $\overline{CSTRB}$ из пассивного состояния в высокоимпедансное после получения фронта $\overline{CREQ}$		9
8.	$t_{d(RQH-RQL)T}$	Задержка перехода $\overline{CREQ}$ в активное состояние после получения фронта $\overline{CREQ}$	P	2P+15.5
9.	$t_{d(RQH-DH)T}$	Задержка переключения выхода $\overline{CDIR}$ с низкого уровня на высокий после получения фронта $\overline{CREQ}$		13.5

Примечание: *P - период тактового сигнала CLK.*

Рис. 13-12. Временная диаграмма цикла чтения из памяти



Примечание:

1) Имеется в виду либо сигнал $\overline{(L)RAS0}$ *или* $\overline{(L)RAS1}$ *при работе с динамической памятью, либо* $\overline{(L)CS0}$ *или* $\overline{(L)CS1}$ *при работе со статической памятью;*

2) Имеются в виду сигналы $\overline{(L)CASH}$ *и* $\overline{(L)CASL}$ *при работе с динамической памятью, в случае обращения к статической памяти см. п.3.;*

3) Имеются в виду сигналы $\overline{(L)WEH}$ *и* $\overline{(L)WEL}$ *при работе со статической памятью;*

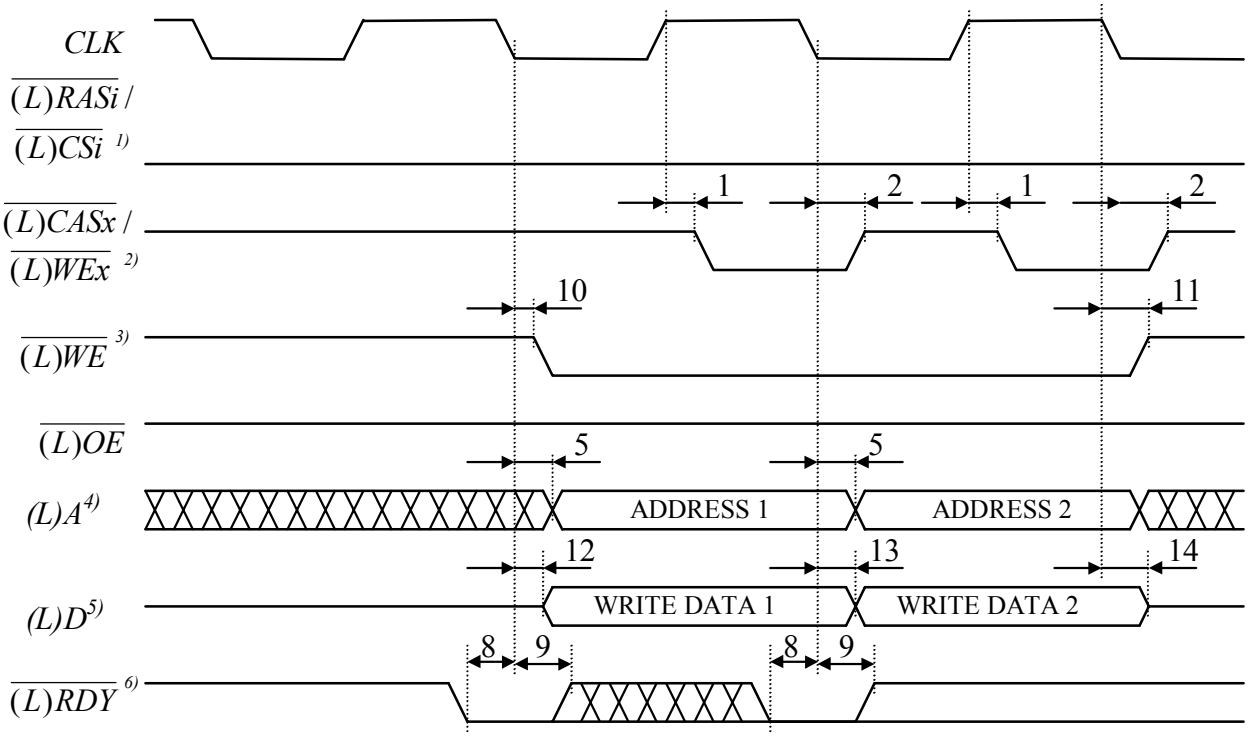
4) Имеется в виду сигнал $\overline{(L)WE}$ *при работе с динамической памятью, в случае обращения к статической памяти см. п.5;*

5) Обозначение L(A) относится к адресной шине LA(15-1) или A(15-1), а также к тем сигналам, которые в данный момент выступают в роли адресных: (L)A16 вместо $\overline{(L)WE}$, *(L)A17 вместо* $\overline{(L)HOLDO}$, *(L)A18 вместо* $\overline{(L)HOLDI}$, *(L)A19 вместо* $\overline{(L)RDY}$;

6) Обозначение (L)D относится к шинам LD(63-0) и D(63-0);

В данном случае $\overline{(L)RDY}$ *используется в качестве внешнего сигнала готовности для формирования дополнительных циклов ожидания.*

Рис. 13-13. Временная диаграмма цикла записи в память



Примечание:

1) Имеется в виду либо сигнал $\overline{(L)RAS0}$ *или* $\overline{(L)RAS1}$ *при работе с динамической памятью, либо* $\overline{(L)CS0}$ *или* $\overline{(L)CS1}$ *при работе со статической памятью;*

2) Имеются в виду либо сигналы $\overline{(L)CASH}$ *и* $\overline{(L)CASL}$ *при работе с динамической памятью, либо сигналы* $\overline{(L)WEH}$ *и* $\overline{(L)WEL}$ *при работе со статической памятью;*

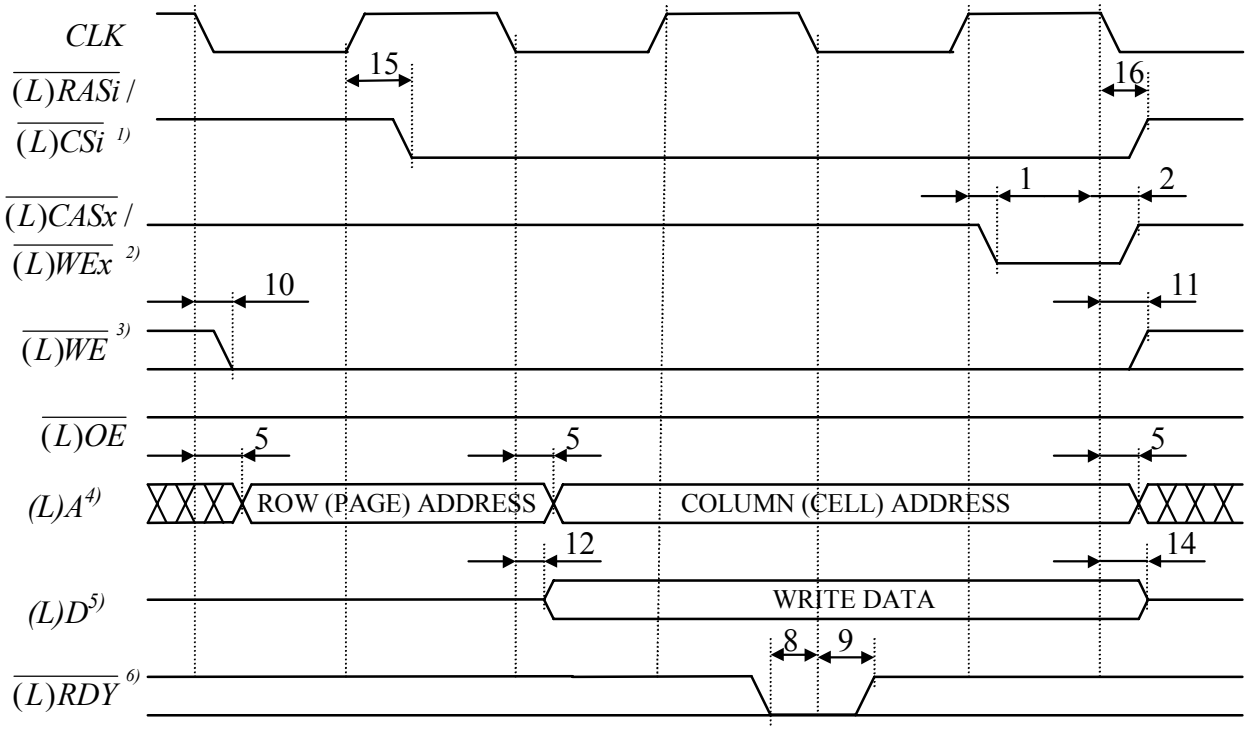
3) Имеется в виду сигнал $\overline{(L)WE}$ *при работе с динамической памятью, в случае обращения к статической памяти см. п.5;*

4) Обозначение (L)A относится к адресной шине LA(15-1) или A(15-1), а также к тем сигналам, которые в данный момент выступают в роли адресных: (L)A16 вместо $\overline{(L)WE}$, *(L)A17 вместо* $\overline{(L)HOLDO}$, *(L)A18 вместо* $\overline{(L)HOLDI}$, *(L)A19 вместо* $\overline{(L)RDY}$;

5) Обозначение (L)D относится к шинам LD(63-0) и D(63-0);

6) В данном случае $\overline{(L)RDY}$ *используется в качестве внешнего сигнала готовности для формирования дополнительных циклов ожидания.*

Рис. 13-14. Временная диаграмма перехода в новую страницу памяти



Примечание:

1) Имеется в виду либо сигнал $\overline{(L)RAS0}$ *или* $\overline{(L)RAS1}$ *при работе с динамической памятью, либо* $\overline{(L)CS0}$ *или* $\overline{(L)CS1}$ *при работе со статической памятью;*

2) Имеются в виду либо сигналы $\overline{(L)CASH}$ *и* $\overline{(L)CASL}$ *при работе с динамической памятью, либо сигналы* $\overline{(L)WEH}$ *и* $\overline{(L)WEL}$ *при работе со статической памятью;*

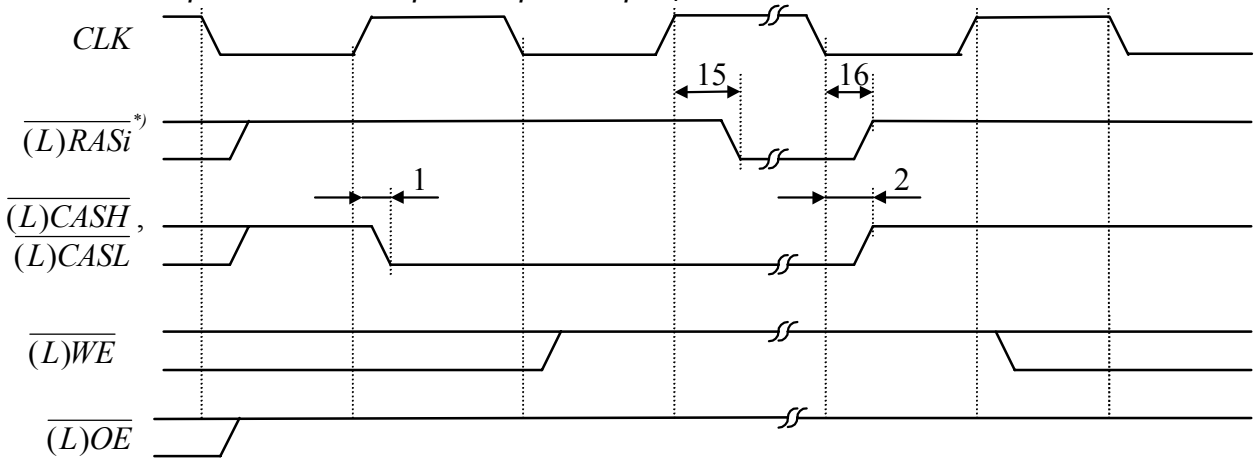
3) Имеется в виду сигнал $\overline{(L)WE}$ *при работе с динамической памятью, в случае обращения к статической памяти см. п.4;*

4) Обозначение (L)A относится к адресной шине LA(15-1) или A(15-1), а также к тем сигналам, которые в данный момент выступают в роли адресных: (L)A16 вместо $\overline{(L)WE}$, *(L)A17 вместо* $\overline{(L)HOLDO}$, *(L)A18 вместо* $\overline{(L)HOLDI}$, *(L)A19 вместо* $\overline{(L)RDY}$;

5) Обозначение (L)D относится к шинам LD(63-0) и D(63-0);

6) В данном случае $\overline{(L)RDY}$ *используется в качестве внешнего сигнала готовности для формирования дополнительных циклов ожидания.*

Рис. 13-15. Временная диаграмма регенерации i-го банка динамической памяти



Примечание: **) Имеется в виду сигнал $\overline{(L)RAS0}$ или $\overline{(L)RAS1}$ при работе с динамической памятью, при работе со статической памятью сигналы $\overline{(L)CS0}$ или $\overline{(L)CS1}$ остаются пассивными.*

Табл. 13-17. Временные параметры сигналов глобального (локального) интерфейса при работе с внешней памятью

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{d(CKR-CASHL)}$	Задержка перехода выводов $\overline{(L)CASH}/\overline{(L)WEH}$ и $\overline{(L)CASL}/\overline{(L)WEL}$ с выдачи высокого уровня на выдачу низкого относительно положительного фронта *CLK*		9.5
2.	$t_{d(CKF-CASLH)}$	Задержка перехода выводов $\overline{(L)CASH}/\overline{(L)WEH}$ и $\overline{(L)CASL}/\overline{(L)WEL}$ с выдачи низкого уровня на выдачу высокого относительно отрицательного фронта *CLK*		11.5
3.	$t_{d(CKF-OEHL)}$	Задержка перехода вывода $\overline{(L)OE}$ с выдачи высокого уровня на выдачу низкого относительно отрицательного фронта *CLK*		13.5
4.	$t_{d(CKF-OELH)}$	Задержка перехода вывода $\overline{(L)OE}$ с выдачи низкого уровня на выдачу высокого относительно отрицательного фронта *CLK*		13
5.	$t_{d(CKF-A)}$	Задержка переключения шины адреса относительно отрицательного фронта *CLK*		12.5
6.	$t_{su(D)}$	Время предустанова данных относительно отрицательного фронта тактового сигнала *CLK*	5	

Табл. 13-17. Временные параметры сигналов глобального (локального) интерфейса при работе с внешней памятью (Продолжение)

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
7.	$t_{h(D)}$	Время удержания данных относительно отрицательного фронта CLK	2	
8.	$t_{su(RDY)}$	Время предустанова сигнала $\overline{(L)RDY}$ относительно отрицательного фронта тактового сигнала CLK	15	
9.	$t_{h(RDY)}$	Время удержания сигнала $\overline{(L)RDY}$ относительно отрицательного фронта CLK	10	
10.	$t_{d(CKF-WEHL)}$	Задержка перехода вывода $\overline{(L)WE}$ с выдачи высокого уровня на выдачу низкого относительно отрицательного фронта CLK		12
11.	$t_{d(CKF-WELH)}$	Задержка перехода вывода $\overline{(L)WE}$ с выдачи низкого уровня на выдачу высокого относительно отрицательного фронта CLK		13
12.	$t_{d(CKF-ZD)}$	Задержка перехода шины данных из высокоимпедансного состояния в режим выдачи относительно отрицательного фронта CLK		18
13.	$t_{d(CKF-D)}$	Задержка переключения шины данных относительно отрицательного фронта CLK		18
14.	$t_{dis(CKF-DZ)}$	Задержка перехода шины данных из режима выдачи в высокоимпедансное состояние относительно отрицательного фронта CLK		9
15.	$t_{d(CKR-RASHL)}$	Задержка перехода выводов $\overline{(L)RAS0}/\overline{(L)CS0}$ и $\overline{(L)RAS1}/\overline{(L)CS1}$ с выдачи высокого уровня на выдачу низкого относительно положительного фронта CLK		10
16.	$t_{d(CKF-RASLH)}$	Задержка перехода выводов $\overline{(L)RAS0}/\overline{(L)CS0}$ и $\overline{(L)RAS1}/\overline{(L)CS1}$ с выдачи низкого уровня на выдачу высокого относительно отрицательного фронта CLK		11.5

Рис. 13-16. Временная диаграмма получения контроля над шиной в режиме работы с общей памятью 00

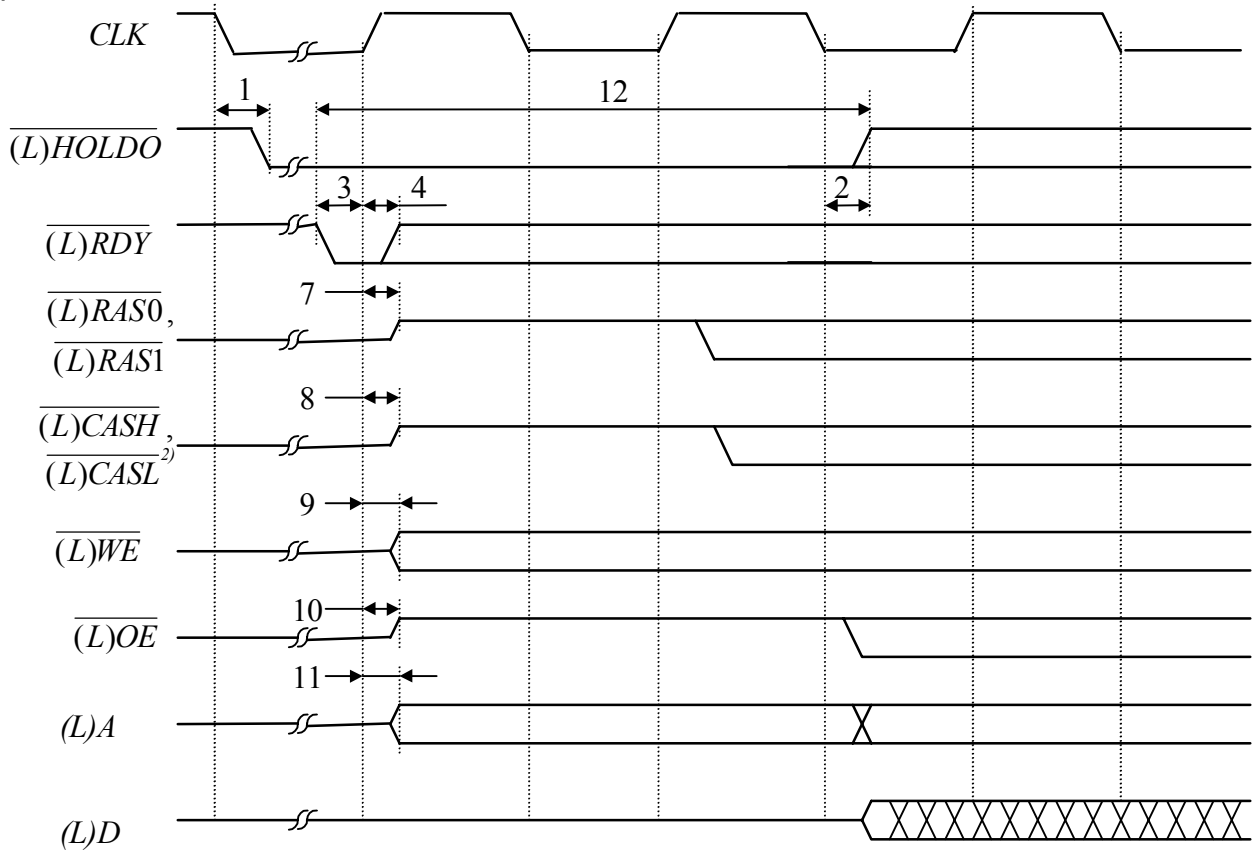


Рис. 13-17. Временная диаграмма получения контроля над банком 1 в режиме работы с общей памятью 01

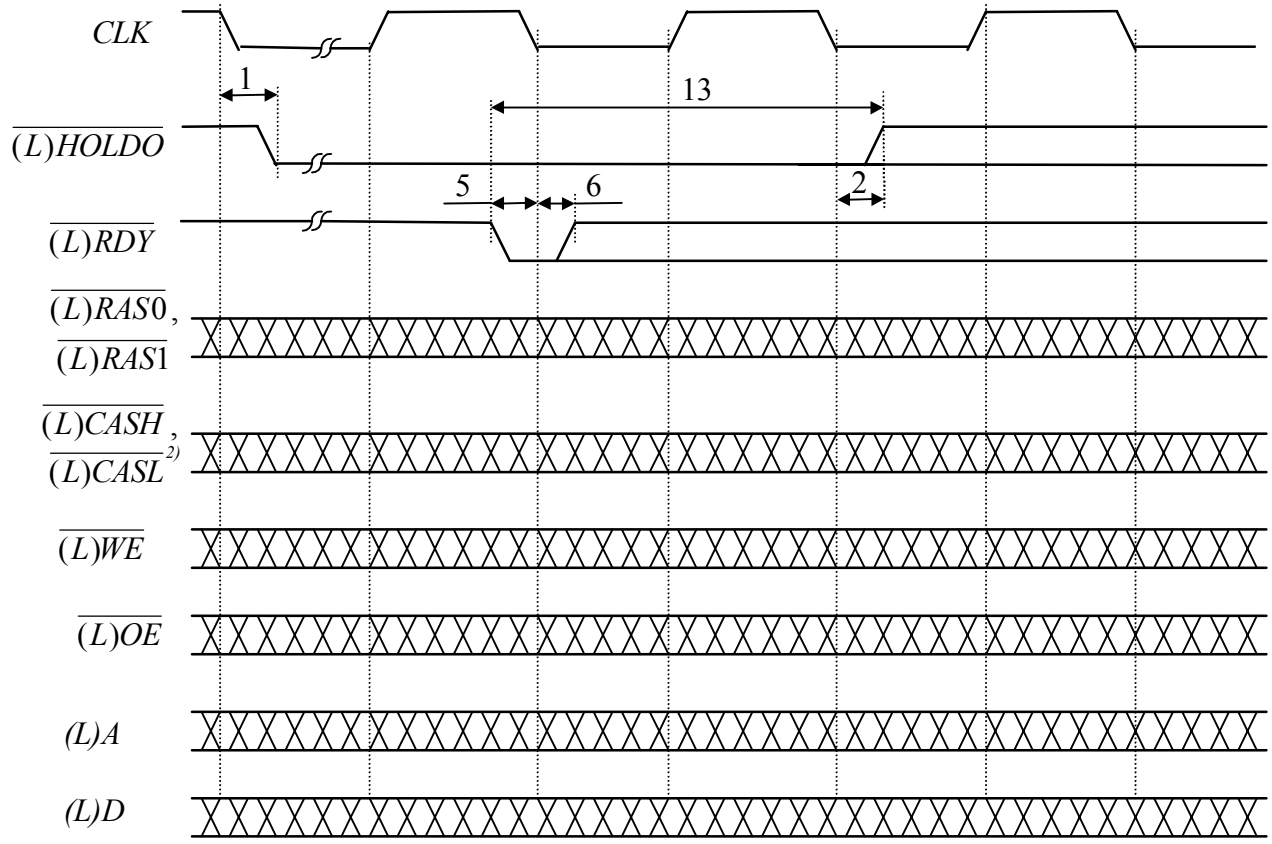


Рис. 13-18. Временная диаграмма получения контроля над банком 1 в режиме работы с общей памятью 10

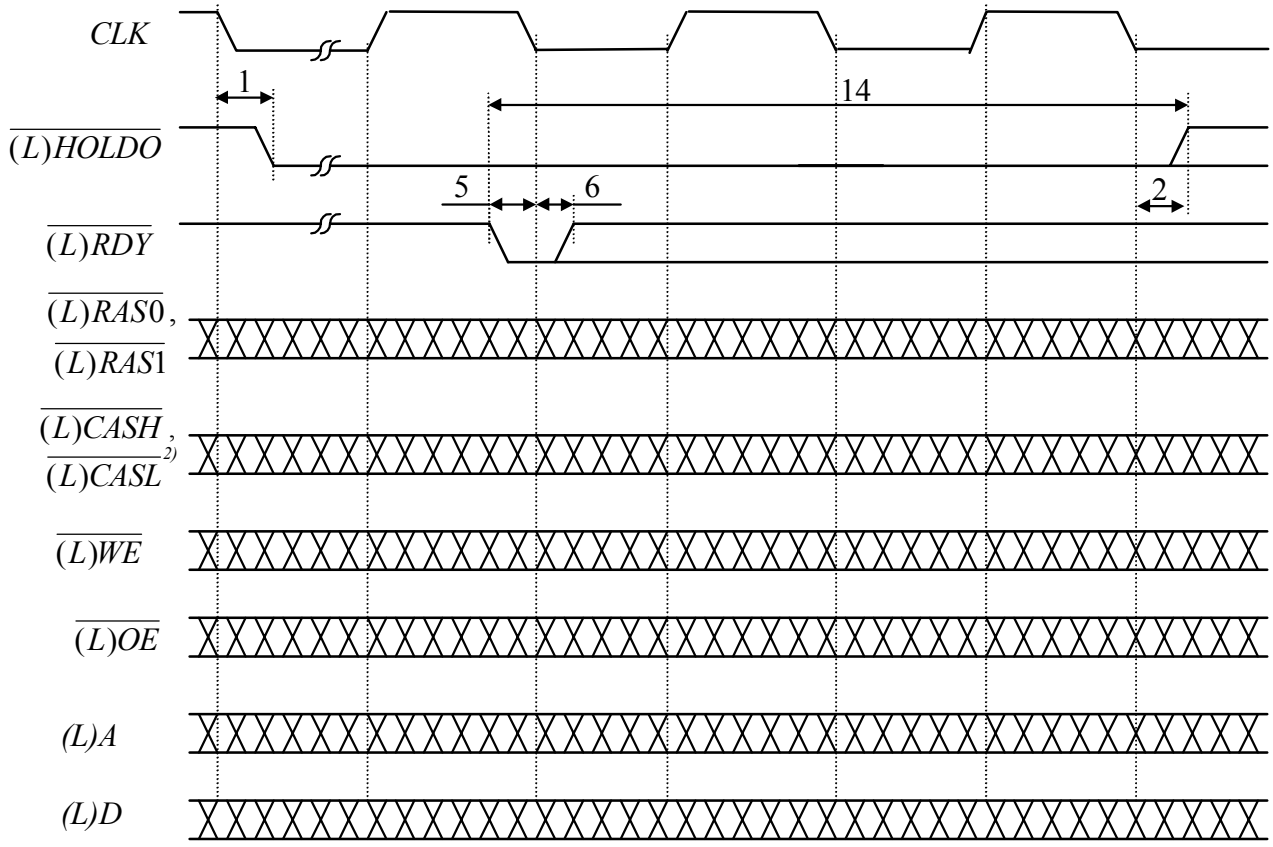


Рис. 13-19. Временная диаграмма передачи контроля над шиной в режиме работы с общей памятью 00

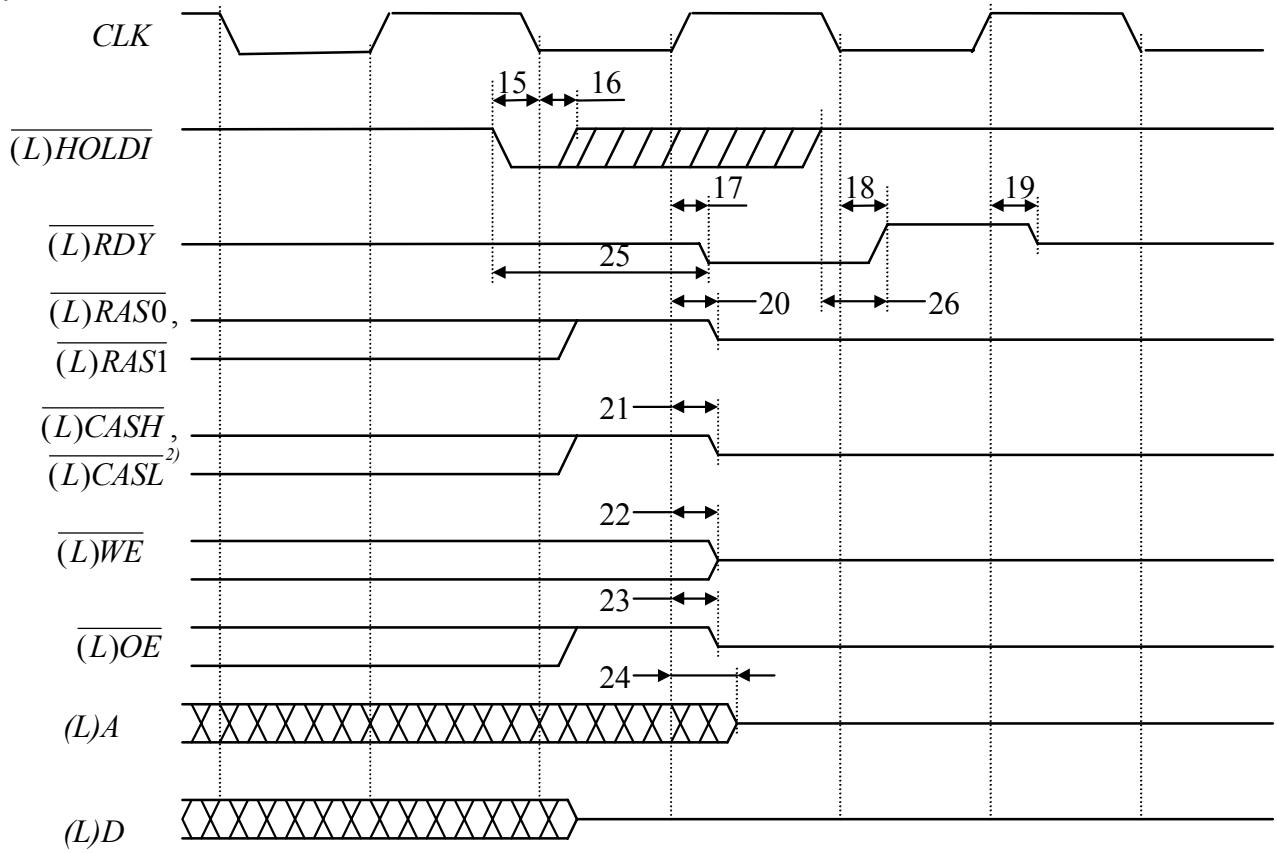


Рис. 13-20. Временная диаграмма передачи контроля над банком 1 в режиме работы с общей памятью 01

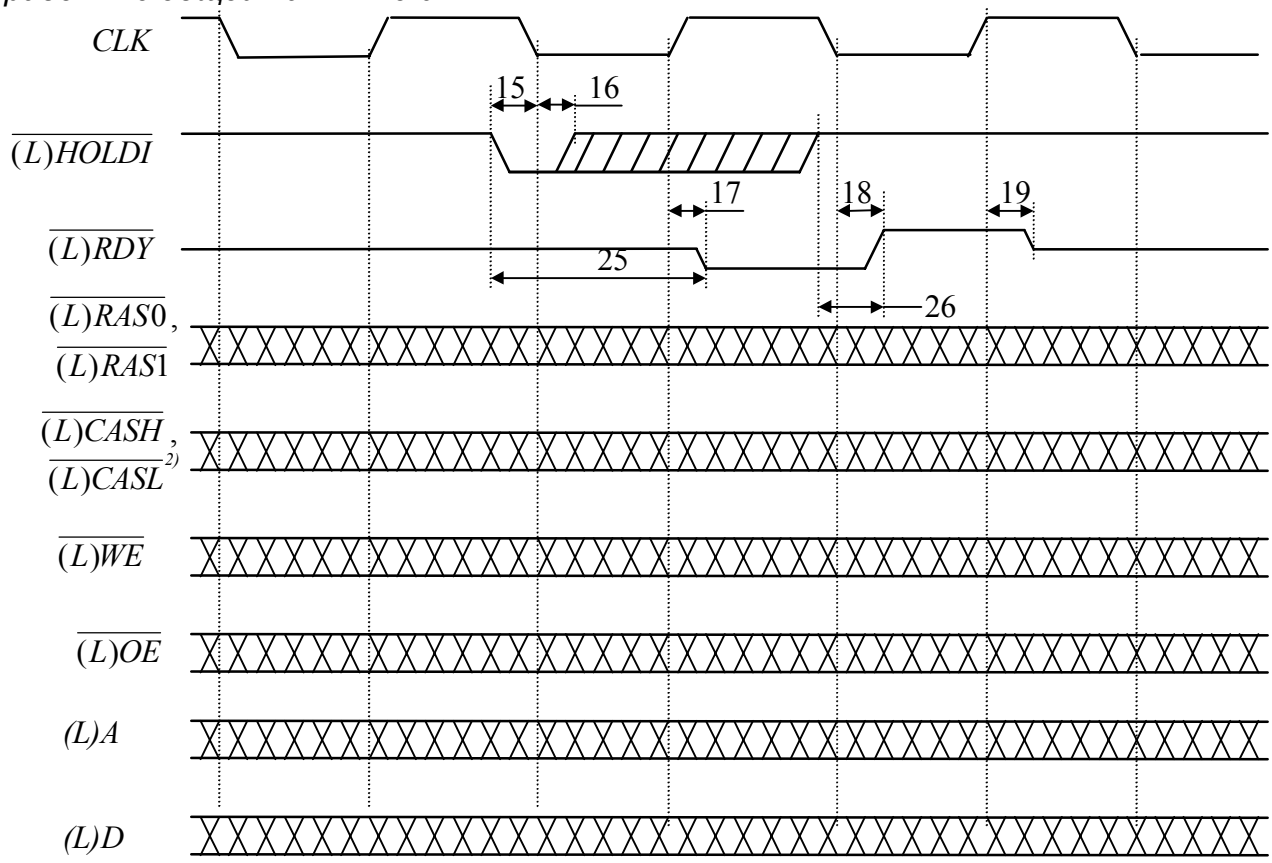


Рис. 13-21. Временная диаграмма передачи контроля над банком 0 в режиме работы с общей памятью 10

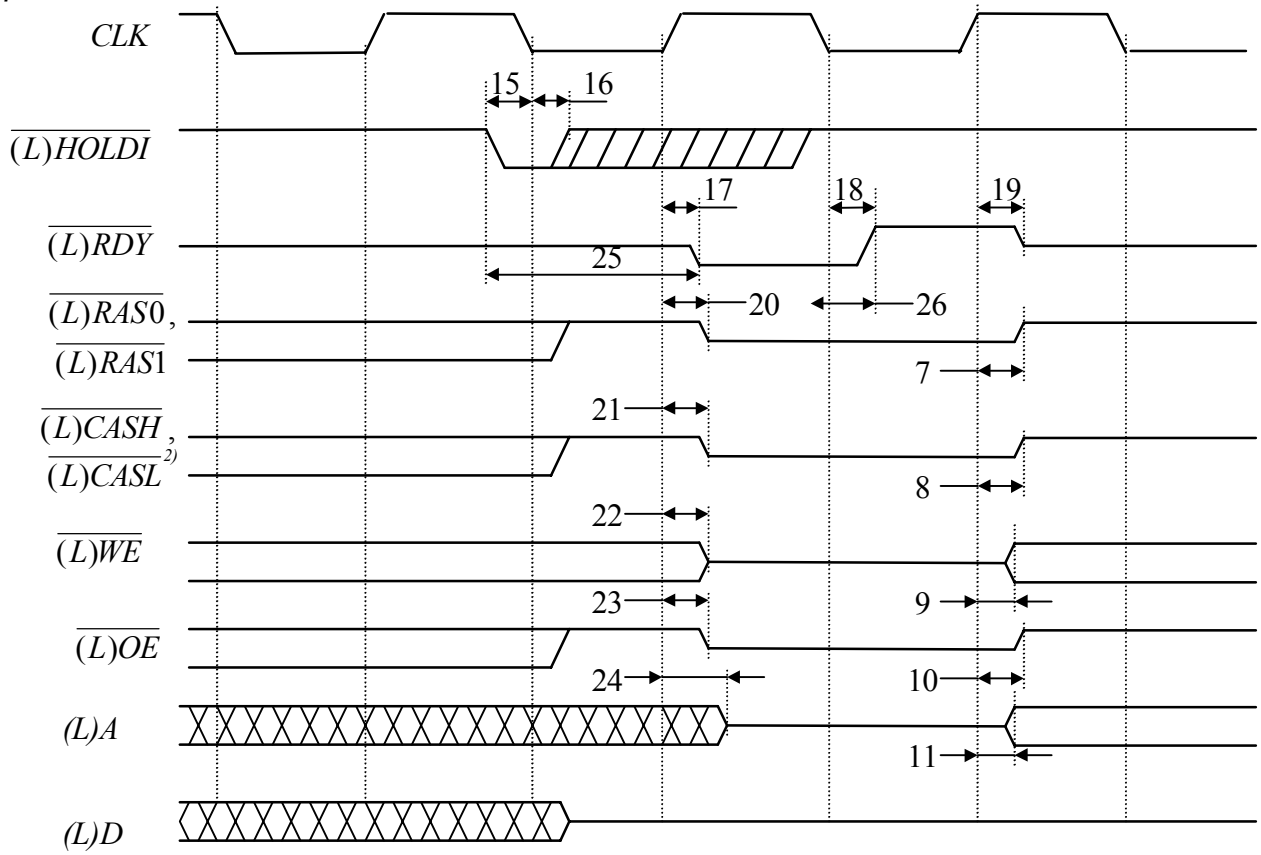


Табл. 13-18. Временные параметры сигналов глобального (локального) интерфейса при работе с общей памятью

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
1.	$t_{d(CKF-HOLDOL)}$	Задержка перехода вывода $\overline{(L)HOLDO}$ с выдачи высокого уровня на выдачу низкого относительно отрицательного фронта CLK		12
2.	$t_{d(CKF-HOLDOH)}$	Задержка перехода вывода $\overline{(L)HOLDO}$ с выдачи низкого уровня на выдачу высокого относительно отрицательного фронта CLK		12.5
3.	$t_{su(RDY-CKR)}$	Время предустанова сигнала $\overline{(L)RDY}$ относительно положительного фронта тактового сигнала CLK	15	
4.	$t_{h(CKR-RDY)}$	Время удержания сигнала $\overline{(L)RDY}$ относительно положительного фронта CLK	10	
5.	$t_{su(RDY-CKF)}$	Время предустанова сигнала $\overline{(L)RDY}$ относительно отрицательного фронта тактового сигнала CLK	15	
6.	$t_{h(CKF-RDY)}$	Время удержания сигнала $\overline{(L)RDY}$ относительно отрицательного фронта CLK	10	
7.	$t_{dis(CKR-RASZH)}$	Задержка перехода выводов $\overline{(L)RAS0}/\overline{(L)CS0}$ и $\overline{(L)RAS1}/\overline{(L)CS1}$ из высокоимпедансного состояния в режим выдачи высокого уровня относительно положительного фронта CLK		13
8.	$t_{dis(CKR-CASZH)}$	Задержка перехода выводов $\overline{(L)CASH}/\overline{(L)WEH}$ и $\overline{(L)CASL}/\overline{(L)WEL}$ из высокоимпедансного состояния в режим выдачи высокого уровня относительно положительного фронта CLK		14.5
9.	$t_{dis(CKR-WE)}$	Задержка перехода вывода $\overline{(L)WE}$ из высокоимпедансного состояния в режим выдачи относительно положительного фронта CLK		14

Табл. 13-18. Временные параметры сигналов глобального (локального) интерфейса при работе с общей памятью (Продолжение)

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
10.	$t_{dis(CKR-OEZH)}$	Задержка перехода вывода $\overline{(L)OE}$ из высокоимпедансного состояния в режим выдачи высокого уровня относительно положительного фронта CLK		11.5
11.	$t_{dis(CKR-ZA)}$	Задержка переключения шины адреса из высокоимпедансного состояния в режим выдачи относительно положительного фронта CLK		14
12.	$t_{d(RDYL-HOLDOH)}$	Задержка выдачи положительного фронта сигнала $\overline{(L)HOLDO}$ относительно отрицательного фронта $\overline{(L)RDY}$ в режиме работы с общей памятью 00	1.5P	2.5P+12.5
13.	$t_{d(RDYL-HOLDOH)}$	Задержка выдачи положительного фронта сигнала $\overline{(L)HOLDO}$ относительно отрицательного фронта $\overline{(L)RDY}$ в режиме работы с общей памятью 01	P	2P+12.5
14.	$t_{d(RDYL-HOLDOH)}$	Задержка выдачи положительного фронта сигнала $\overline{(L)HOLDO}$ относительно отрицательного фронта $\overline{(L)RDY}$ в режиме работы с общей памятью 10	2P	3P+12.5
15.	$t_{su(HOLDI)}$	Время предустанова сигнала $\overline{(L)HOLDI}$ относительно отрицательного фронта тактового сигнала CLK	15	
16.	$t_{h(HOLDI)}$	Время удержания сигнала $\overline{(L)HOLDI}$ относительно отрицательного фронта CLK	10	
17.	$t_{d(CKR-RDYZL)}$	Задержка перехода вывода $\overline{(L)RDY}$ из высокоимпедансного состояния в режим выдачи низкого уровня относительно положительного фронта CLK		12.5
18.	$t_{d(CKF-RDYLH)}$	Задержка перехода вывода $\overline{(L)RDY}$ с выдачи низкого уровня на выдачу высокого относительно отрицательного фронта CLK		12.5

Табл. 13-18. Временные параметры сигналов глобального (локального) интерфейса при работе с общей памятью (Продолжение)

Номер	Обозначение	Функциональное описание	Временной параметр, нс	
			не менее	не более
19.	$t_{dis(CKR-RDYHZ)}$	Задержка перехода вывода $\overline{(L)RDY}$ с выдачи высокого уровня в высокоимпедансное состояние относительно положительного фронта CLK		7.5
20.	$t_{d(CKR-RASHZ)}$	Задержка перехода выводов $\overline{(L)RAS0}/\overline{(L)CS0}$ и $\overline{(L)RAS1}/\overline{(L)CS1}$ с выдачи высокого уровня в высокоимпедансное состояние относительно положительного фронта CLK		13
21.	$t_{dis(CKR-CASHZ)}$	Задержка перехода выводов $\overline{(L)CASH}/\overline{(L)WEH}$ и $\overline{(L)CASL}/\overline{(L)WEL}$ с выдачи высокого уровня в высокоимпедансное состояние относительно положительного фронта CLK		11.5
22.	$t_{dis(CKR-WEZ)}$	Задержка перехода вывода $\overline{(L)WE}$ с режима выдачи в высокоимпедансное состояние относительно положительного фронта CLK		11.5
23.	$t_{dis(CKR-OEHZ)}$	Задержка перехода вывода $\overline{(L)OE}$ с выдачи высокого уровня в высокоимпедансное состояние относительно положительного фронта CLK		12
24.	$t_{dis(CKR-AZ)}$	Задержка переключения шины адреса в высокоимпедансное состояние относительно положительного фронта CLK		11.5
25.	$t_{d(HOLDIL-RDYL)}$	Задержка выдачи отрицательного фронта сигнала $\overline{(L)RDY}$ относительно отрицательного фронта $\overline{(L)HOLDI}$ в режимах работы с общей памятью 00, 01, 10	0.5P	1.5P+12.5
26.	$t_{d(HOLDIH-RDYH)}$	Задержка выдачи положительного фронта сигнала $\overline{(L)RDY}$ относительно положительного фронта $\overline{(L)HOLDI}$ в режимах работы с общей памятью 00, 01, 10		P+12.5

Примечание: *P - период тактового сигнала CLK.*

Система команд NeuroMatrix® NM6403

А.1 Форматы обрабатываемых данных

Процессор NM6403 может работать с данными следующих трех форматов:

- 32-х разрядные целые числа со знаком, представленные в дополнительном коде;
- упакованные в 64-х разрядном слове коды с разрядностью от 1 до 64-х;
- вектора 64-х разрядных слов с упакованными кодами.

В случае упакованных кодов разрядность каждого отдельного кода может различаться, границы

кодов в данном случае определяются содержимым регистра синапсов SB2 или регистра нейронов NB2 (см. раздел 4.1).

А.2 Система команд процессора NM6403

Процессор NM6403 работает с машинными командами 32-х и 64-х разрядного формата (см. Рис А.1, Рис А.2, Рис А.3). Обычно в одной машинной команде задаются две операции процессора NM6403. В этом смысле он представляет собой суперскалярный микропроцессор со статической VLIW-архитектурой.

Команды процессора NM6403

Множество команд процессора NM6403 можно разбить на три основные группы:

- скалярные команды;
- векторные команды;
- команды управления.

Скалярные команды - это команды, выполняющие действия со скалярными операндами. Эти команды включают несколько подгрупп:

- команды загрузки/записи регистров (LS-подгруппа);
- команды пересылки значений регистров (SM-подгруппа);

- команды обработки адресов (SA-подгруппа) и скаляров (SP-подгруппа);

специальные скалярные команды (SN-подгруппа).

Векторные команды - команды, выполняющие действия с векторными операндами, включают следующие подгруппы:

- команды загрузки данных в векторный процессор (VL-подгруппа);
- команды выгрузки данных из векторного процессора (VS-подгруппа);
- команды обработки векторов в векторном процессоре (VP-подгруппа);
- специальные векторные команды (VN-подгруппа).

Команды управлнения (JC-подгруппа) - команды, изменяющие очередность выполнения машинных команд программы, включают:

- команды безусловного и условного перехода;
- команды безусловного и условного обращения к подпрограмме;
- команды возврата из подпрограммы или прерывания.

Машинные команды процессора NM6403

Основное правило попарного объединения команд процессора NM6403 в машинных командах состоит в следующем. Возможны лишь пары вида:

- (скалярная команда, скалярная команда) ;
- (команда управления, скалярная команда);
- (векторная команда, векторная команда).

Первая команда пары хранится в левой части машинной команды, а вторая команда - в правой части. Скалярная или векторная команда, находящаяся слева, должна быть командой загрузки или записи, либо командой адресной арифметики. Формализовано это представляется так:

- (команда подгруппы LS,SM,SA,SS,JC; команда подгруппы SP);
- (команда подгруппы JC; команда подгруппы SP);
- (команда подгрупп VL,VS,VN; команда подгруппы VP).

Кроме пары команд процессора NM6403 машинная команда содержит еще специальное однобитовое поле P (31-ый разряд), управляющее одновременным выполнением команд в процессоре NM6403 на фоне выполнения векторных команд.

Г 0 - выполнение машинной команды откладывется до
I окончания уже выполняющихся векторных команд;
P = <
I 1 - машинная команда выполняется, даже если еще не
L завершены выполняющиеся векторные команды.

Содержательное обозначение команд процессора NM6403

Приведем краткие содержательные обозначения команд процессора NM6403 перечисленных групп.

Здесь и далее будем использовать следующие условные обозначения:

$f_{адр}$	-	функция вычисления адреса операнда;
f_m	-	функция модификации значения адресного регистра после обращения к памяти за операндом;
C_m	-	32-х разрядная константа, представляющая собой целое число в дополнительном коде, либо натуральное число;
$R_{ист}$	-	любой программно доступный регистр, который может использоваться в качестве источника;
$R_{пр}$	-	любой программно доступный регистр, который может использоваться в качестве приемника;
$f_{арифм}$	-	арифметическая функция обработки операндов;
$f_{логич}$	-	логическая функция обработки операндов;
$f_{сдвига}$	-	функция сдвига (арифметического, логическкого, циклического и т.д.);
<-	-	пересылка значения.

Команды загрузки/записи регистров (LS-подгруппа)

1) $R_{пр} <- (f_{адр} (AR_i , GR_i); AR_i <- f_m(AR_i , GR_i))$;
2) $R_{пр} <- (f_{адр} (AR_i , C_{im}); AR_i <- f_m(AR_i , C_m))$;
3) $R_{пр} <- C_{im}$;
4) $R_{ист} -> (f_{адр} (AR_i , GR_i); AR_i <- f_m(AR_i , GR_i))$;
5) $R_{ист} -> (f_{адр} (AR_i , C_{im}); AR_i <- f_m(AR_i , C_m))$;

Команды пересылки значений регистров (SM-подгруппа)

$R_{пр} <- R_{ист}$

Команды обработки адресов скаляров

- обработка адресов (SA-подгруппа)
  - $AR_j <- f_m(AR_i , GR_i)$;
  - $AR_j <- f_m(AR_i , C_m)$;

- обработка скаляров (SP-подгруппак)
  - GR_k <- $f_{арифм}$ (GR_i , GR_j);
  - GR_k <- $f_{логич}$ (GR_i , GR_j);
  - GR_k <- $f_{сдвига}$ (GR_i , GR_j);

Специальные скалярные команды (SN-подгруппа).

- установка слова состояния программы;
- 32-х и 64-х разрядные команды типа “нет операции”.

Векторные команды

В описываемых далее подгруппах векторных команд используются следующие обозначения в дополнение к приведенным для скалярных команд:

AFIFO	-	выходная FIFO-очередь функционального устройства выполнения матричных операций;
AFIFO.n	-	количество 64-х разрядных значений, хранящихся в AFIFO;
RAM	-	внутренняя память для временного хранения векторных операндов;
RAM.n	-	количество 64-разрядных значений, хранящихся в RAM;
WFIFO	-	внутренняя FIFO-память для хранения весовых коэффициентов;
WBUF	-	внутренняя память для хранения теневой матрицы весовых коэффициентов;
WOPER	-	внутренняя память для хранения рабочей матрицы весовых коэффициентов;
WFIFO.s	-	количество 64-разрядных слов, переписываемых за одну операцию загрузки в теневую матрицу весов WBUF из WFIFO, определяется количеством синапсов по значению регистра границ синапсов SB;
n	-	количество элементов вектора, задаваемых непосредственно в команде;
X, Y	-	обозначение векторных операндов, ими могут быть: нулевой вектор длины n, вектор из RAM длиной RAM.n, вектор из AFIFO длиной AFIFO.n, вектор из внешней памяти длиной n элементов;

M	-	обозначение вектора масок для выполняемой векторной операции, в качестве вектора масок может использоваться: вектор из RAM длиной RAM.n, вектор из AFIFO длиной AFIFO.n, вектор из внешней памяти длиной n;
FPX, FPY	-	пороговые функции обработки векторных операндов X и Y;
VR	-	регистр значения порога;
FAX, FAY	-	функции активации для обработки векторных операндов X и Y;
f_{sh}	-	функция циклического сдвига на один разряд вправо операнда X;
(V | L)	-	обозначение вектора, содержащего L элементов, V - либо обозначение внутренней памяти, из которой выбираются элементы, начиная с нулевого и с шагом 1, либо заключенное в скобки {....} адресное выражение, задающее способ вычисления адресов элементов вектора во внешней памяти.

Конструкции вида $[V_1 <-][V_2 <-]...[V_n <-]V$ или $V[->V_1][->V_2]...[->V_n]$ обозначают присваивание значения вектора V векторам V_1, V_2, ... V_n в зависимости от управляющих параметров, заданных в векторной команде.

Отметим, что по сравнению со скалярными командами, для векторных команд имеется более тесная связь по данным команд, объединенных в одной машинной команде. Основная особенность здесь состоит в следующем. Когда в команде загрузки встречается присваивание векторным операндам X, Y или M, то это означает установку соответствующих операндов для парной в машинной команде команды обработки векторов.

Рассмотрим перечисленные выше подгруппы векторных команд процессора NM6403.

Команды загрузки данных в векторный процессор (VL-подгруппа)

1) $[X<-]$ $[Y<-]$ $[M<-]$ $[RAM<-]$ $(\{f_{адр}(AR_i,GR_i);\ AR_i <-f_m(AR_i,GR_i)\}|n);$
   $[WBUF<-]$ (WFIFO | WFIFO.s)

2) $WFIFO<-$ $(\{f_{адр}(AR_i,GR_i);\ AR_i<-f_m(AR_i,GR_i)\}|n);$
   $[WBUF<-]$ (WFIFO | WFIFO.s)

Команды выгрузки данных из векторного процессора (VS-подгруппа)

(AFIFO|AFIFO.n) [->RAM] [->{$f_{адр}(AR_i, GR_i)$; AR_i<-$f_m(AR_i, GR_i)$}]

[WBUF<-] (WFIFO | WFIFO.s)

[X<-][Y<-] (0|n)

Команды обработки векторов в векторном процессоре (VP-подгруппа)

1) AFIFO<-$f_{логич}$(FPX(X), FPY(Y));

[WOPER<-] WBUF;

2) AFIFO<-$f_{арифм}$(FPX(X), FPY(Y));

3) AFIFO<-f_{sh}(FPX(X)) & M + FPY(Y) & (*M);

[WOPER<-] WBUF;

4) AFIFO<-WOPER x (f_{sh}(FAX(X)) & M) + (FAY(Y) & (*M) или VR);

Специальные векторные команды (VN-подгруппа)

- команда выдачи в AFIFO содержимого выделенных управляющих регистров AFIFO<-({F2CR, F1CR, ONB, OSB, VR}|5);
  [WOPER<-] WBUF;
- команды типа “нет операции”.

Команды управления

Команды управления образуют JC-подгруппу команд. Далее используются следующие дополнительные условные обозначения:

f_{km}	-	функция вычисления адреса перехода;
PC	-	счетчик адреса машинных команд;
COND	-	условие выполнения перехода или обращения к подпрограмме, в зависимости от признаков, хранящихся в слове состояния программы PSW;
Jump	-	передача управления;
Call	-	обращение к подпрограмме;
Ret_S	-	выход из подпрограммы;
Ret_I	-	выход из обработки прерывания.

Команды безусловного и условного перехода

1) Jump f_{km} (AR_i, GR_i, PC)

2) Jump f_{km} (AR_i, C_m, PC)

3) Jump COND, f_{km} (AR_i, GR_i, PC)

4) Jump COND, f_{km} (AR_i, C_m, PC)

Команды безусловного и условного обращения к подпрограмме

1) Call f_{km} (AR_i, GR_i, PC)

2) Call f_{km} (AR_i, C_m, PC)

3) Call COND, f_{km} (AR_i, GR_i, PC)

4) Call COND, f_{km} (AR_i, C_m, PC)

Команды возврата из подпрограммы или прерывания

1) Ret_S

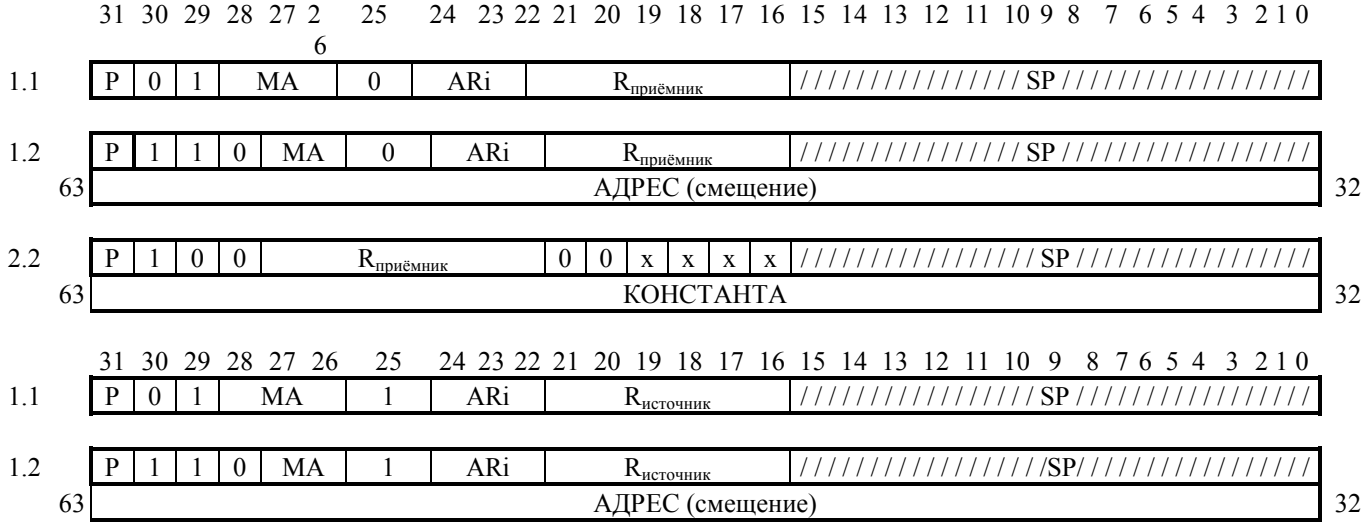
2) Ret_S COND

3) Ret_I

4) Ret _I COND

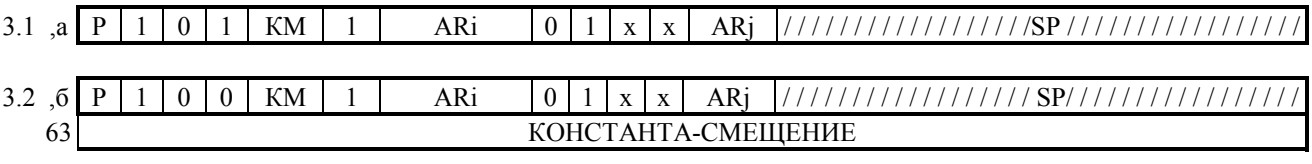
Рис А.1. Форматы скалярных команд

- команды загрузки/записи регистров (LS-полгруппа)



- команда пересылки значений регистров (SM-подгруппа)

- команды обработки адресов и скаляров - адресная обработка (SA-подгруппа)



- обработка скаляров (SP-подгруппа)

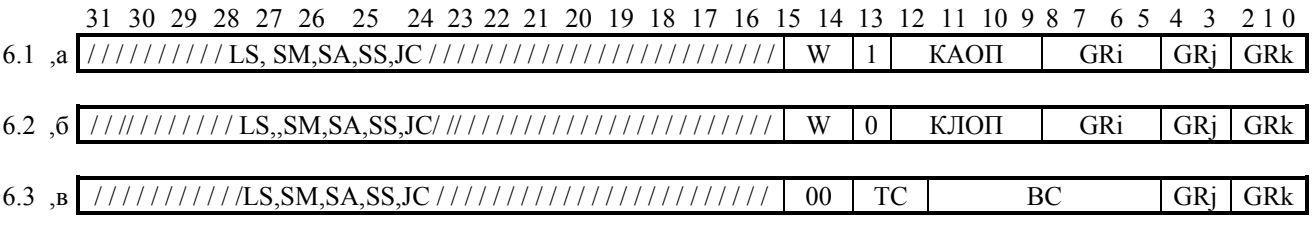
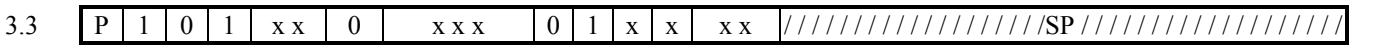


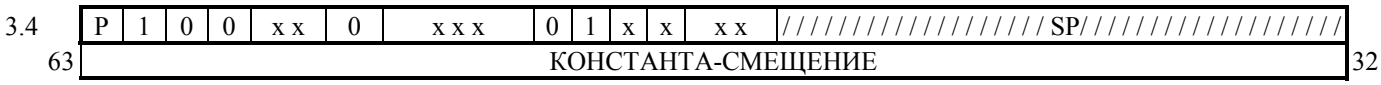
Рис.А.1. Форматы скалярных команд (Продолжение)

- специальные скалярные команды (SN-подгруппа) - установка битов PSW

- 32-разрядная пустая команда



- 64-разрядная пустая команда

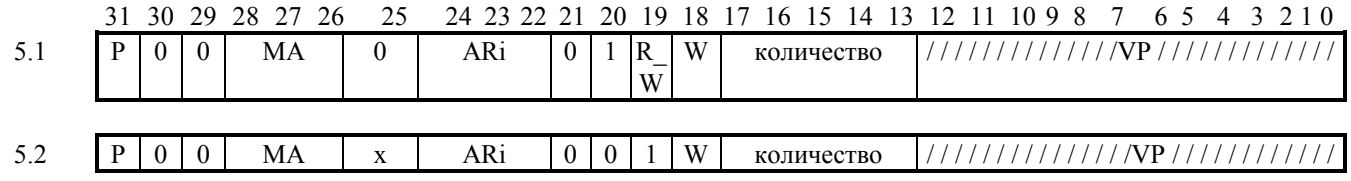


Условные обозначения:

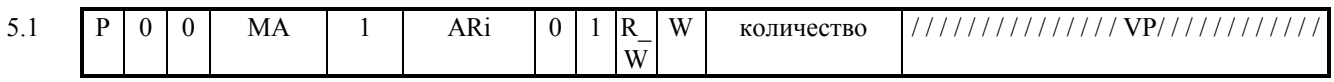
Р	-	признак совмещения с выполнением векторных команд
МА	-	кодировка выбора функций $f_{адр}$ и f_m ; ;
ФА	-	кодировка выбора функции f_m ;
КАОП	-	код арифметической функции;
КЛОП	-	код логической функции;
ВС	-	величина сдвига;
ТС	-	тип сдвига.

Рис А.2. Форматы векторных команд

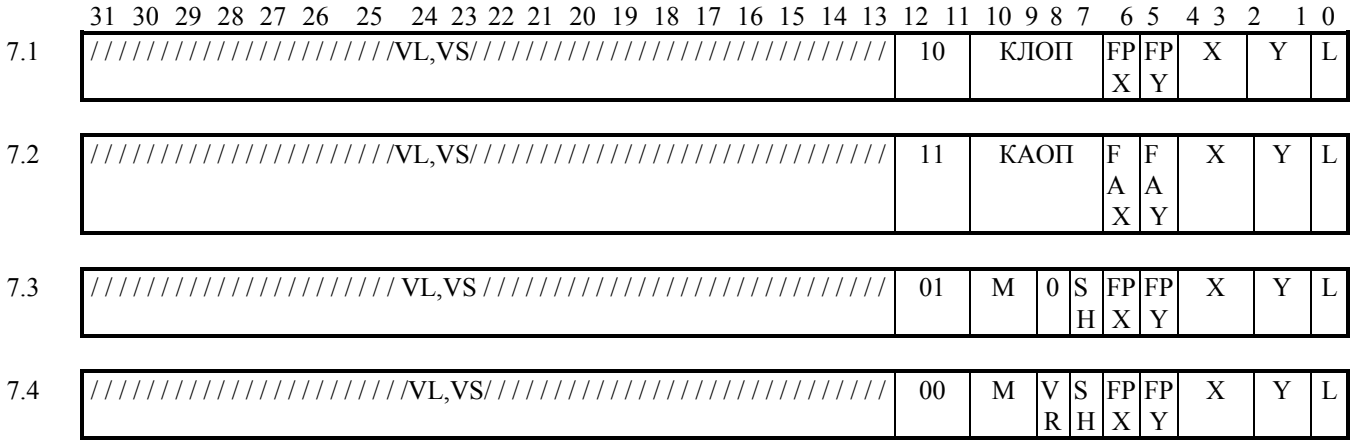
- команды загрузки данных в векторный процессор (VL-подгруппа)



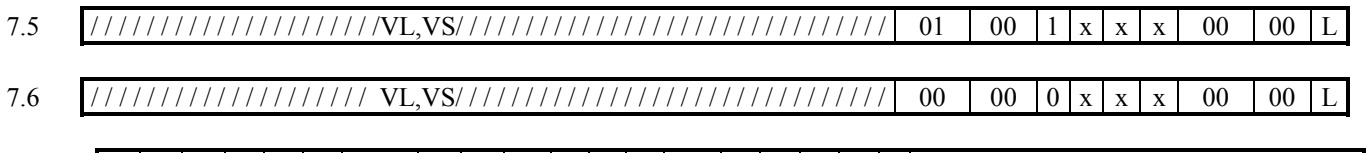
- команды выгрузки данных из векторного процессора (VS-подгруппа)



- команды обработки векторов в векторном процессоре (VP-подгруппа)



- специальные векторные команды (VN-подгруппа)



Условные обозначения:

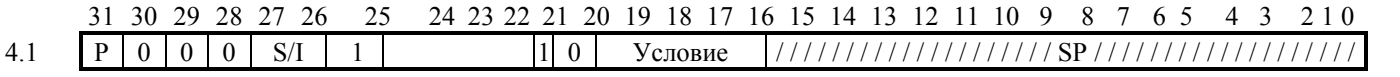
P	-	признак параллельного выполнения с векторными командами;
MA	-	кодировка выбора функций $f_{адр}$ и f_m ; ;
ФА	-	кодировка выбора функции f_m ;
КЛОП	-	код логической операции;
КАОП	-	код арифметической операции;
M	-	поле выбора операнда-маски;
VR	-	управление выборкой регистра VR в качестве операнда Y;
SH	-	управление циклическим сдвигом операнда X на 1 разряд вправо.

Рис А.3. Форматы команд управления

- безусловные и условные передачи управления.

- безусловные и условные обращения к подпрограммам.

- возврат из подпрограммы или прерывания



Условные обозначения:

P	-	признак выхода из совмещения с выполнением векторных команд на данной команде;
KM	-	способ вычисления адреса перехода;
S/I	-	тип возврата (возврат из подпрограммы или выход из прерывания).

Форматы, кодировка и краткое описание команд процессора NM6403

Форматы команд процессора NM6403 приведены Рис А.1, Рис А.2, Рис А.3. Слева от каждого формата указывается его номер.

Полные описания полей форматов команд даны на Рис А.4 - Рис А.16.

Команды загрузки / записи регистров (LS-подгруппа)

Кодировка команд этой подгруппы представлена на Рис А.4, Рис А.5 и Рис А.6.

Команды $R_{пр}$ <- ($f_{адр.}(AR_i, GR_i)$; $AR_i \leftarrow f_m(AR_i, GR_i)$) и

$R_{исп}$ -> ($f_{адр.}(AR_i, GR_i)$; $AR_i \leftarrow f_m(AR_i, Gr_i)$)

Кодировка этих команд дана на Рис А.4, команды имеют формат 1.1.

Бит 25 содержит признак загрузки или признак записи во внешнюю память.

В качестве $R_{ист}$ и $R_{пр}$ может использоваться любой программно доступный регистр, однако при этом следует учитывать доступность

этого регистра на чтение или запись.

Формат операнда из памяти может быть 32-х разрядным, либо 64-х разрядным. Он определяется разрядностью регистров $R_{ист}$ и $R_{пр}$. В зависимости от формата операнда устанавливается параметр “a” в функциях $f_{адр}$ и f_m (см. кодировку поля MA на Рис А.4).

При обращении к памяти за операндом 64-х разрядного формата младший разряд адреса игнорируется. Размещение старших и младших 32-х разрядных слов после в регистре при этом после чтения из памяти не изменяется.

Заданный в команде AR-регистр (поле ARi) дополнительно неявно указывает номер функционального адресного устройства, на котором производится вычисление функций $f_{адр}$ и

$f_{ист}$: AR0, ... AR3 - адресное устройство 0; AR4, ... AR7 - адресное устройство 1.

Вычисления функций $f_{адр}$ и f_m производятся по модулю 2^{32}. При этом следует помнить, что локальная внешняя память имеет адреса 0, 1, ... 2^{31}-1, а глобальная внешняя память - адреса 2^{31}, 2^{31}+1 ,...2^{32}-1.

Если в команде в качестве $R_{ист}$ или $R_{пр}$, либо в качестве GRi для адресных вычислений используется GR-регистр, который каким-либо образом задействован в парной для данной команде SP-подгруппы, то необходимо учитывать следующее. Чтение GR-регистра производится до установки значения этого регистра в парной команде SP-подгруппы, а установка значения этого регистра - после установки значения в парной команде SP-подгруппы.

Если в команде в качестве $R_{пр}$ задан регистр слова состояния программы PSW, а в парной команде SP-подгруппы производится установка признаков N, Z, V, C (это биты PSW), то результирующим значением PSW будет значение, считанное из памяти.

Команды $R_{пр}$ <- ($f_{адр}$ (AR_i, $C_ь$); $AR_i \leftarrow f_m(AR_i, C_ь)$) и
$R_{исп}$ -> ($f_{адр}$ (AR_i, $C_ь$); $AR_i \leftarrow f_m(AR_i, C_ь)$)

Кодировка этих команд дана на Рис А.5, команды имеют формат 1.2.

Бит 25 содержит признак загрузки или признак записи во внешнюю память.

Функции $f_{адр}$ и f_m, применяемые в данной команде, отличаются от соответствующих функций команд загрузки / записи, описанных выше.

Рассматриваемые команды позволяют обращаться к операндам из внешней памяти по абсолютному адресу (MA=00 и MA=10) без изменения заданного в команде AR-регистра (MA=00) и с его изменением (MA=10). При использовании обращений к памяти со смещением относительно значения заданного AR-регистра, которое

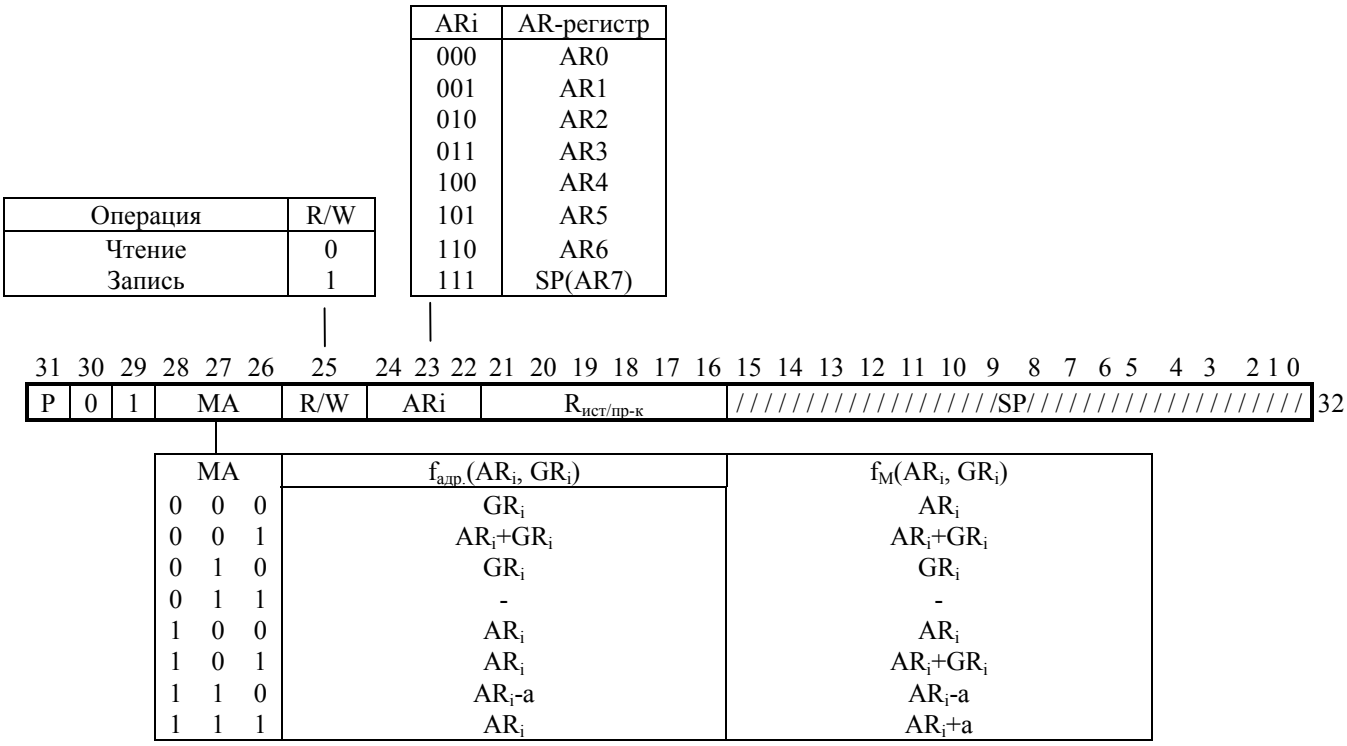
может быть как положительным, так и отрицательным (MA=01), следует помнить, что значение AR-регистра после выполнения команды изменяется.

В остальном выполнение данных команд не отличается от выполнения команд загрузки / записи, рассмотренных ранее.

Команда $R_{пр}$ <- $C_{ь}$

Кодировка этой команды приведена на Рис А.6, команда имеет формат 2.2.

Рис А.4. Кодировка команд LS-подгруппы $R_{пр}$ <- ($f_{адр.}(AR_i, GR_i)$; $AR_i \leftarrow f_m(AR_i, GR_i)$) и $R_{исп}$ -> ($f_{адр.}(AR_i, GR_i)$; $AR_i \leftarrow f_m(AR_i, GR_i)$) , формат 1.1 .



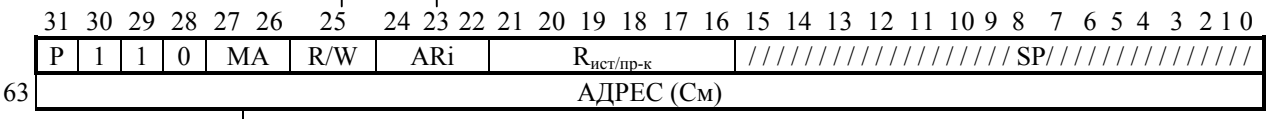
Примечание: *a=1 при адресации 32-разрядных данных*

a=2 при адресации 64-разрядных данных.

Рис А.5. Кодировка команд LS-подгруппы Rпр ← ($f_{адр.}(AR_i, Cm)$); $AR_i \leftarrow f_m(AR_i, Cm)$ и Rист → ($f_{адр.}(AR_i, Cm)$); $AR_i \leftarrow f_m(AR_i, Cm)$, формат 1.2

ARi	AR-регистр
000	AR0
001	AR1
010	AR2
011	AR3
100	AR4
101	AR5
110	AR6
111	SP(AR7)

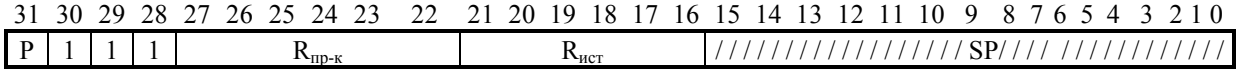
Операция	R/W
Чтение	0
Запись	1



MA	$f_{адр.}(AR_i, C_M)$	$f_M(AR_i, C_M)$
0 0	Cm	AR_i
0 1	AR_i+C_M	AR_i+C_M
1 0	Cm	Cm
1 1	-	-

Рис А.6. Кодировка команды LS-подгруппы Rпр ← Cm, формата 2.2

Рис А.7. Кодировка команды SM-подгруппы Rпр<- Rист, формат 2.1



В качестве $R_{пр}$ может использоваться любой доступный на запись регистр. Если $R_{пр}$ имеет 32-х разрядный формат, то после выполнения команды его значение становится равным заданной в команде константе $C_ь$. Если $R_{пр}$ имеет 64-х разрядный формат, то после выполнения команды константа располагается как в старших, так и в млажших разрядах этого регистра.

Если $R_{пр}$ - это GR-регистр, используемый в парной команде SP-подгруппы, то для вычислений в этой парной команде используется старое значение GR-регистра, а если данный GR-регистр является регистром результата данной команды, то после выполнения записанный в него результат заменяется значением константы $C_ь$.

Если $R_{пр}$ - это регистр, слова состояния программы PSWR, то

независимо от выполнения парной команды SP-подгруппы, значение PSWR после выполнения всей машинной команды будет равным $C_ь$. Вместе с тем, если при выполнении команды SP-подгруппы возникла какая-либо исключительная ситуация, которая не была замаскирована в старом значении PSWR, то прерывание возникнет.

Команды пересылки значений регистров (LS-подгруппа)

Команды $R_{пр}$ <- $R_{ист}$

Кодировка этой команды представлена на Рис А.7, формат команды - 2.1.

В качестве $R_{ист}$ и $R_{пр}$ могут использоваться любые прграммно доступные для записи и чтения регистры 32-х и 64-х разрядного формата.

Если значение 64-х разрядного регистра пересылается в 32-х разрядный регистр, то передаются младшие или старшие 32 разряда, в зависимости от принадлежности $R_{пр}$ к группе L или H.

Если значение 32-х разрядного регистра пересылается в 64-х разрядный регистр, то это значение одновременно устанавливается в старших и младших 32-х разрядах.

Если $R_{ист}$ или $R_{пр}$ - это GR-регистр, задействованный в парной команде SP-подгруппы, то правила использования и установки его значения такие же, как для команд загрузки/записи регистров: обе парные команды считывают значение GR-регистра, бывшее до начала выполнения машинной команды; сначала устанавливается значение GR-регистра парной команды SP-подгруппы, потом - записывается значение GR-регистра командой пересылки.

Если $R_{пр}$ - регистр слова состояния программы PSWR, то независимо от выполнившейся парной команды SP-подгруппы, конечное значение PSWR будет равным переданному из $R_{ист}$ значению.

Команды обработки адресов и скаляров

Команды AR_j <-$f_m(AR_i , GR_i)$ и AR_j <-$f_m(AR_i , C_m)$ (SA-подгруппа)

Кодировка этих команд дана на Рис А.8, команды имеют соответственно форматы 3.1 и 3.2.

В командах данного формата регистр-приемник $AR_{пр}$ кодируется 2-х битовым полем. Это связано с тем, что непосредственно на запись к каждому функциональному устройству адресных вычислний подключено лишь по четыре адресных регистра. Номер функционального адресного устройства задается в команде неявно, по номеру регистра-источника AR_i, используемого при формировании адреса.

Адресные вычисления производятся по модулю 2^{32}.

Если в команде используется GR-регистр (формат 3.1), то считывается его значение, бывшее до выполнения команды, независимо от парной команды SP-подгруппы.

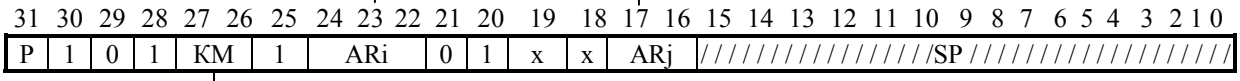
Рис А.8. Кодировка команд SA-подгруппы AR_j<-$f_m(AR_i, GR_i)$ и AR_j<-$f_m(AR_i, C_m)$, формата 3.1. и 3.2.

AR_j <- f_m(AR_i , GR_i)

AR-регистр	ARi
AR0	000
AR1	001
AR2	010
AR3	011
AR4	100
AR5	101
AR6	110
SP(AR7)	111

ARj	AR-регистр	Примечание
00	AR0	Для ARi = 000
01	AR1	001
10	AR2	010
11	AR3	011
00	AR4	Для ARi = 100
01	AR5	101
10	AR6	110
11	SP(AR7)	111

3.1



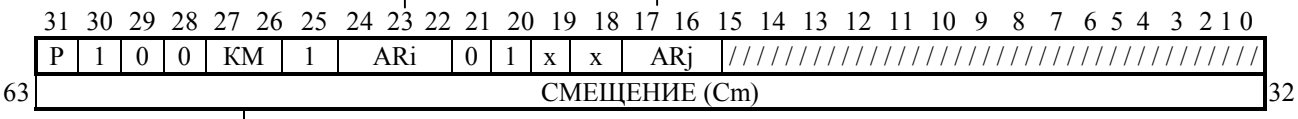
KM	f_M(ARi, GR_i)
0 0	ARi
0 1	AR_i+GR_i
1 0	GR_i
1 1	-

AR_j <- f_m(AR_i , C_m)

AR-регистр	ARi
AR0	000
AR1	001
AR2	010
AR3	011
AR4	100
AR5	101
AR6	110
SP(AR7)	111

ARj	AR-регистр	Примечание
00	AR0	Для ARi = 000
01	AR1	001
10	AR2	010
11	AR3	011
00	AR4	Для ARi = 100
01	AR5	101
10	AR6	110
11	SP(AR7)	111

3.2



KM	f_M(ARi, C_M)
0 0	ARi
0 1	ARi+C_M
1 0	C_M
1 1	-

Команды обработки скаляров (SP-подгруппа)

Команды GR_k <- $f_{арифм}$ (GR_i, GR_j)

Кодировка команд изображена на Рис А.9, команда имеет формат 6.1.

Команды производят арифметические операции, заданные полем КАОП (функция $f_{арифм}$), над значениями GR_i и GR_j, вырабатывают результат, который может быть записан в GR-регистр GR_k, а также признаки для установки соответствующих битов слова состояния программы PSWR.

Запись результата в GR_k и признаков в PSWR управляется полем W (биты 15, 14).

Вырабатываемые командой признаки могут устанавливаться в однобитовых полях N, Z, V и C регистра PSWR.

Признак N равен 31-му разряду значения результата операции (знаковый разряд).

Признак Z равен единице, если все разряды результата нулевые, в противном случае он равен нулю.

Признак V равен нулю для операций шага умножения, для остальных операций он формируется как результат операции “исключающее или” бита переноса из 31-го разряда и бита переноса в 31-й разряд.

Признак C равен биту переноса из 31-го разряда для всех операций, за исключением шага умножения. Для операции шаг умножения признак C равен 1-му разряду множителя, т.е. значения регистра GR7 (см. далее)

В обозначениях арифметических операций “C” - это значение поля C регистра PSWR до начала выполнения команды.

Операция вычитания производится следующим образом. Сначала вырабатывается дополнительный обратный код вычитаемого (GR_j), далее он складывается с первым операндом (GR_i) . Признак C устанавливается по биту переноса из 31-го разряда, возникшего в результате такого сложения. Таким образом, если из 0 вычесть 1, то признак C будет равен 0, т.е. бит C для операции вычитания не является “битом заема”. Можно показать, что бит C - это инвертированный бит заема.

Операция GR_j - GR_i -1 +“C” может использоваться при реализации операции вычитания над целыми числами, состоящими из нескольких 32-х разрядных слов (длинные целые с основанием 2^{32}).

Операция GR_j + “C” может применяться при реализации операции увеличения на единицу длинных целых с основанием 2^{32}.

Операция GR_j - 1 + “C” предназначена для реализации операции уменьшения на единицу длинных целых с основанием 2^{32}.

Операция $GR_j + GR_i + “C”$ предназначена для реализации операции сложения длинных целых с основанием 2^{32}.

Дополнительного пояснения требуют также операции “Первый шаг умножения” (MLF) и “Шаг умножения” (MLS).

Для одной и другой операции умножения множитель должен находиться в регистре GR7, этот регистр в команде явно не указывается.

Множимое всегда размещается в регистре GR_i, он не должен быть регистром GR7.

Результат умножения размещается в регистре GR_k (старшие разряды) и в регистре GR7 (младшие разряды). Заданный в команде регистр GR_j должен совпадать с регистром GR_k, однако GR_j и GR_k не должны совпадать с GR_i.

Операции MLF и MLS реализуют шаги умножения на 2 разряда множителя, начиная с младших разрядов, по алгоритму Бута.

Операция MLF производит специальное умножение значения GR_i на два младших разряда GR7, полученный 34-х разрядный результат записывается в GR_k (32 старших разряда), а в разряды 31, 30 регистра GR7 - два младших разряда результата. Перед записью в GR7 значение этого регистра предварительно сдвигается на 2 разряда вправо. После выполнения операции MLF признак C равен разряду 1 регистра GR7 до его сдвига вправо на 2 разряда. Все значения, участвующие в операции, рассматриваются как целые числа в дополнительном коде.

Операция MLS производит умножение значения GR_i на два младших разряда GR7, сложенных со значением признака C, установленного ранее выполненной операцией MLF или MLS. Полученное значение складывается со значением из GR_k, вычисленным ранее операциями MLF и MLS. Результат такого сложения, содержащий 34 разряда, записывается в GR_k (32 старших разряда) и в разряды 31, 30 - два младших разряда. Перед записью в GR7 его значение предварительно сдвигается на 2 разряда вправо. После выполнения операции MLS признак C равен разряду 1 регистра GR7 до его сдвига вправо на 2 разряда. Все действия производятся с числами в дополнительном коде.

Обычная операция умножения 32-х разрядного числа на 2X n-разрядное, где n=1, 2 ... 16, реализуется цепочкой команд, состоящей из команды MLF, за которой следует (n-1) команда MLS. Значение полей GR_i , GR_j , GR_k в этих командах должны совпадать. Результатом выполнения такой цепочки команд является целое число разрядности 32+2n, его старшие разряды размещаются в регистре GR_k , а 2n младших разрядов - в старших разрядах регистра GR7.

Для всех операций MLS и MLF необходимо, чтобы разряды поля W

(биты 15, 14) оба принимали единичное значение.

Команды GR_k <- $f_{логич}$ (GR_i, GR_j)

Кодировка команд изображена на Рис А.10, команда имеет формат 6.2.

Команды производят логические операции, заданные полем КЛОП (функция $f_{логич}$), над значениями GR-регистров GR_i и GR_j, вырабатывают результат, который может быть записан в GR-регистр GR_k, а также признаки для установки соответствующих битов слова состояния программы PSWR.

Запись результата в GR_k и признаков в PSWR управляется полем W (биты 15, 14).

Вырабатываемые командой признаки могут устанавливаться в однобитовых полях N, Z, V и C регистра PSWR.

Признак N равен 31-му разряду значения результата операции (знаковый разряд).

Признак Z равен единице, если все разряды результата нулевые, в противном случае он равен нулю.

Признаки V и C всегда равены нулю.

В обозначениях кодов логических операций (см. Рис А.10) используются обозначения: & - по-разрядное логическое “И” ; + - по-разрядное логическое “ИЛИ” ; $\oplus$ - по-разрядное сложение по модулю 2 ; $\overline{GR_{ист2}}$ - инвертирование разрядов операнда.

Команды GR_k <- $f_{сдвига}$ (GR_i, GR_j)

Кодировка команд изображена на Рис А.11, команды имеют формат 6.3.

Команды выполняют различные операции сдвига влево и вправо операнда GR_i на явно заданное в команде количество разрядов. Результат выполнения операции записывается в регистр GR_k, устанавливаются признаки N, Z, V и C регистра PSWR.

Тип сдвига задается в поле ТС (биты 13, 12) , а величина сдвига - в поле h (биты 11, 10,... 6). Положительные h задают сдвиг влево, отрицательные - вправо.

Различные типы сдвига на один разряд изображены на Рис А.12. Сдвиги на большее число разрядов эквивалентны многократному сдвигу на единицу, хотя и выполняются за один такт работы процессора NM6403.

Логический сдвиг через “C” может выполняться только на один разряд влево или вправо. Сдвиги остальных типов могут выполняться на любое от 1 до 31 число разрядов влево или вправо.

Выполнение команд сдвига с величиной сдвига от 1 до 31 разряда приводит к изменению признаков слова состояния программы по

следующим правилам.

Признак N равен 31-му разряду значения результата операции (знаковый разряд).

Признак Z равен единице, если все разряды результата нулевые, в противном случае он равен нулю.

Признак V устанавливается в единицу или ноль только для операции арифметического сдвига влево. Для остальных операций он всегда устанавливается в ноль. После выполнения операции арифметический сдвиг влево признак V устанавливается равным единице, если признак C не равен 31-му разряду результата (знаковому). В остальных случаях он равен нулю.

Признак C равен последнему вытолкнутому при сдвиге разряду значения операнда.

Сдвиг на 0 или 32 разряда любого типа воспринимается как код “нет операции”, при этом не изменяется ни приемник результата операции, ни признаки.

Специальные скалярные команды (SN-подгруппа)

Кодировка команд дана на Рис A.13, команды имеют форматы 2.3, 3.3, 6.3 и 3.4.

Команда установки/сброса выделенных битов PSWR позволяют установить в единицу, либо обнулить биты, соответствующие единичным разрядам “маски изменяемых битов”, заданной непосредственно в команде. Поле R/S (бит 27) задает изменение выделенного битом маски разряда - обнуление его или установка в единицу.

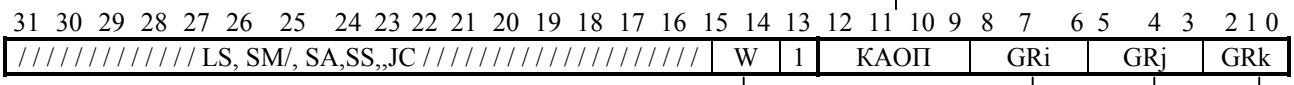
Команда установки/сброса выделенных битов PSWR фиксирует новое значение PSWR после парной с ней команды SP-подгруппы, которая, однако, выполняется под управлением значения PSWR, бывшего до начала выполнения команды.

32-х разрядные пустые команды позволяют заполнять левую и/или правую половину 32-х разрядной машинной команды командой, которая не производит никаких действий. При этом важно отметить, что эти команды не требуют для своего выполнения никаких ресурсов.

64-х разрядная пустая команда может потребоваться, если в каком-либо участке программы необходимо чем-то заполнить 64-разряда, а подходящих для этого содержательных команд в программе нет. Например, это может использоваться после команд управления. 64-х разрядная пустая команда не производит никаких действий и не требует для своего выполнения ресурсов. Значение константы, заданное непосредственно в команде, ни на что не влияет.

Рис А.9. Кодировка команд SP-подгруппы GRk←$f_{арифм}$(GRi, GRj), формат 6.1

N	Z	V	C	Код арифметической операции	КАОП
+	+	+	+	$GR_{ист2}$-$GR_{ист1}$	0 0 0 0
+	+	+	+	$GR_{ист2}$-$GR_{ист1}$-1+"C"	0 0 0 1
+	+	+	+	$GR_{ист2}$+1	0 0 1 0
+	+	+	+	$GR_{ист2}$+"C"	0 0 1 1
+	+	+	+	$GR_{ист2}$-1	0 1 0 0
+	+	+	+	$GR_{ист2}$-1+"C"	0 1 0 1
+	+	+	+	$GR_{ист2}$+ $GR_{ист1}$	0 1 1 0
+	+	+	+	$GR_{ист2}$+ $GR_{ист1}$+"C"	0 1 1 1
?	?	0	+	Первый шаг умножения (MLF)	1 0 0 0
?	?	0	+	Шаг умножения (MLS)	1 0 0 1
				Резерв	1 0 1 X
+	+	+	+	-$GR_{ист2}$	1 1 0 0
				Резерв	1 1 X 1
				Резерв	1 1 1 X



Управление записью в GRпр-к и в регистр признаков	W
Есть запись в GRпр-к, нет записи признаков	01
Есть запись признаков, нет записи в GRпр-к	10
Есть запись в GRпр-к и есть запись признаков	11

GR - регистр	GR
GR0	000
GR1	001
GR2	010
GR3	011
GR4	100
GR5	101
GR6	110
GR7	111

Условные обозначения:

0 - признак обнуляется;

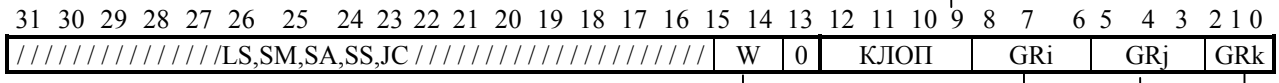
1 - признак устанавливается в 1;

\+ - признак устанавливается по результату операции;

? - признак не определен.

Рис А.10. Кодировка команд SP-подгруппы GRk←$f_{логич}$(GRi, GRj) формата 6.2

N	Z	V	C	Код логической операции	КЛОП
0	1	0	0	0	0 0 0 0
+	+	0	0	$\overline{GR_{ucm2}}$ & $\overline{GR_{ucm1}}$	0 0 0 1
+	+	0	0	GR_{ucm2} & $\overline{GR_{ucm1}}$	0 0 1 0
+	+	0	0	$\overline{GR_{ucm1}}$	0 0 1 1
+	+	0	0	$\overline{GR_{ucm2}}$ & GR_{ucm1}	0 1 0 0
+	+	0	0	$\overline{GR_{ucm2}}$	0 1 0 1
+	+	0	0	$GR_{ucm2} \oplus GR_{ucm1}$	0 1 1 0
+	+	0	0	$\overline{GR_{ucm2}} + \overline{GR_{ucm1}}$	0 1 1 1
+	+	0	0	GR_{ucm2} & GR_{ucm1}	1 0 0 0
+	+	0	0	$GR_{ucm2} \oplus \overline{GR_{ucm1}}$	1 0 0 1
+	+	0	0	GR_{ucm2}	1 0 1 0
+	+	0	0	$GR_{ucm2} + \overline{GR_{ucm1}}$	1 0 1 1
+	+	0	0	GR_{ucm1}	1 1 0 0
+	+	0	0	$\overline{GR_{ucm2}} + GR_{ucm1}$	1 1 0 1
+	+	0	0	$GR_{ucm2} + GR_{ucm1}$	1 1 1 0
1	0	0	0	-1	1 1 1 1



Управление записью в GRпр-к и в регистр признаков	W
Есть запись в GRпр-к, нет запи*си признаков	01
Есть запись признаков, нет записи в GRпр-к	10
Есть запись в GRпр-к и есть запись признаков	11

GR - регистр	GR
GR0	000
GR1	001
GR2	010
GR3	011
GR4	100
GR5	101
GR6	110
GR7	111

Условные обозначения:

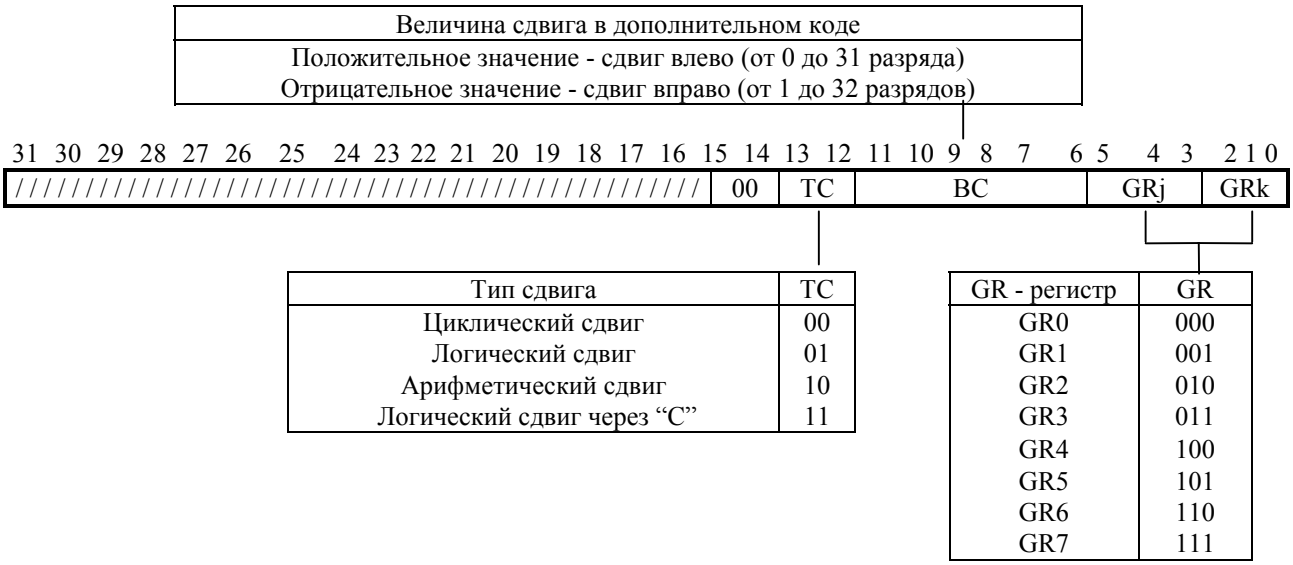
0 - признак обнуляется;

1 - признак устанавливается в 1;

\+ - признак устанавливается по результату операции;

? - признак не определен.

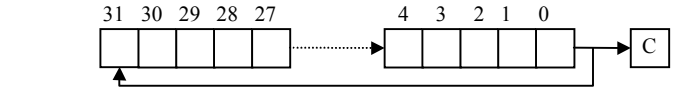
Рис А.11. Кодировка команд SP-подгруппы GRk←$f_{сдвиг}$(GRi, GRj) формата 6.3



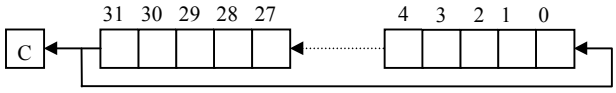
Тип сдвига	TC
Циклический сдвиг	00
Логический сдвиг	01
Арифметический сдвиг	10
Логический сдвиг через “C”	11

GR - регистр	GR
GR0	000
GR1	001
GR2	010
GR3	011
GR4	100
GR5	101
GR6	110
GR7	111

Рис А.12. Варианты сдвигов, выполняемые командой GR_k<-$f_{сдвига}(GR_i, h)$ формата 6.3



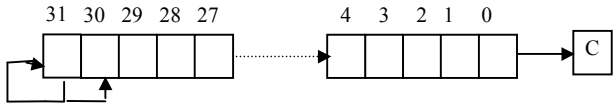
Циклический сдвиг вправо



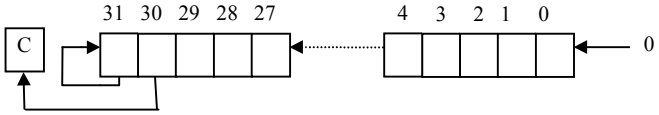
Циклический сдвиг влево

Логический сдвиг вправо

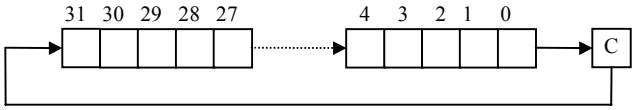
Логический сдвиг влево



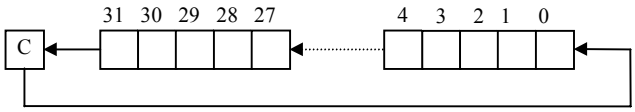
Арифметический сдвиг вправо



Арифметический сдвиг влево

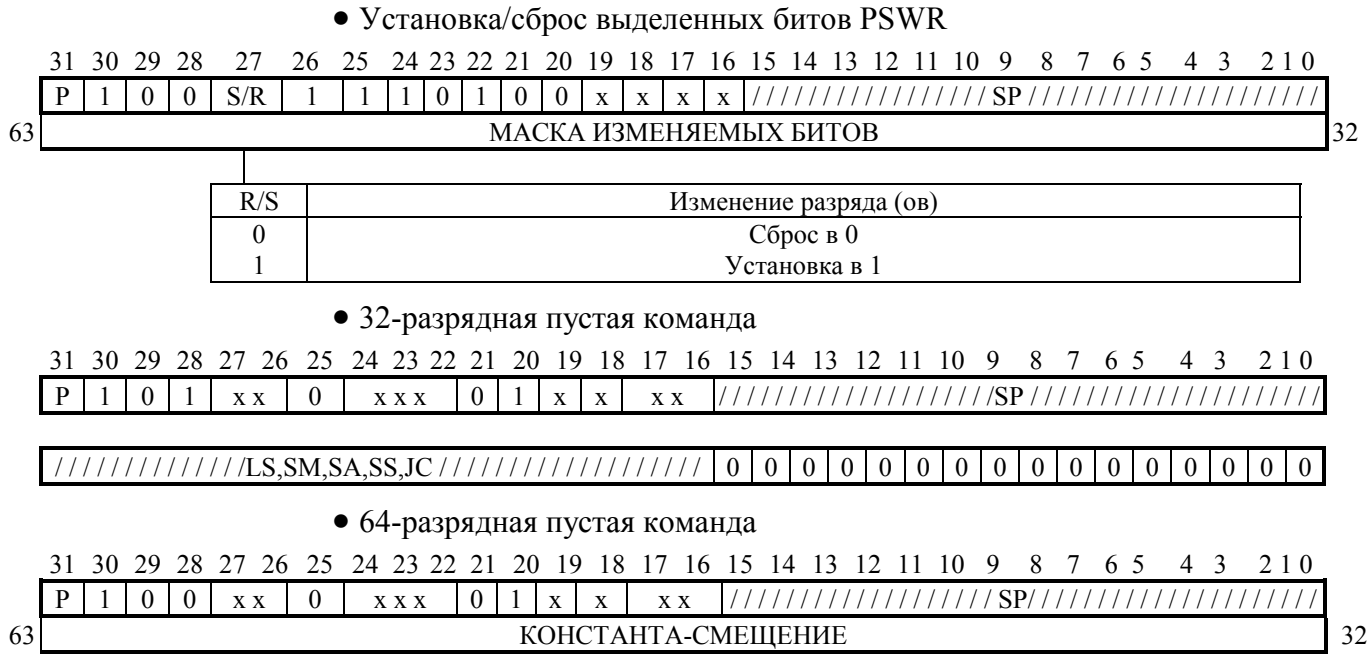


Логический сдвиг вправо через “C”



Логический сдвиг влево через “C”

Рис А.13. Кодировка специальных скалярных команд форматов 2.3, 3.3, 6.3 и 3.4



Векторные команды

Далее рассматриваются векторные команды, т.е. команды, выполняющие действия с векторами 64-х разрядных чисел.

Общими для всех векторных команд являются правила работы с AFIFO, WFIFO и RAM. Полная информация об этих внутренних памятях дана в главе 4. В сжатом виде, основные правила работы с этими объектами таковы.

AFIFO (Accumulator First-In-First-Out)

1) Чтение AFIFO представляет собой последовательное считывание с удалением 64-х разрядных значений из AFIFO, начиная с записанного самым первым.

2) Запись в AFIFO представляет собой последовательное добавление в AFIFO 64-х раз-рядных значений , начиная с первого свободного элемента AFIFO.

3) Чтение из пустого AFIFO всегда запрещено.

4) Запись в непустое AFIFO возможна лишь в том случае, если в этой же машинной команде происходит чтение из AFIFO.

5)Состояние AFIFO отслеживается в регистре запросов на прерывания INTR:

Бит 23 - признак “непустое AFIFO” ,

0 - если AFIFO пусто,

1 - если AFIFO непусто;

Бит 22-18 - количество записанных в AFIFO 64-х разрядных слов, 0 - одно слово,

1 - два слова, ..., 31-32 слова, после завершения всех векторных операций, находившихся в процессоре NM6403;

Бит 12 - признак “пустое AFIFO” ,

0 - если в AFIFO есть информация,

1 - если AFIFO пусто

Бит 11 - признак “полное AFIFO” ,

0 - если AFIFO не заполнено полностью,

1 - если AFIFO заполнен полностью, т.е. содержит 32 64-х разрядных значения.

Бит 14 - В регистре PSWR имеется бит управления очисткой AFIFO:”

если 0 - то очистки AFIFO нет,

если 1 - то очистка есть.

6) При записи в AFIFO всегда известно количество слов, которое будет записано (оно задается явно в команде загрузки/выгрузки или является длиной одного из векторных операторов для команд обработки). Это количество слов сразу устанавливается в AFIFO.n, после чего AFIFO становится доступным на чтение, т.е. возможно выполнение парной VP-команды, либо какой-либо следующей машинной команды, читающей из AFIFO. Таким образом, достигается совмещение выполнения векторных команд, работающих с AFIFO.

7) Приведенные выше правила на уровне пользователя означают, что работа с AFIFO в программе должна быть организована так, чтобы непустое AFIFO содержало всегда один вектор, который либо просто хранится в AFIFO, либо находится в режиме записи-считывания, либо может заменяться другим вектором, но такая замена возможна лишь в пределах одной машинной команды (см. п.2 выше).

WFIFO (Weight First-In-First-Out)

1) Чтение WFIFO представляет собой последовательное считывание с удалением 64-х разрядных значений из WFIFO, начиная с первого элемента WFIFO. Такая операция чтения производится только при загрузке из WFIFO теневой матрицы весовых коэффициентов WBUF. При ‘этом количество считываемых слов равно числу синапсов, определяемому по состоянию управляющего регистра SBR. Для правильной загрузки WBUF также необходимо, чтобы было установлено значение регистра NBR.

2) Запись в WFIFO представляет собой последоватнльное добавление в WFIFO 64-х разрядных значений, начиная с первого свободного элемента AFIFO.

3) Если при чтении из WFIFO оказывается, что число требуемых для загрузки WBUF 64-х разрядных значений оказывается больше количества значений, находящихся в данный момент времени в WFIFO, то выполнение команды чтения приостанавливается до появления в WFIFO требуемых значений.Эта ситуация может привести к самоблокировке (клинчу) работы процессора NM6403.

4) Если при записи в WFIFO оказывается, что в WFIFO уже нет места для записи значений, то выполнение команды записи приостанавливается до появления в WFIFO свободного места.

5) Состояние WFIFO отслеживается в регистре запросов на прерывание INTR:

Бит 10 - признак “пустое WFIFO”,

0 - если в WFIFO есть информация,

1 - если WFIFO пусто.

Бит 9 - признак “полное WFIFO” ,

0 - если WFIFO не заполнено полностью,

1 - если WFIFO заполнено полностью, т.е. содержит 32 64-х разрядных значения.

В регистре PSW имеется бит управления очисткой WFIFO:

Бит 15 - 0 - нет очистки WFIFO,

1 - есть очистка WFIFO.

6) Выполнение операций чтения/ записи для WFIFO возможно в любом порядке. Эти операции могут быть совмещены во времени выполнения. Главное условие их успешного выполнения (без клинчей) - соблюдение “баланса” количества записываемой и считываемой информации.

7) Приведенные выше правила работы с WFIFO на уровне пользователя в целом означают, что WFIFO может одновременно хранить несколько векторов весовых коэффициентов, одновременно с этим в WFIFO может добавляться новый вектор, остальные вектора - просто хранятся, ожидают своей очереди загрузки в теневую матрицу весов WBUF.

RAM (Random Access Memory)

1) Чтение из RAM происходит всегда, начиная с первого элемента. Далее считывается второй элемент, третий и т.д., всего столько элементов, сколько указано в машинной команде, либо столько, сколько имеется в RAM. После считываения значения из RAM не удаляются.

2) Запись в RAM производится всегда, начиная с первого элемента.

3) Логика обработки команд в процессоре NM6403 такова, что из RAM всегда читается столько "элементов", сколько там имеется. Действительно, если это не так, то команда не проходит контроль при дешифрации.

4) Запись в RAM не может привести к ее переполнению, поскольку в команде не может быть задано количество элементов, большее 32-х.

5) Состояние RAM отслеживается в регистре запросов на прерывание INTR: Биты 17-13 - количество 64-х разрядных слов данных, которое будет находиться в RAM после завершения всех векторных операций, находящихся в данный момент времени в процессоре NM6403.

6) Выполнение операций чтения/записи для RAM производится строго последовательно, без совмещений. Если в одной и той же машинной команде одновременно выдаются операции чтения и записи в RAM , такая команда считается ошибочной.

7) На уровне пользователя RAM - это векторный регистр, который используется для временного хранения одного вектора. При этом доступ к этому векторному регистру строго последовательный.

Команды загрузки данных в векторый процессор (VL-подгруппа)

Команда
[X←] [Y←] [M←] [RAM←] ({$f_{адр}$(AR_i,GR_i); AR_i←f_m(AR_i,GR_i)}|n); [WBUF←] (WFIFO|WFIFO.S)

Кодировка команды дана на Рис А.14, ее формат 5.1.

Основная операция , выполняемая командой загрузка вектора 64-х разрядных элементов в векторный процессор. Дополнительная операция - перепись весовых коэффициентов из WFIFO в теневую матрицу WBUF.

При выполнении основной операции первый элемент вектора выбирается из внешней памяти, начиная с адреса, определяемого функцией $f_{адр}$. Адреса остальных элементов также определяются значениями этой функции, однако при вычислении этих значений будут исполь-зоваться значения заданного в команде регистра AR_i (биты 24, 23) , модифицированного после обращения к очередному элементу вектора функцией f_m. Функции $f_{адр}$ и f_m кодируются полем МА (биты 28, 27, 26). Длина выбираемого таким образом из внешней памяти вектора явно задается в поле “n” команды (биты 17-13).

Загрузка векторного процессора состоит: во-первых, из записи выбираемого из внешней памяти вектора в RAM, эта операция может выполняться или нет, что определяется полем R_W команды (бит 19); во-вторых, из установки потока данных из выбранных из

внешней памяти элементов вектора, который может использоваться в парной данной команде VP-подгруппы в качестве операндов X, Y или M, либо вообще не использоваться.

Дополнительная операция выполняется, если бит W (бит 18) команды имеет единичное значение. Следует также помнить, что для выполнения этой операции требуется, чтобы были соответственно установлены регистры границ синапсов и нейронов - SB2 и NB2.

Особенно важное значение при этом имеет правильная установка регистра SB2, поскольку из-за неправильной установки может нарушиться “баланс” в работе с WFIFO и произойдет зависание процессора.

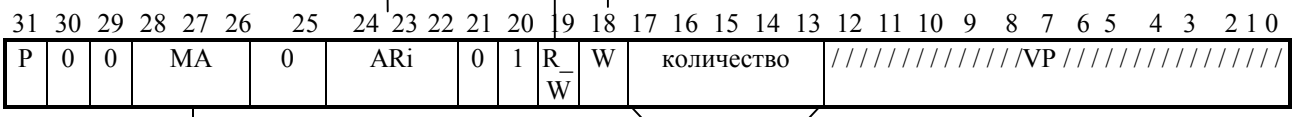
Рис А.14. Кодировка команды VL-подгруппы загрузки данных в векторный процессор

[X←] [Y←] [M←] [RAM←] ({$f_{адр.}(AR_i, Gr_i)$; $AR_i \leftarrow f_m(AR_i, GR_i\}/n)$;

WBUF←] (WFIFO | WFIFO.S); формат 5.1.

AR-регистр	ARi
AR0	000
AR1	001
AR2	010
AR3	011
AR4	100
AR5	101
AR6	110
SP(AR7)c	111

R_W	Управление записью в RAM
0	Нет записи
1	Есть запись
W	Управление загрузкой весов из WFIF0
0	Нет загрузки в теневую матрицу WBUF
1	Есть загрузка в теневую матрицу WBUF



n – длина загружаемого вектора

MA	$f_{адр.}(AR_i, GR_i)$	$f_M(AR_i, GR_i)$
0 0 0	GR_i	AR_i
0 0 1	AR_i+GR_i	AR_i+GR_i
0 1 0	GR_i	GR_i
0 1 1	-	-
1 0 0	AR_i	AR_i
1 0 1	AR_i	AR_i+GR_i
1 1 0	AR_i-2	AR_i-2
1 1 1	AR_i	AR_i+2

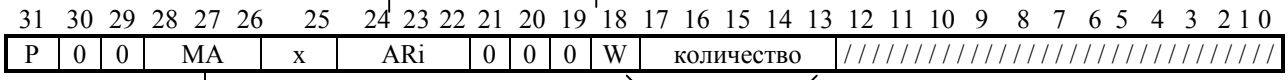
Рис А.15. Кодировка команд форматов 5.1 и 5.2 (команды загрузки данных в векторный процессор)

WFIFO ← ({$f_{адр.}$(AR_i, GR_i)); AR_i←f_m(AR_i, Gr_i}/n);

[WBUF←] (WFIFO|WFIFO.S);

AR-регистр	ARi
AR0	000
AR1	001
AR2	010
AR3	011
AR4	100
AR5	101
AR6	110
SP(AR7)c	111

W	Управление загрузкой весов из WFIF0
0	Нет загрузки в теневую матрицу WBUF
1	Есть загрузка в теневую матрицу WBUF



n длина загруаемого вектора

MA	$f_{адр.}$(AR_i, GR_i)	f_M(AR_i, GR_i)
0 0 0	GR_i	AR_i
0 0 1	AR_i+GR_i	AR_i+GR_i
0 1 0	GR_i	GR_i
0 1 1	-	-
1 0 0	AR_i	AR_i
1 0 1	AR_i	AR_i+GR_i
1 1 0	AR_i-2	AR_i-2
1 1 1	AR_i	AR_i+2

Рис А.16. Кодировка команд формата 5.1 (команды выгрузки данных из векторного процессора)

(AFIFO|AFIFO.n) [→ VDRAM] [→ {$f_{адр.}(AR_i, GR_i)$); $AR_i \leftarrow f_m(AR_i, Gr_i\}$];

[WBUF←] (WFIFO|WFIFO.S);

AR-регистр	ARi
AR0	000
AR1	001
AR2	010
AR3	011
AR4	100
AR5	101
AR6	110
SP(AR7)c	111

R_W	Управление записью в RAM
0	Нет записи
1	Есть запись

W	Управление загрузкой весов из WFIF0
0	Нет загрузки в теневую матрицу WBUF
1	Есть загрузка в теневую матрицу WBUF

31	30	29	28 27 26	25	24 23 22	21	20	19	18	17	16	15	14	13	12 11 10 9 8 7 6 5 4 3 2 1 0
P	0	0	MA	1	ARi	0	1	R_W	W	x	x	x	x	x	//////////////////////////////

MA	$f_{адр.}(AR_i, GR_i)$	$f_M(AR_i, GR_i)$
0 0 0	GR_i	AR_i
0 0 1	AR_i+GR_i	AR_i+GR_i
0 1 0	GR_i	GR_i
0 1 1	-	-
1 0 0	AR_i	AR_i
1 0 1	AR_i	AR_i+GR_i
1 1 0	AR_i-2	AR_i-2
1 1 1	AR_i	AR_i+2

A.3 Программно доступные регистры

В процессоре NM6403 программно доступны следующие регистры:

- адресные регистры, регистры общего назначения и счётчик команд;
- специальные регистры:
  - ♦ слово состояния процессора и регистр запросов на прерывание и ПДП;
  - ♦ регистры управления векторного процессора;
  - ♦ регистры управления интерфейсом локальной и глобальной шины;
  - ♦ регистры каналов ввода/вывода;
  - ♦ таймеры.

Перечень всех регистров с указанием их разрядности и назначения дан в Табл А.1.

Табл А.1. Программно доступные регистры процессора NM6403

Обозначение регистров и их назначение	Разряд-ность
GRi - регистр общего назначения i (i=0,…,7)	32
ARj - адресный регистр j (j=0,…,6)	32
SP(AR7) - указатель стека адресов возврата	32
PC - программный счетчик	32
PSWR - регистр слова состояния процессора	32
PSWRset - код для побитовой установки PSWR в единицу (псевдорегистр)	-
PSWRreset - код для побитового сброса PSWR в ноль (псевдорегистр)	-
Ti - таймер i (i=0,1)	32
GMICR - регистр управления интерфейсом с глобальной шиной	32
LMICR - регистр управления интерфейсом с локальной шиной	32
INTR - регистр запросов на прерывание и ПДП	32
OCAi - регистр адреса канала вывода i (i=0,1)	32
ICAi - регистр адреса канала ввода i (i=0,1)	32
OCCi - счетчик канала вывода i (i=0,1)	32
ICCi - счетчик канала ввода i (i=0,1)	32
DORi - регистр данных канала вывода i (i=0,1)	64
DIRi - регистр данных канала ввода i (i=0,1)	64
FiCR (H,L) - регистр управления функцией активации i (i=1,2) (старшая, младшая часть)	64(32)
VR(H,L) - регистр порога (старшая, младшая часть)	64(32)
NB(H,L) - i-й регистр границ нейронов (старшая, младшая часть)	64(32)
SB(H,L) - регистр границ синапсов (старшая, младшая часть)	64(32)

Указанные в Табл А.1 64-разрядные регистры за исключением DOR0, DOR1, DIR0 и DIR1 могут рассматриваться как пары из двух 32-разрядных регистров, которые обозначаются:

< имя регистра >H – старшие 32 разряда или < имя регистра >L – младшие 32 разряда.

С другой стороны, все 32-х разрядные регистры за исключением INTR образуют две группы регистров – H и L. Регистры этих групп могут образовывать пары в соответствии с определённым порядком. Группы H и L, а также пары регистров этих групп указаны в Табл А.2. В этой же таблице приводится кодировка данных регистров в командах.

Табл А.2. Кодировка ссылок на регистры в машинных командах процессора NM6403 ($R_{источник}$ и $R_{приёмник}$)

$R_{ист/пр-к}$	Регистр-источник	Регистр-приемник	
0 0 0 0 0 0	AR0	AR0	32-разрядные, L группа
0 0 0 0 0 1	AR1	AR1	
0 0 0 0 1 0	AR2	AR2	
0 0 0 0 1 1	AR3	AR3	
0 0 0 1 0 0	AR4	AR4	
0 0 0 1 0 1	AR5	AR5	
0 0 0 1 1 0	AR6	AR6	
0 0 0 1 1 1	AR7(SP)	AR7(SP)	
0 0 1 0 0 0	OCA0	OCA0	
0 0 1 0 0 1	ICA0	ICA0	
0 0 1 0 1 0	OCA1	OCA1	
0 0 1 0 1 1	ICA1	ICA1	
0 0 1 1 0 0	T0	T0	
0 0 1 1 0 1	LMICR	LMICR	
0 0 1 1 1 0	GMICR	GMICR	
0 0 1 1 1 1	PC	PC	
0 1 0 0 0 0	GR0	GR0	32-разрядные, H группа
0 1 0 0 0 1	GR1	GR1	
0 1 0 0 1 0	GR2	GR2	
0 1 0 0 1 1	GR3	GR3	
0 1 0 1 0 0	GR4	GR4	
0 1 0 1 0 1	GR5	GR5	
0 1 0 1 1 0	GR6	GR6	
0 1 0 1 1 1	GR7	GR7	
0 1 1 0 0 0	OCC0	OCC0	
0 1 1 0 0 1	ICC0	ICC0	
0 1 1 0 1 0	OCC1	OCC1	
0 1 1 0 1 1	ICC1	ICC1	
0 1 1 1 0 0	T1	T1	
0 1 1 1 0 1	Резерв	PSWRreset	
0 1 1 1 1 0	INTR	Резерв	
0 1 1 1 1 1	PSWR	PSWR	

Табл А.2. Кодировка ссылок на регистры в машинных командах процессора NM6403 ($R_{источник}$ и $R_{приёмник}$) (Продолжение)

$R_{ист/пр-к}$	Регистр-источник	Регистр-приемник	
1 0 0 0 0 0	GR0, AR0	GR0, AR0	64-разрядные, в том числе пары (H,L)
1 0 0 0 0 1	GR1,AR1	GR1,AR1	
1 0 0 0 1 0	GR2,AR2	GR2,AR2	
1 0 0 0 1 1	GR3,AR3	GR3,AR3	
1 0 0 1 0 0	GR4,AR4	GR4,AR4	
1 0 0 1 0 1	GR5,AR5	GR5,AR5	
1 0 0 1 1 0	GR6,AR6	GR6,AR6	
1 0 0 1 1 1	GR7,AR7	GR7,AR7	
1 0 1 0 0 0	OCC0, OCA0	OCC0, OCA0	
1 0 1 0 0 1	ICC0, ICA0	ICC0, ICA0	
1 0 1 0 1 0	OCC1, OCA1	OCC1, OCA1	
1 0 1 0 1 1	ICC1, ICA1	ICC1, ICA1	
1 0 1 1 0 0	T1, T0	T1, T0	
1 0 1 1 0 1	DIR0	DOR0	
1 0 1 1 1 0	DIR1	DOR1	
1 0 1 1 1 1	PSWR,PC	PSWR,PC	
1 1 0 0 0 0	Резерв	NBL	64-разрядные и 32-разрядные (группы H и L)
1 1 0 0 0 1	Резерв	SBL	
1 1 0 0 1 0	Резерв	F1CRL	
1 1 0 0 1 1	Резерв	F2CRL	
1 1 0 1 0 0	Резерв	NBH	
1 1 0 1 0 1	Резерв	SBH	
1 1 0 1 1 0	Резерв	F1CRH	
1 1 0 1 1 1	Резерв	F2CRH	
1 1 1 0 0 0	Резерв	NB	
1 1 1 0 0 1	Резерв	SB	
1 1 1 0 1 0	Резерв	F1CR	
1 1 1 0 1 1	Резерв	F2CR	
1 1 1 1 0 0	Резерв	VR	
1 1 1 1 0 1	Резерв	PSWRset	
1 1 1 1 1 0	Резерв	VRL	
1 1 1 1 1 1	Резерв	VRH	

АКЦИОНЕРНОЕ ОБЩЕСТВО
НАУЧНО-ТЕХНИЧЕСКИЙ ЦЕНТР

Научно-технический центр Модуль
АЯ 166, Москва, 125190, Россия
Тел: +7 (095) 152-9335
Факс: +7 (095) 152-4661
E-Mail: postmast@module.ru
WWW: http://www.module.ru

Напечатано в России Дата издания: 29.10.99