VI. Организация шин

Совокупность устройств и сигнальных линий, обеспечивающих взаимосвязь всех частей ЭВМ образуют систему шин.

Состав системы:

- Сигнальные линии
- Устройства арбитража
- Разъемы

Для описания (спецификации) конкретной шины необходимо описать:

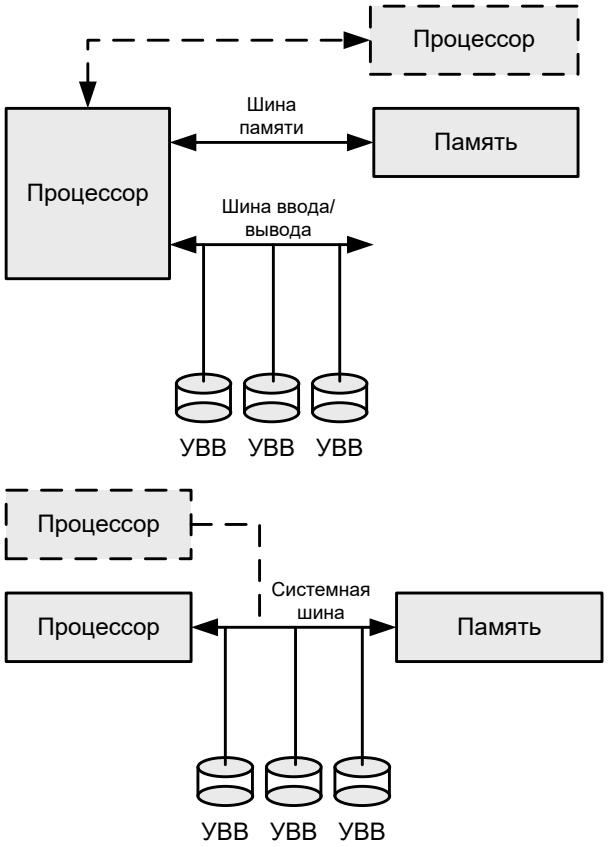
- Совокупность линий (сигналов) и их назначение.
- Протокол: правила взаимодействия устройств с помощью шины (диаграммы, алгоритмы, автоматы).
- Физические, механические и электрические параметры шины и подключаемых устройств (частота, уровни сигналов, способ согласования волновых сопротивлений, длины линий, характеристики разъемов и т.д.).

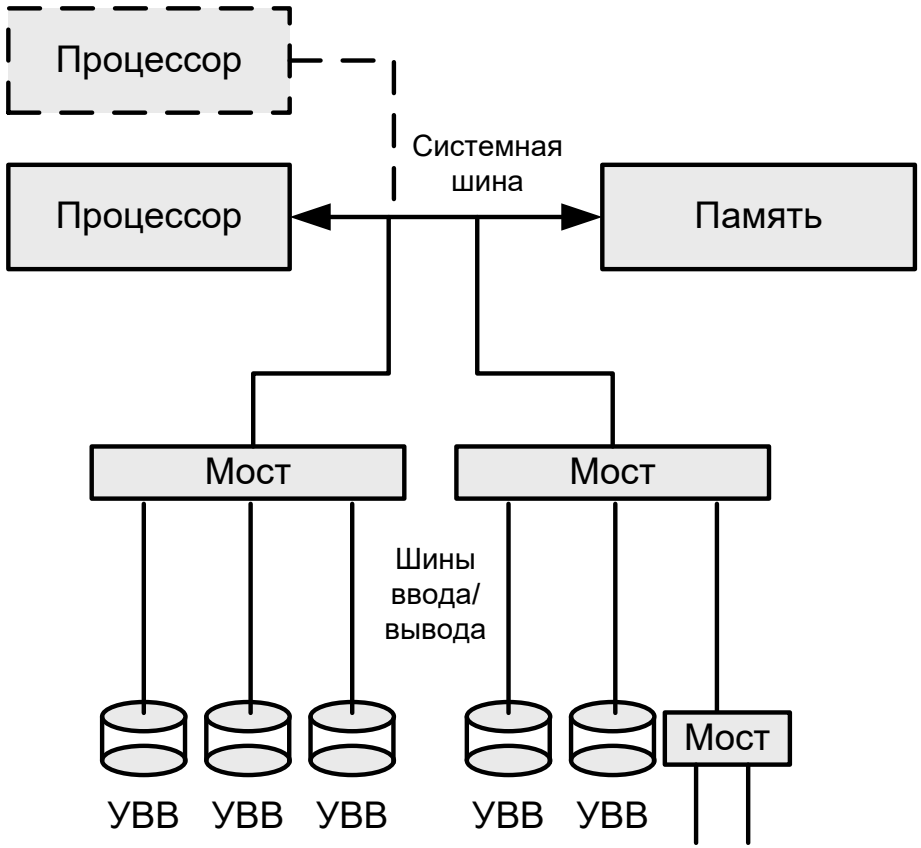
Типы шин:

-Шины «процессор-память», «процессор-кэш», «процессор-процессор» (HyperTrensport, P6 Back-Side Bus, …)

-Шины ввода-вывода (PCI, SCSI, PCI-X, IDE, GX, …)

-Системные шины (EISA, Pentium 4 FSB, Multibus II, NuBus, …)

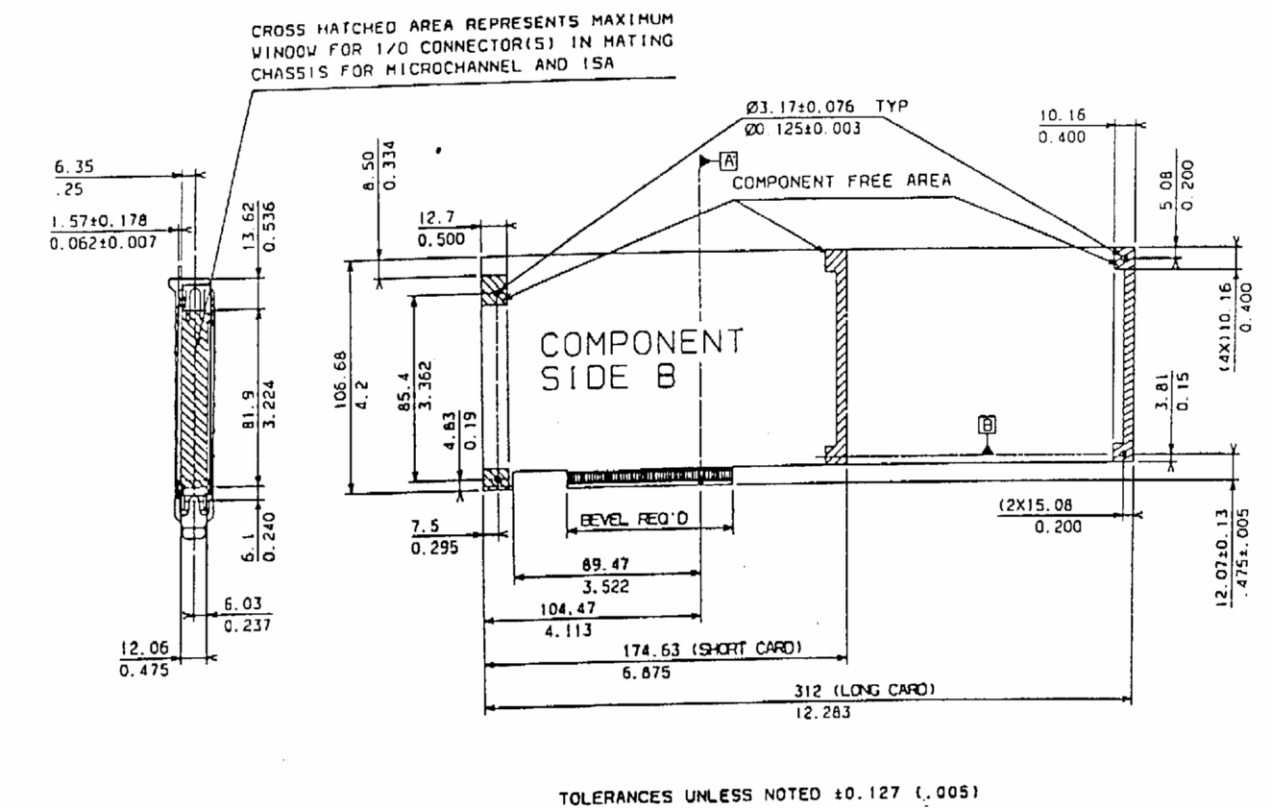




Механические аспекты спецификации шин

Механические аспекты спецификации шин включают описание вариантов исполнения, чертежи разъемов, размеры и допуски печатных плат, описание направляющих и ключей, описание способов испытаний и т.д.

ПРИМЕР



Электрические аспекты спецификации шин

- Спецификация динамических и статических характеристик
- Спецификация синхронизации
- Спецификация инициализации
- Спецификация нагрузки
- Спецификация питания
- Назначения контактов разъема

Для каждого сигнала должны бать определены параметры:

- Уровни сигналов логического «0» и логической «1» (ТТЛ, ЭСЛ, КМОП и др.).
- Время переключения
- Моменты фиксации состояния относительно сигнала синхронизации
- Условия перевода в третье состояние
- Способ согласования.

При распространении сигналов по параллельным линиям необходимо учитывать:

- Скорость распространения (обычно составляет ~70% скорости света 0.7*300 мм/нс).
- Отражение сигнала
- Перекос сигналов
- Перекрестные помехи

Типы сигнальных линий:

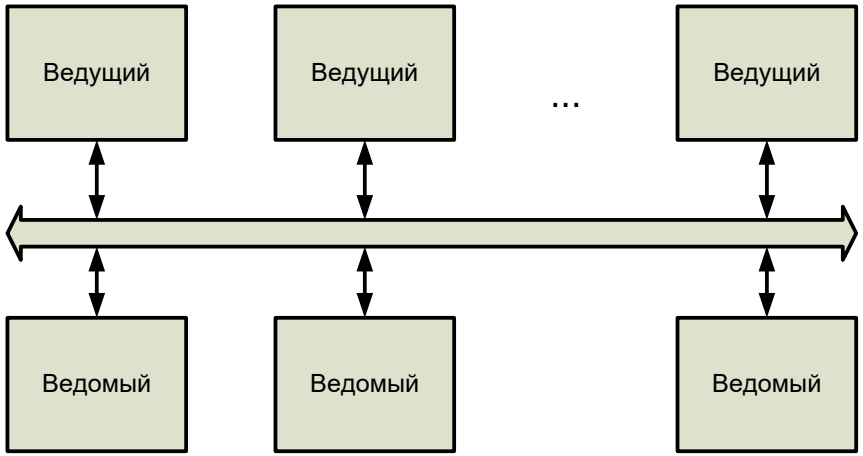
-Линии адреса

-Линии данных (линии адреса и данных могут объединяться).

-Линии управления для передачи типа транзакции, статуса устройства, сигналов арбитража, запросов прерываний, резервных линий, линий константных значений.

Арбитраж шин

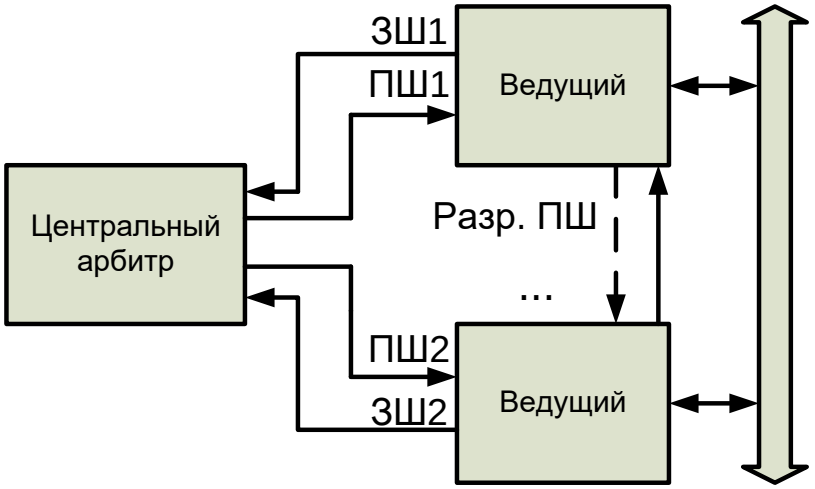
Ведущие устройства используют шину в разное время и должны отдавать и захватывать ее. Для определения очередности подключения ведущих устройств и учета их приоритетности используются: статические приоритеты, динамические приоритеты.



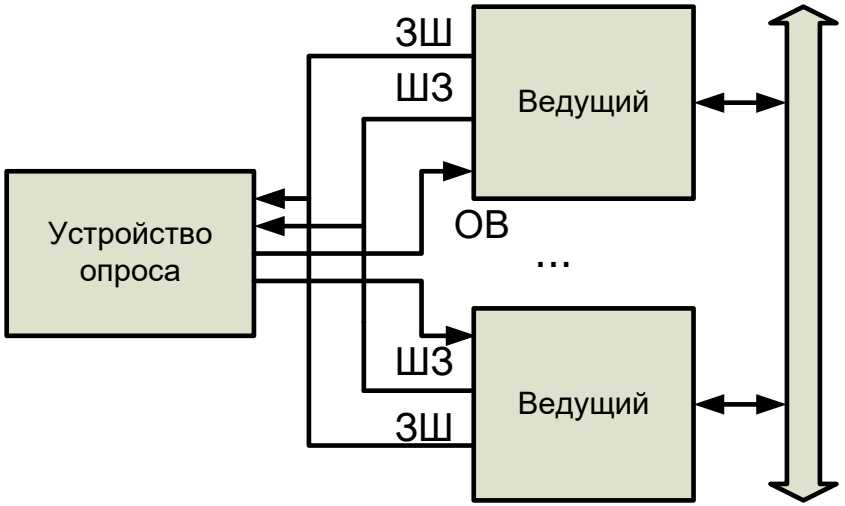
По способу арбитража: централизованный арбитраж, децентрализованный арбитраж.
Схемы арбитража: циклическая схема, циклическая с учетом последнего, рандомизированный, схема с равными приоритетами, схема LRU

Централизованный арбитраж

Параллельный арбитраж

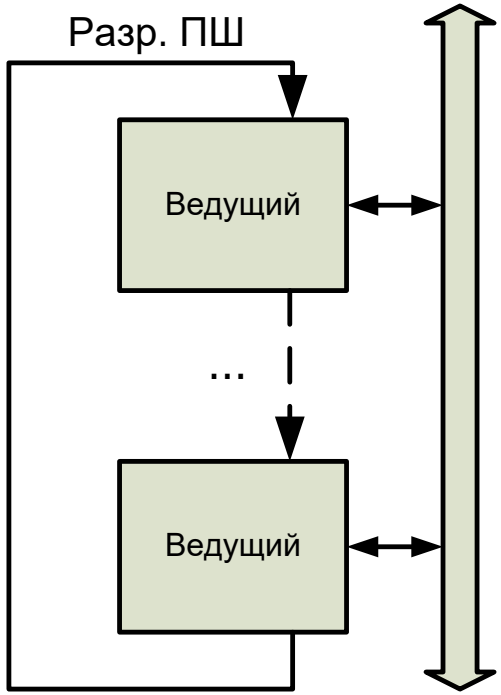


Централизованный опрос

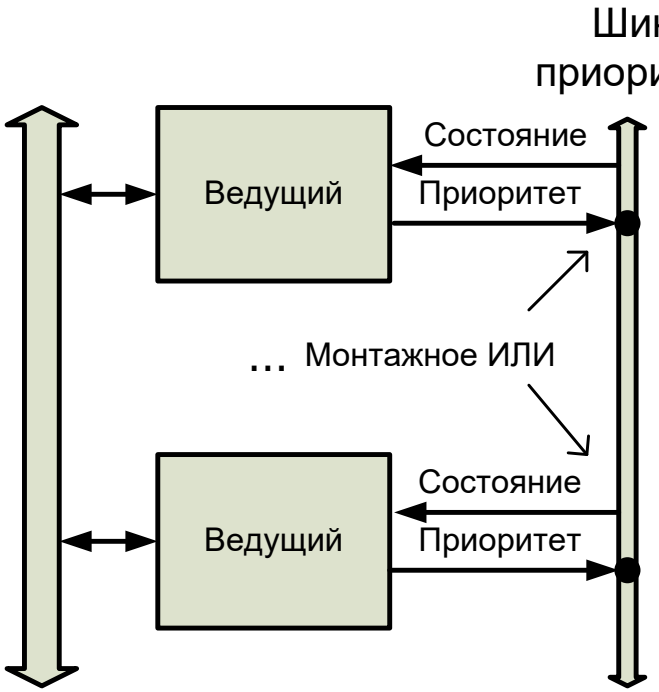


Децентрализованный арбитраж

Цепочечная схема без централизованного арбитража



Распределенный арбитраж



В1	1	0
В2	0	1
ШУ	1	1

⇨

В1	1	0
В2	0	0
ШУ	1	0

Захват шины В1

Системная шина процессоров P6

Частота:66,100,133.

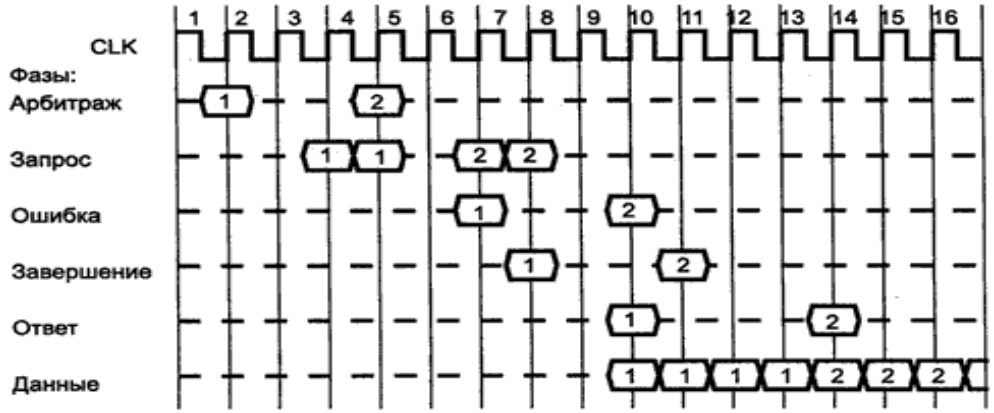
Количество абонентов: 16

Разрядность адреса: 32

Разрядость данных: 64

Контроль информации: PE и ECC

Поддержка операций с кэш



Пакетная и транзакционная передача

Каждое устройство-агент, подключенное к этой шине (например, любой из процессоров), до инициализации запроса должно получить через механизм арбитража право на использование шины запроса. Запрос выходит за два смежных такта: в первом такте передается адрес, тип обращения (чтение-запись памяти или ввода/вывода) и тому подобная информация. Во втором такте передается уникальный идентификатор транзакции, длина запроса, разрешенные байты шины и т. п. Через три такта после запроса проверяется состояние ошибки (error status) для защиты от ошибок передачи или нарушений протокола.

Любая обнаруженная ошибка вызывает повтор запроса, а вторая ошибка для того же запроса вызывает исключение контроля (machine check exception).

Описание сигналов системной шины

ПРИМЕР

Сигнал	I/O	Назначение
A[35:3]#	I/O	Address - сигналы шины адреса. Когда сигнал AdS# активен, на шине присутствует адрес, когда пассивен - информация о типе транзакции. По окончании действия сигнала Reset# процессор с шины адреса получает конфигурационную информацию.
A20M#	I	A20 Mask - маскирование бита A20 физического адреса для эмуляции адресного пространства 8086 в реальном режиме (его использование в защищенном режиме приведет к непредсказуемым результатам). Во время действия сигнала Reset# используется для конфигурирования умножителя частоты.
AdS#	I/O	Address Strobe - строб адреса, вводимый инициатором обмена как индикатор действительности адреса. По этому сигналу все агенты шины начинают проверять проверку паритета и протокола, декодирование адреса, внутреннее слежение и другие операции, связанные с новой транзакцией.
AErr#	I/O	Address Parity Error - ошибка паритета на шине адреса. В зависимости от конфигурирования по включению питания сигнал может приводить к аварийному прекращению транзакции.
AP[1:0]#	I/O	Address Parity - биты паритета шины адреса. AP1# относится к AP[35:24]#, AP0# - к A[23:3]#. Сигнал корректного паритетат должен иметь низкий уровень, если низкий уровень имеет нечетное количество контролируемых им линий.
BClk	I	Bus Clock - синхронизация шины. Значения всех синхронных сигналов действительны по положительному перепаду этого сигнала.

ПРИМЕР

BErr#	I/O	Bus Error - неисправимая ошибка шины.
BInit#	I/O	Bus Initialization - инициализация шины. Если при конфигурировании использование сигнала разрешено, то он вызывает прерывание текущей транзакции с потерей данных и сброс в исходное состояние управляющих автоматов всех агентов шины и их циклических идентификаторов арбитража.
BNR#	I/O	Block Next Request - запрос блокировки следующей транзакции. Вводится любым агентом шины как запрос на приостановку, когда он не может воспринять следующую транзакцию.
BP[3:2]#	I/O	Breakpoint - сигналы от процессоров, указывающие на попадание в точку останова.
BPM[1:0]#	I/O	Breakpoint Monitor - сигналы от процессоров, указывающие на попадание в точку останова или срабатывание счетчиков. используемых для мониторинга производительности процессора.
BPri#	I	Bus priority Request - сигнал, используемый для арбитража запросов на владение шиной.
BR0#	I/O	Bus Request - запрос шины.
BR[3:1]#[1]	I	Эти сигналы процессоры соединяются с линиями шины BReq[3:0]#
BSel#	O	Bus Select - Задает частоту системной шины процессора.

BSel1#	BSel0#	Частота системной шины, МГц
0	0	66
0	1	100
1	0	-
1	1	133

ПРИМЕР

CPUPres#	O	CPU Present - признак наличия процессора в сокете.
D[63:0]#	I/O	Data - сигналы шины данных. Источник данных при передаче указывает на их действительность сигналом DRdy#.
DBsy#	I/O	Data Bus Busy - шина данных занята. Используется агентом, передающим данные, для указания на занятость шины данных
DEFER#	I	Сигнал, указывающий на то, что исходный порядок выполнения транзакций не гарантируется.
DEP[7:0]#	I/O	Data Bus ECC Protection - дополнительные сигналы защиты шины данных ECC-кодом.
DRdy#	I/O	Data Ready - готовность данных. Устанавливается источником данных для указания на присутствие достоверных данных на шине.
FErr#	O	Floating-point Error - ошибка FPU.
Flush#	I	Асинхронный сигнал на очистку внутреннего кэша. По этому сигналу выполняются все обратные записи и аннулируются строки кэша обоих уровней. Новые строки не выделяются до окончания действия сигнала. Значение сигнала во время окончания действия сигнала Reset# используется для конфигурирования процессора.
FRCErr#	I/O	Functional Redundancy Checking Error - сигнал ошибки, обнаруженной проверочным процессором в функционально-избыточной системе.
Hit#,HitM#	I/O	Сигналы результатов операции слежения за транзакцией. Hit# (Snopp hit) указывает на кэш-попадание. HitM#(Hit Modified) указывает на попадание в модифицированную строку, запрещая другим контроллерам шины обращаться к этим данным до выполнения обратной записи.

ПРИМЕР

IErr#	O	Internal Error - сигнал обнаружения внутренней ошибки. Обычно появляется вместе с транзакцией Shutdown. Сигнал ошибки сохраняется до его программного сброса обработчиком NMI или аппаратного сброса сигналами Reset#, BInit# или Init#
IgnNE#	I	Ignore Numeric Error - игнорирование ошибки сопроцессора - запрет вырабатывать исключения. Используется для совместимости с AT, где вместо исключения вырабатывается аппаратное прерывание. Во время действия сигнала Reset# используется для конфигурирования умножителя частоты.
Init#	I	Initialization - "Мягкая" инициализация процессора. Сигнал приводит к сбросу общих регистров и переходу по вектору, заданному при конфигурировании по включению. Содержимое кэш-памяти, буферов записи и регистров FPU не затрагивается. Если сигнал активен во время окончания действия сигнала Reset#, процессор выполняет BIST
LInt[1:0] (NMI,IntR)	I	Local APIC Interrupt - входы прерываний локальных контроллеров APIC. Если работа APIC запрещена, LInt0 становится сигналом IntR, LInt1 - сигналом NMI. По сигналу Reset# работа APIC разрешается и входы работают в режиме APIC, который может быть отменен программно. Во время действия сигнала Reset# используются для конфигурирования умножителя частоты.
Lock#	I/O	Блокировка шины на время транзакции.
PICClk	I	APIC Clock - синхронизация шины APIC. В режиме FRC частота должна быть равной 1/4 BClk.
PICD[1:0]	I/O	APIC Data - двунаправленная последовательная шины обмена сообщениями APIC.

ПРИМЕР

PRdy#	O	Probe Ready - сигнал готовности зонда, используемый аппаратными средствами отладки. Указывает на остановку нормального исполнения в ответ на сигнал R/S#(вход в зондовый режим).
Preq#	I	Probe Request - запрос зонда на отладочную операцию.
PwrGood	I	Power Good - сигнал исправности питания, указывающий на стабильность питающих напряжений и сигнала синхронизации.
Req[4:0]#	I/O	Request Command - запрос команды. Вводится текущим владельцем шины для определения типа активной транзакции.
Reset#	I	Сброс процессора - конфигурирование процессора, инициализация регистров, очистка кэша обоих уровней (без выполнения обратной записи) и переход к вектору сброса (по умолчанию 0FFFFFFF0h).
RP#	I/O	Request Parity - бит паритета запроса, защищающий линии AdS# и Req[4:0]#. Сигнал корректного паритета должен иметь низкий уровень, если низкий уровень имеет нечетное количество контролируемых им линий.
RS[2:0]#	I	Response Status - состояние ответчика. Сигналы управляются агентом, отвечающим за завершение текущей транзакции.
RsP#	I	Response Parity - бит паритета для сигналов RS[2:0]#.
SlotOcc#[2]	O	Slot Occuped - слот занят. Низкий уровень сигнала свидетельствует о наличии процессора или терминатора в слоте Pentium-2. Используется с комбинацией сигналов VID[4:0] для определения наличия процессорного ядра в слоте: при VID[4:0]=11111 в слоте терминатор.

Slp#[2]	I	Sleep - сигнал, переводящий процессор из состояния Stop Grant в состояние Sleep (спящий режим).
SMI#	I	System Management Interrupt - сигнал прерывания для входа в режим SMM.
StpClk#	I	Stop Clock - асинхронный сигнал, переводящий процессор в состояние Stop Grant с малым потреблением .
TCk	I	Test Clock - вход синхронизации шины тестирования Test Bus, называемой также TAP (Test Access Port).
TDI	I	Test Data In - последовательный вход данных интерфейса JTAG.
TDO	O	Test Data Out - последовательный выход данных интерфейса JTAG.
TestHi	I	Сигнал, подключаемый к источнику 2,5 В через резистор 1-10 кОм.
TestLo	I	Сигнал, подключаемый к шине GND.
ThermTrip#	O	Thermal Trip - сигнал останова процессора по перегреву. Если внутренняя температура поднимается примерно до 130°C, процессор останавливается и формирует данный сигнал, который может быть сброшен только сигналом Reset# после снижения температуры ниже этого порога.
TMS	I	Test Mode State - выбор режима тестирования по JTAG.
TRdy#	I	Target Ready - сигнал, которым целевое устройство указывает на готовность приема данных к записи или неявной обратной записи.
TRst#	I	Test Reset - сигнал сброса логики TAP (самосброс происходит автоматически по включению).
VID[4:0]	O	Voltage ID - выводы идентификации для автоматической установки уровня питающего напряжения. Эти выводы могут либо быть свободными, либо подключаться к шине GND. Блок питания должен установить соответствующее напряжение, либо отключиться.
UP#	O	Upgrade Present - признак установки в сокет 8 процессора OverDrive.

ПРИМЕР

Интерфейс CAN

CAN (Controller Area Network — сеть контроллеров) — стандарт промышленной сети, ориентированный прежде всего на объединение в единую сеть различных исполнительных устройств и датчиков. Режим передачи — последовательный, широковещательный, пакетный. Арбитраж — децентрализованный.

CAN является синхронной шиной с типом доступа Collision Resolving (CR, разрешение коллизии), который в отличие от Collision Detect (CD, обнаружение коллизии) сетей (Ethernet) детерминировано (приоритетно) обеспечивает доступ на передачу сообщения, что особо ценно для промышленных сетей управления (fieldbus). Передача ведётся кадрами. Полезная информация в кадре состоит из идентификатора длиной 11 бит (стандартный формат) или 29 бит (расширенный формат, надмножество предыдущего) и поля данных длиной от 0 до 8 байт. Идентификатор говорит о содержимом пакета и служит для определения приоритета при попытке одновременной передачи несколькими сетевыми узлами.

Модель взаимодействия открытых систем ISO OSI

Уровни взаимодействия по протоколу CAN

Для CAN протокола можно выделить два основных уровня взаимодействия устройств:

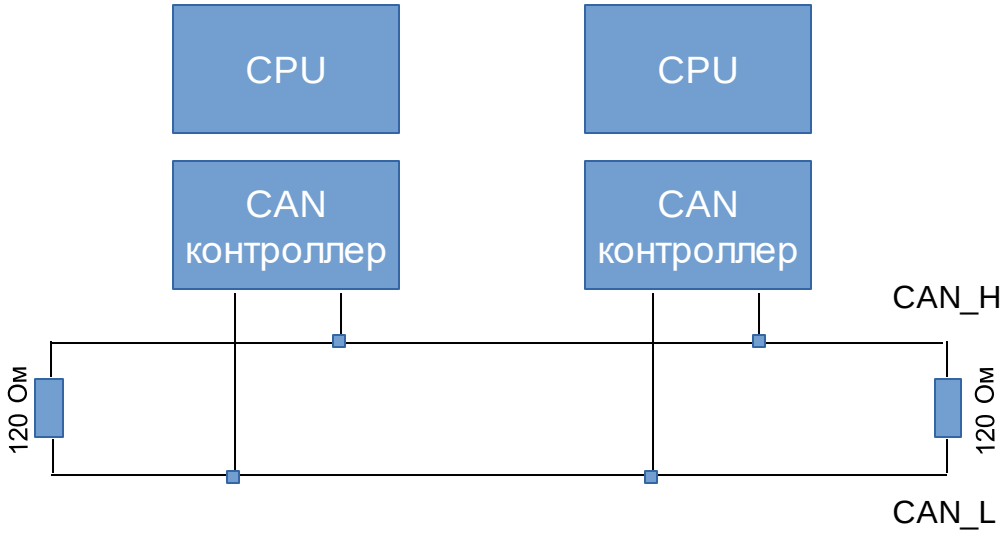
- Физический.
- Канальный.

Сетевой уровень CAN не входит в стандарт и существует в нескольких несовместимых вариантах (CANopen, DeviceNet, SDS, CAN Kingdom и др.), в связи с чем в данной работе рассматриваться не будет. На физическом уровне протокола описываются аспекты передачи бит информации: варианты организации несущей среды, средства сопряжения устройств с шиной, способ кодирования информации, устранение коллизий, низкоуровневый арбитраж. Для канального уровня описываются такие аспекты взаимодействия, как: форматы кадров, процедура передачи сообщений, адресацию, организацию приоритетов, конфигурирование, контроль ошибок и другие.

Основы CAN протокола

Доступ к шине

В CAN используется множественный доступ с опросом несущей и разрешением конфликтов. Каждый узел сети должен контролировать бездействие шины в течение некоторого времени, прежде чем послать сообщение. Как только обнаружено бездействие шины, все узлы сети имеют равную возможность передать данные.



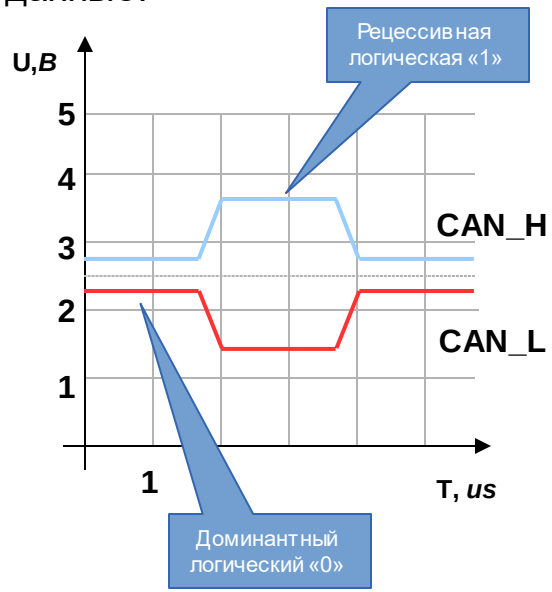
Подключенные к шине модули взаимодействуют через сеть, состоящую из двух линий, по которым передается синфазный код: CAN_H (позитивный сигнал) и CAN_L (негативный сигнал). На концах линий должны находиться резисторы 120 Ом для согласования волнового сопротивления кабеля. Скорость передачи может достигать 1 Мбит в секунду, а длина линии: 1 км. При этом все устройства подключены к шине и могут одновременно начать передачу. В результате этого возможно возникновение коллизий, которые разрешается благодаря конструкции приёмопередатчиков на физического уровня и специальными форматами кадров на канальном уровне.

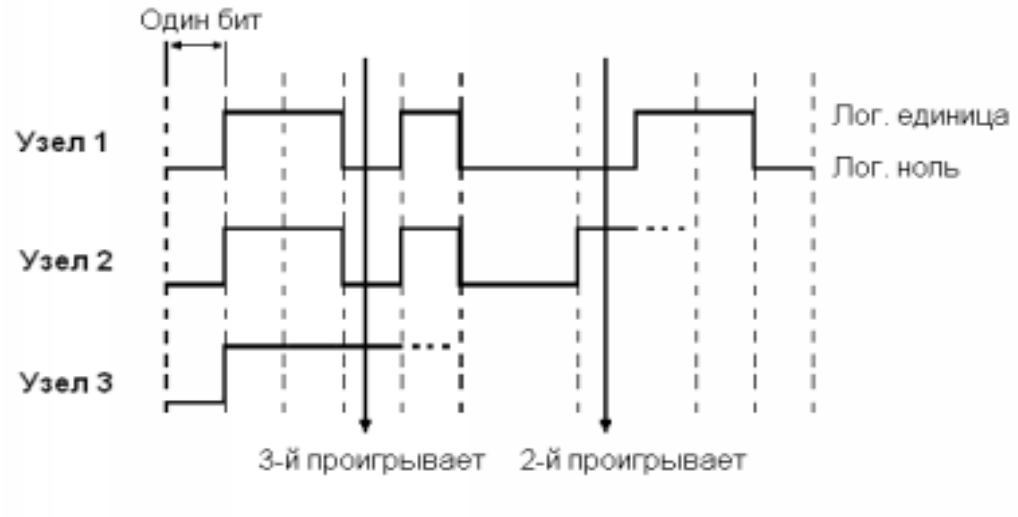
Основы CAN протокола

Арбитраж

В CAN используется неразрушающий поразрядный арбитраж на шине, сохраняющий сообщение не поврежденным при потере инициативы. Арбитраж позволяет передавать сообщения с высоким приоритетом, без каких-либо задержек и искажений. Логические уровни на шине определяются как ‘dominant’ и ‘recessive’. Узел передачи должен контролировать логическое состояние шины для начала передачи данных. На шине CAN логический бит 0 определяется как ‘dominant’, а бит – 1 как ‘recessive’.

Бит ‘dominant’ всегда будет выигрывать арбитраж у ‘recessive’ бита, поскольку имеет низкий уровень сигнала и более высокий приоритет. Каждый узел, передающий данные на шины, должен контролировать наличие передаваемого бита на шине. Сообщение с более низким приоритетом будет формировать на шине бит ‘recessive’, а при проверке обнаружит бит ‘dominant’, в этом случае узел, формирующий сообщение более низкого приоритета, теряет арбитраж. Узел, потерявший арбитраж, должен дождаться отсутствия активности на шине и повторить попытку передать данные.





Основы CAN протокола

Сообщения

В CAN протоколе сообщения не являются адресными, т.е. сообщения не адресуются от одного узла к другому. Сообщение содержит идентификатор источника и собственно данные. Все узлы CAN сети могут принять каждое сообщение на шине и самостоятельно определить: данное сообщение должно быть отвергнуто или обработано.

Другой полезной особенностью CAN протокола является возможность удаленного запроса данных (RTR). В отличие от предыдущего случая, требуемые данные не ожидаются, когда появятся на шине, а запрашиваются у конкретного узла. Проектировщик может использовать эту особенность для снижения трафика шины при сохранении целостности сети. Добавление нового узла в систему не требует перенастройки остальных устройств сети. Новый узел начинает принимать сообщения из сети на основе их идентификаторов.

По протоколу предусмотрено четыре вида кадров:

- **Кадр данных (data frame)** — используется при передаче данных. Размер передаваемой полезной информации может варьироваться в каждом кадре, так как в структуре заголовка кадра предусмотрено поле длины. Максимальный размер данных: 8 байт.
- **Кадр удаленного запроса (remote frame)** — служит для запроса на передачу кадра данных с тем же идентификатором. Кадр запроса позволяет на прикладном уровне строить более сложную логику взаимодействия «запрос-ответ».
- **Кадр перегрузки (overload frame)** — обеспечивает промежуток между кадрами данных или запроса. Данный кадр необходим для перевода приемо-передатчиков в исходное состояние.
- **Кадр ошибки (error frame)** — передаётся узлом, обнаружившим в сети ошибку. Данный кадр использован в механизме контроля ошибок протокола CAN.

Форматы кадра данных

Базовый кадр		Описание	Расширенный кадр	
Название поля	Длина (в битах)		Название поля	Длина (в битах)
Начало кадра	1	Сигнализирует начало передачи кадра	Начало кадра	1
Идентификатор	11	Уникальный идентификатор	Идентификатор A	11
Запрос на передачу (RTR)	1	Должен быть доминантным	–	–
–	–	Должен быть рецессивным. Используется для тестирования несущей	Подмена запроса на передачу (SRR)	1
Бит расширения идентификатора (IDE)	1	Должен быть доминантным (определяет длину идентификатора)	–	–
–	–	Должен быть рецессивным (определяет длину идентификатора)	Бит расширения идентификатора (IDE)	1
–	–	Вторая часть идентификатора	Идентификатор B	18
–	–	Должен быть доминантным	Запрос на передачу (RTR)	1
Зарезервированный бит (r0)	1	Резерв	Зарезервированные биты (r1 и r0)	2
Длина данных (DLC)	4	Длина поля данных в байтах (0-8)	Длина данных (DLC)	4
Поле данных	0-8 байт	Передаваемые данные (длина в поле DLC)	Поле данных	0-8 байт
Контрольная сумма (CRC)	15	Контрольная сумма всего кадра	Контрольная сумма (CRC)	15
Разграничитель контрольной суммы	1	Должен быть рецессивным	Разграничитель контрольной суммы	1
Промежуток подтверждения (ACK)	1	Передатчик шлёт рецессивный, приёмник вставляет доминанту	Промежуток подтверждения (ACK)	1
Разграничитель подтверждения	1	Должен быть рецессивным	Разграничитель подтверждения	1
Конец кадра (EOF)	7	Должен быть рецессивным	Конец кадра (EOF)	7

Контроль шины

Все передающие устройства должны использовать дополняющие биты (bit-stuffing), что гарантирует появление бита противоположного значение в шести битах шины.

Кадры не должны передаваться на шину одновременно, т.е. никакой контроллер не должен начинать передачу, если на шине передается другой кадр.

Если два или более устройств начинают передавать кадры на шину одновременно, при передаче идентификатора обнаруживается, что переданный устройством с меньшим приоритетом идентификатор не соответствует состоянию шины и вынуждены прекратить передачу кадра.

Пример:
10001111110011010011000001100111110
10001111101001101001100000101100111100

Контроль ошибок

- **Контроль передающим устройством**: приемник активного приемо-передатчика сравнивает битовые уровни в сети с передаваемыми битами.
- **Дополняющие биты (bit stuffing):** после передачи пяти одинаковых битов подряд автоматически передаётся бит противоположного значения. Таким образом кодируются все поля кадров данных или запроса, кроме разграничителя контрольной суммы, промежутка подтверждения и EOF.
- **Контрольная сумма:** передатчик вычисляет её и добавляет в передаваемый кадр, приёмник считает контрольную сумму принимаемого кадра в реальном времени (одновременно с передатчиком), сравнивает с суммой в самом кадре и в случае совпадения передаёт доминантный бит в промежутке подтверждения.
- **Контроль значений** полей при приёме.
- **Контроль правильности приема сообщения** передатчиком по биту ACK (подтверждение приема). В случае неподтвержденного приема сообщения, CAN передатчик должен повторить посылку сообщения вплоть до его подтверждения. В связи с этим CAN контроллеры имеют встроенные счетчики ошибок, позволяющие остановить передачу при превышении количества ошибок заранее установленного предела.
- Разработчики оценивают вероятность невыявления ошибки передачи как $4{,}7 \times 10^{-11}$.

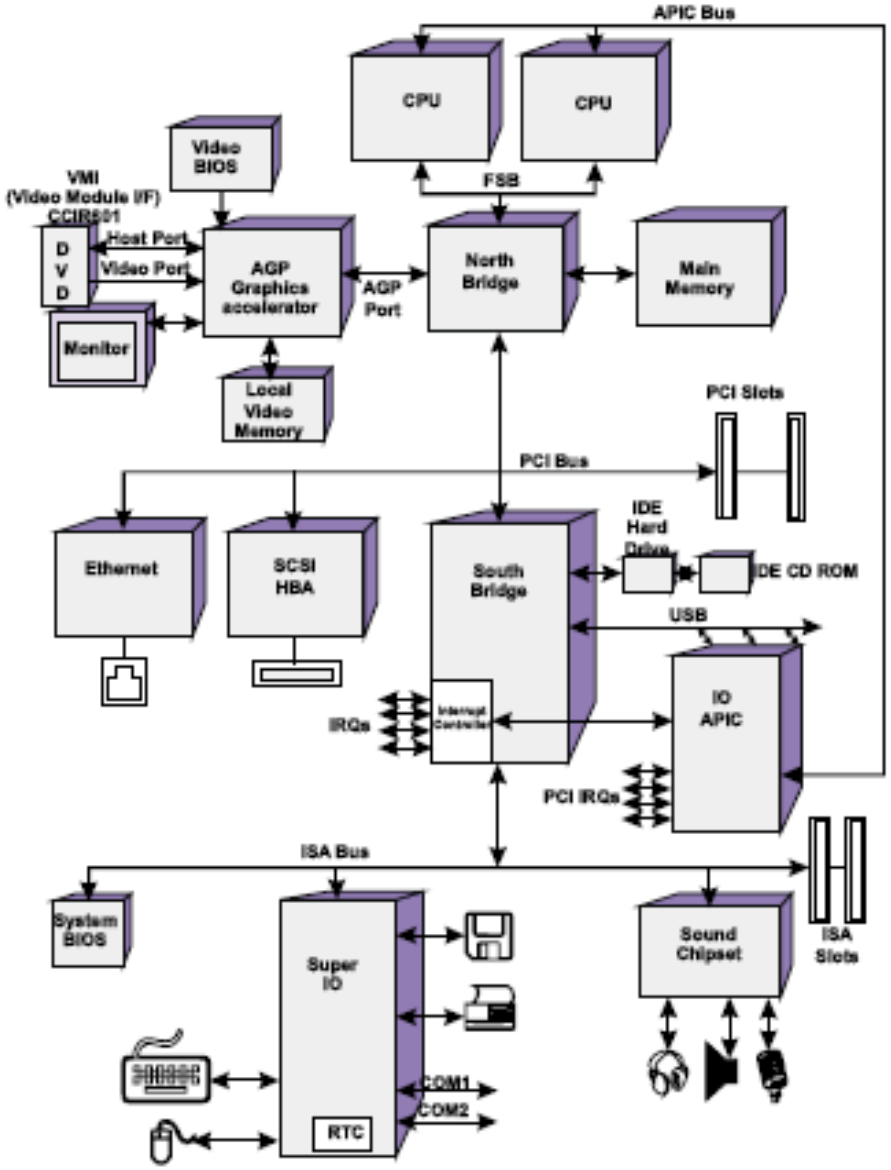
Шина PCI

Частота:33,66.
Количество абонентов: 21
Разрядность адреса/данных: 32,64 (132...528 МБ/сек)
Контроль информации: по четности
Поддержка операций с кэш
Пакетная передача
Поддержка иерархии шин (до 256)
Plug and Play технология.
Поддержка многих управителей шины (Masters) и ведомых (Targets)/
Спецификации: 1.0 (1992),2.0(1993),2.1(1995), 2.2(1999)

Локальная шина PCI - это высокопроизводительная 32-битная или 64-битная шина с мультиплексированными линиями адреса и данных. Она предназначена для использования в качестве связующего механизма между высокоинтегрированными периферийными контроллерами ввода-вывода, периферийными встраиваемыми платами и системами процессор/память.

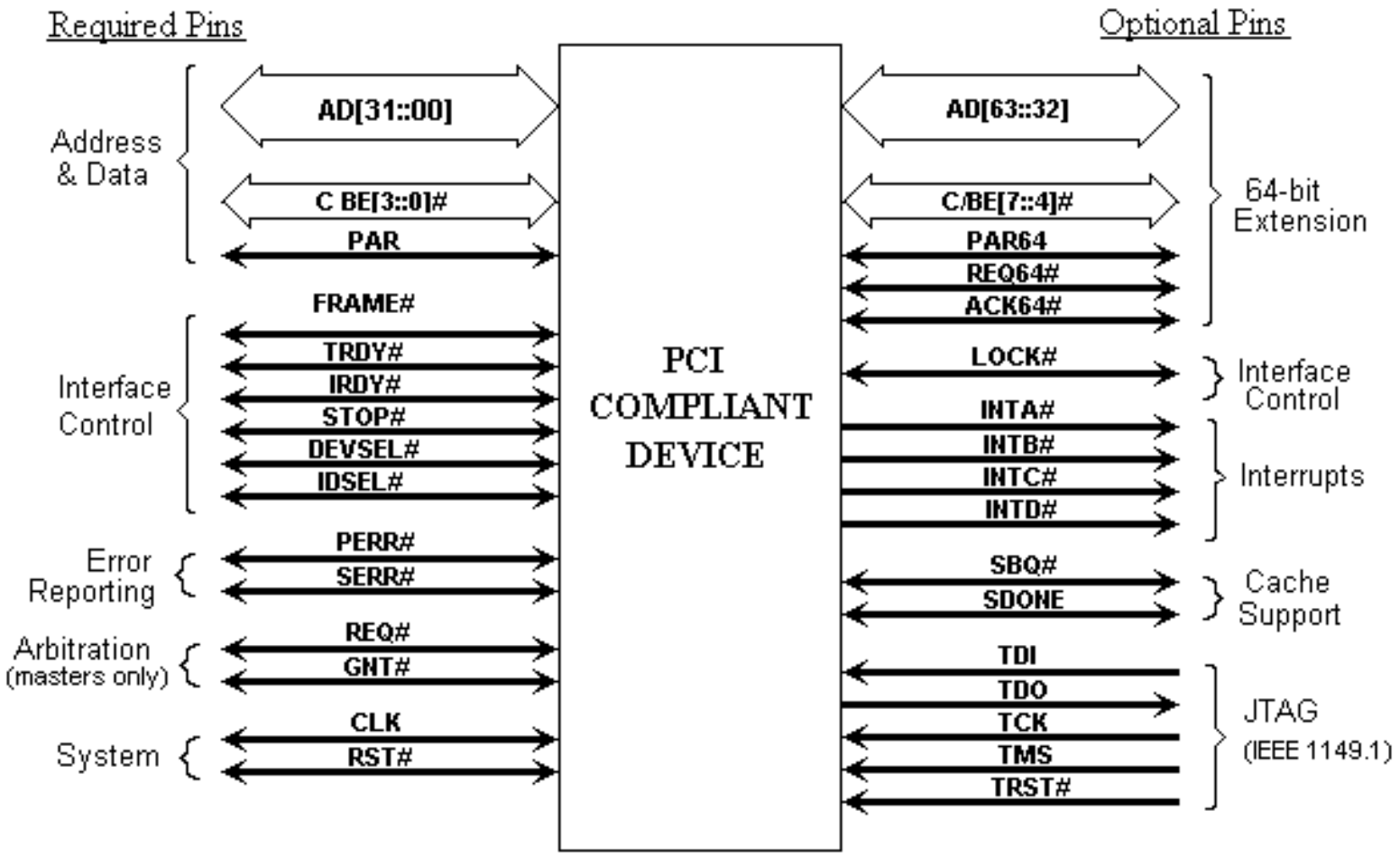
Спецификация локальной шины PCI, реализация 2.0, включает протокол, электрическую, механическую и конфигурационную спецификации для локальной шины PCI и плат расширения. Описания электрических сигналов приводятся для напряжений питания 3.3В и 5.0В.

Пример построения систем на основе шины PCI*

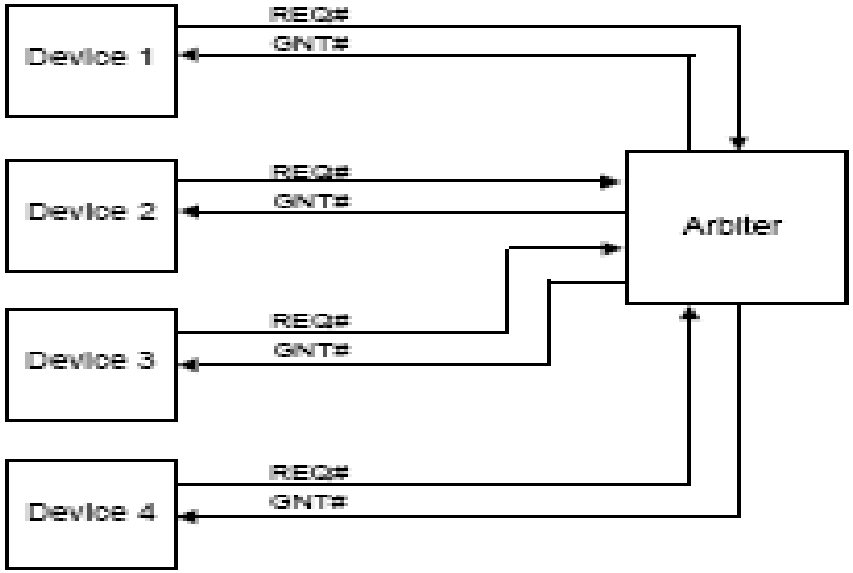


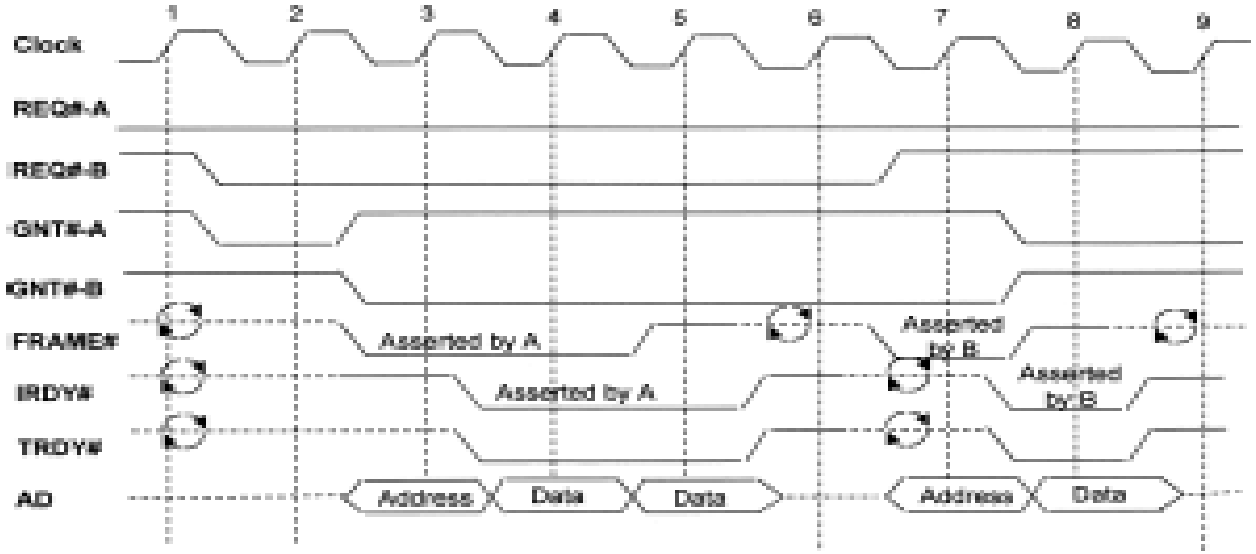
* - Из книги: Tom Shanley, Don Anderson. **PCI System Architecture. Addison-Wesley, 1999**

Сигналы шины PCI



Арбитраж PCI шины





Операции на шине PCI

C/BE[3::0]#	Тип операции
0000	Interrupt Acknowledge (подтверждение прерывания)
0001	Special Cycle (специальный цикл)
0010	I/O Read (чтение при вводе - выводе)
0011	I/O Write (запись при вводе - выводе)
0100	Зарезервировано
0101	Зарезервировано
0110	Memory Read (чтение памяти)
0111	Memory Write (запись в память)
1000	Зарезервировано
1001	Зарезервировано
1010	Configuration Read (чтение конфигурации)
1011	Configuration Write (запись конфигурации)
1100	Memory Read Multiple (множественное чтение памяти)
1101	Dual Address Cycle (двойной цикл адреса)
1110	Memory read Line (линия чтения памяти)
1111	Memory Write and Invalidate (запись в память и недействительные данные)

FRAME# - Управляется мастером для того, чтобы он мог указать начало и конец транзакции.

IRDY# - Управляется мастером, чтобы он мог инициировать циклы ожидания.

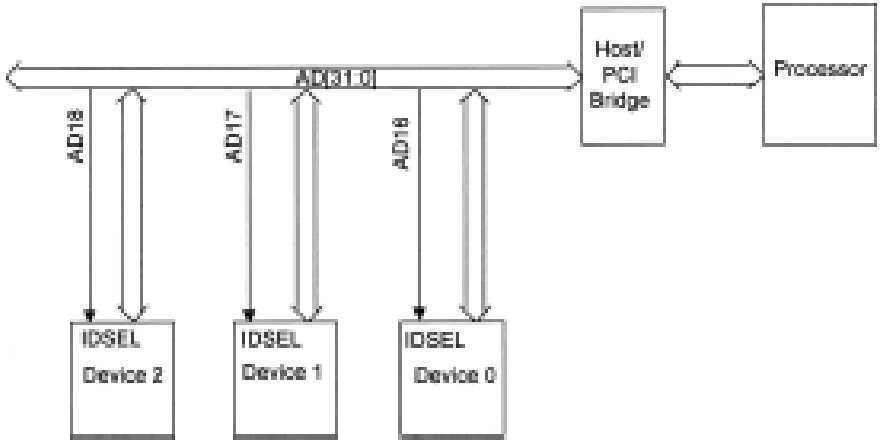
TRDY# - Управляется целевым устройством, чтобы оно могло инициировать циклы ожидания.

Адресация на PCI шине

Все определено три физических адресных пространства. Пространства адресов памяти и ввода-вывода объединены. Адресное пространство конфигураций было введено для обеспечения аппаратной конфигурации PCI.

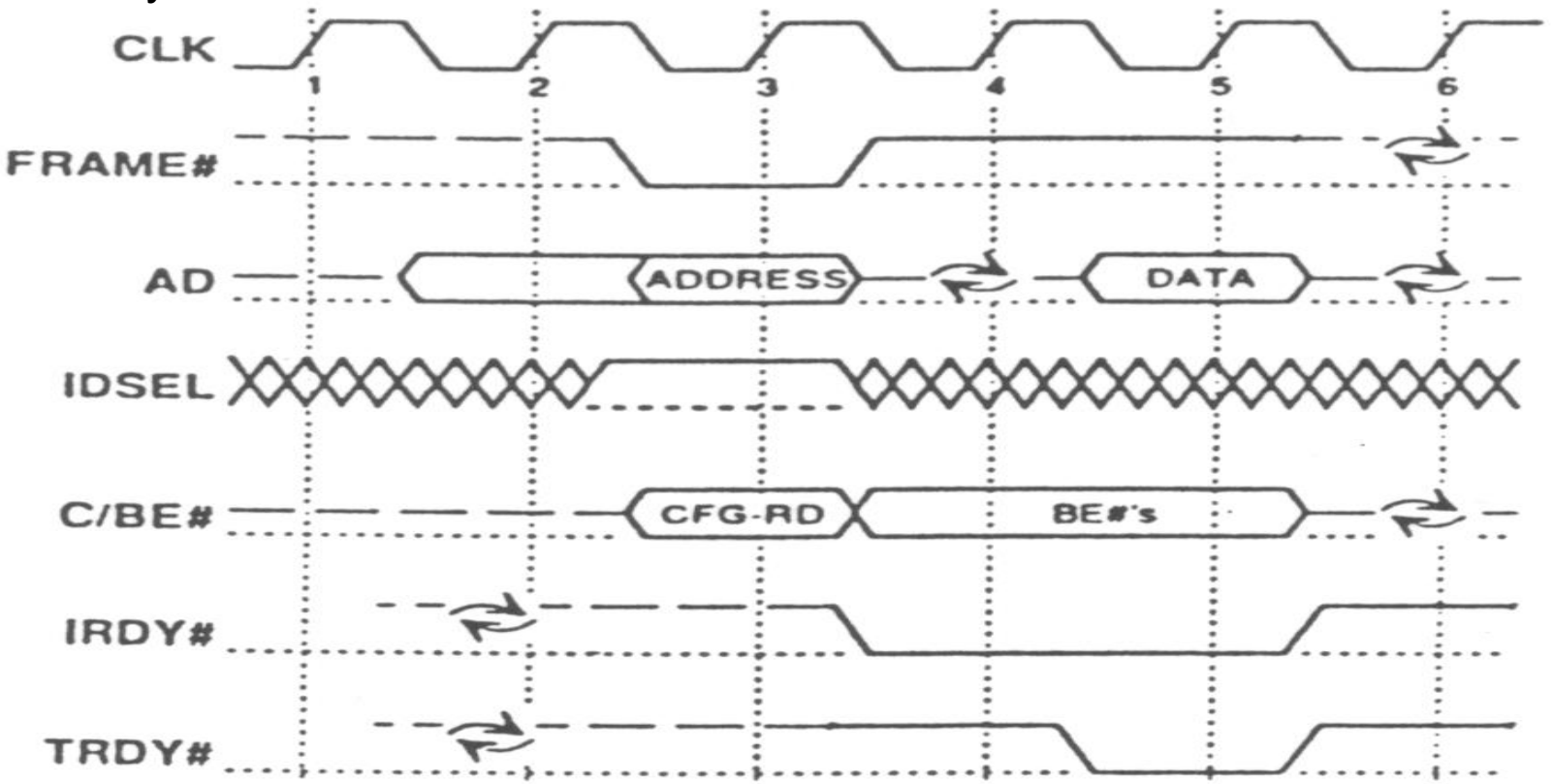
Дешифрирование адреса на шине PCI распределено; это означает, что оно выполняется на каждом устройстве. Это устраняет проблемы для центральной дешифрирующей логики, а также для сигналов выбора устройства, независимо от их использования для конфигурации. Каждый агент отвечает только на свой дешифрированный адрес.

PCI обеспечивает полную программно управляемую инициализацию и конфигурацию через отдельное адресное пространство конфигурации. PCI устройства должны обеспечить 256 байтов регистров конфигурации для этой цели.

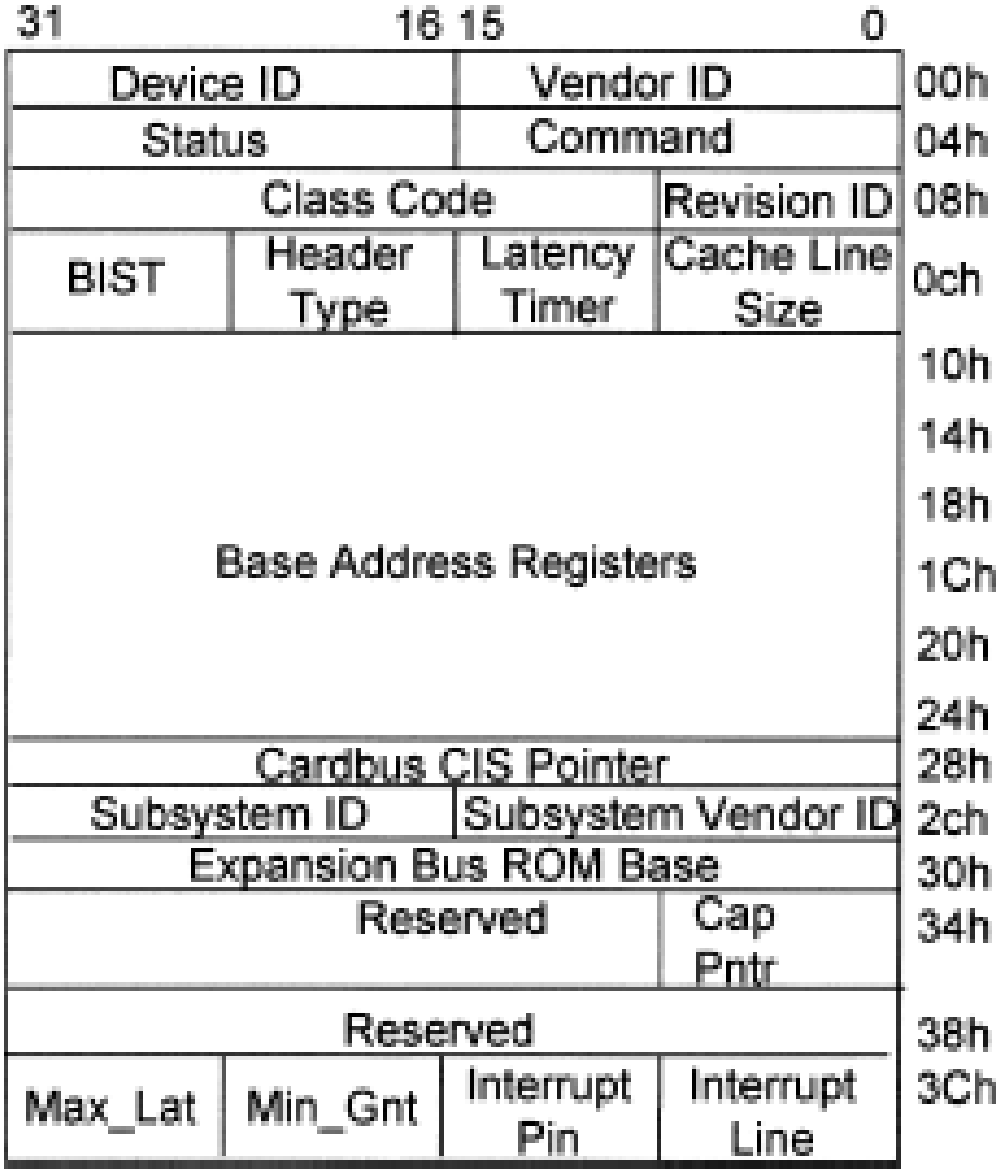


Чтение конфигурации

Для поддержки иерархии PCI шины используются два типа доступа конфигурации. Тип 1 и тип 0 доступа конфигурации различаются значениями на AD[1::0]. Тип 0 цикла конфигурации (когда AD[1::0] = "00") используется, чтобы выбрать устройство на PCI шине, где цикл выполняется. Тип 1 цикла конфигурации (когда AD[1::0] = "01") используется, чтобы передать запрос конфигурации на другую PCI шину.

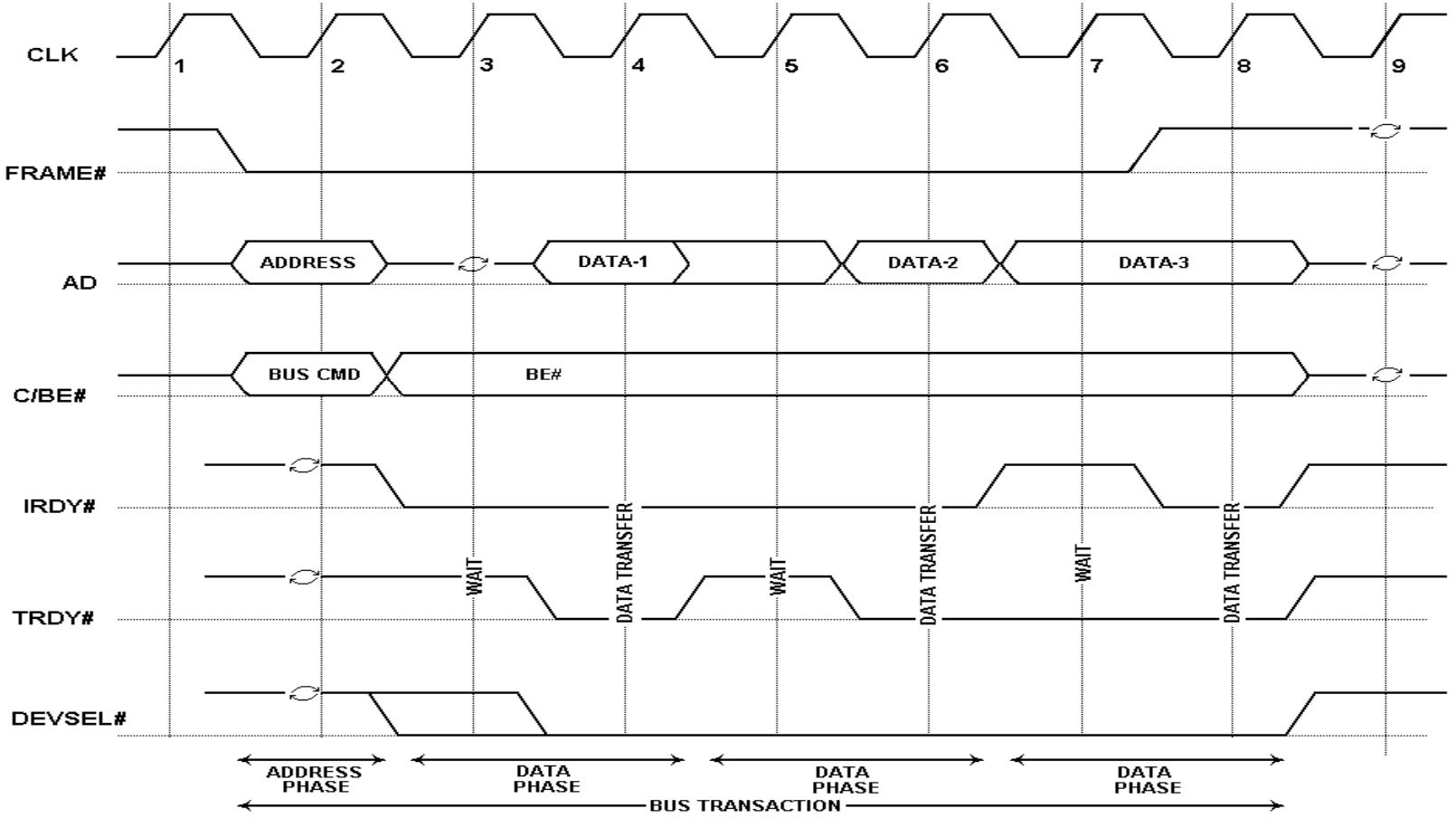


Пространство конфигурации Тип 0



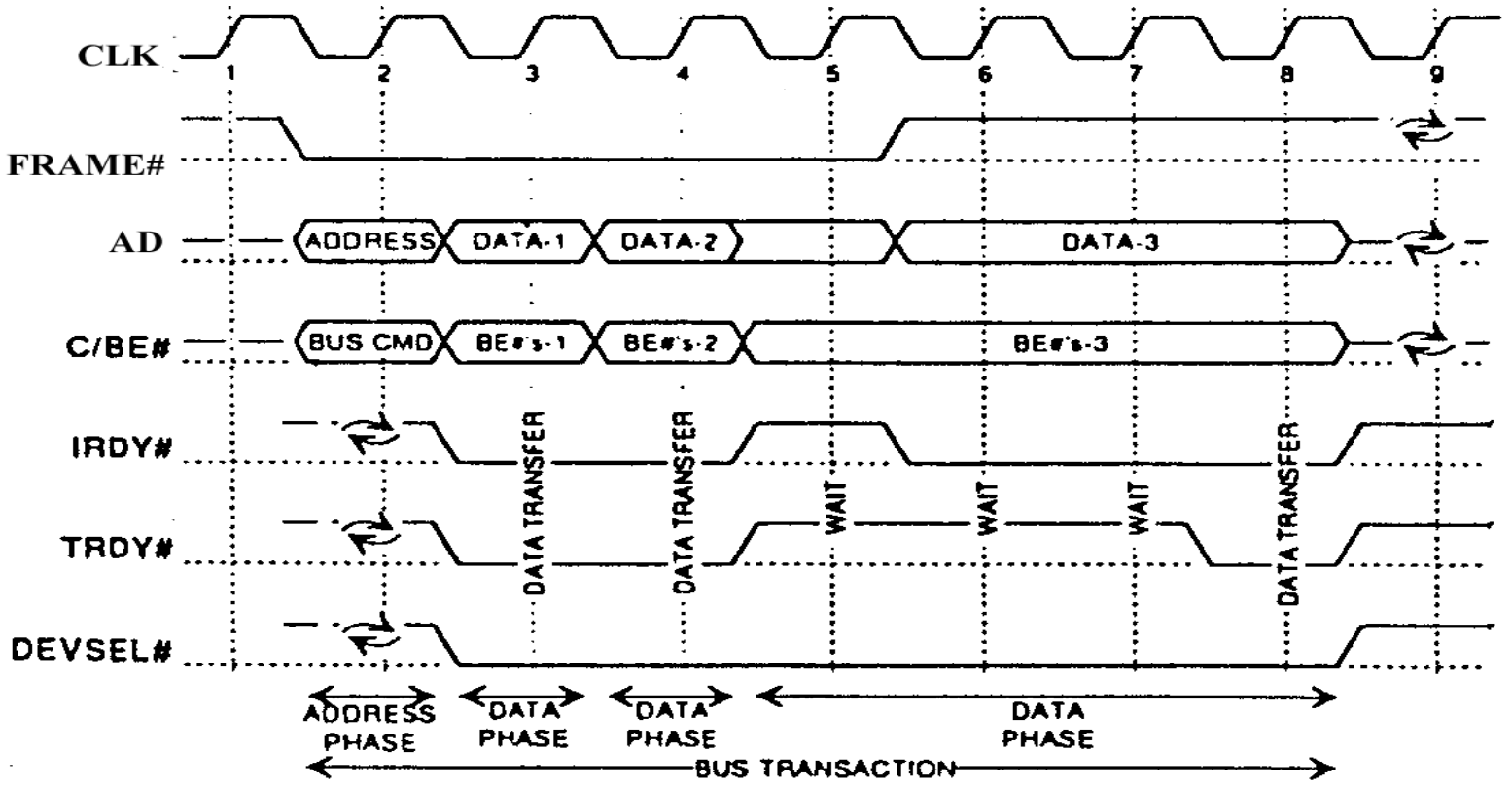
Базовая операция чтения

Транзакция чтения начинается фазой адреса, которая происходит во 2-ом такте, когда впервые устанавливается сигнал FRAME#. В течение фазы адреса AD[31::00] содержит допустимый адрес, а C/BE[3::0]# - допустимую команду шины.



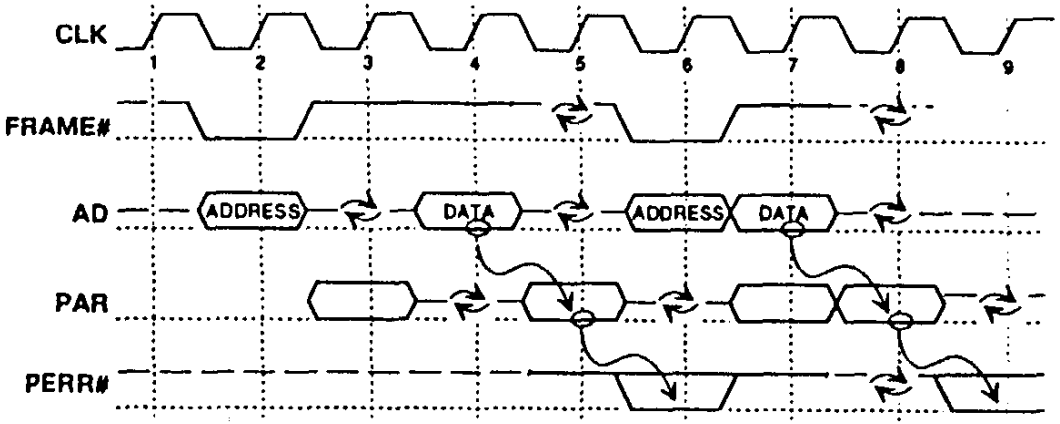
Базовая операция записи

Транзакция записи начинается в такте 2, когда впервые устанавливается в активное состояние сигнал FRAME#. Транзакция записи подобна транзакции чтения, за исключением того, что после фазы адреса не требуется оборотный цикл, так мастер обеспечивает и адрес, и данные. Фазы данных для транзакции записи такие, как и для транзакции чтения.

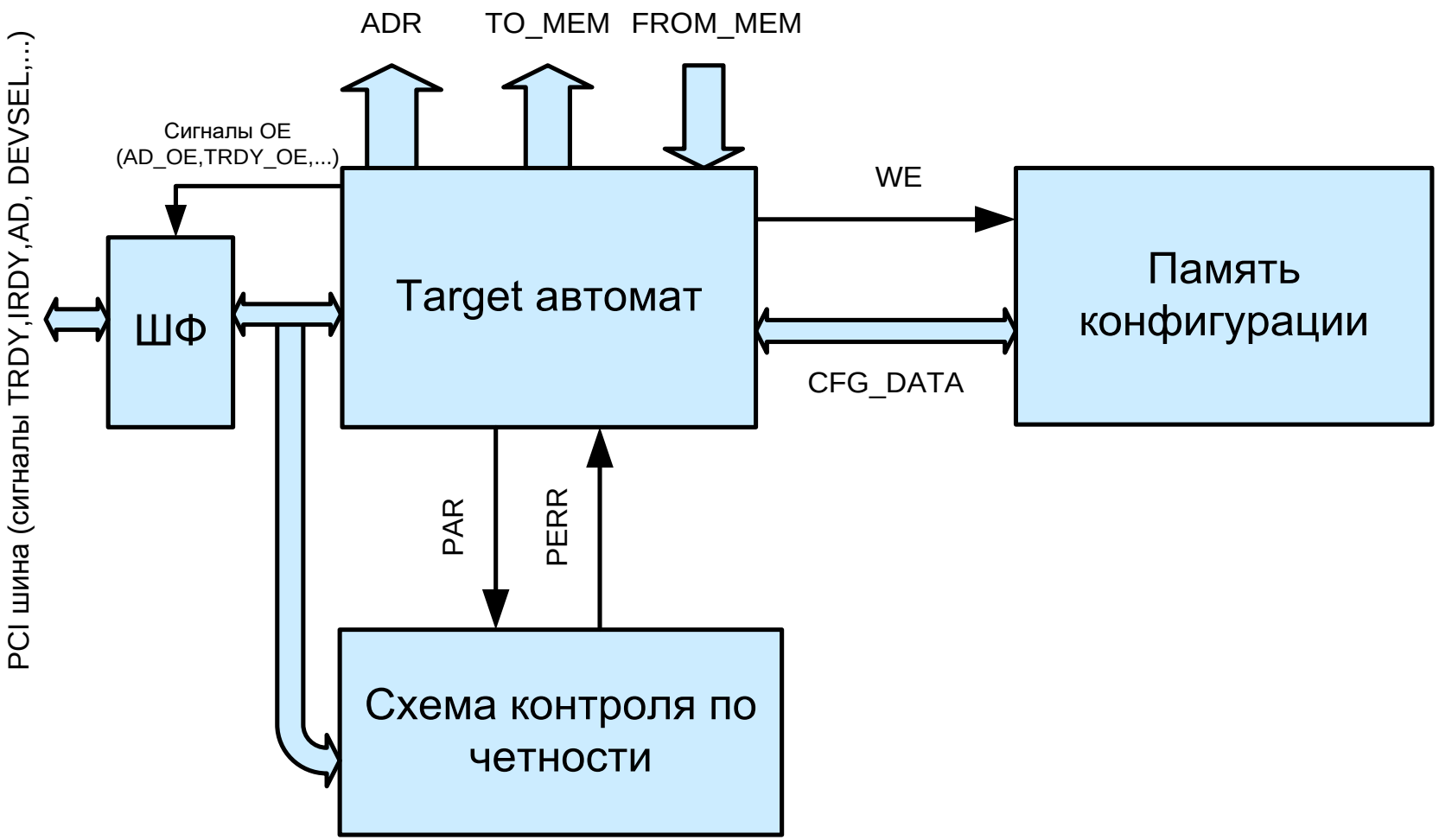


Контроль ошибок

Выводы для сообщения об ошибках требуются всем устройствам:
PERR# s/t/s
Вывод Parity Error (ошибка контроля по четности) предназначен только для сообщения об ошибках контроля по четности во время всех транзакций PCI, за исключением специального цикла (Special Cycle). Вывод PERR# - три-стабильный и должен активно управляться агентом, получающим данные в течение двух тактов, после того, как обнаружена ошибка контроля данных по четности. Минимальная продолжительность PERR# - один такт для любой фазы данных, у которой обнаружена ошибка контроля данных по четности (если идут последовательно несколько фаз данных, каждая из которых имеет ошибку контроля данных по четности, то сигнал PERR# будет установлен за более, чем один такт). PERR# должен быть установлен в высокое состояние за один такт прежде, перед тем, как он перейдет в третье состояние со всеми соответствующими тристабильными сигналами. Не существует никаких специальных условий для случая, когда теряется ошибка контроля данных по четности или сообщается об отсроченной ошибке. Агент не может установить PERR#, пока он не разрешил доступ, установив DEVSEL# и завершив фазу данных.

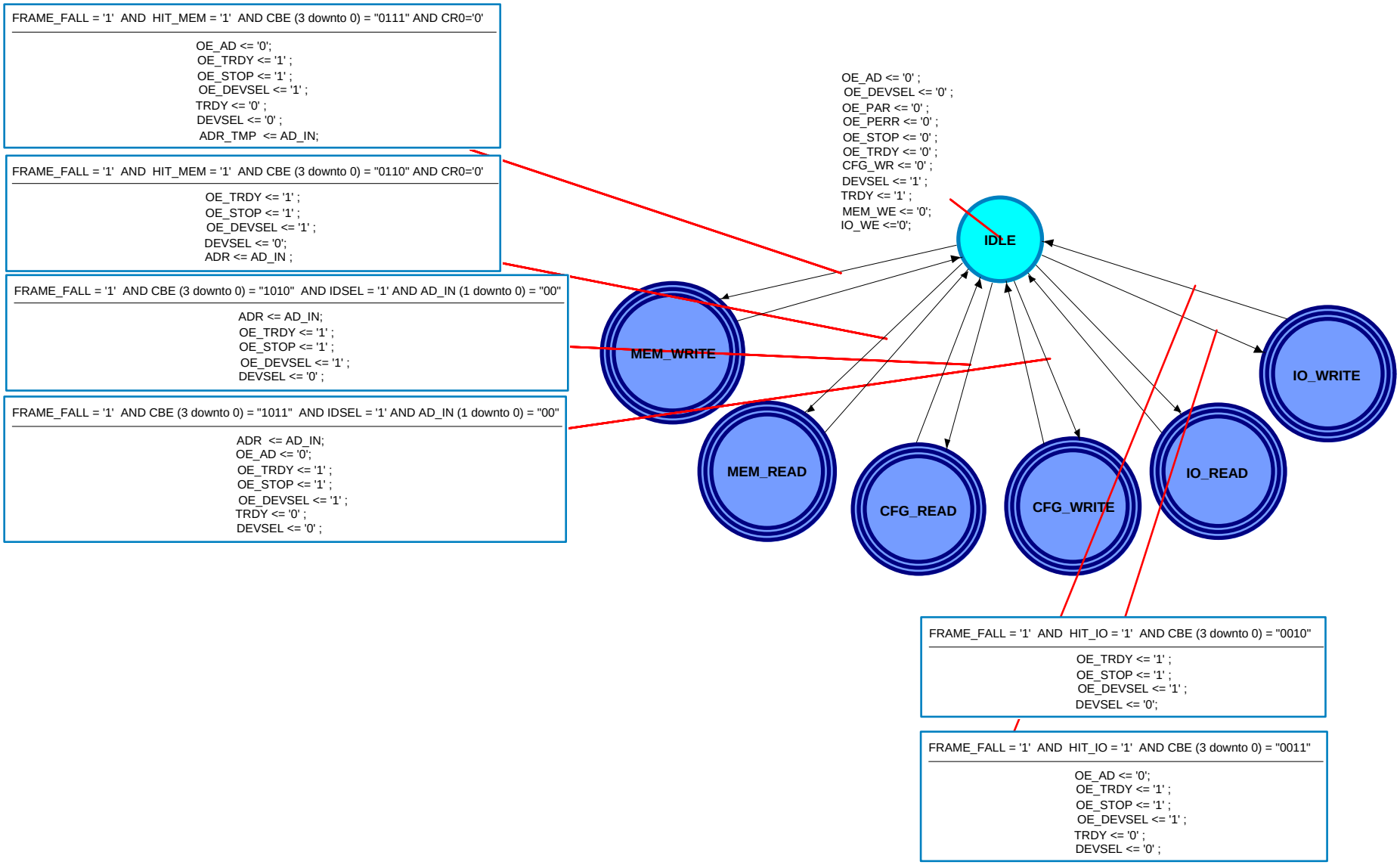


Пример PCI Target устройства



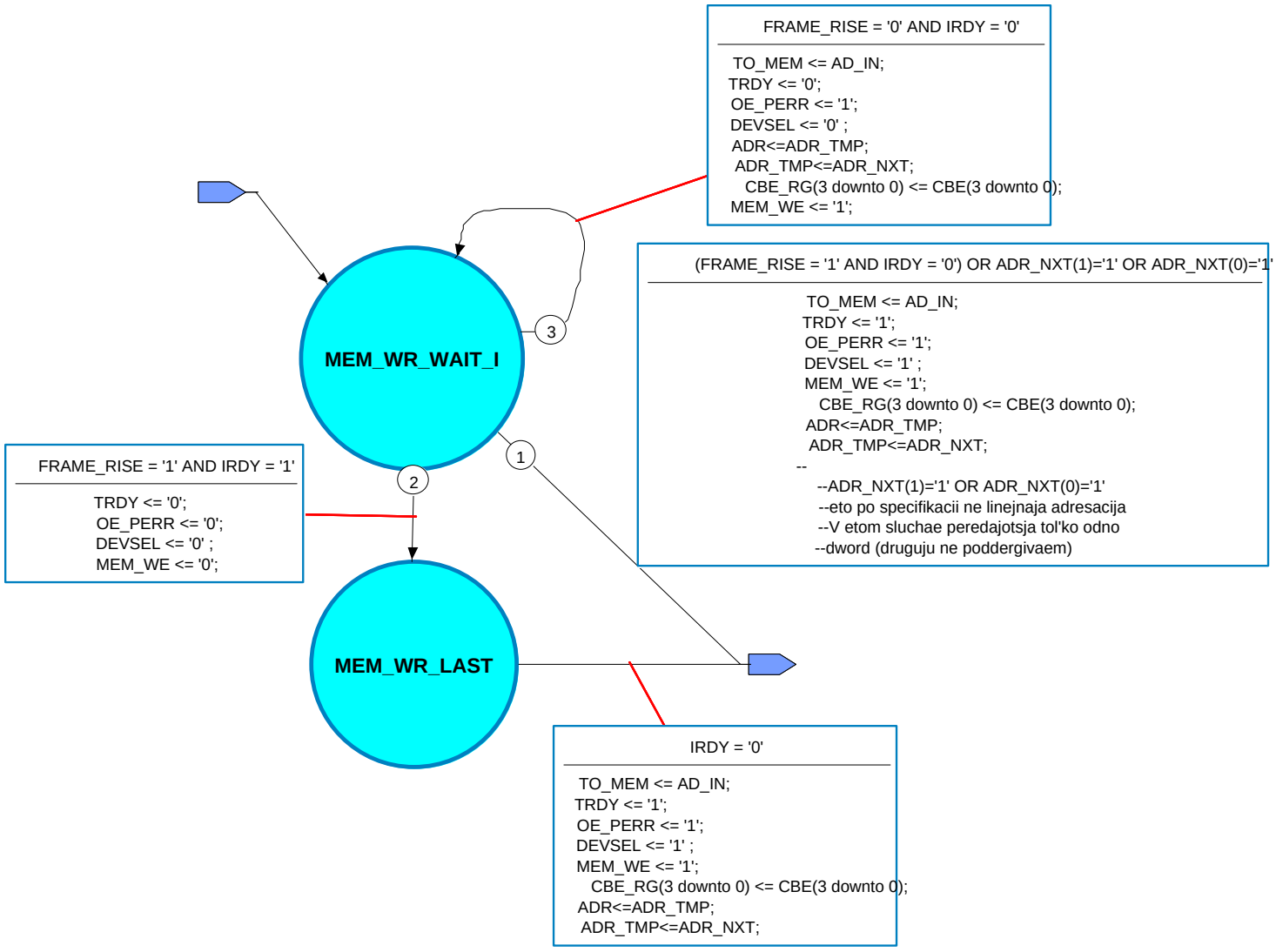
PCI Target автомат

ПРИМЕР



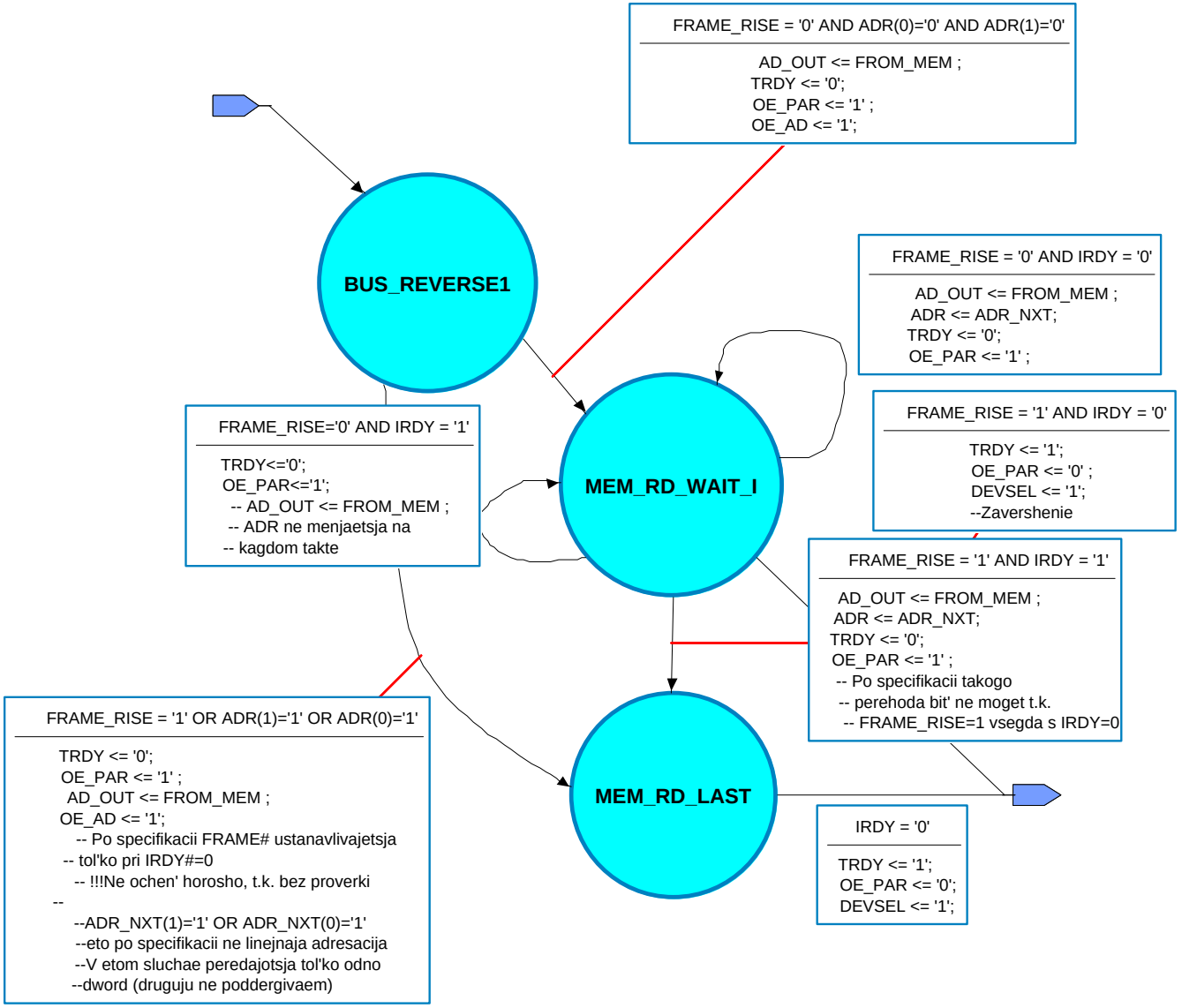
Memory Write

ПРИМЕР

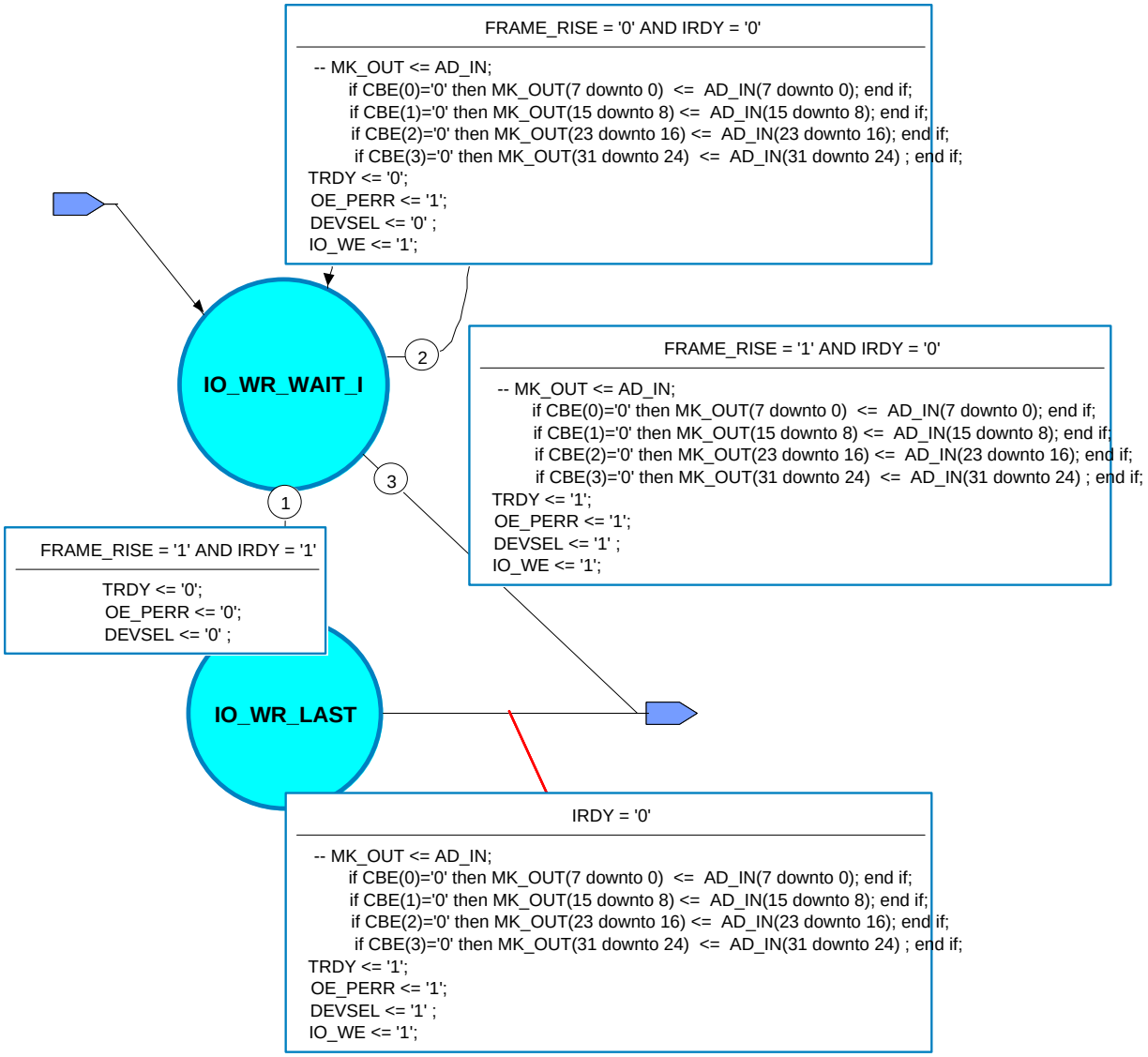


Memory Read

ПРИМЕР



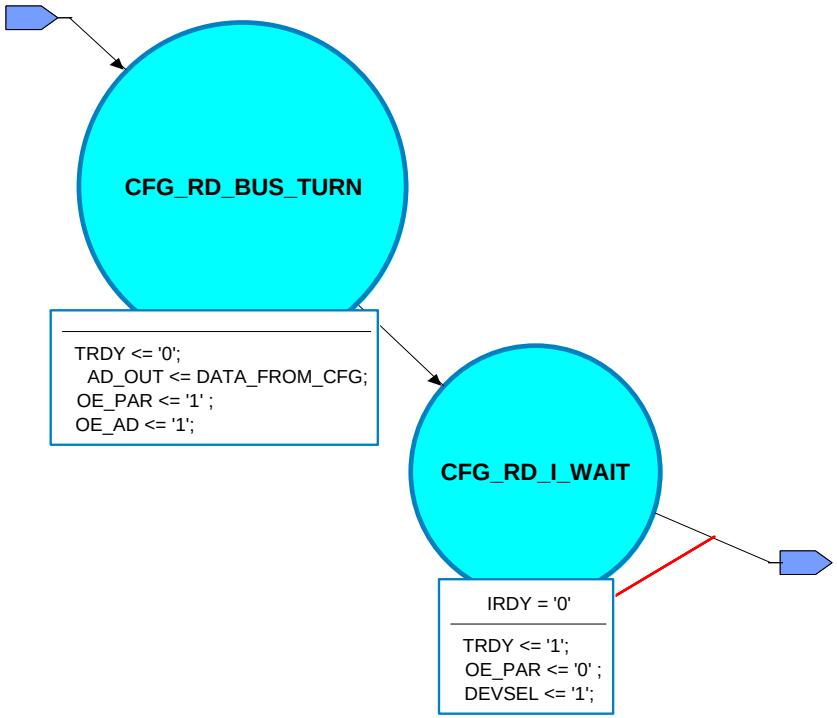
IO_WRITE



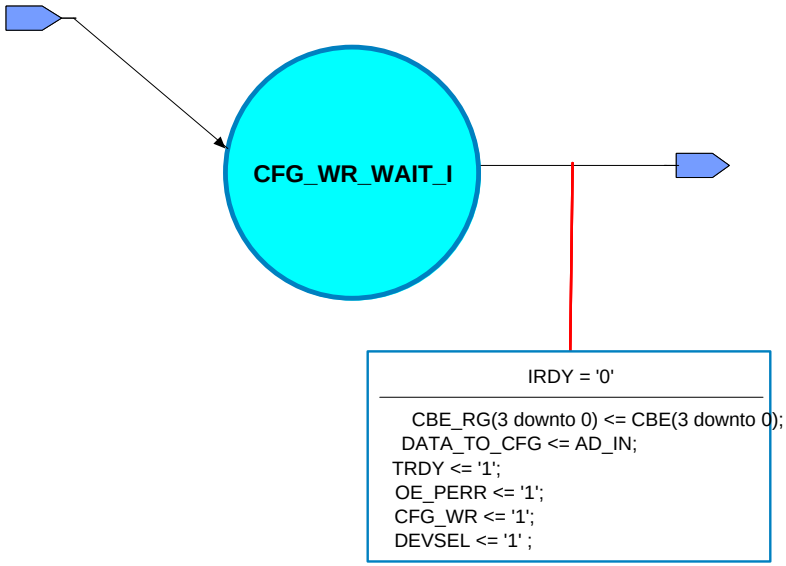
ПРИМЕР

ПРИМЕР

CFG_READ



CFG_WRITE



Пространство конфигурации

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
library WORK;

entity CFG_MEM is
  port(
CLK         : in          std_logic;
RST         : in          std_logic;
       ADR                : in          std_logic_vector(3 downto 0);
D           : in          std_logic_vector(31 downto 0);
WE          : in          std_logic;
CBE         : in          std_logic_vector(3 downto 0);
AD_IN             : in          std_logic_vector(31 downto 0);
PERR              : in          std_logic;
PERR_OE           : in          std_logic;
       Q          : out         std_logic_vector(31 downto 0);
PERR_Enable   :out         std_logic;
HIT_IO            : out         std_logic;
HIT_MEM           : out         std_logic

  );
end CFG_MEM;
```

ПРИМЕР

```
architecture RTL of CFG_MEM is
subtype R_DWORD is std_logic_vector (31 downto 0);
subtype R_HWORD is std_logic_vector (23 downto 0);
subtype R_WORD is std_logic_vector (15 downto 0);
subtype R_BYTE is std_logic_vector (7 downto 0);
constant Z_DWORD :R_DWORD := "00000000000000000000000000000000";
constant    DeviceID   :R_WORD := "0000000000000001";
constant     VendorID  :R_WORD := "0010001100100011"; --RO :2323h
signal      Command  :R_WORD;
signal      Status      :R_WORD;
constant    RevisionID :R_BYTE    := "00000001";
constant    ClassCode:R_HWORD := "000100010000000000000000";
constant    CashLineSize:R_BYTE := "00000000";
constant    LatencyTimer:R_BYTE := "00000000";
constant    HeaderType:R_BYTE  := "00000000";
constant    BIST :R_BYTE   := "00000000";
signal      BAR0       :R_DWORD; --IO BAR0
signal      BAR1       :R_DWORD; --MEM BAR1
constant    BAR2       :R_DWORD      := "00000000000000000000000000000000"; --RW
constant    BAR3       :R_DWORD      := "00000000000000000000000000000000"; --RW
constant    BAR4       :R_DWORD      := "00000000000000000000000000000000"; --RW
constant    BAR5       :R_DWORD      := "00000000000000000000000000000000";
constant    CardbusCISPointer:R_DWORD    := "00000000000000000000000000000000";
constant    SubsystemVendorID:R_WORD          := "0000000000000000";
constant    SubsystemID :R_WORD           := "0000000000000000";
constant    ExpansionBusROMBase:R_DWORD    :="00000000000000000000000000000000";
constant    CapPntr           :R_BYTE   := "00000000";
constant    InterruptLine     :R_BYTE   := "00000000";
constant    InterruptPin      :R_BYTE   := "00000000";
constant    MinGnt            :R_BYTE   := "00000000";
constant    MaxLat            :R_BYTE   := "00000000";
```

```
begin

CFG_RD: process(ADR,Command,Status,BAR0)
begin
        case ADR is
        when "0000" =>      Q <= DeviceID & VendorID;
        when "0001" =>      Q <= Status & Command;
        when "0010" =>      Q <= ClassCode & RevisionID;
        when "0011" =>      Q <= BIST & HeaderType & LatencyTimer & CashLineSize;
        when "0100" =>      Q <= BAR0;
        when "0101" =>      Q <= BAR1;
        when "0110" =>      Q <= BAR2;
        when "0111" =>      Q <= BAR3;
        when "1000" =>      Q <= BAR4;
        when "1001" =>      Q <= BAR5;
        when "1010" =>      Q <= CardbusCISPointer;
        when "1011" =>      Q <= SubsystemID & SubsystemVendorID;
        when "1100" =>      Q <= ExpansionBusROMBase;
        when "1101" =>      Q <= Z_DWORD;
        when "1110" =>      Q <= Z_DWORD;
        when "1111" =>      Q <= MaxLat & MinGnt & InterruptPin & InterruptLine;
        when others => NULL;
        end case;
end process;
```

ПРИМЕР

```
CFG_WR:             process(ADR,CLK,CBE,RST,WE,D,PERR,PERR_OE)
begin
if (RST = '0') then
            Command  <= "0000000001000000";    --RW
                        Status    <= "0000000000000000";       --RW
            BAR0      <= "11111111111111111111111111111101";
            BAR1      <= "11111111111111111111111111000000";
elsif (CLK'event and CLK='1') then
            if WE = '1' then
                        if ADR(3 downto 0) = "0001" then
                                                if CBE(0)='0' then
                                                   Command(0)  <=  D(0); --IO Space
                                                   Command(1)  <=  D(1); --Mem Space
                                                   Command(6)  <=  D(6); --Parity Error
                                    end if;
                                                if CBE(1)='0' then
                                                   Command(8)  <=  D(8); --SERR Enable
                                                end if;
```

ПРИМЕР

```
                if CBE(2)='0' then Status(7 downto 0)   <=  (Status(7 downto 0) and not D(23 downto 16) ); end if;
                if CBE(3)='0' then Status(15 downto 8)  <=  (Status(15 downto 8) and not D(31 downto 24) ); end if;
elsif ADR(3 downto 0) = "0100" then
                        if CBE(0)='0' then BAR0(7 downto 2)     <=  D(7 downto 2); end if;
                        if CBE(1)='0' then BAR0(15 downto 8)    <=  D(15 downto 8); end if;
                        if CBE(2)='0' then BAR0(23 downto 16)   <=  D(23 downto 16); end if;
                        if CBE(3)='0' then BAR0(31 downto 24)   <=  D(31 downto 24); end if;
elsif ADR(3 downto 0) = "0101" then
                        if CBE(0)='0' then BAR1(7 downto 6)     <=  D(7 downto 6); end if;
                        if CBE(1)='0' then BAR1(15 downto 8)    <=  D(15 downto 8); end if;
                        if CBE(2)='0' then BAR1(23 downto 16)   <=  D(23 downto 16); end if;
                        if CBE(3)='0' then BAR1(31 downto 24)   <=  D(31 downto 24); end if;
end if;
     end if;
  if  PERR = '0' and PERR_OE = '1' then Status(15) <= '1'; end if;
end if;
end process;
IO_HIT_DETECT: process(AD_IN)
begin
if (BAR0(31 downto 2) = AD_IN(31 downto 2)) and (Command(0) = '1') then Hit_IO <= '1';
else Hit_IO <= '0'; end if;
end process;
MEM_HIT_DETECT: process(AD_IN)
begin
if (BAR1(31 downto 6) = AD_IN(31 downto 6)) and (Command(1) = '1') then Hit_MEM <= '1';
else Hit_MEM <= '0'; end if;
end process;
PERR_Enable <= Command(6);
end RTL;
```

Контроль по четности

ПРИМЕР

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY Parity_PRT IS
PORT(
        CLK             : IN        std_logic;
        RST             : IN        std_logic;
         AD_IN          : IN        std_logic_vector (31 DOWNTO 0);
        AD_OUT          : IN        std_logic_vector (31 DOWNTO 0);
        CBE             : IN        std_logic_vector (3 DOWNTO 0);
         PAR_IN         : IN        std_logic;
         PERR_IN        : IN        std_logic;
        OE_AD           : IN        std_logic;
        OE_PAR          : IN        std_logic;
        OE_PAR1         : OUT       std_logic;
        OE_PERR         : IN        std_logic;
        OE_PERR2            : OUT       std_logic;
 PAR_OUT        : OUT       std_logic;
        PERR_OUT            : OUT       std_logic;
STOP            : OUT       std_logic;
        PERR_EN         : IN  std_logic
   );
END Parity_PRT;
```

```
ARCHITECTURE  PRT OF Parity_PRT IS

  SIGNAL PAR_NEW : std_logic;
  SIGNAL P_IN: std_logic_vector (8 downto 0);
  SIGNAL CBE_PAR: std_logic;
  SIGNAL P_OUT: std_logic_vector (8 downto 0);
  -- SIGNAL OE_PERR1     : std_logic;

BEGIN
   P0: CBE_PAR <= CBE(0) xor CBE(1) xor CBE(2) xor CBE(3);
   PC0: P_IN(0) <=  AD_IN(0) xor  AD_IN(1) xor  AD_IN(2) xor AD_IN(3);
   PC1: P_IN(1) <=  AD_IN(4) xor  AD_IN(5) xor  AD_IN(6) xor AD_IN(7);
   PC2: P_IN(2) <=  AD_IN(8) xor  AD_IN(9) xor AD_IN(10) xor AD_IN(11);
   PC3: P_IN(3) <= AD_IN(12) xor AD_IN(13) xor AD_IN(14) xor AD_IN(15);
   PC4: P_IN(4) <= AD_IN(16) xor AD_IN(17) xor AD_IN(18) xor AD_IN(19);
   PC5: P_IN(5) <= AD_IN(20) xor AD_IN(21) xor AD_IN(22) xor AD_IN(23);
   PC6: P_IN(6) <= AD_IN(24) xor AD_IN(25) xor AD_IN(26) xor AD_IN(27);
   PC7: P_IN(7) <= AD_IN(28) xor AD_IN(29) xor AD_IN(30) xor AD_IN(31);
   PC8: P_IN(8) <= P_IN(0) xor P_IN(1) xor P_IN(2)xor P_IN(3)xor P_IN(4)xor
P_IN(5)xor P_IN(6)xor P_IN(7) xor CBE_PAR;
```

ПРИМЕР

```
PG0: P_OUT(0) <= AD_OUT(0) xor AD_OUT(1) xor AD_OUT(2) xor AD_OUT(3);
  PG1: P_OUT(1) <= AD_OUT(4) xor AD_OUT(5) xor AD_OUT(6) xor AD_OUT(7);
  PG2: P_OUT(2) <= AD_OUT(8) xor AD_OUT(9) xor AD_OUT(10) xor AD_OUT(11);
  PG3: P_OUT(3) <= AD_OUT(12) xor AD_OUT(13) xor AD_OUT(14) xor AD_OUT(15);
  PG4: P_OUT(4) <= AD_OUT(16) xor AD_OUT(17) xor AD_OUT(18) xor AD_OUT(19);
  PG5: P_OUT(5) <= AD_OUT(20) xor AD_OUT(21) xor AD_OUT(22) xor AD_OUT(23);
  PG6: P_OUT(6) <= AD_OUT(24) xor AD_OUT(25) xor AD_OUT(26) xor AD_OUT(27);
  PG7: P_OUT(7) <= AD_OUT(28) xor AD_OUT(29) xor AD_OUT(30) xor AD_OUT(31);
  PG8: P_OUT(8) <= P_OUT(0) xor P_OUT(1) xor P_OUT(2)xor P_OUT(3)xor P_OUT(4)xor
P_OUT(5)xor P_OUT(6)xor P_OUT(7) xor CBE_PAR;
```

```
NXT_PAR_OUT:  process (CLK,RST,OE_PAR,P_OUT(8))
begin
if RST='0' then
    PAR_OUT <='0';
 elsif (CLK'event and CLK = '1') then
       PAR_OUT <= P_OUT(8);
       OE_PAR1 <= OE_PAR;
     end if;
end process;
CHK_PAR_IN: process (CLK,RST,OE_PERR,PAR_NEW)
begin
if RST='0' then
    PERR_OUT <='1';
elsif (CLK'event and CLK = '1') then
   PERR_OUT <= (PAR_NEW xor PAR_IN) nand PERR_EN;
--           (PAR_NEW xor PAR_IN) (PERR_EN) (PERR_OUT)
--           1 1 0
--           0 1 1
--           1 0 1
--           0 0 1
     end if;
end process;
```

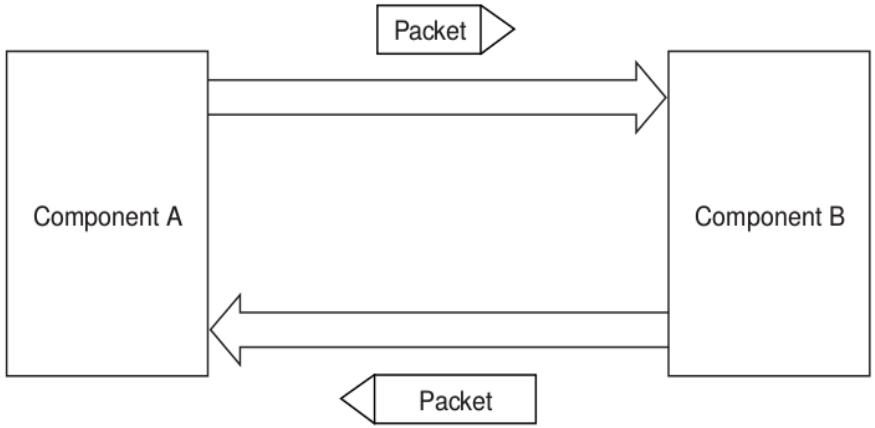
ПРИМЕР

```
process (CLK,RST,OE_PERR,P_IN(8),PAR_IN)
begin
if RST='0' then
PAR_NEW <= '1';
       OE_PERR2 <='0';
 elsif (CLK'event and CLK = '1') then
   PAR_NEW <= P_IN(8);
         OE_PERR2 <= OE_PERR;
     end if;
end process;
END PRT;
```

Шина PCI express

Преимущества последовательных шин и интерфейсов:

- Больший акцент на сложную логику при простой топологии физического уровня;
- Перспектива перехода на оптический физический уровнеь;
- Экономия пространства печатных плат и снижение сложности монтажа;
- Простота реализации PnP и динамическую конфигурацию в любом смысле;
- Возможность выделять гарантированные и изохронные каналы;
- Переход от разделяемых шин с арбитражем к более предсказуемым соединениям точка-точка;
- Лучшая с точки зрения затрат и более гибкая с точки зрения топологии масштабируемость;



Сравнение пропускной способности шин семейства PCI

PCI и PCI-X

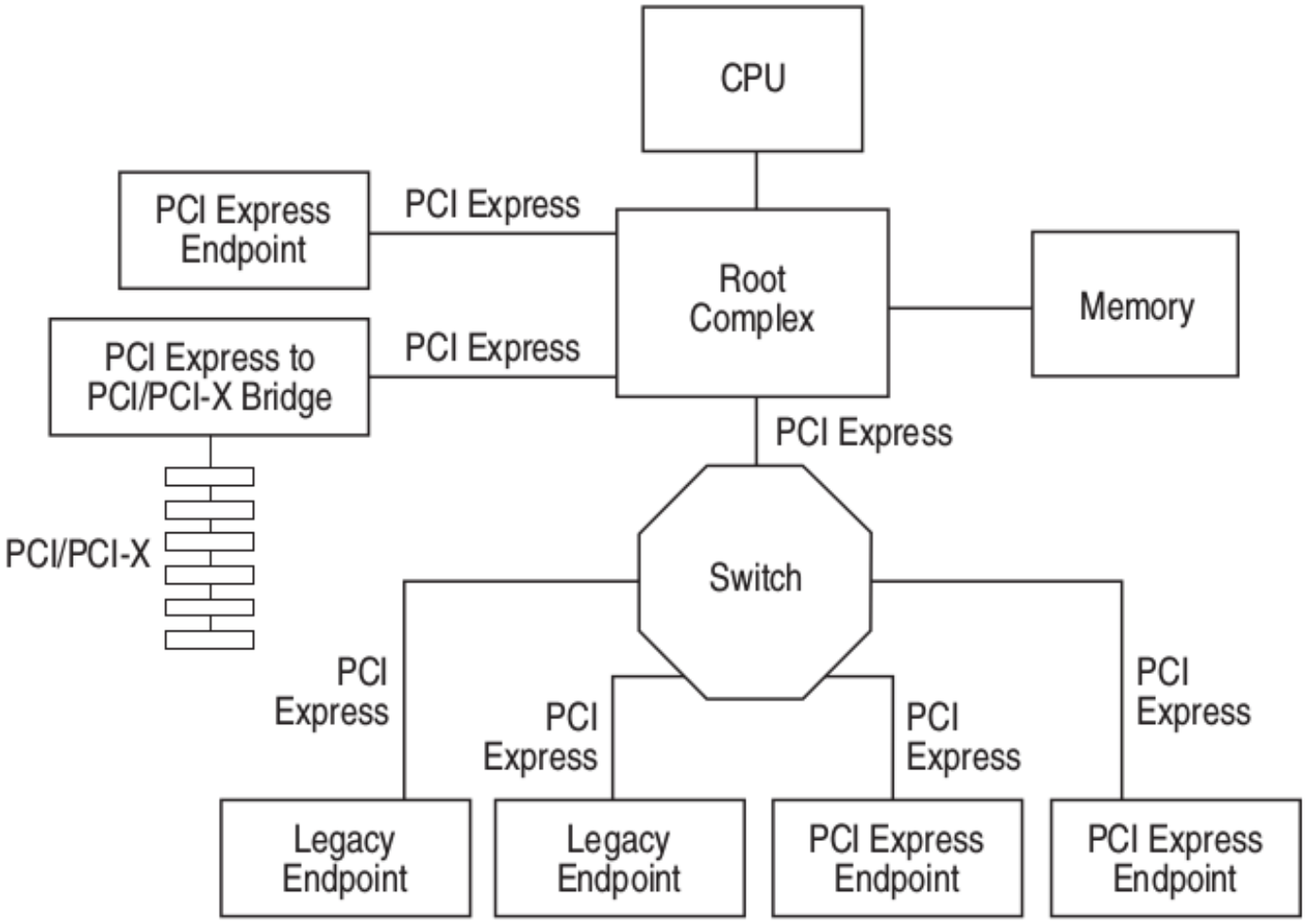
Bus Type	Clock Frequency	Peak Bandwidth *	Number of Card Slots per Bus
PCI 32-bit	33 MHz	133 MBytes/sec	4-5
PCI 32-bit	66 MHz	266 MBytes/sec	1-2
PCI-X 32-bit	66 MHz	266 MBytes/sec	4
PCI-X 32-bit	133 MHz	533 MBytes/sec	1-2
PCI-X 32-bit	266 MHz effective	1066 MBytes/sec	1
PCI-X 32-bit	533 MHz effective	2131 MByte/sec	1

* Double all these bandwidth numbers for 64-bit bus implementations

PCI Express

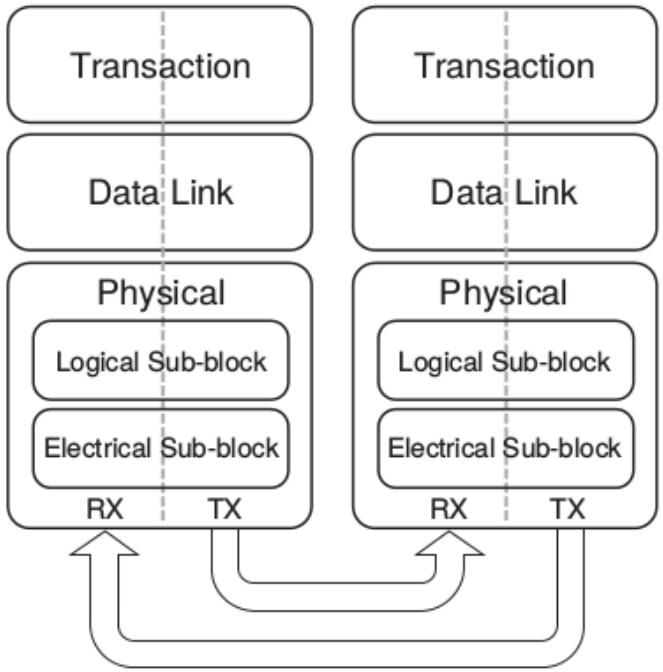
PCI Express version	Line code	Transfer rate[i]	Throughput[i]				
			×1	×2	×4	×8	×16
1.0	8b/10b	2.5 GT/s	250 MB/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s
2.0	8b/10b	5.0 GT/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s	8 GB/s
3.0	128b/130b	8.0 GT/s	984.6 MB/s	1.97 GB/s	3.94 GB/s	7.9 GB/s	15.8 GB/s
4.0	128b/130b	16.0 GT/s	1969 MB/s	3.94 GB/s	7.9 GB/s	15.8 GB/s	31.5 GB/s
5.0[30][31] *(expected in Q2 2019)*[33]	128b/130b	32.0 GT/s[ii]	3938 MB/s	7.9 GB/s	15.8 GB/s	31.5 GB/s	63.0 GB/s

Принципы взаимодействия устройств по PCI Express



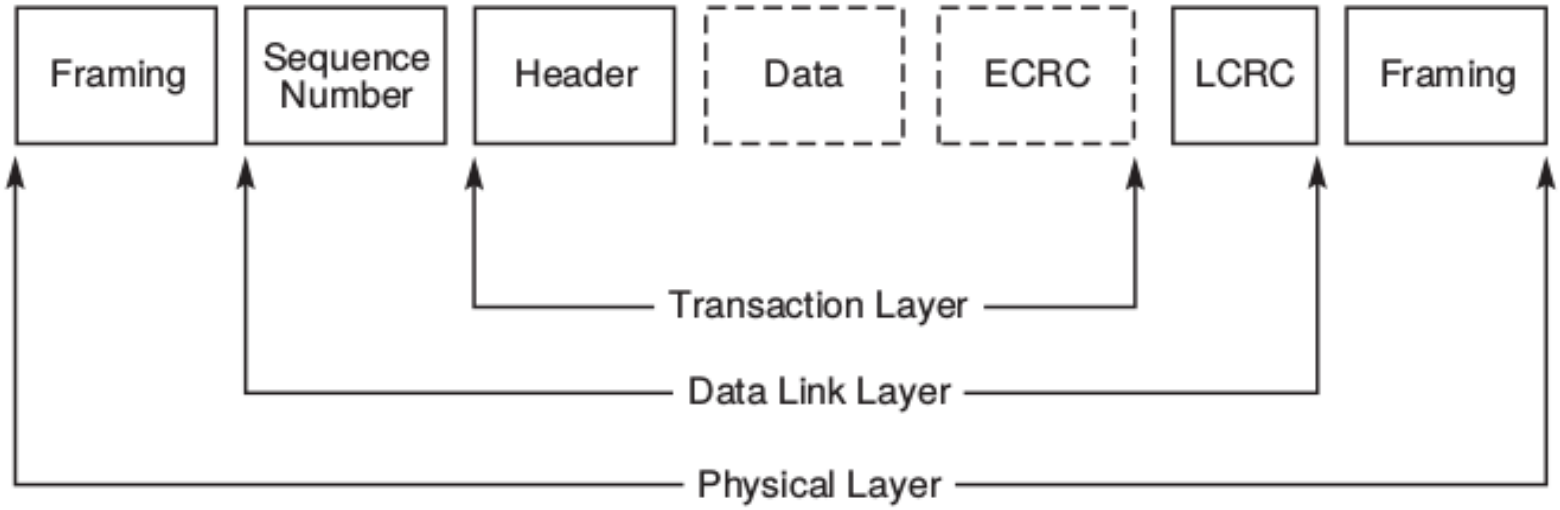
Соединение между двумя устройствами PCI Express называется link, и состоит из одного (называемого 1x) или нескольких (2x, 4x, 8x, 12x, 16x и 32x) двунаправленных последовательных соединений lane. Каждое устройство должно поддерживать соединение 1x.

Принципы взаимодействия устройств по PCI Express



- Уровень транзакций отвечает за взаимодействие с пригладным уровнем, сборку и разборку TLP пакетов (используются для передачи транзакций на нижние уровни, чтение и запись, передача событий). Уровень транзакций также отвечает за управление кредитами.
- Уровень линка данных принимает TLP пакеты от транзакционного уровня и осуществляет их передачу через физический уровень. Основные обязанности уровня линка данных включают управление линками и обеспечение целостности данных при передаче, включая обнаружение ошибок и исправление ошибок.
- Физический уровень включает в себя все необходимые схемы для работы интерфейса, параллельно-последовательные преобразователи, кодеры и декодеры 8/10, ФАПЧ(ы), схемы согласования сопротивлений.

Принципы взаимодействия устройств по PCI Express



- TLP, которые не проходят проверку целостности данных (LCRC и порядковый номер) или которые теряются при передаче из одного компонента в другой, повторно отправляются передатчиком.
- Передатчик хранит копию всех отправленных TLP, повторно отправляет эти копии, когда это необходимо, и очищает копии только тогда, когда он получает положительное подтверждение безошибочного получения от другого компонента.
- Если положительное подтверждение не было получено в течение указанного периода времени, Передатчик автоматически начнет повторную передачу.
- Приемник может запросить немедленную повторную передачу, используя отрицательное подтверждение NAK.

Принципы взаимодействия устройств по PCI Express

Transaction Type	Non-Posted or Posted
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read (Type 0 and Type 1)	Non-Posted
Configuration Write (Type 0 and Type 1)	Non-Posted
Message	Posted

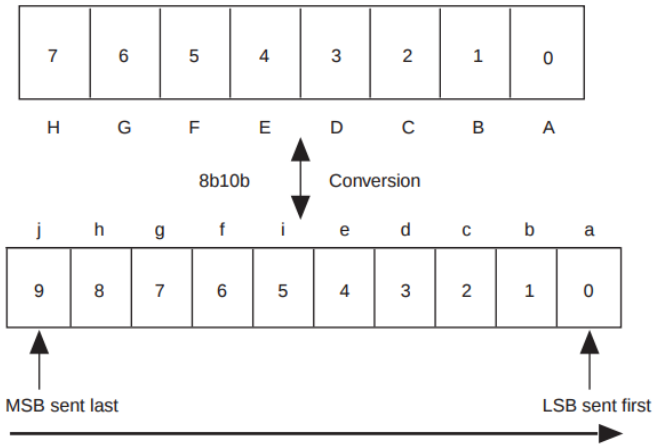
Участниками транзакции является Requester (инициатор) и Completer (исполнитель).

Транзакции Non-Posted предполагают получение ответа инициатором от исполнителя об успешном завершении.

Транзакция Posted не требует ответа от исполнителя.

Способы кодирования данных при приемопередаче

Кодирование 8/10 (PCIe v1.0, v2.0)



Всевозможных 10-битовых комбинаций больше, чем реально используется для представления 256 обычных символов. В наборе символов представлены числовые данные D и специальные символы K (control symbols).

Каждый символ имеет два образа: с положительным/отрицательным балансом нулевых и единичных бит (или с одинаковым количеством нулевых и единичных символов).

При получении 10-битовой последовательности битов, не со
сим

Encoding	Symbol	Name	Description
K28.5	COM	Comma	Used for Lane and Link initialization and management
K27.7	STP	Start TLP	Marks the start of a Transaction Layer Packet
K28.2	SDP	Start DLLP	Marks the start of a Data Link Layer Packet
K29.7	END	End	Marks the end of a Transaction Layer Packet or a Data Link Layer Packet
K30.7	EDB	EnD Bad	Marks the end of a nullified TLP
K23.7	PAD	Pad	Used in Framing and Link Width and Lane ordering negotiations
K28.0	SKP	Skip	Used for compensating for different bit rates for two communicating Ports
K28.1	FTS	Fast Training Sequence	Used within an Ordered Set to exit from L0s to L0
K28.3	IDL	Idle	Used in the Electrical Idle Ordered Set (EIOS)
K28.4			Reserved
K28.6			Reserved
K28.7	EIE	Electrical Idle Exit	Reserved in 2.5 GT/s Used in the Electrical Idle Exit Ordered Set (EIEOS) and sent prior to sending FTS at data rates other than 2.5 GT/s

Data Byte Name	Data Byte Value	Bits HGF EDCBA	Current RD - abcdei fghj	Current RD + abcdei fghj
K28.0	1C	000 11100	001111 0100	110000 1011
K28.1	3C	001 11100	001111 1001	110000 0110
K28.2	5C	010 11100	001111 0101	110000 1010
K28.3	7C	011 11100	001111 0011	110000 1100
K28.4	9C	100 11100	001111 0010	110000 1101
K28.5	BC	101 11100	001111 1010	110000 0101
K28.6	DC	110 11100	001111 0110	110000 1001
K28.7	FC	111 11100	001111 1000	110000 0111
K23.7	F7	111 10111	111010 1000	000101 0111
K27.7	FB	111 11011	110110 1000	001001 0111
K29.7	FD	111 11101	101110 1000	010001 0111
K30.7	FE	111 11110	011110 1000	100001 0111

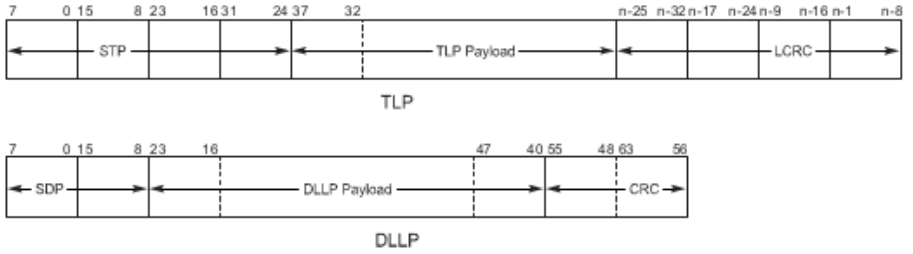
Physical Layer Protocol

Формат пакетов TS1 на Physical Layer

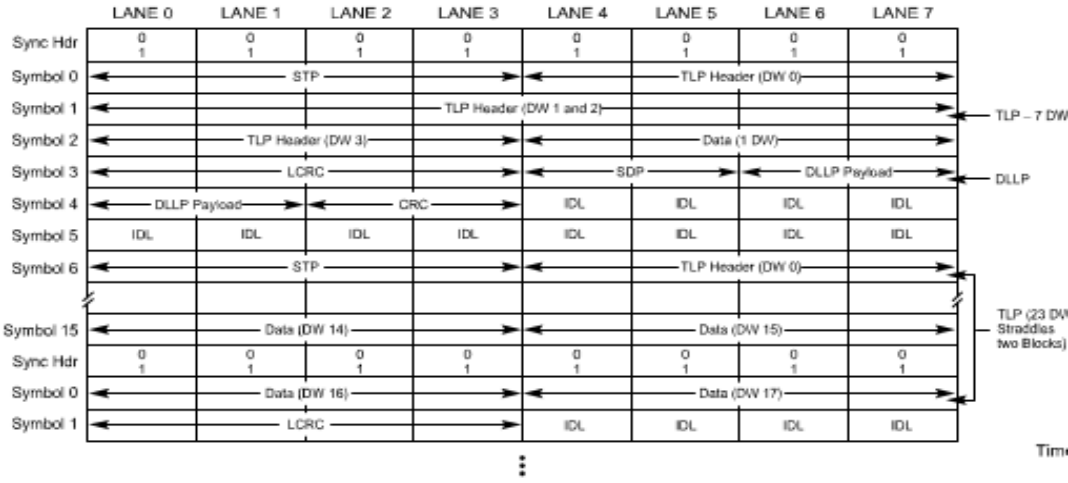
Table 4-5: TS1 Ordered Set

Symbol Number	Description
0	When operating at 2.5 or 5.0 GT/s: COM (K28.5) for Symbol alignment. When operating at 8.0 GT/s or above: Encoded as 1Eh (TS1 Ordered Set).
1	Link Number. Ports that do not support 8.0 GT/s or above: 0-255, PAD. Downstream Ports that support 8.0 GT/s or above: 0-31, PAD. Upstream Ports that support 8.0 GT/s or above: 0-255, PAD. When operating at 2.5 or 5.0 GT/s: PAD is encoded as K23.7. When operating at 8.0 GT/s or above: PAD is encoded as F7h.
2	Lane Number within Link. When operating at 2.5 or 5.0 GT/s: 0-31, PAD. PAD is encoded as K23.7. When operating at 8.0 GT/s or above: 0-31, PAD. PAD is encoded as F7h.
3	N_FTS. The number of Fast Training Sequences required by the Receiver: 0-255.
4	Data Rate Identifier Bit 0 – Reserved Bit 1 – 2.5 GT/s Data Rate Supported. Must be set to 1b. Bit 2 – 5.0 GT/s Data Rate Supported. Must be set to 1b if Bit 3 is 1b. Bit 3 – 8.0 GT/s Data Rate Supported. Bit 4:5 – Reserved. Bit 6 – Autonomous Change/Selectable De-emphasis. Downstream Ports: This bit is defined for use in the following LTSSM states: Polling.Active, Configuration.LinkWidth.Start, and Loopback.Entry. In all other LTSSM states, it is Reserved. Upstream Ports: This bit is defined for use in the following LTSSM states: Polling.Active, Configuration, Recovery, and Loopback.Entry. In all other LTSSM states, it is Reserved. Bit 7 – speed_change. This bit can be set to 1b only in the Recovery.RcvrLock LTSSM state. In all other LTSSM states, it is Reserved.
5	Training Control Bit 0 – Hot Reset Bit 0 = 0b, De-assert Bit 0 = 1b, Assert Bit 1 – Disable Link Bit 1 = 0b, De-assert Bit 1 = 1b, Assert Bit 2 – Loopback Bit 2 = 0b, De-assert Bit 2 = 1b, Assert Bit 3 – Disable Scrambling in 2.5 GT/s and 5.0 GT/s data rates; Reserved in other data rates Bit 3 = 0b, De-assert Bit 3 = 1b, Assert Bit 4 – Compliance Receive Bit 4 = 0b, De-assert Bit 4 = 1b, Assert Ports that support 5.0 GT/s and above data rate(s) must implement the Compliance Receive bit. Ports that support only 2.5 GT/s data rate may optionally implement the Compliance Receive bit. If not implemented, the bit is Reserved. Bit 5:7 – Reserved

Формирование пакетов DLLP и TLP уровней



Пример передачи пакетов на PHY уровне



Инициализация физического соединения

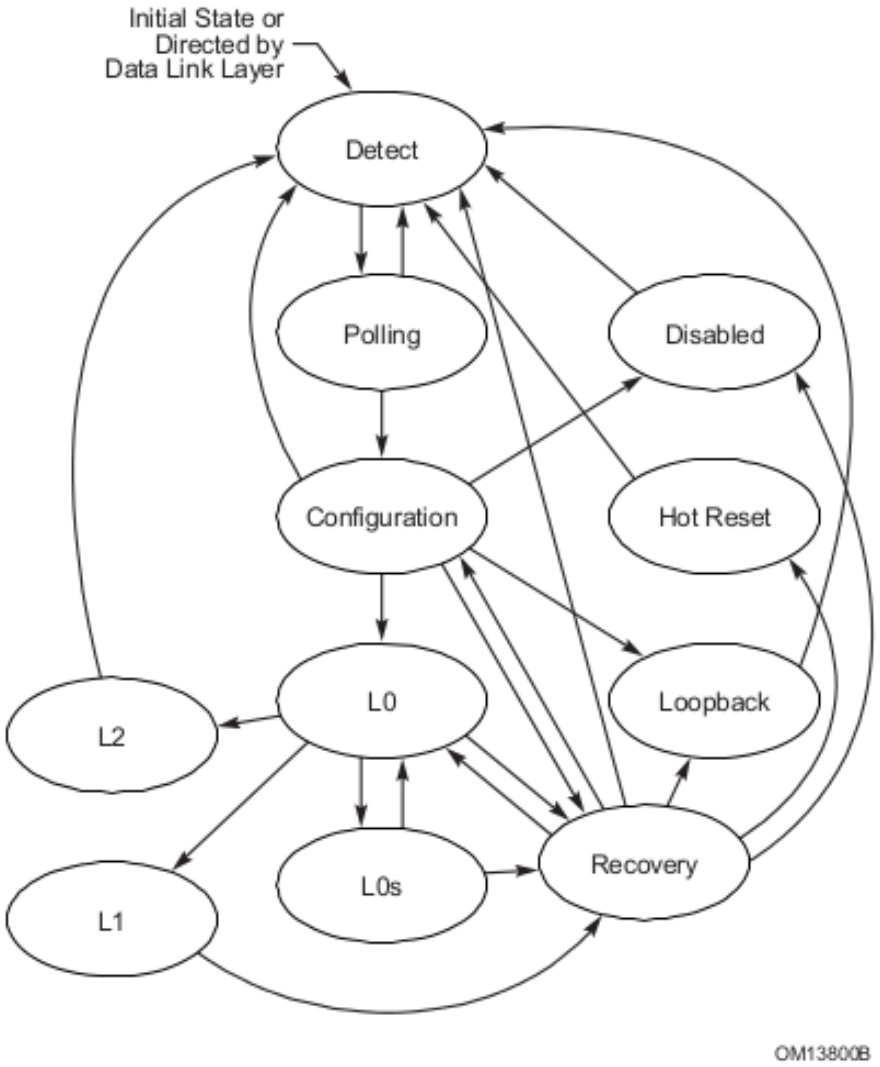
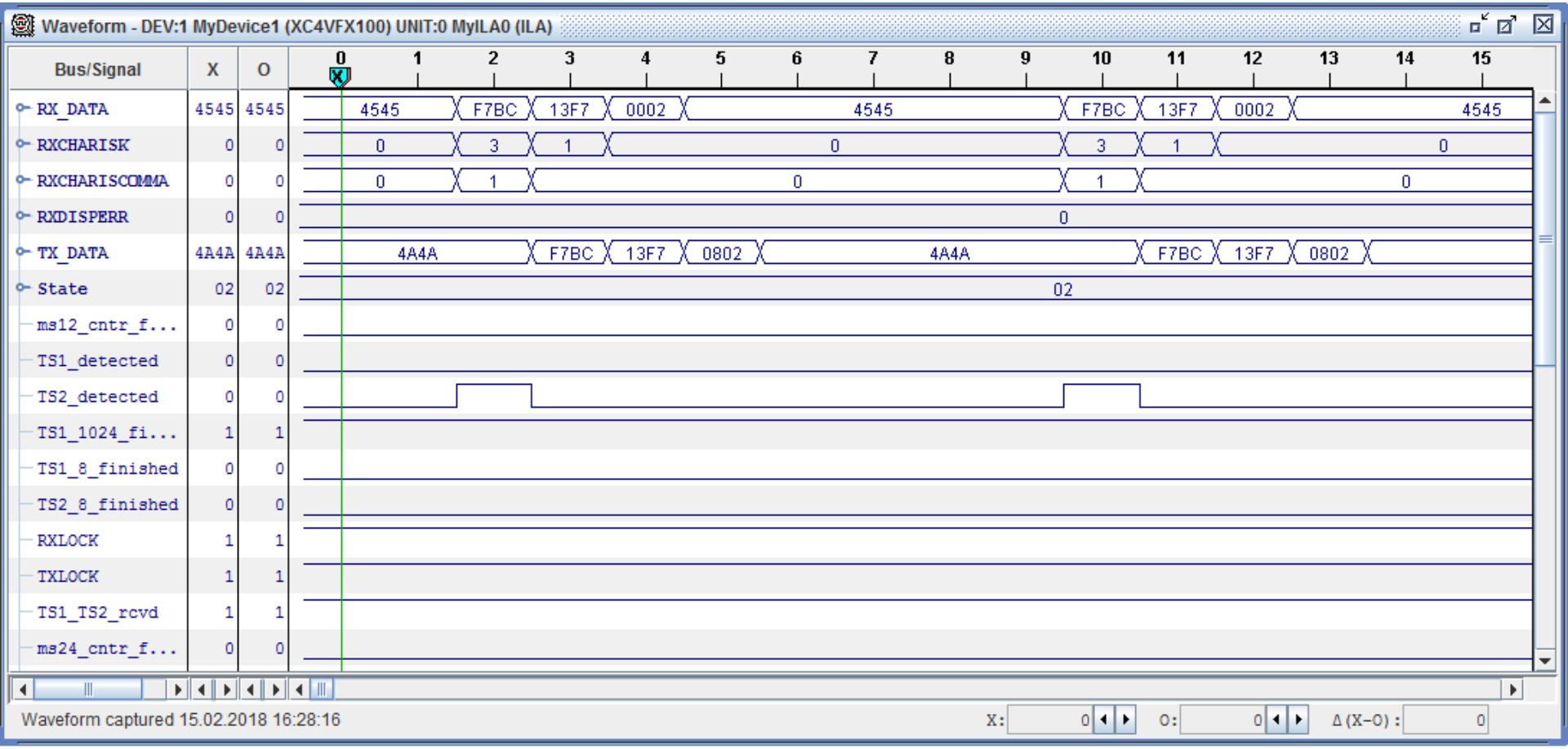
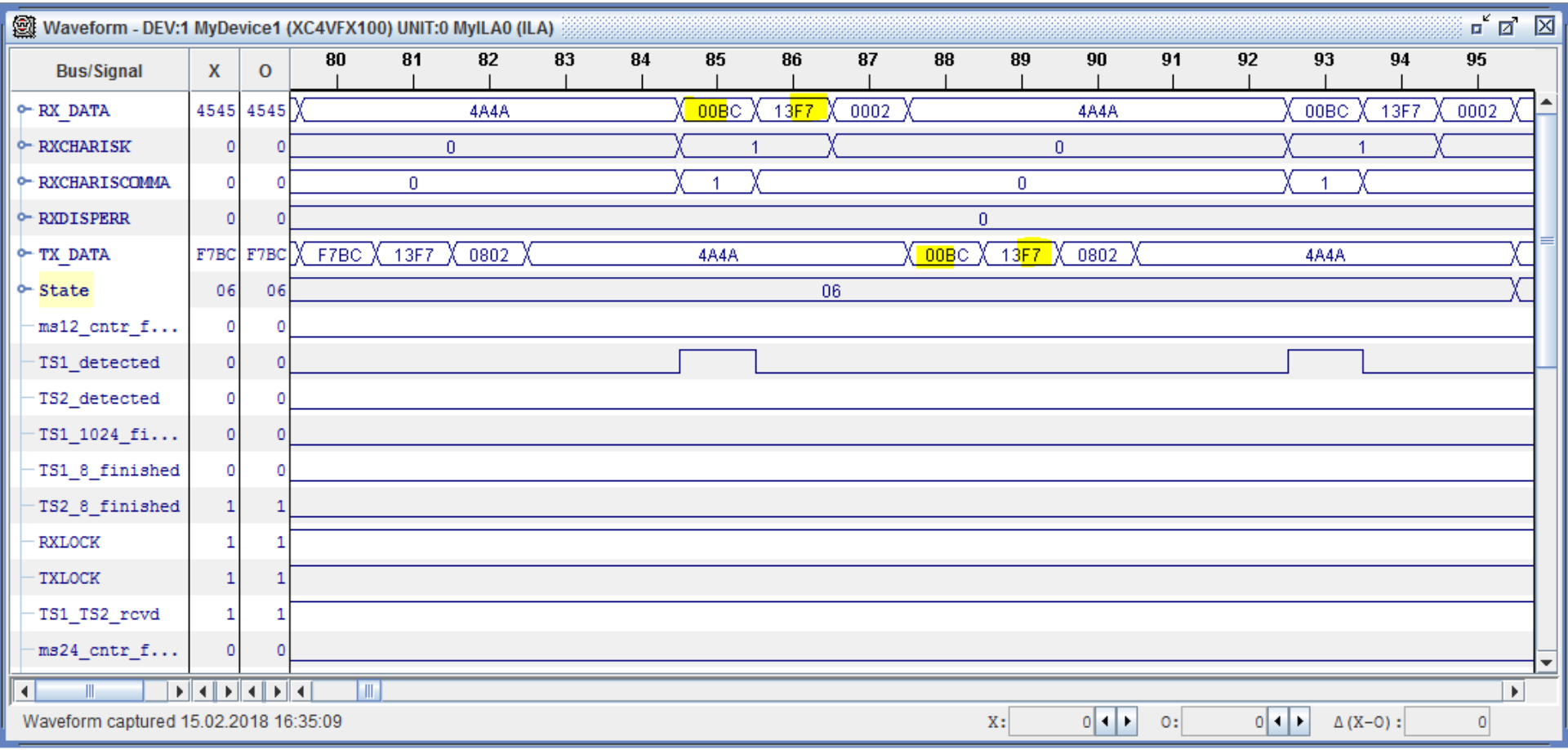


Figure 4-22: Main State Diagram for Link Training and Status State Machine

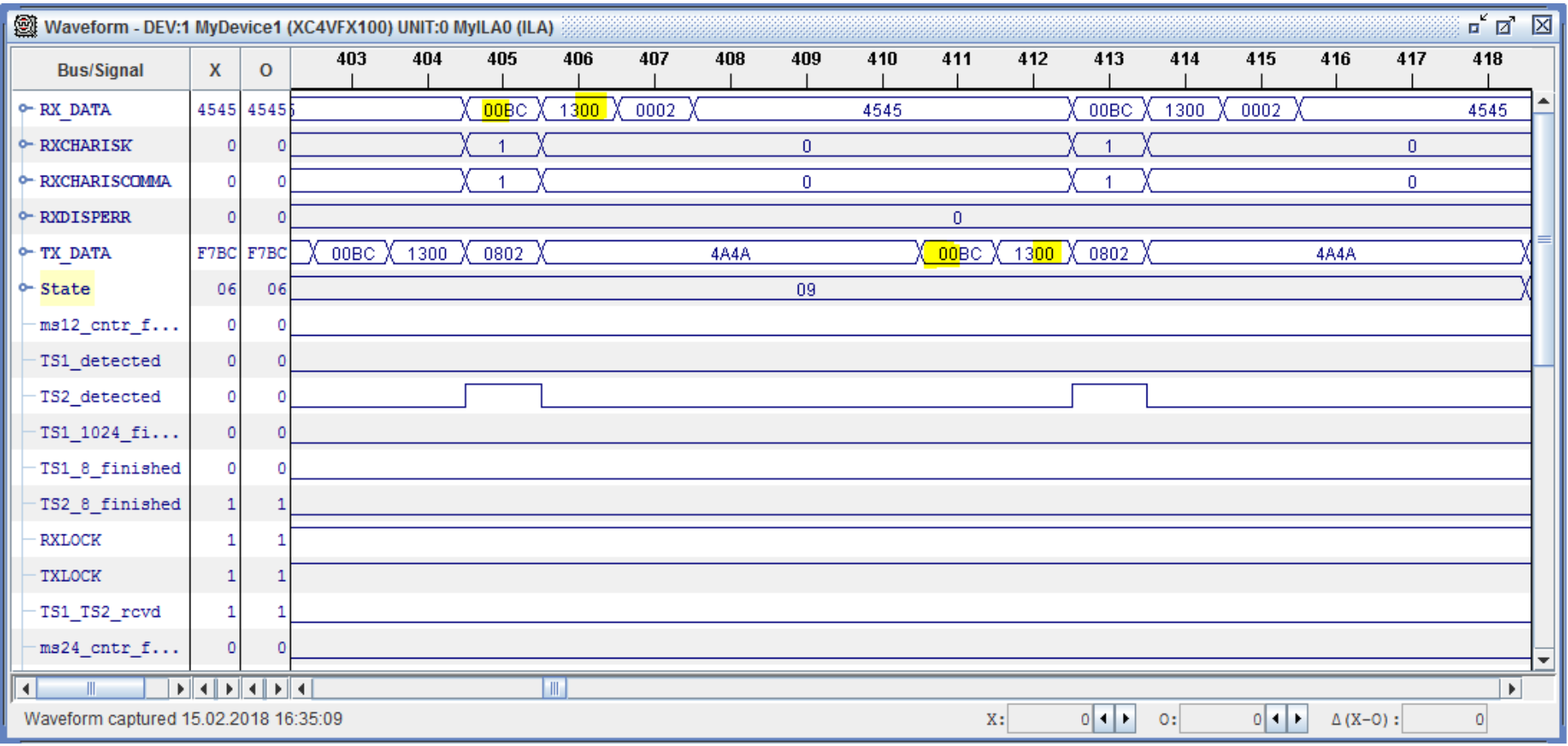
Опрос линий (POLLING_ACTIVE)



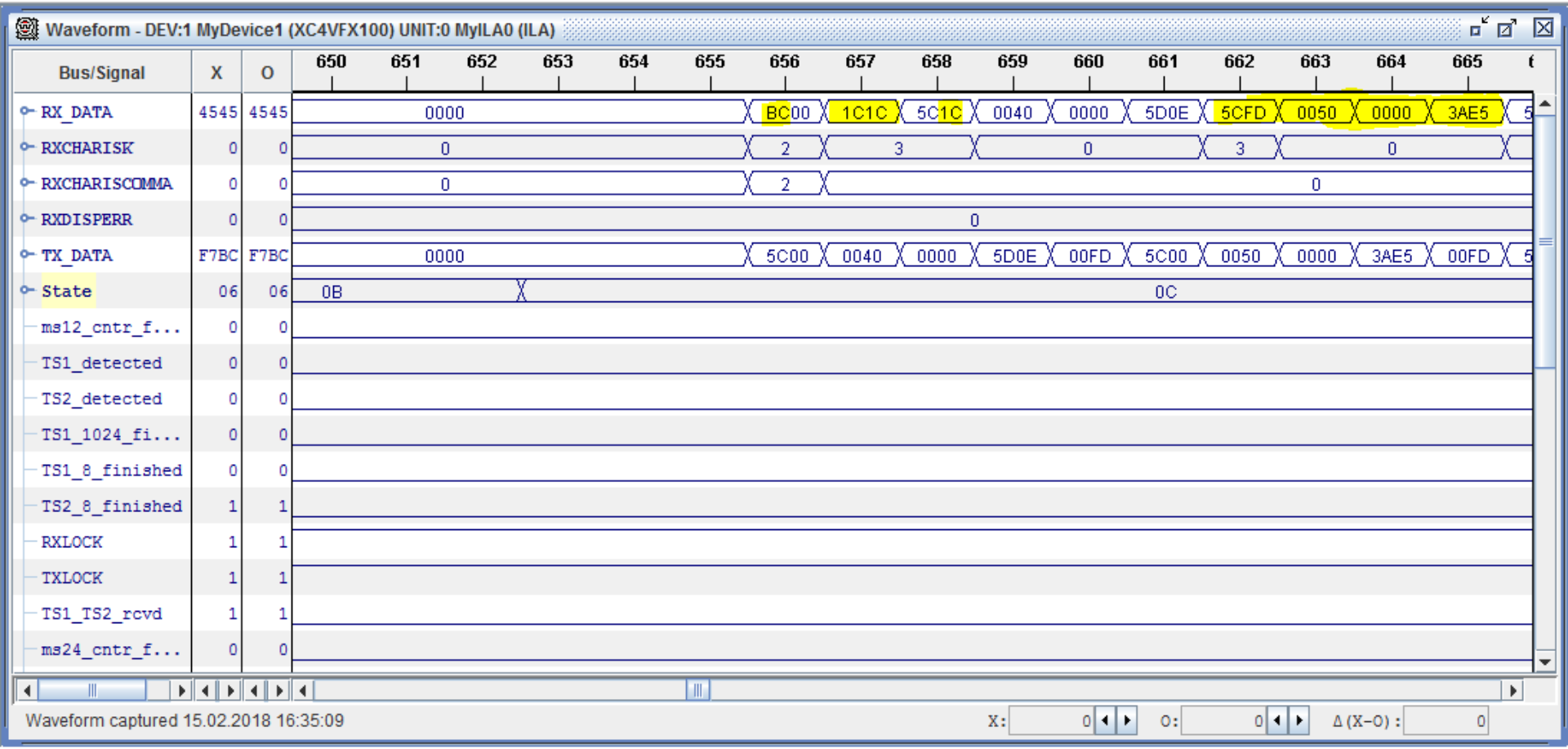
Определение разрядности линка (CONFIGURATION_LINKWIDTH_START)



Определение номеров линий (CONFIGURATION_LANENUM_ACCEPT)

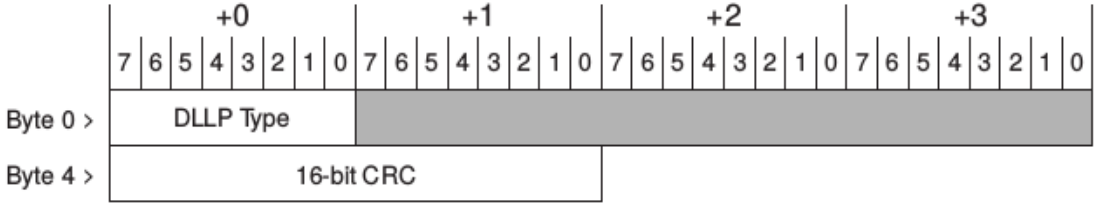


Готовность к приему и передаче (L0)

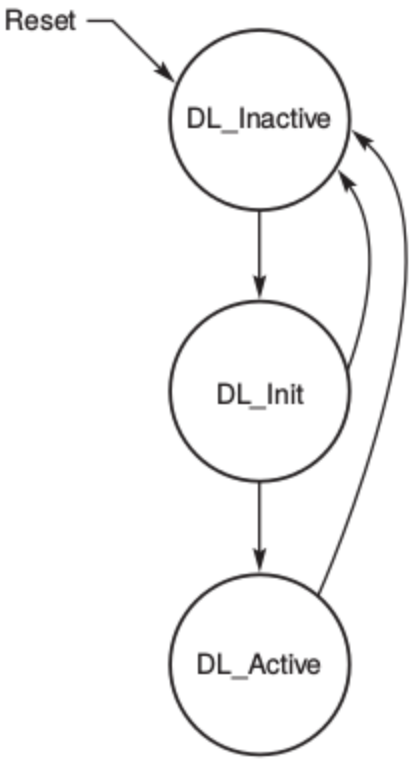


Data Link Layer Protocol

Формат Data Link Layer Packet



Состояния DLL



Типы пакетов DLLP

Encodings	DLLP Type
0000 0000	Ack
0001 0000	Nak
0010 0000	PM_Enter_L1
0010 0001	PM_Enter_L23
0010 0011	PM_Active_State_Request_L1
0010 0100	PM_Request_Ack
0011 0000	Vendor Specific – Not used in normal operation
0100 $0v_2v_1v_0$	InitFC1-P (v[2:0] specifies Virtual Channel)
0101 $0v_2v_1v_0$	InitFC1-NP
0110 $0v_2v_1v_0$	InitFC1-Cpl
1100 $0v_2v_1v_0$	InitFC2-P
1101 $0v_2v_1v_0$	InitFC2-NP
1110 $0v_2v_1v_0$	InitFC2-Cpl
1000 $0v_2v_1v_0$	UpdateFC-P
1001 $0v_2v_1v_0$	UpdateFC-NP
1010 $0v_2v_1v_0$	UpdateFC-Cpl
All other encodings	Reserved

Форматы DLLP пакетов

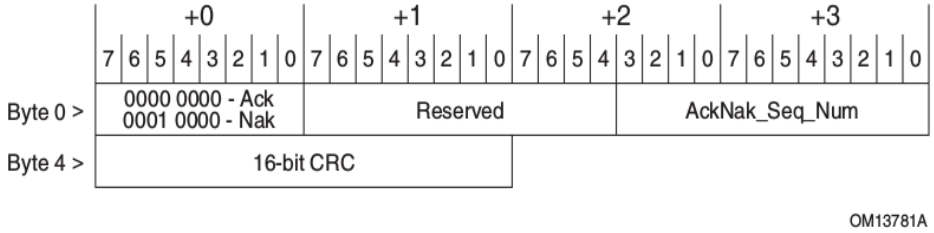


Figure 3-6: Data Link Layer Packet Format for Ack and Nak

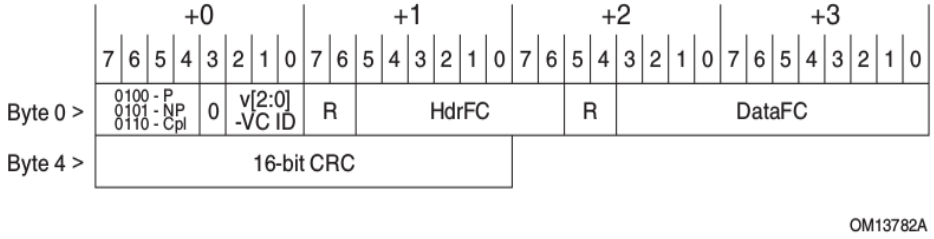
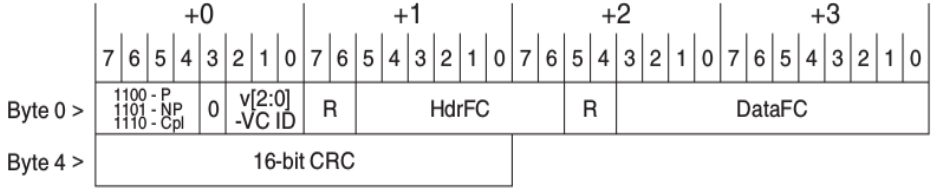


Figure 3-7: Data Link Layer Packet Format for InitFC1



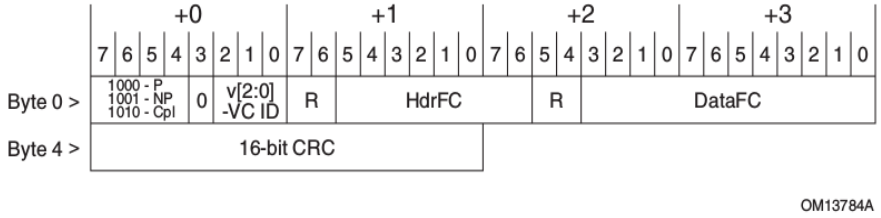


Figure 3-9: Data Link Layer Packet Format for UpdateFC

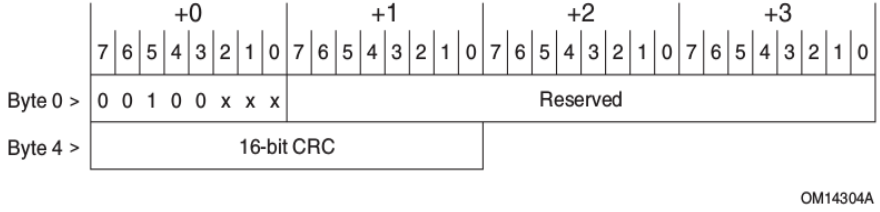
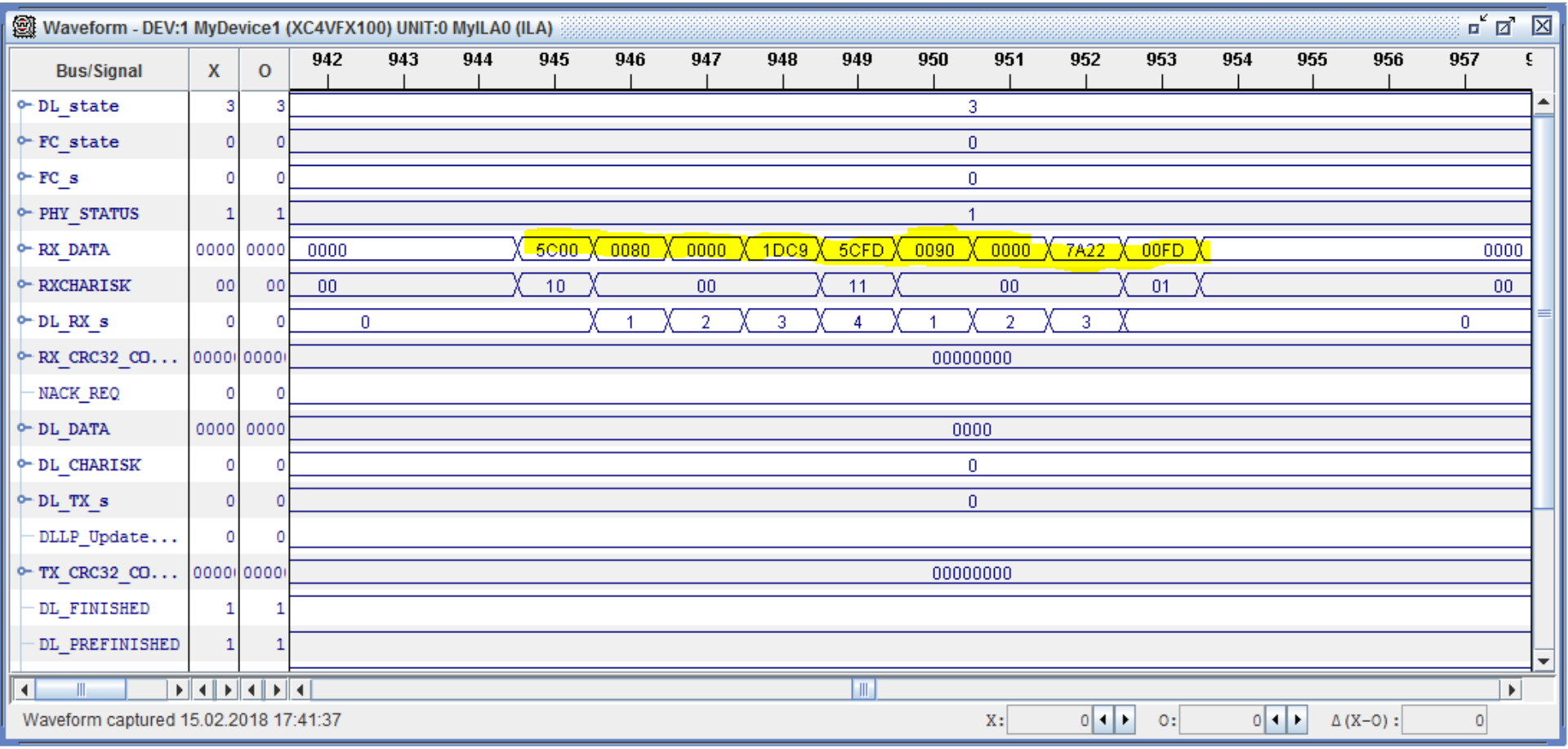
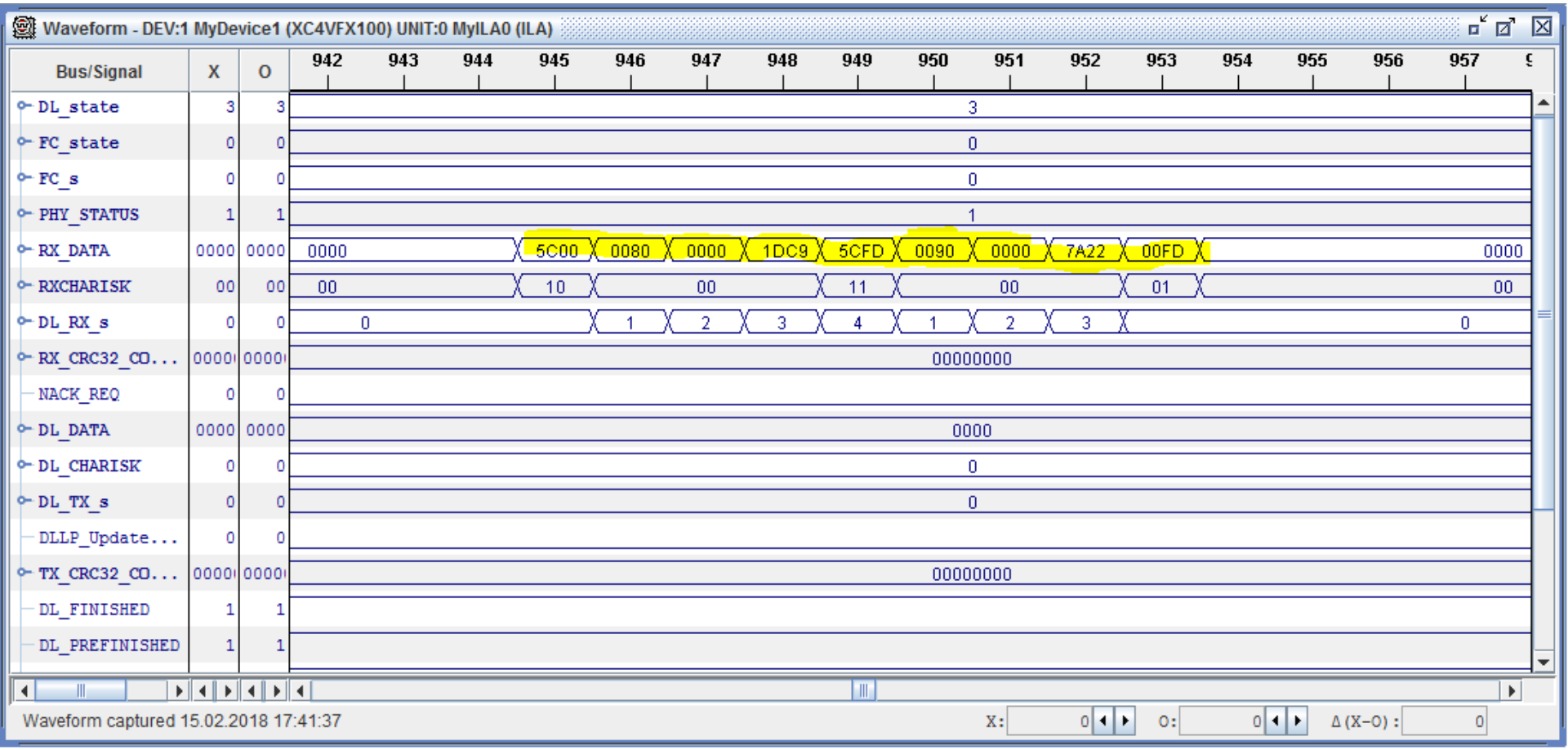


Figure 3-10: PM Data Link Layer Packet Format

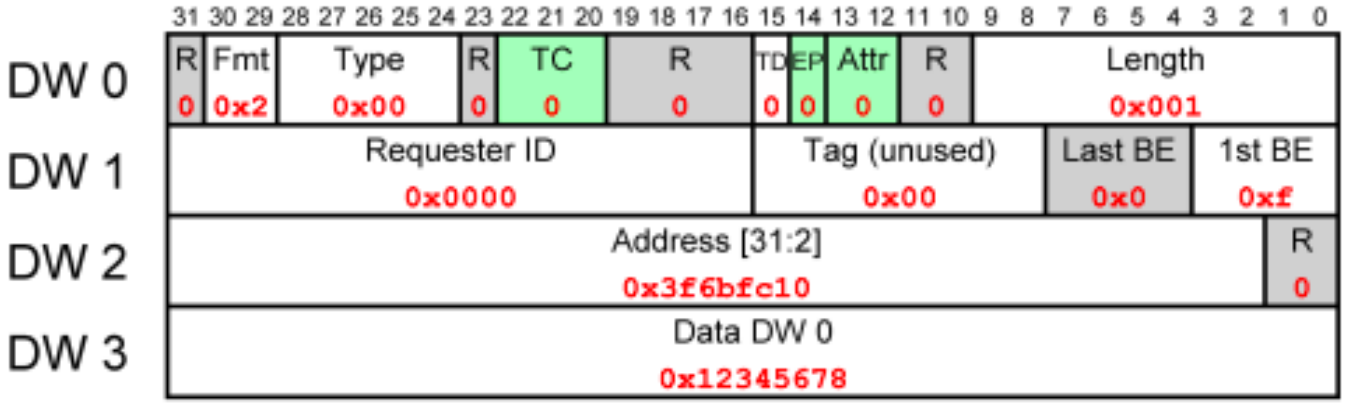
Получение кредитов FC1, FC2, UpdateFC по DLLP



Передача кредитов FC1, FC2, UpdateFC по DLLP

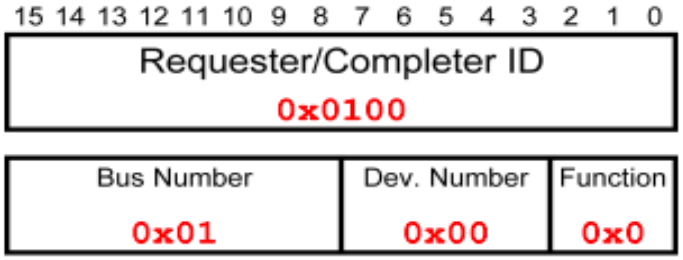


Пример пакета TLP записи*



Значение 0x12345678 читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Формат идентификатора устройства



(*)
- Серые поля зарезервированы (игнорируются получателем).
- Зеленые поля могут иметь ненулевые значения, но используются редко.
- Значения определенного пакета помечаются красным цветом.

Поля заголовка пакета

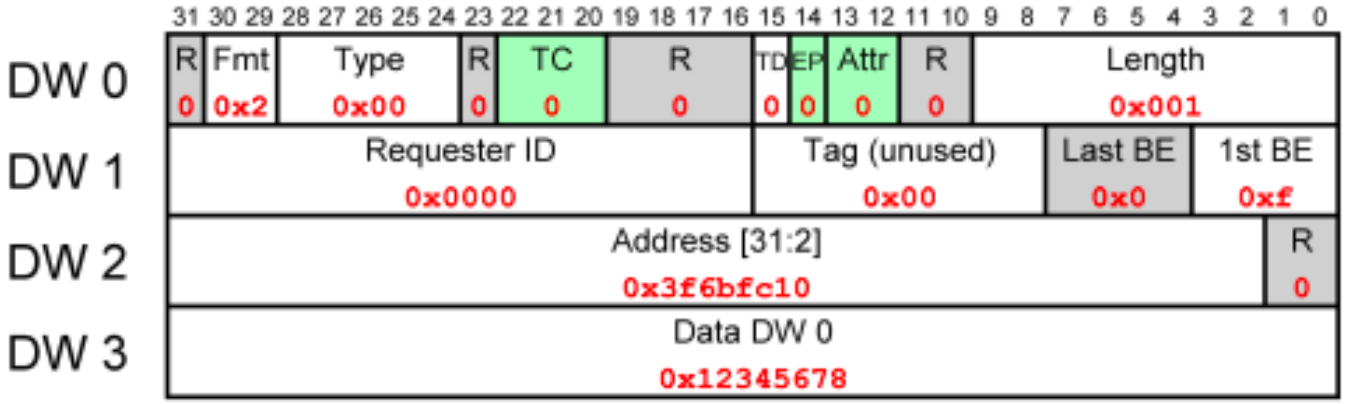
- Формат пакета
- Тип пакета
- Длина для всех связанных данных
- Описатель транзакций, включая:
  - идентификатор транзакции
  - Атрибуты
  - класс трафика
- Информация об адресе/маршрутизации
- Разрешенние байт
- Кодировка сообщения
- Состояние завершения

Типы транзакций

TLP Type	Fmt [2:0][2] (b)	Type [4:0] (b)	Description
MRd	000 001	0 0000	Memory Read Request
MRdLk	000 001	0 0001	Memory Read Request-Locked
MWr	010 011	0 0000	Memory Write Request
IORd	000	0 0010	I/O Read Request
IOWr	010	0 0010	I/O Write Request
CfgRd0	000	0 0100	Configuration Read Type 0
CfgWr0	010	0 0100	Configuration Write Type 0
CfgRd1	000	0 0101	Configuration Read Type 1
CfgWr1	010	0 0101	Configuration Write Type 1
TCfgRd	000	1 1011	Deprecated TLP Type[3]
TCfgWr	010	1 1011	Deprecated TLP Type[3]
Msg	001	$1\ 0r_2r_1r_0$	Message Request – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-18).
MsgD	011	$1\ 0r_2r_1r_0$	Message Request with data payload – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-18).
Cpl	000	0 1010	Completion without Data – Used for I/O and Configuration Write Completions with any Completion Status. Also used for AtomicOp Completions and Read Completions (I/O, Configuration, or Memory) with Completion Status other than Successful Completion.
CplD	010	0 1010	Completion with Data – Used for Memory, I/O, and Configuration Read Completions. Also used for AtomicOp Completions.
CplLk	000	0 1011	Completion for Locked Memory Read without Data – Used only in error case.
CplDLk	010	0 1011	Completion for Locked Memory Read – otherwise like CplD.

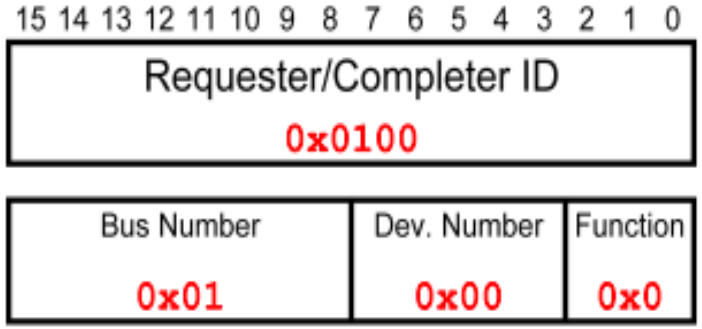
TLP Type	Fmt [2:0][2] (b)	Type [4:0] (b)	Description
FetchAdd	010 011	0 1100	Fetch and Add AtomicOp Request
Swap	010 011	0 1101	Unconditional Swap AtomicOp Request
CAS	010 011	0 1110	Compare and Swap AtomicOp Request
LPrfx	100	$0L_3L_2L_1L_0$	Local TLP Prefix – The sub-field L[3:0] specifies the Local TLP Prefix type (see Table 2-30).
EPrfx	100	$1E_3E_2E_1E_0$	End-End TLP Prefix – The sub-field E[3:0] specifies the End-End TLP Prefix type (see Table 2-31).
			All encodings not shown above are Reserved (see Section 2.3).

Пример пакета TLP записи

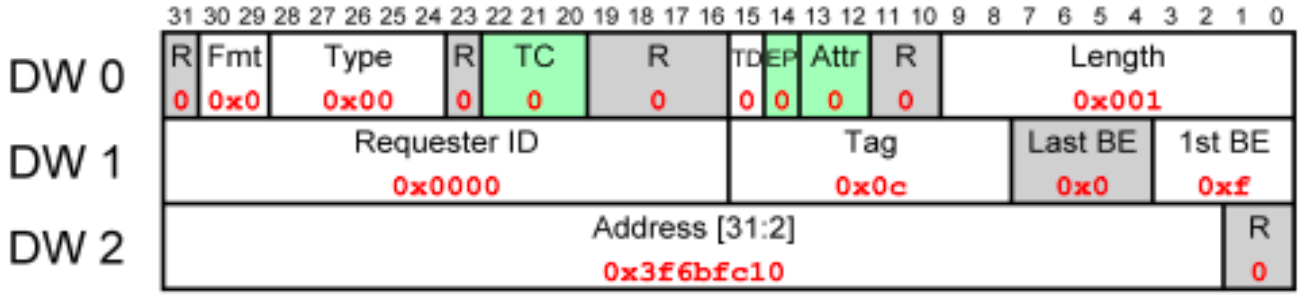


Значение 0x12345678 читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Формат идентификатора устройства

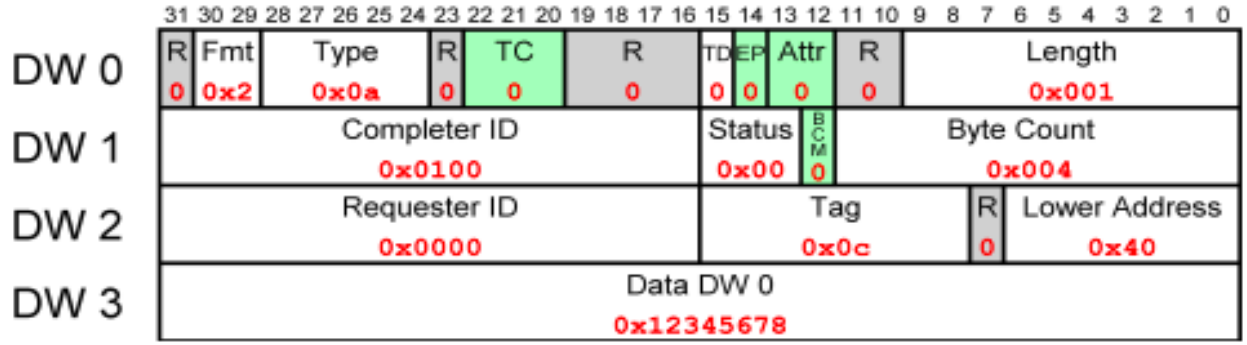


Пример пакета TLP чтения



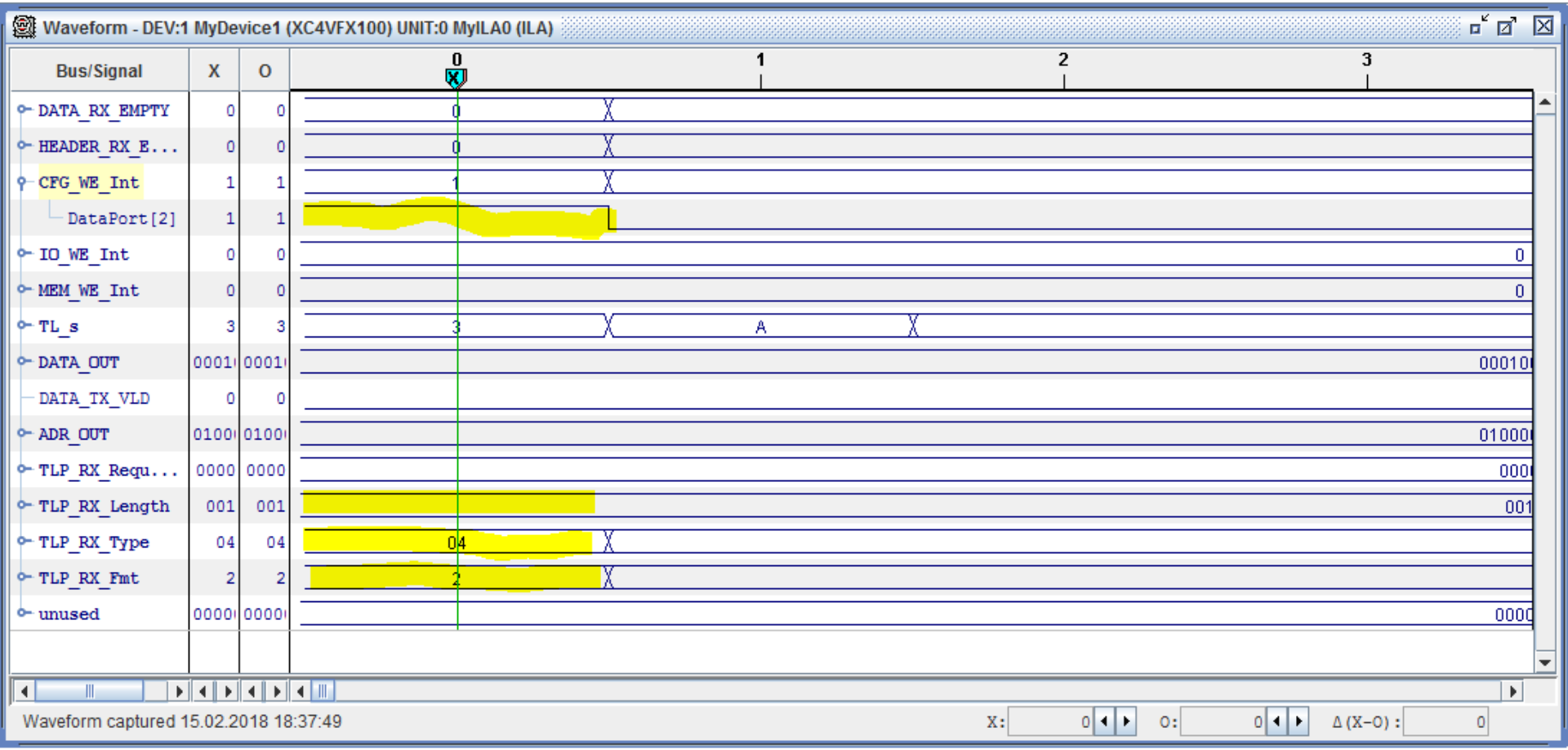
Значение читается по адресу 0xfdaff040 (2 LSB бита не передаются)

Пример пакета завершения (completion)



Значение 0x12345678 передаются инициатору

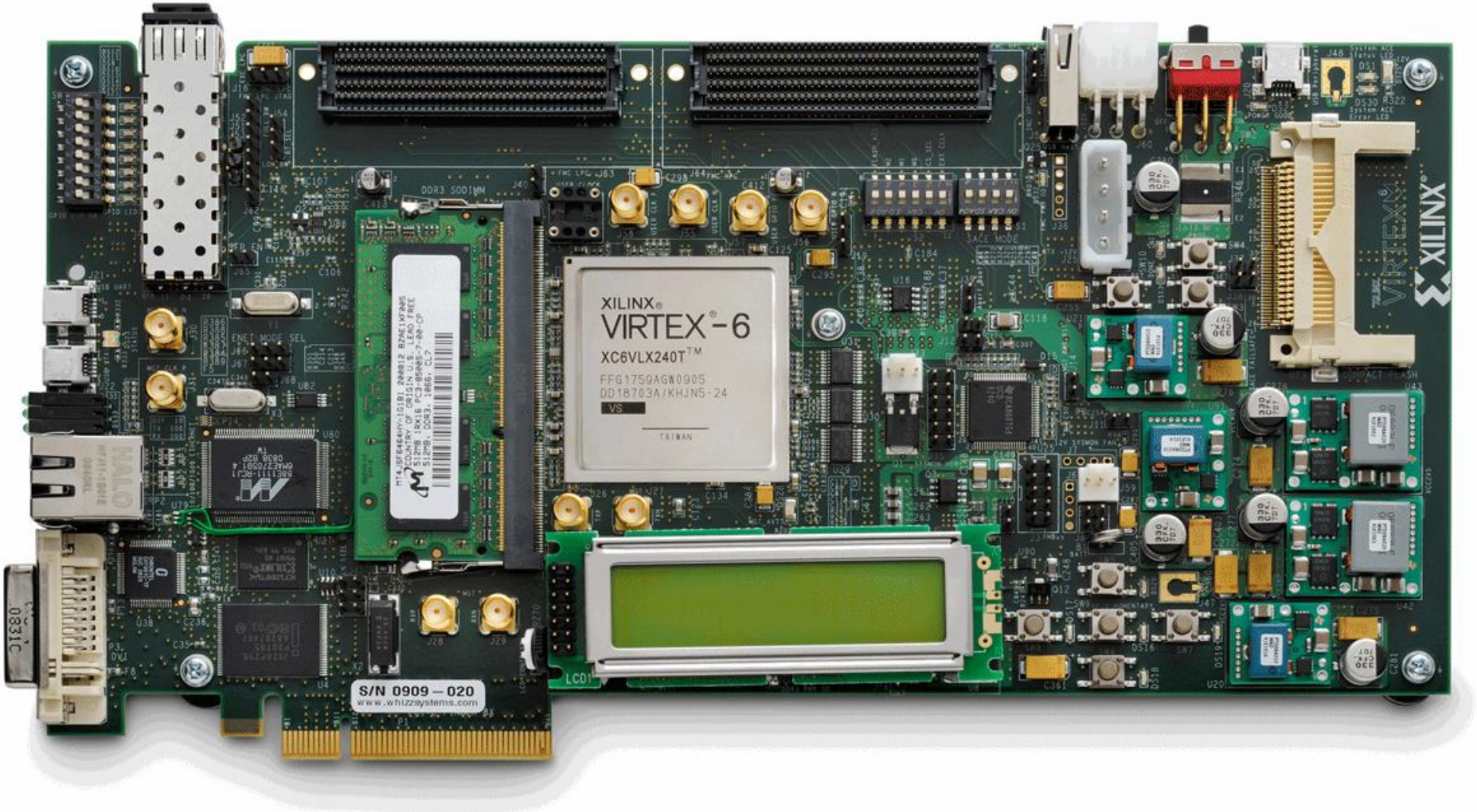
Прием и декодирование пакетов TLP



Сигналы разъема PCI Express

Pin	Side B	Side A	Description
1	+12 V	PRSNT1#	Must connect to farthest PRSNT2# pin
2	+12 V	+12 V	Main power pins
3	+12 V	+12 V	
4	Ground	Ground	
5	SMCLK	TCK	SMBus and JTAG port pins
6	SMDAT	TDI	
7	Ground	TDO	
8	+3.3 V	TMS	
9	TRST#	+3.3 V	
10	+3.3 V aux	+3.3 V	Standby power
11	WAKE#	PERST#	Link reactivation; fundamental reset
Key notch			
12	CLKREQ#	Ground	Request running clock
13	Ground	REFCLK+	Reference clock differential pair
14	HSOp(0)	REFCLK−	Lane 0 transmit data, + and −
15	HSOn(0)	Ground	
16	Ground	HSIp(0)	Lane 0 receive data, + and −
17	PRSNT2#	HSIn(0)	
18	Ground	Ground	
PCI Express ×1 cards end at pin 18			

Плата ML605


XILINX
VIRTEX-6
XC6VLX240T
FFG1759AGW0905
DD18703A/KHJN5-24
TAIWAN
S/N 0909 – 020
www.whizzsystems.com
DDR3 SODIMM
LCD1