
PCI EXPRESS: ВЗГЛЯД DIY-РАЗРАБОТЧИКА И ХАКЕРА

ДМИТРИЙ ОЛЕКСЮК



ОБЩИЕ ДЕТАЛИ

- Высокоскоростная последовательная шина разработанная в качестве замены для PCI
- Современный компьютер представляет собой скорее сеть из специализированных компьютеров в одном устройстве:
 - CPU и GPU
 - Intel Management Engine / AMD Platform Security Processor
 - Сетевой контроллер (Ethernet, 802.11, Bluetooth)
 - USB, SATA и другие контролеры
- PCI Express – протокол этой “сети” включающий в себя как физический так и более высокие уровни
- Занимает особое место в модели угроз платформы

О ЧЕМ ДОКЛАД

- Обзор архитектуры PCI Express
- PCI-E шина с точки зрения безопасности платформы
- Использование FPGA для работы с PCI-E шиной на уровне транзакций
- DMA атаки через PCI-E шину на практике
- Полезность и бесполезность IOMMU
- DMA атаки в контексте platform / firmware security

СТАНДАРТ PCI EXPRESS

- Версия 1.0а вышла в 2003 году и предлагала скорость передачи данных через один lane равную 2.5 GT/s, версия 4.0 со скоростью 16 GT/s вышла в 2011 году
 - <https://pcisig.com/specifications/pciexpress/>
- Широко применяется как шина передачи данных и стандарт плат расширения
 - Суперкомпьютеры, сервера, рабочие станции, ноутбуки, мобильные устройства, промышленное оборудование и многое другое
- Иногда ошибочно считается улучшенной версией PCI однако на самом деле является новой шиной с принципиально отличающейся архитектурой
 - PCI Express имеет гораздо больше схожестей с Ethernet чем с PCI
 - Многие платформы (напр. Intel x86) предлагают частичную совместимость между программным интерфейсом PCI и PCI-E

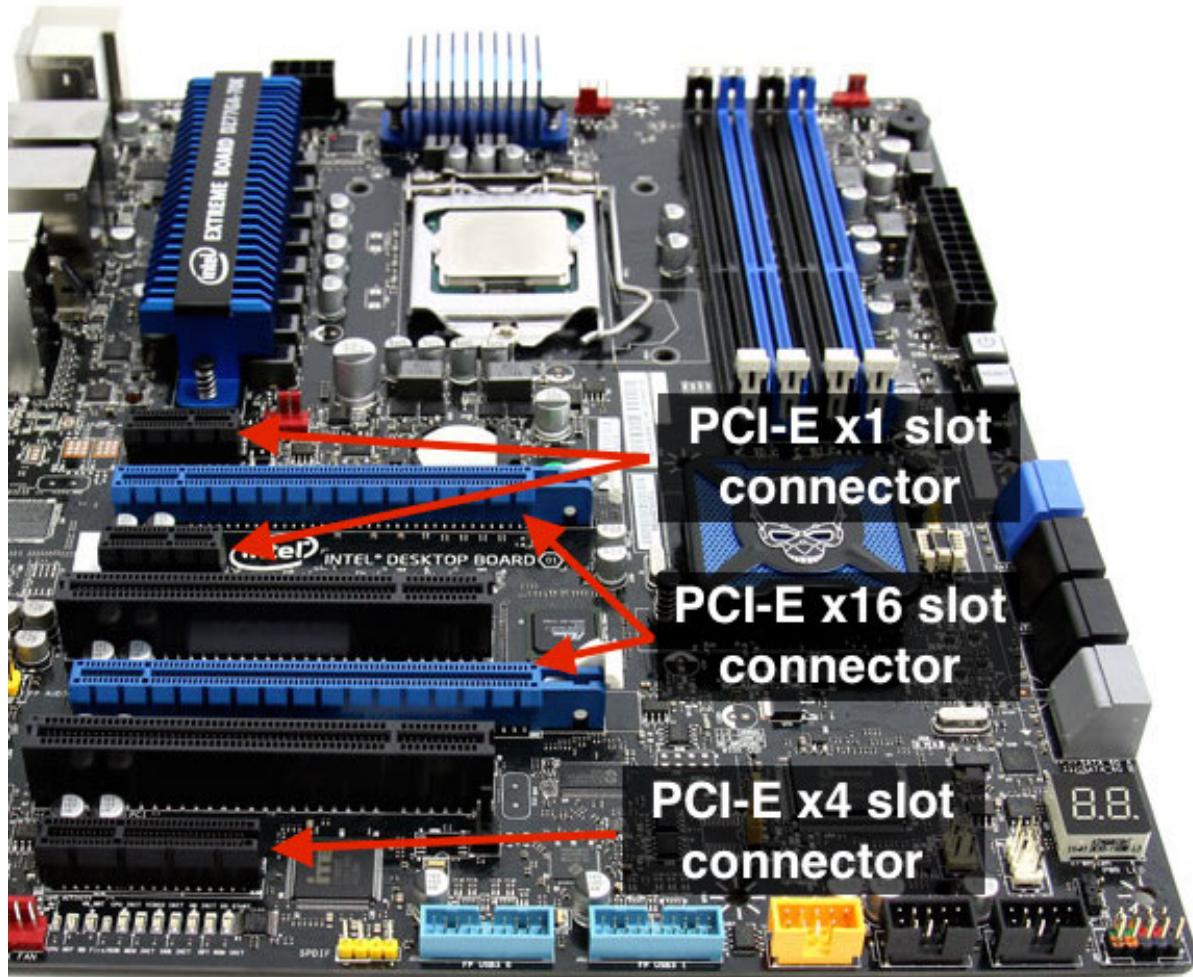
PCI EXPRESS И БЕЗОПАСНОСТЬ

- PCI-E устройства имеют прямой доступ к контроллеру памяти (DMA)
 - Возможность доступа к системной RAM в процессе работы платформы при отсутствии IOMMU
- PCI-E устройства как правило имеют специализированный CPU со своей собственной firmware которая находится в отдельном ROM и предусматривает механизмы программного обновления из ОС
 - Персистентный вредоносный код выживающий переустановку ОС, обход Secure Boot цепочки
 - Особо интересный вектор атак в сочетании с возможностью прямого доступа к RAM
- PCI-E устройства могут иметь беспроводные интерфейсы и сложную firmware которая реализует те или иные уровни стека протоколов Wi-Fi, Bluetooth, GSM, UMTS, LTE и других
 - Отличный удаленный вектор атаки с прямым доступом к RAM by design в качестве бонуса
 - Google Project Zero продемонстрировали удаленное исполнение произвольного кода в ядре Android благодаря уязвимости в firmware Wi-Fi/Bluetooth чипа от Broadcom в сочетании с DMA атакой, применимо к мобильным устройствам на любой платформе которые используют аналогичные чипы подключенные к SoC по PCI-E шине

PCI EXPRESS И БЕЗОПАСНОСТЬ

- PCI-E устройства (так же как и PCI) могут иметь option ROM
 - BIOS или UEFI читает option ROM в процессе загрузки платформы и исполняет находящийся там код драйвера PCI-E устройства
 - Стандарт UEFI актуальной версии подразумевает проверку цифровых подписей для DXE драйверов из option ROM при активном Secure Boot, в реальности компьютеры с включенным Secure Boot и не содержащей уязвимости UEFI совместимой firmware встречаются не часто
- PCI-E шина часто доступна в виде внешних портов (FireWire, Thunderbolt, ExpressCard, USB-C, док-станции и многое другое)
 - Атаки подразумевающие физический доступ
 - Атаки подразумевающие заражение firmware внешнего устройства с FireWire или Thunderbolt интерфейсом как вектор проникновения на другие компьютеры
 - Фorenзика: дамп RAM, извлечение ключей полнодискового шифрования, обход ограничений доступа на вход в систему, извлечение данных из заблокированных мобильных устройств

ТАКОЙ РАЗНЫЙ PCI EXPRESS

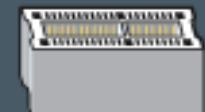


PCI Express Example Connectors

x1

BANDWIDTH

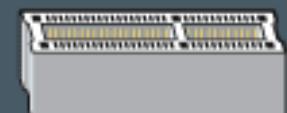
Single direction: 2.5 Gbps/200 MBps
Dual Directions: 5 Gbps/400 MBps



x4

BANDWIDTH

Single direction: 10 Gbps/800 MBps
Dual Directions: 20 Gbps/1.6 GBps



x8

BANDWIDTH

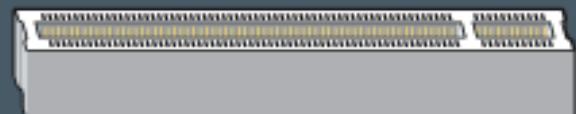
Single direction: 20 Gbps/1.6 GBps
Dual Directions: 40 Gbps/3.2 GBps



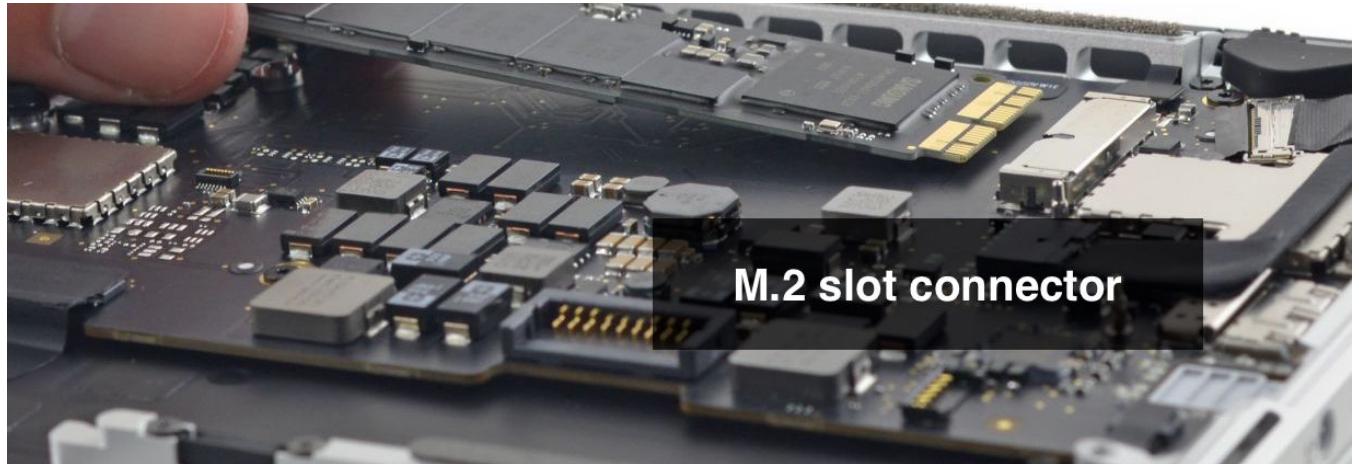
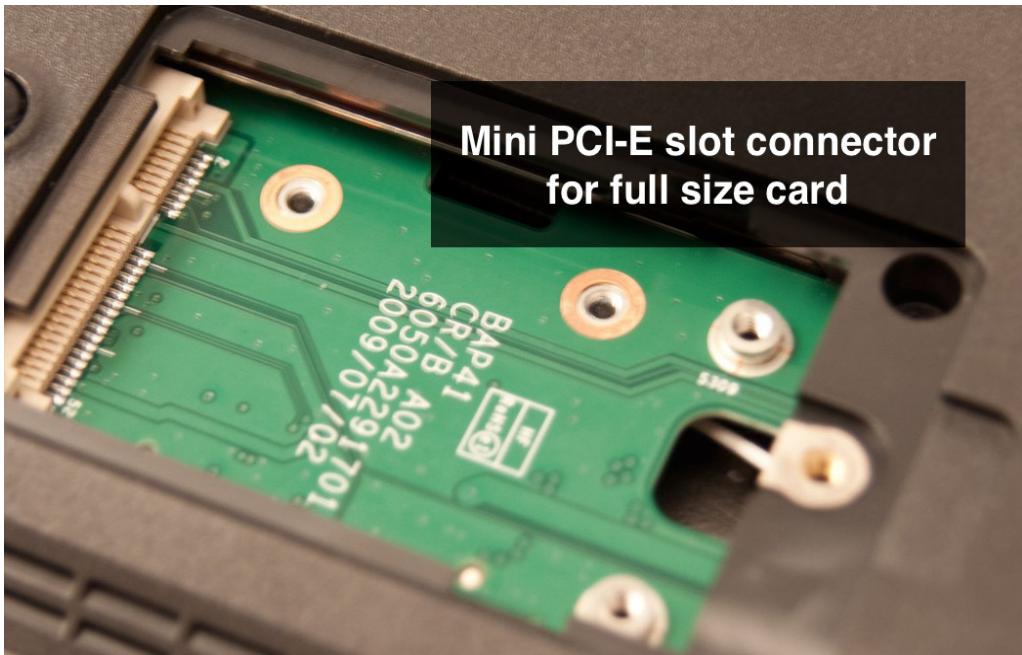
x16

BANDWIDTH

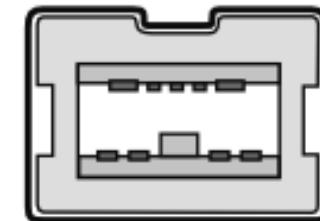
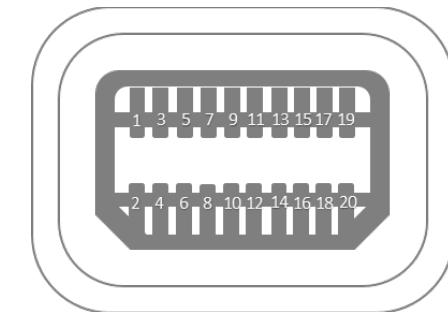
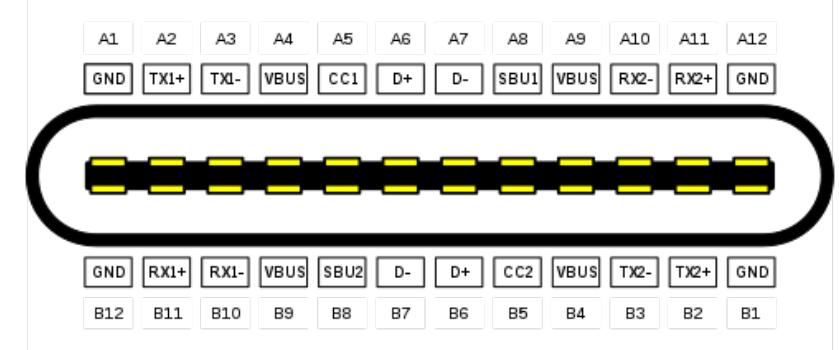
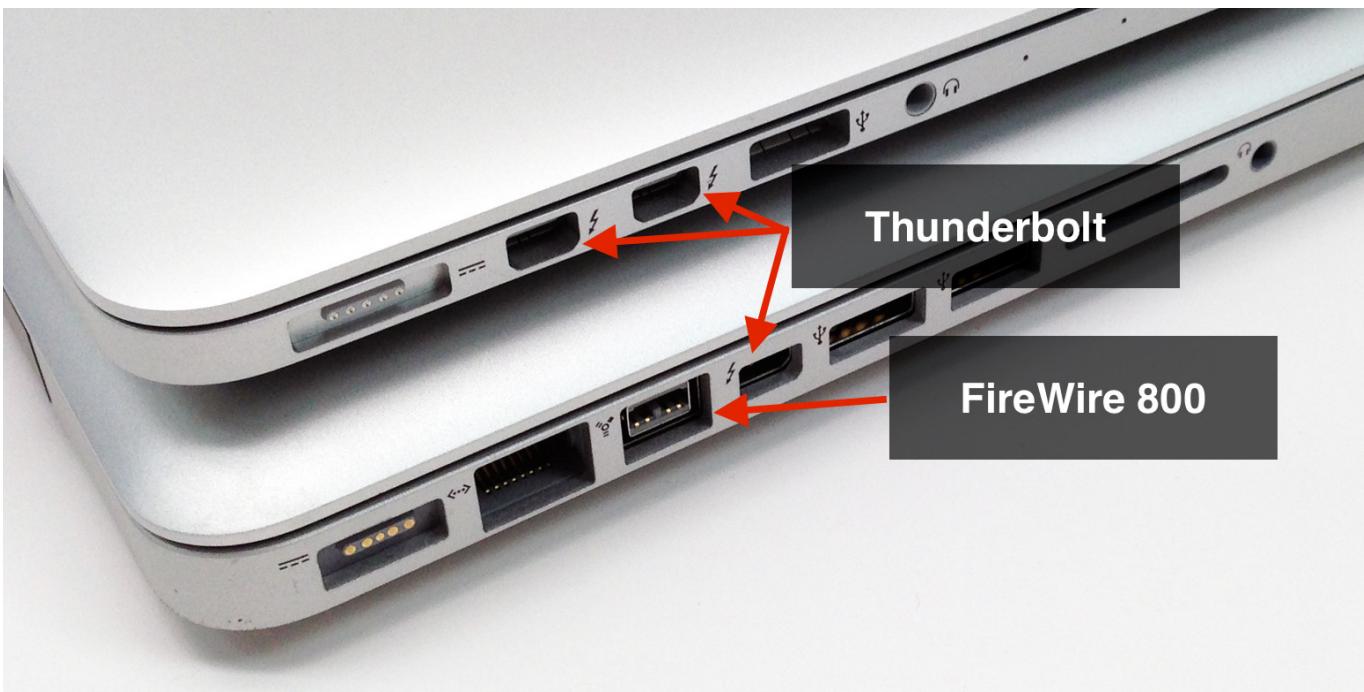
Single direction: 40 Gbps/3.2 GBps
Dual Directions: 80 Gbps/6.4 GBps



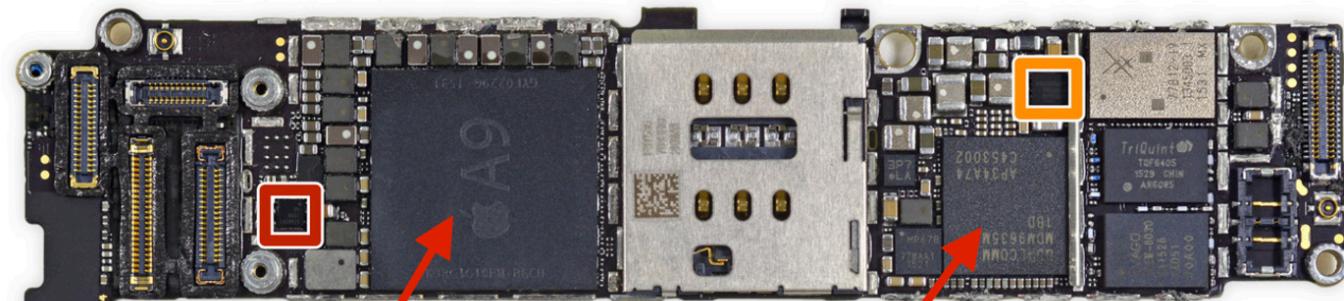
ТАКОЙ РАЗНЫЙ PCI EXPRESS



ТАКОЙ РАЗНЫЙ PCI EXPRESS



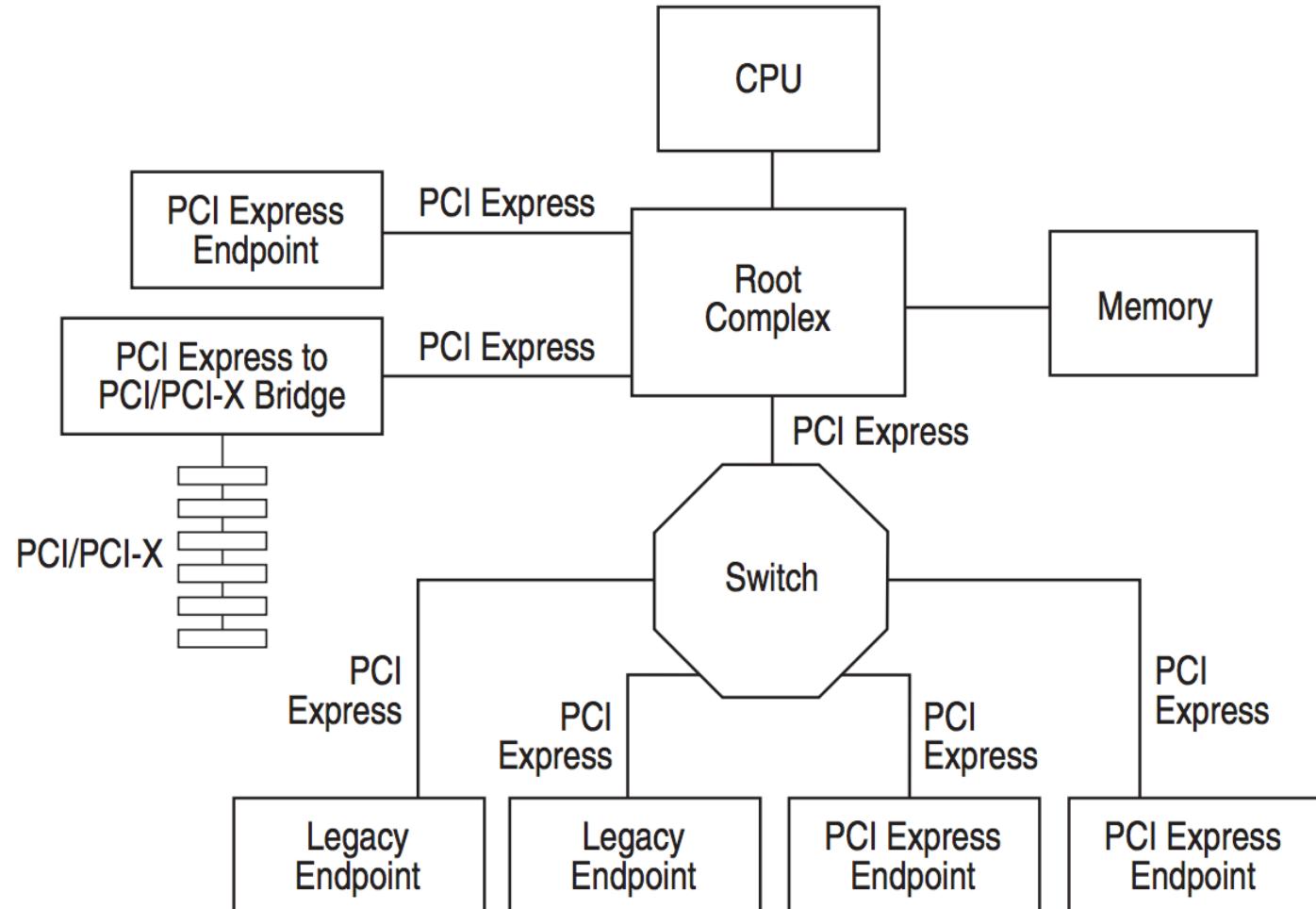
ТАКОЙ РАЗНЫЙ PCI EXPRESS



U_BB_RF	
MDM9635M	BGA
SYM 5 OF 8	
USB_PCIE	
LK RADIO_BB	PCIE_EP_REFCLK_P
	PCIE_EP_REFCLK_N
	PCIE_TX_P
	PCIE_TX_M
	PCIE_RX_P
	PCIE_RX_M
	PCIE_REXT
U10	PCIE0 AP TO BB REFCLK P
W10	PCIE0 AP TO BB REFCLK N
AA11	PCIE0 BB TO AP TX P
Y11	PCIE0 BB TO AP TX N
AA13	PCIE0 AP TO BB TX P
Y13	PCIE0 AP TO BB TX N
AA10	PCIE CAL RES
1R3302_RF	1.43K
1%	1/32W
MF	01005
2RADIO_BB	PLACE
PLACE	CLOSE
CLOSE	TO AA10

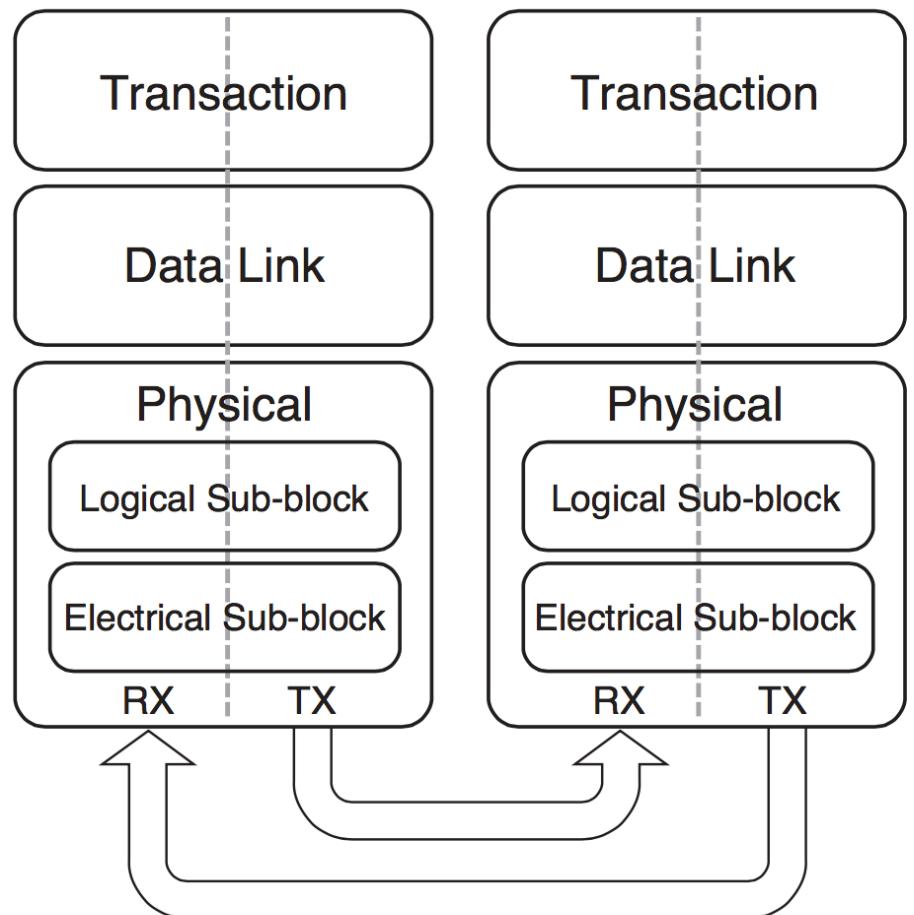
ТОПОЛОГИЯ PCI-E ШИНЫ

- Root Complex – особое PCI-E устройство соединяющее с шиной CPU и подсистему памяти
- Switch – маршрутизация пакетов транзакционного уровня между несколькими портами
- Endpoint – конечное устройство

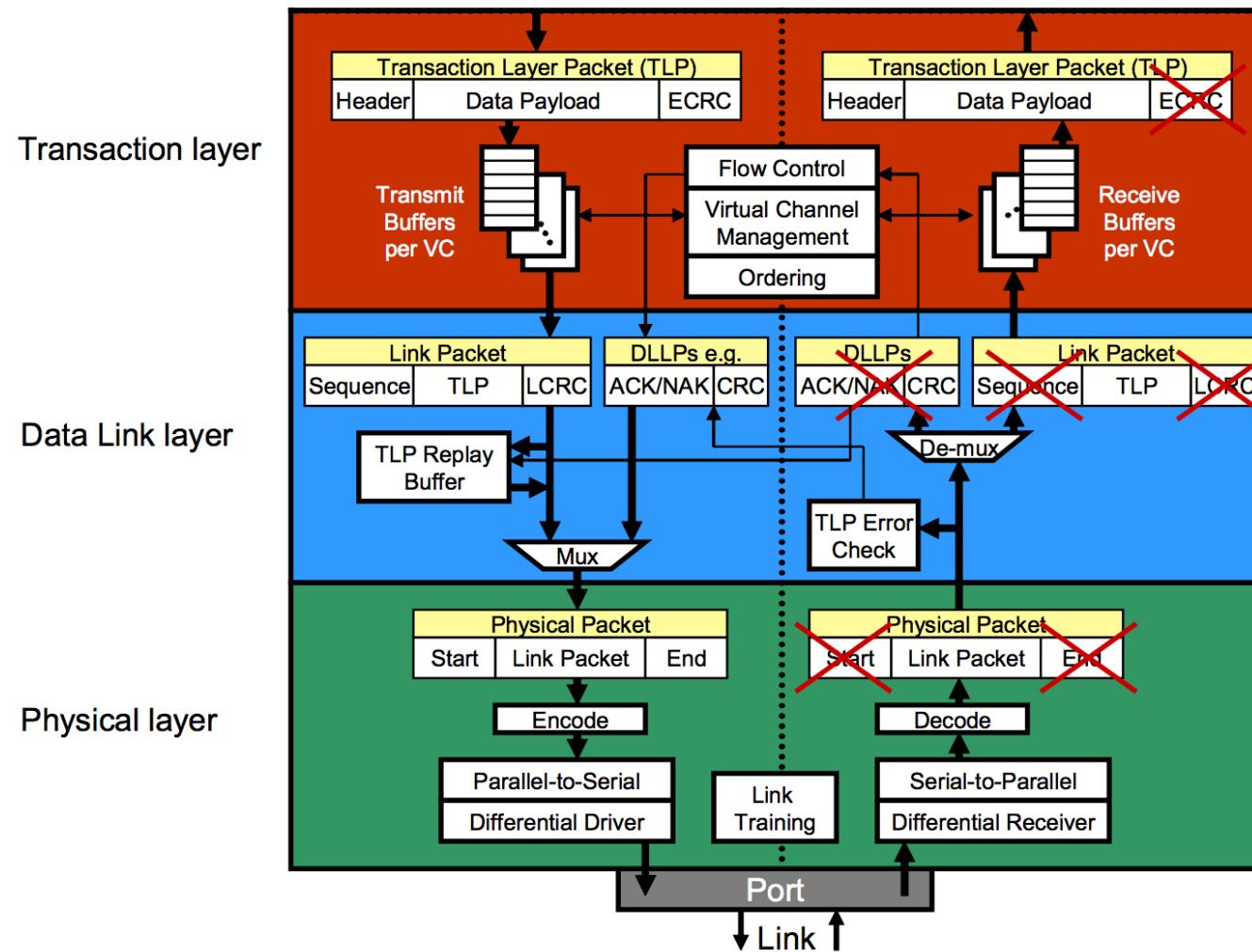


АРХИТЕКТУРА PCI-E

- Физический уровень – один или несколько lane каждого из которых это пара последовательных дифференциальных линий на скорости 2.5, 5, 8 или 16 GT/s
- Data link уровень – отвечает за контроль целостности данных (CRC, повторная передача, обработка ошибок) используя ACK/NAK протокол
- Transaction уровень – транзакции для доступа к памяти, пространству I/O и конфигурационному пространству PCI-E

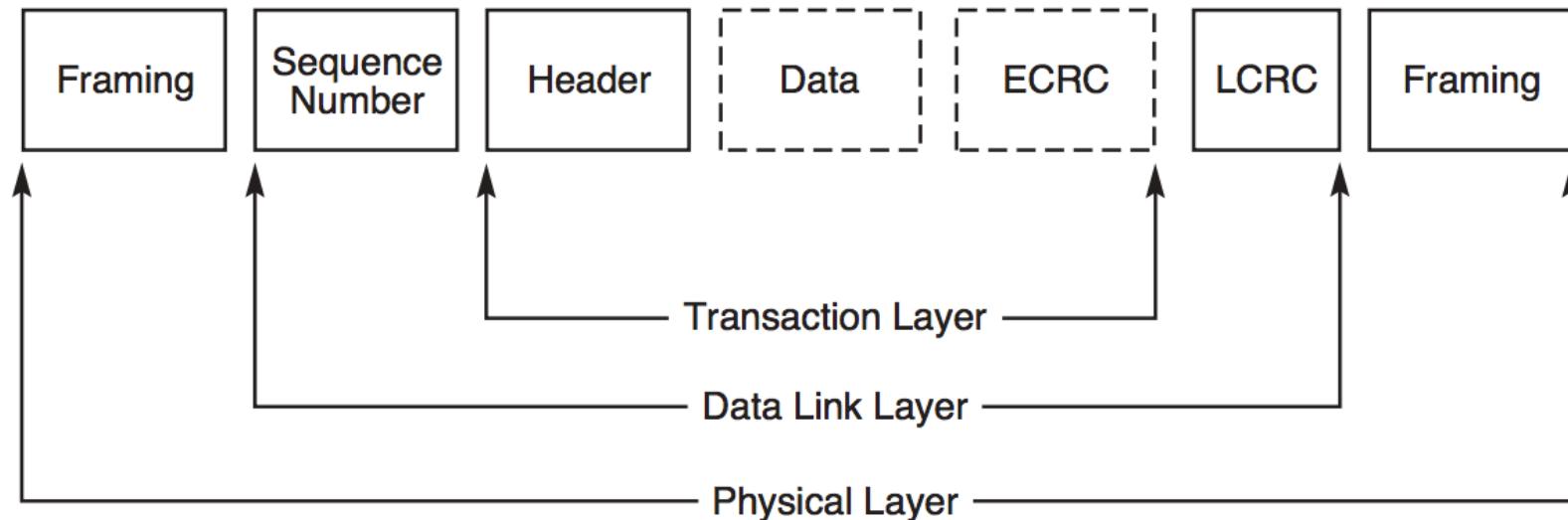


АРХИТЕКТУРА PCI-E

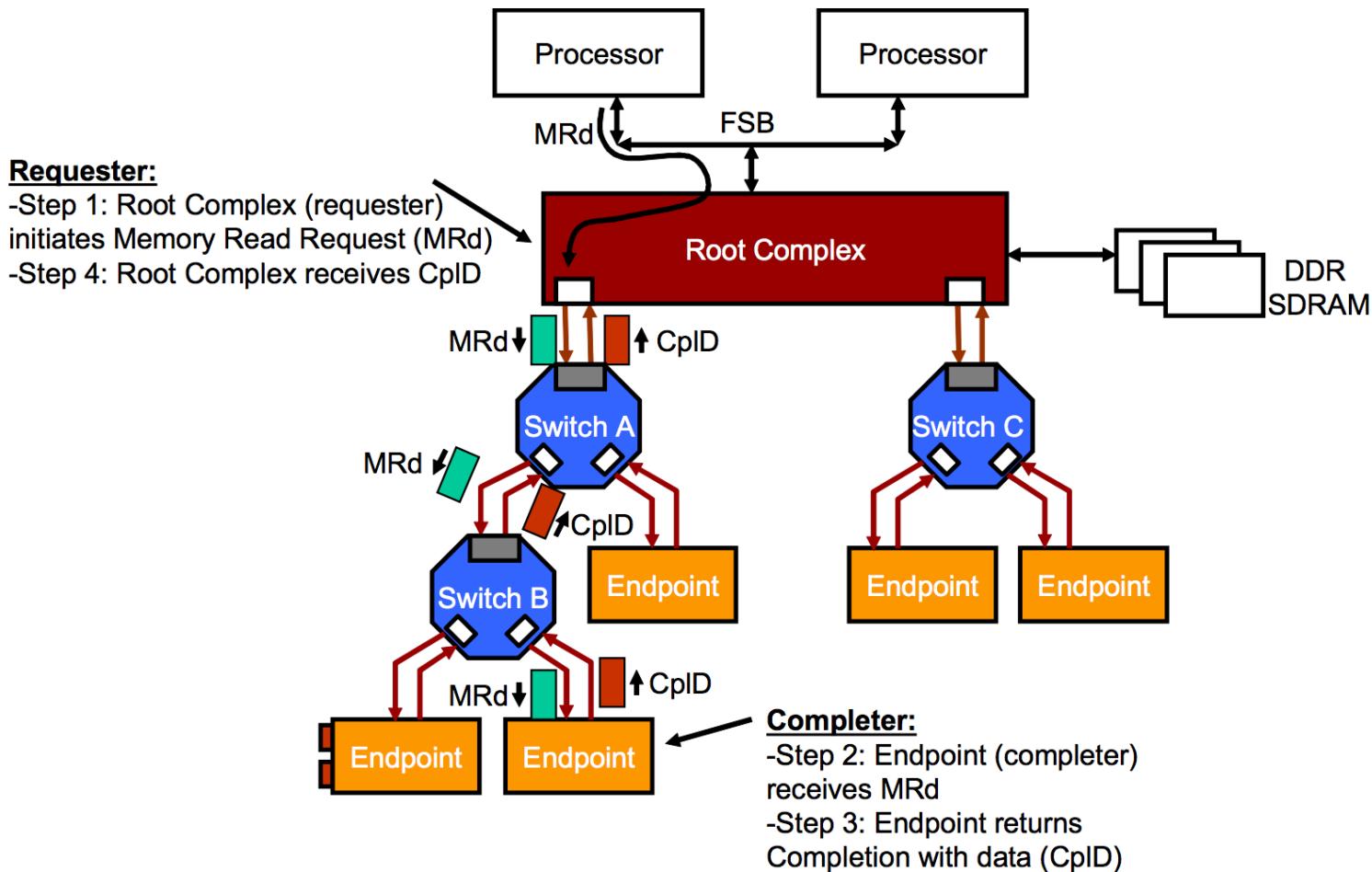


ПАКЕТЫ PCI-E

- Вершиной стека протоколов PCI-E являются TLP пакеты различных типов состоящие из заголовка и данных
- Каждое конечное устройство имеет выделенный канал связи с PCI-E switch который осуществляет маршрутизацию TLP пакетов в соответствии с определенными в спецификации правилами

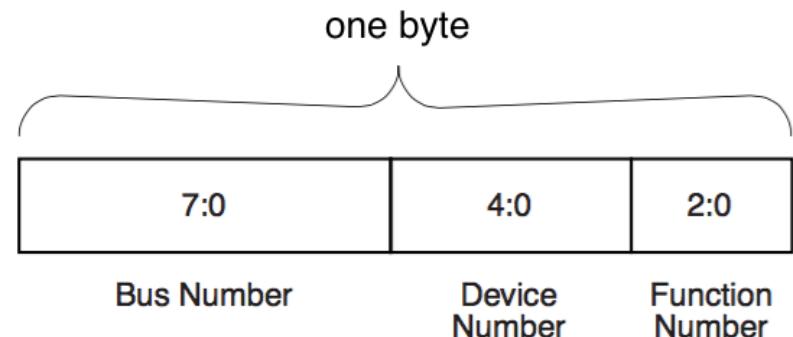


МАРШРУТИЗАЦИЯ TLP ПАКЕТОВ



ПРОГРАММНЫЙ ИНТЕРФЕЙС PCI-E

- Наверное, единственное общее между PCI и PCI-E
- Каждое PCI-E устройство имеет идентификатор вида **XX:YY.Z** состоящий из **bus** (номер шины), **device** (номер устройства нашине) и **function** (номер функции устройства)



```
$ lspci -tv
-[0000:00]-+00.0 Intel Corporation 82P965/G965 Memory Controller Hub
    +-01.0-[01]---00.0 Parallels, Inc. Accelerated Virtual Video Adapter
    +-03.0 Parallels, Inc. Virtual Machine Communication Interface
    +-05.0 Realtek Semiconductor Co., Ltd. RTL-8029(AS)
    +-0a.0-[02]--
    +-0e.0 Red Hat, Inc Virtio memory balloon
    +-1d.0 Intel Corporation 82801FB/FBM/FR/fw/FRW (ICH6 Family) USB UHCI #1
    +-1d.6 NEC Corporation uPD720200 USB 3.0 Host Controller
    +-1d.7 Intel Corporation 82801FB/FBM/FR/fw/FRW (ICH6 Family) USB2 EHCI Controller
    +-1e.0-[03]--
    +-1f.0 Intel Corporation 82801HB/HR (ICH8/R) LPC Interface Controller
    +-1f.1 Intel Corporation 82801BA IDE U100 Controller
    \-1f.2 Intel Corporation 82801HR/H0/HH (ICH8R/D0/DH) 6 port SATA Controller [AHCI mode]
```

КОНФИГУРАЦИОННОЕ ПРОСТРАНСТВО PCI-E

- Каждое PCI-E устройство имеет отдельное конфигурационное пространство
 - Первичное: 0-255 байты
 - Расширенное: 256-4095 байты
- Содержит информацию о PCI-E устройстве (производитель, класс устройства, опции энергопитания), регистры для управления ним и многое другое

31	16 15	0
Device ID		Vendor ID
Status		Command
Class Code		Revision ID
BIST	Header Type	Lat. Timer
		Cache Line S.
Base Address Registers		
Cardbus CIS Pointer		
Subsystem ID	Subsystem Vendor ID	
Expansion ROM Base Address		
Reserved		Cap. Pointer
Reserved		
Max Lat.	Min Gnt.	Interrupt Pin
		Interrupt Line

00h
04h
08h
0Ch
10h
14h
18h
1Ch
20h
24h
28h
2Ch
30h
34h
38h
3Ch

КОНФИГУРАЦИОННОЕ ПРОСТРАНСТВО PCI-E НА X86

- Intel IA-32 архитектура имеет два механизма для доступа к конфигурационному пространству PCI-E устройства:
 - Memory mapped – конфигурационное пространство каждого PCI-E устройства спроектировано в пространство физической памяти по адресу которой хранится в MCFG таблице ACPI
 - I/O mapped – используя I/O порт 0xCF8 для адреса и порт 0xCFC для данных:

```
1  ;
2  ; Read PCI config space byte at offset 0xDC for device 00:1f.0
3  ;
4  push  edx
5  mov   eax, 0x8000f8dc    ; config space address value
6  mov   dx, 0xcf8          ; address port number
7  out   dx, eax           ; write config space address I/O port
8  xor   eax, eax
9  mov   dx, 0xcfc          ; data port number
10 in    al, dx             ; read config space data I/O port
11 pop   edx
```

ПРЯМОЙ ДОСТУП К ПАМЯТИ

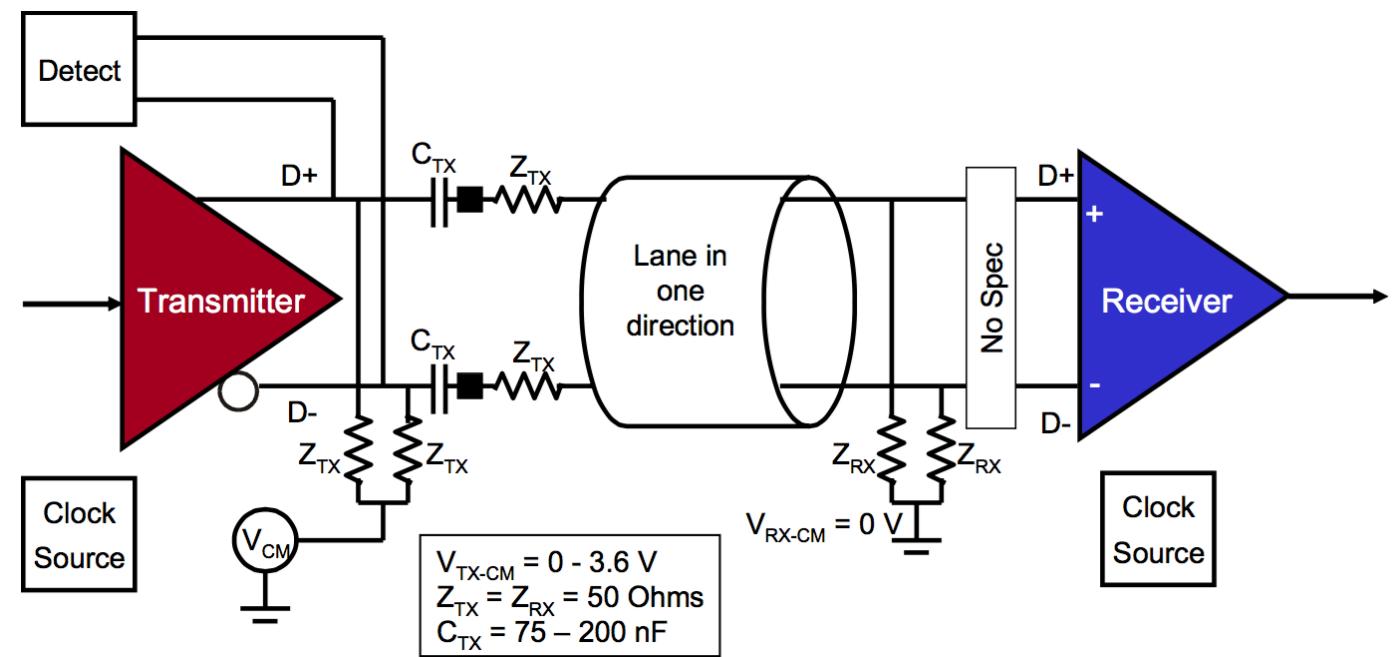
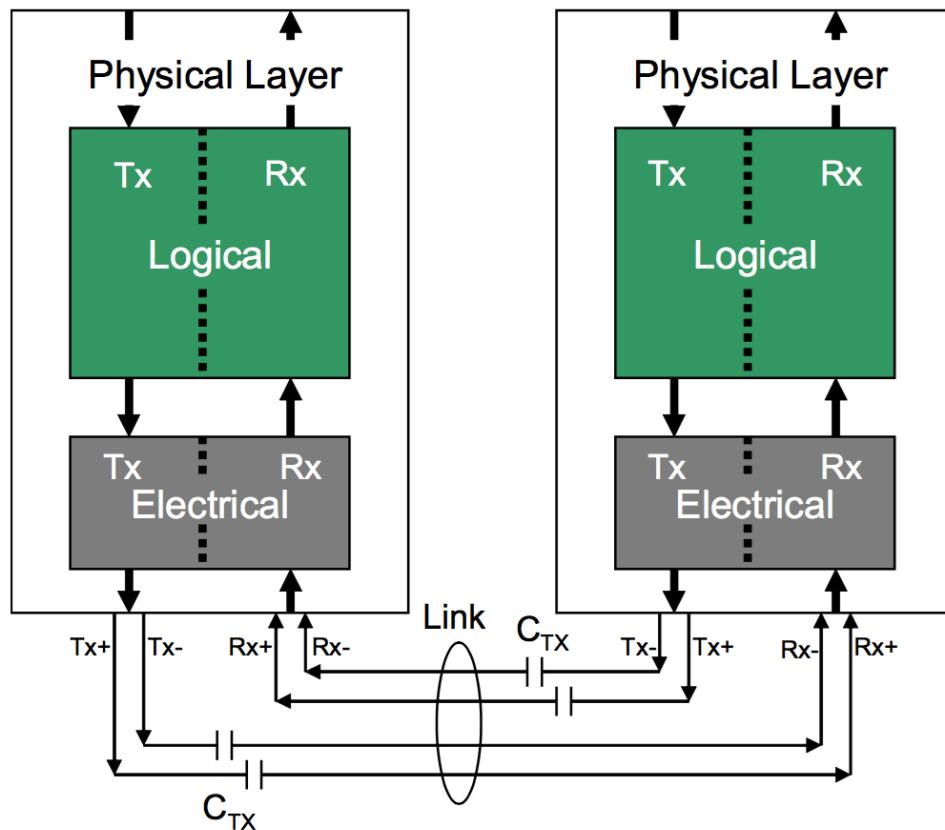
- PCI-E устройства могут сами читать или писать данные из RAM пока CPU занят другими задачами
- DMA на примере получения данных от дискового контроллера:
 - Драйвер диска выделяет физически непрерывный регион памяти в качестве I/O буфера и создает для него запись в PRDT (специальной таблице DMA контроллера)
 - Bus master регистр дискового контроллера (находится в конфигурационном пространстве PCI) устанавливается в '1' сообщая ему о том, что устройство может обращаться к RAM
 - В командный регистр дискового контроллера записывается номер DMA read команды ATA/ATAPI, после этого CPU переключается на другую задачу ожидая завершения операции
 - Контроллер читает данные с диска и записывает их в указанный регион физической памяти путем отправки Memory Write TLP запросов через шину PCI-E
 - Контроллер отправляет Interrupt Signaling Message TLP запрос генерируя прерывание которое сообщает CPU об успешном завершении I/O операции

КАК ПОПАСТЬ НА УРОВЕНЬ TLP

- Разработка PCI Express устройства с нуля – сложная и дорогостоящая задача
 - Огромные скорости передачи данных требуют приемники и передатчики логика которых реализована непосредственно в кремнии
 - Signal integrity налагает серьезные требования на разработку печатной платы (расчёт геометрии проводников, соблюдение норм производства)
 - Тысячи страниц спецификаций
- За основу PCI-E устройства можно взять FPGA
 - Готовая реализация физического и data link уровней, возможность отправки любых TLP
 - Высокая стоимость чипов и ПО
- За основу PCI-E устройства можно взять SoC или специализированный контроллер
 - Работа с PCI-E шиной ограничена программным интерфейсом, возможности произвольной генерации TLP пакетов нет

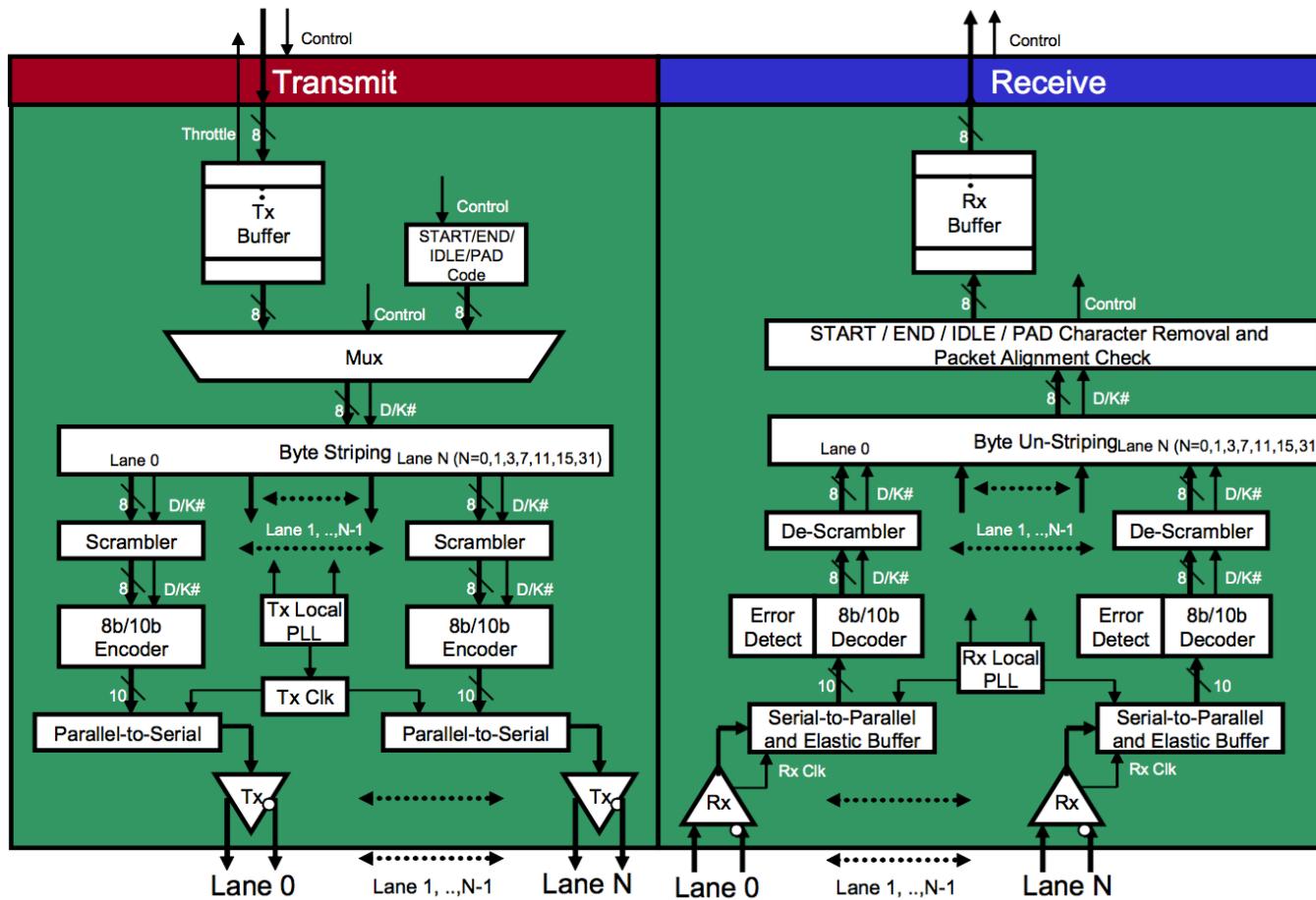
КАК ПОПАСТЬ НА УРОВЕНЬ TLP

- Вам понадобится реализация электрической части физического уровня PCI-E:



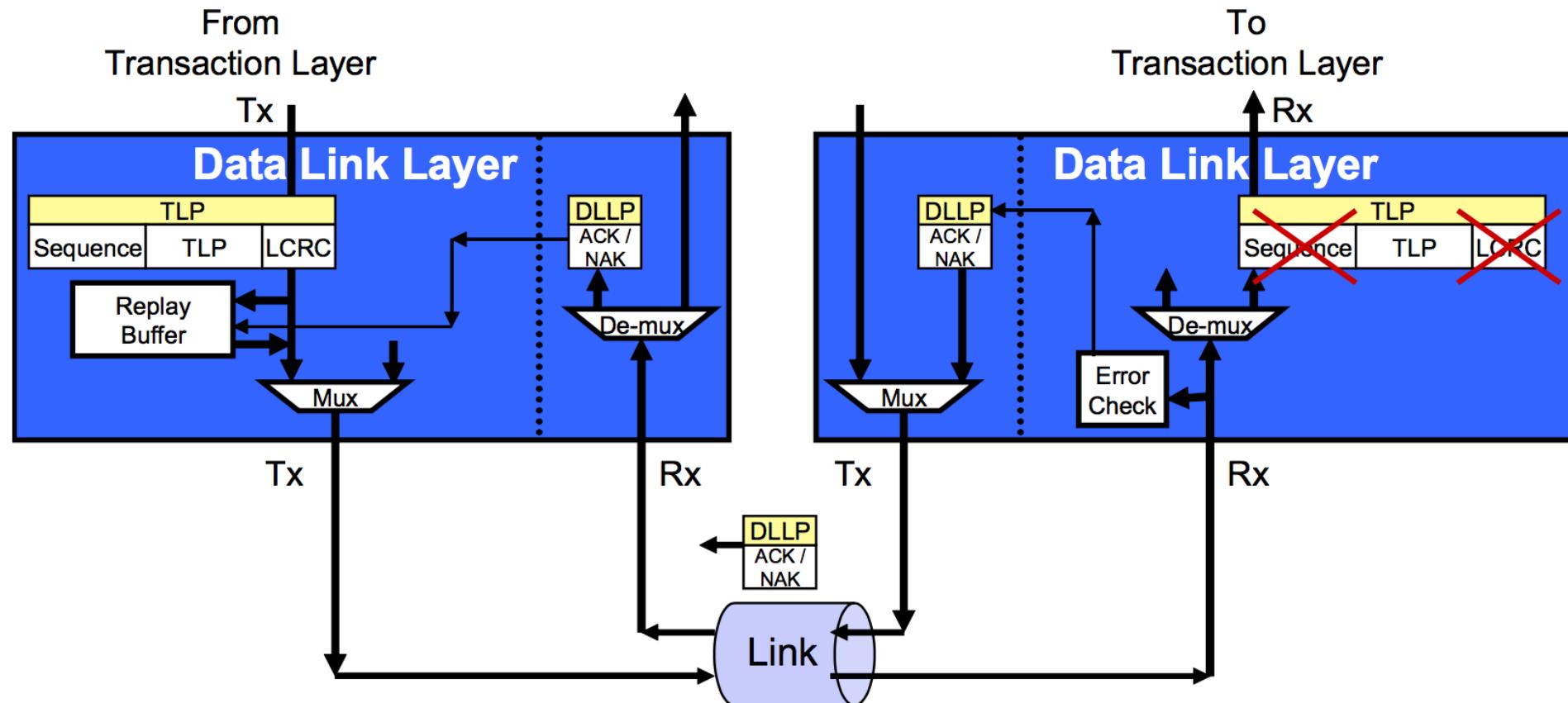
КАК ПОПАСТЬ НА УРОВЕНЬ TLP

- ... и реализация логической части физического уровня PCI-E:



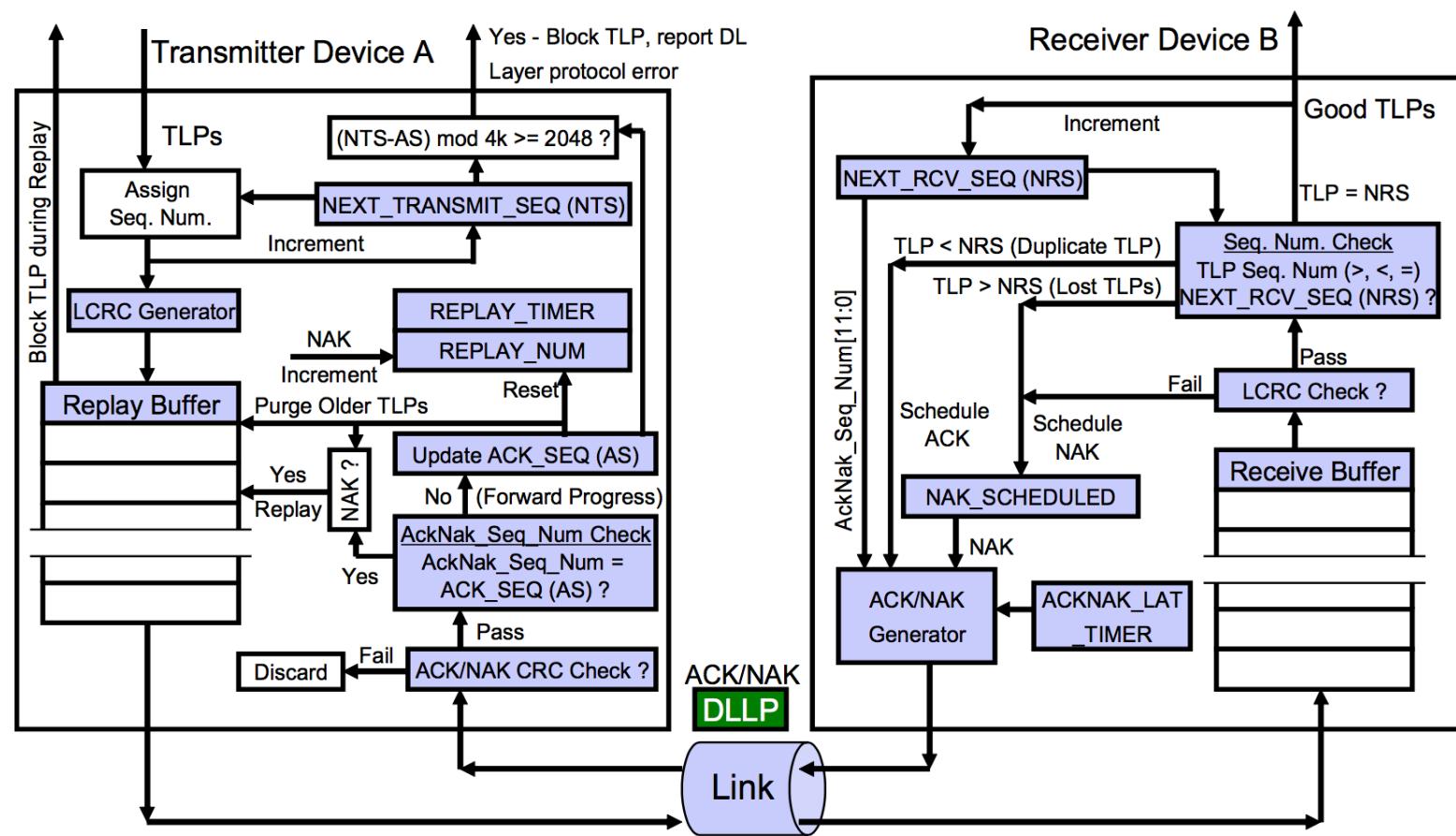
КАК ПОПАСТЬ НА УРОВЕНЬ TLP

- Вам понадобится реализация data link уровня PCI-E



КАК ПОПАСТЬ НА УРОВЕНЬ TLP

- ... включающая в себя реализацию ACK/NAK протокола:

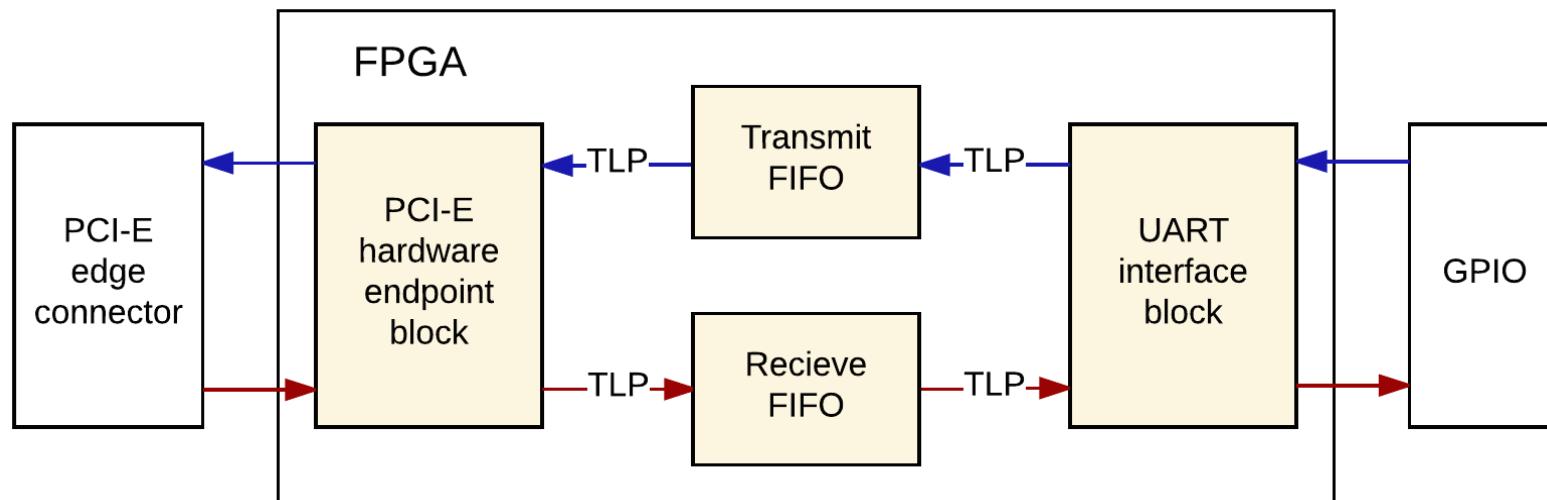


КАК ПОПАСТЬ НА УРОВЕНЬ TLP

- В реальности инженеры не разрабатывают реализацию физического и data link уровней PCI-E собственными силами, они опираются на готовые решения
- Что мы можем себе позволить с бюджетом до \$500:
 - USB3380EVB evaluation board
 - Mini PCI-E плата с USB3380 контроллером USB 3.0 интерфейса и открытым SDK
 - Для данной платы была выпущена удобная программа для DMA атак под названием PCILeech
 - Возможность работы с PCI-E шиной ограничена программным интерфейсом контроллера, нельзя генерировать произвольные TLP, нельзя обращаться к RAM выше 4GB
 - Наборы для разработки PCI-E устройств на базе FPGA
 - Есть у всех крупных производителей FPGA, как правило идут в комплекте с evaluation лицензиями на необходимые инструменты разработки и IP ядра (библиотеки в мире разработки под FPGA)
 - Оптимальный выбор, однако достойного внимания открытого ПО для DMA атак с использованием FPGA нет

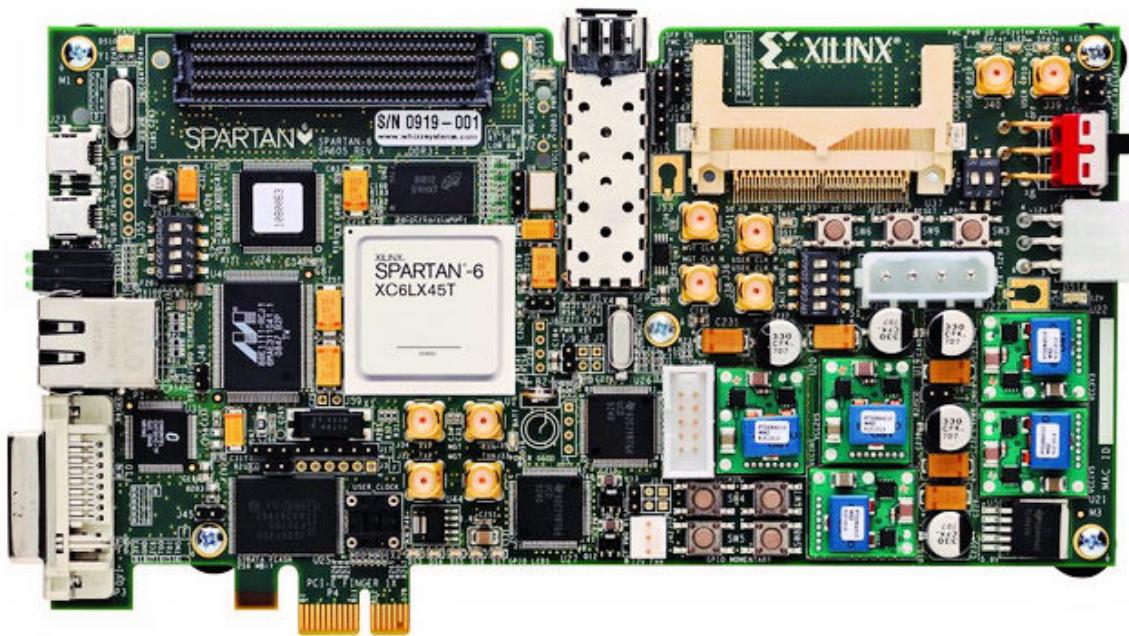
РАЗРАБОТКА С ИСПОЛЬЗОВАНИЕМ FPGA

- Массив из большого числа простых логических блоков на одном чипе в сочетании с аппаратными блоками для работы с различными высокоскоростными шинами. Конфигурация соединения этих блоков описывается разработчиком на высокоуровневом HDL языке (VHDL, Verilog)
- Пример моста TLP уровня между PCI-E и UART на базе FPGA:



РАБОТА С PCI-E НА БАЗЕ FPGA ОТ XILINX

- Мой выбор пал на набор разработки Xilinx SP605 с FPGA семейства Spartan-6 (XC6SLX45T)
 - Старая плата, сейчас ее можно приобрести за цену порядка \$500
 - Интерфейсы Ethernet, SFP, DVI, Multi-Gigabit трансиверы подключенные к SMA разъемам, CF слот, FMC порт для плат расширения
 - PCI-E x1 edge connector подключенный на прямую к FPGA, поддерживает PCI-E 1.1 с 2.5 GT/s в режиме endpoint



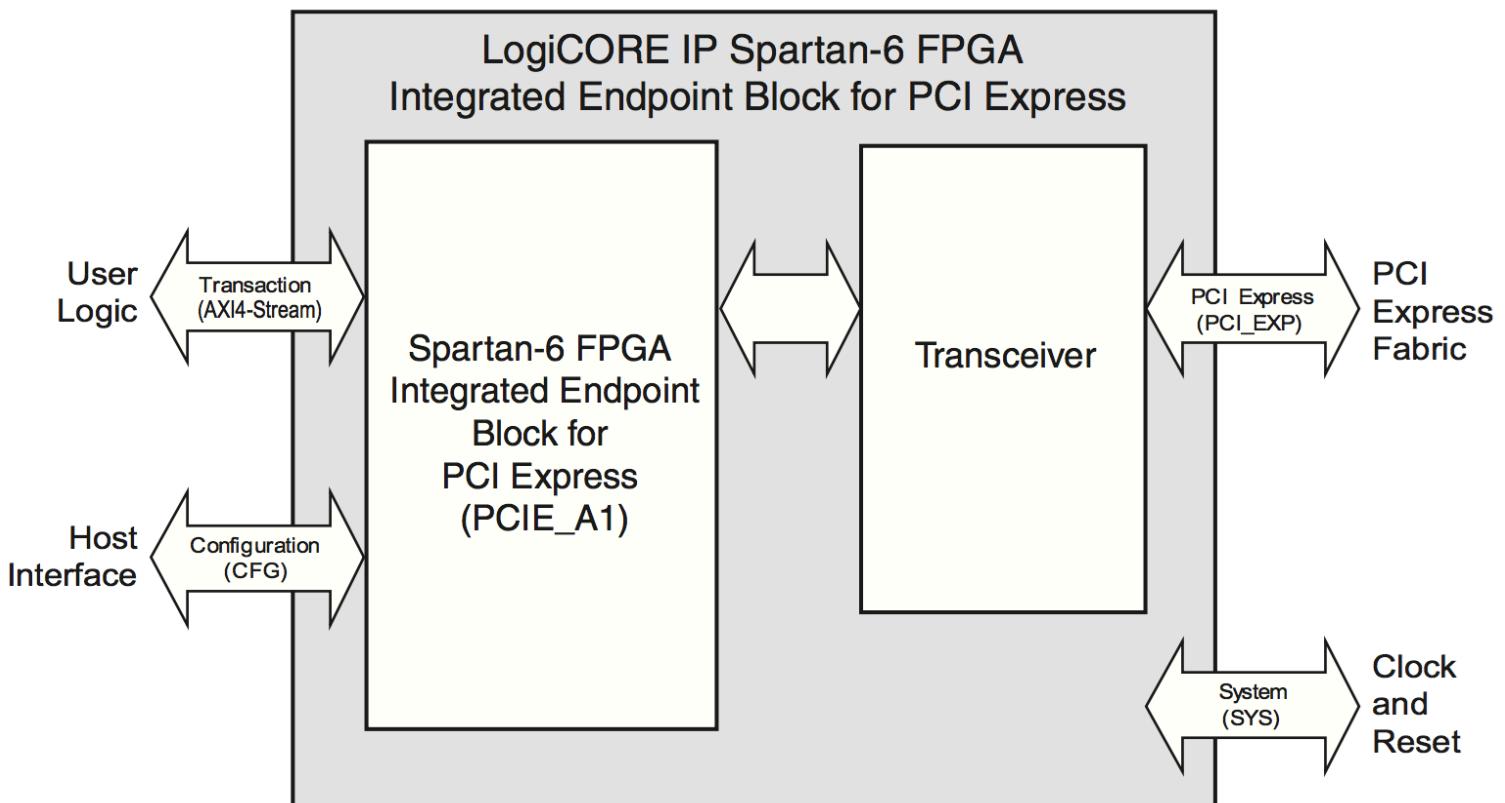
РАБОТА С PCI-E НА БАЗЕ FPGA ОТ XILINX

- PCI-E интерфейс на FPGA серии Spartan-6 реализован на базе аппаратного блока который включает в себя физический и data link уровни
- PCI-E блок имеет высокоскоростной последовательный интерфейс, Xilinx IP Core Generator из состава Xilinx ISE позволяет сгенерировать код IP ядра которое позволяет работать с этим интерфейсом как с AXI4-Lite или AXI4-Stream slave устройством
 - Так же это IP ядро берет на себя управление клоком с использованием тактового сигнала PCI-E шины и другие задачи
- В качестве реализации UART для FPGA был выбран UART6 из открытого проекта PicoBlaze, он отлично оптимизирован для Spartan-6
- В качестве UART to USB интерфейса который подключается GPIO пинам SP605 была выбрана плата FT2232H Mini Module с интерфейсным чипом FTDI FT2232H
 - Максимальная скорость в режиме UART – 921600 bit/s



INTEGRATED PCI-E ENDPOINT BLOCK

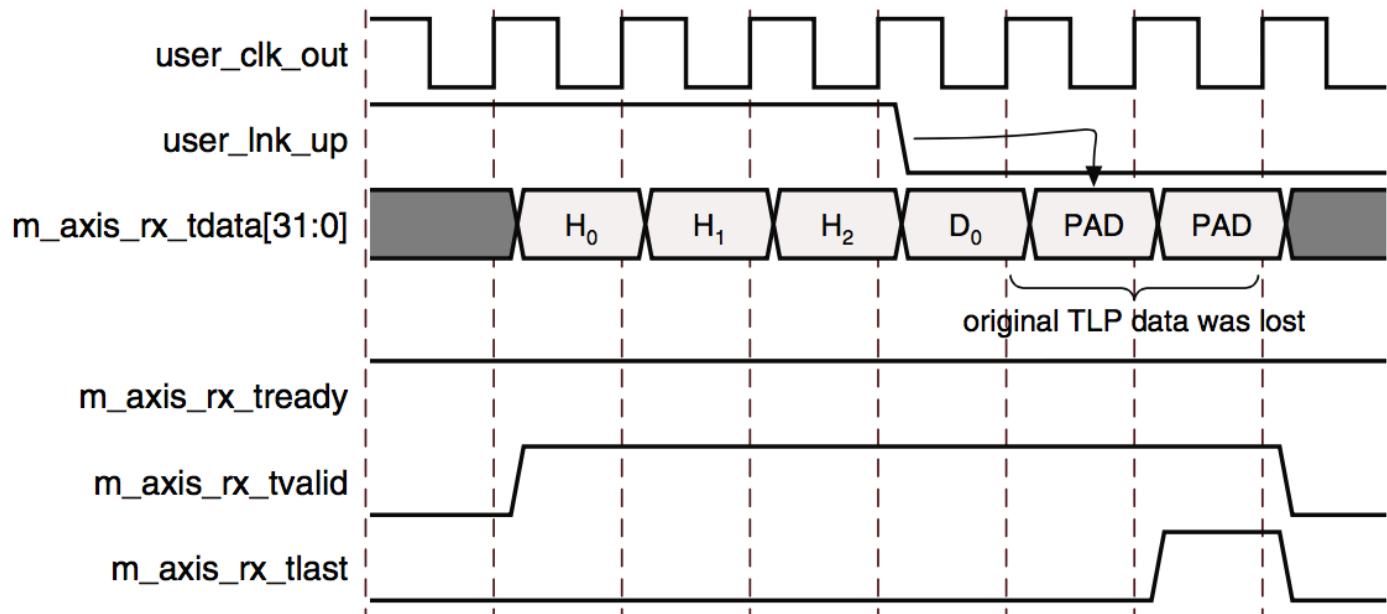
- Для работы с аппаратным PCI-Е блоком было выбрано IP ядро Spartan-6 FPGA Integrated Endpoint Block for PCI Express (s6_pcie) версии 2.4 с AXI4-Stream интерфейсом который передает по одному DWORD-у TLP пакета за один такт 32-х разрядной синхронной шины
 - Условно бесплатно для некоммерческого использования



INTEGRATED PCI-E ENDPOINT BLOCK

- HDL модуль IP ядра имеет несколько различных интерфейсов: Core, Transaction (transmit и receive) и Error Reporting
 - Сложное ядро, много десятков input и output сигналов и параметров. К счастью, нам интересны главным образом Transaction Interface
- Rx сигналы Transaction Interface для приема TLP пакетов

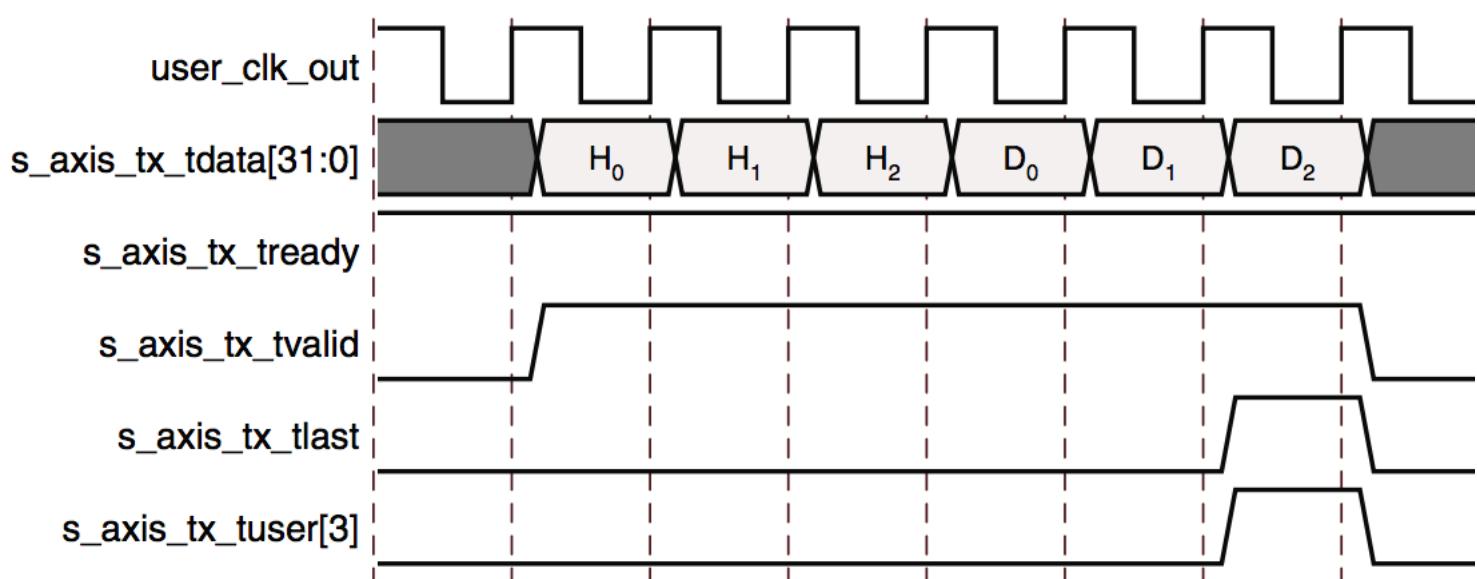
```
output [31:0]    m_axis_rx_tdata,  
output [3:0]     m_axis_rx_tkeep,  
output          m_axis_rx_tlast,  
output          m_axis_rx_tvalid,  
input           m_axis_rx_tready,  
output [21:0]   m_axis_rx_tuser,  
input          rx_np_ok,
```



INTEGRATED PCI-E ENDPOINT BLOCK

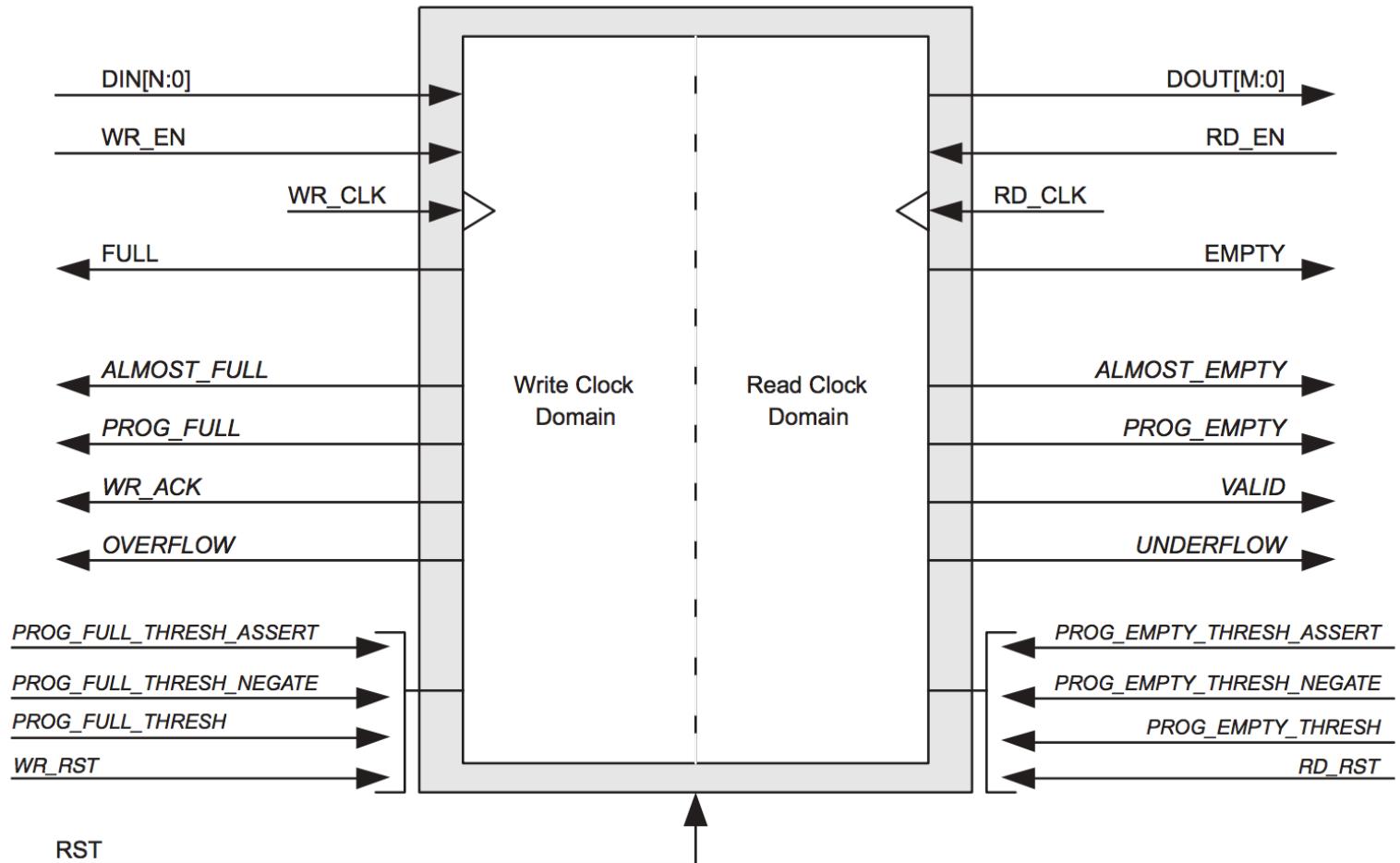
- Tx сигналы Transaction Interface для отправки TLP пакетов

```
output      s_axis_tx_tready,
input [31:0] s_axis_tx_tdata,
input [3:0]  s_axis_tx_tkeep,
input [3:0]  s_axis_tx_tuser,
input       s_axis_tx_tlast,
input       s_axis_tx_tvalid,
```



FIFO QUEUE

- FIFO очередь нужна для того что бы связать 32-х разрядный интерфейс PCI-E ядра и 8-ми разрядный интерфейс UART трансивера
 - Выполняет роль буфера
 - Обеспечивает передачу данных между *clock domains*
- Использовано стандартное IP ядро FIFO очереди от Xilinx (*fifo_generator*) версии 8.4
 - По одной очереди для Tx и Rx
 - Данные хранятся в SRAM
 - Простой синхронный интерфейс



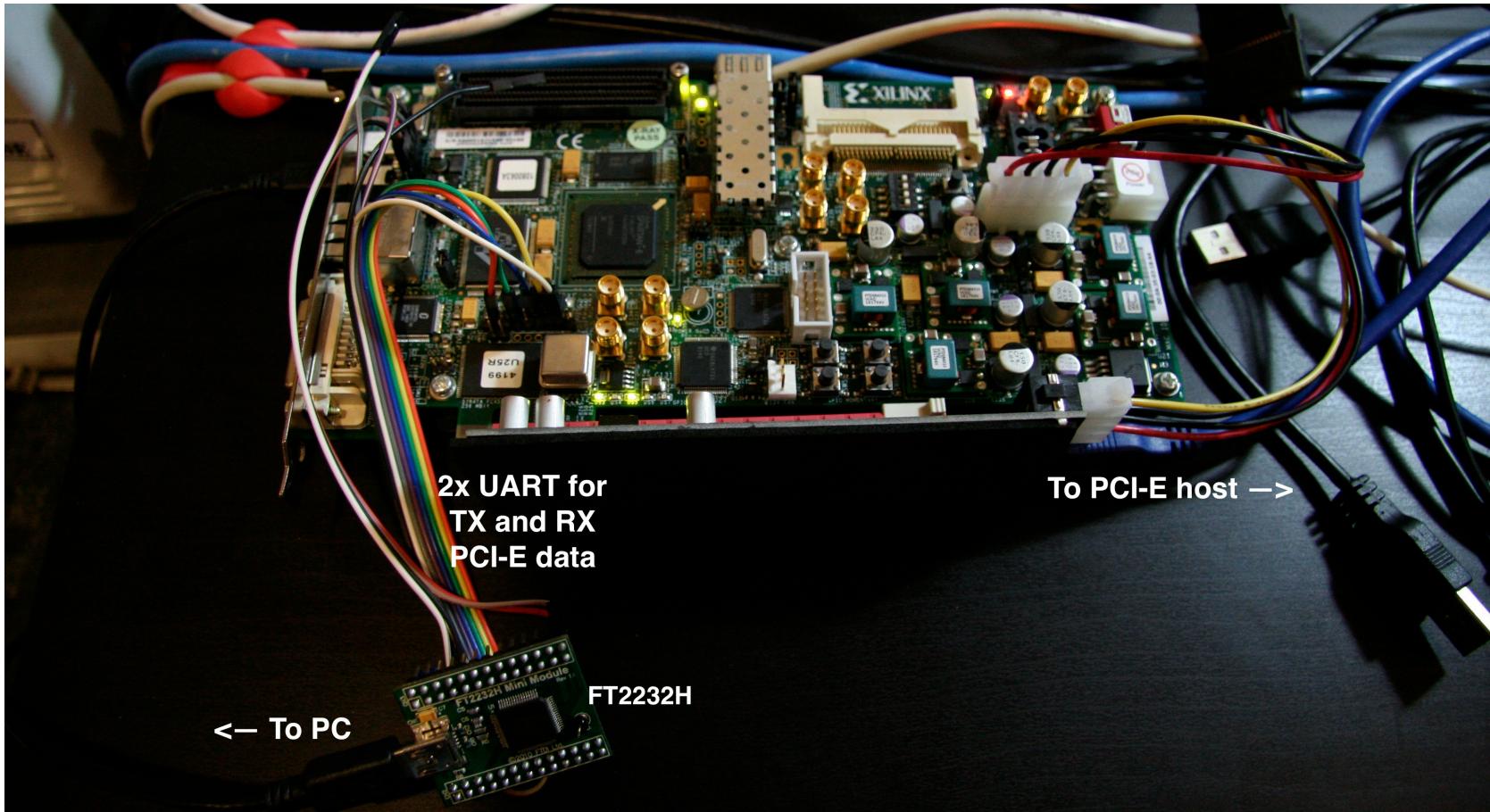
PCI-E TO UART BRIDGE НА FPGA

- Довольно простой проект с несколькими IP ядрами и несколькими сотнями строк кода (не считая сгенерированных) на языке Verilog
- PCI-E и UART части проекта работают в разных clock domains (62.5 МГц и 250 МГц)
- Второй канал FT2232H подключен ко второму UART трансиверу на FPGA который используется для доступа к конфигурационному пространству PCI-E устройства

```
assign uart1_tx_data = {  
    16'h0,  
    // actual PCI-E device address  
    cfg_bus_number,  
    cfg_device_number,  
    cfg_function_number,  
  
    // configuration space data out  
    cfg_do  
};
```

```
assign uart0_tx_data = {  
    1'b1,                      // indicates valid packet  
    PCIE_UART_S_RX, // indicates PCI-E rx data  
    8'h0,                      // not used, fill with 0's  
  
    // command register bits  
    cfg_command_interrupt_disable,  
    cfg_command_serr_en,  
    cfg_command_bus_master_enable,  
    cfg_command_mem_enable,  
    cfg_command_io_enable,  
  
    // actual PCI-E device address  
    cfg_bus_number,  
    cfg_device_number,  
    cfg_function_number,  
  
    // AXI bus data to transmit  
    m_axis_rx_tlast,  
    m_axis_rx_tdata  
};
```

PCI-E TO UART BRIDGE HA FPGA

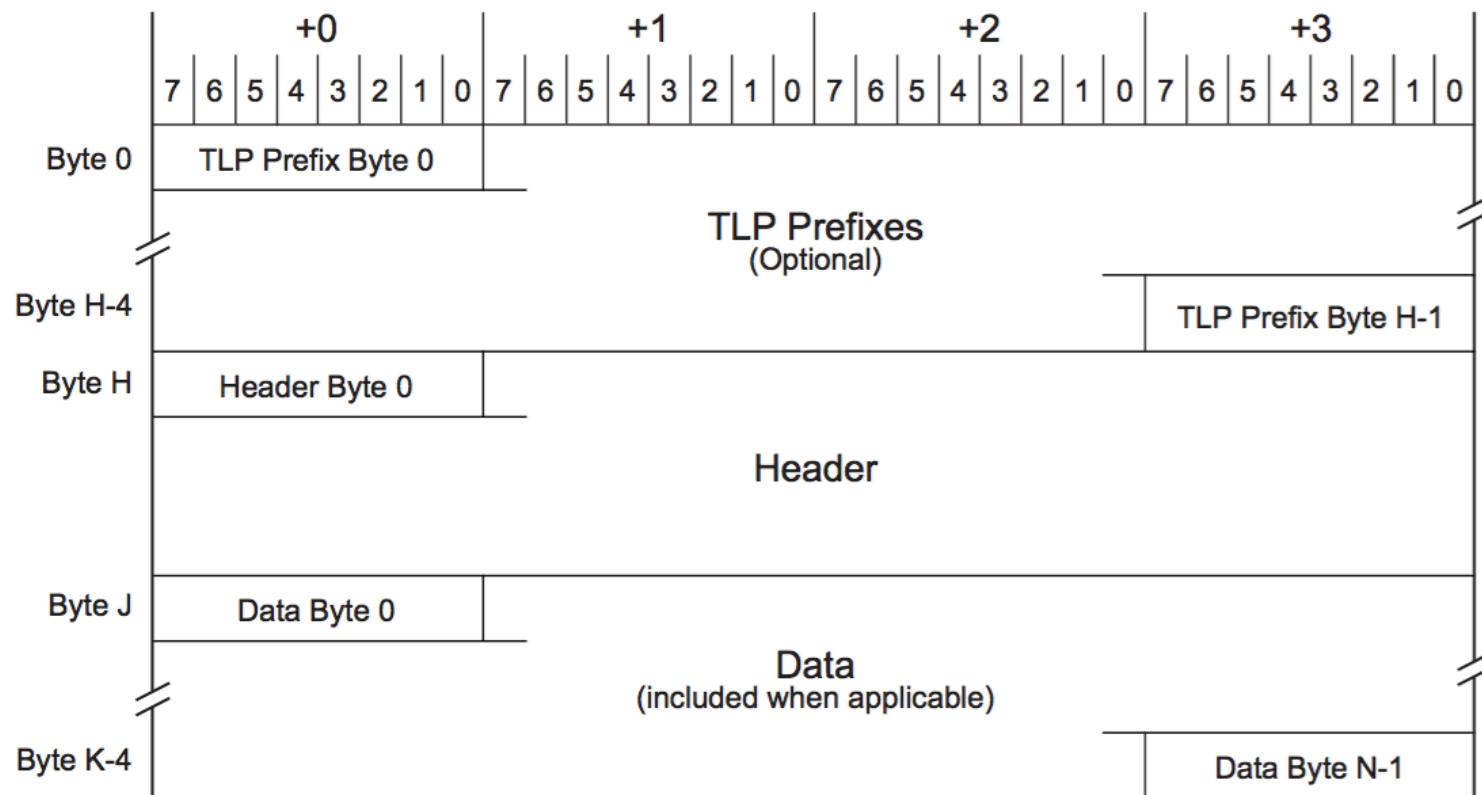


ФОРМАТ TLP ПАКЕТОВ

- TLP пакеты состоят с из 3 или 4 DWORD-ов заголовка и опциональных данных
- Формат первого DWORD-а заголовка TLP пакета является постоянным, формат остальных DWORD-ов а так же их количество зависит от типа TLP пакета
- TLP пакеты могут иметь один или несколько префиксов
 - Бывают довольно разнообразными но встречаются не часто
- Существует около двух десятков типов TLP пакетов
 - MRd, MRdLk, MWr – доступ к памяти
 - CfgRdo, CfgWro, CfgRdI, CfgWrI – доступ к конфигурационному пространству
 - IORd, IOWr – доступ к I/O пространству
 - Msg, MsgD – интерфейс сообщений
 - Cpl, CpID, CpILk, CpILkD – completion TLP
 - FetchAdd, Swap, CAS – TLP для атомарных операций с памятью

ФОРМАТ TLP ПАКЕТОВ

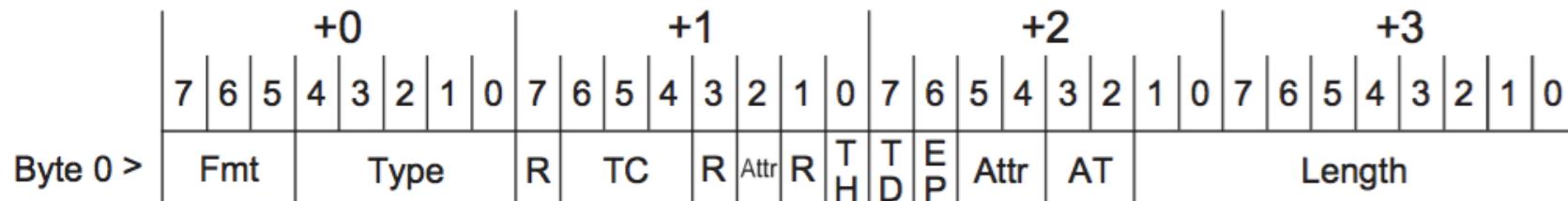
- Общий формат TLP пакета



ФОРМАТ TLP ПАКЕТОВ

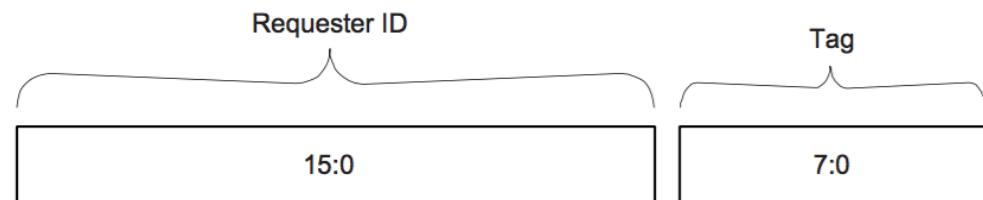
- Формат первого DWORD-а TLP пакета
 - Поле Fmt кодирует формат заголовка пакета
 - Поле Length кодирует количество DWORD-ов в этом пакете
 - Поле Type кодирует тип TLP пакета

Fmt[2:0]	Corresponding TLP Format
000b	3 DW header, no data
001b	4 DW header, no data
010b	3 DW header, with data
011b	4 DW header, with data
100b	TLP Prefix
	All encodings not shown above are Reserved (see Section 2.3).

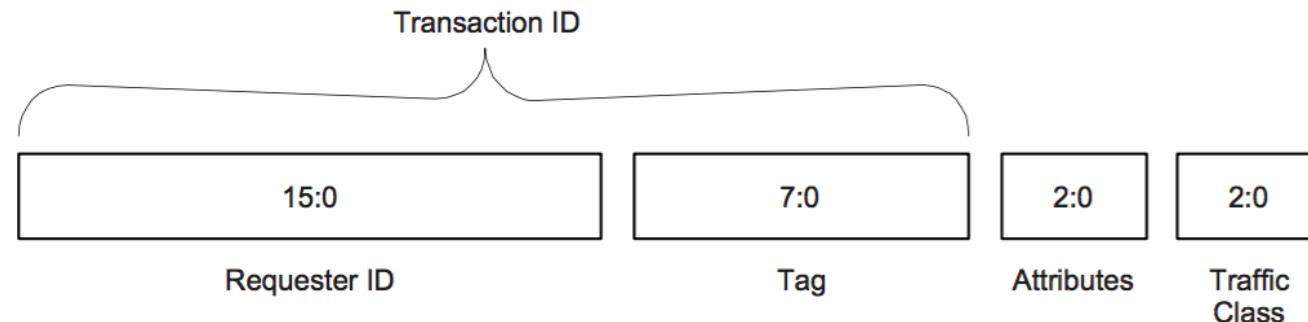


ФОРМАТ TLP ПАКЕТОВ

- Маршрутизация TLP пакетов происходит по разному в зависимости от их типа
 - По адресу (TLP для доступа к памяти)
 - По идентификатору устройства
- Второй DWORD заголовка TLP пакета обычно содержит идентификатор пакета:

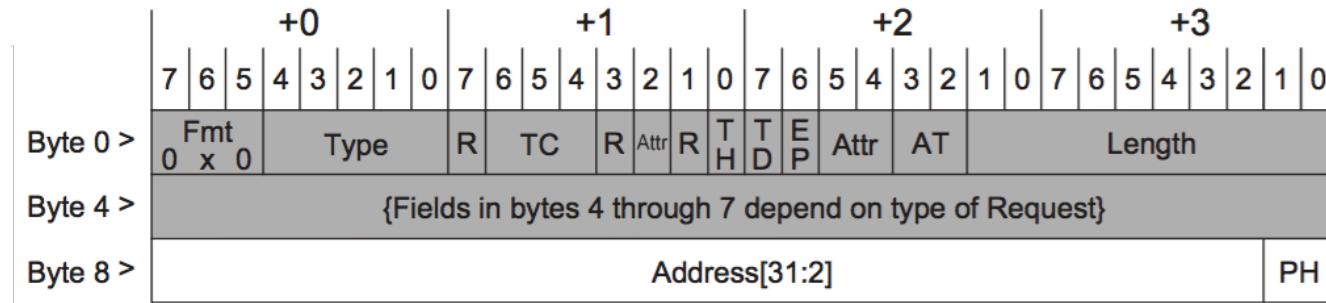
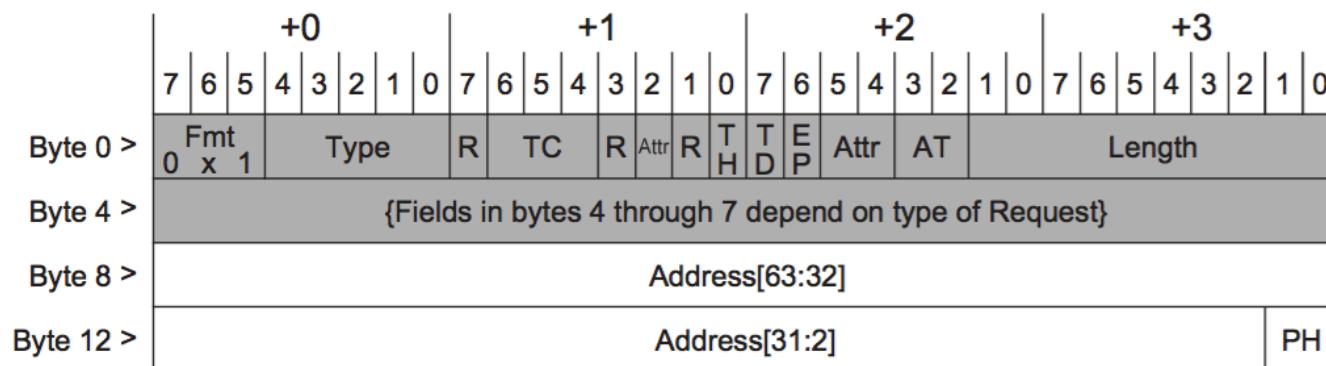


- ... который вместе с полями Attributes и Traffic Class составляет идентификатор транзакции:



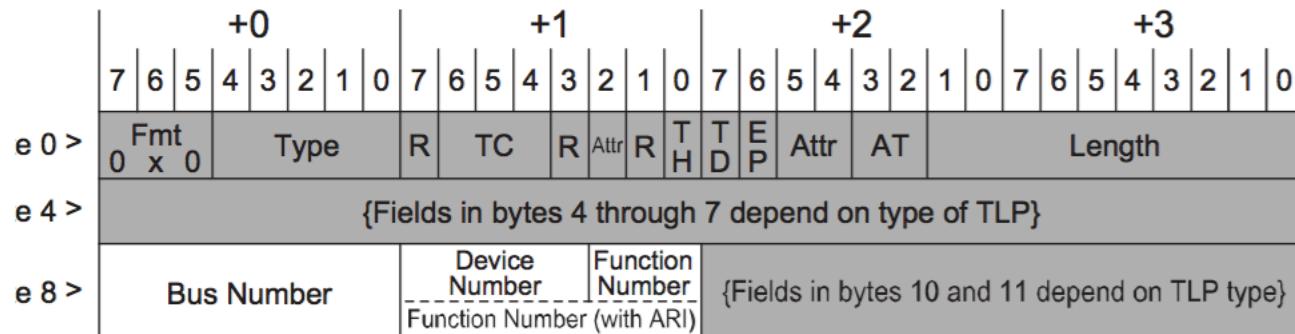
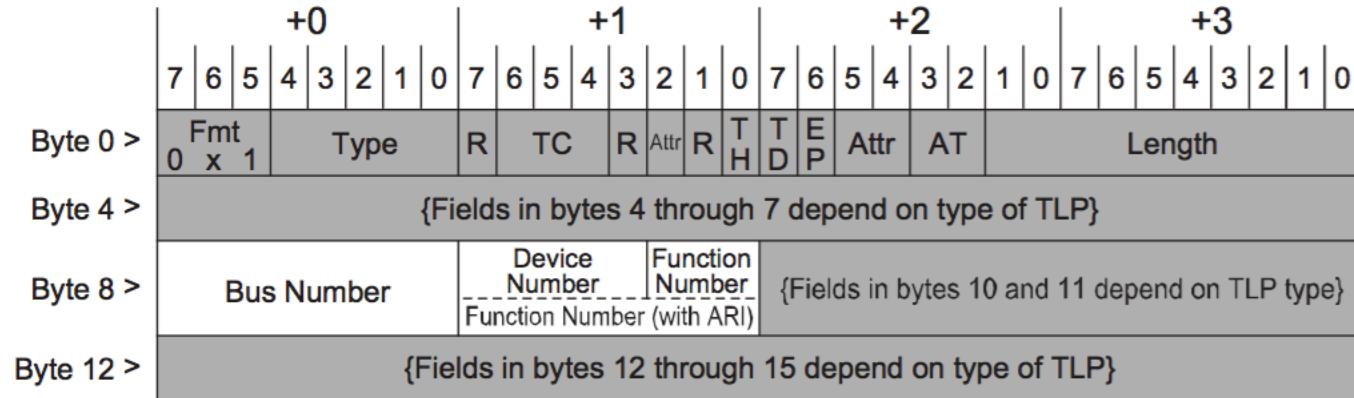
ФОРМАТ TLP ПАКЕТОВ

- Формат заголовка TLP пакета который маршрутизируется по адресу памяти:



ФОРМАТ TLP ПАКЕТОВ

- Формат заголовка TLP пакета который маршрутизируется по идентификатору устройства:



ПРИМЕРЫ TLP ТРАФИКА

- Чтение из памяти с использованием TLP пакета MRd
 - В ответ приходит CplD пакет с данными в случае успеха или CplI пакет с кодом ошибки

```
TLP TX: size = 0x04, source = 01:00.0, type = MRd64  
tag = 0x00, bytes = 0x8, addr = 0x00010000
```

```
0x20000002 0x01000ff 0x00000000 0x00010000
```

```
TLP RX: size = 0x05, source = 00:00.0, type = CplD  
tag = 0x00, bytes = 8, req = 01:00.0, comp = 00:00.0
```

```
0x4a000002 0x00000008 0x01000000  
0x00000000 0x00000000
```

ПРИМЕРЫ TLP ТРАФИКА

- Запись в память с использованием TLP пакета MWr

```
TLP TX: size = 0x05, source = 01:00.0, type = MWr64  
tag = 0x00, bytes = 0x4, addr = 0x00010004
```

```
0x60000001 0x010000ff 0x00000000 0x00010004  
0x00000000
```

СОФТ

- Для обмена данными через USB интерфейс FT2232H Mini Module была использована Python библиотека `pylibftdi`
 - Доступна на Windows, Linux и OS X, минимум зависимостей
- Была разработана Python библиотека `rcie_lib`
 - Классы для передачи TLP пакетов по протоколу UART
 - Классы для парсинга и генерации TLP пакетов различных форматов
 - API для чтения физической памяти
- Скрипты на Python использующие `rcie_lib` реализовывают программно управляемое на TLP уровне PCI-E устройство
 - Как FaceDancer, только для PCI Express

ФРАГМЕНТЫ КОДА

- Формирование TLP пакета из данных полученных по UART:

```
def read(self):  
    ret = []  
  
    while True:  
  
        # read status dword and TLP dword  
        status, data = self._read(2)  
  
        # check status dword  
        assert self.status_valid(status)  
        assert self.status_rx(status)  
  
        # update device address information  
        self.bus_id = dev_id_decode(self.status_bus_id(status))  
  
        ret.append(data)  
  
        # check for last dword in TLP status  
        if self.status_tlast(status): break  
  
    return ret
```

ФРАГМЕНТЫ КОДА

- Отправка TLP пакетов MWr для записи данных в физическую память:

```
def _mem_write(self, addr, data):  
    size, chunk_size, ptr = len(data), min(len(data), self.MEM_WR_TLP_LEN), 0  
  
    assert addr % self.MEM_ALIGN == 0  
    assert size % self.MEM_ALIGN == 0  
  
    # read memory by blocks  
    while ptr < size:  
  
        chunk_addr = addr + ptr  
  
        # memory r/w TLP should reside to the single memory page  
        max_chunk_size = PAGE_SIZE if chunk_addr & 0xffff == 0 else \  
        | | | | | (align_up(chunk_addr, PAGE_SIZE) - chunk_addr)  
  
        cur_chunk_size = min(chunk_size, max_chunk_size)  
  
        # get data chunk as dwords list  
        tlp_data = unpack('>' + ('I' * (cur_chunk_size / 4)), data[ptr : ptr + cur_chunk_size])  
  
        # create 64-bit memory read TLP  
        tlp = self.PacketMWr64(self.bus_id, chunk_addr, list(tlp_data))  
  
        # send TLP to the system  
        self.write(tlp)  
  
        ptr += cur_chunk_size
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ

```
00000050  39 e7 00 f0 d4 06 00 e8  2e e8 00 f0 d2 ef 00 f0  19.....|  
00000060  00 e0 00 f0 f2 e6 00 f0  6e fe 00 f0 53 ff 00 f0  1.....n...S...|  
00000070  53 ff 00 f0 a4 f0 00 f0  c7 ef 00 f0 b1 99 00 c0  IS.....|  
00000080  
root@intel-q77:~# lspci -vvs 01:00.0  
01:00.0 Ethernet controller: Xilinx Corporation Default PCIe endpoint ID  
    Subsystem: Xilinx Corporation Default PCIe endpoint ID  
    Control: I/O- Mem+ BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr- Stepping-  
    Status: Cap+ 66MHz- UDF FastB2B- ParErr- DEVSEL=fast >TAbort- <MAbor  
  
× vagrant@pentest: /vagrant_home/Documents/Xilinx/s6_pcie_uart (vagrant)  
vagrant@pentest:/vagrant_home/Documents/Xilinx/s6_pcie_uart$ python pcie_uart.py  
  
[+] Sending 32-bit memory read TLP...  
  
TLP TX: size = 0x03, source = 01:00.0, type = MRd32  
tag = 0x00, bytes = 0x4, addr = 0x00000060  
  
0x00000001 0x0100000f 0x00000060  
  
[+] Reading completion TLP...  
  
TLP RX: size = 0x04, source = 00:00.0, type = CplD  
tag = 0x00, bytes = 4, req = 01:00.0, comp = 00:00.0  
  
0x4a000001 0x00000004 0x01000060  
0x00e000f0  
  
[+] Readed value: 0x00e000f0
```

ПРОТИВОДЕЙСТВИЕ DMA АТАКАМ

- **IOMMU** позволяет реализовать изоляцию памяти для PCI-E устройств
- Поддержка **IOMMU** есть в современных версиях Linux, Windows и OS X
 - В случае с Windows потребуется Windows 10 Enterprise со включенным Device Guard
 - В случае с Linux потребуется `CONFIG_IOMMU_SUPPORT=y`
 - Поскольку BIOS инициализирует PCI-E шину задолго до того как ОС инициализирует **IOMMU** – злое PCI-E устройство имеет достаточно времени для того что бы взять платформу под полный контроль в процессе ее загрузки
- **Firmware** современных компьютеров от Apple инициализирует **IOMMU** на ранних этапах загрузки еще до того как PCI-E шина включена
 - В теории это обеспечивает эффективную защиту от DMA атак на всех этапах работы платформы
 - Apple единственный известный мне на рынке производитель x86 компьютеров предлагающий поддержку **IOMMU** в firmware