МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САМАРСКИЙ
ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Д. В. Корнилин, И. А. Кудрявцев

Аппаратные и программные средства систем обработки информации на основе ПЛИС и микропроцессоров

Электронное учебное пособие

САМАРА

2012

Авторы: **Корнилин Дмитрий Владимирович,**
Кудрявцев Илья Александрович

Корнилин, Д. В. Аппаратные и программные средства систем обработки информации на основе ПЛИС и микропроцессоров [Электронный ресурс] : электрон. учеб. пособие / Д. В. Корнилин, И. А. Кудрявцев; Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). - Электрон. текстовые и граф. дан. (1,8 Мбайт). - Самара, 2012. - 1 эл. опт. диск (CD-ROM).

С помощью пособия аспирантам предоставляется возможность углубленного изучения технологий разработки аппаратного и программного обеспечения микропроцессорных систем и систем на базе современных программируемых логических интегральных схем (ПЛИС).

Учебное пособие обеспечивает подготовку по дисциплинам «Технические средства приема, преобразования и передачи информации» и «Программирование ПЛИС и микропроцессоров» образовательных программ аспирантуры по специальности 05.13.05, реализуемых на радиотехническом факультете.

Учебное пособие разработано на кафедре радиотехнических устройств.

Содержание

Введение

Разрабатываемое учебное пособие призвано усовершенствовать образовательный процесс в рамках подготовки аспирантов по приоритетному направлению развития (ПНР) в рамках Программы развития Государственного образовательного учреждения высшего профессионального образования "Самарский государственный аэрокосмический университет имена академика С.П. Королёва (национальный исследовательский университет)" на 2009-2018 годы: «Авиационно-космическая наука, технологии и техника: компьютерное моделирование и информационная поддержка изделий; разработка опережающих производственных и космических геоинформационных технологий; проведение научных исследований и подготовка кадров мирового уровня с использованием научно-образовательных суперкомпьютерных и грид-систем».

Разрабатываемое учебное пособие может быть использовано для многоуровневой подготовки кадров высшей квалификации (уже имеющих высшее профессиональное образование), обладающих междисциплинарными ключевыми компетенциями, для авиационно-космической, радиоэлектронной и других высокотехнологичных отраслей экономики, а также творчески развивает и преумножает лучшие традиции российской инженерно-конструкторской радиотехнической школы на основе достижений фундаментальной науки, прорывных технологий и компьютеризации образования.

Разрабатываемое учебное пособие полностью соответствует федеральным государственным образовательным стандартам третьего поколения (ФГОС-3), предусматривает сквозное компьютерное проектирование по специальным дисциплинам, комплексы лабораторных работ и вычислительных практикумов с использованием компьютерного моделирования.

Разрабатываемое учебное пособие обеспечивает подготовку по дисциплинам «Технические средства приема, преобразования и передачи информации» и «Программирование ПЛИС и микропроцессоров» образовательных программ аспирантуры по специальности 05.13.05, реализуемых на радиотехническом факультете.

Целью учебного пособия является ознакомление аспирантов с основами построения современных систем управления, обработки информации, микропроцессорных систем хранения и передачи данных. Основное содержание пособия составляют: элементы схем цифровой и вычислительной техники, запоминающих устройств, интерфейсов передачи данных, основы синтеза микропроцессорных систем обработки данных, современные микроконтроллеры и микропроцессоры, техника разработки программного обеспечения вычислительных и управляющих систем. Также рассматривается техника разработки устройств управления и обработки данных на базе ПЛИС и технологии отладки и диагностики программного и аппаратного обеспечения.

С помощью пособия аспирантам предоставляется возможность углубленного изучения технологий разработки аппаратного и программного обеспечения микропроцессорных систем и систем на базе современных программируемых логических интегральных схем (ПЛИС).

1 Современные микроконтроллеры и микропроцессоры

1.1 Микроконтроллеры семейства PIC18FXX фирмы Microchip

PIC-контроллеры фирмы Microchip имеют следующие отличительные особенности:
- 8-разрядные данные и 16-разрядные команды (75 команд);
- 8-разрядный умножитель;
- до 32К(128К) внутренней памяти программ и до 1536 байт ОЗУ;
- до 256 байт встроенной FLASH- памяти;
- Наличие 10-разрядного АЦП;
- 3 таймера, обеспечивающие различные режимы (CAPTURE, COMPARE, PWM);
- Сторожевой таймер;
- Возможность работы в режиме пониженного энергопотребления;
- Асинхронный приемопередатчик;
- Синхронный приемопередатчик с поддержкой I^2C;
- Возможность внутрисхемного программирования и отладки посредством двухпроводного последовательного интерфейса.

1.1.1 Организация памяти

Память программ

PIC-контроллеры обладают гарвардской архитектурой. Встроенная память программ, выполненная по FLASH - технологии, имеет объем до 32К(128К). В микроконтроллере предусмотрен механизм, который позволяет программным способом изменять содержимое памяти программ.

Вектора прерывания расположены по адресу 0008h (высокоприоритетный) и 0018h (низкоприоритетный), после сброса микроконтроллер обращается за первой командой по адресу 0000h.

Стек глубиной 31 организован в отдельной области памяти (RAM) и используется, в основном, для хранения адреса (21 бит) возврата из подпрограмм. Имеются команды PUSH и POP, которые могут быть использованы для сохранения/восстановления счетчика команд. Большая часть команд занимает одно слово, но есть и команды, занимающие два слова. Вершина стека доступна для чтения/записи через регистры TOSL,TOSH,TOSU.

Память данных

Память данных разбита на несколько банков объемом 256 байт, при этом часть ОЗУ занята регистрами специального назначения (SFR). Номер банка определяется регистром BSR (4 младших бита). Все регистры имеют 12-битный адрес. Формат команд позволяет разработчику использовать два режима доступа:

В **первом режиме** доступное ОЗУ (Access Bank) имеет объем 128 байт, а еще 128 байт отведено под регистры специальных функций. При этом BSR не используется.

Во **втором режиме** используется BSR, а регистры специальных функций размещены в старшей половине 16-го банка памяти.

Косвенная адресация использует три индексных регистра FSR, состоящих из двух

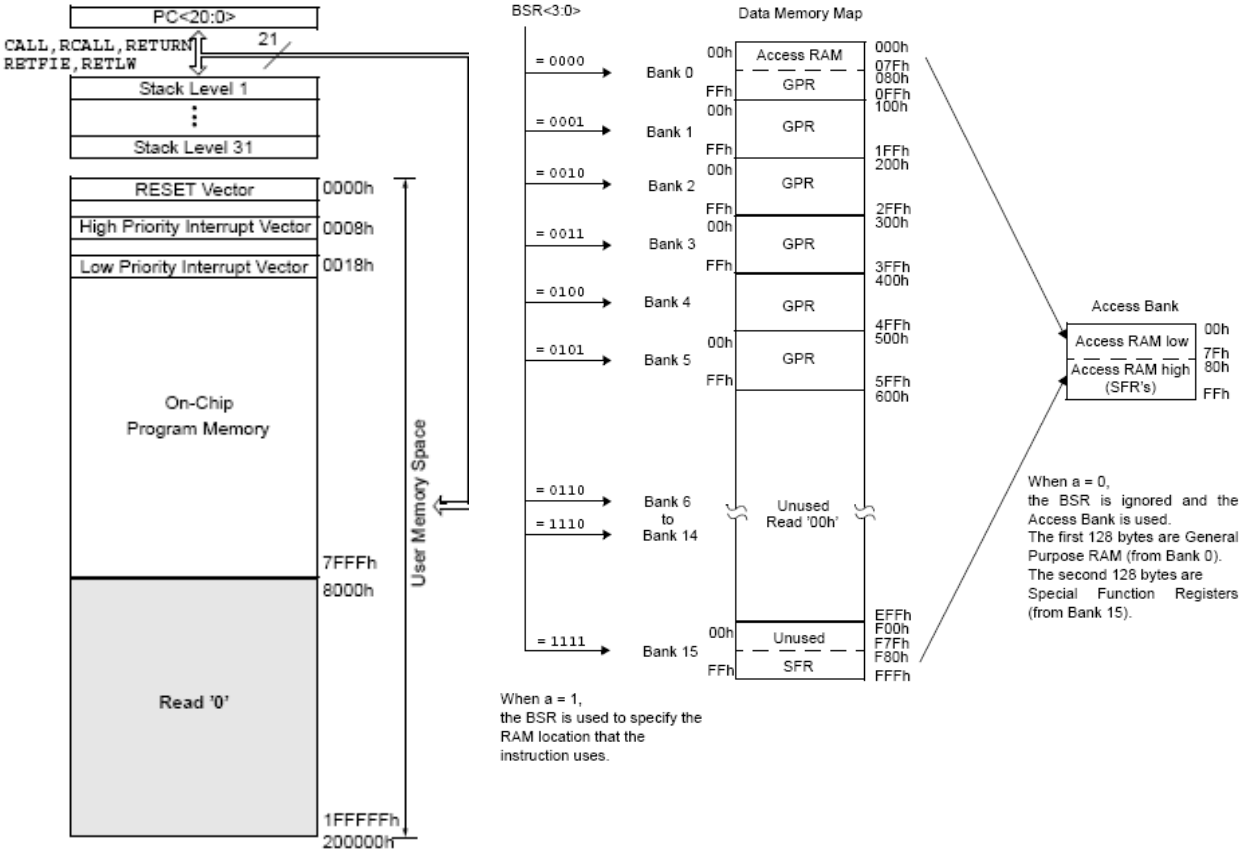
частей (FSRXH:FSRXL). Косвенное обращение к памяти при этом реализуется с помощью псевдорегистра INDFX. В системе команд предусмотрена инструкция загрузки обеих частей в один прием, также имеется возможность автоинкремента/автодекремента и использования содержимого аккумулятора, как дополнительного смещения.

FLASH память

FLASH память данных используется как энергонезависимая память хранения констант и данных настройки. Для записи и чтения FLASH памяти используется специальный механизм и регистры специального назначения. Необходимо помнить, что запись во FLASH память является довольно длительной процедурой (порядка 10мс).

Механизм защиты памяти

Микроконтроллеры семейства PIC имеют несколько уровней защиты, предусматривающих частичную или полную блокировку чтения - записи памяти программ и данных как извне, так и снаружи. Для этого служат биты CPn, WRTn и EBTRn (для памяти программ) и биты CPD и WRTD (для FLASH памяти данных) слова конфигурации, доступного во время операции программирования, а также специальной программной процедурой (снятие защиты не допускается).



Память программ Память данных

Рисунок 1.1 — Структура памяти МК PIC18FXX

1.1.2 Система команд

Система команд микроконтроллеров семейства PIC18FXX состоит из 75 команд. В нижеприведенных таблицах приняты обозначения:

Если d=1, то результат помещается в регистр, если 0 - в аккумулятор.

Если a=0, регистр BSR игнорируется и осуществляется доступ к Access Bank, в

противном случае доступ осуществляется с учетом BSR.
Для упорядочивания описания команд сгруппируем их по функциональному признаку: команды пересылки, арифметические и логические команды, команды передачи управления и битовые команды, специальные команды.

Таблица 1.1 - Команды пересылки

Мнемоника	*Описание*
movf f,d,a	Пересылка *f* в аккумулятор (d=0) или в себя самого (d=1) (f→d) с установкой флагов
movwf f,a	Пересылка аккумулятора в регистр (w→f)
movlw dat	Пересылка значения *dat* в аккумулятор
movlb dat	Пересылка значения *dat* в регистр BSR(3:0)
movff fs,fd	Пересылка *fs* в *fd*
swapf f,d	Перестановка тетрад (f<7:4> ↔ d<3:0>)
lfsr f,dat	Загрузка соответствующего регистра fsr (f=0,1,2) 12-разрядной константой *dat*

Таблица 1.2 - Арифметические и логические команды

Мнемоника	*Описание*
addwf f,d,a	Сложение аккумулятора с регистром с установкой флагов
addwfc f,d,a	То же с учетом флага переноса
addlw dat	Сложение *dat* с аккумулятором (*w+dat→w)*
subwf f,d,a	Вычитание аккумулятора из регистра с установкой флагов
subwfb f,d,a	То же с учетом флага переноса (заёма)
subfwb f,d,a	Вычитание регистра из аккумулятора с установкой флагов и учетом заёма
sublw dat	Вычитание аккумулятора из *dat (dat-w→w)* с установкой флагов
incf f,d,a	Инкремент регистра *f*
decf f,d,a	Декремент регистра *f*
andwf f,d,a	Логическое И аккумулятора с регистром с установкой флагов
andlw dat	Логическое И *dat* с аккумулятором *w&dat→w*
iorwf f,d,a	Логическое ИЛИ аккумулятора с регистром с установкой флагов
iorlw dat	Логическое ИЛИ *dat* с аккумулятором *w|dat→w*
xorwf f,d,a	Исключающее ИЛИ аккумулятора с регистром с установкой флагов
xorlw dat	Исключающее ИЛИ *dat* с аккумулятором *w^dat→w*
comf f,d,a	Инверсия регистра *f*
negf f,d,a	Вычисление отрицательной величины в доп. коде ($\overline{f}-1\blacklozenge\ f$)
clrf f,a	Очистка содержимого *f (f=0)*
setf f,a	Установка всех битов регистра *f*
mulwf f,a	Умножение аккумулятора с регистром. Результат в *PRODH:PRODL*
mullw dat	Умножение аккумулятора на *dat*. Результат в *PRODH:PRODL*
rlcf f,d,a	Циклический сдвиг влево через перенос
rlncf f,d,a	Циклический сдвиг влево без учета переноса
rrcf f,d,a	Циклический сдвиг вправо через перенос
rrncf f,d,a	Циклический сдвиг вправо без учета переноса

Таблица 1.3 - Команды битовых операций и передачи управления

Мнемоника	*Описание*
bcf f,b,a	Сброс бита *b* в регистре *f*
bsf f,b,a	Установка бита *b* в регистре *f*
btg f,b,a	Инверсия бита *b* в регистре *f*

btfss f,b,a	Проверка состояния бита *b* в регистре *f* и переход: Если *b*=1, то следующая команда не выполняется; Если *b*=1, то выполняется следующая команда.
btfsc f,b,a	Проверка состояния бита *b* в регистре *f* и переход: Если *b*=0, то следующая команда не выполняется; Если *b*=1, то выполняется следующая команда.
goto l	Безусловный переход по адресу *l* (команда занимает два слова)
bra l	Безусловный переход по адресу *l* (относительный переход ±1024 слова)
bc l	Переход, если флаг переноса установлен (относительный переход ±128 слов)
bnc l	Переход, если флаг переноса сброшен (относительный переход ±128 слов)
bn l	Переход, если флаг знака (N) установлен (относительный переход ±128 слов)
bnn l	Переход, если флаг знака (N) сброшен (относительный переход ±128 слов)
bz l	Переход, если флаг нуля установлен (относительный переход ±128 слов)
bnz l	Переход, если флаг нуля сброшен (относительный переход ±128 слов)
bov l	Переход, если флаг переполнения установлен (относит. переход ±128 слов)
bnov l	Переход, если флаг переполнения сброшен (относит. переход ±128 слов)
call l,s	Переход к подпрограмме по адресу *l* (команда занимает два слова) *s*=FAST – означает быстрый вызов с сохранением WREG, STATUS, BSR
rcall l	Переход к подпрограмме по адресу *l* (относительный переход ±1024 слова)
return s	Возврат из подпрограммы (*s*=FAST – быстрый возврат)
retfie s	Возврат из подпрограммы обработки прерывания (*s*=FAST – быстрый возврат)
retlw dat	Возврат из подпрограммы с константой в аккумуляторе
incfsz f,d,a	Инкремент регистра *f* и переход: Если результат нулевой, то следующая команда не выполняется; Если результат ненулевой, то выполняется следующая команда.
incfsnz f,d,a	Инкремент регистра *f* и переход: Если результат ненулевой, то следующая команда не выполняется; Если результат нулевой, то выполняется следующая команда.
decfsz f,d,a	Декремент регистра *f* и переход: Если результат нулевой, то следующая команда не выполняется; Если результат ненулевой, то выполняется следующая команда.
dcfsnz f,d,a	Декремент регистра *f* и переход: Если результат ненулевой, то следующая команда не выполняется; Если результат нулевой, то выполняется следующая команда.
tstfsz f,d,a	Проверка регистра f и переход: Если *f* =0, то следующая команда не выполняется; В противном случае выполняется следующая команда.
cpfseq f,a	Сравнение *f* с *WREG* и переход: Если *f =WREG*, то следующая команда не выполняется; В противном случае выполняется следующая команда.
cpfslt f,a	Сравнение *f* с *WREG* и переход: Если *f <WREG*, то следующая команда не выполняется; В противном случае выполняется следующая команда.
cpfsgt f,a	Сравнение *f* с *WREG* и переход: Если *f >WREG*, то следующая команда не выполняется; В противном случае выполняется следующая команда.

Таблица 1.4 - Специальные команды

Мнемоника	*Описание*
clrwdt	Обнуление счетчика сторожевого таймера

nop	Нет операций
sleep	Переход в холостой режим
daw	Десятичная коррекция аккумулятора
pop	Извлечение из стека регистра TOS (результат уничтожается)
push	Адрес следующей команды погружается в стек
reset	Программный сброс микроконтроллера
*tblrd**	Чтение памяти программ (TBLPTR)→TABLAT
tblrd+*	Чтение памяти программ (TBLPTR)→TABLAT, TBLPTR=TBLPTR+1
tblrd-*	Чтение памяти программ (TBLPTR)→TABLAT, TBLPTR=TBLPTR-1
*tblrd+**	Чтение памяти программ TBLPTR=TBLPTR+1 (TBLPTR)→TABLAT
*tblwt**	Запись памяти программ TABLAT →(TBLPTR)
tblwt+*	Запись памяти программ TABLAT →(TBLPTR), TBLPTR=TBLPTR+1
tblwt-*	Запись памяти программ TABLAT →(TBLPTR), TBLPTR=TBLPTR-1
*tblwt+**	Запись памяти программ TBLPTR=TBLPTR+1 TABLAT →(TBLPTR)

При отсутствии указания значения *a* в команде по умолчанию считается, что *a* =0 и доступ к памяти данных осуществляется в пределах Access Bank. Все регистры специальных функций расположены в старшей половине последнего банка памяти, вследствие чего доступ к ним целесообразно осуществлять, указав *a*=0.

Команды *push* и *pop* предназначены для программной организации стека пользователя и работают исключительно с адресами. Сохранение данных с их помощью невозможно.

При использовании косвенной адресации загрузку индексного регистра удобно осуществлять командой *lfsr*. Обращение к самим косвенно адресуемым данным осуществляется с помощью символьных имен INDFX, POSTINCX, PREINCX, POSTDECX, PLUSWX. Последний вариант предназначен для косвенной адресации со смещением, хранящимся в аккумуляторе (WREG), при этом FSRX не изменяется.

1.1.3 Порты ввода - вывода

Микроконтроллеры семейства PIC имеют до 5 портов ввода-вывода, обозначаемых PORTA…PORTE. Выводы портов имеют обычно одну или несколько альтернативных функций, выбор которых осуществляется программным путем.

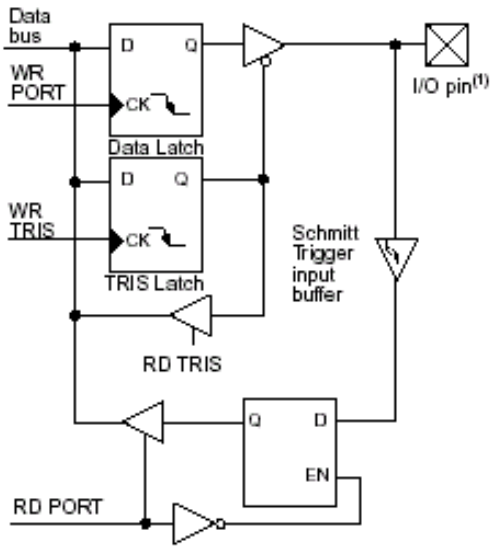


Рисунок 1.2 — Схемотехника портов ввода-вывода

Все порты являются двунаправленными и могут служить для вывода и ввода данных, причем

направление определяется специальными регистрами направления TRISA…TRISE. Каждый разряд порта может быть индивидуально запрограммирован на ввод информации (установкой соответствующего бита регистра направления в 1) или вывод (сбросом в 0).

Необходимо учитывать, что нагрузочная способность портов обычно ограничена величиной тока 25мА, однако есть дополнительные ограничения на суммарный ток всей микросхемы и отдельных портов. Чтение порта означает чтение состояния выводов, а запись - запись в защелку. Этот момент необходимо учитывать при переключении режимов порта, так как если часть выводов сконфигурирована как выходы, а другая, как входы, то при переключении входов в режим выхода, состояние порта может оказаться непредсказуемым. Имеются также некоторые особенности в реализации отдельных портов. В частности, RA4 реализован по схеме с открытым стоком, а выводы порта B подключены к напряжению питания через специальные резисторы утечки (эта особенность может быть программно заблокирована). Изменение состояния выводов RB4…RB7 может генерировать прерывание при соответствующей конфигурации режима порта.

1.1.4 Система прерываний

Система прерываний имеет два вектора с адресами 0x0008 (высокоприоритетное прерывание) и 0x0018 (низкоприоритетное прерывание). Схема приоритетов включается битом IPEN регистра RCON. При сброшенном бите IPEN все прерывания имеют одинаковый приоритет и подпрограмма обслуживания располагается по адресу 0x0008.

Все прерывания являются маскируемыми и могут быть заблокированы индивидуально, а также глобально. Для каждого из векторов имеется свой глобальный бит блокировки GIEH/GIEL (при сброшенном IPEN бит GIEH блокирует все прерывания, однако бит GIEL при этом дополнительно блокирует так называемые периферийные прерывания).

Управление системой прерывания осуществляется через регистры INTCON, INTCON2, INTCON3, PIE1, PIE2, PIR1, PIR2, IPR1, IPR2. В этих регистрах содержатся биты разрешения/запрета прерываний, флаги прерываний и биты, определяющие приоритет конкретного прерывания.

Система прерываний поддерживает внешние прерывания от линий RB0, RB1, RB2, прерывание от изменения состояния линий RB4..RB7, а также прерывания от периферийных модулей, рассматриваемых ниже.

При обработке прерывания в стеке автоматически сохраняется адрес возврата, восстанавливаемый при возврате управления основной программе, сохранение/восстановление остальных регистров возлагается на программу обработки прерываний.

При возникновении запроса прерывания флаг устанавливается автоматически, независимо от того, замаскировано прерывание или нет, при этом в дальнейшем снятие маскировки может автоматически привести к активизации прерывания. Некоторые флаги сбрасываются только аппаратно при определенных действиях, но большинство должно сбрасываться программно обработчиком прерываний.

При обработке прерывания соответствующий флаг глобального разрешения/запрета устанавливается автоматически, при возврате из обработчика командой retfie так же автоматически восстанавливается. Высокоприоритетные прерывания могут прерывать обработчик низкоприоритетных прерываний.

Таким образом, для работы с прерываниями необходимо:

① *Подготовить программу - обработчик прерываний, которая должна сохранять необходимый контекст (WREG, STATUS, BSR и другие регистры при необходимости), сбрасывать флаг прерываний, осуществлять необходимые операции и возвращать управление командой retfie.*

② *Установить индивидуальный бит разрешения прерываний.*

⑦При использовании приоритетной схемы установить бит IPEN и установить необходимый приоритет прерывания
⑧Установить глобальный бит разрешения прерываний с данным приоритетом.
Пример программы – обработчика прерываний

```
ORG 0x0008
movwf W_TEMP                ; сохранение аккумулятора
movff STATUS, ST_TEMP       ; сохранение регистра статуса
movff BSR, B_TEMP           ; сохранение BSR и других нужных регистров
…
btfsc INTCON, INT0IF        ; проверка флага прерываний из числа разрешенных
call INTPROC                      ; вызов обработчика
btfsc…                      ; проверка других флагов
…
movf W_TEMP,W               ; восстановление аккумулятора
movff B_TEMP,BSR            ; восстановление BSR и других регистров
…
movff ST_TEMP,STATUS        ; восстановление регистра статуса
retfie                      ; возврат управления

INTPROC   bcf INTCON, INT0IF ; сброс флага прерывания
          …                  ; собственно обработка прерывания
          return             ; возврат в цикл опроса флагов прерываний
```

В приведенном примере приведен общий обработчик, сохраняющий контекст и опрашивающий флаги. При сохранении/восстановлении регистров необходимо внимательно относиться к регистру статуса, чтобы не допустить его модификации другими командами.

При необходимости можно использовать функцию быстрого возврата с автоматическим восстановлением из специального стека регистров STATUS, WREG и BSR, используя для возврата управления команду RETFIE FAST. При этом их не требуется сохранять при входе в обработчик. Такой подход исключает применение команд CALL label, FAST в основной программе и самом обработчике прерываний и неприменим для низкоуровневых прерываний.

1.1.4 Таймеры

Микроконтроллеры семейства PIC18FXX имеют в своем составе 8/16-разрядный таймер/счетчик Timer0, 16-разрядный таймер/счетчик Timer1, 8-разрядный таймер Timer2 и 16-разрядный таймер/счетчик Timer3.

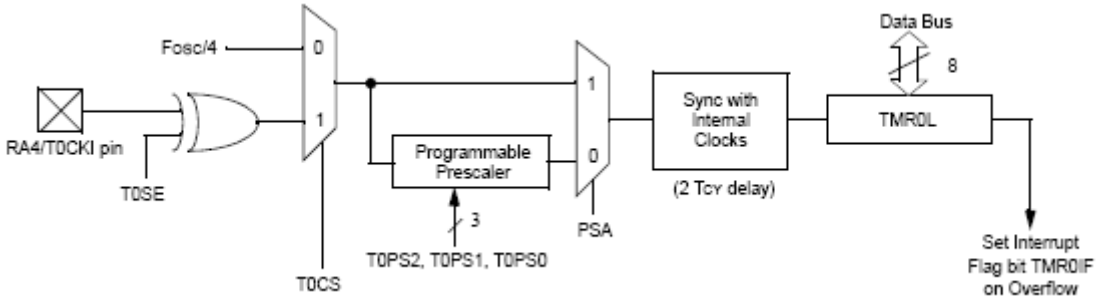


Рисунок 1.3 — Структура Timer0 в 8-разрядном режиме

Таймер0 может работать в режиме таймера или счетчика внешних событий (с программным выбором реакции на фронт или спад импульса) и снабжен управляемым предделителем. Переполнение счетчика вызывает прерывание микроконтроллера.

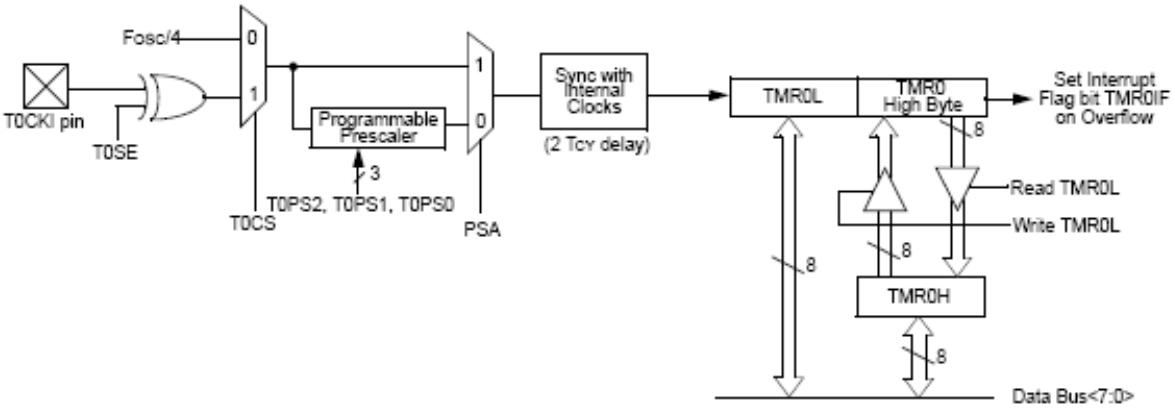


Рисунок 1.4 — Структура Timer0 в 16-разрядном режиме

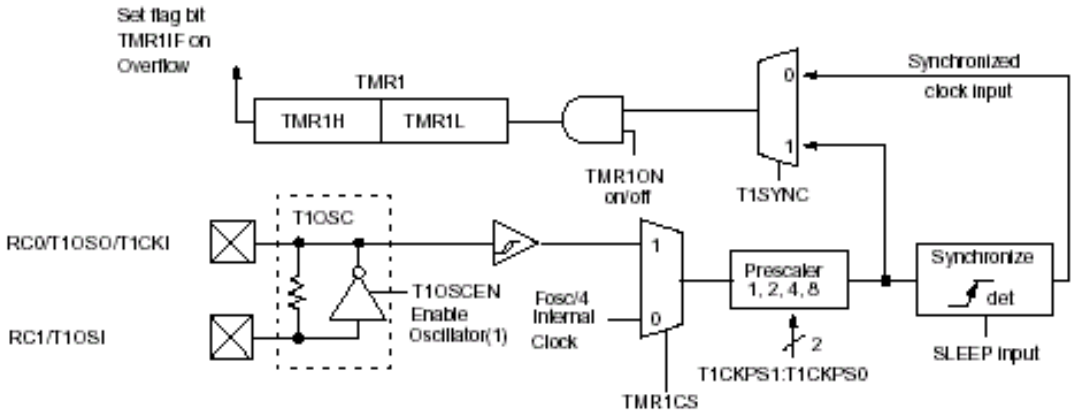
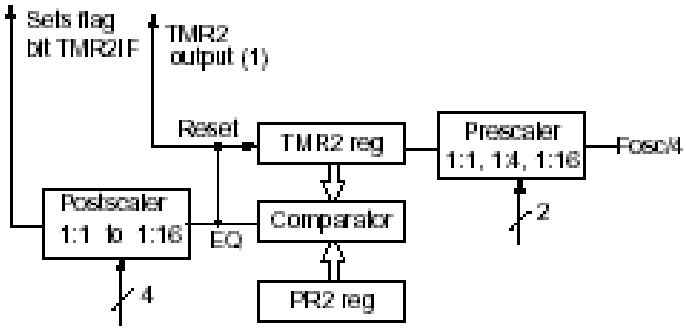


Рисунок 1.5 — Структура Timer1

Таймер1 может также работать в режиме таймера/счетчика, кроме того, он работает в составе модулей CAPTURE, COMPARE и имеет возможность синхронизации от отдельного внешнего тактового генератора. Имеется также возможность синхронизации импульсной последовательности внешнего генератора с тактовым генератором микроконтроллера.



Note 1: TMR2 register output can be software selected by the SSP Module as a baud clock.

Рисунок 1.6 — Структура Timer2

Таймер2 может работать в режиме таймера с генерацией прерывания в момент совпадения содержимого счетчика с содержимым регистра PR2, а также служит для тактирования модулей PWM микроконтроллера. Кроме этого модуль синхронного приемопередатчика может использовать таймер 2 в качестве задающего генератора.

Таймер3 имеет схему, аналогичную таймеру 1 и использует те же внешние выводы при работе в режиме счетчика.

Режим CAPTURE

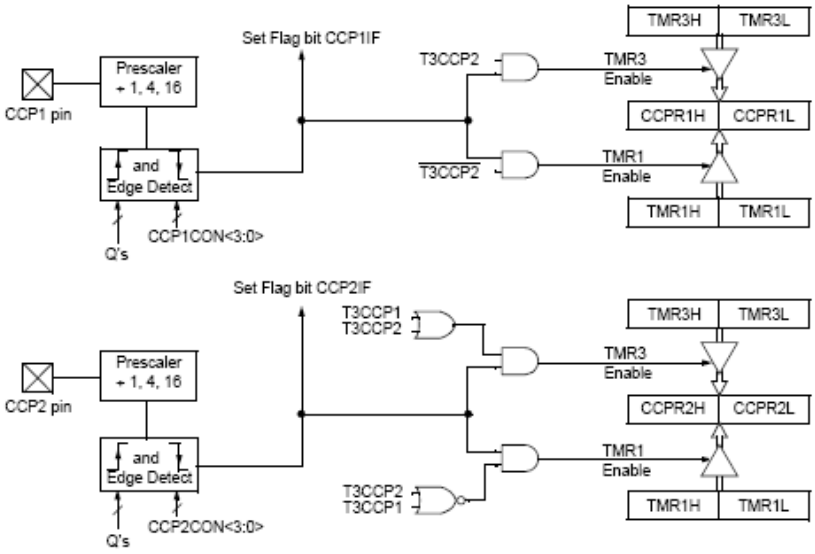


Рисунок 1.7 — Режим CAPTURE

В контроллере имеется два модуля CAPTURE, базой для которых являются таймер 1 и таймер 3. Биты в регистре T3CON указывают, какой из таймеров является базой для модулей CATURE. Возможно три варианта выбора: таймер1 для обоих модулей, таймер3 для обоих модулей, таймер1 для CCP1 и таймер3 для CCP2.

Принцип действия модулей достаточно прост – Когда на выводе микроконтроллера появляется фронт/спад импульса, содержимое выбранного таймера счетчика записывается в регистры CCPRXH:CCPRXL и устанавливается флаг прерывания CCPXIF. Модули CAPTURE удобно применять для измерения временных интервалов (период или длительность импульсов).

Режим COMPARE

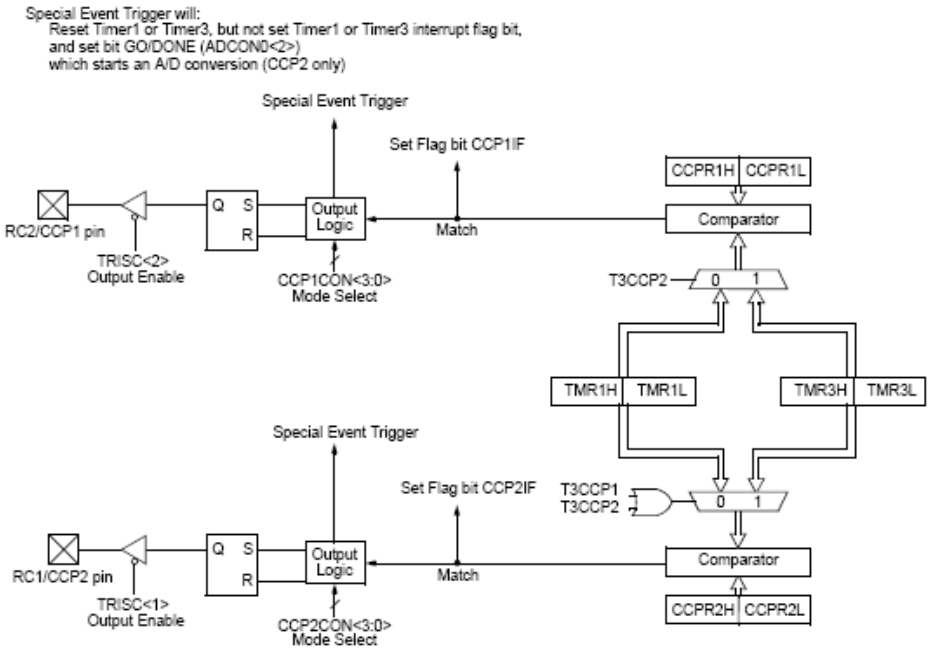


Рисунок 1.8 — Режим COMPARE

В режиме COMPARE используются те же самые модули, однако, принцип другой. Значение сравнения заранее записывается в регистры CCPRXH:CCPXL и постоянно

сравнивается с текущим значением выбранного таймера. Когда обнаруживается совпадение, на выводах микроконтроллера формируется событие (сброс, установка или переключение), генерируется запрос прерывания и устанавливается триггер специального события. Этот триггер может быть использован для запуска АЦП.

Режим генерации ШИМ

Режим генерации ШИМ сигнала используется для формирования последовательности прямоугольных импульсов с управляемыми периодом и длительностью. Наиболее часто используется схема с постоянным периодом и переменной длительностью. С помощью НЧ фильтрации из ШИМ сигнала может быть выделен аналоговый сигнал произвольной формы.

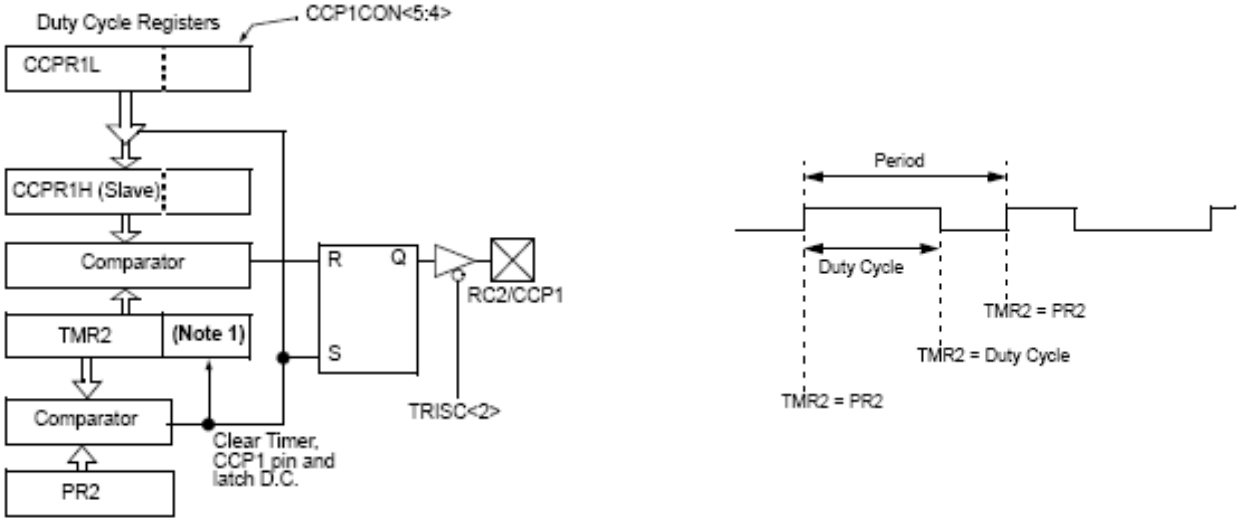


Рисунок 1.9 — Режим генерации ШИМ

В некоторых микроконтроллерах имеется несколько таких модулей для управления многофазными двигателями или модуль ШИМ с расширенной функциональностью. Базой для модуля ШИМ является таймер2.

1.1.4 Синхронный приемопередатчик

Синхронный приемопередатчик позволяет организовать прием и передачу информации по двух- или трехпроводному последовательному интерфейсу (например, SPI) в режиме MASTER или SLAVE, а также реализовать интерфейс I^2C в режиме MASTER или SLAVE.

Режим SPI

В режиме SPI побайтовая передача информации осуществляется через вывод SDO, прием - через вывод SDI, синхроимпульсы передаются через вывод SCK. При работе в режиме SLAVE может быть дополнительно использован вывод $\overline{SS}$.

Передача активируется записью в регистр SSPBUF, после чего данные выводятся в формате MSB First (старшим битом вперед). По окончании передачи устанавливается флаг запроса прерывания SSPIF. Если регистр SSPBUF будет перезаписан в процессе передачи, будет установлен флаг коллизии, однако перезапись не повлияет на передачу текущего байта.

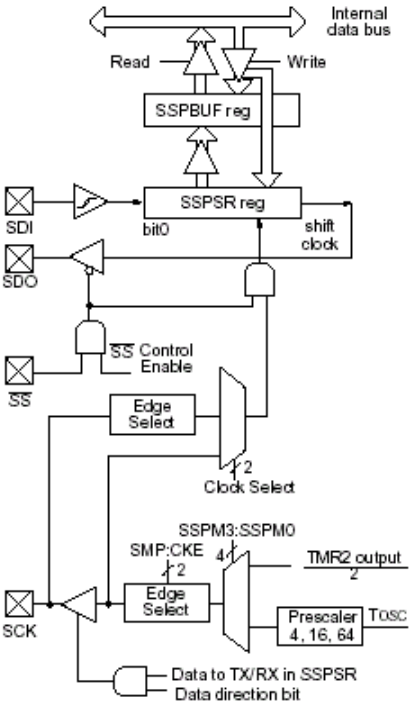


Рисунок 1.9 — Структура в режиме SPI

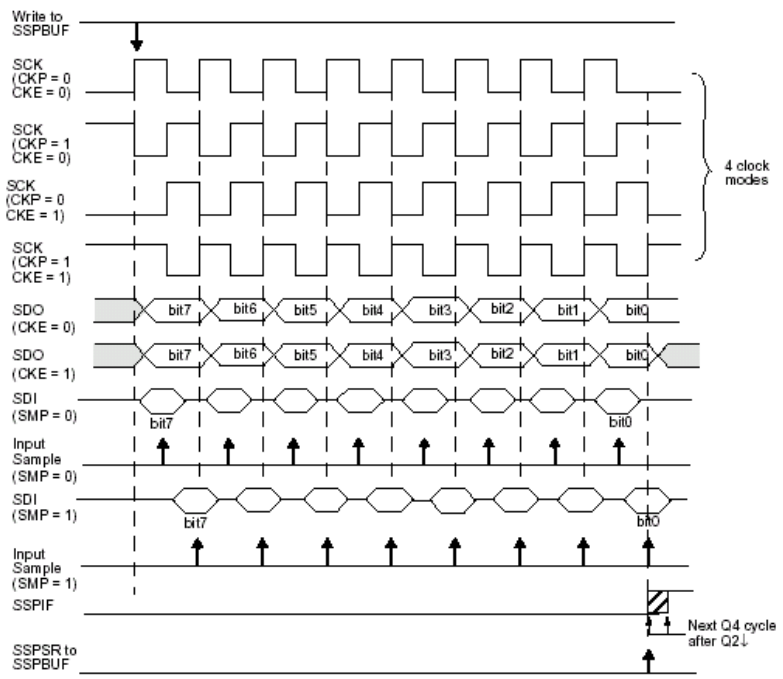


Рисунок 1.10 — Диаграммы обмена в режиме SPI

Прием данных инициируется установкой управляющего бита, после чего 8 тактовых импульсов тактируют запись байта в сдвиговый регистр SSPSR. После приема байта информация переписывается в SSPBUF и устанавливается флаг запроса прерывания SSPIF. Двойное буферирование позволяет начать прием следующего байта, даже если текущий не прочитан. При записи принятого байта в SSPBUF устанавливается бит BF, который сбрасывается аппаратно при чтении SSPBUF.

В режиме SLAVE синхронизация осуществляется от внешнего источника, что позволяет модулю работать в режиме SLEEP. При реализации двухпроводного режима выводы SDO и SDI могут быть объединены при соответствующем арбитраже.

Режим I^2C

В режиме I^2C осуществляется поддержка стандартного интерфейса I^2C, предназначенного для организации внутриприборного взаимодействия различных устройств. Все устройства, подключаемые к этому интерфейсу, конфигурируются, как ведущие (MASTER) или ведомые (SLAVE) и объединяются двухпроводной синхронной шиной. Микроконтроллеры PIC поддерживают оба режима. Частота синхронизации может принимать значения 100 кГц, 400 кГц и 1МГц.

Режим SLAVE

В этом режиме сигнал синхронизации поступает извне. Возможна 10-битовая и 7-битовая адресация. Модуль активируется т.н. стартовым состоянием (отрицательный перепад SCL при высоком уровне SDA), после которого 7 следующих перепадов SCL тактируют байт адреса и еще один - бит R/W (указывающее направление передачи: 0 - режим записи, 1 - режим чтения). Принятый адрес записывается в регистр SSPSR и сравнивается с адресом модуля, предварительно записанным в SSDADD (младший бит SSPADD должен быть равным нулю). Если адреса совпадают, адрес переписывается в SSPBUF, генерируется подтверждение $\overline{ACK}$ и устанавливается бит запроса прерывания. В противном случае

$\overline{ACK}$ не генерируется.

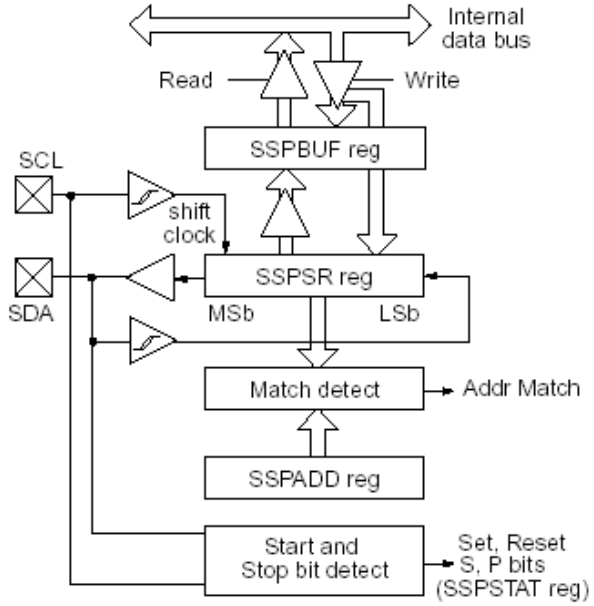


Рисунок 1.11 — РежРежим SLAVE

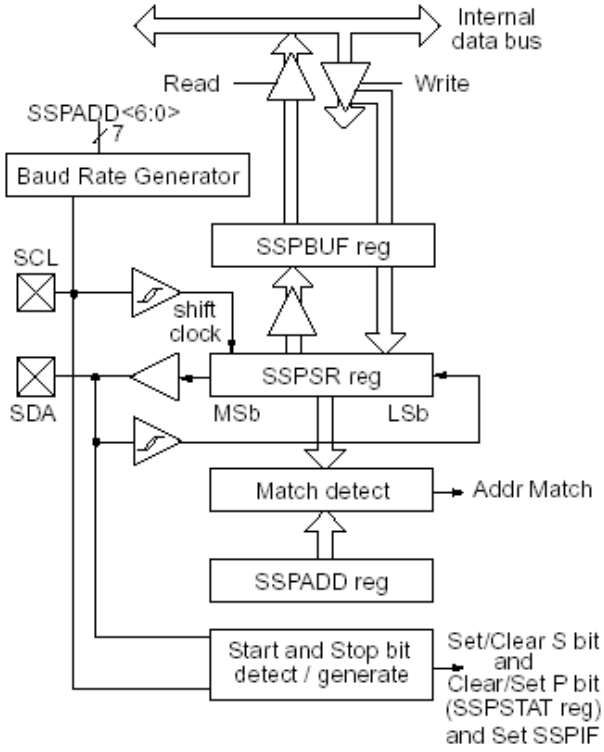


Рисунок 1.12 — РежРежим MASTER

Далее в зависимости от значения R/W происходит прием или передача серии байт данных. В режиме приема каждый принятый байт подтверждается сигналом $\overline{ACK}$. Программа пользователя должна считывать принятые байты до поступления очередного байта, в противном случае генерируется флаг переполнения (бит SSPOV в регистре SSPCON) и выдается сигнал $\overline{ACK}$ =1. Бит SSPOV также должен быть сброшен программным путем.

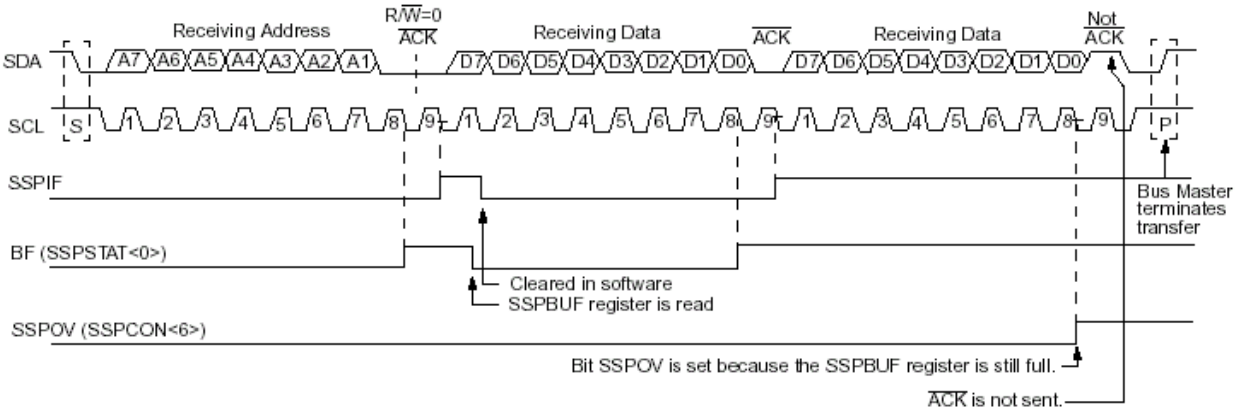


Рисунок 1.13 — Режим приема данных

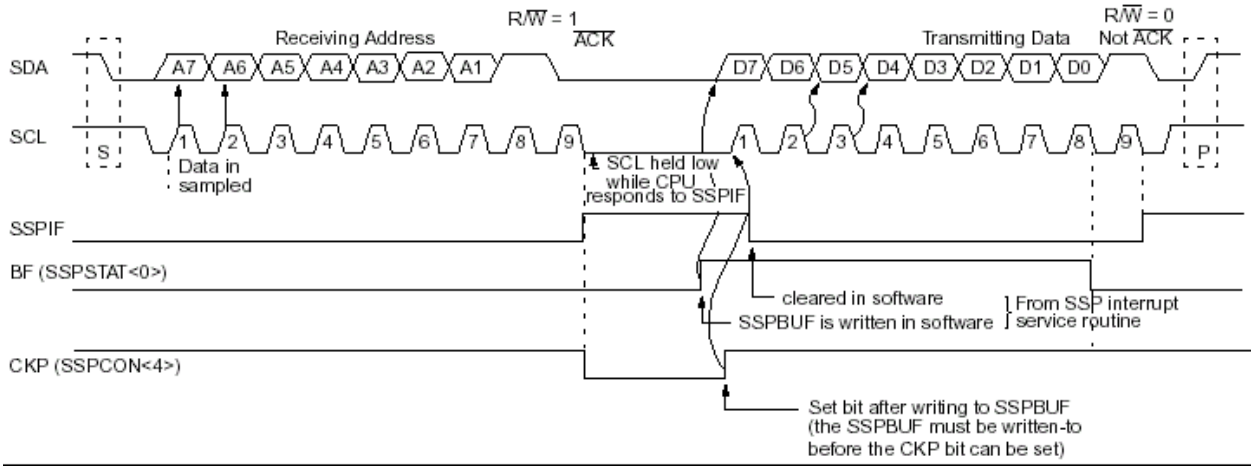


Рисунок 1.14 — Режим передачи данных

Адрес 0000000 соответствует широковещательной рассылке, результат приема такого адреса аналогичен приему индивидуального адреса. Прием широковещательных адресов может быть программно заблокирован.

В режиме передачи данных модуль контролирует $\overline{ACK}$ после каждого байта. Если MASTER генерирует $\overline{ACK}$ =1, модуль прекращает передачу и переходит в режим ожидания нового состояния START.

Линии интерфейса I^2C должны быть подключены к линии питания +5В через резисторы утечки, так как выходные каскады формирователей SDA и SCL выполнены по схеме с открытым стоком.

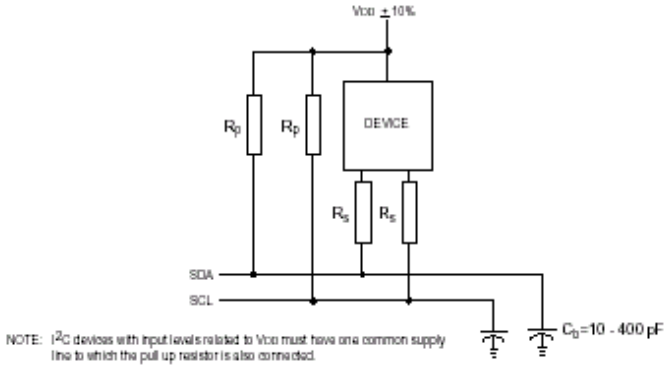


Рисунок 1.15 — Подключение линий интерфейса I^2C

Величина сопротивлений R_p определяется минимальным током стока (3мА). Отсюда при напряжении питания +5В величина сопротивления должна составить около 1.7К.

Режим MASTER

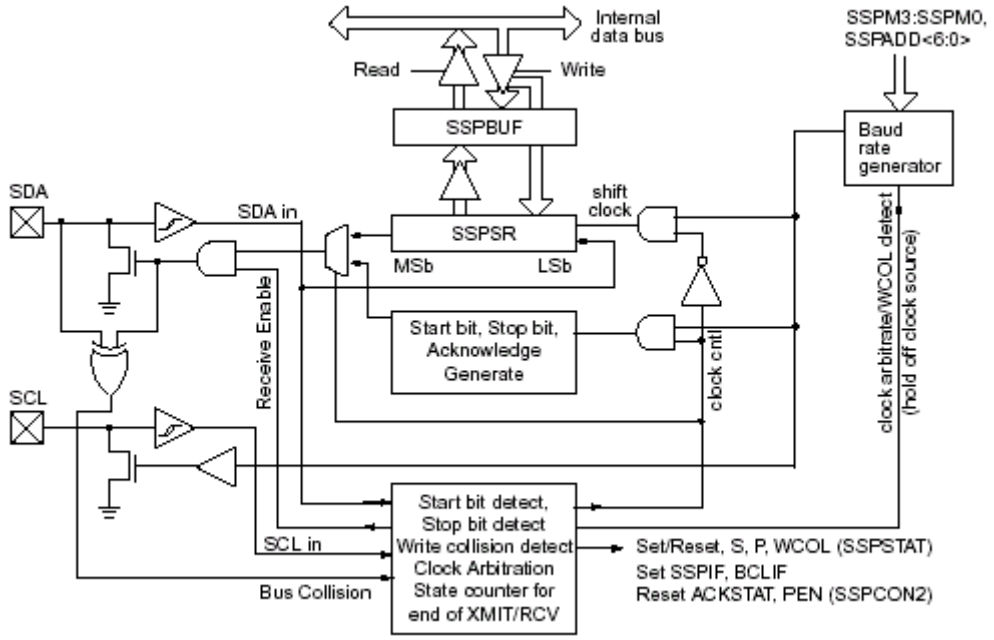


Рисунок 1.16 — Режим MASTER

В режиме MASTER модуль обеспечивает генерацию состояний START, STOP и формирует сигнал синхронизации. В режиме передачи данных модуль действует в следующей последовательности:

1. Генерирует состояние START и ждет его окончания (SSPIF);
2. Загружает адрес приемника в SSPBUF, после чего начинается передача адреса;
3. Принимает и проверяет сигнал $\overline{ACK}$;
4. Загружает байт данных в SSPBUF и начинает передачу данных;
5. Принимает и проверяет сигнал $\overline{ACK}$ в конце каждого передаваемого байта;
6. Генерирует состояние STOP.

Аналогичным образом реализован прием данных. Особым случаем является одновременная работа нескольких модулей в режиме MASTER и разрешение коллизий в этом случае.

1.1.6 Универсальный синхронно-асинхронный приемопередатчик (USART)

USART предназначен для организации синхронного или асинхронного обмена данными и может работать в одном из трех режимов:

1. Асинхронный (дуплексный);
2. Синхронный ведущий (MASTER) полудуплексный
3. Синхронный ведомый (SLAVE) полудуплексный

Для ввода/вывода синхросигнала и информации используются выводы RC6 и RC7, причем RC6 необходимо сконфигурировать, как выход, а RC7 – как вход. Управление передатчиком осуществляется с помощью регистра TXSTA, а управление приемником – с помощью регистра RCSTA. Скорость передачи/приема данных (Baudrate) в обоих режимах устанавливается с помощью регистра SPBRG. Имеется два режима – высокоскоростной и низкоскоростной. Скорость передачи рассчитывается по формулам:

$Baudrate = \frac{F_{OSC}}{64(X+1)}$ - для низкоскоростного режима (*бит BRGH в TXSTA сброшен*)

$Baudrate \Ⓟ \frac{F_{OSC}}{16(X-1)}$ - для высокоскоростного режима (*бит BRGH в TXSTA установлен*)

Асинхронный режим передачи

Асинхронный режим - это наиболее часто используемый режим USART. В этом режиме микроконтроллер поддерживает широко распространенные асинхронные интерфейсы RS-232, RS-422/485.

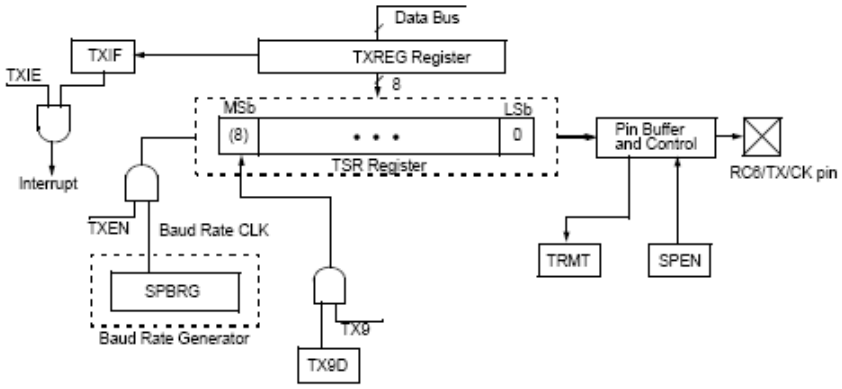


Рисунок 1.17 — Асинхронный передатчик

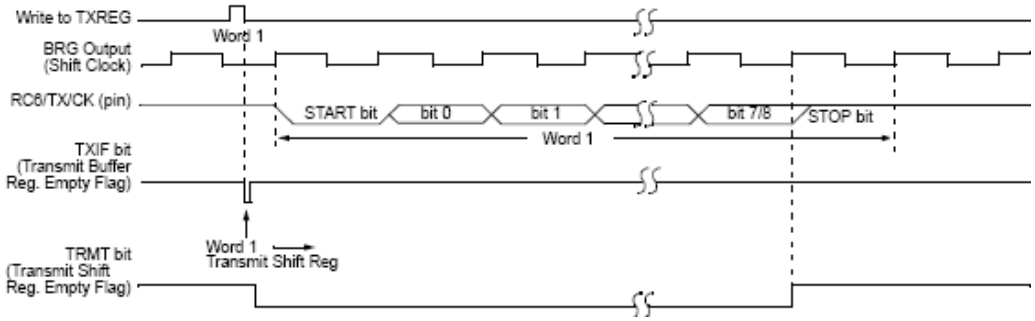


Рисунок 1.18 — Асинхронная передача данных

Передача данных инициируется записью в TXREG, откуда переписывается в сдвиговый регистр TSR (если предыдущая передача завершена), осуществляющий передачу с выбранной скоростью. Бит TXIF генерирует прерывание, сигнализирующее контроллеру, что регистр TXREG пуст. После передачи стоп-бита устанавливается бит TRMT, который сигнализирует об окончании текущей передачи. Следует учитывать, что при первой записи в TXREG, запрос прерывания будет установлен сразу после выполнения команды, несмотря на то, что передача еще даже не начата. При необходимости 9-битной передачи, 9-й бит загружается в TSR из бита TX9 в регистре TXSTA.

Асинхронный режим приема

Приемник должен быть включен битом CREN, однако сам прием начинается после обнаружения отрицательного перепада на входе RC7. Для этого RC7 постоянно сканируется с частотой в 16 раз выше скорости приема. При обнаружении отрицательного перепада модуль Data Recovery синхронизируется и далее определяет уровень сигнала в середине бита по мажоритарному принципу (два из трех).

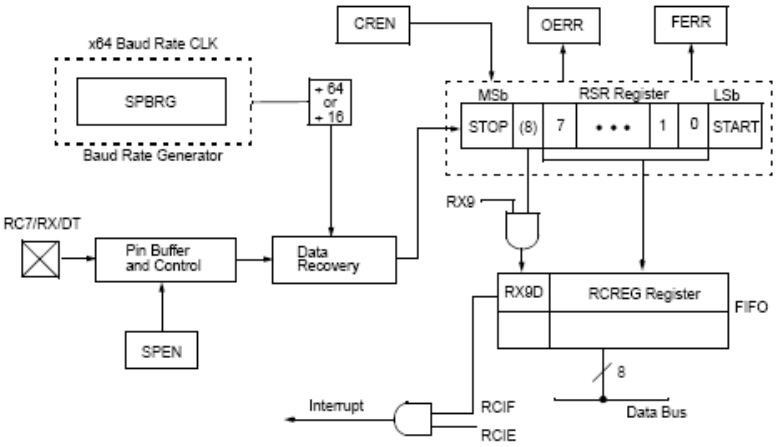
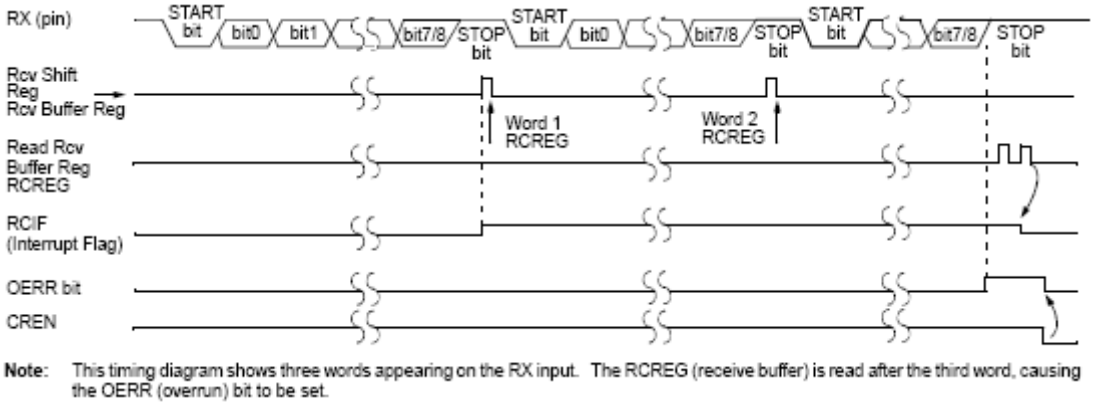


Рисунок 1.19 —Асинхронный приемник



Note: This timing diagram shows three words appearing on the RX input. The RCREG (receive buffer) is read after the third word, causing the OERR (overrun) bit to be set.

Рисунок 1.20 —Асинхронный прием данных

Если обнаружен старт-бит, производится прием всех остальных битов посылки, включая стоп-бит, устанавливается бит RXIF (запрос прерывания) и посылка переписывается в RXREG. Если стоп-бит имеет нулевой уровень, устанавливается бит ошибки FERR (ошибка кадра). Если предыдущий принятый байт к моменту окончания приема текущего не прочитан, устанавливается бит OERR (переполнение буфера).

9-битная передача может использоваться в протоколах типа точка-многоточка, когда единичное значение указывает на передачу адреса, а нулевое – на передачу данных. Приемопередатчик может быть запрограммирован на автоматическое распознавание своего адреса при установленном 9-м бите посылки. В этом случае запрос прерывания генерируется только при совпадении адресов, остальные посылки игнорируются.

1.1.7 Параллельный интерфейс

Модуль параллельного интерфейса имеется в 40-выводных версиях PIC контроллеров, включая PIC18F452, используемый в лабораторном практикуме. В качестве параллельной шины данных используется порт D, а управляющие сигналы передаются через порт E ($\overline{CS}, \overline{RD}, \overline{WR}$). Параллельный интерфейс работает в режиме ведомого (SLAVE). Запись и чтение управляются извне, и после каждой операции генерируется запрос прерывания, обрабатывая который пользователь может считать принятый байт или выставить очередной байт для последующей операции чтения.

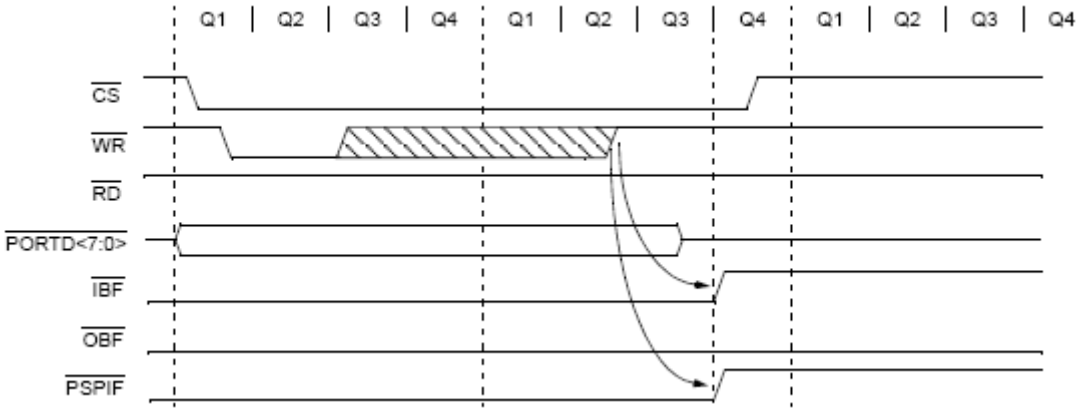


Рисунок 1.21 —Запись в порт

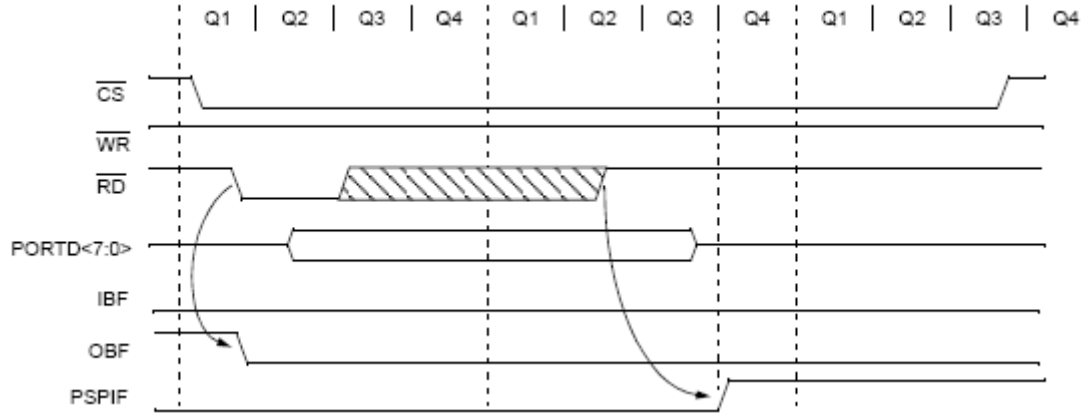


Рисунок 1.22 —Чтение порта

1.1.7 Модуль АЦП

10-битный модуль АЦП обеспечивает сбор данным по 5-8 каналам, для чего предусмотрен встроенный коммутатор каналов и УВХ. Результат преобразования сохраняется в регистрах ADRESH и ADRESL, причем выравнивание может быть и вправо и влево (8 старших бит в регистре ADRESH или 8 младших бит в регистре ADRESL). Опорное напряжение может формироваться модулем или подаваться извне.

Цикл преобразования состоит из двух фаз: фазы формирования отсчета (заряд конденсатора в УВХ) и собственно преобразования.

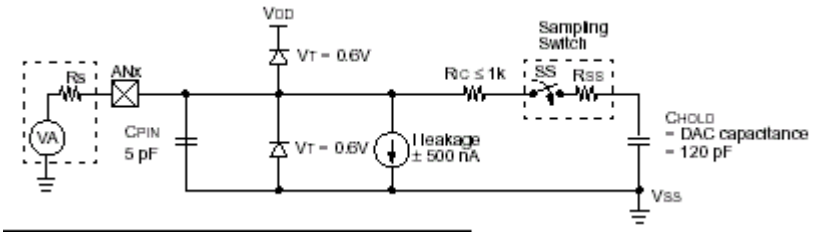


Рисунок 1.23 — Входная цепь АЦП

Длительность первой фазы определяется внутренним сопротивлением источника сигнала, температурой окружающей среды и быстродействием усилителя. Длительность второй фазы для 10-разрядного преобразования занимает не менее 20мкс и определяется выбранной частотой тактирования АЦП.

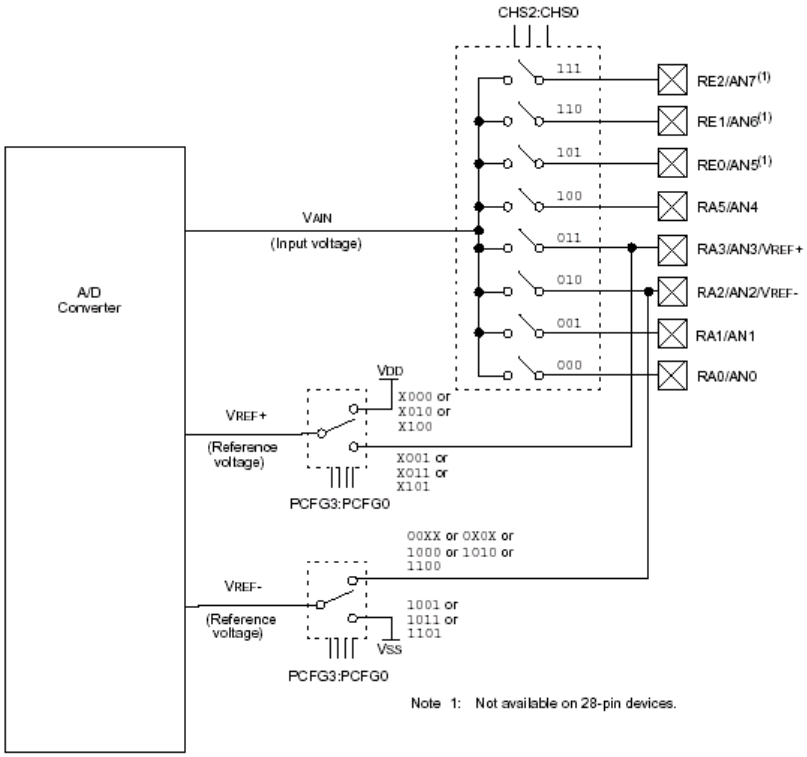


Рисунок 1.24 — Схема входного мультиплексора АЦП

Готовность результатов преобразования может определяться методом опроса флага готовности или с помощью прерывания, которое генерирует модуль АЦП по окончании преобразования.

1.1.8 Особенности PIC - микроконтроллеров

Синхронизация

Для синхронизации могут быть использованы: внешний кварцевый или керамический резонатор или RC- цепочка, кроме этого, может быть использован внешний генератор, подключаемый к выводу OSC1.

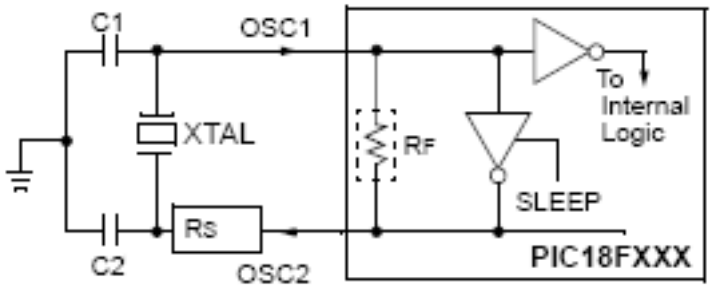


Рисунок 1.25 — Схема включения кварцевого резонатора

Как правило, для синхронизации применяются кварцевые резонаторы, обеспечивающие высокую стабильность частоты колебаний. При этом предусматривается несколько режимов. От выбранного режима зависит величина емкости конденсаторов, кроме того, при программировании необходимо указывать выбранный режим, в противном случае, схема синхронизации может оказаться неработоспособной.

Таблица 1.2 - Выбор режима и элементов схемы синхронизации

Mode	Freq	C1	C2
LP	32.0 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	22-68 pF	22-68 pF
	1.0 MHz	15 pF	15 pF
	4.0 MHz	15 pF	15 pF
HS	4.0 MHz	15 pF	15 pF
	8.0 MHz	15-33 pF	15-33 pF
	20.0 MHz	15-33 pF	15-33 pF
	25.0 MHz	15-33 pF	15-33 pF

Микроконтроллеры PIC18FXX допускают работу на частоте до 40МГц, при этом необходимо выбирать режим HS+PLL, тогда частота внешнего кварцевого резонатора будет увеличиваться в четыре раза с помощью внутренней схемы умножителя частоты.

Микроконтроллеры имеют возможность выбора источника тактирования и переключения к низкочастотному генератору, входящему в состав таймера1.

Особенности аппаратного сброса

PIC18XX имеют несколько источников сброса:

- ♦Сброс при включении питания;
- ♦Аппаратный сброс по выводу MCLR;
- ♦Сброс от сторожевого таймера;
- ♦Сброс при переполнении стека;
- ♦Сброс при исчерпании стека;
- ♦Сброс при обнаружении провала в напряжении питания (BOR);
- ♦Программный сброс (командой reset).

Сброс при включении питания генерируется при увеличении напряжения на выводе питания с 1.2 до 1.7В. Микроконтроллер имеет программируемый встроенный таймер сброса, который обеспечивает фиксированную задержку 72мс, начиная от момента сброса по питанию. Эта задержка необходима для стабилизации напряжения питания. После окончания этой задержки реализуется дополнительная задержка на 1024 периода синхросигнала, необходимая для стабилизации уровня и частоты синхросигнала. При использовании PLL (ФАПЧ) к указанной задержке добавляется еще примерно 2мс.

Сброс BOR (Brown-Out Reset) генерируется при снижении напряжения питания ниже примерно 4В. После восстановления нормального напряжения питания генерируется задержка в 72мс, после чего выполнение программы начинается с адреса 0000h.

Тип сброса может быть определен путем анализа состояния регистра RCON, содержащего флаги, устанавливаемые при каждом из перечисленных событий.

Сторожевой таймер

Сторожевой таймер (WATCHDOG) предназначен для периодической генерации сигнала сброса, что эффективно для систем повышенной надежности. Он представляет собой счетчик, тактируемый специальным RC-генератором, не связанным с системой тактирования контроллера. При его переполнении генерируется сброс. Сторожевой таймер, включенный при записи слова конфигурации, не может быть заблокирован программно, однако, если он не включен битом в слове конфигурации, он может быть включен/выключен программно.

Счетчик таймера может быть сброшен специальной командой CLRWDT. Период генерации сигнала сброса сторожевым таймером определяется специальным делителем, конфигурируемым при записи слова конфигурации.

Модуль LVD (обнаружения низкого напряжения)

Модуль LVD представляет собой компаратор, к одному входу которого подключен источник опорного напряжения (1.2В), а к другому измеряемое напряжение.

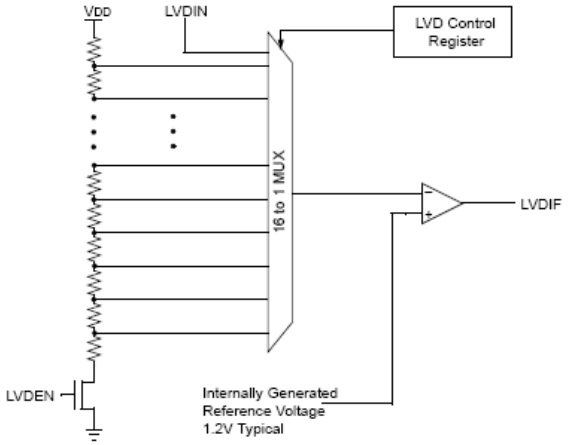


Рисунок 1.25 — Модуль LVD

Измеряемое напряжение можно подать на вход LVDIN (RA5) или от источника питания через внутренний резистивный делитель. При снижении напряжения на входе компаратора ниже 1.2В модуль генерирует прерывание, которое может быть использовано, например для регистрации факта провала в напряжении питания.

Режим пониженного энергопотребления

Этот режим целесообразно применять, если разрабатываемое устройство подключено к автономному источнику питания (батарея, аккумулятор и т.п.). Вход в режим пониженного энергопотребления осуществляется по команде Sleep. В режиме пониженного энергопотребления генератор тактовых импульсов выключается, и тактирование ядра и периферии прекращается, при этом ток потребления снижается до величины не более 10мкА. Содержимое памяти и всех регистров и портов ввода/вывода не изменяется. Напряжение питания может быть снижено примерно до 1.5В. Тем не менее, возможна работа некоторых периферийных модулей, например, АЦП, таймеров1/3 при работе от внешнего генератора или в режиме счетчика, модуля SPI/I^2C в режиме SLAVE, USART в синхронном режиме (также в конфигурации SLAVE, модуля LVD и сторожевого таймера.

Выход из этого режима осуществляется с помощью любого разрешенного прерывания или с помощью сброса. При активизации механизма сброса устройство осуществляет сброс, как описано в р. 10.2, при этом можно контролировать факт выключения питания с помощью бита $\overline{PD}$ в регистре RCON. Если выход из режима пониженного энергопотребления осуществляется с помощью прерывания, то контроллер продолжает свою работу с инструкции, следующей за командой SLEEP. При этом «пробуждение» состоится, даже если бит GIE сброшен, однако в этом случае обработчик прерывания не получит управления.

1.1.9 Программирование микроконтроллера

Программирование микроконтроллеров PIC на практике осуществляется двумя способами:

- помощью внешнего программатора;

Через загрузчик (bootloader).

Собственно программирование состоит в записи информации во FLASH-память программ, FLASH-память данных и записи конфигурационной информации. FLASH-память программ используется, естественно, для хранения кода программы, а также может быть использована для хранения констант. Существует механизм программного изменения памяти программ, однако, необходимо помнить, что количество циклов стирания/записи ограничено. FLASH память данных используется для хранения информации о настройке и также может быть изменена программно с аналогичными ограничениями.

Конфигурационная информация хранит настройки режимов микроконтроллера, а также заводской идентификатор микроконтроллера, считывая который программатор может определить тип микроконтроллера. Пользователь может добавить также и информацию о конкретной прошивке, которая обычно включает наименование фирмы и номер версии/релиза прошивки.

Вся вышеописанная информация может включаться в hex-файл, формируемый средой разработки или добавляться непосредственно при программировании средствами программатора.

Защита интеллектуальной собственности может осуществляться путем установки специальных битов защиты, блокирующих внешний доступ к памяти программ (допускается только стирание содержимого программы).

Программирование с помощью внешнего программатора, в основном определяется характеристиками оборудования, и обычно осуществляется внутрисхемным методом после монтажа микросхемы на печатную плату.

Альтернативный вариант предполагает работу с загрузчиком и программирование через USART.

1.2 Микроконтроллеры семейства ARM7 фирмы NXP

Архитектура ARM7 в настоящее время является одной из наиболее распространенных архитектур ядра 32-разрядных микроконтроллеров различных фирм. Фирма ARM разработала целое семейство архитектур, причем сама она не производит кристаллов, а продает лицензии на разработанные архитектуры. Фирмы, приобретающие лицензии, могут добавлять к ядру различные периферийные модули, получая при этом кристаллы разной степени универсальности и разного предназначения. Мы будем рассматривать микроконтроллеры фирмы NXP, также имеющие широкое распространение при разработке систем управления и обработки данных, включая бортовые системы летательных и космических аппаратов. Фирма NXP производит микроконтроллеры семейства LPC2000 с ядром ARM7TDMI трех типов или поколений. С целью приближения изложения к потребностям лабораторного практикума остановимся на втором поколении (группа LPC2142/44/46/48). Учитывая специфику ARM микроконтроллеров, сначала рассмотрим общие особенности и систему команд ядра ARM, после чего перейдем к рассмотрению особенностей, привнесенных разработчиками фирмы NXP.

1.2.1 Архитектура ARM7TDMI

ARM7TDMI-S – процессор семейства 32-битных фоннеймановских RISC микропроцессоров общего назначения фирмы ARM. Является воплощением архитектуры v4T
Основные особенности его архитектуры:

• наличие трёхступенчатого конвейера позволяет выполнять последовательно поступающие команды за один такт (в том числе команды умножения);

• ядро может функционировать в двух состояниях: ARM и THUMB. В состоянии ARM процессор выполняет 32-разрядные команды, в состоянии THUMB — 16-

разрядные команды;
• в состав ядра включены отладочные интерфейсы JTAG и ETM.

Конвейер команд

Для ускорения прохождения команд через ЦПУ в процессоре ARM7TDMI-S используется конвейер с тремя аппаратно-независимыми ступенями, благодаря которым одновременно с выполнением первой команды, осуществляется декодирование второй и выборка третьей.
Трех ступенчатый конвейер является самой простой разновидностью конвейеров и не подвержен возникновению различных опасных ситуаций, таких как «чтение раньше записи», которые встречаются в конвейерах с большим числом ступеней. Однако даже трехступенчатый конвейер ускоряет прохождение команд через ЦПУ на столько, что большинство команд ARM может выполняться за один такт.
Поскольку конвейер является составной частью ЦПУ, он полностью скрыт для программиста. Однако важно помнить одну деталь: Счетчик команд (PC) указывает на выбираемую из памяти инструкцию, а не на выполняемую, поэтому надо быть очень аккуратным при вычислении смещений в случае относительной адресации с использованием счетчика команд.

Доступ к памяти

Процессор ARM7TDMI-S имеет архитектуру Фон Неймана с единственной 32-х битной шиной данных, по которой передаются как команды, так и данные.
Только команды загрузки, хранения и обмена имеют доступ к данным в памяти.
ARM7TDMI-S процессор рассматривает память как линейную последовательность пронумерованных байтов в возрастающем порядке от нуля:
• байты от 0 до 3 содержат первое сохраненное слово
• байты 4 - 7 содержат второе сохраненное слово
4
• байты 8 - 11 содержат третье сохраненное слово.
ARM7TDMI-S процессор может обрабатывать слова в памяти, сохраненные в одном из форматов :
• Big-endian
• Little-endian.

Big-endian формат

В этом формате, процессор сохраняет наиболее значимый байт слова в самом младшем пронумерованном байте, а наименее значимый байт - в самом старшем пронумерованном байте. Так байт 0 системы памяти соединен с линиями данных от 31 до 24.

Little-endian формат

В формате Little-endian, младший пронумерованный байт в слове считают менее значимым , а старший пронумерованный байт - наиболее значимым. Так байт 0 системы памяти соединен с линиями данных от 7 до 0.

Состояния функционирования

ARM7TDMI-S процессор имеет два состояния:

ARM состояние в этом состоянии выполняются 32-разрядные команды ARM.
Thumb состояние в этом состоянии выполняются 16-разрядные команды Thumb.

Набор команд Thumb

Набор команд Thumb является подвидом наиболее часто используемого 32-х битного набора команд ARM. Каждая команда Thumb имеет длину 16 бит и соответствующую ей 32-х битную команду ARM, которая оказывает тот же эффект на процессорную модель. Команды Thumb оперируют со стандартным набором регистров ARM, обеспечивая великолепную функциональную совместимость между режимами ARM и Thumb.
При выполнении, 16-ти битные команды прозрачно для программиста, трансформируются в полные 32-х битные команды в режиме реального времени и без потерь производительности.
Thumb имеет все преимущества 32-х битного ядра:
- 32-х битное адресное пространство
- 32-х битные регистры
- 32-х битный сдвиговый регистр и АЛУ
- 32-х битный передатчик памяти

Поэтому набор команд Thumb предлагает длинный диапазон перехода, мощный аппарат арифметических операций и большое адресное пространство.
Код Thumb обычно занимает 65% от размера ARM кода и обеспечивает 160% его производительности, когда выполняется на процессоре, работающем с 16-ти битной памятью. Поэтому, Thumb делает процессор ARM7TDMI-S идеально подходящим для вложенных задач с ограниченной пропускной способностью памяти, где важна плотность кода.
Возможность использования наборов команд Thumbs и ARM дает разработчикам гибкость в выборе либо производительности, либо размера кода на подпрограммном уровне, следуя требованиям разрабатываемых ими приложений. Например, критический цикл для такого приложения, как быстрое прерывание или алгоритм цифровой обработки данных может быть реализован с помощью набора команд ARM, а вызываться с помощью Thumb- кода.
Замечание 1: Стоит отметить, что в состоянии Thumb Счетчик команд (PC) при переходе на следующую команду изменяется не на 8, а на 4.
Переключение между двумя этими режимами осуществляется с помощью команды BX (Thumb->ARM) или BLX (ARM->Thumb).

Кроме того, автоматическое переключение в режим ARM происходит при сбросе или при входе в режим обработки исключительной ситуации.
Замечание 2: Переход между состояниями ARM и Thumb не затрагивает режим процессора или содержание регистров.
Замечание 3: Вся обработка особых ситуаций выполняется в состоянии ARM. Если особая ситуация возникает в состоянии Thumb, процессор переключается в состояние ARM. Возврат к состоянию Thumb происходит автоматически после обработки исключительной ситуации.

Режимы функционирования процессора

Процессор ARM7TDMI-S может функционировать в одном из следующих режимов:
• Пользовательский режим User- используется для выполнения большинства прикладных программ.
При возникновении исключительной ситуации режим работы процессора изменяется.
• Режим быстрого прерывания (FIQ) (Fast IRQ) - режим быстрой реакции на пре-

рывания, в который попадает процессор при поступлении запроса высшего уровня на вход FIQ.
• Прерывание (IRQ) режим обработки прерываний, в который попадает процессор при поступлении запроса прерывания низшего уровня на вход IRQ.
• Привилегированный режим Supervisor - защищенный режим, обеспечивающий работу под управлением операционной системы (ОС), которая оперирует данными, недоступными программам пользователя.
• режим Abort режим, который реализуется при ошибке обращения к памяти (ошибки такого рода — обращение по несуществующему адресу, попытка записи в ПЗУ и другие, фиксируются контроллером прерываний, который выдаёт процессорному ядру запрос Abort).
• режим System - режим выполнения системных программ, при котором ОС работает с данными пользователя.
• режим Undefined реализуется при выборке неопределенной команды.
Все режимы, кроме пользовательского, известны как привилегированные режимы. Привилегированные режимы используются, чтобы обслужить прерывания, особые ситуации, или защищенные ресурсы.
Текущий режим можно узнать с помощью регистра CPSR, который мы рассмотрим чуть позже.

Регистры состояния ARM

Пользовательские регистры

Основной регистровый файл состоит из 16 пользовательских регистров.
Регистры R0…R12 – регистры общего назначения, предназначенные исключительно для нужд пользователя (используемые для хранения данных или адресов) и не выполняющие никаких других функций.
Регистры R13…R15 имеют дополнительные (специальные) функции:
• Регистр R13 используется в качестве указателя стека (Stack Pointer –SP)
• Регистр R14 называется регистром связи (Link Register – LR). (При вызове подпрограммы адрес возврата автоматически запоминается в регистре связи, откуда затем считывается при возврате. Такое решение позволяет быстро переходить к «концевым» функциям, т.е. к функциям, которые не вызывают других функций, и возвращаться из них. Если же функция входит в состав «ветви», т.е. вызывает другие функции, содер-

жимое регистра связи необходимо сохранять в стеке (R13)). При выполнении команды «Переход со ссылкой» (BL) (например вызов подпрограммы) регистр R14 получает копию регистра R15. Важно не потерять этот адрес возврата, а так этот регистр можно использовать как регистр общего назначения.
• Регистр R15 выполняет функции счетчика команд (PC). Хочу обратить внимание, что в состоянии ARM, биты [1:0] R15 являются нулевыми, а биты [31:2] содержат PC; а в состоянии Thumb, бит [0] является нулевым, а биты [31:1] содержат PC.
Стоит отметить, что, несмотря на свои специальные функции, многие команды могут работать с регистрами R13…R15, как с обычными пользовательскими регистрами.
Регистр текущего состояния программы (Current Program Status Register - CPSR).
Регистр текущего состояния программы содержит набор флагов, которые управляют режимом функционирования процессора и отображают его состояние.
Регистр сохраненного состояния программы (Saved Program Status Register – SPSR). Данный регистр содержит набор флагов, аналогичных флагам CPSR, сохраненных в результате возникновения особой ситуации, вызвавшей данный режим.

Если в момент возникновения исключительной ситуации программа находилась в режиме User, то происходит смена режима, и текущее содержимое регистра CPSR сохраняется в регистре SPSR. После обработки исключительной ситуации (при возврате из обработчика) содержимое регистра CPSR восстанавливается из SPSR, обеспечивая возобновление выполнения прикладной программы.
Замечание 4: данный регистр имеется во всех режимах, кроме User и System.
Биты M4-M0 определяю шесть режимов процессора, которым соответствуют несколько отличающиеся наборы регистров.

Регистры состояния Thumb

Регистры состояния Thumb являются подмножеством регистров состояния ARM.
Так как в формате большинства команд Thumb под номер регистра отведено только 3 бита, поэтому прямое обращение возможно только к регистрам R0-R7(к так называемым «младшим регистрам»). А регистры R8-R12(«старшие регистры») доступны только через специальные команды загрузки (MOV, ADD, CMP).
В наборе команд Thumb отсутствуют команды MSP и MRS, поэтому изменять регистры CPSR и SPSR можно только с помощью косвенной адресации. Если необходимо изменить пользовательские биты в регистре CPSR, то необходимо переключаться в режим ARM.

Исключительные ситуации

Исключительные ситуации появляются, когда нормальное выполнение программы должно быть временно остановлено, например, при прерывании от периферийного устройства. Перед обработкой исключительной ситуации ARM7TDMI-S сохраняет текущее состояние процессора так, чтобы основная программа могла быть продолжена, после того, как обработчик завершит свою работу.
Каждый источник исключительной ситуации имеет свой фиксированный приоритет, и если две или несколько исключительные ситуации возникают одновременно, то они рассматриваются в соответствии со своими приоритетами.
Приоритеты прерываний установлены в следующем порядке (таблица 1.3)

Таблица 1.3 — Приоритеты прерываний

Приоритет	Исключительная ситуация
Высший 1	Reset
2	Data Abort
3	FIQ
4	IRQ
5	Prefetch Abort
Низший 6	Undefined instruction. SWI

Встроенные периферийные устройства обслуживаются прерываниями FIQ и IRQ.
Приоритеты прерываний от периферийных устройств можно назначать внутри этих групп.
ARM7 поддерживает векторные прерывания: при возникновении исключительной ситуации изменяется режим работы ЦПУ (это мы уже знаем) и в PC загружается адрес соответствующего вектора прерывания. Вектора прерываний располагаются в памяти по-

следовательно, начиная с нулевого адреса. Каждый вектор содержит 4 байта, которые являются адресами первой команды обработчика прерываний.

Таблица 1.4 — Таблица векторов прерываний

Адрес	Вектор
0x00000000	Запуск (Reset)
0x00000004	Неопеделенная команда
0x00000008	Программное прерывание
0x0000000C	Ошибка выборки инструкции
0x00000010	Ошибка выборки данных
0x00000014	— (резервировано)
0x00000018	IRQ
0x0000001C	FIQ

Замечание 1: в таблице векторов имеется «дырка», поскольку вектор с адресом 0x00000014 отсутствует. Этот адрес использовался в ранних версиях процессоров ARM, а в процессоре ARM7 он сохранен, чтобы обеспечить программную совместимость между различными архитектурами ARM.
Замечание 2: в таблице отсутствуют вектора прерываний по запросам внешних устройств. Дело в том, что микроконтроллеры, реализуемые на базе ядра ARM7TDMI, имеют разнообразный набор периферийных устройств. Поэтому в состав микроконтроллера вводится контроллер прерываний, который транслирует запрос от любого периферийного устройства в запрос IRQ или FIQ. Далее программа-обработчик должна самостоятельно определить источник запроса, используя регистры контроллера прерываний.
При поступлении внешнего запроса прерывания или обнаружении ошибки последовательность действий процессора такова:
• Сохраняет адрес следующей выполняемой команды в регистре связи LR.
• Копирует регистр CPSR в регистр SPSR конечного режима.
• Выставляет значения битов M4-0 регистра CPSR, в зависимости от вида исключительной ситуации.
• Заносит адрес вектора прерывания режима исключительной ситуации в регистр PC.
В то же время меняется режим работы процессора, в результате чего регистры R13 и R14 заменяются соответствующими регистрами нового режима (Производит выборку следующей команды из вектора исключительной ситуации)
• !!! ARM7TDMI-S также устанавливает флаг запрета прерывания по исключительной ситуации. Если требуется использовать вложенные прерывания, то необходимо вручную разрешить прерывания и занести содержимое регистра связи в стек, чтобы сохранить исходный адрес возврата.
С вектора прерывания программа перейдет к выполнению подпрограммы обработки прерываний. Первое, что необходимо сделать в данной подпрограмме – сохранить в стеке нового режима все регистры из диапазона R0…R12, которые будут в ней использоваться. А уже затем можно приступать собственно к обработке исключительной ситуации.
После завершения обработки исключительной ситуации необходимо продолжить выполнение программы с прерванного места.

Однако в наборе команд ARM отсутствуют команды типа «возврат» или «возврат из подпрограммы», поэтому манипуляции со счетчиком команд PC необходимо осуществ-

лять, используя обычные команды.

Сброс (Reset)

Реализуется при подаче сигнала запуска на внешний вывод процессора.
Если сигнал nRESET имеет уровень LOW, то ARM7TDMI-S отбрасывает выполняемую команду.
Если же сигнал nRESET имеет уровень HIGH, то процессор выполняет следующие действия:
1. Устанавливает режим Supervisor (биты M[4:0] регистра CPSR устанавливаются в значение 10011)
2. Устанавливает биты I и F регистра CPSR (запрещает прерывания)
3. Очищает бит T регистра CPSR (переход в режим ARM)
4. PC выбирает следующую команду по адресу 0x00000000
5. восстанавливает режим ARM и возобновляет выполнение программы
После сброса определенны значения только PC и CPSR регистров.

Аварийное завершение

Аварийное завершение показывает, что текущий доступ к памяти не может быть завершен. Об этом сигнализирует внешний входной сигнал ABORT. (ARM7TDMI-S проверяет данную исключительную ситуацию в конце каждого цикла доступа к памяти).
Существует два типа аварийного завершения:
• Data Abort возникает при выборке данных.
• Prefetch Abort возникает при предварительной выборке команды

Data Abort

Ошибка при обращении к данным (фиксируется контроллером прерываний).
Данный механизм аварийного прерывания позволяет выполнять подкачку виртуальной системной памяти: если адрес данных не действителен, то менеджер памяти Memory Management Unit (MMU) инициирует аварийное прерывание, и обработчик выполняет действия, определенные данным прерыванием. После чего необходимо повторить прерванную команду.
При возникновении исключительной ситуации Data Abort, выполняемые действия зависят от типа выполняемой команды:
• команды одиночной передачи данных (LDR, STR) записывают изменения основных регистров. Об этом должен быть осведомлен обработчик.
• команда обмена (SWP) прекращается, так как не может быть выполнена. (Аварийное завершение должно выполниться при чтении SWP команды).
• команды блочной передачи данных (LDM, STM) выполняются полностью. Если команда будет перезаписывать базу с данными (когда имеется базовый регистр в списке передачи), то ARM7TDMI-S запретит перезапись.
При возникновении данного прерывания процессор переходит в режим Abort.
Возврат из обработчика осуществляется командой:
SUBS PC,R14_abt,#8
После возврата повторяет прерванную команду.

Быстрый запрос прерывания (FIQ)

Реализуется при подаче сигнала прерывания на один из выводов nFIQ (активный уровень LOW).
При возникновении данного прерывания процессор переходит в режим FIQ.

При поступлении данного прерывания в регистровом банке происходит переключение 7 регистров R8-R14 (подробней в разделе «Регистровая модель»), что позволяет сократить

или исключить операции сохранения содержимого регистров в стеке. За счёт этого переход к обработке прерывания происходит быстрее.
Возврат из обработчика осуществляется командой:
SUBS PC,R14_fiq,#4
После возврата повторяет прерванную команду.
Возможно отключение FIQ в привилегированном режиме при установке флага F в 1 в регистре CPSR.

Запрос прерывания (IRQ)
Возникает при обычном прерывании при уровне LOW на входе nIRQ.
При возникновении данного прерывания процессор переходит в режим IRQ.
Возврат из обработчика осуществляется командой:
SUBS PC,R14_irq,#4
После возврата повторяет прерванную команду.
Возможно отключение IRQ в привилегированном режиме при установке бита I в 1 в регистре CPSR.

Prefetch Abort
Ошибка при выборке команды (фиксируется контроллером прерываний).
При возникновении исключительной ситуации Prefetch Abort, ARM7TDMI-S помечает выбираемую команду как неверную, однако исключительная ситуация возникает только после того, как команда попадает на стадию выполнения в конвейере.
При возникновении данного прерывания процессор переходит в режим Abort.
Возврат из обработчика осуществляется командой:
SUBS PC,R14_abt,#4
После возврата повторяет аварийную команду

Неопределенная команда (Undefined instruction)
Возникает, если ARM7TDMI-S встречает команду, которая не может быть обработана системой.
Программное обеспечение может использовать механизм расширенной системы команд ARM, эмулирующий неопределенную команду сопроцессора.
При возникновении данного прерывания процессор переходит в режим Undefined.
Возврат из обработчика осуществляется командой:
MOVS PC,R14_und
После возврата выполняет команду, следующую за неопределенной командой.

Команда программного прерывания Software Interrupt (SWI)
Программное прерывание по команде SWI. Используются, как правило, для вызова функций ОС.
При возникновении данного прерывания процессор переходит в режим Supervisor.
Возврат из обработчика осуществляется командой:
MOVS PC, R14_svc
После возврата выполняет команду, следующую за SWI.

1.2.2 Система команд
Система команд состоит из 32-разрядных команд ARM и подмножества 16-разрядных команд Thumb, которые при выполнении транслируются в команды ARM.

Способы адресации

Таблица 1.5 — Способы адресации в ARM7

Обозначение	Название
#Imm	Непосредственная
Rn	Регистровая
Rn, shift #n	Регистровая с масштабированием
[Rn]	Косвенно-регистровая
[Rn,±Imm]	Преиндексная с непосредственным смещением
[Rn,±Rm]	Преиндексная с регистровым смещением
[Rn,±Rm, shift #n]	Преиндексная с масштабированным регистровым смещением
[Rn],±Rm	Постиндексная с регистровым смещением
[Rn],±Rm, shift #n	Постиндексная с масштабированным регистровым смещением

При непосредственной адресации операнд Imm входит в состав команды. При регистровой адресации в команде задаётся имя регистра Rn, содержимое которого является операндом или результатом операции.
Особенностью системы команд ARM является наличие регистровой адресации с масштабированием. При масштабировании содержимое регистра Rn сдвигается на число разрядов n, указанное в команде (от 1 до 31). Вместо "shift" в ассемблерном тексте используется один из четырёх символов, задающих вид реализуемого сдвига:
• LSL — логический сдвиг влево;
• LSR — логический сдвиг вправо;
• ASR — арифметический сдвиг вправо;
• ROR — циклический сдвиг вправо.
При выполнении арифметических и логических сдвигов последний выдвигаемый разряд поступает в регистр CPSR в качестве признака переноса C. При циклическом сдвиге бит C включается в цепь сдвига, только если число разрядов сдвига задано равным нулю. В этом случае выполняется циклический сдвиг операнда на один разряд вправо через бит C в регистре CPSR.
При косвенно-регистровой адресации содержимое указанного регистра Rn содержит адрес ячейки памяти, где хранится операнд или результат.
Индексная адресация имеет две разновидности: преиндексная и постиндексная. В случае преиндексной адресации адресом служит содержимое базового регистра Rn, которое индексируется перед выполнением операции путём сложения или вычитания непосредственного операнда Imm, содержимого регистра Rm или масштабированного содержимого Rm. Если в поле операнда содержится символ {!}, то индексированное содержимое Rn сохраняется после выполнения операции. В случае постиндексной адресации адресом служит содержимое базового регистра Rn, которое индексируется после выполнения операции.

Система команд ARM

Ядро ARM7TDMI-S является RISC-процессором, поэтому оно выполняет относи-

тельно небольшой набор команд.
Можно выделить следующие особенности системы команд:
1. Отсутствие аппаратной поддержки стека.
Стек организуется программно, причём в качестве указателя стека рекомендуется использовать регистр R13, хотя в принципе в этой роли может быть любой другой регистр общего назначения. Обычно обращение к стеку производится с помощью команд групповой пересылки STM и LDM.
2. Установка флагов условий.
Установка в регистре CSPR признаков N, Z, C, V по результатам выполнения операций производится при наличии в команде суффикса S. При его отсутствии признаки в регистре CPSR не изменяются.
3. Условное выполнение команд.
В состоянии ARM каждая команда поддерживает условное выполнение в зависимости от соответствия значения кодов условия регистра CPSR и поля условия команды. Это поле (биты 31:28) определяют условия, при которых команда должна выполняться. Если состояния C, N, Z и V флагов регистра CPSR удовлетворяют условиям поля условия, команда выполняется, иначе она игнорируется.
Существует 16 возможных условий, каждое из которых представляется двухсимвольным суффиксом, который может быть добавлен к обозначению команды.
Примечание 1: Условия "Выше", "Ниже", "Выше или равно", "Ниже или равно" используются при сравнении чисел без знака; условия "Больше", "Меньше", "Больше или равно", "Меньше или равно" - при сравнении чисел со знаком.
Примечание 2: 16-ое возможное условие зарезервировано и не используется.
Таким образом, команду «Переход» (B – в языке Ассемблера) можно преобразовать в команду BEQ, и она приобретет значение «Переход по равенству», т.е. выполнит переход только в случае, если установлен флаг Z.
При отсутствии суффикса, поле условия многих команд устанавливается в состояние «Всегда» (суффикс AL). Что означает, что команда выполняется всегда, не учитывая CPSR кодов условия.

Таблица 1.6 — Система команд ARM

Суффикс	Логическое выражение	Условие
EQ	Z = 1	Равно
NE	Z = 0	Не равно
CS	C = 1	Выше или равно
CC	C = 0	Ниже
MI	N = 1	Отрицательный результат
PL	N = 0	Положительный результат
VS	V = 1	Переполнение
VC	V = 0	Нет переполнения
HI	C = 1, Z = 0	Выше
LS	C = 0, Z = 1	Ниже или равно
GE	N = V	Больше или равно
LT	N ≠ V	Меньше
GT	Z = 0, N = V	Больше
LE	Z = 1, N ≠ V	Меньше или равно
AL		Всегда (безусловное выполнение)

Целью условного выполнения команд является обеспечение непрерывности потока команд через конвейер, т.к. при каждом выполнении команд перехода конвейер сбрасывается, и на его повторное заполнение требуется время, что резко снижает общую производительность. На практике существует некоторый порог, при котором принудительное «проталкивание» команд NOP через конвейер оказывается эффективней выполнения традиционных команд условного перехода и связанного с этим повторного заполнения буфера. Этот порог равен трем командам.

Все множество команд ARM можно разбить на 4 основные группы:

- Команды ветвления,
- Команды пересылки,
- Команды обработки данных,
- Команды поддержки сопроцессора

Команды ветвления

Команды передачи управления служат для изменения хода программы

Таблица 1.7 — Команды ветвления

Мнемокод	Команда	Операция
B	Переход	PC := PC + (rel<<1)
BL	Переход с сохранением адреса возврата в LR	LR := PC, PC := PC + (rel<<1)
BX	Переход с возможностью смены состояния	PC := Rn, T := Rn[0]
SWI	Программное прерывание	режим Supervisor, LR := PC, PC := 0x0008

Команда перехода B с соответствующим суффиксом обеспечивает выполнение условных или безусловных переходов (в диапазоне до 32 Мбайт как вперед, так и назад.).
Переход к подпрограмме осуществляется командой BL, при этом текущее содержимое программного счётчика PC (адрес возврата) (текущее значение PC, увеличенное на четыре) сохраняется в регистре связи LR.
В качестве операнда в этих командах задаётся 24-разрядное смещение rel, которое сдвигается на один разряд влево и после знакового расширения добавляется к текущему содержимому PC.
При вызове вложенных подпрограмм необходимо программным путём организовать сохранение промежуточных значений адресов возврата (содержимого LR) в стеке, используя регистр SP в качестве указателя. Также программным путём обеспечивается, в случае необходимости, сохранение в стеке содержимого регистров. Замечание: Заданный адрес перехода должен быть чётным, если процессор находится в состоянии THUMB, или кратным четырём, если процессор в состоянии ARM.
Команда BX позволяет осуществить переход с одновременным изменением состояния процессора. Адрес перехода (чётный или кратный четырём) определяется разрядами 31-1 или 31-2 содержимого регистра Rn, заданного в команде, а состояние процессора - нулевым битом этого регистра, который копируется в регистр CPSR в качестве бита T.
Замечание 1: это единственный способ, который можно использовать для изменения используемого набора команд, т.к. непосредственные манипуляции с флагом T регистра CPSR могут привести к непредсказуемым результатам.

Команда программного прерывания SWI используется для доступа к функциям ОС.
При выполнении данной команды процессор переходит в режим Supervisor, запоминает адрес возврата в регистре LR и переходит на адрес 0x0008. По этому адресу располагается команда перехода на обработчик прерывания.
Замечание 2: выполнение команды SWI - единственный способ, позволяющий перевести процессор из режима User в привилегированный режим Supervisor.

Команды пересылки

Из группы команд пересылки наиболее интересны команды групповой загрузки–сохранения содержимого регистров LDM и STM. Они позволяют пересылать содержимое нескольких регистров, перечисленных в команде: в формате команды есть 16-бит поле, в котором каждый бит соответствует одному из регистров R15-0. Если бит равен единице, содержимое данного регистра сохраняется в памяти (по команде STM) или загружается из памяти (по команде LDM).

Таблица 1.8 — Команды пересылки

Мнемокод	Команда	Операция
LDM	Групповая загрузка содержимого регистров из памяти	
LDR	Загрузка регистра из памяти	Rd := (адрес)
MOV	Пересылка регистра или константы	Rd := Op2
MRS	Пересылка из CPSR или SPSR в регистр	Rn := CPSR (SPSR)
MSR	Пересылка из регистра в CPSR или SPSR	CPSR (SPSR): = Rm
MVN	Пересылка с побитной инверсией	Rd := НЕ Op2
STM	Групповое сохранение содержимого регистров в памяти	
STR	Пересылка из регистра в память	<адрес> := Rn
SWP	Обмен содержимого регистра и памяти	Rd := [Rn], [Rn] := Rm

Из группы команд пересылки наиболее интересны команды групповой загрузки–сохранения содержимого регистров LDM и STM. Они позволяют пересылать содержимое нескольких регистров, перечисленных в команде: в формате команды есть 16-бит поле, в котором каждый бит соответствует одному из регистров R15-0. Если бит равен единице, содержимое данного регистра сохраняется в памяти (по команде STM) или загружается из памяти (по команде LDM).
При записи в регистр CPSR с помощью команды MSR в режиме User изменяются только признаки N, Z, V, C. Остальные биты сохраняют ранее установленное значение. В любом режиме изменение бита T с помощью команды MSR не реализуется. Для программной смены состояния процессора используется команда BX (табл. 5.2).
Благодаря команде обмена (SWP) обеспечивается поддержка семафоров реального времени. Эта атомарная команда осуществляет одновременный обмен содержимого регистра и памяти (позволяет также переставлять местами содержимое двух регистров). Она выполняется за два такта, однако обрабатывается как одна элементарная команда. Благо-

даря такому решению предотвращается прерывание процесса обмена критическими данными при возникновении исключительной ситуации.
Замечание: Эта команда недоступна из языка Си и поддерживается встроенными функциями библиотеки компилятора.

Команды обработки данных

При выполнении арифметических и логических операций (табл. 5.4) один из операндов размещается в регистре, второй Op2 - в регистре, ячейке памяти или задаётся непосредственно. Результат всегда размещается в регистре Rd.

Таблица 1.9 — Команды обработки данных

Мнемокод	Команда	Операция
ADC	Сложение с учётом переноса	Rd := Rn+Op2+C
ADD	Сложение	Rd := Rn+Op2
AND	Логическое И	Rd := Rn AND Op2
BIC	Очистка битов (маскирование)	Rd := Rn AND (НЕ Op2)
CMN	Сравнение с отрицательным числом	CPSR flags := Rn+Op2
CMP	Сравнение	CPSR flags := Rn–Op2
EOR	Исключающее ИЛИ	Rd := Rn XOR Op2
MLA	Умножение с накоплением	Rd := (RmrRs)+Rn
MUL	Умножение	Rd := RmxRs
ORR	Логическое ИЛИ	Rd := Rn OR Op2
RSB	Обратное вычитание	Rd := Op2–Rn
RSC	Обратное вычитание с заёмом	Rd := Op2–Rn–1+C
SBC	Вычитание с заёмом	Rd := Rn–Op2–1+C
SUB	Вычитание	Rd := Rn - Op2
TEQ	Побитное сравнение	CPSR flags:= Rn XOR Op2
TST	Тестирование битов	CPSR flags:= Rn AND Op2

Замечание: процессор не выполняет операцию деления, поэтому ее необходимо реализовывать программным путём.
Ядро ARM выполняет несколько разновидностей операции умножения.
Существуют две основные команды:
• простое умножение MUL
• умножение с накоплением MLA.
Любая из этих команд может иметь либо короткую (результат записывается в один регистр с потерей старших разрядов), либо длинную форму (произведение записывается в два регистра). При записи команды длинная форма обозначается суффиксом L.
В первом случае операции умножения можно производить как со знаковыми, так и с беззнаковыми целыми числами.
Длинная форма команды умножения имеет два варианта — знаковый и беззнаковый, которые отличаются префиксом: S и U, соответственно. Таким образом, команда умножения имеет следующие разновидности: MUL, SMULL, UMULL. Аналогично, команда умножения с накоплением имеет разновидности MLA, SMLAL, UMLAL.
Команда обратного вычитания RSB позволяет изменить порядок записи операндов

в команде вычитания. Она соответствует команде "Вычесть из операнда Op2 содержимое регистра Rn и записать в регистр Rd".
Операция CMN позволяет сравнить два операнда, у одного из которых при сравнении изменяется знак.
Бит S используется для управления флагами условий. Если этот бит установлен, флаги условий изменяются в соответствии с результатом выполнения команды.
Замечание: если этот бит установлен, а в качестве регистра результата указан счетчик команд (R15), производится копирование содержимого регистра SPSR текущего режима в регистр CPSR. Эта возможность используется для восстановления PC и переключения в исходный режим в конце обработки исключительных ситуаций.
Замечание: в ЦПУ ARM7 имеется многорегистровое устройство циклического сдвига (barrel shifter), позволяющее при выполнении команды сдвигать значение второго операнда на величину до 32-х битов.

1.2.3 Контроллер векторных прерываний
PrimeCell Контроллер векторных прерываний (VIC) обеспечивает программный доступ к системе прерываний. В системе ARM доступны два уровня прерываний:
. • Запрос на быстрое прерывание (FIQ) для быстрой обработки прерывания
• Запрос Прерывание (IRQ) для обычных прерываний.
Замечание: для обеспечения действительно быстрых прерываний в системе должен присутствовать только один FIQ источник. Это даст следующие преимущества:
. • Возможность начать обслуживание прерывания, непосредственно, не определяя источник прерывания.
. • Будет уменьшено время ожидания прерывания. Кроме того, можно будет использовать пустые регистры, доступные для FIQ прерываний более эффективно, потому что не потребуется сохранять контекст.
В контроллере прерываний есть 32 линии прерываний. Программное обеспечение контроллера прерываний может управлять каждой линией запроса, для этого VIC использует битовые позиции для каждого различного источника прерывания.
В контроллере поддерживаются 16 векторных IRQ прерываний. Векторные и невекторные IRQ прерывания предоставляют контроллеру адрес Прерывающей программы (ISR).
Чтение из векторного регистра адреса прерывания, VICVectAddr, обеспечивает адрес ISR, и обновляет настройки контроллера, который маскирует любые запросы на прерывание имеющие такой же или более низкий приоритет.
Запись в регистр VICVectAddr, указывает контроллеру, что текущее прерывание обслужено, и позволяет прерываниям с более низкими приоритетами активизироваться.
Прерывание FIQ имеет самый высокий приоритет, сопровождаемый вектором прерывания 0, чтобы прервать вектор 15.Не векторные IRQ прерывания имеют самый низкий приоритет.
Программный вызов прерываний позволяет вам производить программный контроль прерываний. Обычно этот регистр используется для того, чтобы понизить прерывание FIQ до прерывания IRQ.
Примечание: то, что приоритет FIQ выше, чем IRQ установлено ARM. VIC может поднять оба, и FIQ, и IRQ одновременно.
И у IRQ, и FIQ логики есть асинхронный канал. Это позволяет прерываниям функционировать, когда отключена синхронизация.
В соответствии с соглашением, для прерывания IRQ, биты 1-5, должны использоваться так, как определено в Таблице 2-1. Использование бита 0, и битов начиная с 6ого – произвольно. Для прерывания FIQ, биты могут использоваться произвольно.

Логика запроса прерываний

Interrupt request logic (Логика запроса прерываний) получает запросы на прерывание от периферии и объединяет их с программными запросами на прерывание. После этого происходит маскирование не активных прерываний, а активные прерывания направляются к IRQ или FIQ. Логика запроса не векторных FIQ прерываний генерирует сигнал прерывания FIQ, объединяя запросы на прерывание FIQ в контроллере прерывания и запросы на прерывание от каскадно-связанных контроллеров. Логика запроса не векторных IRQ прерываний объединяет не векторные запросы на прерывание и генерирует не векторный сигнал прерывания. Далее этот сигнал посылается по адресу векторного прерывания IRQ и на логику приоритета.
В контроллере прерываний есть 16 блоков векторных прерываний. Блоки векторных прерываний получают запросы на IRQ прерывание и устанавливают VectIRQx, только при следующих условиях:
. • выбранное прерывание активно
. • выбранное прерывание - в настоящее время имеет самый высокий приоритет.
Каждый блок векторных прерываний также обеспечивает VectorAddrx [31:0] выход для использования в блоке приоритета прерывание.

Программные прерывания

Программное обеспечение может управлять линиями прерываний, и таким образом генерировать программные прерывания.
Замечание: эти прерывания генерируются до маскирования прерываний, так же как и внешние прерывания.
Программные прерывания очищаются посредством записи в специальный регистр очистки программных прерываний, VICSoftIntClear (см. регистр очистки программных прерываний, VICSoftIntClear). Обычно это делается в конце обслуживания прерывающей подпрограммы.

1.2.4 Шинная архитектура

AMBA (Advanced Microcontroller Bus Architecture) - шина разработанная фирмой ARM для организации эффективного взаимодействия компонентов приборов, построенных на базе ядер фирмы. Шина AMBA - стандартная встроенная ASIC шина обеспечивающая быстрое модульное проектирование систем при упрощении многократного использования схемотехники и тестов. ARM также обеспечивает возможность использования библиотеки PrimeCell периферии, которая соответствует AMBA стандарту и обеспечивают простую разработку ASIC и ASSP. При использовании AMBA с синтезируемыми версиями периферийных устройств, аппаратные средства системы и программное обеспечение могут быть разработаны на начальном этапе проектирования и, следовательно, может быть снижен риск ошибок проектирования конечной системы.
Типовая шина AMBA содержит системную шину (в данном случае AHB) и шину периферии (APB).
Системная шина соединяет встраиваемые процессоры, такие как ARM ядра, с высокопроизводительной периферией, контроллерами DMA, встроенными памятью и интерфейсами. Это высокоскоростная, с широкой полосой пропускания шина, поддерживающая, для обеспечения максимальной производительности, управление с большим количеством ведущих устройств (Multi-master bus management).
Шина периферии - работает с упрощенным протоколом и разработана для организации интерфейса с периферийными устройствами общего назначения или дополнительными периферийными устройствами. С системной шиной она соединяется через мост (bridge),

способствующий снижению потребления системы.
В спецификации шины AMBA определена методология тестирования, обеспечивающая быстрое тестирование модулей и кэш.
APB предназначена для организации интерфейса с встроенными периферийными устройствами общего назначения, такими как таймеры, контроллеры прерываний, UART, порты I/O и т.п., и дополнительными периферийными устройствами. С основной системной шиной шина периферии соединяется мостом, обеспечивающим разгрузку системной шины и снижающим общее потребление системы.
В соответствии с новой спецификацией AMBA Rev 2.0 Specification шина отвечает современным требованиям последовательности синтеза приборов класса "система-на-кристалле".
Простая шина
Бесконвейерная архитектура
Простое использование - все периферийные устройства обслуживаются как ведомые
Малое количество используемых вентилей
Малое потребление
Снижение загрузки основной системной шины за счет ее изоляции от периферии мостом
Сигналы на шине периферии активны только во время медленных пересылок периферии

1.3 Особенности микроконтроллеров LPC21XX

1.3.1 Система памяти
FLASH размещается с адреса 0x00000000, ОЗУ – с адреса 0x40000000. Начиная с адреса 0x7FFFD000, размещается boot загрузчик, область 0xE0000000 …0xE0200000 – периферия VHB, верхняя область памяти – периферия AHB.
Примечание: регистры специальных функций состоят из трех регистров: установки, сброса и статуса. Сброс и установка битов осуществляется записью 1 в соответствующий регистр, чтение производится из регистра статуса.
Область векторов прерываний занимает младшие 64 байта памяти, однако имеется несколько режимов отображения, управляемых регистром MEMMAP, в которых на эту область памяти отображаются соответствующие регионы ОЗУ, boot загрузчика или внешней памяти, что позволяет изменять расположение программ.
При сбросе всегда управление передается загрузчику, который проверяет сторожевой таймер, состояние P0.14 и корректность содержимого FLASH памяти. Последняя процедура заключается в вычислении сигнатуры (доп. код суммы) таблицы векторов прерываний и сравнении с содержимым двойного слова по адресу 0x00000014. Загрузчик позволяет загрузить память контроллера через модуль USART путем выполнения специальных команд. Команды загрузчика можно также вызывать из программы пользователя
С помощью внешних выводов микроконтроллера возможно также использовать внутренний загрузчик или загрузчик из внешней памяти (если имеется возможность подключения внешней памяти).

Модуль акселерации памяти (MAM)

Модуль MAM пытается своевременно обеспечить в своей локальной памяти наличие следующей команды ARM, которую должен будет выполнять ЦПУ. Для этого вся FLASH–память разбивается на два банка, доступ к которым может осуществляться независимо (т.е. при одном обращении к FLASH–памяти может загружаться 4 команды ARM или 8 команд THUMB). Код программы распределяется между двумя банками таким образом, чтобы при выполнении линейного кода одновременно осуществлялось выполнение

команд, считанных модулем MAM из одного банка, и предварительная выборка следующего 128 – битного блока команд из другого. За счет этого гарантируется, что к моменту завершения предыдущих 128 бит кода указанные команды будут готовы к выполнению. Для поддержки коротких циклов и переходов в модуле MAM предусмотрены специальные буферы, где хранятся последние загруженные команды, которые, при необходимости, можно выполнить повторно.
Вся структура MM прозрачна для пользователя, а его конфигурация определяется двумя регистрами – регистром тайминга MAMTIM и регистром управления MAMCR. Кроме того, имеется несколько дополнительных регистров, содержащих информацию о текущей эффективности работы модуля
Операции программирования флэш не контролируют модуль MAM, но манипулируют им как отдельной функцией. Сектор “Загрузочный блок” содержит алгоритмы программирования флэш, которые могут быть вызваны как часть прикладной программы, и загрузчика, который может быть выполнен для разрешения последовательного программирования флэш-памяти.
Флэш-памяти связаны так, чтобы каждый сектор существовал в обоих банках, и чтобы операция стирания сектора действовала на оба банка одновременно. В действительности, существование двух банков прозрачно для функций программирования.
При изменении MAM синхронизации, MAM должен быть сначала выключен - записью нуля в MAMCR. Тогда новое значение может быть записано в MAMTIM. Наконец, MAM может быть включен снова, записью значения (1 или 2) приводящему к желательному операционному режиму в MAMCR.
Для системной частоты меньше 20 МГц, MAMTIM может быть 001. Для системной частоты между 20 МГц и 40 МГц предложено время доступа к флэш 2 CCLKs, в то время как в системах с системной частотой больше чем 40 МГц предложены 3 CCLKs.

1.3.2 Схема ФАПЧ

На вход схемы фазовой автоподстройки частоты (Phase Locked Loop – PLL), или сокращенно ФАПЧ, подается сигнал частотой 10…25 МГц, полученный с помощью кварцевого резонатора, а на выходе формируется сигнал частотой до 60 МГц, обеспечивающий синхронизацию (тактирование) ЦПУ ARM7 и периферийных устройств. Благодаря этому микроконтроллеры могут работать на максимально допустимой частоте, используя внешние генераторы с относительно низкой частотой. Таким образом, снижается до минимума электромагнитное излучение микросхем. Кроме того, выходная частота схемы ФАПЧ может динамически изменяться, что позволяет замедлять работу устройства для экономии энергии в ждущем режиме.
Работа схемы ФАПЧ определяется двумя константами, значениями которых необходимо задаться для определения частоты тактового сигнала ЦПУ и шины AHB этот сигнал называется CCLK. Первая константа является коэффициентом умножения. После этой операции выходная частота сигнала ФАПЧ определяется выражением CCLK=M * FOSC. В цепи обратной связи схемы ФАПЧ включен генератор, управляемый током (CCO), который должен работать в диапазоне 156…320 МГц. Вторая константа определяет коэффициент деления программируемого делителя, который обеспечивает нахождение частоты CCO в заданных пределах. Рабочая частота CCO определяется выражением: FCCO=CCLK*2*P.
Значения, записанные в пользовательские РСФ, не будут переписаны во внутренние регистры блока ФАПЧ до тех пор, пока в регистр запуска не будет записана определенная последовательность чисел. Для инициализации схемы ФАПЧ необходимо записать соответствующие значения констант P и M в регистр PLLCFG, после чего разрешить работу схемы, используя регистр PLLCON. (рисунок). В результате этих действий схема

ФАПЧ запустится, однако для установки стабильного сигнала, пригодного для использования в качестве тактового сигнала CCLK , потребуется некоторое время. Процесс запуска схемы ФАПЧ можно контролировать, отслеживая состояние бита LOCK регистра PLLSTATUS. Установка этого бита означает, что выходной сигнал схемы ФАПЧ можно использовать в качестве основного источника тактовых импульсов. При захвате частоты может также генерироваться прерывание. Как только тактовый сигнал стабилизируется, его можно будет использовать в качестве источника CCLK вместо внешнего генератора. Это переключение осуществляется с помощью бита PLCC регистра PLLCON.
Необходимо аккуратно подходить к выбору значений, сохраняемых в регистре PLLCFG. Числа, записываемые в регистр, соответствуют значениям P-1 и M-1, благодаря чему гарантируется, что константы P и M никогда не будут равны нулю. Кроме того, значение M является 5-битным, так что младший бит значения P располагается не на границе полубайтов.
При переходе микроконтроллера в режим “Power Down” схема ФАПЧ выключается и отключается от линии тактового сигнала. При выходе из спящего режима работа схемы ФАПЧ не возобновляется, так что при каждом выходе из спящего режима необходимо выполнять ту же последовательность операций, что и при инициализации.

2 Интерфейсы передачи данных

Все интерфейсы можно разделить на последовательные и параллельные. Параллельные предполагают одновременную передачу всего слова данных, сопровождаемую сигналом синхронизации. Достоинства: высокая скорость, недостаток небольшие расстояния передачи данных из-за внешних помех, взаимного влиянияотдельных линий друг на друга и неодинаковой скорости распространения отдельных сигналов. Основное применение - внутренние интерфейсы МП систем и передача данных на небольшие расстояния (CENTRONIX). В настоящее время параллельные интерфейсы используются преимущественно для внутренних связей, требующих высокоскоростных передач.

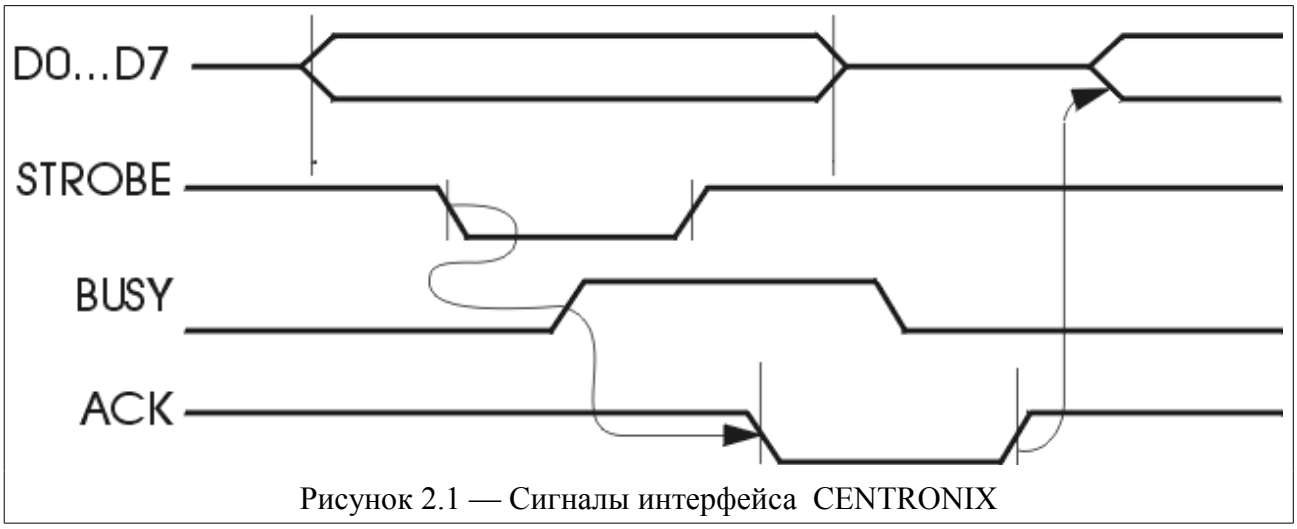


Рисунок 2.1 — Сигналы интерфейса CENTRONIX

Последовательные интерфейсы предназначены для передачи данных побитно. Можно разделить эти интерфейсы на синхронные и асинхронные. Синхронные интерфейсы имеют два канала передачи данных - собственно данные и сигнал синхронизации. По сравнению с асинхронными имеют более высокую скорость и надежность передачи данных. Недостаток - необходимость нескольких линий дляпередачи.

Асинхронные интерфейсы предполагают один канал передачи данных, приэтом передатчики и приемник должны синхронизироваться, используя канал данных. При этом используются следующие способы:

1. Код NRZ (без возвращения к нулю) Достоинства - простая реализация и минимальные требования к пропускной способности канала. Недостаток - невозможность извлечения синхросигнала из передаваемых данных (вероятная потеря синхронизации при передаче длинных пакетов данных). Для синхронизации используется стартовый служебный бит. Служит основой для реализации RS-232.

2. RZ - трехуровневый код. Особенность - в наличии возврата к нулю при передаче любого бита (нет рассинхронизации при передаче пакетов любой длины). Недостаток - требуется двойная полоса пропускания канала связи. Основное применение в оптоволоконных сетях.

3. Код "Манчестер-II". Самосинхронизирующийся код с двумя уровнями сигнала. Единица - отрицательный переход в центре бита, нуль - положительный. Также требуется двойная полоса пропускания, однако отсутствует постоянная составляющая, что позволяет использовать трансформаторы для гальванической развязки сигналов.

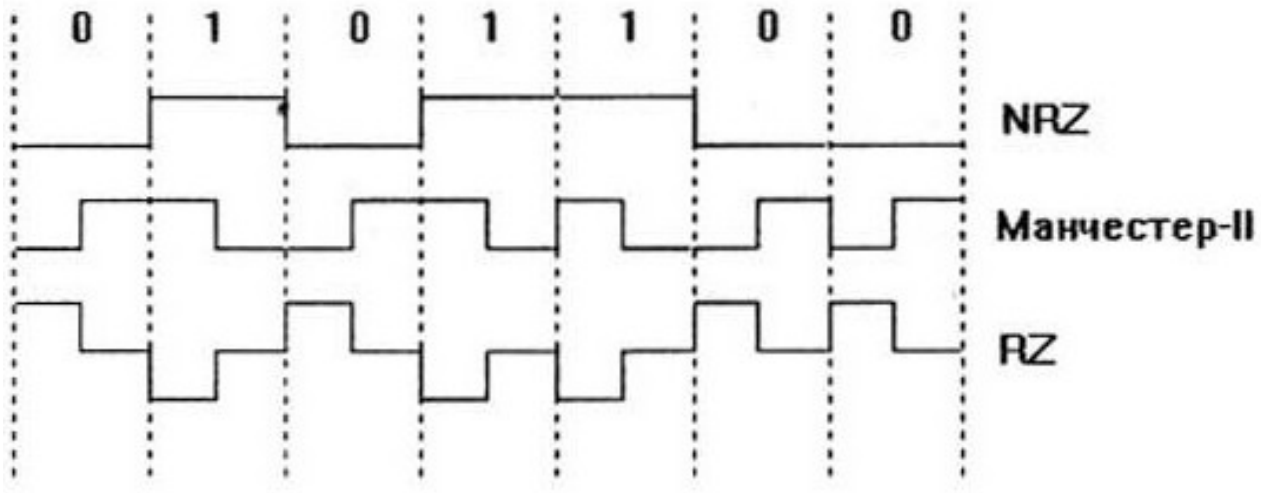


Рисунок 2.2 — Наиболее распространенные коды передачи информации

На рис. 3 представлены схемы шифраторов, а на рис. 4 — дешифраторов кода Манчестер.

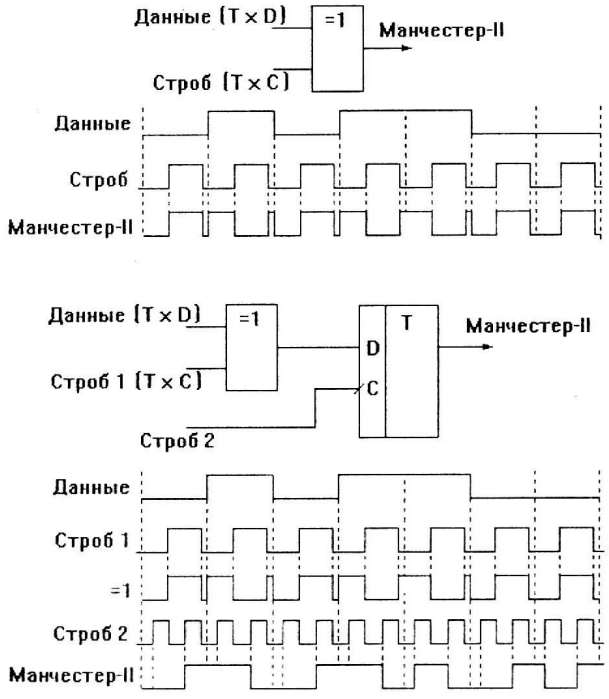


Рисунок 2.3 — Шифраторы кода Манчестер II

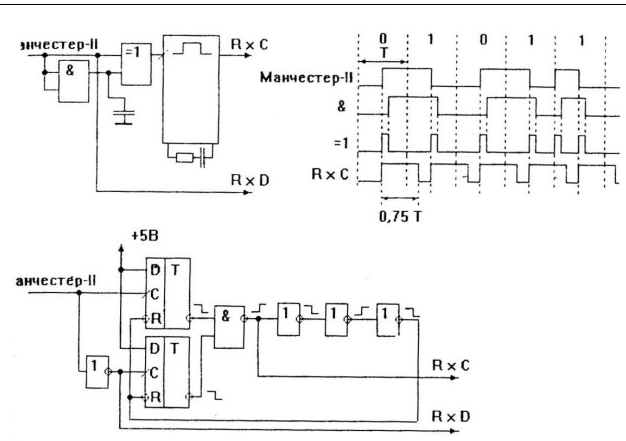


Рисунок 2.4 — дешифраторы кода Манчестер II с использованием формирователей временных задержек

В первой схеме на рис. 4 применяемый одновибратор не должен иметь перезапуск, во второй - можно и с перезапуском.

2.1 Интерфейсы RS-232, RS-422, RS-423, RS-485

Все эти интерфейсы используют код NRZ, и структура посылки имеет вид, представленный

на рис. 5.

Рисунок 2.5 — Формат асинхронной передачи

Скорости передачи стандартизованы (110, 150, 300, 600…38400, 57600, 115200 бит/с).
Возможен (но редко используется синхронный вариант передачи данных).
Стандарты последовательного интерфейса приведены на рис. 6.

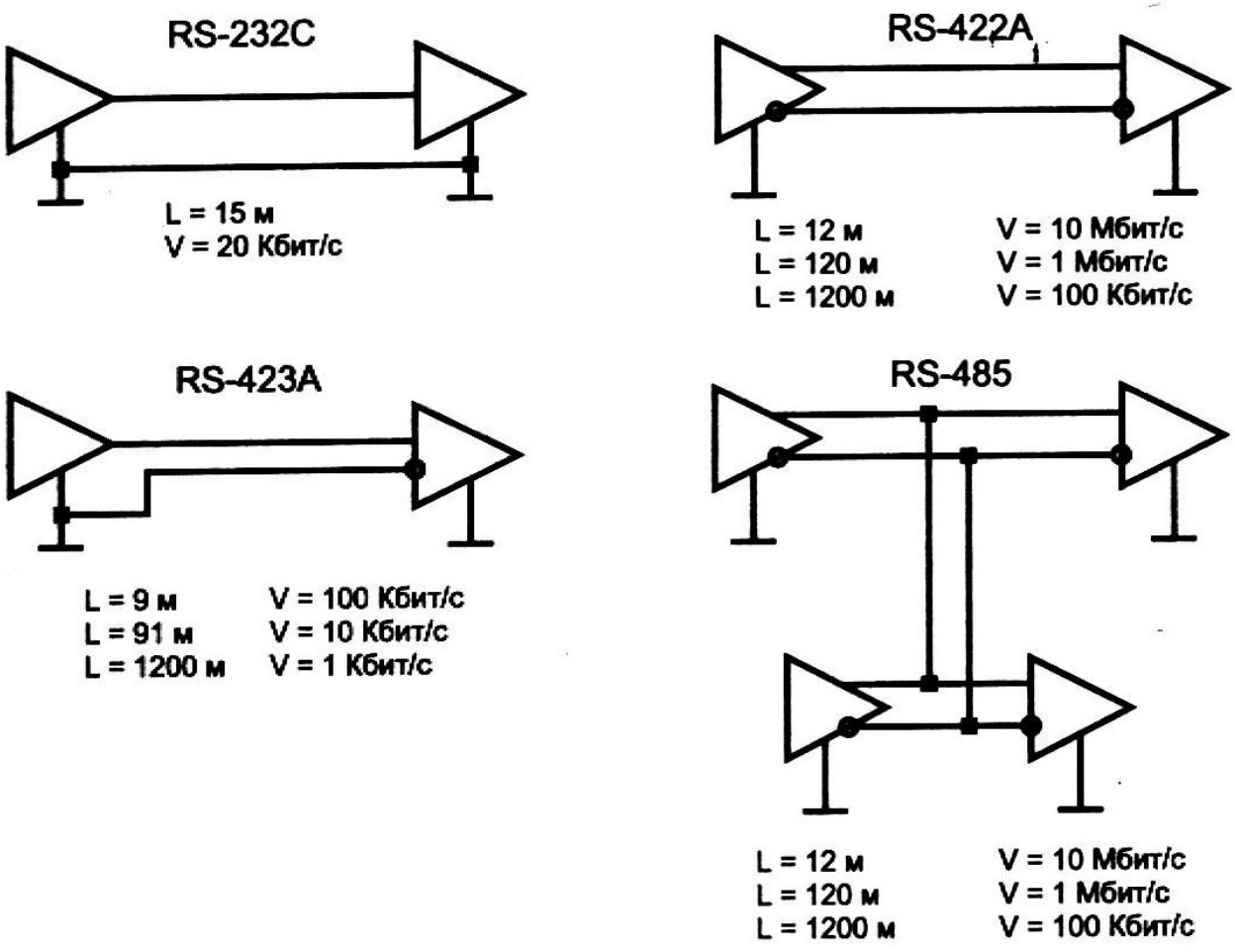


Рисунок 2.6 — Стандарты последовательного интерфейса

RS-232 - один из наиболее распространенных интерфейсов, реализован во всех персональных компьютерах (COM - порт). На физическом уровне RS-232 использует два уровня сигналов - (±3…12В), причем положительное напряжение соответствует нулю. Наиболее часто используются интерфейсы RS-232 и RS-485.

Протокол обмена данными предусматривает как односторонний, так и двусторонний обмен данными. Возможны аппаратный и программный виды контроля потока данных. При

реализации аппаратного метода используют дополнительные сигналы (RTS, CTS, DTR, DSR), останавливающие и возобновляющие обмен данными. При программном контроле используются специальные символы (XON(11H), XOFF (13H)), причем, естественно, в потоке данных они не должны встречаться, как несущие иную информацию, что требует применения специальных алгоритмов фильтрации данных. Как правило, поддержка описанных протоколов возлагается на системный сервис МП системы, а аппаратная реализация на специальные программируемые контроллеры (USART) (типа КР580ВВ51). Для согласования цифровых сигналов, действующих в системе, и сигналов интерфейсов используются специальные интерфейсные микросхемы (ADM232, ADM485 и др.).

2.2 Интерфейс CAN

CAN (Controller Area Network) - это последовательный протокол связи с эффективной поддержкой распределения контроля в реальном времени и очень высоким уровнем безопасности. Основное назначение: организация передачи информации в сложных условиях, таких как среды с высоким уровнем различного рода помех. Этот протокол передачи применяется в автомобильной электронике, машинных устройствах управления, датчиках при передаче информации со скоростями до 1 Мбит/сек.

Протокол CAN можно разделить на следующие уровни:

- ♦объектный уровень
- ♦канальный уровень
- ♦физический уровень

Объектный и канальный уровни включают весь сервис и функции передачи данных определяемых ISO/OSI моделью. Область объектного уровня включает:

- ♦Поиск сообщений для передачи.
- ♦Фильтрация сообщений, полученных от канального уровня
- ♦Обеспечение связи между прикладным уровнем и аппаратными средствами.

Объектный уровень можно реализовывать различными способами.

Область канального уровня главным образом - протокол передачи, т.е. управление кадрами, выполнение арбитража, проверка и сигнализация ошибок, типизация ошибок. Внутри канального уровня решается, является ли шина свободной для начала новой передачи. Все что находится внутри канального уровня, не имеет ни какой свободы к модификации.

Область физического уровня - фактическая передача бит между различными узлами. Внутри одной сети физический уровень должен быть одинаков для всех узлов. Физический уровень можно реализовать различными способами.

На базе CAN стандартизованы протоколы ISO11898 (для высокоскоростных приложений), ISO11519 (для низкоскоростных приложений) и J1939 (для грузовиков и автобусов). Все эти протоколы используют в качестве физического носителя дифференциальную электрическую линию передачи. CAN относится к семейству протоколов CSMA/CD Carrier Sense Multiple Access with Collision Detection (прослушивание несущей с разрешением коллизий). Это означает, что передатчик может начинать передачу, только если линия свободна в течение определенного интервала времени, а если два и более узлов начинают передачу одновременно, то они должны обнаружить конфликт и предпринять соответствующие действия. Протокол CAN предусматривает недеструктивный арбитраж, что означает, что обнаружение коллизии не ведет к повреждению передаваемого сообщения.

2.2.1 Основные особенности

Сообщения

Информация по шине посылается в фиксированном формате сообщений различной, но фиксированной длины. Когда шина свободна, любой узел может начать передачу нового сообщения.

Уровни шины

Шина может принимать одно из дополняющих друг друга значений: "dominant" и "recessive". В случае одновременной подачи "dominant" бита и "recessive" бита, возникающее в результате значение шины будет "dominant". Для электрического носителя обычно "recessive" соответствует "1", а "dominant" - "0".

Синхронизация

CAN использует для синхронизации "bit stuffing" (вставка бита), при которой за пятью последовательными битами одной полярности вставляется бит противоположной полярности. Этот бит используется приемниками сообщений для синхронизации, но игнорируется, как носитель информации.

Информационная маршрутизация

В CAN нет никакой информации относительно конфигурации сети (например, адреса узла). Это имеет несколько важных следствий:

Гибкость системы:

Узел может быть добавлен в CAN - сеть, без каких либо изменений в программном или аппаратном обеспечении, какого - либо узла в сети.

Маршрутизация сообщений:

Содержание сообщения определяется идентификатором. Идентификатор не указывает адреса сообщения, а описывает значение данных так, чтобы все узлы сети были способны решить фильтрацией сообщений, нужны им эти данные или нет.

Передача группе:

Как следует из фильтрации сообщений, любое число узлов может одновременно получать и реагировать на одно и тоже сообщение.

Скорость передачи информации

Скорость передачи информации в CAN - сети может быть различной для каждой сети. Однако в каждой конкретной сети скорость передачи информации фиксирована.

Приоритеты

Идентификатор и RTR - бит определяют статический приоритет сообщения в течение доступа к шине.

Удаленный запрос данных

Посылая кадр удаленного запроса данных, узел может потребовать данные от другого узла. Кадр данных и кадр удаленного запроса данных должны иметь одинаковый идентификатор.

Multimaster

Когда шина свободна, любой узел может начать передачу сообщения. Доступ к шине получает узел, передающий кадр с наивысшим приоритетом.

Арбитраж

Когда шина свободна, любой узел может начать передачу сообщения. Если два или больше узла начинают передавать сообщения в одно и тоже время, конфликт при доступе к шине будет решен поразрядным арбитражем используя идентификатор и RTR - бит. Механизм арбитража гарантирует, что ни время, ни информация не будут потеряны. Если кадр данных и кадр удаленного запроса данных начинают передаваться в одно время, то кадр данных имеет более высокий приоритет, чем кадр удаленного запроса данных. В течение арбитража каждый передатчик сравнивает уровень переданного бита с уровнем, считываемым с шины. Если эти уровни одинаковы, узел может продолжать посылать данные дальше. Если был послан уровень лог. '1' (recessive), а с шины считан уровень лог. '0' (dominant), то узел теряет право дальнейшей передачи данных и должен прекратить посылку

данных на шину.

Соединения

Линия связи по протоколу CAN - это шина, к которой может быть подключён ряд узлов. Количество узлов не имеет никакого теоретического предела. Фактически количество узлов будет ограничено временами задержек и/или электрической нагрузкой на линии шины. Способ, которым выполнена шина, не установлен в данной спецификации. Например, это может быть одиночный провод (+земля), два дифференциальных провода, оптическое стекловолокно.

Подтверждение

Все приёмники проверяют непротиворечивость принимаемого сообщения и подтверждают непротиворечивое сообщение.

2.2.2 Обработка ошибок

Узлы в сети, объединенной интерфейсом CAN, могут обнаруживать ошибки и реагировать на них несколькими способами. При кратковременных сбоях работоспособность восстанавливается автоматически, при серьезных ошибках узел может отключиться от сети, таким образом, изолируя источник ошибки. CAN определяет пять видов возможных ошибок и три состояния, в которых могут находиться узлы, в зависимости от серьезности и количества ошибок.

Типы ошибок

Ошибка CRC

Если значение, указанное в поле CRC не совпадает со значением, вычисленным узлом, то генерируется кадр ошибки. Если хотя бы одно устройство обнаружило такую ошибку, то сообщение повторяется через определенный интервал времени.

Ошибка подтверждения

После передачи сообщения передатчик проверяет значение бита в поле подтверждения. Если значение этого бита - "dominant", то это значит, что ни один узел не подтвердил получения сообщения. При этом генерируется кадр ошибки, и сообщение повторяется через определенный интервал времени.

Ошибка кадра

При получении доминантного бита в некоторых определенных служебных полях сообщения (конец кадра, межкадровый интервал, разделители CRC и поля подтверждения), констатируется состояние ошибки, и сообщение повторяется через определенный интервал времени.

Битовая ошибка

Фиксируется передатчиком при несовпадении передаваемого бита и состояния шины. Если при передаче поля арбитража или поля подтверждения фиксируется доминантный уровень, то состояние ошибки не фиксируется (нормальное состояние при арбитраже или подтверждении приема), в других случаях генерируется кадр ошибки и сообщение поторяется.

Ошибка синхронизации

Если при передаче данных приемник обнаруживает шесть и более последовательных бит одной полярности, то фиксируется состояние ошибки и сообщение повторяется.

Состояния узлов

Активный узел (ERROR ACTIVE)

Узел в этом состоянии активно работает на шине, обменивается сообщениями, генерирует кадры ошибок с активным флагом, вынуждая другие узлы также отвечать на них.

Это состояние является нормальным состоянием узла и требует, чтобы значения счетчиков ошибок передачи и приема не превышали 128.

Пассивный узел (ERROR PASSIVE)

Узел становится пассивным, когда значения счетчиков превышают 128. Такому узлу не разрешается передавать активный флаг ошибки, он должен передавать пассивный флаг ошибки. Когда такой узел является единственным передатчиком на шине, то его флаг нарушает синхронизацию и другие устройства передают ответные кадры ошибок, если он является приемником или имеются другие передатчики, то его флаг не влияет на состояние шины.

Отключенный узел

Узел переходит в это состояние, когда значение счетчика ошибок передачи превышает 255. В этом состоянии узел не може передавать и принимать сообщения и генерировать какие-либо кадры ошибок. Протокол CAN предусматривает специальную последовательность восстановления шины, когда такой узел возвращается в активное состояние после устранения проблемы.

2.2.3 Формат сообщений

При передаче информации с помощью протокола CAN используется четыре типа кадров. Кадр данных содержит данные, передаваемые передатчиком приёмнику (ам). Кадр удаленного запроса данных передается на шину для запроса передачи кадра данных с тем же самым идентификатором. Кадр ошибки передаётся при обнаружении ошибки на шине. Кадр перегрузки используется для обеспечения дополнительной задержки между предшествующим и последующим кадрами данных или кадрами удаленного запроса данных. Кадры данных и кадры удаленного запроса данных отделяются от предшествующих кадров межкадровым пространством.

Структура кадра данных

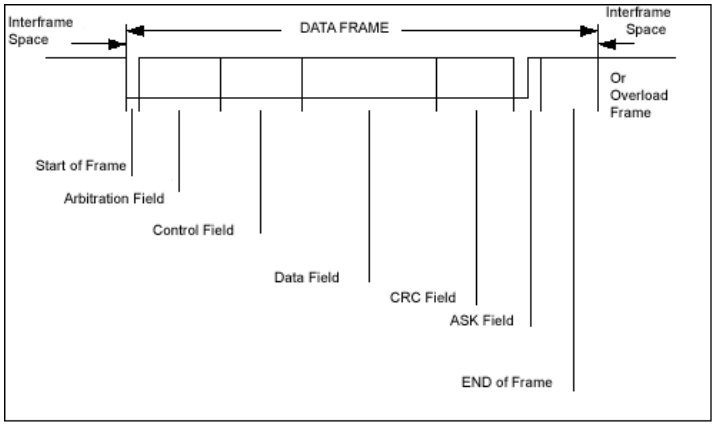


Рисунок 2.7 — Кадр данных (DATA FRAME)

Кадр данных состоит из 7 различных полей: "Начало кадра", "поле арбитража", "поле контроля", "поле данных", "поле CRC", "поле подтверждения", "конец кадра".

Начало кадра (Start of Frame)

Отмечает начало кадра данных или кадра удаленного запроса данных. Состоит из бита с лог. '0'.

Узлу разрешено начинать передачу только при свободной шине (см. простой шины). Все узлы должны быть синхронизированы по началу фронта, вызванного полем "начало кадра" (см. аппаратная синхронизация) узла, начавшего работу первым.

Поле арбитража (Arbitration Field)

Поле арбитража (рисунок 8) состоит из идентификатора и RTR-бита

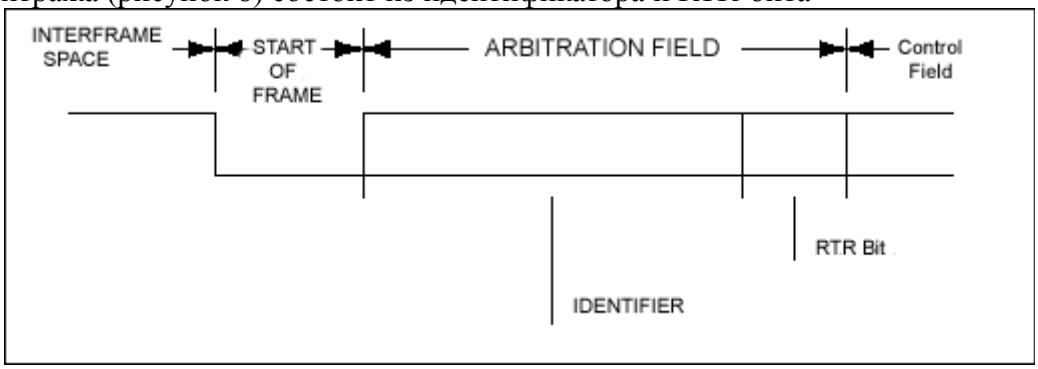


Рисунок 2.8 — Поле арбитража

Идентификатор (Identifier) имеет длину 11 бит. Эти биты должны быть переданы в порядке от ID10 до ID4. Самый старший бит ID0. 7 старших битов не должны быть все битами с лог '1'. Расширенный формат предполагает 29 бит в этом поле (базовый идентификатор и расширение). RTR-бит (Бит запроса передачи) в кадре данных - "0", в кадре удаленного запроса данных - "1". Для совместимости расширенного и стандартного форматов 12-й бит идентификатора расширенного формата является рецессивным.

Поле контроля (CONTROL FIELD)

Поле контроля (рисунок 9) включает 6 бит. Это - код длины данных (4бита) и 2 бита зарезервированные под будущие расширения. Зарезервированные биты должны быть "0".

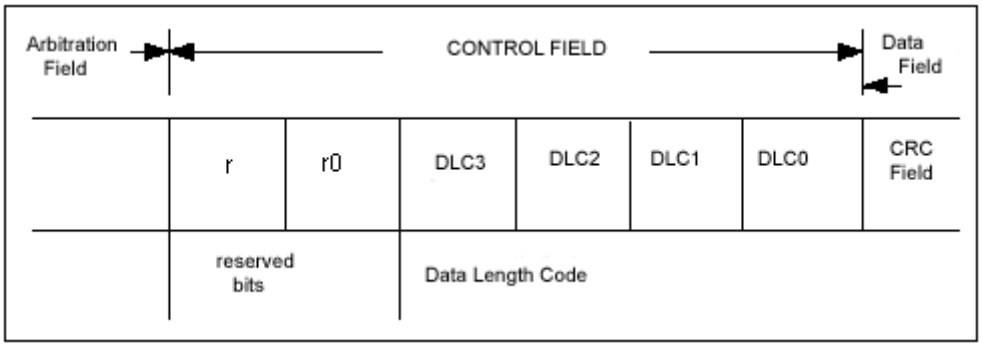


Рисунок 2.9 — Поле контроля

Код длины данных (DLC)

Показывает количество байт в поле данных. Код длины данных имеет размер 4 бита и передаётся внутри контрольного поля. Допустимые значения: 0.....8. Другие значения использоваться не могут.

Поле данных (DATA FIELD)

Включает данные, передаваемые внутри кадра данных. Оно может содержать от 0 до 8 байт, каждый из которых содержит 8 бит.

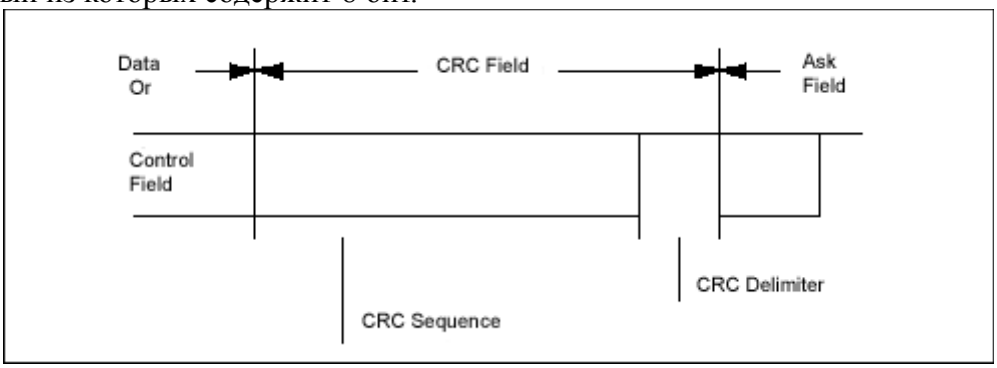


Рисунок 2.10 — Поле CRC

CRC поле (CRC FIELD)

Содержит CRC - последовательность, сопровождаемую разделителем.

CRC-последовательность (CRC Sequence):

Для вычисления CRC полинома, полином, коэффициенты которого задаются потоком, состоящим из значений, бит полей: "начало кадра ", "поле арбитража", "поле контроля", "поле данных" (если имеется) (самые младшие 15 коэффициентов полинома =0), должен быть разделён на полином следующего вида:

x^15+x^14+x^10+x^8+x^7+x^4+x^3+1

Остаток этого полиномиального деления есть CRC-последовательность, передаваемая по шине.

CRC-разделитель (CRC Delimiter):

CRC-последовательность сопровождается CRC-разделителем, который всегда равен лог. "1".

Поле подтверждения (ACK FIELD)

Длина 2 бита. Содержит область подтверждения (1 бит) и разделитель подтверждения (1 бит). В поле подтверждения передающий узел посылает два бита с лог. "1". Приемник, получивший правильное сообщение, информирует об этом передатчик, посылая бит с лог. "0" (т.е. перезаписывая бит в области подтверждения с лог. "1" на бит с лог. "0").

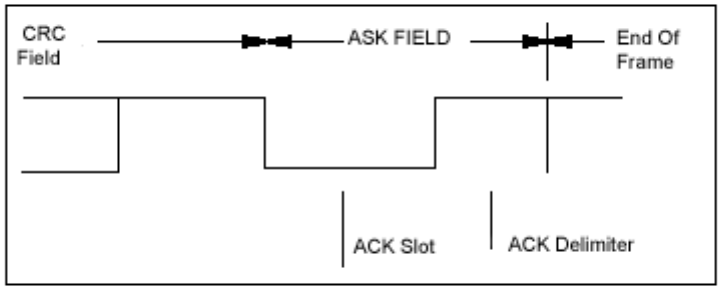


Рисунок 2.11 — Поле подтверждения

Область подтверждения (ACK Slot)

Все узлы, получившие соответствующую CRC-последовательность, сообщают об этом внутри области подтверждения перезаписью бита с лог. "1" на бит с лог. "0".

Разделитель подтверждения (ACK Delimiter)

Второй бит области подтверждения должен быть - лог. "1". Следовательно, область подтверждения окружена битами с лог. "1" (CRC-разделитель и разделитель подтверждения).

Конец кадра (END OF FRAME)

Каждый кадр данных и кадр удаленного запроса данных разграничены последовательностью флагов, состоящей из семи битов с лог. "1".

Структура кадра удаленного запроса данных (REMOTE FRAME)

Узел может инициализировать передачу кадра данных другим узлом, посылая кадр удаленного запроса данных. Этот кадр состоит из 6 полей: "Начало кадра", "поле арбитража", "поле контроля", "поле CRC", "поле подтверждения", "конец кадра".

В отличие от кадра данных , RTR бит = "1". Здесь нет поля данных, зависящего от значения "кода длины данных", внутри этого поля может быть записано любое из допустимых значений (0…….8). Полярность RTR бита показывает, является ли передаваемый кадр кадром данных или кадром удаленного запроса данных

Структура кадра ошибки (ERROR FRAME)

Состоит из двух различных полей. Первое поле является суперпозицией флагов ошибки различных узлов, второе поле - поле разделителя ошибки. Флаги ошибки могут быть двух типов: активный флаг ошибки (ACTIVE ERROR FLAG) и пассивный кадр ошибки (PASSIVE ERROR FLAG). Активный флаг формируется активным узлом (см. ниже) и состоит из шести доминантных битов (нарушающих синхронизацию или структуру полей подтверждения или конца кадра). Пассивный флаг формируется пассивным узлом и состоит из шести рецессивных бит. После передачи флага ошибки устройство посылает рецессивные биты до тех пор, пока не обнаружит их на шине, после чего передает еще семь рецессивных битов.

Структура кадра перегрузки

Кадр перегрузки состоит из двух флаговых битов и разделителя. Кадр передается в двух случаях:

① При необходимости задержки следующего кадра данных или кадра запроса данных по внутренним обстоятельствам;

① При обнаружении доминантного бита во время межкадрового интервала.

В первом случае передача должна начинаться во время передачи первого бита межкадрового интервала, во втором случае - после обнаружения доминантного бита. Флаг состоит из шести доминантных битов, разделитель - из восьми рецессивных.

Межкадровый интервал

Разделяет последовательные кадры данных и запроса данных и состоит из трех рецессивных бит собственно интервала и далее из неограниченного числа рецессивных бит свидетельствующих о незанятой шине. Узлам шины не разрешается начинать передачу во время межкадрового интервала, за исключением передачи кадра перегрузки. Узлы, находящиеся в состоянии пассивной ошибки после передачи собственного кадра данных должны передавать дополнительные восемь рецессивных бит после трех бит общего интервала.

2.3 Протокол LIN

LIN (Local Interconnection Network) разработаны консорциумом европейских автопроизводителей и других известных компаний, включая Audi AG, BMW AG, и другие. Протокол LIN предназначен для создания дешевых локальных сетей для обмена данными на коротких расстояниях.

Основные задачи, возлагаемые на LIN консорциумом европейских автомобильных производителей, — объединение различных автомобильных подсистем и узлов (таких как дверные замки, стеклоочистители, стеклоподъемники, управление магнитолой и климат-контролем, электролюк и т. д.) в единую электронную систему. LIN утвержден Европейским Автомобильным Консорциумом как дешевое дополнение к сверхнадежному протоколу CAN. Как последовательный протокол связи LIN эффективно поддерживает управление электронными узлами в автомобильных системах с шиной класса «А» (двунаправленный полудуплексный), что подразумевает наличие в системе одного главного (Master) и нескольких подчиненных (slave) узлов.

Протокол LIN поддерживает двунаправленную передачу данных по одному проводу длиной до 40 метров, используя микроконтроллер с генератором на RC-цепочке, без использования кварцевого резонатора. Основная идеология — как можно больше задач переложить на программное обеспечение с целью уменьшения стоимости конструкции. Контроллеры автоматически проводят самосинхронизацию при каждой посылке данных.

Основное отличие протокола LIN от шины CAN заключается в том, что концепция LIN — это система связи с очень низкой стоимостью за счет снижения эффективности. Технические требования линейного приемопередатчика удовлетворяют стандартам ISO 9141.

Структура шины представляет собой нечто среднее между I^2C и RS-232. Шина подтягивается вверх к источнику питания через резистор в каждом узле и вниз через открытый коллекторный переход приемопередатчика, как в I^2C. Но вместо стробирующей линии, каждый передаваемый байт обрамляется стартовым и стоповым битами и передается асинхронно, как в RS-232.

Для обмена данными используется один сигнальный провод. Активным состоянием является низкий уровень на шине данных, в это состояние ее может перевести любой узел, послав в шину низкий уровень. В пассивном состоянии на шине напряжение питания Vbat (9...18 В), что означает, что все узлы на шине в неактивном состоянии. Рабочее напряжение питания находится в пределах 9…18 В, но все узлы должны выдерживать перегрузки и сохранять работоспособность при увеличении напряжения на шине вплоть до 40 В. Обычно микроконтроллер в каждом узле подключен к шине через приемопередатчик, который и обеспечивает защиту от перегрузок. Это позволяет использовать обычный микроконтроллер с напряжением питания 5 В, в то время как сама шина работает на больших напряжениях. Для устройства задатчика (master) значение терминального резистора составляет 1 кОм, для устройств исполнителей (slave) — 20…47 кОм.

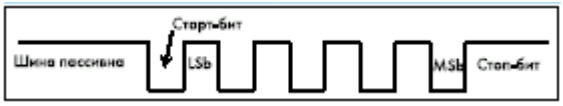


Рисунок 2.12 — Порядок установления сигналов на шине для обмена данными

Все управление шиной осуществляет задатчик (master). Он посылает в шину запрос с адресом интересующего его исполнителя, а затем осуществляет с ним обмен данными. Исполнители (slave) лишь передают или принимают данные по запросу задатчика. Передача сообщения начинается задатчиком с посылки сигнала «Synch Break», которое представляет собой 13 последовательно идущих нулей и сообщает всем исполнителям, что начался цикл обмена; затем идет поле синхронизации (Synch Field) и поле идентификации (Ident Field).

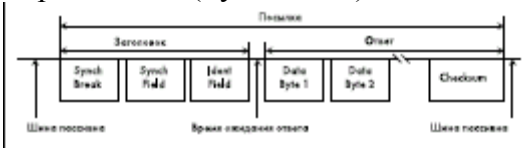


Рисунок 2.13 — Структура пакета данных

Поле синхронизации передается задатчиком в начале каждого сообщения, и все исполнители должны принять это сообщение и подстроить частоту своего собственного приемопередатчика.

Второй байт каждой посылки — поле идентификации (адреса), в котором сообщается, с каким исполнителем начинается обмен данными в этой посылке, и сколько байт будет содержаться в ответе исполнителя. Только этот исполнитель имеет право передать данные задатчику. Но как только этот ответ появляется на шине, любой другой исполнитель также может принять эти данные. Таким образом, для того чтобы передать данные от одного исполнителя другому, совершенно необязательно пересылать их непосредственно через задатчика. Протокол LIN подразумевает использование RC-цепочки в качестве задающего

генератора микроконтроллеров исполнителей. Поэтому каждое сообщение содержит поле синхронизации и каждый исполнитель обязан подстроить по этому полю частоту своего приемопередатчика. Для того, чтобы определить время передачи одного бита, необходимо засечь время четырех периодов стартовой посылки, разделить на 8 и округлить.

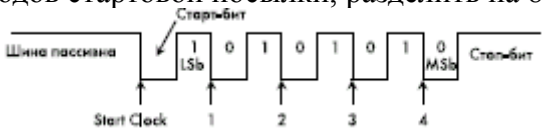
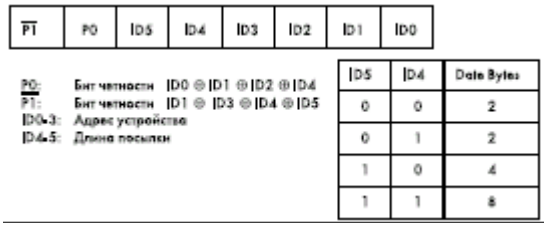


Рисунок 2.14 — Синхронизация

В идентификационном поле сообщается информация о том, что же, собственно, последует дальше. Поле идентификации разделено на три части: четыре бита (0–3) содержат адрес исполнителя, с которым будет производиться обмен информации, два бита (4–5) указывают количество передаваемых байт и последние два бита (6–7) используются для контроля четности. Четыре бита адреса могут выбирать одного из 16-ти исполнителей, каждый исполнитель может отвечать 2-мя, 4-мя, или 8-ю байтами, таким образом получаем 64 типа различных сообщений на шине.



$\overline{P1}$	P0	ID5	ID4	ID3	ID2	ID1	ID0

P0: Бит четности ID0 ⊕ ID1 ⊕ ID2 ⊕ ID4
P1: Бит четности ID1 ⊕ ID3 ⊕ ID4 ⊕ ID5
ID0-3: Адрес устройства
ID4-5: Длина посылки

ID5	ID4	Data Bytes
0	0	2
0	1	2
1	0	4
1	1	8

Рисунок 2.15 — Структура идентификационного поля

Спецификация LIN не устанавливает каких-либо жестких рамок на передаваемую информацию (за исключением команды «Sleep»), оставляя свободу творчества для программистов. Задатчик может послать команду всем исполнителям перейти в микромощный режим («Sleep»), выставив в поле идентификации байт 0x80. Исполнители, приняв его, освобождают шину и переходят в «спящий» режим с выходом из него по изменению состояния на шине. Любой исполнитель может активизировать шину, передав байт 0x80. После этого все узлы ожидают дальнейших опросов задатчика в обычном режиме.

Программная реализация

Протокол LIN можно организовать программно на любом микроконтроллере. Очень удобно для этих целей применять малогабаритные и дешевые PIC12C508 и PIC16C505. Практически все реализуется программно, необходимо лишь соединить микроконтроллер с шиной через приемопередатчик на дискретных элементах.

2.4 PROFIBUS

2.4.1 Особенности:

Топология сети м.б. выполнена в виде общей шины с терминатором или без такового;

В качестве среды распространения используется витая пара, позволяющая организовать сеть протяженностью до 1.2 км и обеспечить скорость передачи до 1.5 Мбит/с;

PROFIBUS предполагает асинхронную полудуплексную передачу данных без вставки дополнительных битов;

Целостность данных контролируется с помощью кодов Хемминга;

Диапазон адресов - до 127;

Все станции сети делятся на два класса: мастера и пассивные станции. Рекомендуется ограничить число мастеров до 32;

Минимальная конфигурация сети включает две станции. Передача данных между мастерами обеспечивается с помощью передачи маркеров.

В соответствии с моделью ISO/OSI спецификация PROFIBUS описывает физический уровень, уровень передачи данных и уровень приложений. На уровне передачи данных действуют специальные службы, осуществляющие следующие операции:

Передачу данных одному узлу с подтверждением (SDA);

Передачу данных одному, нескольким или всем узлам без подтверждения (SDN);

Одиночную посылку данных узлу с одновременным запросом приема данных (SRD);

Циклическую посылку данных узлу с одновременным запросом приема данных (CSRD);

Спецификация интерфейса PROFIBUS определяет тип и электрические параметры разъемов, кабеля, способы подключения и заземления.

PROFIBUS реализует гибридный метод доступа к сети: децентрализованный метод, основанный на передаче маркера между узлами сочетается с централизованной моделью типа MASTER-SLAVE. Передача может быть инициирована только мастером, имеющим в текущий момент право доступа к линии, т.е. обладающим маркером. Маркер передается от мастера к мастеру в логическом кольце, определяя, таким образом, право доступа к линии. Каждая станция-мастер знает своего предшественника, от которого она получает маркер и наследника, которому она передает маркер. Эта информация формируется при активации сети и далее обновляется по специальному алгоритму. При такой организации необходимо обрабатывать ряд ошибок, исключений и событий, как, например, потеря маркера, дублированные адреса, множественные маркеры, отключение и подключение станций.

2.4.2 Принципы операций с маркером

Маркер передается от одной станции к другой в порядке возрастания адреса, причем соблюдается принцип кольца, т.е. станция с максимальным адресом передает маркер станции с минимальным адресом.

Станция принимает маркер только от предшественника, если имеет место попытка передачи маркера от незнакомой станции, адресат должен отказаться от приема и сгенерировать состояние ошибки. При повторной попытке маркер должен быть принят и информация о конфигурации сети изменена.

Если после передачи маркера предшественник обнаруживает на шине активность в течение определенного времени, он констатирует передачу маркера (успешную или нет) и переходит в пассивное состояние. В противном случае он дважды повторяет попытку передачи маркера наследнику, и в случае неудачи, пробует передать маркер следующей станции в кольце. Эта операция повторяется, пока маркер не будет передан или не произойдет захват линии другим мастером. Если никто не принимает маркер, станция констатирует одиночество и хранит маркер у себя или пересылает его себе, если не требуется циклов передачи данных. Если при следующем сеансе регистрации будет обнаружен наследник, станция вновь пытается передать маркер.

Станции могут отсоединяться и присоединяться в любой момент, при этом каждая станция отвечает за присоединение и отсоединение соседних станций, лежащих в своей области адресов, для чего периодически сканирует это адресное пространство.

При первичной инициализации или потере маркера производится полная или частичная реконфигурация сети, при которой маркер захватывается станцией с наименьшим адресом.

2.5 Современные высокоскоростные интерфейсы

Проблема пропускной способности цифровых интерфейсов является «узким местом» в реализации высокоскоростных цифровых устройств. Физическая реализация линий

передачи цифровых данных ограничивает скорость по соображениям помехоустойчивости и взаимного влияния соседних проводников. Кроме этого с ростом частоты приходится принимать во внимание и вопросы согласования сопротивлений линии.

В настоящее время для высокоскоростной передачи используются дифференциальные линии. Такой тип носителя данных характерен для последовательных интерфейсов типа RS-422/485, а также для интерфейса LVDS (Low-Voltage Differential Signaling).

LVDS обладает следующими особенностями, обуславливающими его широкое распространение:

- Совместимость с низковольтными стандартами;
- Низкий уровень генерируемых помех;
- Высокая помехоустойчивость;
- Надежная и устойчивая передача сигналов;
- Возможность интеграции в ИС.

В настоящее время LVDS стандартизован для передачи данных со скоростями от нескольких сотен Мбит/с до 2 Гбит/с.

LVDS был предложен фирмой National Semiconductor в 1994 г.

В качестве носителя данных LVDS использует дифференциальную двухпроводную линию с разностью потенциалов в несколько сотен милливольт.

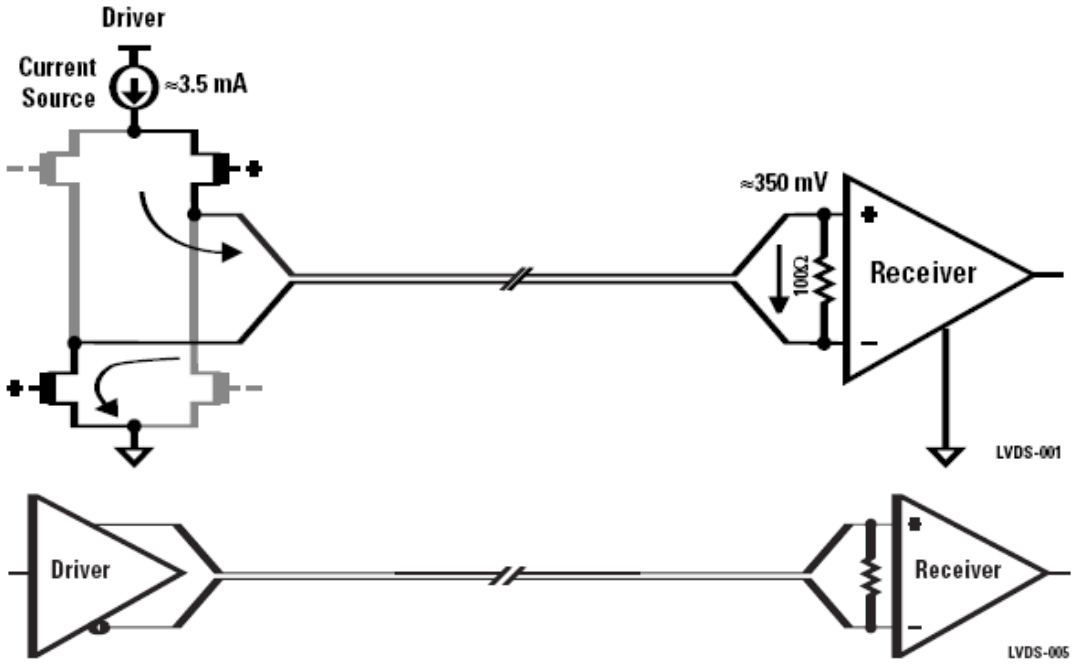


Рисунок 2.16 — Особенности схемотехники интерфейса LVDS

Ток создаваемый мостовой схемой, питающейся от источника тока течет по двухпроводной линии, заканчивающейся сопротивлением, через которое протекает основная часть тока (из-за высокого входного сопротивления приемника). При переключении транзисторов моста направление тока меняется на противоположное. Как правило, используются линии с волновым сопротивлением 100 Ом и соответствующие согласующие сопротивления (терминаторы), которые реализуются как внешние компоненты или встроенные в приемник.

Действующий стандарт ANSI/TIA/EIA-644-A (1995 г.) рекомендует ограничить скорость передачи данных до 655 Мбит/с и устанавливает теоретический предел в 1.923 Гбит/с при использовании идеального носителя (без потерь). В интересах расширения сферы внедрения LVDS стандарт не регламентирует требований к носителю данных, технологии изготовления приемников и передатчиков, а также напряжению питания.

Применение LVDS в настоящее время сводится к использованию нескольких типов устройств, поддерживающих описанный стандарт:

①Передатчики/приемники LVDS;
②Сериализаторы/десериализаторы;
③Коммутаторы;
④Устройства шин LVDS.

Сериализаторы/десериализаторы, как правило, используются для мультиплексирования нескольких низкоскоростных линий в одну высокоскоростную (LVDS), тем самым сокращая число выводов устройства.

Шинные устройства обеспечивают все преимущества LVDS при реализации шин передачи данных, гарантируя высокие скорости, малое энергопотребление и работу на несколько потребителей.

Конфигурации линий передачи данных по стандарту LVDS предусматривают соединение «точка-точка» с двунаправленной полудуплексной передачей, а также структуру с несколькими приемниками на одной линии или многоточечную линию. В этом случае терминатор устанавливается на самом удаленном конце линии, а ответвления выполняются максимально короткими (менее 2.5 см).

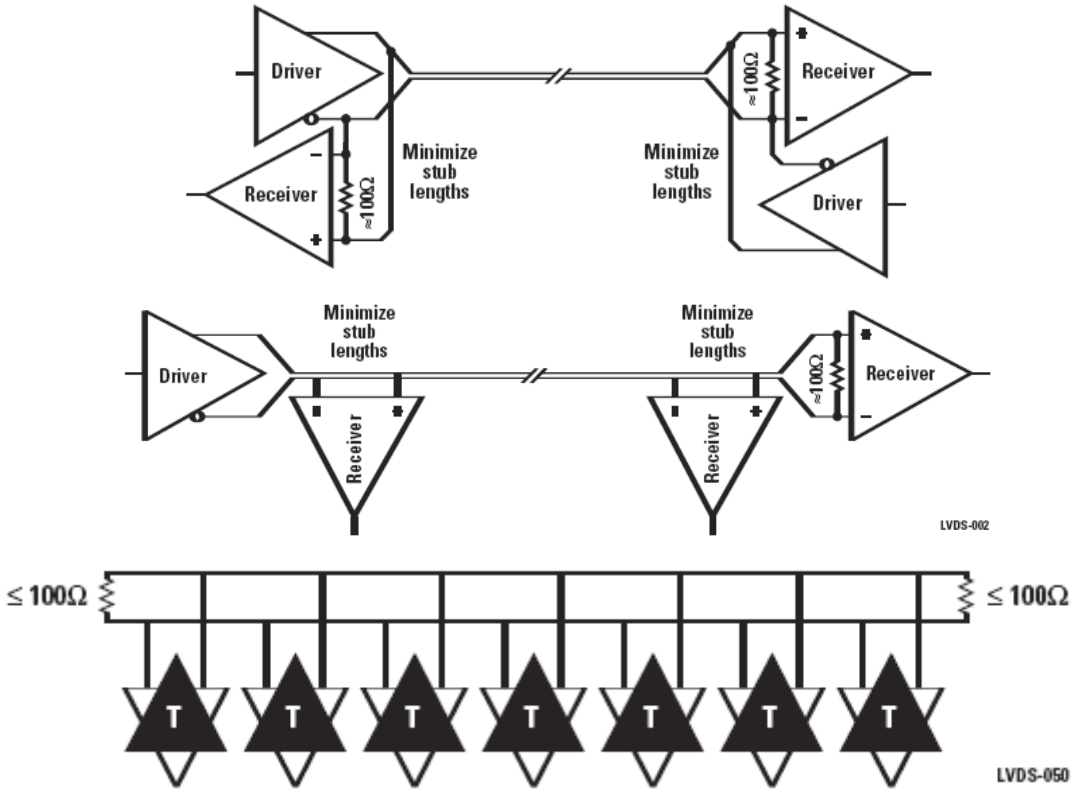


Рисунок 2.17 — Особенности подключения нескольких передатчиков и приемников к шине LVDS

Следует отметить, что в настоящее время драйверами LVDS оснащаются большинство ПЛИС, таким образом, избавляя разработчика от необходимости применения дополнительных специализированных микросхем. Однако, следует отметить, что реализация LVDS устройств на базе ПЛИС, в среднем не позволяет достичь максимальной скорости передачи из-за увеличенной емкости по сравнению со специализированными ИС.

Технологические требования к реализации LVDS линий передачи

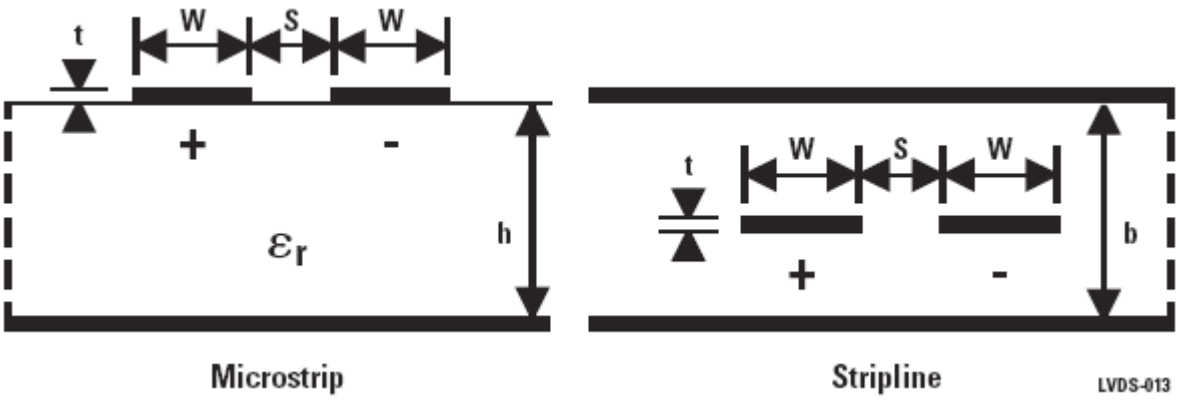


Рисунок 2.18 — Параметры проводников на печатной плате

Для достижения требуемого сопротивления линии рекомендуется проектировать топологию трасс с учетом соотношений:

Microstrip

$$Z_{DIFF} \approx 2 \times Z_0 \left(1 - 0.48 e^{-0.96\frac{S}{h}}\right) \Omega$$

Stripline

$$Z_{DIFF} \approx 2 \times Z_0 \left(1 - 0.374 e^{-0.29\frac{S}{h}}\right) \Omega$$

Microstrip

$$Z_0 = \frac{60}{\sqrt{0.457\varepsilon_r + 0.67}} \ell n\left(\frac{4h}{0.67(0.8W + t)}\right) \Omega$$

Stripline

$$Z_0 = \frac{60}{\sqrt{\varepsilon_r}} \ell n\left(\frac{4b}{0.67\pi(0.8W + t)}\right) \Omega$$

Здесь Z_0 - волновое сопротивление одиночной трассы, Z_{DIFF} - дифференциальное сопротивление парной трассы. Также рекомендуется соблюдать следующие правила:

① Расстояние между соседними LVDS трассами должно превышать двойное расстояние между проводниками;

② Избегать изгибов на 90 (использовать 45 диагонали);

③ Размещать LVDS трассы подальше от одиночных скоростных трасс с тем, чтобы уменьшить уровень наводок;

④ Для высокоскоростных разработок рекомендуется использовать высококачественный материал (GETEC) вместо FR-4, который подходит для частот меньше 0.8ГГц;

⑤ Выполнять трассы поближе друг к другу, стремясь обеспечить максимальную симметрию, при этом избегая переходов и др. неоднородностей.

3 Элементы схем цифровой и вычислительной техники в системах обработки сигналов

Высокопроизводительные системы обработки сигналов строят на основе цифровых сигнальных процессоров (ЦСП).

ЦСП – цифровой сигнальный процессор (DSP – Digital Signal Processor). ЦСП предназначены для выполнения специфических задач цифровой обработки сигналов (ЦОС). В основном ЦОС сводится к дискретизации и квантованию аналогового сигнала, выполнению определенных алгоритмов обработки полученных цифровых данных и обратному преобразованию в аналоговую форму, пригодную для восприятия человеком. ЦОС имеет ряд преимуществ перед чисто аналоговой техникой преобразования сигналов и позволяет достичь результатов, невозможных при традиционной аналоговой обработке. Алгоритмы ЦОС имеют ряд общих особенностей, учитывая которые, возможно создать специализированные процессоры, которые обеспечивают более
высокую эффективность по сравнению с универсальными при решении задач ЦОС.

Основные отличия ЦСП от универсальных процессоров с архитектурой ОШ:

−архитектура;

−система макрокоманд.

В основу ЦСП положены следующие принципы:

1. Использование Гарвардской архитектуры;
2. Сокращение длительности командного цикла;
3. Применение внутренней конвейеризации;
4. Применение аппаратного умножителя;
5. Включение в систему команд специальных команд для цифровой обработки сигналов (ЦОС) и использование специальных методов адресации.

ЦСП применяется в следующих областях:

−цифровая фильтрация;

−кодирование и декодирование информации;

−распознавание звука и речи;

−обработка изображений;

−спектральный анализ;

−цифровая звукотехника;

−медицина (томографы);

−измерительная техника;

−системы управления;

Классическая гарвардская архитектура предполагает хранение данных и команд в разных ЗУ. В ЦСП применяется супергарвардская архитектура, имеющая несколько шин адреса и данных для доступа к памяти данных и памяти команд с возможностью одновременного доступа к нескольким областям памяти. Обычно выделяются изолированные памяти для хранения первого и второго операнда и память для хранения программ. Преимущества такой организации в следующем: несколько разделенных шин данных и адреса позволяют совмещать по времени выборку данных и использование команд.

Короткий командный цикл необходим для работы в реальном масштабе времени и высокой производительности процессора. Следует отметить, что реальный масштаб времени предполагает, что данные на выходе должны появляться в том же темпе, что и поступающие на вход (с частотой дискретизации), откуда следует, что алгоритм обработки отдельной выборки должен завершаться за время, меньшее интервала дискретизации. С развитием технологии производства СБИС и уменьшением проектных норм (более тонкие

соединительные линии между активными элементами и меньшая площадь, на которой реализован транзистор) снижается длительность командного цикла процессоров и повышается тактовая частота, однако этот процесс, как считают в настоящее время многие эксперты, достиг почти максимума своих возможностей. Дальнейшее повышение тактовой частоты и уменьшение проектных норм сталкивается с серьезными физическими ограничениями и вряд ли приведет к революционным качественным изменениям по сравнению с достигнутыми результатами. Повышение производительности в настоящее время связывают с применением многоядерных структур, алгоритмов распараллеливания обработки, конвейеризацией и т.п.

Конвейерный режим используется для сокращения времени командного цикла. Обычно в ЦСП используется двух- или трехкаскадный конвейер. Он позволяет одновременно выполнять 2– 3 команды на разных стадиях выполнения.

Стадиями обработки в трехкаскадном конвейере являются: выборка, декодирование и исполнение. Очевидно, что наибольшая эффективность будет достигаться при линейном выполнении программы с минимальным количеством ветвлений. Наиболее просто реализуется конвейер прификсированной длине команды.

Таблица 3.1 - Стадии выполнения команд трехкаскадного конвейера

Цикл	Состояние внутреннего конвейера		
	выборка	декодирование	исполнение
инициализация			
предварительный 1	команда 1		
предварительный 2	команда 2	команда 1	
1	команда 3	команда 2	команда 1
2	команда 4	команда 3	команда 2
3	команда 5	команда 4	команда 3
…	…	…	…

Различают внутреннюю и внешнюю конвейеризации. При внутренней обработка команд осуществляется внутри ЦСП. Конвейеризация на внутреннем уровне позволяет реализовать только внутренний командный цикл. Внешняя конвейеризация, по сути, есть способ распараллеливания вычислений между несколькими отдельными процессорами. Такая конвейеризация применяется чаще всего в систолических алгоритмах и вычислителях. Систолические вычислители предоставляют возможность повышения производительности не за счет внутренних архитектурных особенностей процессоров, а за счет использования принципов параллельной и конвейерной обработки данных на физическом уровне. Они реализуются с помощью процессорной платы со многими процессорами. Примерами реализации систолических архитектур являются матричные операции, фильтрация, преобразования Фурье и др. Соответствующие систолические ВА позволяют заменить программы, подпрограммы или функции аппаратной реализацией алгоритма и позволяют сократить время выполнения программы на 1 – 3 порядка.

Аппаратный умножитель применяется в ЦСП для сокращения длительности времени исполнения одной из основных операций ЦОС – операции умножения. Например, при цифровой фильтрации необходимо перемножение матрицы коэффициентов на вектор с компонентами сигнала для разных моментов времени. В алгоритмах быстрого преобразования Фурье применяется операция умножения с накоплением (Y=Y+A*B). В ЦСП операция умножения выполняется за один командный цикл благодаря специальному аппаратному умножителю.

Специальные команды для ЦОС вводятся в систему команд ЦСП для ее оптимизации и для выполнения базовых задач и алгоритмов ЦОС в масштабе реального времени. Как правило, специальные команды выполняют за один цикл операцию, для выполнения которой

универсальному процессору понадобилось бы несколько циклов. В ряде случаев ЦСП используют комбинированные команды, представляющие собой комбинацию нескольких элементарных операций, выполняемых параллельно. Ряд ЦСП имеют в своем составе специальные периферийные модули, для управления которыми в системе команд предусматриваются специальные инструкции.

Историческая справка

В 1982 году фирма Texas Instruments выпустила ЦСП TMS 32010. Т.к. он был первым, то он стал своеобразным промышленным стандартом в области ЦСП, т.е. другие фирмы в качестве стандарта стали использовать некоторые его архитектурные решения. Он был 16-разрядным, выполнял 5 млн. операций типа "×" и "+" в секунду. В составе TMS 32010 имелось встроенное ОЗУ, емкостью 144 или 256 слов, встроенное ПЗУ, емкостью 2048 слов, арифметико-логического устройства (АЛУ) с аккумулятором разрядностью 32 бита, встроенный умножитель 16×16 бит (операнды) с 32-разрядным результатом.

3.1 Архитектура ЦСП

С учетом вышесказанного типичный ЦСП имеет гарвардскую (супергарвардскую) архитектуру с несколькими шинами адреса и данных, высокопроизводительные АЛУ, аппаратный умножитель и сдвигатель, набор периферийных модулей.

Рисунок 3.1 — Архитектура ЦСП

Для эффективного управления шинами в состав ЦСП включаются специальные устройства генерации адреса (УГА), функцией которых является манипуляция адресами и поддержка специальных режимов адресации. К специальным режимам адресации относятся циклическая и бит-реверсивная адресация. Циклическая адресация обеспечивает автоматическую поддержку замкнутых циркулярных буферов с произвольным шагом (инкрементом или декрементом).

Циклическая адресация в ЦСП реализуется автоматически средствами УГА без затраты дополнительных циклов на вычисление необходимого адреса.

Бит-реверсивная адресация применяется при реализации алгоритмов быстрого преобразования Фурье (БПФ), при котором операнды выбираются в нестандартном порядке.

Рисунок 3.2 — Циклическая адресация в ЦСП

Прямая реализация бит-реверсии программным образом требует довольно много манипуляций, поэтому этот вид адресации реализуется либо специальными командами либо аппаратным способом с помощью УГА, что значительно более выгодно с точки зрения производительности процессора.

Таблица 3.2 — Прямая и бит-реверсивная адресации

Порядок следования *n* исходных отсчетов *x* (*n*)			
Прямой		Бит-реверсивный	
ятичная система	Двоичная система	Двоичная система	Десятичная система
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

В настоящее время ЦСП выпускаются многими фирмами, наиболее известными следует признать Texas Instruments, Analog Devices, Freescale. Все многообразие выпускаемых ЦСП можно классифицировать следующим образом:

− ЦСП со стандартной архитектурой
− ЦСП с улучшенной стандартной архитектурой
− VLIW ЦСП

− Суперскалярные ЦСП
− Гибридные ЦСП

ЦСП стандартной и улучшенной стандартной архитектуры в основном соответствуют вышеописанным принципам и отличаются разрядностью, шинной архитектурой, наличием и объемом памяти и т.п. Повышение производительности в процессорах улучшенной стандартной архитектуры достигается, в основном увеличением количества одновременно выполняемых операций. Суперскалярные и VLIW процессоры были рассмотрены ранее, гибридные ЦСП представляют собой скорее микроконтроллеры, чем процессоры, предназначены для выполнения относительно узких функций (управление электроприводом, телекоммуникационные задачи), однако имеют все черты ЦСП.

Принято также делить ЦСП на процессоры с фиксированной и плавающей точкой. Процессоры с фиксированной точкой оперируют целочисленными значениями или дробными с фиксированным форматом (1.15 или 1.31), ЦСП с плавающей точкой поддерживают несколько международных стандартных форматов (наиболее популярен стандарт IEEE754), различающихся по точности, но имеющих одинаковую структуру следующего вида:

Рисунок 3.3 — Формат представления числа с плавающей точкой

Здесь порядок n-разрядного нормализованного числа задается в так называемой смещенной форме: если для задания порядка выделено k разрядов, то к истинному значению порядка, представленного в дополнительном коде, прибавляют смещение, равное (2k-1 — 1). Например, порядок, принимающий значения в диапазоне от —128 до +127, представляется смещенным порядком, значения которого меняются от 0 до 255.

Использование смещенной формы позволяет производить операции над порядками, как над беззнаковыми числами, что упрощает операции сравнения, сложения и вычитания порядков, а также упрощает операцию сравнения самих нормализованных чисел.

Чем больше разрядов отводится под запись мантиссы, тем выше точность представления числа.

Чем больше разрядов занимает порядок, тем шире диапазон от наименьшего отличного от нуля числа до наибольшего числа, представимого в машине при заданном формате.

Стандартные форматы представления вещественных чисел:

1) одинарный — 32-разрядное нормализованное число со знаком, 8-разрядным смещенным порядком и 24-разрядной мантиссой (старший бит мантиссы, всегда равный 1, не хранится в памяти, и размер поля, выделенного для хранения мантиссы, составляет только 23 разряда).

2) двойной — 64-разрядное нормализованное число со знаком, 11-разрядным смещенным порядком и 53-разрядной мантиссой (старший бит мантиссы не хранится, размер поля, выделенного для хранения мантиссы, составляет 52 разряда).

3) расширенный — 80-разрядное число со знаком, 15-разрядным смещенным порядком и 64-разрядной мантиссой. Позволяет хранить ненормализованные числа.

Следует отметить, что вещественный формат с m-разрядной мантиссой позволяет абсолютно точно представлять m-разрядные целые числа, т. е. любое двоичное целое число, содержащее не более m разрядов, может быть без искажений преобразовано в вещественный формат.

ЦСП с плавающей точкой обычно сложнее и дороже процессоров с фиксированной точ-

кой, но обладают рядом преимуществ:
−Высокая точность;
−Практически неограниченный диапазон значений;
−Улучшенная совместимость с языком Си;
−Отсутствие необходимости в масштабировании.
Под масштабированием здесь понимается часто используемый в целочисленной арифметике прием, когда один из операндов масштабируют, т.е. специально умножают на масштабный множитель, чтобы добиться необходимой точности при операции деления.

4 Запоминающие устройства микропроцессорных систем

Рассмотрим пример построения системы памяти микропроцессорной системы обработки данных на основе процессоров Blackfin.

Память разделена на два уровня: L1 и L2. L1 - это внутренняя память процессора, состоящая из нескольких блоков статического ОЗУ (SRAM), загрузочного ПЗУ и набора регистров, отображаемых в пространство памяти. Память L1 работает с полной скоростью ядра. Память L2 - это внешняя память, состоящая из четырех банков асинхронной памяти объемом до 1 Мб и одного банка SDRAM объемом от 16 до 128 Мб.

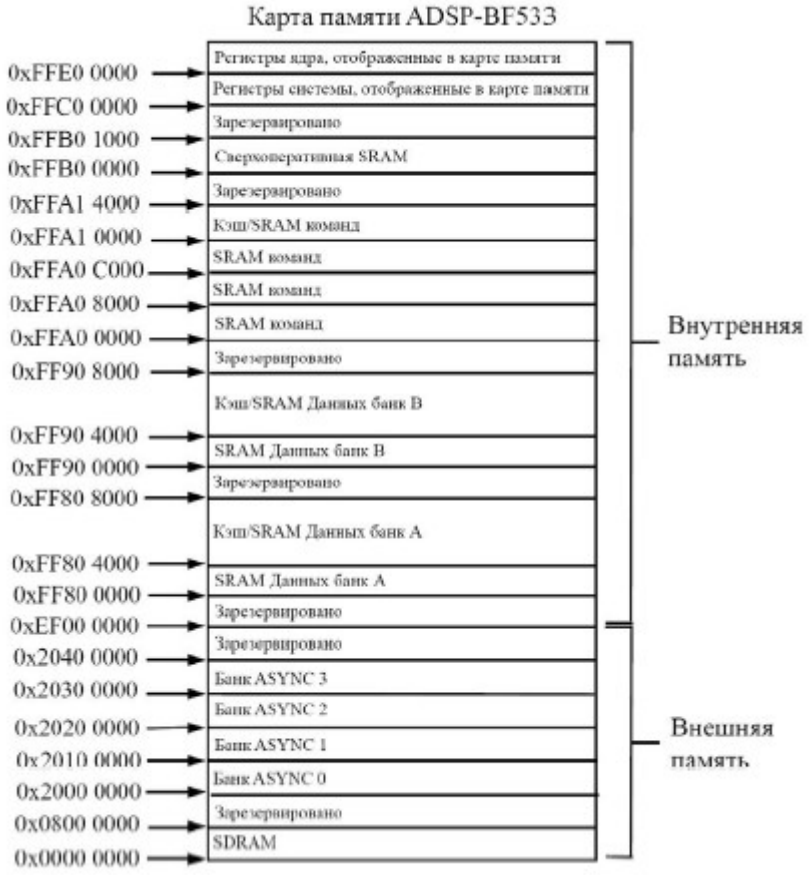


Рисунок 4.1 —Карта памяти BF-533

4.1 Внутренняя память

Внутренняя память состоит из памяти команд, памяти данных и блока сверхоперативной памяти данных. Память команд состоит из двух блоков: один блок может конфигурироваться как память команд или кэш, второй блок - только память команд. Управление памятью команд осуществляется с помощью регистра IMEM_CONTROL. Обращение к памяти команд L1 производится по 64-битной шине, соответственно все команды выравниваются по границе 64 бит. Необходимо отметить, что прямое обращение к памяти команд через УГА запрещается и вызывает исключение. Если часть памяти команд сконфигурирована как кэш, прямой доступ к ней становится невозможным. Кэш поддерживает стратегию LRU и имеет достаточно гибкий механизм управления и блокировки.

Память данных также состоит из нескольких блоков, некоторые из которых могут

конфигурироваться как кэш данных. Управление памятью данных производится с помощью регистра DMEM_CONTROL. С точки зрения программиста, имеется два отдельных банка памяти данных: A и B, каждый из которых может быть кэшем (частично), а также отдельное сверхоперативное ОЗУ. При использовании кэша возможна как сквозная запись, так и отложенная.

Устройство управления памятью (MMU) позволяет реализовывать страничный алгоритм обращения с памятью и защищенную память.

Данные располагаются в памяти по принципу «младший байт по младшему адресу». Процессоры Blackfin поддерживают RISC концепцию загрузки сохранения данных, при которой арифметические операции отделены от операций, требующих доступа к памяти. Такой подход позволяет программисту (или компилятору) оптимизировать программу, размещая между командами, требующими длительных обращений к памяти, операции ядра, не задействующие операции загрузки или сохранения данных. В частности, при записи в память команда считается завершенной за один такт, хотя до завершения физической операции может пройти несколько тактов. Если при чтении окажется, что предыдущая команда загрузки не завершена, конвейер будет остановлен на соответствующее число тактов. При использовании данного механизма могут возникать различные побочные эффекты, в целях гарантии отсутствия непредсказуемого поведения в критичных ситуациях необходимо применять специальные команды CSYNC и SSYNC. Команда CSYNC приводит к задержке и гарантирует завершение всех ожидающих событий ядра, включая обработку прерываний и исключений, и очистку буфера ядра. Команда SSYNC гарантирует завершение всех операций в интерфейсе между памятью L1 и остальной частью системы. В основном применение этих операций необходимо при настройке устройств ввода-вывода и периферийных устройств, возвращающих данные по алгоритму FIFO.

Данные в памяти данных выравниваются, операции с невыравненной памятью явно не поддерживаются, однако иногда они являются необходимыми, в этом случае используется команда DISALGNEXPCT, блокирующая исключения выравнивания.

Регистры, отображенные в память, являются 32-разрядными и требуют 32-разрядного доступа, при этом доступ к ним возможен только в режиме супервизора.

4.2 Интерфейс внешней памяти

Интерфейс внешней памяти обслуживается устройством интерфейса внешней шины (EBIU), которое получает запросы от ядра или контроллера DMA. Неверные обращения вызывают аппаратное исключение доступа, которое может обрабатываться ядром. Для управления внешней SDRAM и асинхронной памятью в процессоре имеются отдельные контроллеры, которые совместно осуществляют арбитраж внешней шины.

При работе с медленными устройствами можно запрограммировать дополнительные такты ожидания, гарантирующие корректные операции записи/чтения. При работе с асинхронной памятью формируются необходимые сигналы, поддерживающие наиболее распространенные стандарты операций записи/чтения включая поддержку байтового и словарного обмена данными.

4.3 Генераторы адреса данных

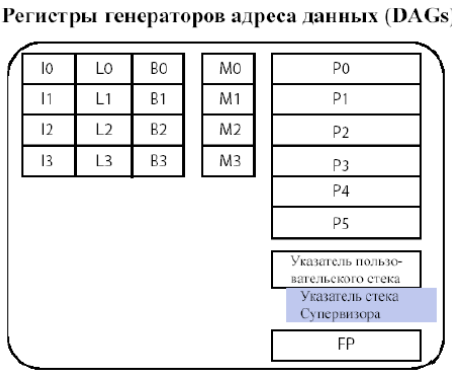


Рисунок 4.2 — Подсистема DAG

Подсистема DAG включает два арифметических устройства DAG, девять регистров указателей, четыре индексных регистра и четыре полных набора соответствующих регистров модификации, базового адреса и длины. Эти регистры содержат значения, используемые генераторами адреса данных для формирования адресов.

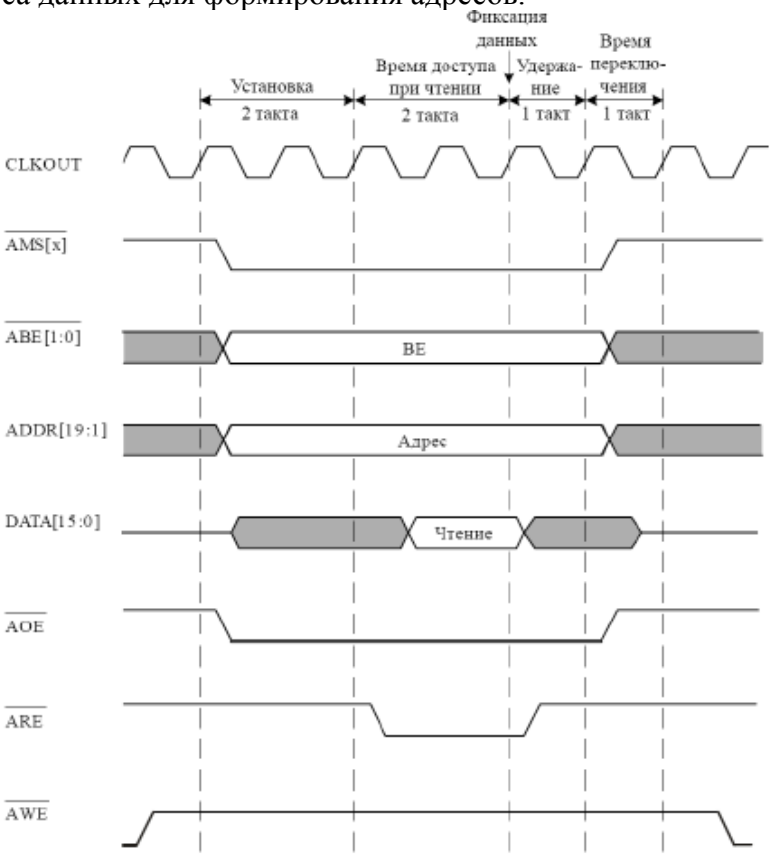


Рисунок 4.2 — Цикл чтения асинхронной памяти

Индексные регистры, I[3:0]. Беззнаковые 32-разрядные индексные регистры являются указателями на адрес в памяти. Например, при выполнении команды R3=[I0] в регистр R3 загружается значение, расположенное в ячейке памяти, на которую указывает регистр I0. Индексные регистры могут использоваться при 16- и 32-разрядных обращениях к памяти.

Регистры модификации, M[3:0]. Знаковые 32-разрядные регистры модификации обеспечивают инкремент или шаг, с которым индексный регистр постмодифицируется при пересылке регистра. Например, команда R0=[I0 ++ M1] приводит к тому, что DAG: - выдает

адрес в регистре I0; - загружает в R0 содержимое ячейки памяти, на которую указывает регистр I0; - изменяет содержимое регистра I0 на величину, содержащуюся в регистре M0.

Регистры длины и базового адреса, B[3:0] и L[3:0]. Беззнаковые 32-разрядные регистры длины и базового адреса определяют начальный адрес и диапазон адресов циклического буфера. Каждая пара регистров B и L всегда используется совместно с соответствующим I-регистром. Например, I3, B3, L3.

Регистры указателей, P[5:0], FP, USP и SP. 32-разрядные регистры указателей являются указателями на адреса в памяти. Различные команды могут использовать поле P[5:0], регистры FP (указатель кадра) и SP/USP (указатель стека/указатель пользовательского стека) и манипулировать ими. Например, при выполнении команды R3=[P0] в регистр R3 загружается значение, содержащееся в ячейке памяти, на которую указывает регистр P0. Регистры указателей не влияют на адресацию циклических буферов. Они могут использоваться при 8-, 16- и 32-разрядных обращениях к памяти. В целях дополнительной защиты режима регистр SP доступен только в режиме Супервизора, а регистр USP доступен в Пользовательском режиме.

4.4 Команды перемещения данных

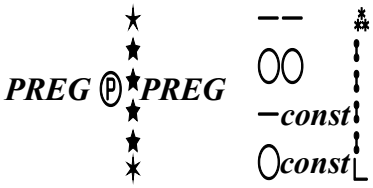
Команды перемещения данных включают в себя команды загрузки данных из памяти, сохранения данных в памяти с использованием всех режимов адресации, а также команды перемещения данных между различными регистрами.

Непосредственная загрузка регистров

В качестве регистров, которые можно загружать непосредственно, используются регистры данных, все регистры указателей и оба аккумулятора. Загружаемые данные могут быть знаковыми или беззнаковыми 8- или 16-разрядными константами. При загрузке в приемник большего размера возможно расширение знаком (X) или нулями (Z). Примеры: R0=0x300; R1.L=0x20 (X); SP=0xAAAA (Z); Для аккумуляторов возможна непосредственная загрузка только нулями A0=A1=0;

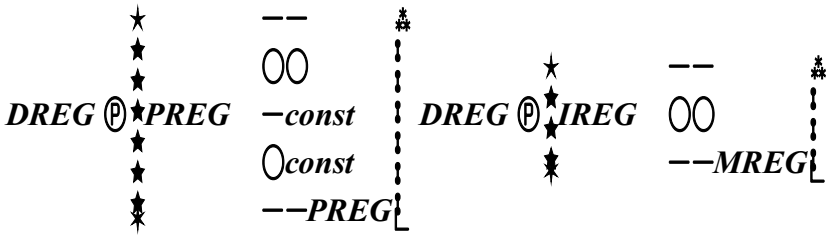
Загрузка указателей из памяти

Регистры P5-0 можно загружать из памяти с использованием косвенной адресации в следующем формате



Загрузка регистров данных из памяти

32-разрядные регистры данных (R0-7) могут быть загружены из памяти сл. инструкциями:



При загрузке байтами или словами используется префикс W или B и указатель расширения нулями (Z) или знаковыми битами (X).

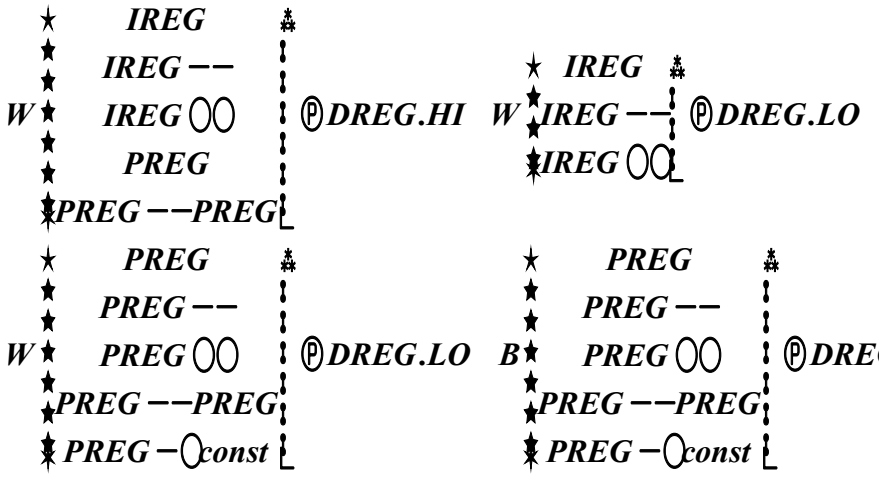
Загрузка половин регистров

Старшая или младшая половины 32-разрядных регистров могут быть загружены в

формате:

Сохранение содержимого регистров может выполняться аналогично:

Старшие и младшие половины регистров данных, а также байты сохраняются в формате:



Перемещения данных между регистрами

Возможно много вариантов таких перемещений, фактически данные из любого регистра могут перемещаться в любой другой, однако имеются некоторые особенности:

- ◯ Обмен данными с системными регистрами возможен только через регистры данных или регистры P5-0;
- ◯ Пересылка из 40-разрядного аккумулятора A0 возможна в регистры данных с четными номерами, а из A1 - с нечетными. При этом возможно дополнительное преобразование форматов, а также параллельные пересылки;

genreg = *genreg* ; /* (a) */
genreg = *dagreg* ; /* (a) */
dagreg = *genreg* ; /* (a) */
dagreg = *dagreg* ; /* (a) */
genreg = USP ; /* (a)*/
USP = *genreg* ; /* (a)*/
Dreg = *sysreg* ; /* sysreg to 32-bit D-register (a) */
sysreg = *Dreg* ; /* 32-bit D-register to sysreg (a) */
sysreg = *Preg* ; /* 32-bit P-register to sysreg (a) */
sysreg = *USP* ; /* (a) */
A0 = A1 ; /* move 40-bit Accumulator value (b) */
A1 = A0 ; /* move 40-bit Accumulator value (b) */
A0 = *Dreg* ; /* 32-bit D-register to 40-bit A0, sign extended
(b)*/
A1 = *Dreg* ; /* 32-bit D-register to 40-bit A1, sign extended
(b)*/

Accumulator to D-register Move:

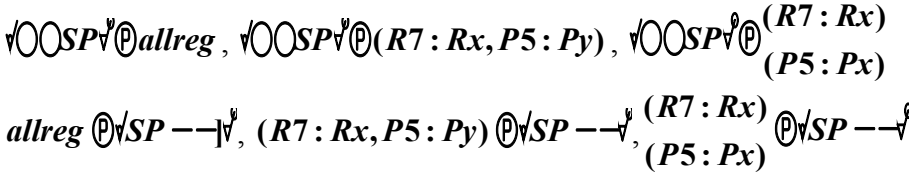
Dreg_even = A0 (*opt_mode*) ; /* move 32-bit A0.W to even Dreg
(b) */
Dreg_odd = A1 (*opt_mode*) ; /* move 32-bit A1.W to odd Dreg (b)
*/
Dreg_even = A0, *Dreg_odd* = A1 (*opt_mode*) ; /* move both Accumulators
to a register pair (b) */

Dreg_odd = A1, *Dreg_even* = A0 (*opt_mode*) ; /* move both Accumulators
to a register pair (b) */

Возможны условные пересылки в формате:

16-разрядные половины регистров можно пересылать в 32-х разрядные регистры со знаковым или нулевым расширениями. Возможны также пересылки в соответствующие половины аккумуляторов и пересылки из расширенной части аккумуляторов.

Работа со стеком сводится к пересылкам в следующем формате:



4.5 Программный автомат

Программный автомат отвечает за организацию циклов, программных переходов, а также организацию обслуживания прерываний и исключений.

Таблица 4-1. Регистры программного автомата

Имя регистра	Описание
SEQSTAT	Регистр состояния программного автомата
RETX RETN RETI RETE RETS	Регистры адреса возврата: см. раздел "События и выполнение программы". Возврат из исключения Возврат из немаскируемого прерывания Возврат из прерывания Возврат из эмуляции Возврат из подпрограммы
LC0, LC1 LT0, LT1 LB0, LB1	Регистры циклов с нулевыми непроизводительными затратами: Счётчики циклов Регистры начала цикла Регистры конца цикла
FP, SP	Указатель кадра и указатель стека: см. раздел "Указатели кадра и стека" в главе 5.
SYSCFG	Регистр конфигурации системы
CYCLES, CYCLES2	Счётчики тактов: см. раздел "Регистры счётчиков тактов выполнения (CYCLES и CYCLES2)" в главе 19.
PC	Счётчик команд

Регистр состояния программного автомата (SEQSTAT), содержит информацию о текущем состоянии программного автомата, а также диагностическую информацию о последнем событии. Регистр SEQSTAT доступен только для чтения и только в режиме Супервизора.

Два набора регистров циклов с нулевыми непроизводительными затратами обеспечивают организацию циклов, используя для оценки условий выхода из циклов аппаратные счётчики вместо команд программы. После оценки условия выполнение программы начинается с нового адреса.

Регистр конфигурации системы (SYSCFG), управляет конфигурацией процессора. Данный регистр доступен только в режиме Супервизора.

4.6 Переходы и управление переходами

Одним из типов непоследовательного выполнения программы, поддерживаемых программным автоматом, являются переходы. Переход происходит, когда по команде JUMP или CALL процессор начинает выполнение команд с адреса, отличного от следующего адреса в линейной последовательности адресов. При выполнении команд JUMP и CALL процесс выполнения программы переходит к другому адресу. Различие между командами JUMP и CALL заключается в том, что при выполнении команды CALL адрес возврата автоматически помещается в регистр RETS. Адресом возврата является адрес, следующий за адресом команды CALL. Таким образом, адрес возврата становится доступен команде возврата, соответствующей команде CALL, что упрощает организацию выхода из подпрограммы. По команде возврата программный автомат выполняет выборку команды по адресу возврата, содержащемуся в регистре RETS (при возврате из подпрограммы). К типам команд возврата относятся возврат из подпрограммы (RTS), возврат из прерывания (RTI), возврат из исключения (RTX), возврат из эмуляции (RTE) и возврат из немаскируемого прерывания (RTN). Каждому типу возврата соответствует отдельный регистр для хранения адреса возврата.

Переходы могут быть прямыми и косвенными. При прямом переходе адрес задаётся командой (например, JUMP 0x30), а при косвенном переходе адрес формируется генератором адреса данных (например, JUMP (P3)). Все типы команд JUMP и CALL могут быть относительными (выполняются относительно счётчика команд JUMP(PC+P3)). Косвенные переходы могут быть абсолютными или относительными. В зависимости от расстояния перехода различают короткие JUMP.S и длинные JUMP.L переходы. Возможно использование универсальной мнемоники JUMP.

Команды JUMP могут быть условными, зависящими от состояния бита CC в регистре ASTAT. Команды такого типа выполняются без задержки. Программный автомат может оценивать значение бита состояния CC для принятия решения о необходимости выполнения перехода. Если условие не определено, переход выполняется всегда. В условных командах JUMP для снижения задержки, вызванной конвейером, используется статическое предсказание переходов.

Флаг кода условия.

Процессор поддерживает использование бита флага кода условия (CC), применяемого для принятия решения о необходимости перехода. Доступ к данному флагу может осуществляться в восьми случаях:

- ◯ Значение CC используется для оценки условного перехода.
- ◯ Значение регистра данных может быть скопировано в CC, и значение CC может быть скопировано в регистр данных.
- ◯ Флаг состояния может быть скопирован в CC, и значение CC может быть скопировано во флаг состояния.
- ◯ Флаг CC используется в команде BITTST.
- ◯ Бит CC может быть установлен в соответствии с результатом сравнения регистров указателей.
- ◯ Бит CC может быть установлен в соответствии с результатом сравнения регистров данных.
- ◯ В некоторых командах устройства сдвига (циклический сдвиг или BXOR) CC используется в качестве части операнда/результата.
- ◯ Бит CC может устанавливаться командами проверки и установки битов.

Перечисленные способы использования бита CC применяются при управлении процессом выполнения программы. Команды перехода не относятся к командам, устанавливающим флаги арифметического состояния. В коде команды присутствует

отдельный бит, определяющий интерпретацию значения CC. Данный бит указывает на необходимость перехода по истинному или по ложному значению CC.

Операции сравнения имеют вид CC = expr, где expr (выражение) включает пару регистров одного типа (например, регистров данных, регистров указателей или комбинацию регистра и небольшой непосредственно заданной константы). Непосредственно задаваемая константа может быть 3-разрядным (от –4 до 3) знаковым числом при знаковом сравнении или 3-разрядным (от 0 до 7) беззнаковым числом при беззнаковом сравнении. Бит CC может устанавливаться по результату операций “равно” (==), “меньше” (<) или “меньше или равно” (<=). Существуют также операции проверки битов, определяющие, установлен ли бит в 32-разрядном регистре, и устанавливающие или сбрасывающие бит CC.

CC = Dreg == Dreg ; / equal, register, signed (a) */*
CC = Dreg == imm3 ; / equal, immediate, signed (a) */*
CC = Dreg < Dreg ; / less than, register, signed (a) */*
CC = Dreg < imm3 ; / less than, immediate, signed (a) */*
CC = Dreg <= Dreg ; / less than or equal, register, signed (a) */*
CC = Dreg <= imm3 ; / less than or equal, immediate, signed (a) */*
CC = Dreg < Dreg (IU) ; / less than, register, unsigned (a) */*
CC = Dreg < uimm3 (IU) ; / less than, immediate, unsigned (a) */*
CC = Dreg <= Dreg (IU) ; / less than or equal, register, unsigned (a) */*
CC = Dreg <= uimm3 (IU) ; / less than or equal, immediate unsigned (a)*/*
CC = *Preg == Preg* ; /* equal, register, signed (a) */
CC = *Preg == imm3* ; /* equal, immediate, signed (a) */
CC = *Preg < Preg* ; /* less than, register, signed (a) */
CC = *Preg < imm3* ; /* less than, immediate, signed (a) */
CC = *Preg <= Preg* ; /* less than or equal, register, signed (a) */
CC = *Preg <= imm3* ; /* less than or equal, immediate, signed (a) */
CC = *Preg < Preg* (IU) ; /* less than, register, unsigned (a) */
CC = *Preg < uimm3* (IU) ; /* less than, immediate, unsigned (a) */
CC = *Preg <= Preg* (IU) ; /* less than or equal, register, unsigned (a) */
CC = *Preg <= uimm3* (IU) ; /* less than or equal, immediate unsigned (a) */

Программный автомат поддерживает механизм организации циклов с нулевыми непроизводительными затратами. Программный автомат содержит два устройства организации циклов, каждое из которых включает три регистра – регистр начала цикла (LT0, LT1), регистр конца цикла (LB0, LB1) и регистр счётчика цикла (LC0, LC1).

Когда выполняется команда по адресу X, и X совпадает с содержимым LB0, следующей выполняемой команда будет команда по адресу, содержащемуся в LT0. Другими словами, когда PC == LB0, выполняется явный переход к адресу, являющемуся содержимым LT0.

Возврат к началу цикла возможен только, если значение счётчика больше или равно 2. Если значение счётчика не равно нулю, счётчик декрементируется на 1. Например, рассмотрим случай цикла с двумя итерациями. В начале значение счётчика равняется 2. При достижении конца первой итерации цикла, значение счётчика декрементируется и становится равным 1, и процесс выполнения программы переходит к началу тела цикла (для выполнения второй итерации). Когда вновь достигается конец цикла, счётчик декрементируется и становится равным 0, но возврат к началу тела цикла не производится (так как команды в теле цикла были выполнены дважды).

Так как имеется два устройства организации цикла, циклу 1 назначен больший приоритет, и он может использоваться в качестве внутреннего во вложенных циклах. Другими словами, при возврате к началу тела цикла 1 по определённой команде (PC == LB1, LC1 >= 2), даже при совпадении адресов, в цикле 0 по этой команде не произойдёт переход к началу тела

цикла. Возврат к началу тела цикла 0 возможен только по истечению счётчика цикла 1.
Для одновременной загрузки всех трёх регистров устройства организации цикла может использоваться команда LSETUP. Каждый из регистров цикла может быть загружен индивидуально при помощи регистровых передач, однако, это может повлечь значительные непроизводительные затраты, если во время передачи счётчик цикла содержит ненулевое значение (цикл активен).

Листинг 4-1. Пример цикла

```
P5 = 0x20;
LSETUP ( lp_start, lp_end ) LCO = P5;
lp_start:
R5 = R0 + R1(ns) || R2 = [P2++] || R3 = [I1++];
lp_end: R5 = R5 + R2;
```

При выполнении команды LSETUP программный автомат загружает адрес последней команды цикла в LBx и адрес первой команды цикла в LTx. Адреса начала и конца цикла вычисляются относительно счетчика команд на основании адреса команды LSETUP и смещения. В каждом случае к положению команды LSETUP прибавляется значение смещения.

LC0 и LC1 являются беззнаковыми 32-разрядными регистрами, каждый из которых поддерживает до 2^{32}-1 итераций в цикле.

Существует и другая форма команды организации циклов:

LOOP loop_name loop_counter	
LOOP_BEGIN loop_name	первая инструкция цикла
LOOP_END loop_name	последняя инструкция цикла

5 Основы синтеза высокопроизводительных микропроцессорных систем обработки данных

Наиболее известной классификацией высокопроизводительных микропроцессорных систем обработки данных, является предложенная М. Флинном в конце 60-х годов прошлого века. Она базируется на понятиях двух потоков: команд и данных. На основе числа этих потоков выделяется четыре класса архитектур:

1. SISD (Single Instruction Single Data) - единственный поток команд и единственный поток данных. По сути дела это классическая машина фон Неймана. К этому классу относятся все однопроцессорные системы.
2. SIMD (Single Instruction Multiple Data) - единственный поток команд и множественный поток данных. Типичными представителями являются матричные компьютеры, в которых все процессорные элементы выполняют одну и ту же программу, применяемую к своим (различным для каждого ПЭ) локальным данным. Некоторые авторы к этому классу относят и векторно-конвейерные компьютеры, если каждый элемент вектора рассматривать как отдельный элемент потока данных. 3. MISD (Multiple Instruction Single Date) - множественный поток команд и единственный поток данных. М. Флинн не смог привести ни одного примера реально существующей системы, работающей на этом принципе. Некоторые авторы в качестве представителей такой архитектуры называют векторно-конвейерные компьютеры, однако такая точка зрения не получила широкой поддержки. 4. MIMD (Multiple Instruction Multiple Date) - множественный поток команд и множественный поток данных. К этому классу относятся практически все современные многопроцессорные системы. Данная классификация не является всеобъемлющей и на данный мо- мент имеется целый ряд архитектур вычислительных систем, имеющих соб- ственное название и полностью или частично подпадающих под приведен- ную классификацию. Рассмотрим некоторые из них.

5.1 VLIW - процессоры

Устройства, в которых сравнительно длинная команда (суперкоманда)
делится на множество полей, каждое из которых управляет отдельным опе- рационным блоком. Такие структуры получили название VLIW (Very Long Instruction Word). Недостатком VLIW процессоров следует признать слож- ность разработки компиляторов, способных эффективно формировать супер- команды.

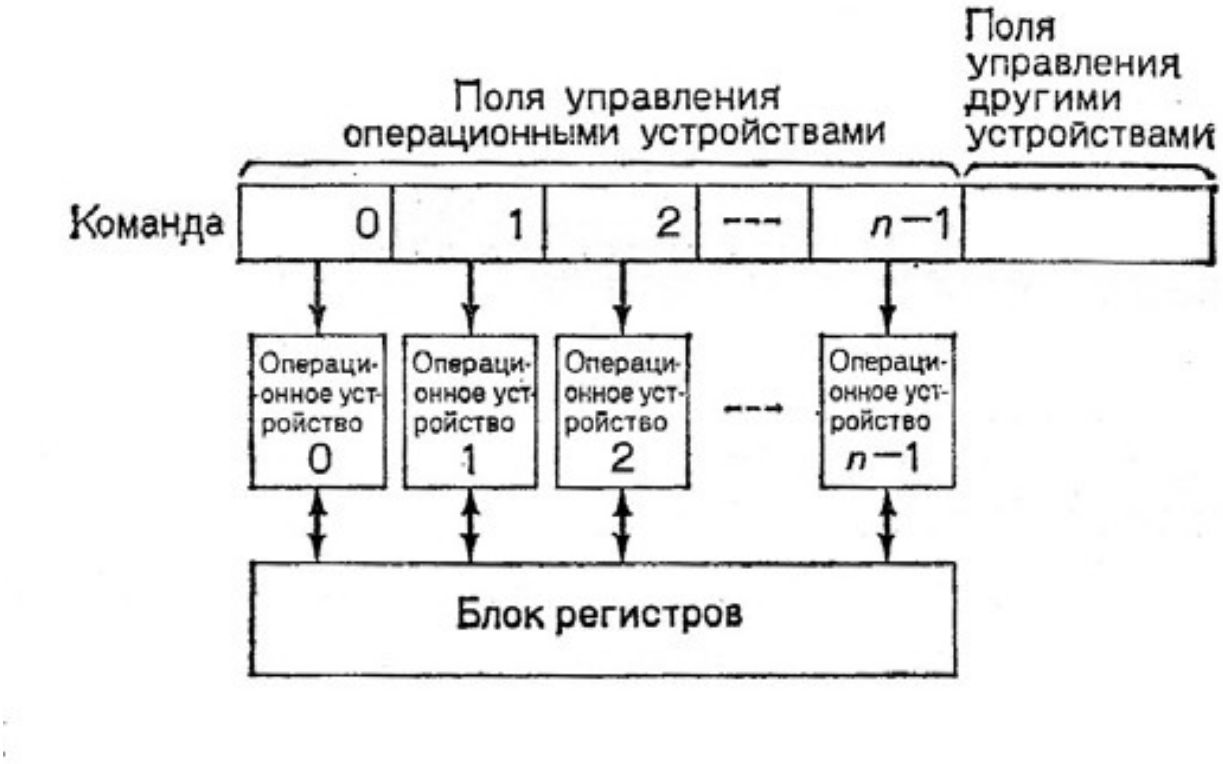


Рисунок 5.1 — Структура VLIW-процессоров

5.2 SIMD - системы

Устройства с одним потоком команд и несколькими потоками данных (Single Instruction Multiple Data stream). В этих системах одна команда парал- лельно выполняется несколькими операционными узлами, выполняющими одну и ту же операцию. Управляющее устройство разрешает или запрещает выполнение операции устройствам в зависимости от состояния специального флага (SF).

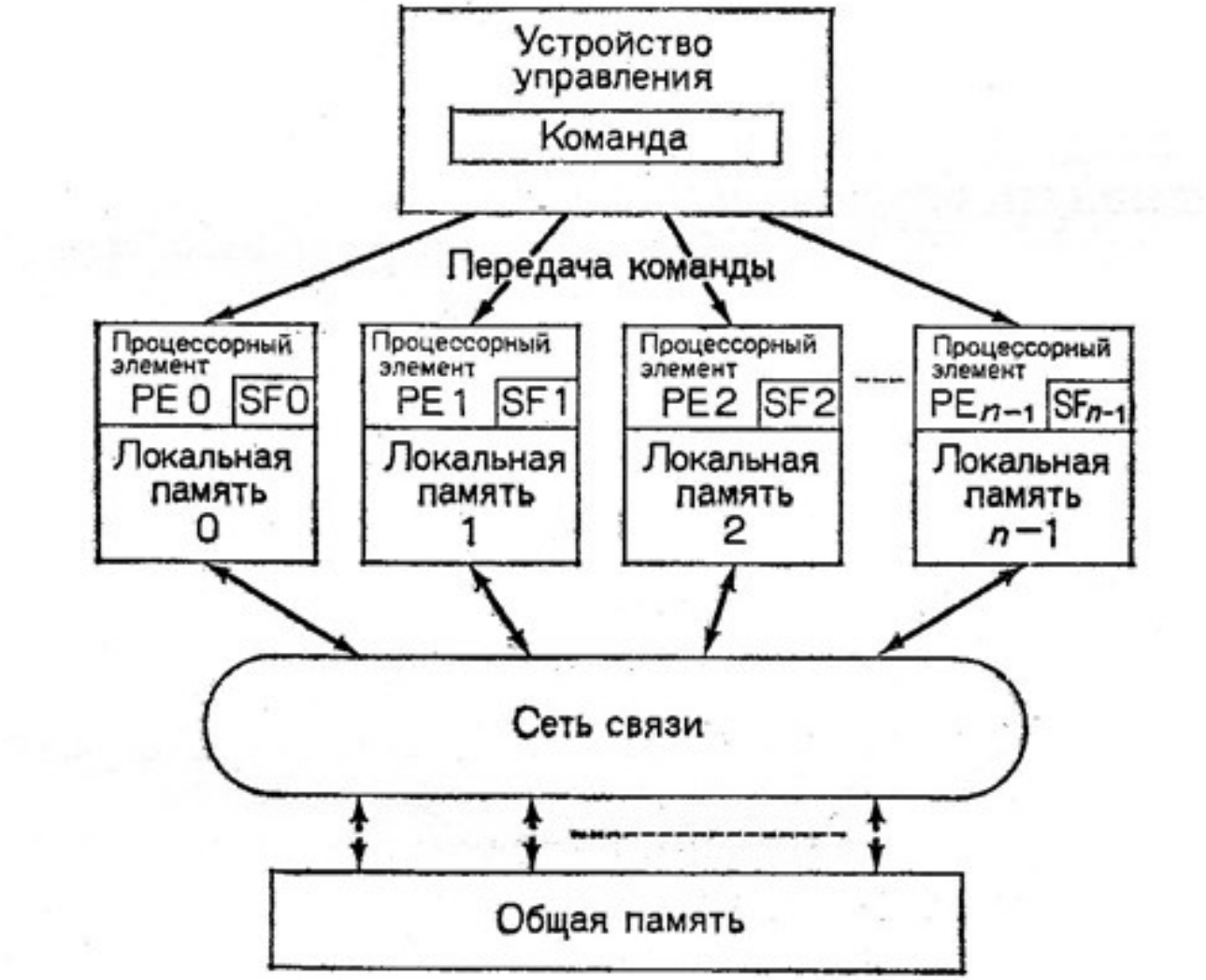


Рисунок 5.2 — Структура SIMD - системы

5.3 Векторные процессоры

Векторные процессоры, в отличие от скалярных, в качестве операндов могут использовать векторы (матрицы). При этом в структуре процессора предусматриваются отдельные узлы для выполнения отдельных операций, работающие по принципу конвейера или же параллельно выполняющие один и тот же алгоритм, что переводит их в группу SIMD.

5.4 Мультипроцессорные системы

Реализуют архитектуру MIMD. Предназначены для распараллеливания обработки за счет взаимодействия множества процессоров. Известны систе- мы с одной шиной, с кольцевой шиной, матричной коммутацией, многоуров- невой шиной, многоступенчатой коммутацией. Фирмой INMOS разработаны специальные блоки (транспьютеры), имеющие специальные порты для орга- низации взаимодействия между отдельными блоками, и предназначенные для построения мультипроцессоров с расширенными коммутационными функциями. Разработан также специальный язык Occam, эффективно реали- зующий внутрипроцессорные связи между блоками. На данный момент концепция MIMD воплощается в многоядерных процессорах.

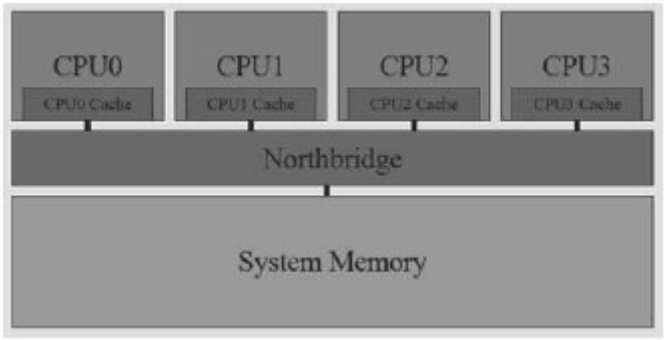


Рисунок 5.3 — Архитектура SMP
(Synchronous Multi Processing)

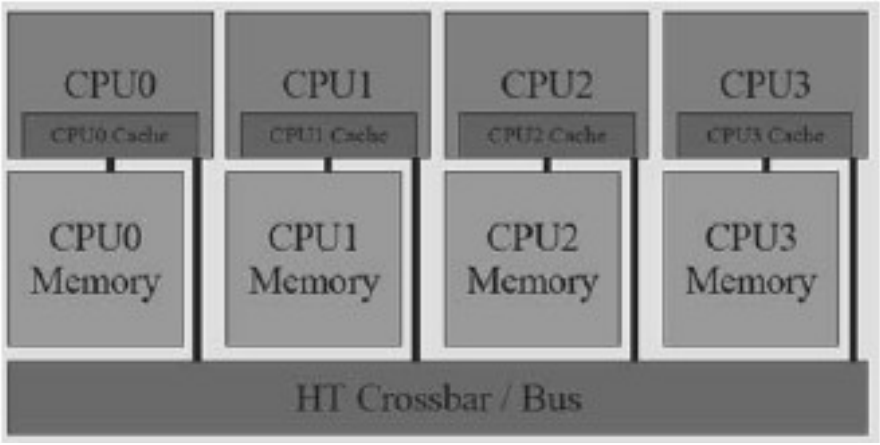


Рисунок 5.4 — Архитектура NUMA
(Non Uniform Memory Access)

Одной из главных проблем в мультипроцессорных системах является взаимодействие с памятью. Очевидно, что при одновременном обращении к уникальному ресурсу двух процессоров один из них будет вынужден про- стаивать, снижая общую эффективность системы. Вариант SMP предусмат- ривает общую шину, которая, как раз и является «узким местом», хотя про- блема частично решается за счет наличия у каждого процессора собственно- го кэша.

5.5 Суперскалярный процессор

В процессорах этого типа имеется возможность одновременно выпол- нять несколько команд (параллелизм на уровне команд). Для этого в составе такого процессора должно иметься устройство, определяющее степень неза- висимости команд и распределяющая их. Разновидность суперскалярного процессора – процессор Out-Of-Order, обеспечивающий переупорядочение потока команд с целью оптимизации.

5.6 Машины, управляемые потоками данных

В машинах, управляемых данными, команды с готовыми операндами могут выполняться одновременно, в отличие от систем с контроллерным управлением. В машине, управляемой данными, результаты выполнения не- которых команд передаются непосредственно

машинным командам, нуж- дающимся в них. При управлении данными снимается проблема переписы-
вания глобальных переменных и др., представляющих проблему в машинах с управлением контроллерами. Различают три способа организации машин, управляемых потоками данных:
− С жестким объединением операционных узлов, при этом достигается вы- сокая скорость, но теряется гибкость;
− С коммутируемым (матричный переключатель) объединением узлов, ог- раничен сложностью коммутатора;
− С логическим объединением операционных узлов с помощью сети связи (пакетная связь).

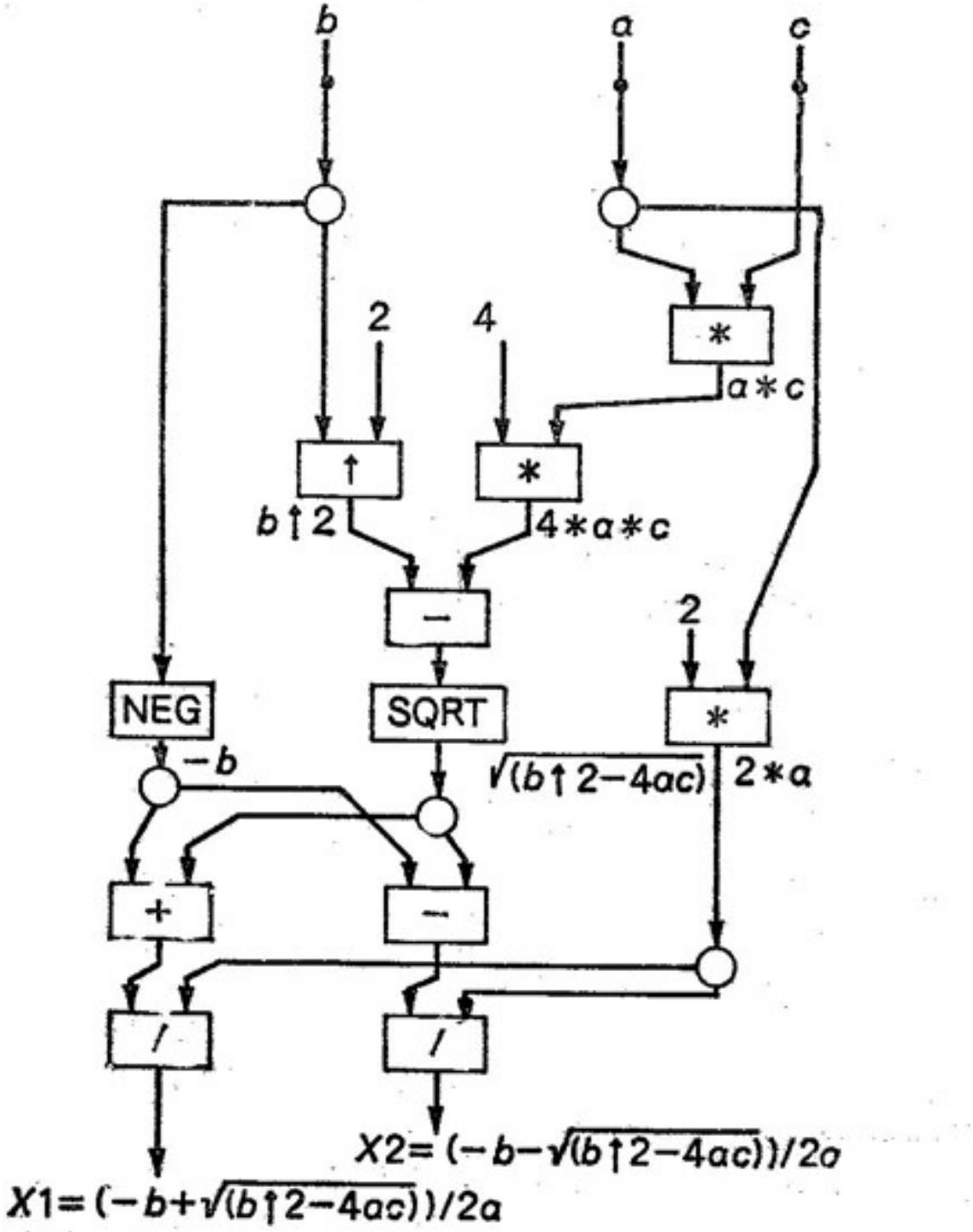


Рисунок 5.5 — Пример реализации машины, управляемой потоком данных

6 Техника разработки устройств управления и обработки данных на базе ПЛИС

В настоящее время от разработчика цифровых устройств требуется не только зна-ние основ алгебры логики, принципов действия основных узлов цифровой техники и вла-дение классическими принципами синтеза комбинационных и последовательностных ав-томатов, но и владение современной элементной базой, включая современные методы

проектирования. Технологии разработки цифровой аппаратуры, преобладающие в XX веке, не могут удовлетворить требованиям XXI века по ряду причин, главные из которых:

• Недостаточная гибкость, значительный интервал времени от начала разработки до

выхода готовой продукции;

• Трудности с реализацией высокоскоростных интерфейсов, обусловленные физическими ограничениями печатных плат.

Решением этих проблем является применение программируемых логических интегральных схем (ПЛИС), представляющих собой набор конфигурируемых логических блоков, из которых разработчик может создать разнообразные структуры, не изменяя печатной платы устройства. Разработка цифрового устройства на базе ПЛИС сводится к разра-ботке принципиальной схемы или текстового описания на специальном языке описания аппаратуры и оптимизации структуры под конкретные требования. Рынок современных ПЛИС достаточно широк, можно перечислить такие фирмы, как XILINX, ALTERA, ACTEL, INTEL, LATTICE, ATMEL.

Системы на кристалле

ПЛИС "Система на кристалле" предполагают наличие на кристалле специализированных областей (аппаратные ядра), реализующие определенные функции. Введение та-ких ядер снижает универсальность ПЛИС, но сокращает площадь кристалла при реализа-ции сложных функций. В качестве аппаратных ядер используются блоки ОЗУ, умножите-ли, ядра интерфейса JTAG, контроллеры PCI и др. Примеры - микросхемы FLEX фирмы

ALTERA и VIRTEX фирмы XILINX. В настоящее время системы на кристалле (SoC) по-лучили широкое распространение, причем на кристалле могут присутствовать и аналоговые и цифровые узлы различной степени интеграции, что позволяет практически на одном кристалле реализовать целое устройство, для которого ранее потребовалось бы несколько отдельных микросхем разного класса.

JTAG

Интерфейс JTAG - это совокупность средств и операций, позволяющая произво-дить тестирование БИС без физического доступа к каждому ее выводу. Тестирование схе-мы по стандарту JTAG называется также периферийным сканированием. Для обеспечения периферийного сканирования в состав схемы должен входить набор ячеек периферийного сканирования (BSC - Boundary Scan Cells) и схем управления их работой. Ячейки BSC располагаются между выводом устройства и внутренней схемой и работают в двух режимах - пропуска сигналов и тестирования. В первом режиме BSC просто пропускают сигнал, не влияя на него, во втором - объединяются в регистр, который может принимать и переда-вать

данные. С помощью JTAG можно проверять функционирование самих микросхем, а также соединения между ними, для чего проверяемые трассы должны объединять ИМС с JTAG.

ПЛИС фирмы ALTERA

В настоящее время используются следующие семейства ПЛИС:

− MAX3000, MAXII – CPLD

− Семейства Cyclone, Stratix, Arria - FPGA

Stratix II — IV — мощные FPGA с числом вентилей до 800000, имеющие на кри-сталле встроенные множители, цифровые преимопередатчики.

Семейство Cyclone считается экономичным как с позиции энергопотребления, так и по цене, тем не менее имеющие до 150000 вентилей и способные решать широкий круг задач.

Семейство Arria – для телекоммуникационных применений.

Семейство MAX — экономичные CPLD,

ПЛИС фирмы XILINX

В настоящее время ПЛИС фирмы XILINX можно условно разделить на следующие семейства:

CPLD серий XC, CoolRunner XPLA3 и CoolRunner II – разнообразные конфигура-ции ;FPGA Spartan-3,Spartan-5, Spartan-6 - FPGA с емкостью до 200000 логических ячеек

FPGA Virtex, Virtex-II, 4,6 - FPGA и системы на кристалле, включающие в себя ядра процессоров PowerPC, нацеленные на разработку систем цифровой обработки сигна-лов.

6.1 Использование языка VHDL

Возрастающая степень интеграции ПЛИС, новые концепции проектирования (си-стема на кристалле) накладывают свой отпечаток на способы описания проекта на ПЛИС.

Языки описания аппаратуры (Hardware Description Language), являются формальной запи-сью, которая может быть использована на всех этапах разработки цифровых электронных систем. Это возможно вследствие того, что язык легко воспринимается как машиной, так и человеком. Он может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры так же, как и для передачи данных о проекте, модификации и сопровождения. Существует несколько разновидностей этих языков: AHDL, VHDL, VerilogHDL, Abel и др. Известны также случаи использования стандартных языков про-граммирования, например Си, для описания структуры БИС.

Одним из наиболее универсальных языков описания аппаратуры является VHDL, первый стандарт которого был разработан в 1983–1987 годах при спонсорстве миноборо-ны США. На этом языке возможно как поведенческое, так структурное и потоковое описа-ние цифровых схем.VHDL поддерживает три различных стиля для описания аппаратных архитектур.• Первый из них - структурное описание (structural description), в котором архитекту-ра представляется в виде иерархии связанных компонентов. • Второй - потоковое описание (data-flow description), в котором архитектура пред-ставляется в виде множества параллельных операций языка, каждая из которых мо-жет управляться логическими сигналами. Потоковое описание соответствует стилю описания, используемому в языках регистровых передач. • И, наконец, поведенческое описание (behavioral description), в котором логические преобразования описываются последовательными программными

предложениями, которые похожи на имеющиеся в любом современном языке программирования высокого уровня. Все три стиля могут совместно использоваться в одной VHDL программе.

Структурное и потоковое описание используется для проектирования цифровых схем, поведенческое - в основном для моделирования.

Наиболее важными в языке VHDL являются такие понятия как объект проекта, интерфейс, порт, архитектура, сигнал, процесс, параллелизм и иерархия.

Объект проекта (entity) представляет собой описание компонента проекта, имею-щего чётко заданные входы и выходы и выполняющие чётко определённую функцию.

Объект проекта может представлять всю проектируемую систему, или некоторую подси-стему, При этом в описании объекта проекта можно использовать компоненты, которые, в свою очередь, могут быть описаны как самостоятельные объекты проекта более низкого уровня. Каждый компонент объекта проекта может быть связан с объектом проекта более низкого уровня. В результате такой декомпозиции пользователь строит иерархию объек-тов проекта, представляющих весь проект в целом и состоящую из нескольких уровней абстракций. Такая совокупность объектов проекта называется Иерархией проекта (design hierarchy). Каждый объект проекта состоит, как минимум, из двух различных типов опи-саний: описания интерфейса и одного или более архитектурных тел.

Интерфейс описывается в Объявлении объекта проекта (entity declaration) и определяет только входы и выходы объекта проекта.

Для описания поведения объекта или его структуры служит архитектурное тело (architecture body).

В языке VHDL предусмотрен механизм пакетов для часто используемых описаний, констант, типов, сигналов. Эти описания помещаются в Объявление пакета (package declaration).

Если пользователь использует нестандартные операции или функции, их интерфей-сы описываются в объявлении пакета, а тела содержатся в Теле пакета (package body).

Таким образом, при описании цифровых схем на языке VHDL, возможно использование

пяти различных типов описаний: объявление объекта проекта, архитектурное тело, объяв-ление конфигурации, объявление пакета и тело пакета. Каждое из описаний является

самостоятельной конструкцией языка VHDL, и поэтому оно может быть независимо про-анализировано моделирующей программой. Такое описание получило название Модуль

проекта (design unit).

Модули проекта, в свою очередь, можно разбить на две категории: первичные и вторичные. К первичным модулям относятся различного типа объявления. К вторичным - отдельно анализируемые тела первичных модулей. Один или несколько модулей проекта могут быть записаны в один файл, который называется файлом проекта (design file).

Каждый проанализированный модуль проекта помещается компилятором Active VHDL в Библиотеку проекта (design library) и становится Библиотечным модулем (library unit). Такой подход позволяет создать любое число библиотек проекта. Каждая библиотека проекта в языке VHDL имеет логическое имя (идентификатор). Фактическое имя файла, содержащего эту библиотеку, может совпадать или не совпадать с логическим именем библиотеки проекта.

Объекты данных (data object) являются хранилищами для значений определённого

типа. Следует заметить, что VHDL - сильно типизированный язык, и он допускает опера-ции только с данными одного типа. Для преобразования типов служат ряд специально раз-работанных функций, которые обычно группируются в пакеты. Описание на VHDL содер-жат объявления, которые создают объекты данных всего лишь четырёх классов: констан-ты, переменные, сигналы и файлы.

Алфавит VHDL

При работе с языком VHDL следует принимать во внимание следующие особенно-сти синтаксиса:

1. Программы на VHDL не различают прописные и заглавные буквы, т.е нет разницы

между верхним и нижним регистром;

2. Идентификатор должен начинаться с буквы и состоять из букв и цифр;

3. Допускается применение знака подчеркивания, но не двух подряд и не последним сим-волом;

4. VHDL содержит набор ключевых слов, которые не могут быть идентификаторами;

5. Комментарии начинаются двойным тире и продолжаются до конца строки.

Литералы могут быть следующих типов:

1. Числовые (целые и вещественные, допускается символ E, указывающий на порядок);

2. Символьные (символ между апострофами, например 'A');

3. Строка символов (последовательность символов в двойных кавычках, располагающая-ся в одной строке);

4. Строка бит (последовательность цифр и символов A,B,C,D,E,F в двойных кавычках.

Строка бит может быть двоичной, восьмеричной или шестнадцатеричной, на что ука-зывает соответственно символ B, O, X перед кавычками. Например X"AB12". Допус-кается применение подчеркивания в таких константах, но оно не является значимым.

Типы

1. Целый (integer)

2. Логический (boolean принимает два значения true и false);

3. Битовый (BIT принимает значения '0' и '1');

4. STD_LOGIC и STD_ULOGIC - типы, определенные стандартом IEEE 1164;

5. Перечислимый (ENUMERATED), используемый для задания пользовательских типов;

6. Символьный (CHARACTER);

7. Физический (время).

Допускается использование массивов (ARRAY) следующих предопределенных ти-пов:

1. BIT_VECTOR;

2. STD_LOGIC_VECTOR и STD_ULOGIC_VECTOR;

3. STRING.

Указанные массивы представляют собой одномерные массивы соответствующих

типов. Направление и границы диапазонов должны быть указаны при объявлении

объек-тов данных типов. Для объявления направления и границ диапазона используются ключе-вые слова to и downto.

Например, BIT_VECTOR(7 downto 0), STD_LOGIC_VECTOR(1 to 8). Также пользователь может объявить собственные массивы, пользуясь декларацией вида:

TYPE X IS ARRAY(1 to 2) OF REAL;

Аналогично объявляются перечислимые типы, состоящие из конечного множества

значений: TYPE DIGIT IS (0,1,2,3,4,5,6,7,8,9);

Допускается объявление объектов целого типа с ограниченным множеством значе-ний: TYPE BYTE_INT 0 TO 255;

Константы

В языке VHDL допускается объявление констант следующим образом:

CONSTANT <name> : <type> := initial value;

Применение констант улучшает читаемость и повышает гибкость VHDL - описа-ний.

Атрибуты

Атрибутами называются значения, связанные с именованными объектами языка

VHDL. Для целей разработки цифровой аппаратуры нам потребуется только атрибут

EVENT.

Ex<= '0' after 2s;

'1' after 5s;

'0' after 6s;

'1' after 8s;

Операторы

Язык VHDL предполагает выполнение операций над предусмотренными типами

данных слева направо и с учетом приоритета и скобок. Допускаются: возведение в степень, деление, умножение, вычисление остатка и модуля, сложение и вычитание. Логические операции: И, ИЛИ, НЕ, исключающее ИЛИ, И-НЕ, ИЛИ-НЕ, исключающее ИЛИ-НЕ.

Операторы сравнения: <=,<,>=,>,=,/= (не равно). Объекты типа строка символов могут

сцепляться вместе с помощью операции конкатенации (&).

Операторы сложения и вычитания выполняются над данными одного типа, логиче-ские операции - над данными типа BIT, STD_LOGIC и соответствующими векторами.

Различают последовательные и параллельные операторы языка. Последователь-ные операторы выполняются один за другим и применяются в описании процессов, проце-дур и функций. Параллельные операторы выполняются одновременно, и порядок их вы-полнения не связан с местом расположения в тексте программы.

Составные части VHDL программы

VHDL программа объекта проекта состоит следующих основных частей:

• описание интерфейса объекта проекта (ENTITY), включающее (необязательно)

PORT (список входных и выходных сигналов) и GENERIC (список настраиваемых

констант, например, задержек, переключения и разрядностей данных);

• архитектура объекта проекта ARCHITECTURE , включающая объявление переменных и дополнительных (внутренних) сигналов проекта и операторную часть,

описывающую объект на структурном или поведенческом уровне

Важно придерживаться определенного стиля написания программы на VHDL. Правильное расположение составных частей программы в процессе ее написания поможет

избежать ряда ошибок как при ее создании, так и облегчит процесс сопровождения

программного продукта в течении его жизни.

Интерфейсная спецификация начинается с ключевого слова Entity и описывает

входные и выходные порты объекта. Другими словами, интерфейсная декларация (entity

declaration) описывает интерфейс между объектом и окружением, в котором находится

объект.

Синтаксис данной декларации следующий:

entity <identifier_name> is

port (<identifier> : [<mode>] <type_indication> ;

<identifier> : [<mode>] <type_indication>);

end [<identifier_name>] ;

В простейшем случае возможно отсутствие декларации port.

После ключевого слова port перечисляются сигналы, связывающие объект с внеш-ним миром. Если объект имеет несколько однотипных сигналов, их идентификаторы перечисляются через запятую. После двоеточия следует указание направления сигнала, кото-рое может принимать одно из следующих значений:

<mode> = in, out, inout, buffer, linkage

in = входной;

out = выходной;

inout = двунаправленный;

buffer = двунаправленный буферный;

linkage = связной (не используется при проектировании).

Пример:

Синтаксис языка VHDL допускает настраиваемое описание интерфейса объекта с

помощью ключевого слова generic. Его использование позволяет легко изменять настраи-ваемый параметр объекта, не внося других изменений в VHDL-код.

Entity Processor is

generic(BusWidth: Integer:=3);

port(DataBus: inout_bit_vector(BusWidth-1 downto 0);

end Processor

Архитектурная спецификация начинается с ключевого слова Architecture и опи-сывает

функциональное назначение проектируемого объекта. Другими словами, архитек-турная декларация определяет тело компоненты проектируемого объекта, выполняемые

функции которой зависят от значений входных и выходных сигналов или других парамет-ров, задаваемых в интерфейсной декларации. Имя Архитектурной спецификации может

быть любым, однако сама декларация привязана к имени Интерфейсной спецификации.

Синтаксис данной декларации следующий:

architecture <architecture_name> of <entity_identifier> is

[<architecture_declarative_part>]

begin

<architecture_statement_part>

end [<architecture_name>];

Поскольку интерфейсному описанию одного проектируемого объекта может соот-ветствовать несколько архитектурных деклараций с разными идентификаторами, то мы

можем описывать устройство с разной степенью детализации.

В декларативной части архитектуры могут содержаться объявления типов, сигна-лов, констант, подпрограмм и компонентов.

Описание архитектуры может быть выполнено в соответствии с одним из трех ти-пов VHDL - описания: структурным, потоковым и поведенческим.

Сигналы и процессы

Понятие "сигнал" соответствует физическому проводнику на схеме устройства.

Сигналу может быть присвоено значение с помощью оператора <=. Логические сигналы

могут передаваться и обрабатываться параллельно, для чего в VHDL предусмотрен меха-низм процессов. Каждый процесс представляет собой блок моделируемой схемы, причем

все процессы выполняются параллельно.

Интерфейс объекта, описываемый в декларации entity, использует сигналы, посту-пающие на вход устройства и формируемые им. Сигналы, не перечисленные в entity,

должны быть объявлены в декларативной части архитектуры.

Синтаксис процесса имеет следующий вид:

<label:> process [<(sensitivity list)>]

<[process_declarations]>;

begin

<sequential operators>;

end process [<label>];

Процесс может иметь список чувствительности, в котором перечислены сигналы,

активизирующие процесс. Каждый сигнал должен иметь только один источник (драйвер),

в противном случае потребуется специальная функция разрешения. Каждый процесс

может находиться в одном из трех состояний:

− выполняющийся (когда выполняется системой моделирования);

− активный (ожидающий выполнения);

− приостановленный (когда не является активным или выполняющимся).

Моделирование работы устройства посредством механизма процессов осуще-ствляется следующим образом:

Параллельные процессы, которые необходимо выполнить (активные) формируют

очередь, из которой ЭВМ выбирает один процесс и выполняет его, после чего он считает-ся приостановленным. Затем выполняется следующий активный процесс, приостанавлива-ется, и так далее до тех пор, пока в очереди остаются активные процессы. После этого на-чинается следующий цикл моделирования. Приостановленный процесс становится актив-ным при изменении хотя бы одного сигнала из списка чувствительности. Если у процесса

отсутствует список чувствительности, то он выполняется всегда. Альтернативой списку

чувствительности является оператор wait. Наличие сигнала в списке чувствительности эк-вивалентно оператору wait on в конце процесса. Запрещается совместное использование

wait и списка чувствительности.

Переменные

Переменные могут быть объявлены в области операторов процессов и подпро-грамм, область их видимости будет ограничена телом того объекта, внутри которого она

декларирована (в отличие от сигналов, которые не могут быть объявлены внутри процесса

или подпрограммы). Значение переменным может быть присвоено в любой момент с по-мощью оператора := (изменение сигналов происходит только после окончания действия

процесса), понятие времени не ассоциируется с понятием переменной. Переменные могут

быть аргументами подпрограмм (так же, как и сигналы).

Условный оператор

Условный оператор является последовательным и имеет следующий синтаксис:

If condition then

Sequence of sequential statements

[elsif condition then

Sequence of sequential statements]

[else

Sequence of sequential statements]

End if;

Выражение condition должно иметь тип boolean. В операторе if может быть одна

или несколько (или не одной) частей elsif, и только одна (или не одной) часть else.

Оператор case

Оператор case также является последовательным оператором и имеет следующий

синтаксис:

Case expression is

When choice=> Sequence of sequential statements

[when choice=> Sequence of sequential statements]

[when others=> Sequence of sequential statements]

end case;

Выражение expression должно иметь дискретный тип или тип одноразмерного мас-сива символов, состоящих из строк или строк битов. Выражение choice должно иметь та-кой же тип, как и expression. Все значения выбора должны быть покрыты. Выбор others

должен покрывать значения, не покрытые предыдущими альтернативами. Допускается

указание множественного выбора (с разделением символом "|"), а также указание диапазо-на (с применением to или downto).

Оператор цикла

Оператор цикла имеет следующий синтаксис:

[label:] [while condition | for identifier in discrete_range]

loop Sequence of sequential statements

end loop [label];

В случае использования в условии цикла конструкции while, сначала вычисляется

условие condition. При значении TRUE выполняется последовательность операторов, в

противном случае цикл завершается.

При необходимости пропустить выполнение оператора цикла при каком-либо усло-вии используется конструкция next со следующим синтаксисом:

Next when condition

При этом происходит переход к концу цикла и выполнение следующей итерации.

При необходимости прервать выполнение цикла используется оператор exit:

Exit label when condition

Параллельные операторы

Параллельные операторы определяют параллельное поведение схем. К параллель-ным относится оператор процесса и оператор параллельного вызова подпрограммы, кото-рый представляет собой оператор процесса с последовательным вызовом подпрограммы.

Параллельный оператор условного назначения сигнала

Синтаксис следующий:

Signal <= expression when condition else

[Signal <= expression when condition else ..]

```
q <= 3 WHEN high = '1' ELSE -- when high
2 WHEN mid  = '1' ELSE -- when mid but not high
1 WHEN low  = '1' ELSE -- when low but not mid or high
0; -- when not low, mid, or high
```

Данный оператор эквивалентен оператору if в процессе.

Параллельный оператор выборочного назначения сигнала

Синтаксис:

with selection select signal <= expression when choice, expression when choice…

Аналогично оператору case допускаются множественные выборы и применение ключево-го слова others.

```
WITH s SELECT -- creates a 4-to-1 multiplexer
output <= d0 WHEN 0,
d1 WHEN 1,
d2 WHEN 2,
d3 WHEN 3;
```

Оператор конкретизации компонента

Синтаксис:

Label : component name

Port_map_description;

Метка является обязательной, описание Port_map_description может быть двух ти-пов: с позиционным сопоставлением и с ключевым сопоставлением.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY meth IS
PORT(data, clock, clearn, presetn : IN STD_LOGIC;
q_out : OUT STD_LOGIC;
a, b, c, gn : IN STD_LOGIC;
d   : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
y, wn : OUT STD_LOGIC);
END meth;
ARCHITECTURE a OF meth IS
BEGIN
dff1 : DFF PORT MAP (d =>data, q => q_out, clk => clock, clrn => clearn, prn => presetn);
mux: MUXX PORT MAP (c, b, a, d, gn, y, wn);
END a;
```

В данном случае компонент DFF присоединен с ключевым соответствием (явно

указано соответствие сигналов), а выводы компонента MUXX присоединены по порядку:

первый вывод к c, второй - к b и т.д.

Для выходных неиспользуемых портов необходимо указать ключевое слово open.

P1: add1 port map(b1=>'0', b2=>'1', c1=>open);

Параллельный оператор генерации

Синтаксис:

label: for parameter_specification

generate [if condition generate]

parallel statements

end generate [label];

Параметр генерации - константа дискретного типа в определенном диапазоне (параметром не может быть объявленная переменная или сигнал).

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY meth IS
PORT (data, clock, clearn, presetn : IN STD_LOGIC;
q_out : OUT STD_LOGIC);
END meth;
ARCHITECTURE a OF meth IS
COMPONENT DRFF
  PORT (d,clk, clrn,prn : IN STD_LOGIC;
        q   : OUT STD_LOGIC);
END COMPONENT;
signal TS: STD_LOGIC_VECTOR(8 downto 0);
BEGIN
TS(8)<=data;
q_out<=TS(0);
G0:  for i in 7 downto 0 generate
Allbit:DRFF PORT MAP (d=>TS(i+1),q=>TS(i),clk=>clock,clrn=>clearn,prn=>presetn);
End generate;
END a;
entity DRFF is
PORT (d,clk,clrn,prn : IN STD_LOGIC;
        q   : OUT STD_LOGIC);
END DRFF;
```

```
Architecture ADRFF of DRFF is
begin
process(clk,clrn,prn)
begin
if clrn='1' then q<='0';
elsif prn='1' then q<='1';
elsif clk'event and clk='1' then q<=d;
end if;
end process;
end ADRFF;
```

Функции преобразования

В VHDL предусмотрены функции приведения типов: CONV_INTEGER(x), CONV_SIGNED(x,size), CONV_UNSIGNED(x,size), CONV_STD_LOGIC_VECTOR(x,size).

Данные функции доступны при инициализации арифметической библиотеки (USE ieee_std_logic_arith.all).

6.2 Синтез автоматов с памятью

Все автоматы с памятью делятся на синхронные и асинхронные. В асинхронных переключение триггеров производится под воздействием информационных сигналов, скорость переключения определяется задержками в цепях распространения сигналов.

В синхронных автоматах имеется глобальный сигнал синхронизации, под воздействием которого все элементы памяти переключаются одновременно, при этом все пере-ходные процессы завершаются в паузах между синхроимпульсами.

В настоящее время считается, что основным способом построения автоматов с памятью следует считать применение синхронных устройств.

Проектирование таких автоматов принято делить на следующие этапы:

1. Формализация задания проектирования;
2. Минимизация состояний;
3. Кодирование состояний;
4. Составление таблицы переходов;
5. Определение функций возбуждения триггеров;
6. Составление логической схемы автомата.

Формализация задания проектирования осуществляется путем перехода к табли-цам, логическим функциям и диаграммам состояния. Минимизация и кодирование со-стояний

осуществляются различными способами, в частности, путем объединения со-стояний с одинаковыми состояниями триггеров. При автоматизированном проектиро-вании возможна автоматизация данного процесса.

Автоматы могут быть построены на базе любых триггеров, однако наибольшее

распространение получили JK-триггеры, как универсальные, или D-триггеры, исполь-зование которых сокращает число требуемых функций возбуждения. При использовании ПЛИС, обладающих значительными ресурса-ми триггеров, а также при высоких требованиях к быстродействию автомата целесооб-разно воспользоваться способом кодирования «1 из N» (One Hot Encoding), при кото-ром число триггеров равно числу состояний в таблице переходов.

Комбинационная часть устройства строится с использованием логических эле-ментов, мультиплексоров или ПЛИС, также эффективным приемом может являться

использование готовых узлов средней степени интеграции типа счетчиков, регистров и

т.д.

Рассмотрим пример: Пусть необходимо разработать трехразрядный счетчик, ра-ботающий в коде Грея (при переходе от текущей комбинации к следующей меняется

только один разряд по принципу: 000-001-011-010-110-111-101-100).

Тогда таблица переходов будет иметь вид:

Q2n Q1n Q0n Q2n+1 Q1n+1 Q0n+1

0 0 0 0 0 1

0 0 1 0 1 1

0 1 0 1 1 0

0 1 1 0 1 0

1 0 0 0 0 0

1 0 1 1 0 0

1 1 0 1 1 1

1 1 1 1 0 1

7 Техника разработки программного обеспечения вычислительных и управляющих систем

Разработка программного обеспечения всегда считалась делом творческим, можноговорить о стиле того или иного программиста, можно обсуждать оформление листингапрограмм, вести дискуссии, но всетаки есть некоторые общие соображения, более-менее разделяемые всеми программистами.

Одно из главных требований к разработчику программного обеспечения для микроконтроллеров (особенно для работающих с языком ассемблера) – это необходимость полного контроля над всеми ресурсами и ответственность за все аспекты поведения микроконтроллера во всех режимах. Также для разработчиков, работающих с микроконтроллерами, актуальной является ограниченность ресурсов, прежде всего, памяти, что не особенно актуально для программистов, работающих на «больших компьютерах». Сформулируем основные «золотые» правила программиста микроконтроллерных систем.

1. Все задачи, выполняемые микроконтроллером, следует разделить на две группы:критичных ко времени и некритичных. Ко второй группе относится большинство задач взаимодействия с пользователем, к первой группе – измерения временных интервалов, обработка «быстрых» событий, работа со скоростными интерфейсами передачи данных.
2. Задачи, критичные ко времени, следует полностью или частично решать с помо щью механизма прерываний, еще лучше использовать возможности микрокон троллера, не загружающие ядро (таймеры, ПДП и т.п.)
3. Подпрограммы обработки прерываний должны выполняться максимально быстро, часто их функция сводится к установке флага, который затем проверяется в основ ной программе. Таким образом, реализуется отложенная обработка событий, по зволяющая максимально эффективно загружать ядро микроконтроллера.
4. Крайне нежелательно помещать в программы обработки прерываний какие-либо циклы ожидания, которые могут резко снизить производительность процессорного ядра и даже привести к «зависанию». Реализацию длительных пауз желательно возлагать на таймеры, не использующие для этого ресурсов процессорного ядра.
5. Длительные вычислительные процедуры и циклы ожидания желательно реализо вывать в основной программе таким образом, чтобы они допускали прерывания в любой момент времени.
6. С точки зрения энергопотребления во многих случаях целесообразно в основной программе переводить микроконтроллер в режимы пониженного энергопотребле ния с пробуждением от прерываний, вызванных внешними событиями. Таким образом, значительное количество программ микроконтроллеров, построен ных с учетом вышеприведенных требований может сводиться к общей схеме, в которой в основной программе реализуется бесконечный цикл опроса флагов, которые устанавли ваются подпрограммами обработки прерываний. Разумеется, кроме опроса флагов, в ос новной программе могут присутствовать регулярные операции, например, опрос нажатия кнопок или обслуживание индикаторов. Примерно таким образом построены операцион ные системы, в которых роль основной программы выполняет обработчик событий или планировщик. С развитием микроконтроллеров и снижением стоимости ресурсов, по- видимому, все большую роль будут играть инженерные решения на базе операционных систем различного типа, берущих на себя рутинные операции и сводящие задачу разра ботчика к написанию подпрограмм – процессов, входящих в операционную систему на уровне выполняемых потоков или разработке драйверов аппаратных устройств. Особен ности работы с

операционными системами мы рассмотрим позднее.

7.1 Разработка программ на языке ассемблера

Программист, использующий ассемблер, имеет максимальные возможности управ ления микроконтроллером (кроме разве что программистов, работающих на уровне ма шинных кодов), однако, на него же ложится максимум ответственности за мелкие детали работы процессора. Кроме знания системы команд и аппаратных особенностей исполь зуемого микроконтроллера, такому разработчику необходимо знать еще и особенности применяемого инструмента – компилятора. Необходимо помнить, что создаваемый лис тинг программы является заготовкой, транслируемой компилятором в машинный код, и результат такой трансляции зависит от настроек компилятора. Процессом трансляции можно управлять с помощью директив ассемблера, которые не транслируются в машин ные команды, но изменяют поведение компилятора, облегчают труд программиста и де лают программу более «читаемой».

Основные директивы ассемблера

− Директива ORG предписывает компилятору размещать следующий код, начиная с указанного адреса;

− Директива DEFINE позволяет определить некоторую константу, как удобочитаемую текстовую строку;

− Директива EQU подобна DEFINE и позволяет указать компилятору на эквивалент ность операндов;

− Директива DB (Define Byte) указывает компилятору на необходимость зарезервиро вать байт в памяти данных или программ и иногда проинициализировать эту память необходимым значением. Необходимо помнить, что инициализация с помощью дирек тив ячеек памяти в оперативной памяти большинства микроконтроллеров бессмыс ленна, так как директивы не являются исполняемыми командами процессорного ядра.

Аналогично DB используются директивы DW (Word), DD (Double Word).

− Директива INCLUDE позволяет включить в листинг исходный код из другого файла Кроме упомянутых общих директив, существует немало специальных директив, привязанных к конкретному компилятору или микроконтроллеру и намеренно здесь не рассматриваемых. Разработчик должен помнить, что для эффективной работы необходимо хорошо знать особенности конкретного рабочего инструмента, используемого для созда ния программы.

7.2 Программирование на языках высокого уровня

Наибольшей популярностью при разработке программного обеспечения микро контроллеров пользуются язык Си (Си++) и язык Java. Главным преимуществом языков высокого уровня является автоматическая реализация многих рутинных операций, кото рая существенно облегчает работу программиста по сравнению с программированием на ассемблере. Оборотной стороной является ограничение свободы программиста по реали зации частных решений и некоторое снижение эффективности кода в плане быстродей ствия и использования памяти. Как правило, программисты используют для решения та ких проблем ассемблерные вставки в наиболее критических местах программы.

Развитие микроконтроллеров, переход на 16-ти и 32-разрядные архитектуры делает программирование ня языках высокого уровня более предпочтительным, так как на пер вый план выходит не экономия памяти или достижение быстродействия, а срок разра ботки работоспособной программы. В настоящее время при работе с такими контроллера ми целесообразно разрабатывать программное обеспечение с использованием языков вы сокого

уровня с применением коротких ассемблерных вставок в критических местах.

7.3 Этапы создания программы

В большинстве случаев создание программ сводится к трансляции исходного кода (текста, содержащего мнемоники команд и директивы) в объектный бинарный код, и да лее к компоновке исполняемого модуля с учетом директив ассемблера и аппаратных осо бенностей микроконтроллера. Чем сложнее архитектура микроконтроллера, тем сложнее управление процессами компиляции и компоновки. Для программирования памяти мик роконтроллера обычно создается специальный файл «прошивки» в универсальном форма те (обычно hex), пригодном для обработки универсальными программаторами.

На этапе разработки и отладки программ неизбежны синтаксические ошибки, ко торые ведут к остановке трансляции с выдачей сообщений, кроме того, некоторые компи ляторы формируют предупреждения (Warnings), которые не носят аварийного характера, но обычно свидетельствуют о подозрительных обстоятельствах и нерациональных реше ниях программиста (использование переменной без инициализации, резервирования пе ременной без дальнейшего её использования и т.п.)

Отладка программного обеспечения предполагает использование специальных инструментов — отладчиков. Существует две основных формы отладки — симуляция, при которой поведение микроконтроллера имитируется на программном уровне и эмуля ция, при которой используются аппаратные средства, обеспечивающие выполнение про граммы непосредственно микроконтроллером. Симуляция имеет целый ряд ограничений, не позволяющих гарантировать безошибочное выполнение программы на «живом» про цессоре, однако, она позволяет отлаживать большинство вычислительных операций и вы явить грубые ошибки. Эмуляция требует внешних аппаратных средств, часто дорогих, она позволяет точнее приблизиться к реальным условиям работы устройства, однако, не га рантирует полного соответствия «живому» процессору. При выполнении отладки часто выгоднее экспериментировать с разрабатываемым устройством, предусматривая в про грамме диагностические возможности, например, передачу диагностической информации через какой-либо интерфейс, формирование тестовых уровней напряжения на неиспользу емых выводах микроконтроллера и т.п.

Программирование микроконтроллеров обычно осуществляется либо с помощью внешнего программатора, имеющего панель для микроконтролера с последующим пере носом запрограммированной микросхемы в панель, установленную на плате устройства, либо внутрисхемным методом, при котором микроконтроллер впаян в плату и программи руется подключением программатора к разъему, предусматриваемому для этой цели. В настоящее время второй метод получил большее распространение, прежде всего, из-за конструктивных особенностей корпусов современных микросхем. Многие современные микроконтроллеры имеют встроенный программно-аппаратный механизм записи во вну треннюю энергонезависимую память, не требуя, таким образом, специальных аппаратных устройств для программирования. Запись осуществляется через какой-либо интерфейс ми кроконтроллера под управлением встроенной программы загрузчика либо с помощью стандартного интерфейса JTAG.

В процессе разработки программного обеспечения необходимо использовать инструменты оптимизации — профилировщики, которые позволяют проанализировать код программы на предмет наличия неоптимальных конструкций, «узких мест», ограничи вающих быстродействие и т.д.

8 Технологии отладки и диагностики программного и аппаратного обеспечения

В данном разделе рассматриваются микропроцессорные системы для высокопроизводительной обработки сигналов. Одними из таких процессоров являются сигнальные процессоры семейства BLACKFIN. Они обладают встроенными ресурсами, позволяющими разработчику производить эффективную отладку разрабатываемых устройств и программных алгоритмов. Функции отладки реализованы в процессоре аппаратно и сгруппированы в несколько уровней.

Таблица 8.1 — Функции отладки программ

Функция отладки	Описание
Точки останова	Определяют диапазон адресов и условия, по совпадению которых выполняется останов процессора
История трассировки	Последние 16 значений счётчика команд до переходов сохраняются во внутреннем буфере трассировки
Счётчик тактов	Обеспечивает поддержку любых функций профилирования (profiling)
Мониторинг выполнения	Позволяет выполнять мониторинг и измерение загрузки внутренних ресурсов процессора без вмешательства в их работу

Блок точек останова обеспечивает несколько механизмов исследования по ведения программы, выполняя мониторинг адресов на шинах данных и команд. После нескольких совпадений с заданным адресом блок генерирует событие. Ин формация, обеспечиваемая блоком точек останова, помогает оптимизировать код.

При помощи этого блока также упрощается процедура корректирования кода (patching) исполняемых файлов. Блок точек останова содержит следующие реги стры, отображённые в карте памяти, которые доступны в режимах Супервизора и Эмуляции:

• регистр состояния точек останова (WPSTAT),
• шесть регистров адресов точек останова по командам (WPIA[5:0]),
• шесть регистров счётчиков точек останова по командам (WPIACNT[5:0]),
• регистр управления точками останова по командам (WPIACTL),
• два регистра адресов точек останова по данным (WPDA[1:0]),
• два регистра счётчиков точек останова по данным (WPDACNT[1:0]),
• регистр управления точками останова по данным (WPDACTL).

Точки останова по командам реализуются при помощи двух операций:

• Значения, содержащиеся в шести регистрах адресов точек останова по ко мандам (WPIA[5:0]), сравниваются с адресами на шине команд.
• При каждом совпадении декрементируются соответствующие значения счётчиков в регистрах счётчиков точек останова по командам (WPIACNT[5:0]).

Точки останова по данным реализуются при помощи двух операций:

• Значения, содержащиеся в двух регистрах адресов точек останова по дан ным (WPDA[1:0]), сравниваются с адресами на шине данных.
• При каждом совпадении декрементируются соответствующие значения счётчиков в регистрах счётчиков точек останова по данным (WPDACNT[1:0]). каждая точка останова по командам управляется тремя битами регистра WPIACTL.

Таблица 8.2 — Регистры отладки

Название бита	Описание
EMUSWx	Определяет, тип события, вызываемого совпадением адреса команды (эмуляция или исключение)
WPICNTENx	Активизирует 16-разрядный счётчик, считающий число совпадений адреса. Если счётчик отключён, событие вызывается по каждому совпадению.
WPIAENx	Активизирует функцию точки останова по адресу

События также могут вызываться комбинацией точек останова по данным и командам. Если в регистре WPIACTL установлен бит WPAND, событие вызыва ется только при совпадении и точки останова по адресу команды, и точки остано ва по адресу данных. Если бит WPAND равен нулю, событие вызывается при сов падении любой из разрешённых точек останова по адресу или диапазону.

Функция корректировки кода позволяет заменять секции существующего кода новым кодом. Регистры точек останова используются для вызова исключе ния по начальным адресам предыдущей версии кода. После этого выполняется переход по вектору программы обработки исключения к ячейке памяти, содер жащей новый код.

Функция корректировки кода в процессоре может реализовываться записью начального адреса предыдущей версии кода в один из регистров WPIAn и на стройкой соответствующего бита EMUSWx на вызов исключения. В программе обслуживания исключения выполняется чтение регистра WPSTAT для определе ния точки останова, вызвавшей исключение. Затем в регистр RETX записывается начальный адрес нового кода, и при выходе из программы обслуживания исклю чения начинает выполняться новый код. Так как для корректировки кода исполь зуется механизм исключений, невозможно исправление программ обслуживания событий одинакового или более высокого приоритета (исключения, NMI и сбро са).

Блок трассировки сохраняет историю последних 16 изменений программно го потока, выполненных программным автоматом. Это позволяет пользователю воссоздать процесс работы программного автомата. Возможно разрешение гене рации исключения по заполнению буфера трассировки. В соответствующей про грамме обслуживания исключения выполняется сохранение элементов буфера трассировки в память. Таким образом, возможно воссоздание полного пути про граммного автомата с момента активизации блока трассировки. Изменения про граммного потока, вызванные циклами с нулевыми непроизводительными затра тами, не сохраняются в буфер трассировки. При отладке кода, “зависающего” при выполнении цикла с нулевыми непроизводительными затратами, можно отслежи вать значения регистров счётчиков циклов, LC0 и LC1. Буфер трассировки можно сконфигурировать на пропуск записи изменений программного потока, совпадающих с последним или одним из двух последних элементов, содержащихся в нём. Пропуск записи значений, совпадающих с этими элементами, предотвращает переполнение буфера, вызываемое выполнением циклов в программе. Так как изменения, вызываемые циклами с нулевыми непро изводительными затратами, не сохраняются в буфер трассировки, эта опция мо жет использоваться для предотвращения переполнения, возникающего при вы полнении до четырёх вложенных циклов.

При чтении регистра буфера трассировки (TBUF) возвращается значение вершины стека блока трассировки, содержащего 16 элементов. Каждый элемент содержит пару адресов: адрес команды перехода (branch source) и адрес перехода (адрес, по которому производится переход, branch target). При чтении регистра TBUF сначала возвращается последний сохранённый элемент, начиная с адреса перехода. При следующем чтении возвращается адрес команды перехода. Значе ние количества достоверных элементов в TBUF содержится в поле TBUFCNT регистра TBUFSTAT. Значение поля TBUFCNT инкрементируется при каждом вто ром чтении TBUF. Так как каждый элемент состоит из двух слов данных, регистр

TBUF опустошается после выполнения 2 × TBUFCNT операций чтения. Два 32разрядных счетчика (регистры счетчиков монитора выполнения, PFCNTR[1:0]) считают число появлений события ядра процессора в течение периода мониторинга. Эти регистры обеспечивают способ отображения баланса загрузки различных ресурсов кристалла. С их помощью можно производить анализ и сравнение ожидаемого и действительного коэффициентов использования. Кроме того, они также позволяют отслеживать такие события, как неправильное предсказание переходов и циклы останова.
Счетчик тактов считает такты CCLK во время выполнения программы. Ко гда процессор находится в режиме Супервизора или Пользовательском режиме производится счет всех тактов, включая такты, затрачиваемые на выполнение ко манд, состояния ожидания, обработку прерываний и событий. В режиме Эмуля ции работа счетчика тактов останавливается. Счетчик тактов является 64- разрядным и инкрементируется на каждом такте. Значение счетчика хранится в двух 32-разрядных регистрах, CYCLES и CYCLES2. Регистр CYCLES содержит младшие 32 бита, регистр CYCLES2 – старшие 32 бита. Для включения счетчика тактов необходимо установить бит CCEN в регистре SYSCFG.

8.1 Разработка программ, работающих под управлением операционной системы

При создании программного обеспечения для микроконтроллеров и микро процессоров разработчику приходится затрачивать большое количество времени на реализацию функций «общего» назначения, таких как функции вводавывода, обслуживания стандартных прерываний, работа с памятью и др. Использование операционной системы позволяет сосредоточиться на реализации собственно ал горитма, оставив рутинные процессы для обработки операционной системой. При разработке программного обеспечения для процессоров BLACKFIN та кой операционной системой является VDK (Visual DSP Kernel). При использова нии VDK программист также получает преимущества, обусловленные тесной ин теграцией со средой разработки Visual DSP.

8.2 Основные понятия, необходимые для использования VDK

Все процессы, выполняемые под управлением VDK, представляют собой т.н. потоки (threads), выполняющиеся по очереди под управлением операционной системы. Фактически разработчик создает свое приложение в виде потока, вхо дящего в состав операционной системы. Поведение потока определяется путем описания типа потока, который является шаблоном, регламентирующим поведение потока и описывающим структуру данных, с которыми он работает. Каждый поток имеет свой собственный стек, собственные локальные переменные и переменные окружения. Применение потоков эффективно, когда имеется несколько наборов данных, над которыми производятся одинаковые действия или разные операции осуществляются над одним набором данных.
Операционная система осуществляет планирование выполнения потоков на основании приоритетов и распределения доступа к общим ресурсам. Каждому по току назначается один из тридцати уровней приоритета и ядро передает управле ние ожидающему потоку с наивысшим приоритетом.

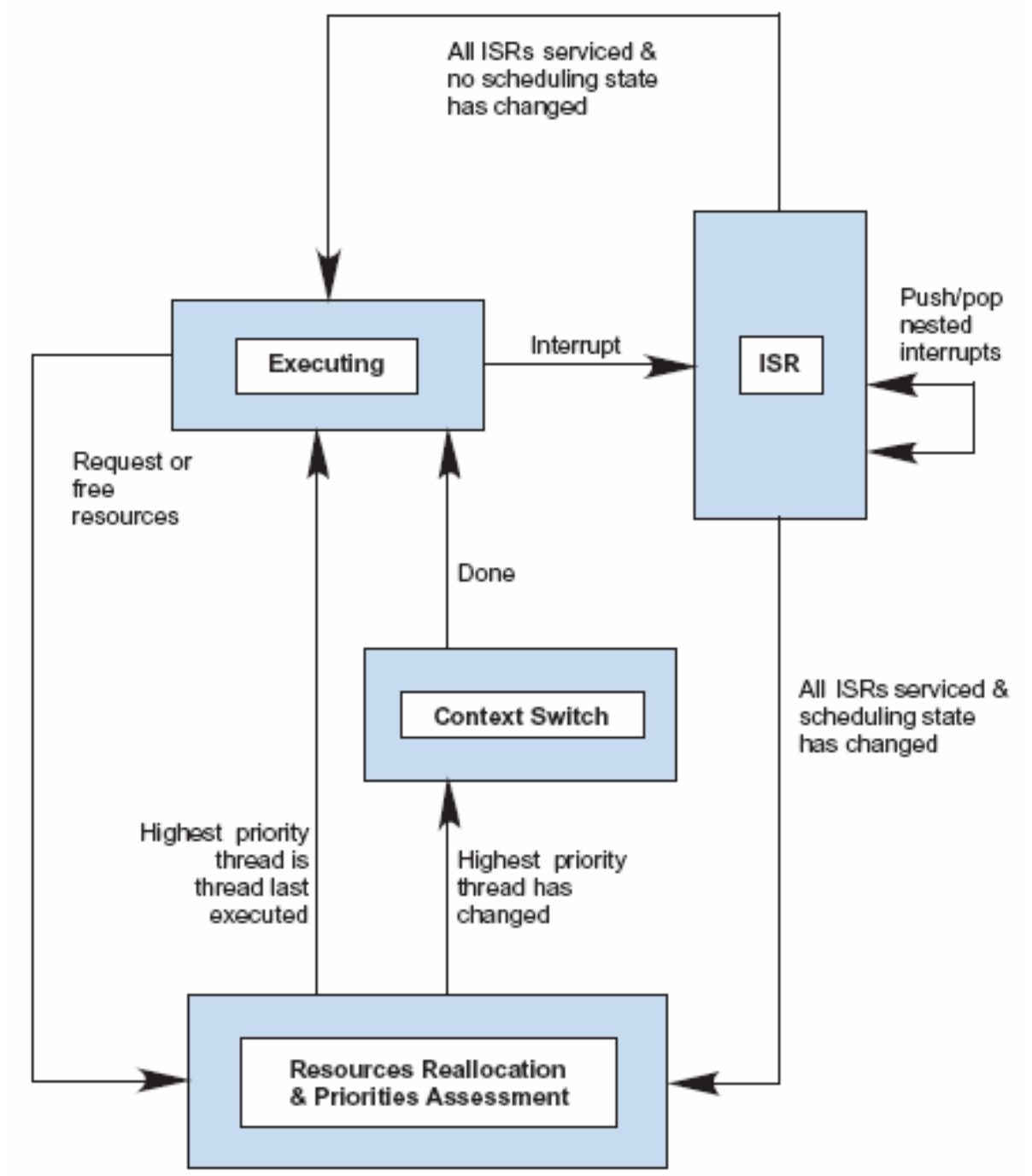


Рисунок 8.1 — Структура VDK

Выполняющийся поток продолжает выполнение до тех пор, пока он не за вершится или не запросит доступ к ресурсам системы посредством обращения к ядру. Если запрашиваемый ресурс свободен, поток продолжает выполнение, если нет, то он удаляется из очереди потоков ожидающих выполнения. Ядро не преры вает выполнения потока, пока его приоритет остается наивысшим, даже если он освобождает ресурс и в системе, таким образом появляются ожидающие потоки равного или низшего приоритета. Выполняющийся поток может быть прерван внешним событием, по завершении обработки прерывания управления будет пе редано потоку с наивысшим приоритетом. К некоторым ресурсам необходим ато марный доступ (монопольный), для реализации которого предусмотрены секции кода, исключающие планирование (unsheduled), а также критические секции. Оба

средства могут комбинироваться. Разница состоит в том, что код, выполняющий ся в критической секции, исключает возможность прерывания внешним событи ем, а код в «непланируемой» секции блокирует только переключение контекста планировщиком. Для синхронизации потоков, а также обеспечения взаимодействия между ними используются также семафоры, сообщения, события и флаги устройств. Память является системным ресурсом и операционная система поддержива ет механизм так называемых «куч», из которых потокам могут выделяться блоки по их запросу.

Операционная система обеспечивает также абстрагирование программного обеспечения от деталей аппаратной реализации, что дает возможность легко пе реносить код между различными аппаратными платформами, а также произво дить обновление аппаратной части с минимальными изменениями программного обеспечения. Концепция состоит в том, что потоки пользователя взаимодейству ют с аппаратными средствами через драйверы устройств и прерывания. Драйверы устройств и функции обслуживания прерываний тесно взаимо связаны. Как правило, функции обслуживания прерываний пишутся на языке ас семблера с тем, чтобы обеспечить максимальную эффективность в смысле скоро сти обработки асинхронных внешних событий, вместе с тем, выполняемые ими операции должны сводиться к минимуму с тем, чтобы сократить возможную за держку выполнения других важных участков кода.

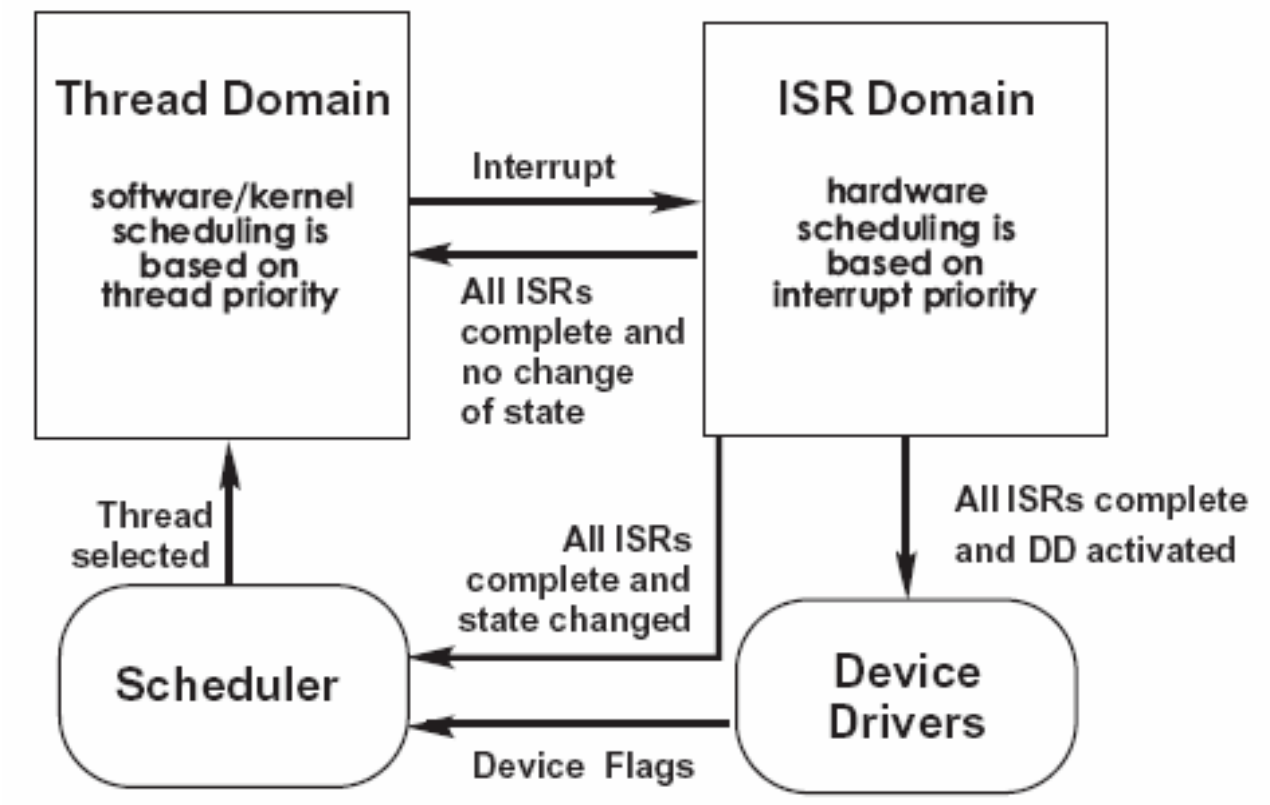


Рисунок 8.2 — Взаимодействие элементов VDK

Основной интерес вызывает процесс возвращения управления от функции обработки прерывания. Если в результате обработки прерывания схема приоритетов не изменилась, управление просто передается прерванному потоку, минуя планировщик, в противном случае планировщик может осуществить переключе ние выполняемого потока.

Драйверы устройств занимают положение, промежуточное между потоками пользовательского режима и функциями обработки прерываний. Код драйверов не затрагивается планировщиком напрямую, равно как и аппаратным контролле ром прерываний. Драйверы реализованы, как объекты C++, которые выполняются в стеке

вызывающего потока, но не принадлежат ему и могут использоваться всеми потоками. Взаимодействие с драйверами сводится к вызову ряда предопределенных функций потоками и активизации после завершения обработки прерываний (для управления флагами устройств). VDK резервирует уровень выполнения 15 для кода потоков пользователя и уровень 14 для внутреннего кода ядра, таким образом, весь код выполняется в режиме супервизора, что открывает доступ коду пользователя ко всем ресурсам системы. При этом программы обработки прерываний используют стек выполняющейся к моменту вызова прерывания программы, следовательно, он должен иметь достаточный запас пространства.

Заключение

В пособии рассматриваются вопросы, связанные с разработкой аппаратных и программных средств систем обработки информации на основе ПЛИС и микроконтроллеров. Представленный материал позволяет аспирантам подготовиться к экзамену по специальности, а также использовать знания для разработки элементов и устройств вычислительной техники и систем управления в части получения информации с выходов первичных преобразователей различных величин, построению аппаратных средств на основе микроконтроллеров и ПЛИС для их цифровой обработки и выдаче сигналов управления, написанию и отладке программ, построению распределенных систем обработки информации и управления на основе использования различных интерфейсов.

Список литературы

1. Бибило, Петр Николаевич. VHDL. Эффективное использование при проектировании цифровых систем [Текст] : рассмотрены пакеты языка VHDL: STD_ LOGIC_ 1164, NUMERIC_STD, EXEMPLAR_1164, TEXTIO, STD_LOGIC_TEXTIO, VITAL; использование ModelSim b LeonardoSpect rum / П. Н. Бибило, Н. А. Авдеев. - М. : СОЛОН-Пресс, 2006. - 341 с. - (Серия "Системы проектирования"). - ISBN 5-98003-293-2
2. Браммер, Юрий Александрович. Импульсные и цифровые устройства [Текст] : [учеб. для сред. проф. электрорадиоприборостроит. учеб. заведений] / Ю. А. Браммер, И. Н. Пащук. - Изд. 8-е, стер. - М. : Высш. шк., 2006. - 351 с. - ISBN 5-06-004354-1
3. Потехин, Дмитрий Станиславович. Разработка систем цифровой обработки сигналов на базе ПЛИС [Текст] / Д. С. Потехин, И. Е. Тарасов. - М. : Горячая линия - Телеком, 2007. - 248 с. - (Современная электроника). - ISBN 978-5-93517-341-7
4. Зотов, Валерий Юрьевич. Проектирование цифровых устройств на основе ПЛИС фирмы XILINX в САПР WebPACK ISE [Текст] / В. Ю. Зотов. - М. : Горячая линия - Телеком, 2003. - 624 с. - (Современная электроника). - ISBN 5-93517-136-8
5. Лапин, Алексей Анатольевич. Интерфейсы. Выбор и реализация [Текст] / А. Лапин. - М. : Техносфера, 2005. - 167 с. - (Мир электроники ; VII-15). - ISBN 5-94836-058-X
6. Бойт, Клаус.Цифровая электроника [Текст] / К. Бойт ; пер. с нем. М. М. Ташлицкого. - М. : Техносфера, 2007. - 471 с. - (Мир электроники ; VII-31). - ISBN 978-5-94836-124-6
7. Новожилов, Олег Петрович Основы микропроцессорной техники [Текст] :учеб. пособие : в 2 т. / О. П. Новожилов. - М. : РадиоСофт, 2007 - .Т. 1. - 2007. - 431 с. - ISBN 5-93037-165-2
8. Нарышкин, Александр Кириллович. Цифровые устройства и микропроцессоры [Текст] : [учеб. пособие для радиотехн. специальностей] / А. К. Нарышкин. - М. : Academia, 2006. - 318 с. - (Высшее профессиональное образование. Радиоэлектроника). - ISBN 5-7695-161- 5 экз.
9. Солонина, Алла И. Алгоритмы и процессоры цифровой обработки сигналов [Текст] : учеб. пособие для студентов, обуч. по направлению 654400 "Телекоммуникации" / А. И. Солонина, Д. А. Улахович, Л. А. Яковлев. - СПб. : БХВ-Петербург, 2002. - 454 с. - ISBN
10. Сергиенко, Александр Борисович. Цифровая обработка сигналов [Текст] : учеб. пособие для вузов по направлению подгот. дипломир. специалистов "Информатика и вычисл. техника" / А. Б. Сергиенко. - М. : Питер, 2003. - 603 с. - (Учебник для вузов). - ISBN