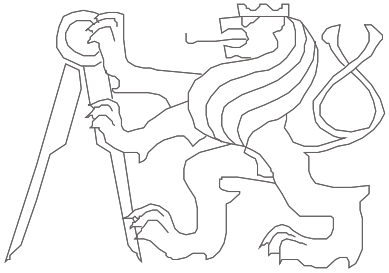
Computer Architectures

I/O Subsystem Part 1

Pavel Píša, Petr Štěpán, Richard Šusta,
Michal Štepanovský, Miroslav Šnorek

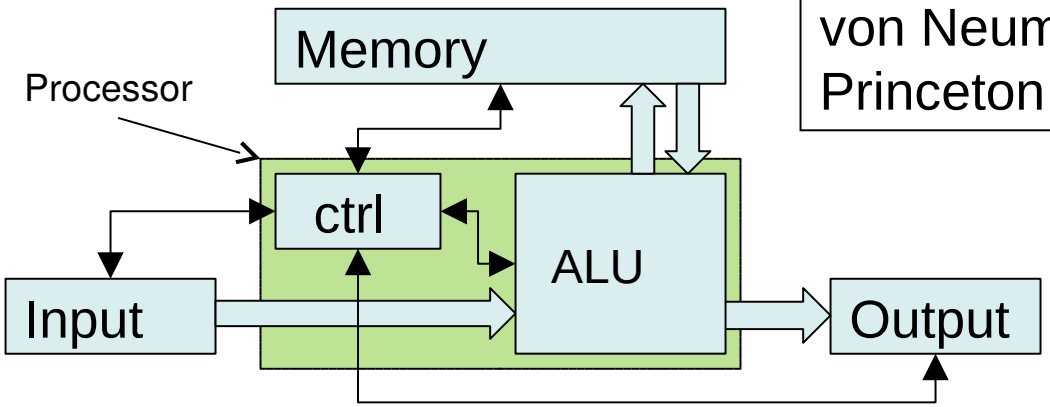


Czech Technical University in Prague, Faculty of Electrical Engineering

Lecture outline

- I/O subsystem – introduction
- Memory mapped I/O
- QtRVSim I/O Examples
- Use of buses in real computers
- PCI Bus – Peripheral Component Interconnect
- PCIe – Peripheral Component Interconnect Express
- HDD, SSD and RAID

John von Neumann's Computer Architecture



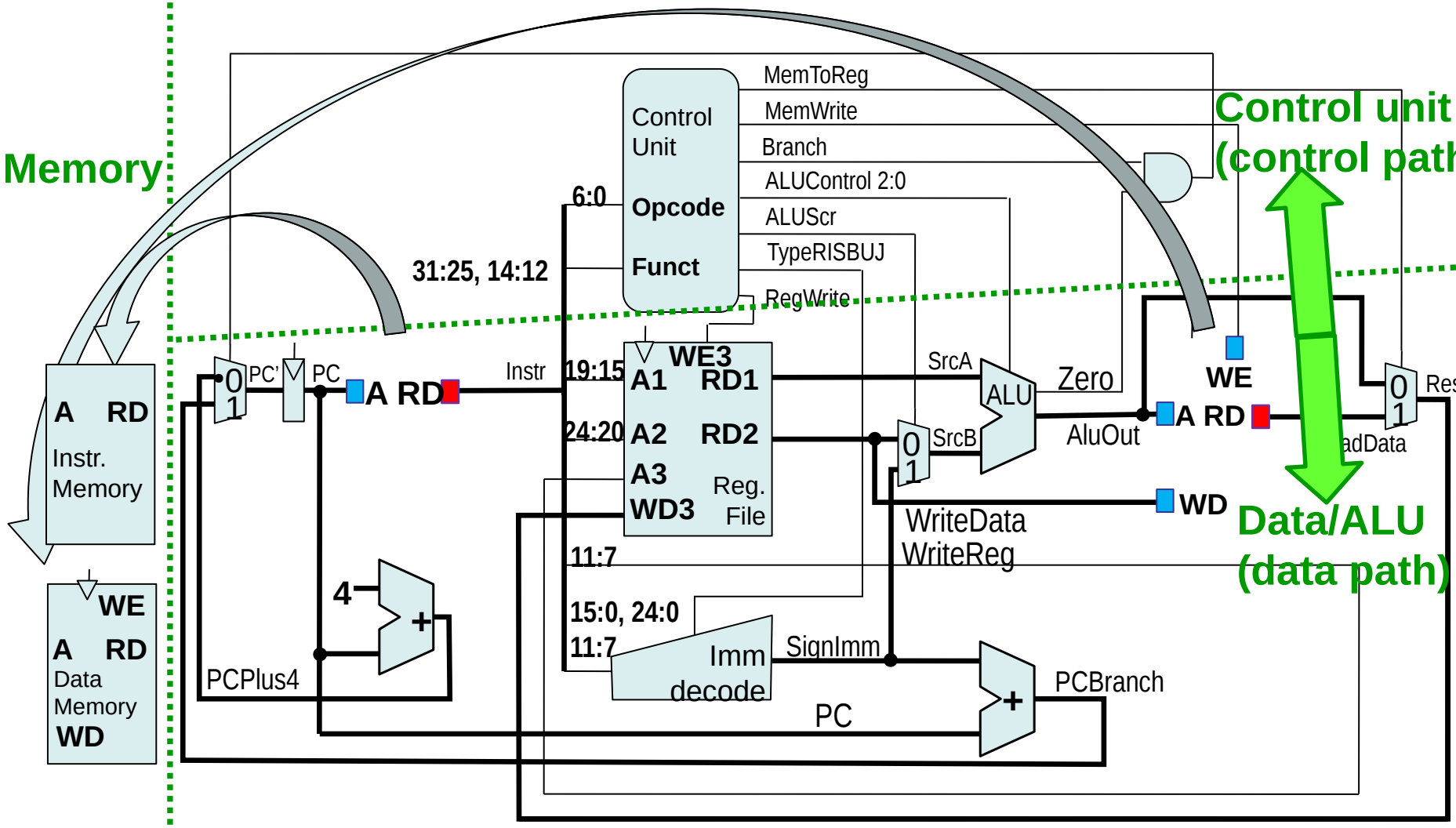
von Neumann's computer architecture
Princeton Institute for Advanced Studies

28. 12. 1903 -
8. 2. 1957

- 5 functional units – control unit, arithmetic logic unit, memory, input (devices), output (devices)
- An computer architecture should be independent of solved problems. It has to provide mechanism to load program into memory. The program controls what the computer does with data, which problem it solves.
- Programs and results/data are stored in the same memory. That memory consists of a cells of same size and these cells are sequentially numbered (address).
- The instruction which should be executed next, is stored in the cell exactly after the cell where preceding instruction is stored (exceptions branching etc.).
- The instruction set consists of arithmetics, logic, data movement, jump/branch and special/control instructions.

Data Path, Control and Memory from Our CPU Design

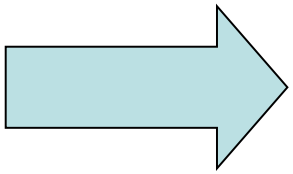
Return back to non-pipelined CPU version



Signals to Connect CPU to External Memory and Peripherals

Processor (CPU)

Memory or I/O peripherals



Address bus (A0..A31)

can be separate or multiplexed or encoded on same signals as data

Data bus (D0..D31)

can be bidirectional or separate parallel or serial lane for each direction

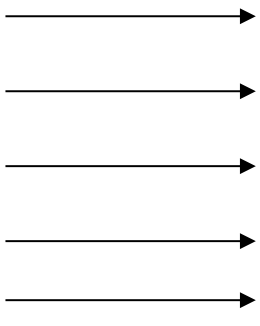
MEMW#

MEMR#

IOW#

IOR#

BE0 to 3#



Control bus or signals

IOW, IOR if input output is separated from memory operations

Byte enable if bytes writes are supported on bus wider than 8-bit

Classification of Input/Output Devices/Peripherals

- Behavior
  - Input read only
  - Output write only, cannot be read
  - Storage can be reread and usually rewritten
  - Communications, often hierarchy (fast bus (PCIe), slower SPI, ADC)
- Partner
  - What's on the other end? Human or Machine
- Data Rate
  - Peak Rate of transfer between I/O and Memory or CPU
- Example
  - Keyboard → Input Device → Used by Human → 10 B/s
  - Robotic sensors and actuators
- Today peripherals are usually complex, even sensor or input requires setup, output (i.e. motor driver) requires monitoring

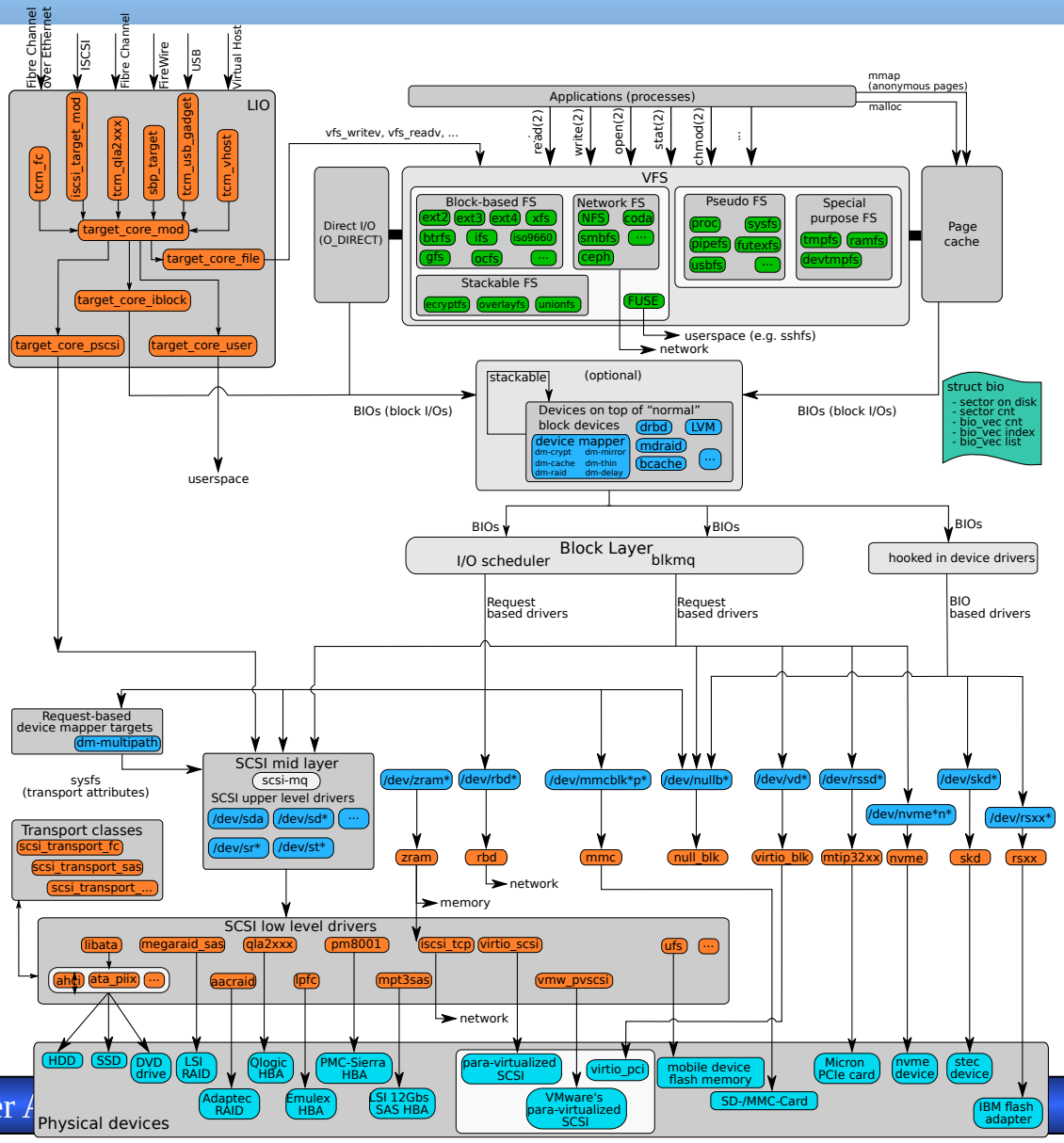
Diversity of Use and Needs of IO Devices

Device	Behavior	Partner	Data rate (Mbit/s)	Max latency (msec)
Keyboard	Input	Human	0.0001	200
Mouse	Input	Human	0.0038	50
Voice input	Input	Human	0.2640	10 / 0.02
Sound input	Input	Machine	3.0000	10 / 0.02
Scanner	Input	Human	3.2000	na
Voice output	Output	Human	0.2640	10 / 0.02
Sound output	Output	Human	8.0000	10 / 0.02
Laser printer	Output	Human	3.2000	na
Graphics display	Output	Human	800 - 8000	40 / nsec range
Cable modem	Bidirectional	Machine	0.1 - 100	100 / nsec range
Network/WLAN	Bidirectional	Machine	11-8000	10 / nsec range
Magnetic disk	Storage	Machine	800-3000	30
Flash disk NVMe	Storage	Machine	Up to 30000	10
DC motor control	Actuator+FB	Environment	0.080	0.05 – 0.5
PMSM DQ control	Actuator+FB	Environment	0.8	0.01 – 0.05

Layers of the I/O Software System

- User application communicating with device
- User/level I/O software
- Operating systems interfaces (lecture about Syscall)
  - Device-independent operating system software
  - Generic subsystems interfaces (filesystems, block management and queues, networking, terminals, GPIO)
  - Device drivers – function interfaces
  - Device drivers – physical devices and bus drivers
  - Interrupt handlers and or direct memory access (DMA)
  - Hardware

Linux Storage Stack Diagram (Simplified but Complex Still)

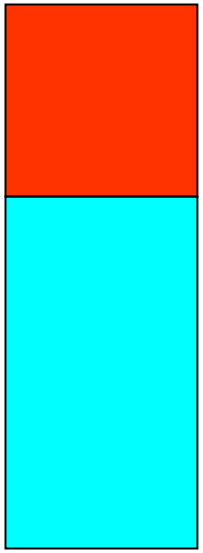


Source: Werner Fischer and Georg Schönberger – simplified

Memory-mapped I/O

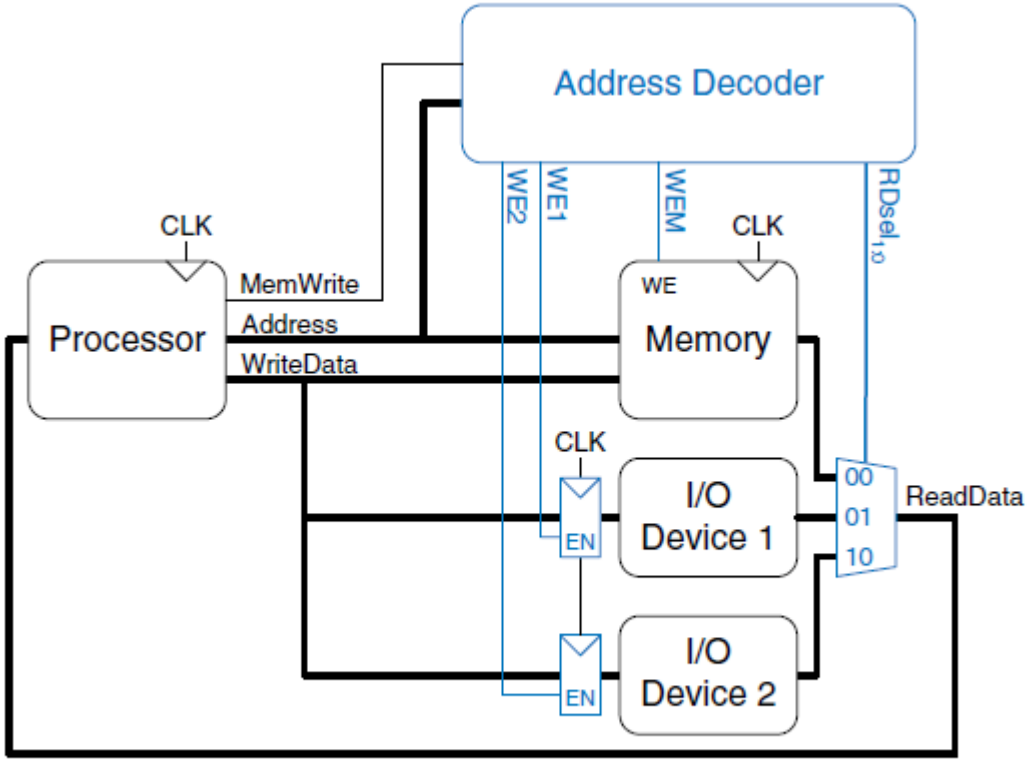
- The idea: Processors can use the interface used for memory access (RISC-V: **`lw`**, **`sw`** instructions) to communicate with *input/output (I/O)* devices such as keyboards, monitors,

Common address space for I/O and memory



memory-mapped peripheral (I/O) registers

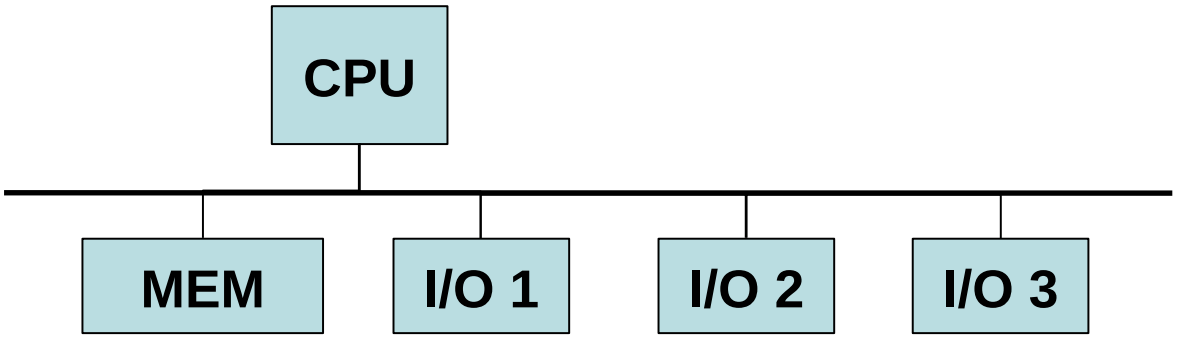
Main/system memory

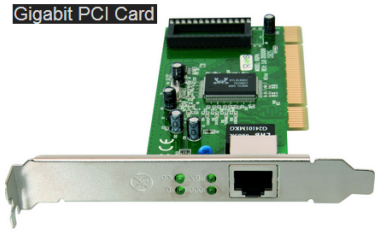


Support hardware for memory-mapped I/O

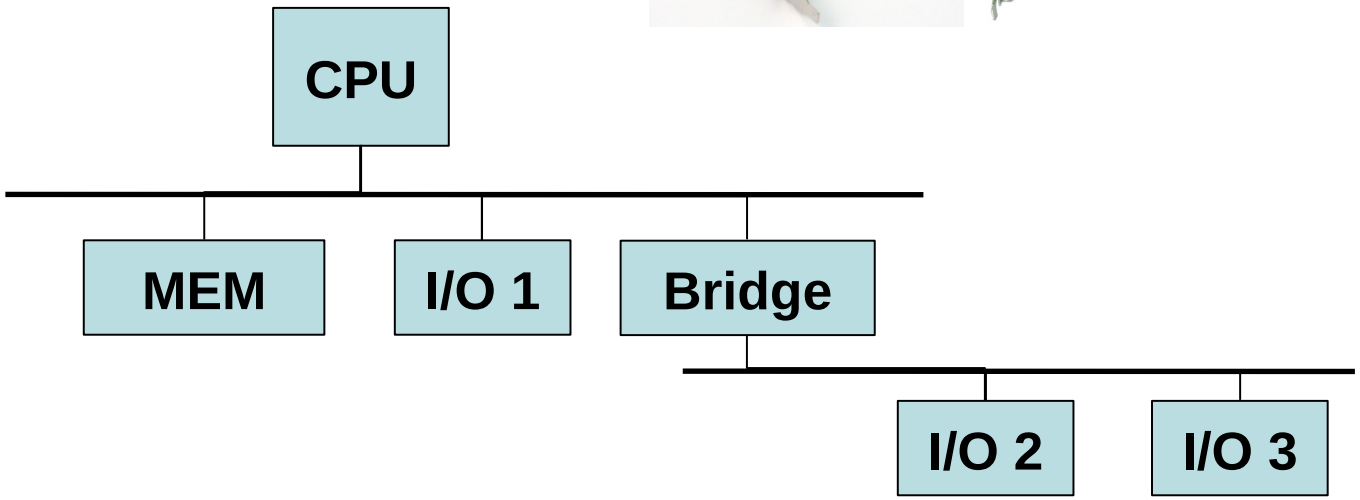
Address Decoder – Idea

- Logical Structure:
- (Illusion)



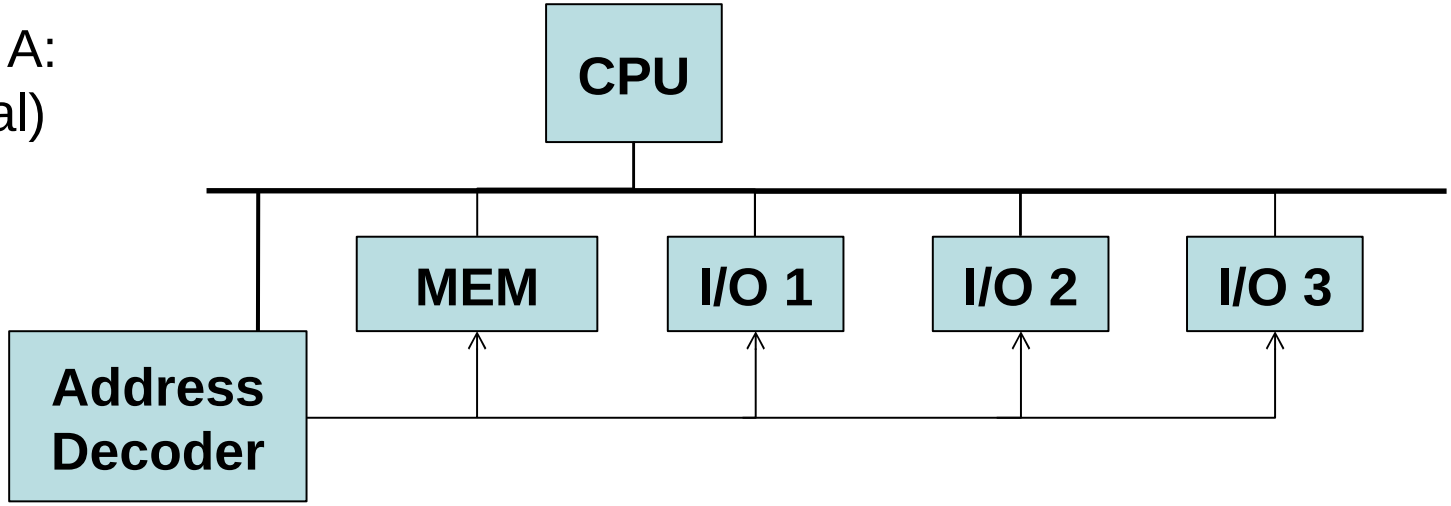


- Possible physical arrangement :

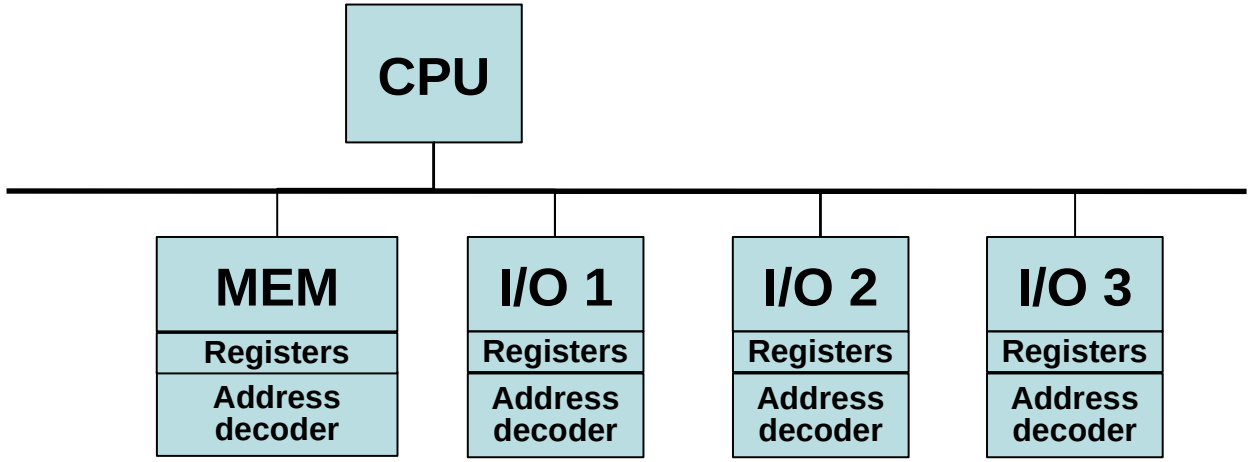


Address Decoder – Central or on Addon Boards

- Option A:
  (Central)

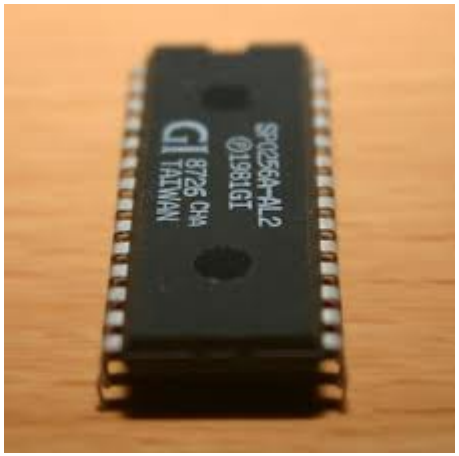


- Option B:
- (Autonomous)

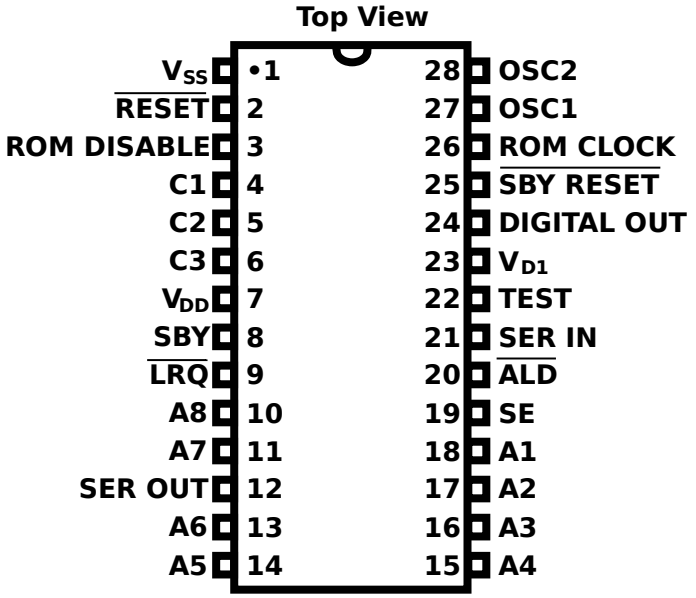


Example: Speech Synthesizer – Hardware

- Words are composed of one or more *allophones*, the *fundamental* units of sound. The 64 different allophones appear in the English language.
- **Problem**: Integrate HW support and write synthesizer driver
- Simplified assumption: 5 units (allophones) are placed at address 0x10000000. They are read by driver and sent to SP0256 synthesizer chip.

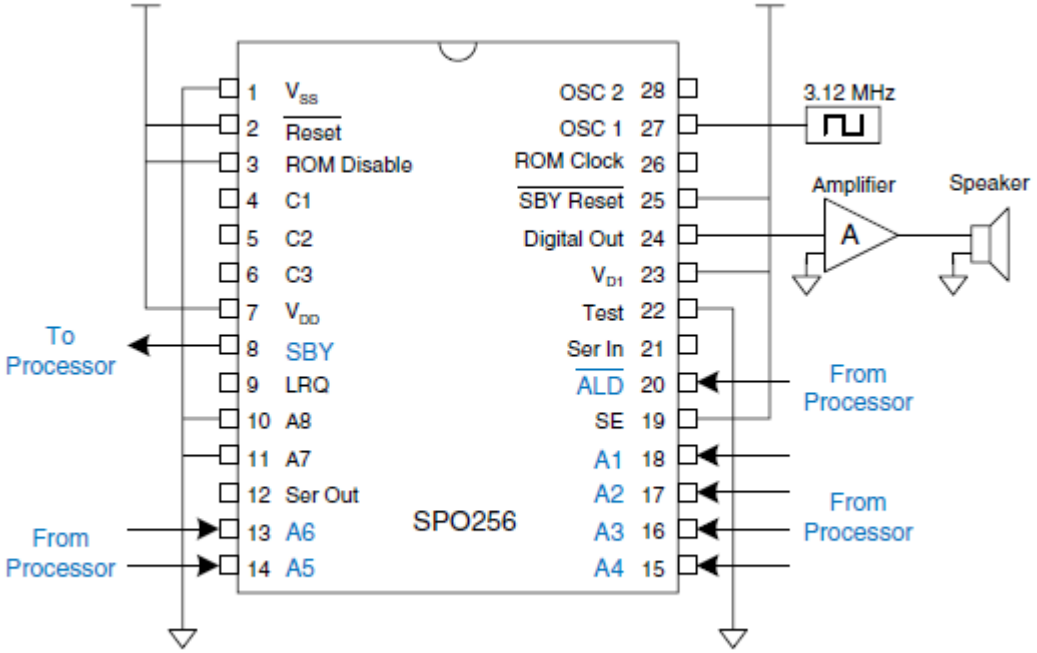


http://little-scale.blogspot.cz/2009/02/sp0256-al2-creative-commons-sample-pack.html

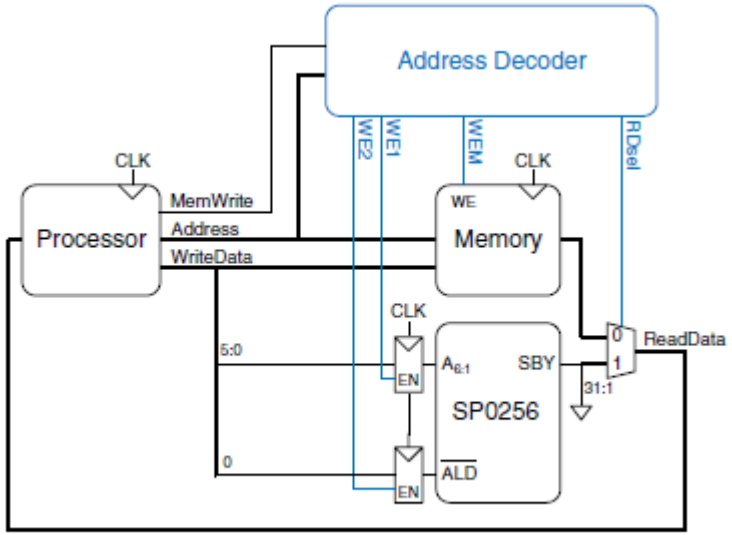


Czech: allophone – artikulace určitého fonému

Example: Speech Synthesizer – Integration



SPO256 speech synthesizer chip pinout



Hardware for driving the SPO256 speech synthesizer

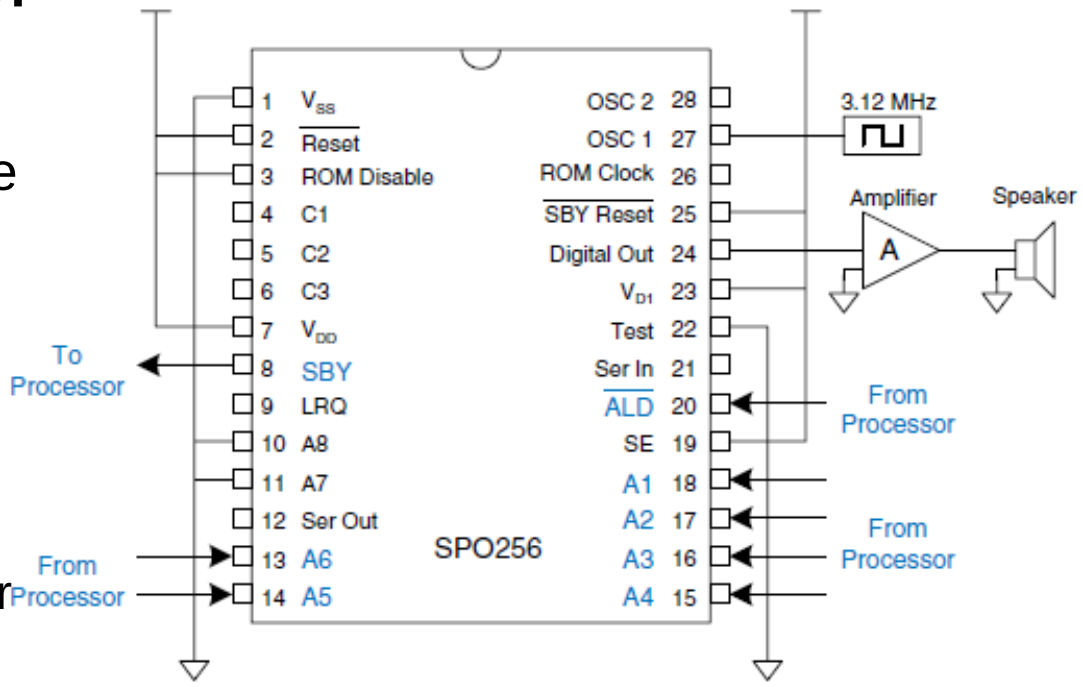
- When the **SBY** output is 1, the speech chip is standing by and is **ready** to receive a new allophone. On the falling edge of the address load input **ALD#**, the speech chip reads the allophone specified by **A6:1**.
- We arbitrarily have chosen that the **A6:1** port is mapped to address **0xFFFFFF00**, **ALD#** to **0xFFFFFF04**, and **SBY** to **0xFFFFFF08**.

Example: Speech Synthesizer – Driver

The device driver controls the speech synthesizer by sending an appropriate series of allophones over the memory-mapped I/O interface. It follows the protocol expected by the SPO256 chip, given below:

1. Set **ALD#** to 1
2. Wait until the chip asserts **SBY** *to indicate that it is finished speaking* the previous allophone and is ready for the next
3. Write a 6-bit code selecting allophone to **A6:1**
4. Reset **ALD#** *to 0 to initiate speech*

This sequence can be repeated for any number of allophones and speech is synthesized



SPO256 speech synthesizer chip pinout

Example: Speech Synthesizer – Driver on RISC-V

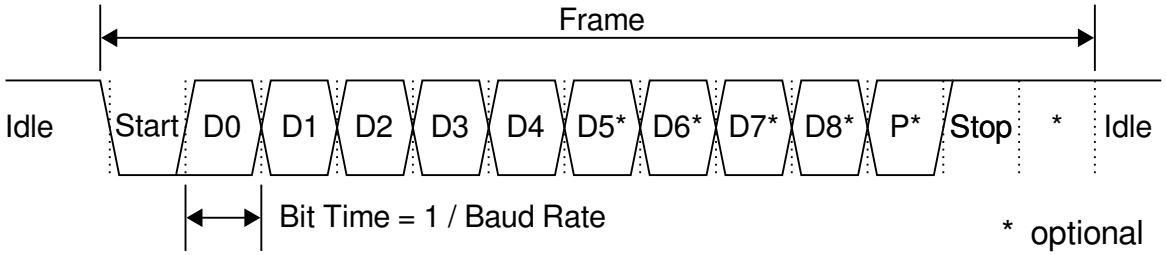
1. Set **ALD#** to 1
2. Wait until the chip asserts **SBY** *to indicate that it is finished speaking* the previous allophone and is ready for the next
3. Write a 6-bit allophone code to **A6:1**
4. Reset **ALD#** *to 0 to initiate speech*

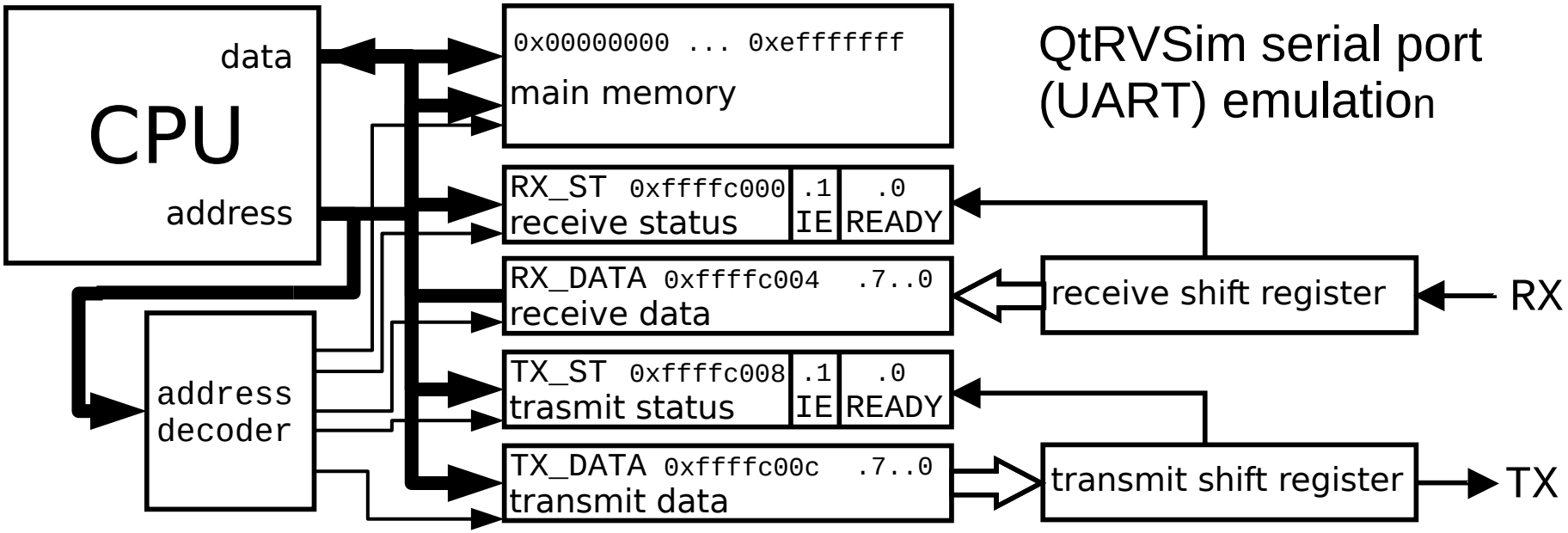
Notice polling loop to check for ready to speak condition. CPU is blocked to do useful work.

```
init:
  li   s0,0xffffff00  // s0 peripheral base address
  addi t1,zero,1      // t1 = 1 (value to write to ALD#)
  addi t2,zero,20     // t2 = array size ×4 (20 bytes)
  li   t3,0x1000      // t3 = array base address
  addi t4,zero,0      // t4 = 0 (array index)
start:
  sw t1,0x04(s0)      // ALD#=1
loop:
  lw t5,0x08(s0)      // t5 = SBY (monitor state)
  beq zero,t5,loop    // loop until SBY == 1
  add t5,t3,t4        // t5 = address of allophone
  lw  t5,0(t5)        // t5 = allophone
  sw t5,0x00(s0)      // A6:1 = allophone
  sw zero,0x04(s0)    // ALD# = 0 (to initiate speech)
  addi t4,t4,4        // increment array index
  beq t4,t2,done      // all allophone in array done?
  j start             // repeat
done:
```

Instead of polling, the processor could use an *interrupt connected to SBY. When SBY rises,* the processor stops what it is doing and jumps to code that handles the interrupt.

Serial Port – UART, The First Chance to Say Hello for CPU





Source: Wikipedia

QtRVSim Serial Port Addresses

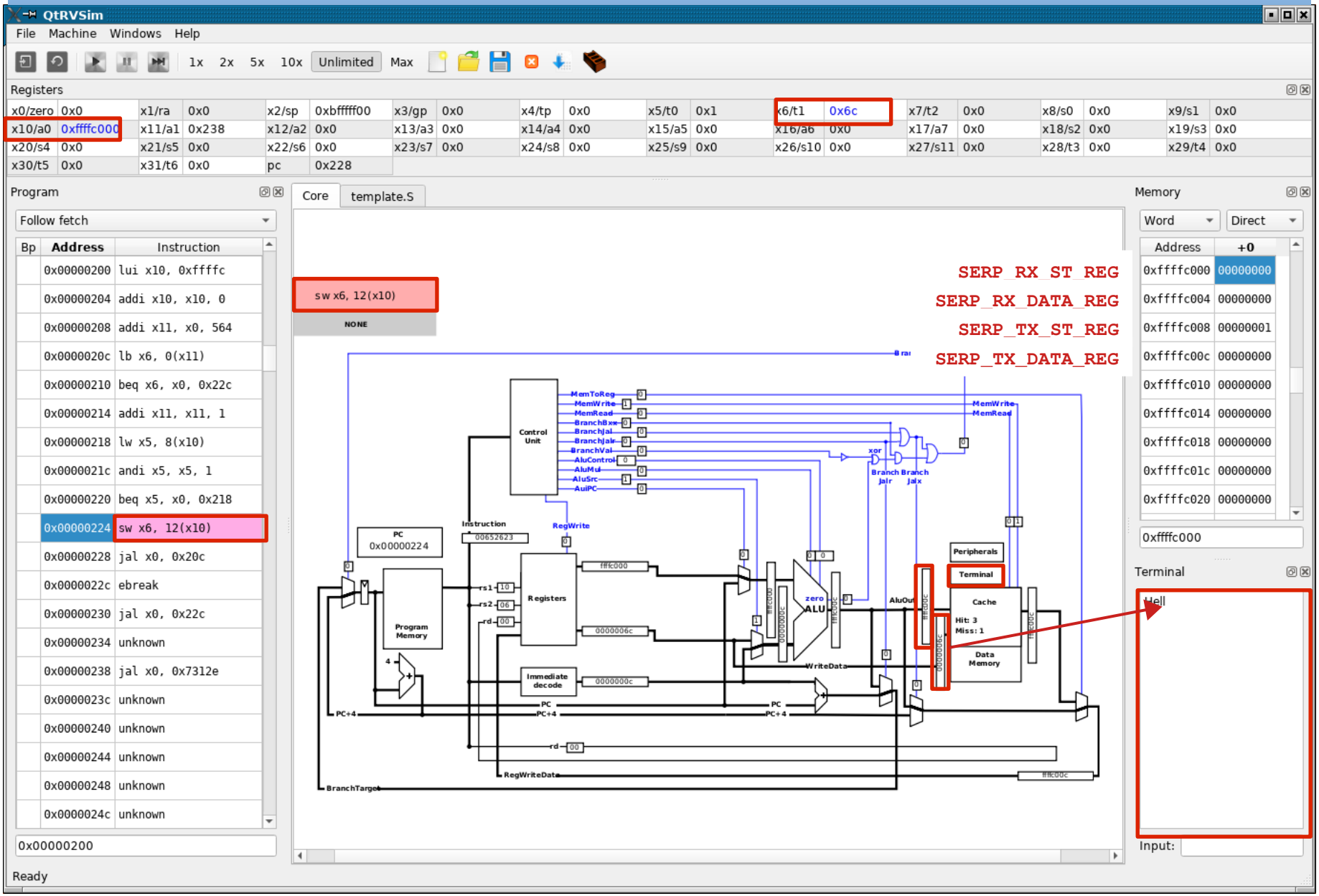
SERIAL_PORT_BASE 0xffffc000
base address of QtRVSim serial port, mirrors on 0xffff0000 for MARS and QtSpim

SERP_RX_ST_REG	**0xffffc000**	Receiver status register
SERP_RX_ST_REG_o	0x0000	Offset of RX_ST_REG
SERP_RX_ST_REG_READY_m	0x1	Data byte is ready to be read
SERP_RX_ST_REG_IE_m	0x2	Enable Rx ready interrupt
SERP_RX_DATA_REG	**0xffffc004**	Received data byte in 8 LSB bits
SERP_RX_DATA_REG_o	0x0004	Offset of RX_DATA_REG
SERP_TX_ST_REG	**0xffffc008**	Transmitter status register
SERP_TX_ST_REG_o	0x0008	Offset of TX_ST_REG
SERP_TX_ST_REG_READY_m	0x1	Transmitter can accept next byte
SERP_TX_ST_REG_IE_m	0x2	Enable Tx ready interrupt
SERP_TX_DATA_REG	**0xffffc00c**	Write word to send 8 LSB bits to terminal
SERP_TX_DATA_REG_o	0x000c	Offset of TX_DATA_REG

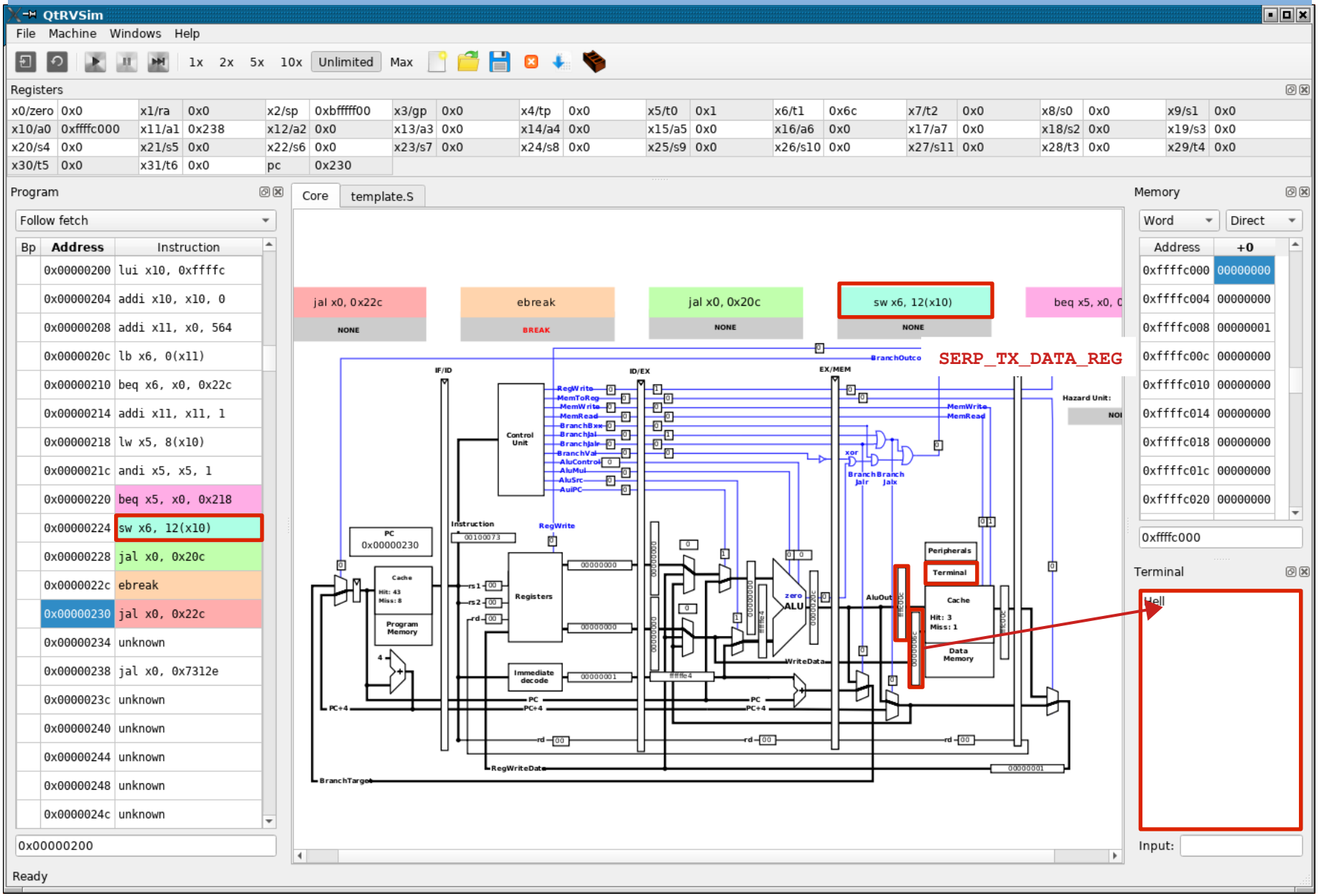
QtRVSim Write Hello World to Serial Port

```
loop:
    li   a0, SERIAL_PORT_BASE // load base address of serial port
    addi a1, zero, text_1     // load address of text
next_char:
    lb   t1, 0(a1)            // load one byte after another
    beq  t1, zero, end_char   // is this the terminal zero byte
    addi a1, a1, 1            // move pointer to next text byte
tx_busy:
    lw   t0, SERP_TX_ST_REG_o(a0)  // read status of transmitter
    andi t0, t0, SERP_TX_ST_REG_READY_m  // mask ready bit
    beq  t0, zero, tx_busy    // if not ready wait for ready condition
    sw   t1, SERP_TX_DATA_REG_o(a0)  // write byte to Tx data register
    jal  zero, next_char      // unconditional branch to process next byte
end_char:
    ebreak                    // stop continuous execution
    jal  zero, end_char

.data
text_1:
    .asciz  "Hello world.\n"  // store zero terminated ASCII text
```

QtRVSim Serial Port – Single Cycle



QtRVSim Serial Port – Pipelined



QtRVSim – Simple I/O Peripherals

base of SPILED port region

SPILED_REG_BASE 0xffffc100

RGB LED 1 color components – 8 bits each

SPILED_REG_LED_RGB1 0xffffc110

SPILED_REG_LED_RGB1_o 0x0010

RGB LED 2 color components – 8 bits each

SPILED_REG_LED_RGB2 0xffffc114

SPILED_REG_LED_RGB2_o 0x0014

Three 8 bit knob values

SPILED_REG_KNOBS_8BIT 0xffffc124

SPILED_REG_KNOBS_8BIT_o 0x0024

line of 32 LEDs for binary value representation

SPILED_REG_LED_LINE 0xffffc104

SPILED_REG_LED_LINE_o 0x0004

QtRVSim Peripherals Documentation and Frame Buffer

- All peripherals supported by QtRVSim are described at QtRVSim project page (README.md file)

  https://github.com/cvut/qtrvsim/#peripherals
- The simple 16-bit per pixel (RGB565) framebuffer

  The size corresponds to MZ_APO 480 x 320 pixel display

  components
  - bits 11 .. 15 red
  - bits 5 .. 10 green
  - bits 0 .. 4 blue

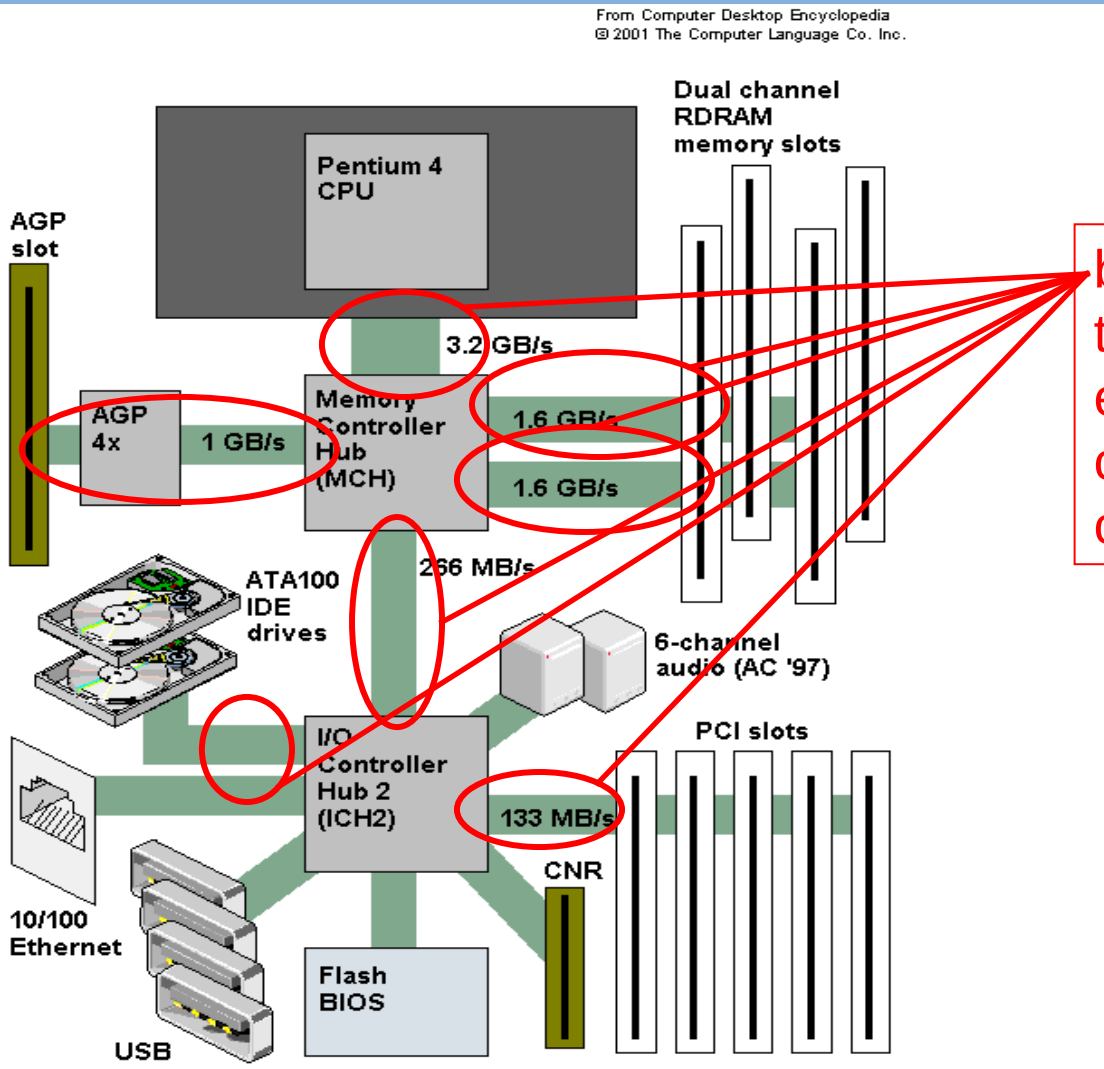
  Frame buffer starts at address
  - **`LCD_FB_START`** `0xffe00000`
  - **`LCD_FB_END`** `0xffe4afff`

Generalized Summary Based on Example

- There are two methods for I/O devices (peripherals) access
  - memory mapped I/O
  - I/O specialized instructions (if implemented/available) – they use address space independent of memory access
- There are address range(s) dedicated to device access in the case of memory mapped I/O. Reads/writes from/to these addresses are interpreted as commands or data transfers from/to peripheral devices. Memory subsystem is informed about I/O ranges and ignores these accesses. I/O devices/bus controller is aware of addresses assigned to it and fulfills requests.
- The CPU can be informed about I/O device request for service by:
  - repeated monitoring of its ready condition (status register) – **polling**
  - interrupt request – **interrupt-driven I/O** – it is asynchronous to the actual program execution (is initiated by device when it needs servicing)
- Have you noticed address decoder function?
- What about caches in the case of I/O range/region access?

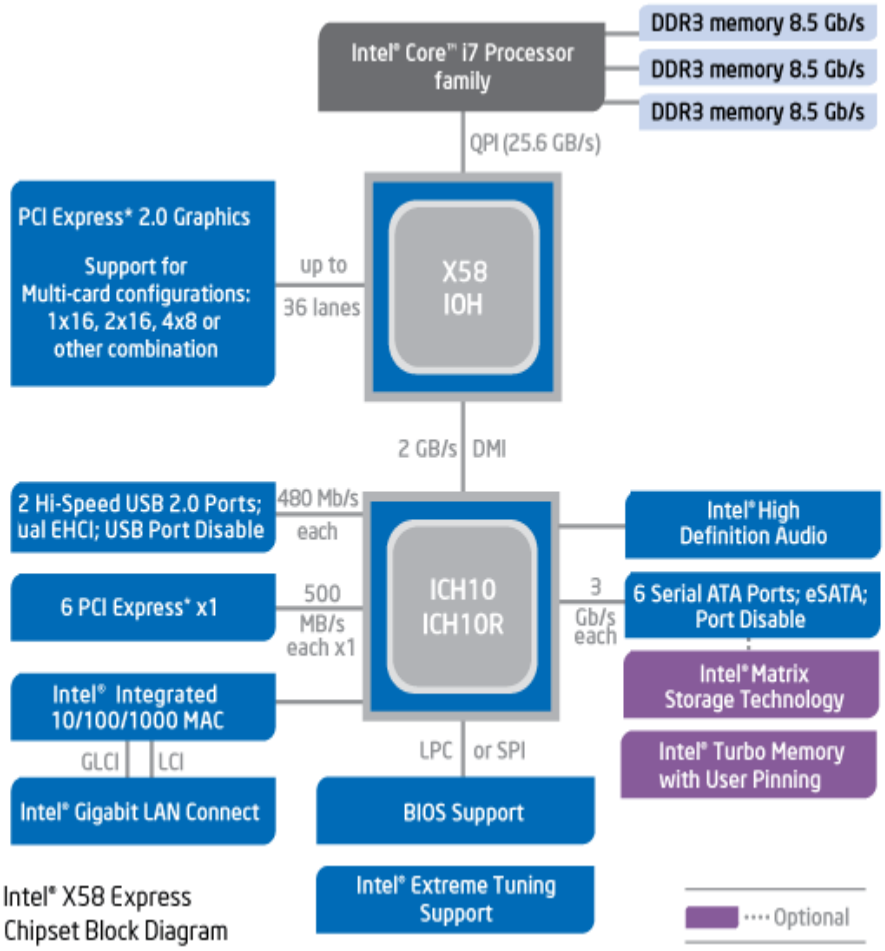
Use of Buses for Input/Output on Real Systems

Motivation to Use and Study Buses

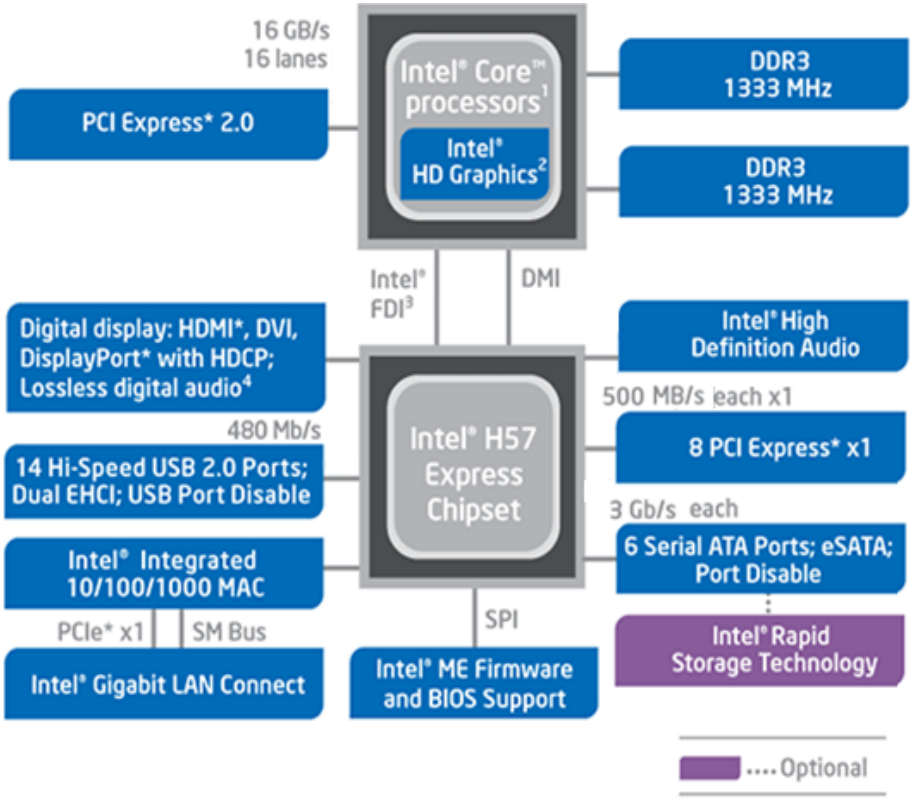


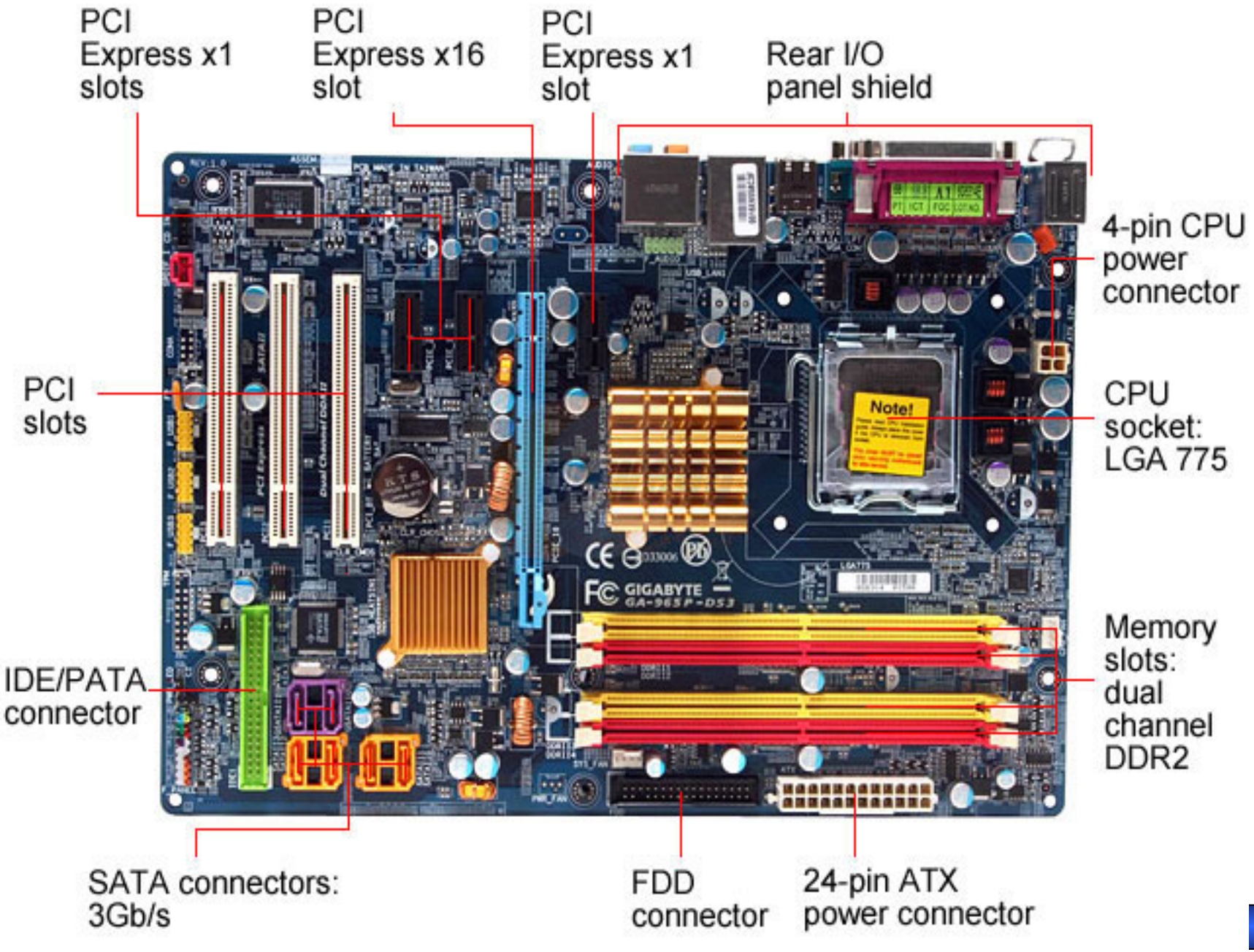
bus interconnection technologies are used everywhere to connect computer system components/subsystems

Motivation – Intel – Only as an Example



Intel® X58 Express Chipset Block Diagram




PCI Express x1 slots
PCI Express x16 slot
PCI Express x1 slot
Rear I/O panel shield
4-pin CPU power connector
PCI slots
CPU socket: LGA 775
Memory slots: dual channel DDR2
IDE/PATA connector
SATA connectors: 3Gb/s
FDD connector
24-pin ATX power connector
Note!
GIGABYTE
GA-965P-DS3

What is the Main Task of I/O Subsystem?

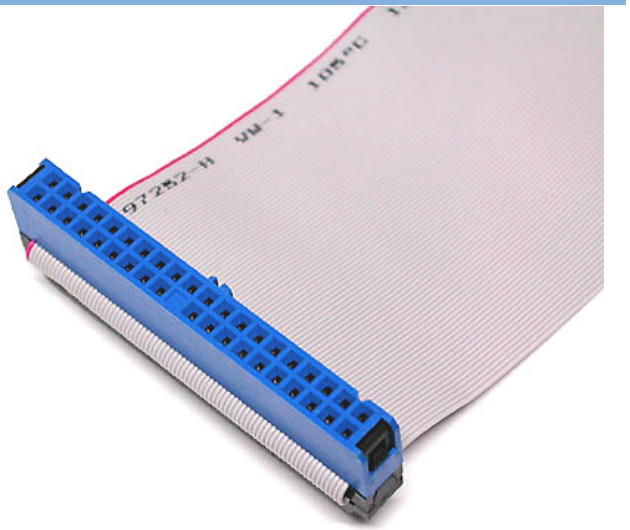
- Interconnection of subsystems inside computer, connection of external peripherals and computers together
- Demands on I/O subsystem:

  Creating optimal data paths, especially critical for the most demanding peripherals (graphic cards, external memories)
- Possible solutions:

  There have to be compromises due to price/performance ratio when it is
  - possible to share data paths, or
  - advantageous to share them.

Some Other Examples and Solutions

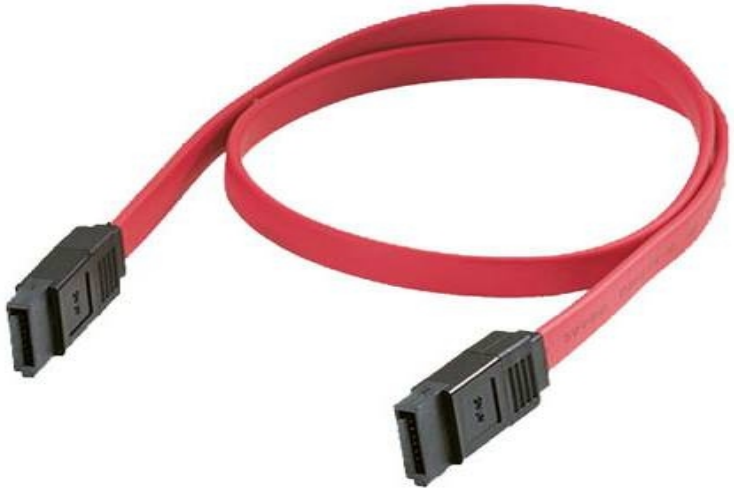
- Do you know Parallel **ATA** (PATA)?
- Integrated Drive Electronics (**IDE**) nebo **EIDE** (Enhanced IDE) from Western Digital may be more known term
- ATA = *Advanced Technology Attachment*

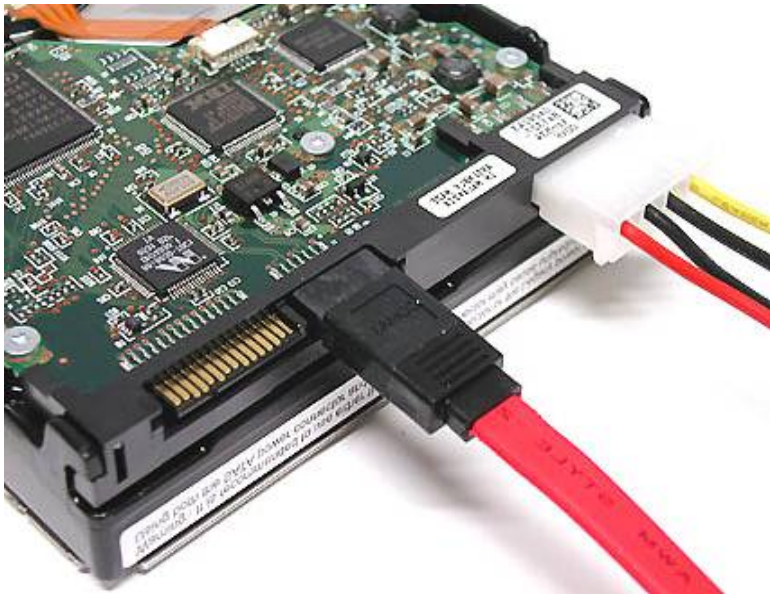


- **It has been the most used interface to connect hard-drives and optical units**
- 40-pin header -> 40 leads (16 of these for data)
- later 80 leads used for better signal integrity (shielding), but 40-pin header preserved

Serial ATA – Solution Used Today

- **Serial ATA** (SATA) more used today
- SATA 1.0: 150 MB/s (PATA:130MB/s)
- SATA 2.0: 300 MB/s
- SATA 3.0: 600 MB/s
- SATA 3.2: about 2 GB/s





http://en.wikipedia.org/wiki/Serial_ATA

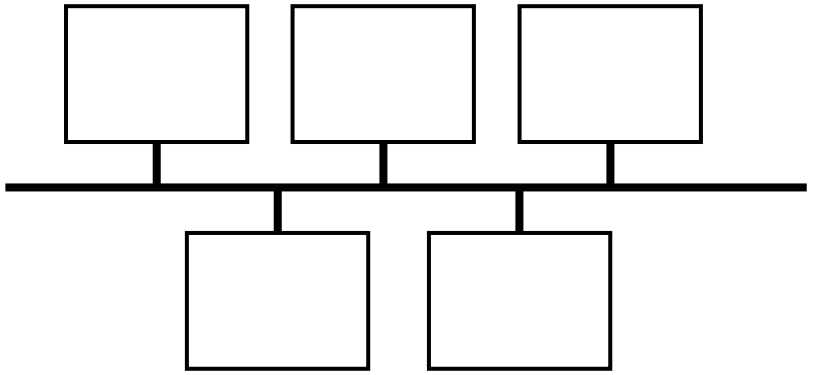
- **Interface used for drives and optical units connection today**
- 7 leads only!!!

Pin	Mating	Function
1	1st	Ground
2	2nd	A+ (Transmit)
3	2nd	A− (Transmit)
4	1st	Ground
5	2nd	B− (Receive)
6	2nd	B+ (Receive)
7	1st	Ground

Interfacing Terminology – Important Terms:

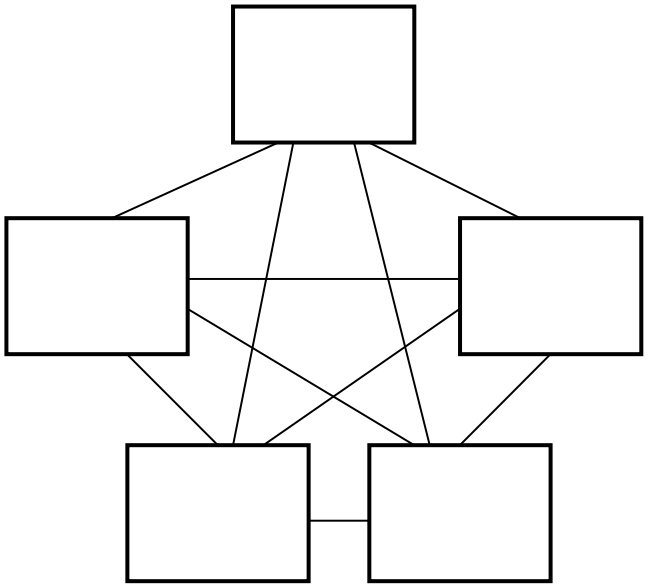
- Interface
  - Common communication part shared by two systems, equipment or programs.
  - Includes also boundary and supporting control elements necessary for their interconnection.
- Bus × point-to-point connection.
- Address, data, control bus.
- Multiplexed/separate bus.
- Processor, system, local, I/O bus.
- Bus cycle, bus transaction.
- Open collector, 3-state output, switched multiplexers
- Initiator/target

Reminder: bus x point-to-point connection

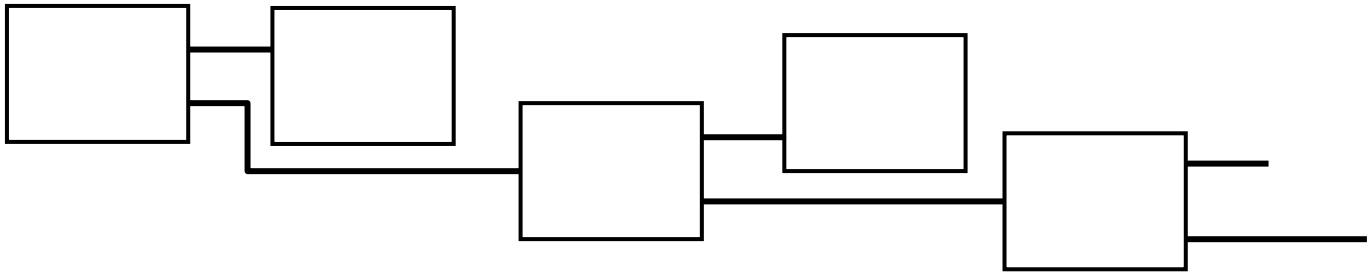


Bus – shared data path

Remark: logical topology as seen from computer system inside (OS, programs) can differ from the physical topology



Point-to-point connection

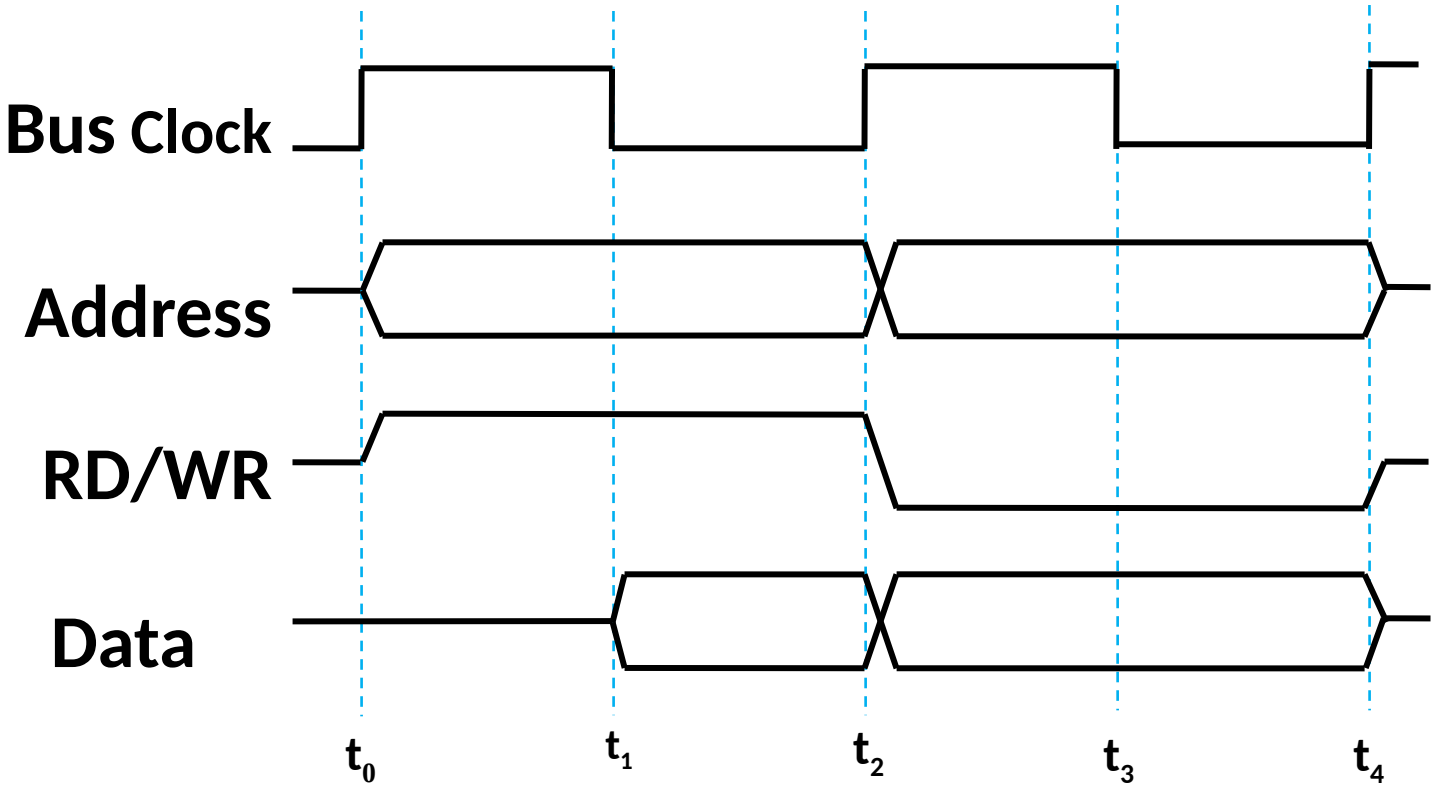


Many different combinations in between in real systems

Synchronous Parallel Data Transfer

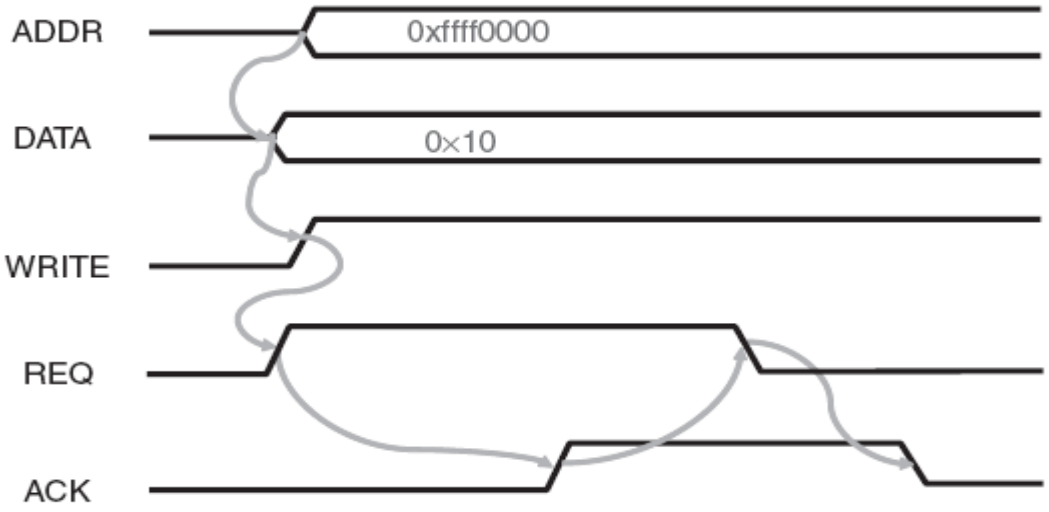
The events are determined by a clock!

Good synchronization of all signals is absolutely essential!



Asynchronous Bus

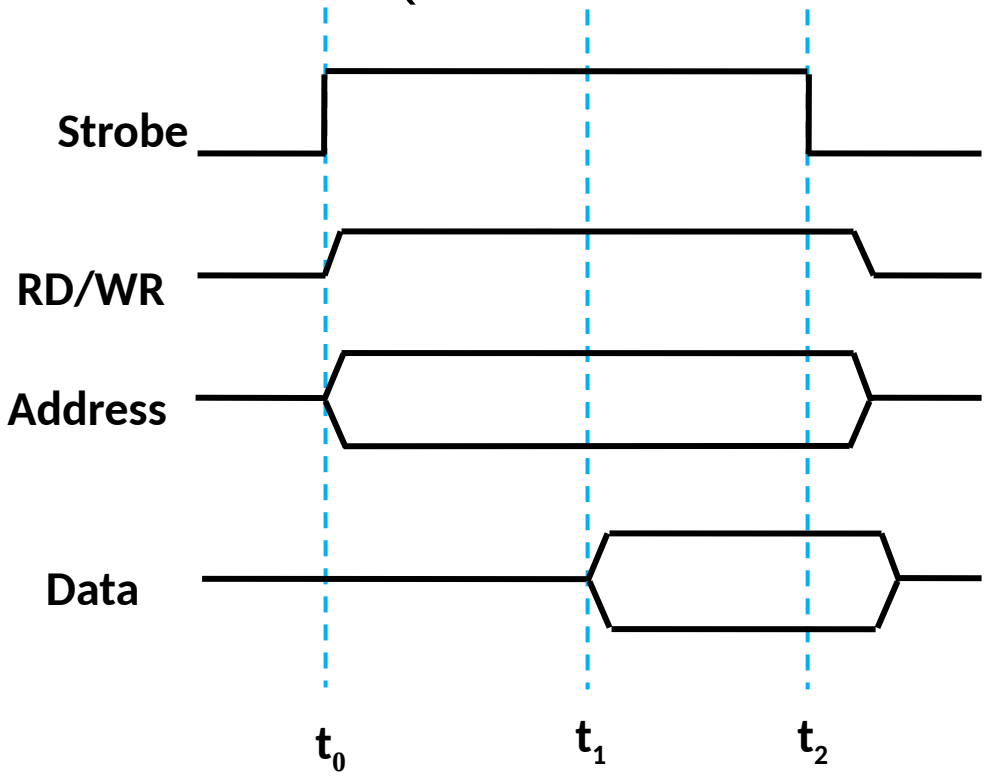
- Not clocked
- Requires a handshaking protocol (*It behaves as TCP internet protocol*)
  - performance not as good as that of synchronous bus
  - No need for frequency converters, but does need extra lines
- Does not suffer from clock skew like the synchronous bus



Source: Sudeep Pasricha,
On Chip Communication, Colorado 2011

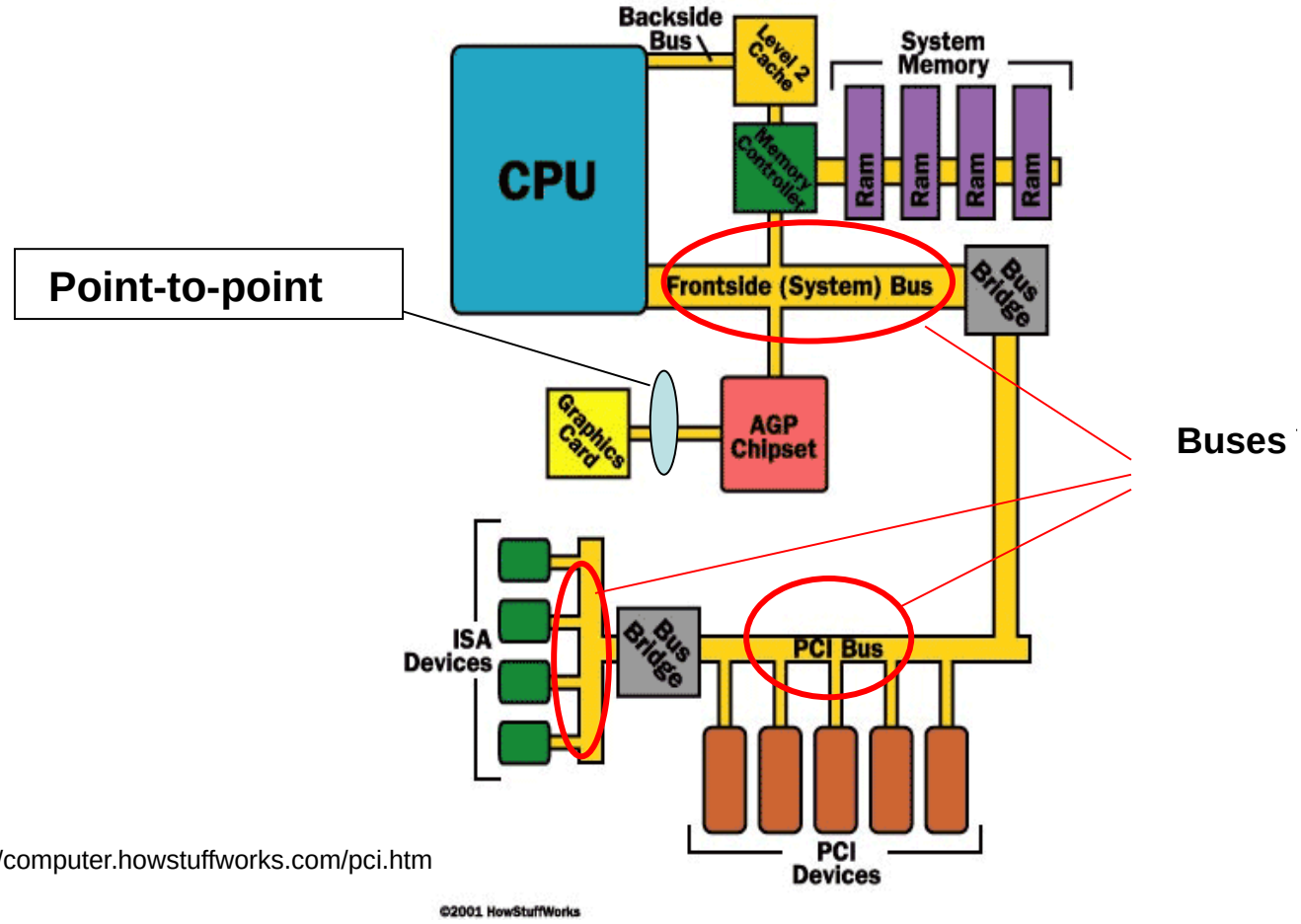
Asynchronous Parallel Data Transfer by Strobing

Strobing accelerates asynchronous bus operations,
but it is less reliable. (*It behaves as UDP internet protocol*)



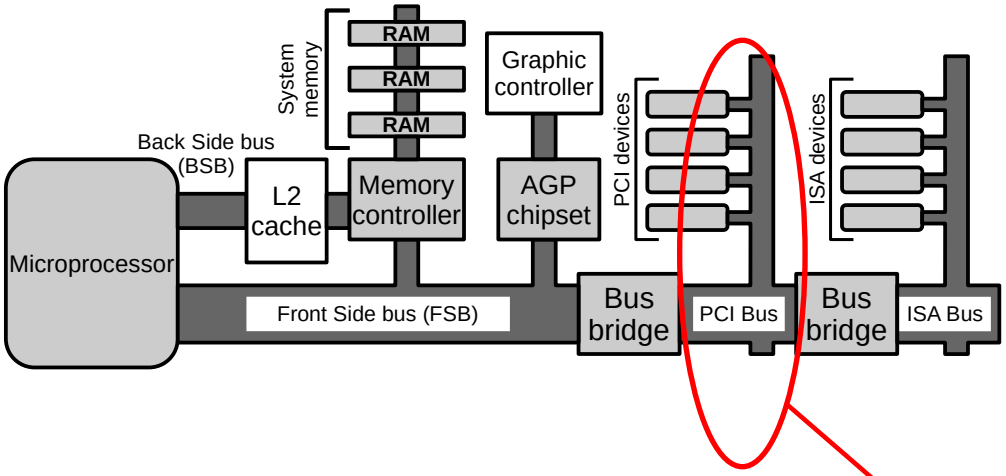
Source: Sudeep Pasricha,
On Chip Communication, Colorado 2011

PC Architecture (cca 2000+) … Based on PCI



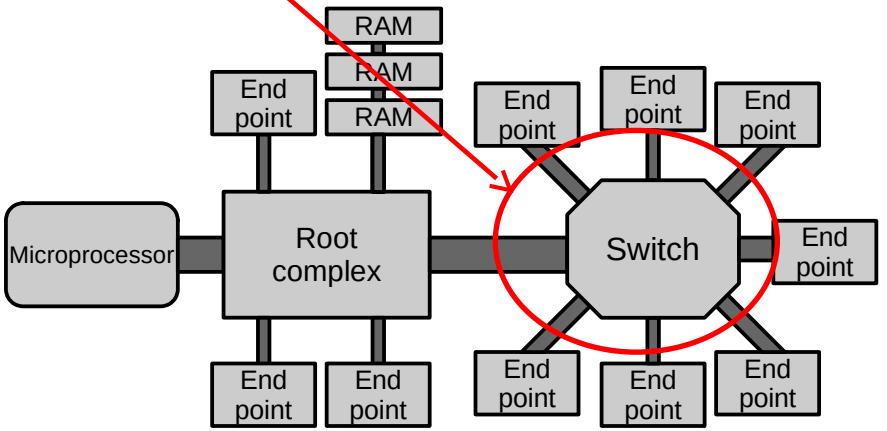
Source: http://computer.howstuffworks.com/pci.htm

PCIe Architecture - Bus is Replaced by Shared Switch



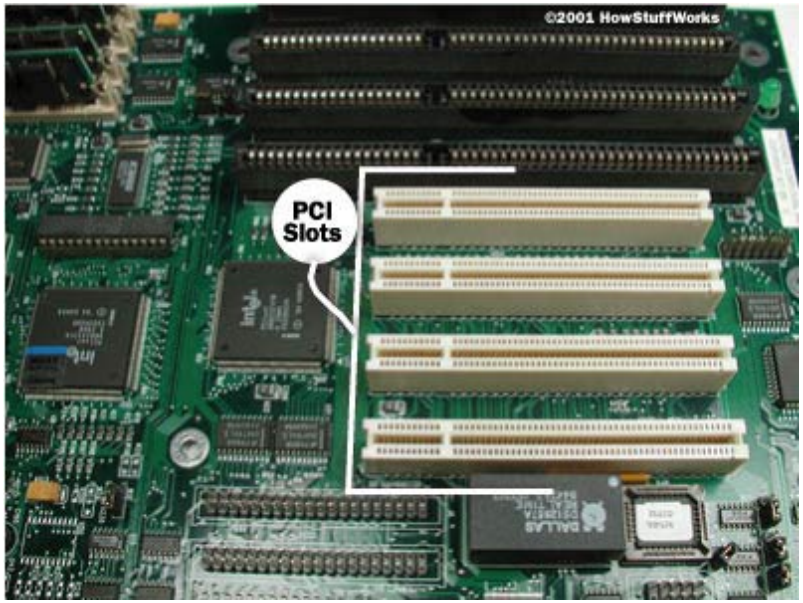
More details later …

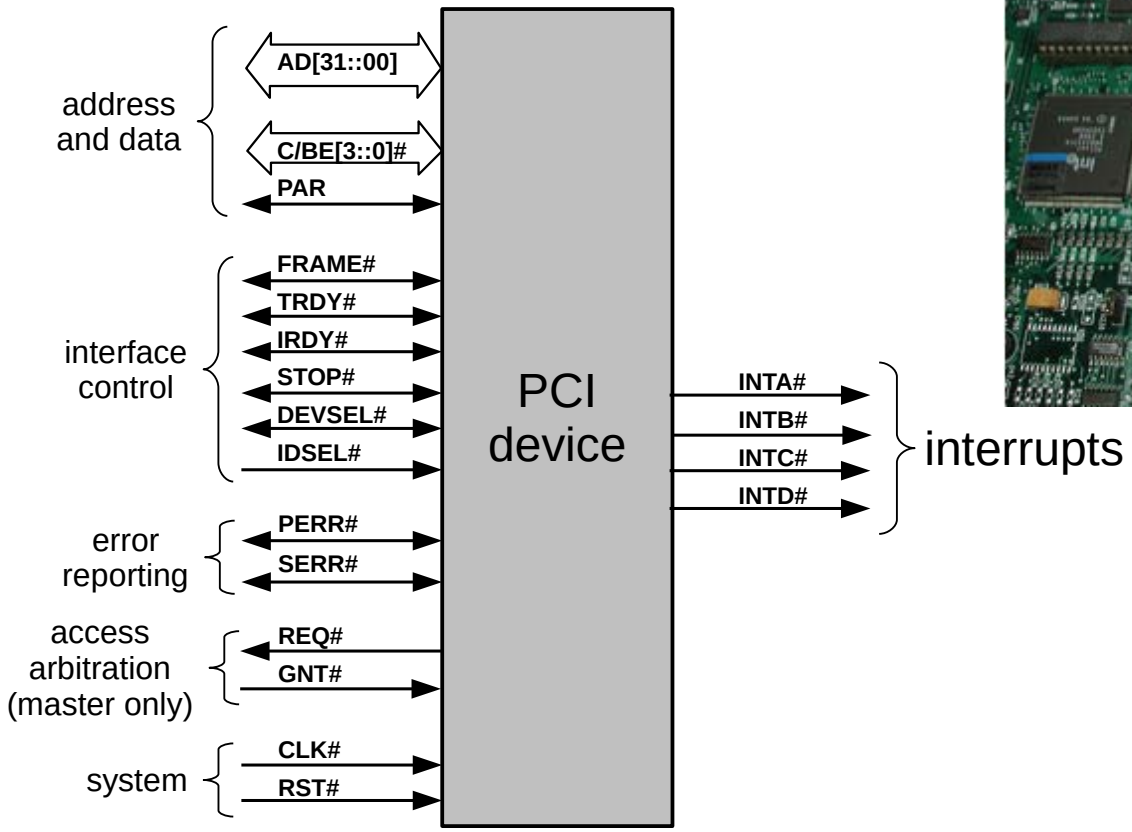
PCIe = PCI Express



PCI Bus – Peripheral Component Interconnect

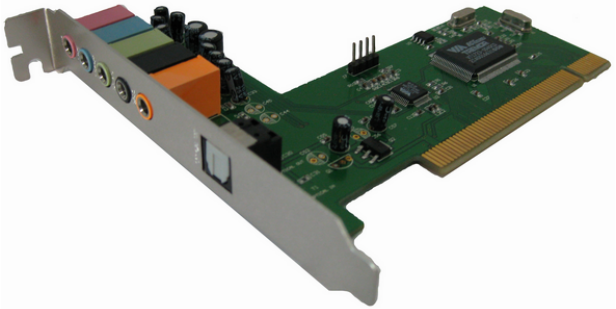
PCI Signals – 32-bit Version



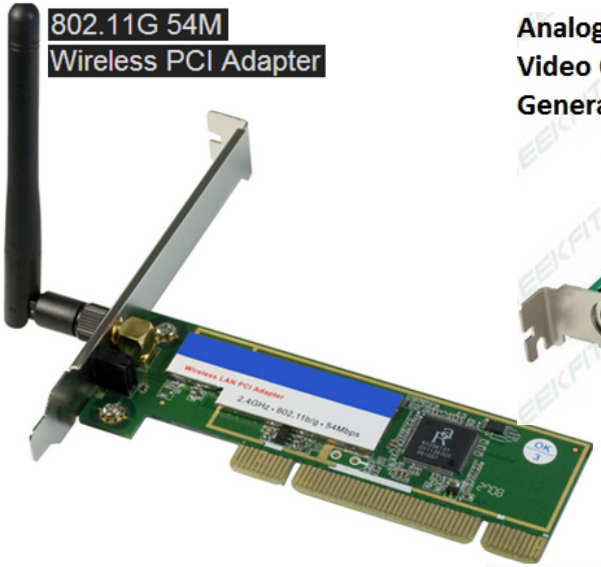


PCI Devices Examples

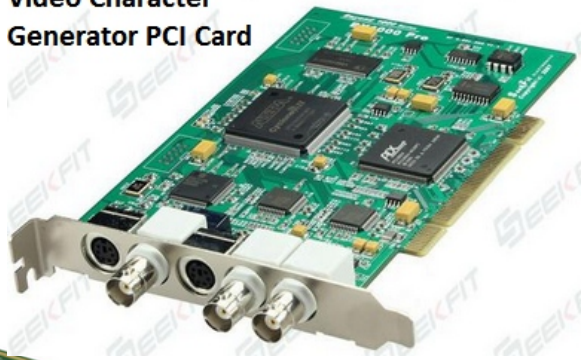
PCI Sound Card 6 Channel



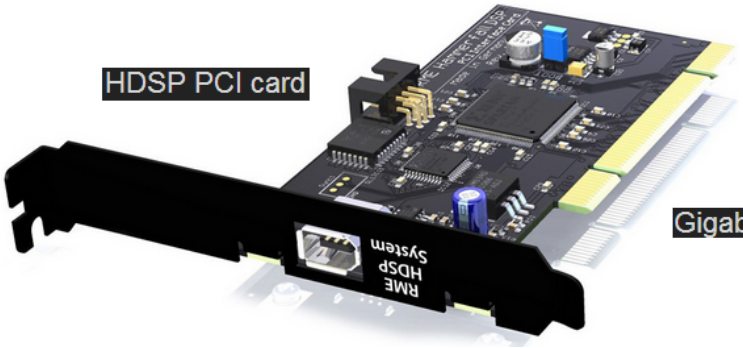
802.11G 54M
Wireless PCI Adapter



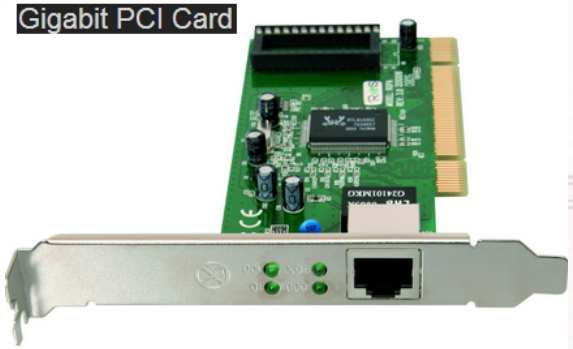
Analog Cvbs/S-Video
Video Character
Generator PCI Card



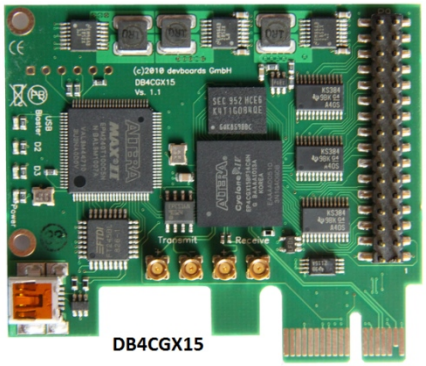
HDSP PCI card



Gigabit PCI Card



USB 2.0 + 1394 Firewire
Combo PCI Card



DB4CGX15

PCI 32-bit Bus Signals

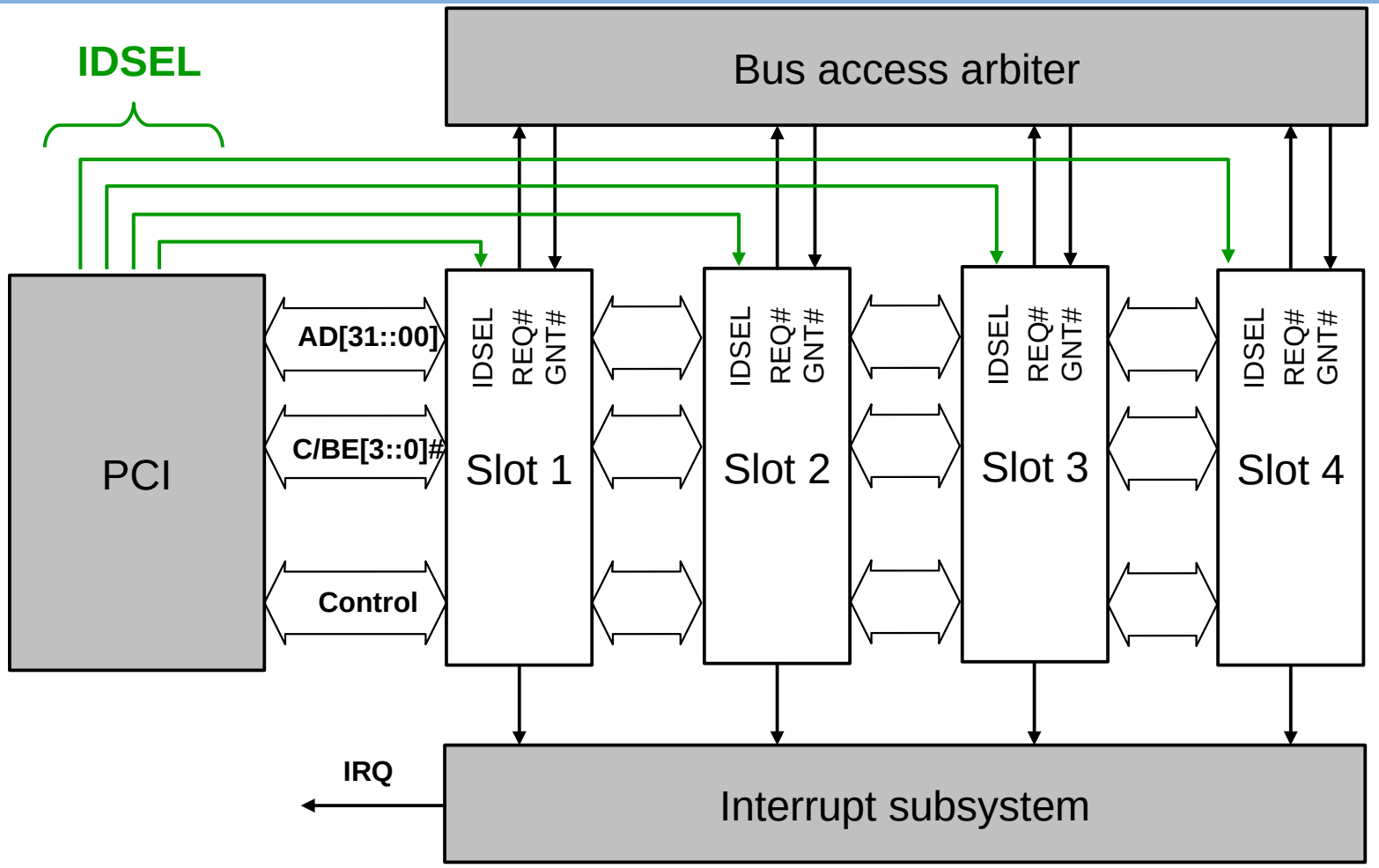
Pin #	Name	PCI Pin Description	Pin #	Name	PCI Pin Description
A1	TRST	Test Logic Reset	B1	-12V	-12 VDC
A2	+12V	+12 VDC	B2	TCK	Test Clock
A3	TMS	Test Mde Select	B3	GND	Ground
A4	TDI	Test Data Input	B4	TDO	Test Data Output
A5	+5V	+5 VDC	B5	+5V	+5 VDC
A6	INTA	Interrupt A	B6	+5V	+5 VDC
A7	INTC	Interrupt C	B7	INTB	Interrupt B
A8	+5V	+5 VDC	B8	INTD	Interrupt D
A9	-----	Reserved	B9	PRSNT1	Present
A10	+5V	Power (+5 V or +3.3 V)	B10	-----	Reserved
A11	-----	Reserved	B11	PRSNT2	Present
A12	GND03	Ground or Keyway for 3.3/Universal PWB	B12	GND	Ground or Keyway for 3.3/Universal PWB
A13	GND05		B13	GND	
A14	3.3Vaux	-----	B14	RES	Reserved
A15	RESET	Reset	B15	GND	Ground
A16	+5V	Power (+5 V or +3.3 V)	B16	CLK	Clock
A17	GNT	Grant PCI use	B17	GND	Ground
A18	GND08	Ground	B18	REQ	Request
A19	PME#	Power Managment Event	B19	+5V	Power (+5 V or +3.3 V)
A20	AD30	Address/Data 30	B20	AD31	Address/Data 31
A21	+3.3V01	+3.3 VDC	B21	AD29	Address/Data 29
A22	AD28	Address/Data 28	B22	GND	Ground
A23	AD26	Address/Data 26	B23	AD27	Address/Data 27
A24	GND10	Ground	B24	AD25	Address/Data 25
A25	AD24	Address/Data 24	B25	+3.3V	+3.3VDC
A26	IDSEL	Initialization Device Select	B26	C/BE3	Command, Byte Enable 3
A27	+3.3V03	+3.3 VDC	B27	AD23	Address/Data 23
A28	AD22	Address/Data 22	B28	GND	Ground
A29	AD20	Address/Data 20	B29	AD21	Address/Data 21
A30	GND12	Ground	B30	AD19	Address/Data 19
A31	AD18	Address/Data 18	B31	+3.3V	+3.3 VDC

Pin #	Name	PCI Pin Description	Pin #	Name	PCI Pin Description
A32	AD16	Address/Data 16	B32	AD17	Address/Data 17
A33	+3.3V05	+3.3 VDC	B33	C/BE2	Command, Byte Enable 2
A34	FRAME	Address or Data phase	B34	GND13	Ground
A35	GND14	Ground	B35	IRDY#	Initiator Ready
A36	TRDY#	Target Ready	B36	+3.3V06	+3.3 VDC
A37	GND15	Ground	B37	DEVSEL	Device Select
A38	STOP	Stop Transfer Cycle	B38	GND16	Ground
A39	+3.3V07	+3.3 VDC	B39	LOCK#	Lock bus
A40	-----	Reserved	B40	PERR#	Parity Error
A41	-----	Reserved	B41	+3.3V08	+3.3 VDC
A42	GND17	Ground	B42	SERR#	System Error
A43	PAR	Parity	B43	+3.3V09	+3.3 VDC
A44	AD15	Address/Data 15	B44	C/BE1	Command, Byte Enable 1
A45	+3.3V10	+3.3 VDC	B45	AD14	Address/Data 14
A46	AD13	Address/Data 13	B46	GND18	Ground
A47	AD11	Address/Data 11	B47	AD12	Address/Data 12
A48	GND19	Ground	B48	AD10	Address/Data 10
A49	AD9	Address/Data 9	B49	GND20	Ground
A50	Keyway	Open or Ground for 3.3V PWB	B50	Keyway	Open or Ground for 3.3V PWB
A51	Keyway	Open or Ground for 3.3V PWB	B51	Keyway	Open or Ground for 3.3V PWB
A52	C/BE0	Command, Byte Enable 0	B52	AD8	Address/Data 8
A53	+3.3V11	+3.3 VDC	B53	AD7	Address/Data 7
A54	AD6	Address/Data 6	B54	+3.3V12	+3.3 VDC
A55	AD4	Address/Data 4	B55	AD5	Address/Data 5
A56	GND21	Ground	B56	AD3	Address/Data 3
A57	AD2	Address/Data 2	B57	GND22	Ground
A58	AD0	Address/Data 0	B58	AD1	Address/Data 1
A59	+5V	Power (+5 V or +3.3 V)	B59	VCC08	Power (+5 V or +3.3 V)
A60	REQ64	Request 64 bit	B60	ACK64	Acknowledge 64 bit
A61	VCC11	+5 VDC	B61	VCC10	+5 VDC
A62	VCC13	+5 VDC	B62	VCC12	+5 VDC

PCI 64-bit Signals

Pin #	Name	PCI Pin Description	Pin #	Name	PCI Pin Description
A63	GND	Ground	B63	RES	Reserved
A64	C/BE[7]#	Command, Byte Enable 7	B64	GND	Ground
A65	C/BE[5]#	Command, Byte Enable 5	B65	C/BE[6]#	Command, Byte Enable 6
A66	+5V	Power (+5 V or +3.3 V)	B66	C/BE[4]#	Command, Byte Enable 4
A67	PAR64	Parity 64	B67	GND	Ground
A68	AD62	Address/Data 62	B68	AD63	Address/Data 63
A69	GND	Ground	B69	AD61	Address/Data 61
A70	AD60	Address/Data 60	B70	+5V	Power (+5 V or +3.3 V)
A71	AD58	Address/Data 58	B71	AD59	Address/Data 59
A72	GND	Ground	B72	AD57	Address/Data 57
A73	AD56	Address/Data 56	B73	GND	Ground
A74	AD54	Address/Data 54	B74	AD55	Address/Data 55
A75	+5V	Power (+5 V or +3.3 V)	B75	AD53	Address/Data 53
A76	AD52	Address/Data 52	B76	GND	Ground
A77	AD50	Address/Data 50	B77	AD51	Address/Data 51
A78	GND	Ground	B78	AD49	Address/Data 49
A79	AD48	Address/Data 48	B79	+5V	Power (+5 V or +3.3 V)
A80	AD46	Address/Data 46	B80	AD47	Address/Data 47
A81	GND	Ground	B81	AD45	Address/Data 45
A82	AD44	Address/Data 44	B82	GND	Ground
A83	AD42	Address/Data 42	B83	AD43	Address/Data 43
A84	+5V	Power (+5 V or +3.3 V)	B84	AD41	Address/Data 41
A85	AD40	Address/Data 40	B85	GND	Ground
A86	AD38	Address/Data 38	B86	AD39	Address/Data 39
A87	GND	Ground	B87	AD37	Address/Data 37
A88	AD36	Address/Data 36	B88	+5V	Power (+5 V or +3.3 V)
A89	AD34	Address/Data 34	B89	AD35	Address/Data 35
A90	GND	Ground	B90	AD33	Address/Data 33
A91	AD32	Address/Data 32	B91	GND	Ground
A92	RES	Reserved	B92	RES	Reserved
A93	GND	Ground	B93	RES	Reserved
A94	RES	Reserved	B94	GND	Ground

PCI - Architecture



Note: *During initial configuration transactions only, the IDSEL is used to indicate to a PCI endpoint (or bridge) that it is currently selected.*

PCI terms and definitions I.

- Two devices participate in each bus transaction:
  - **Initiator** (starts the transaction) × **Target** (obeys request)
  - **Initiator** = Bus Master,
  - **Target** = Slave for current transaction.
- Initiator and target role does not directly impose data source and receiver role!

PCI terms and definitions II.

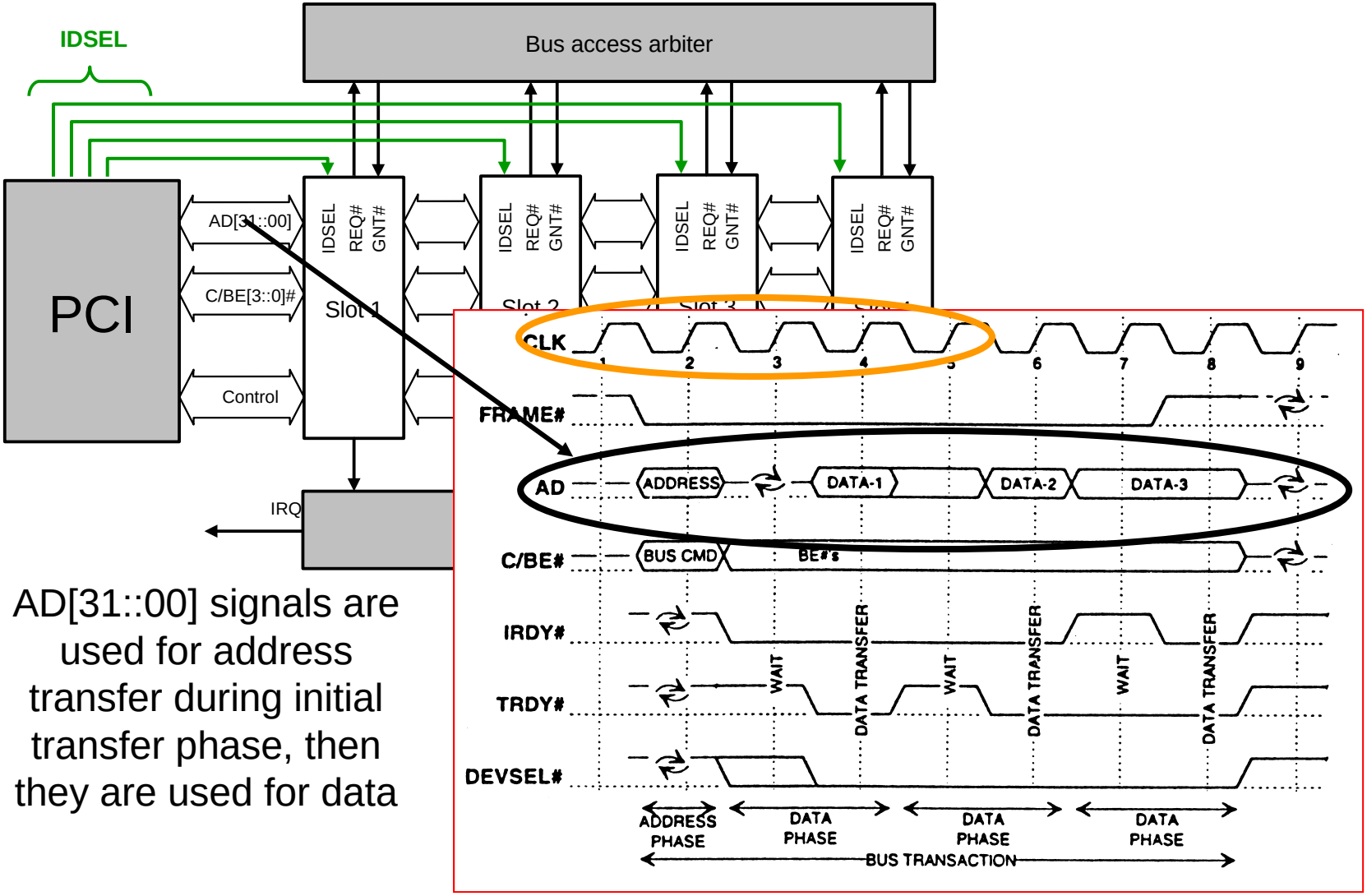
- What does it mean that bus is multimaster?
  - More participants can act as Bus Master – initiate transfers!
- When bus signals are shared, then transactions need to be serialized and only one device can act as master at any given time instant
  - Bus master is responsible to grant bus to requesting participant
- Who takes care of bus arbitration?
  - One dedicated device or bus backplane

Note: A decentralized (distributed) bus arbitration also exists, i.e., all devices participate in the selection of the next bus master, but not for PCI.

PCI terms and definitions III.

- The rising clock edge is reference/trigger point for all phase of the bus cycle/transaction timing
- Bus cycle is formed in most cases by
  - **address phase**
  - **data phase**
- Premature termination of data transfer is possible during bus cycle
- Transfer synchronization itself is pseudo-synchronous

Reduce Bus/Signal Lines Number – Share/Multiplexing

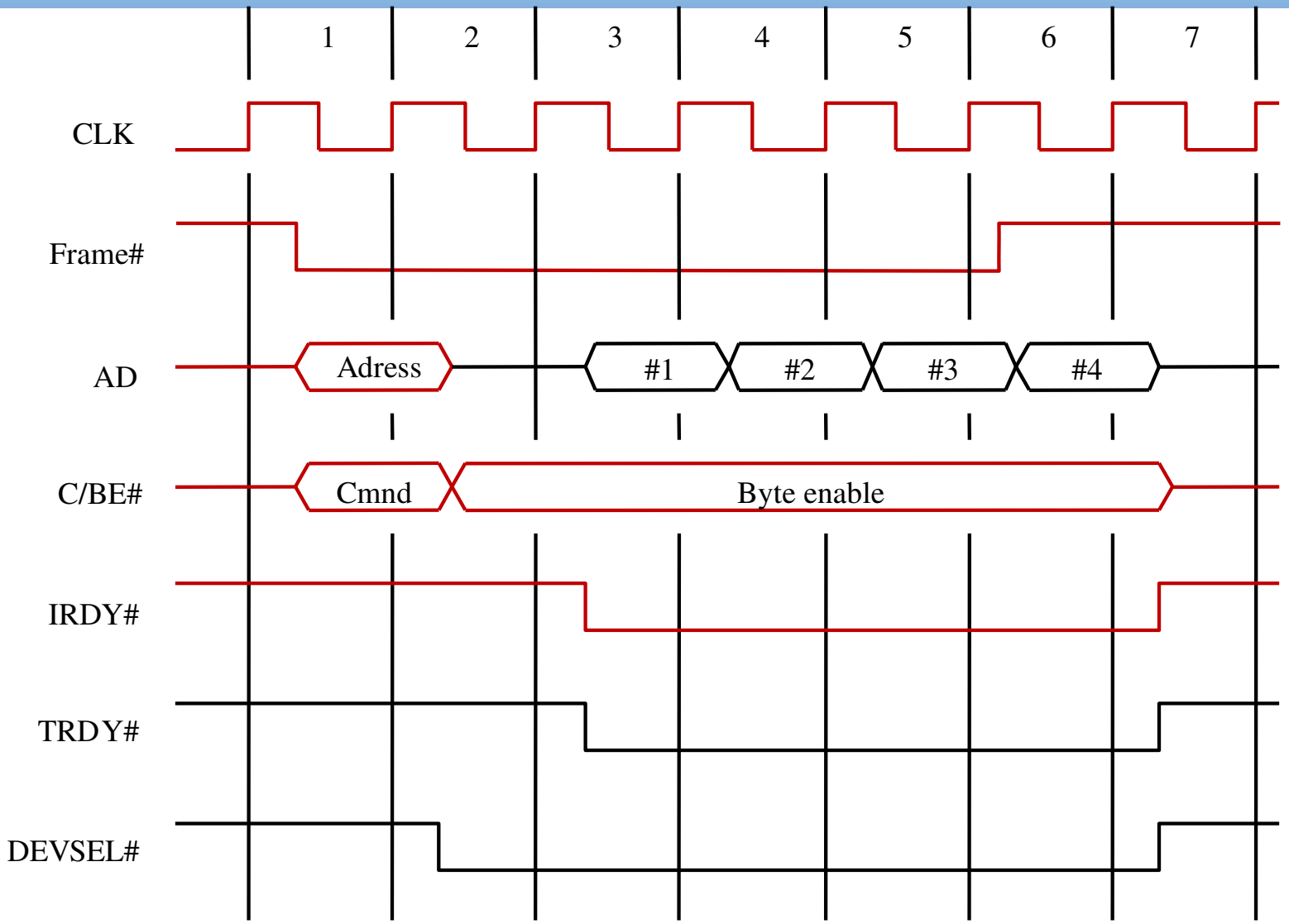


AD[31::00] signals are used for address transfer during initial transfer phase, then they are used for data

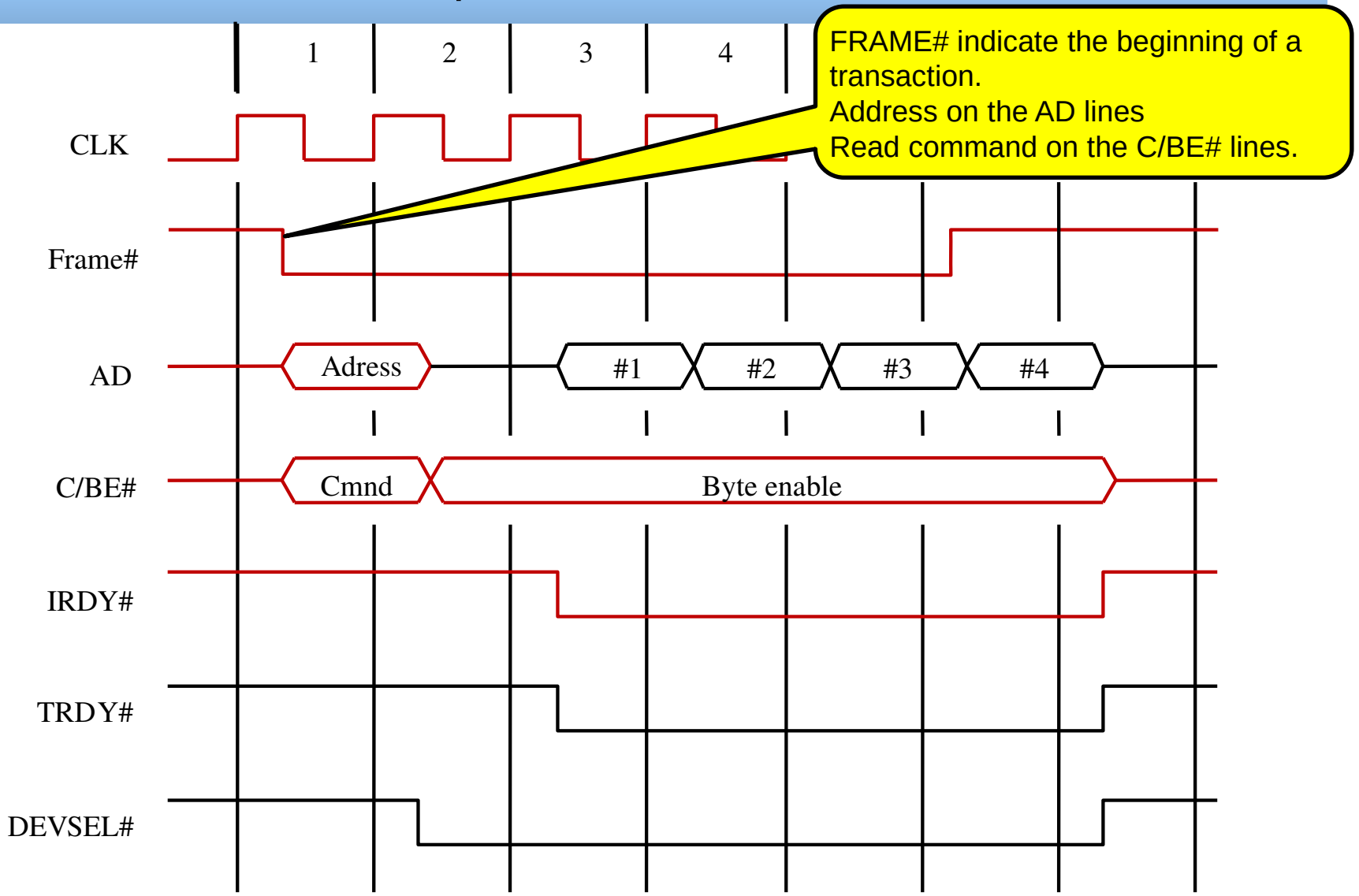
Bus Cycle Kind/Direction – Command – Specified by C/BE

C/BE[0::3]#	Bus command (BUS CMD)
0000	Interrupt Acknowledge
0001	Special Cycle
0010	I/O Read
0011	I/O Write
0100	Reserved
0101	Reserved
0110	Memory Read
0111	Memory Write
1000	Reserved
1001	Reserved
1010	Configuration Read (only 11 low addr bits for fnc and reg + IDSEL)
1011	Configuration Write (only 11 low addr bits for fnc and reg + IDSEL)
1100	Memory Read Multiple
1101	Dual Address Cycle (more than 32 bits for address – i.e. 64-bit)
1110	Memory Read Line
1111	Memory Write and Invalidate

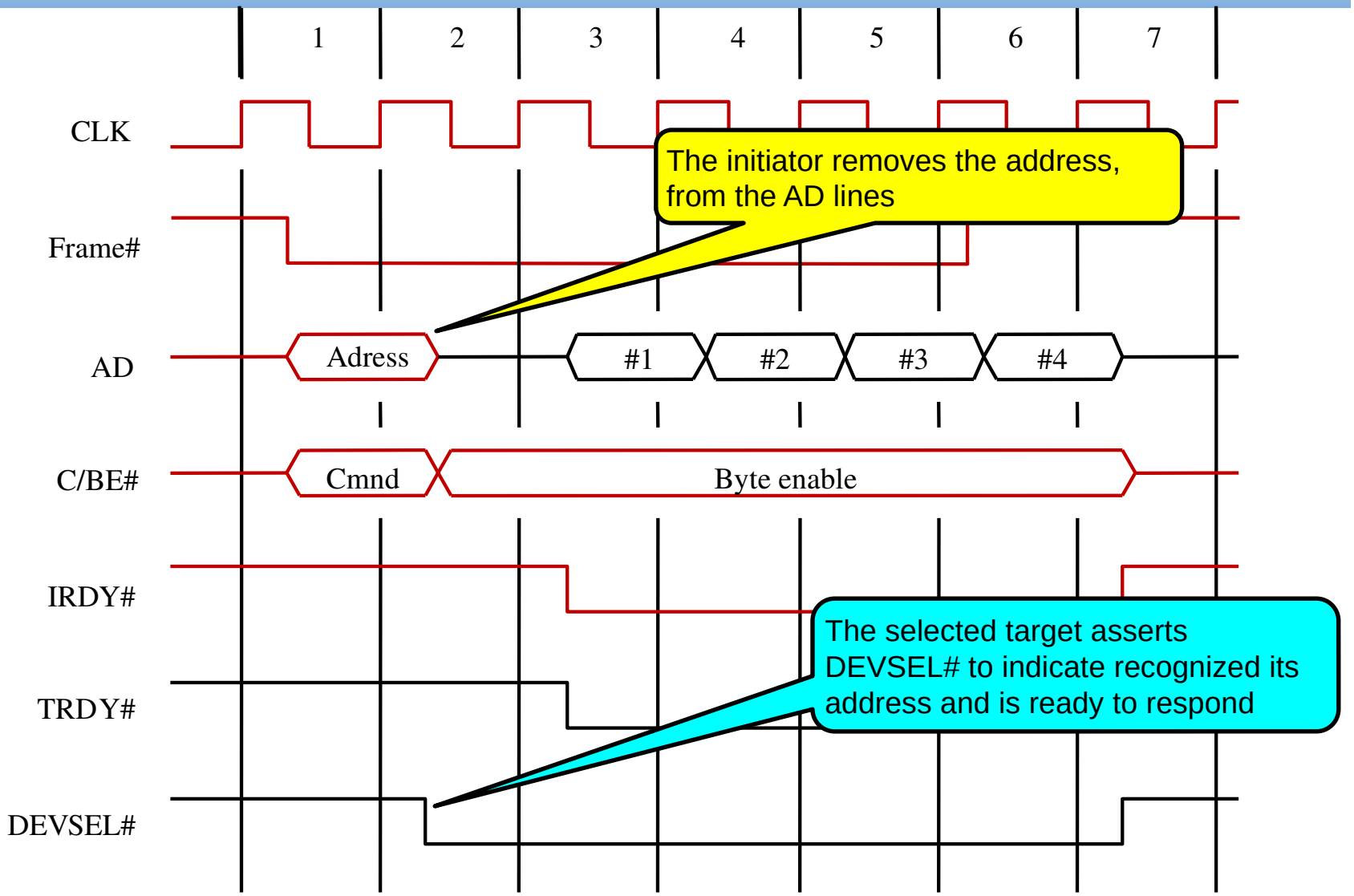
A Read Operation on the PCI Bus



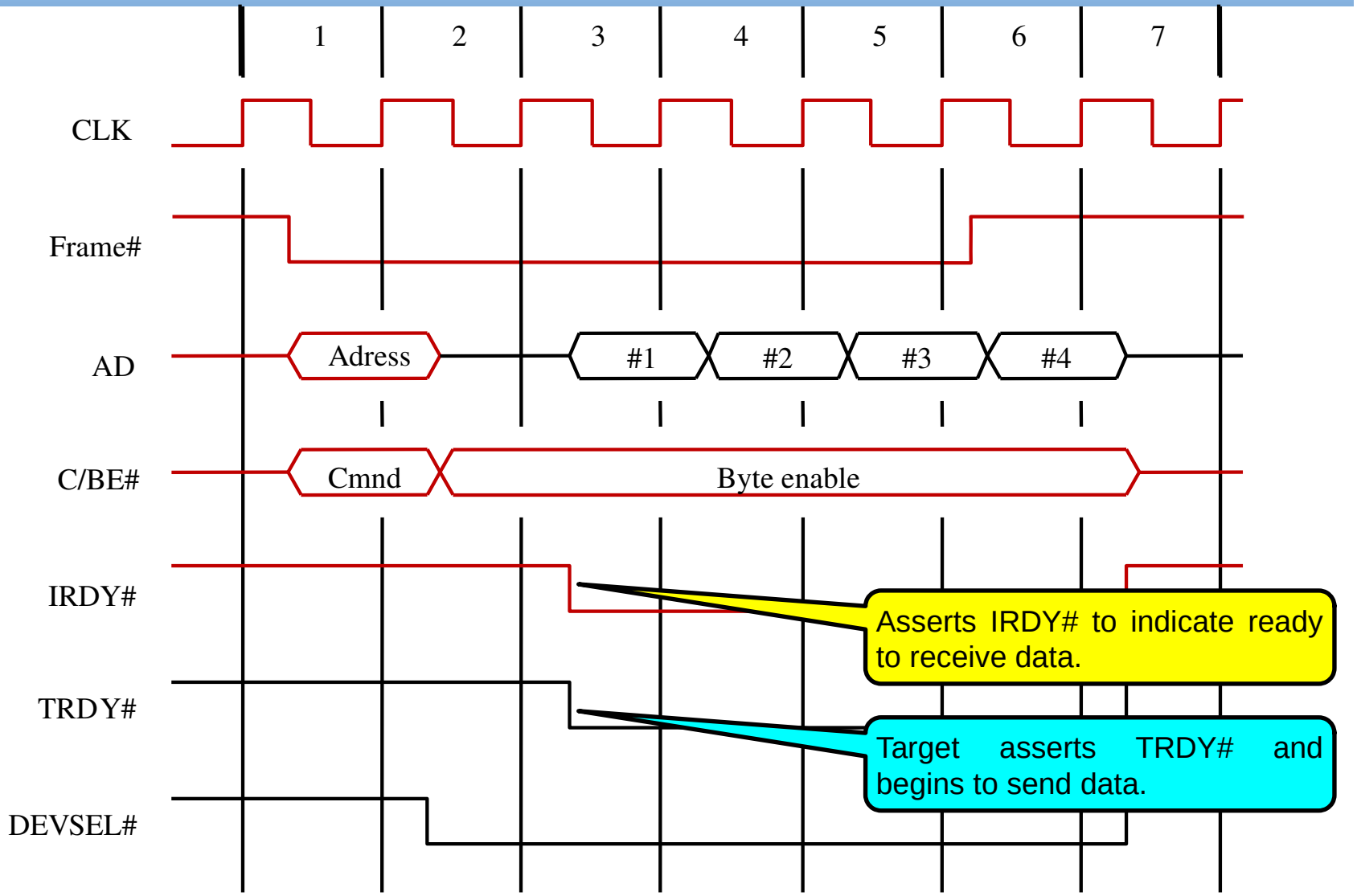
A Read Operation on the PCI Bus



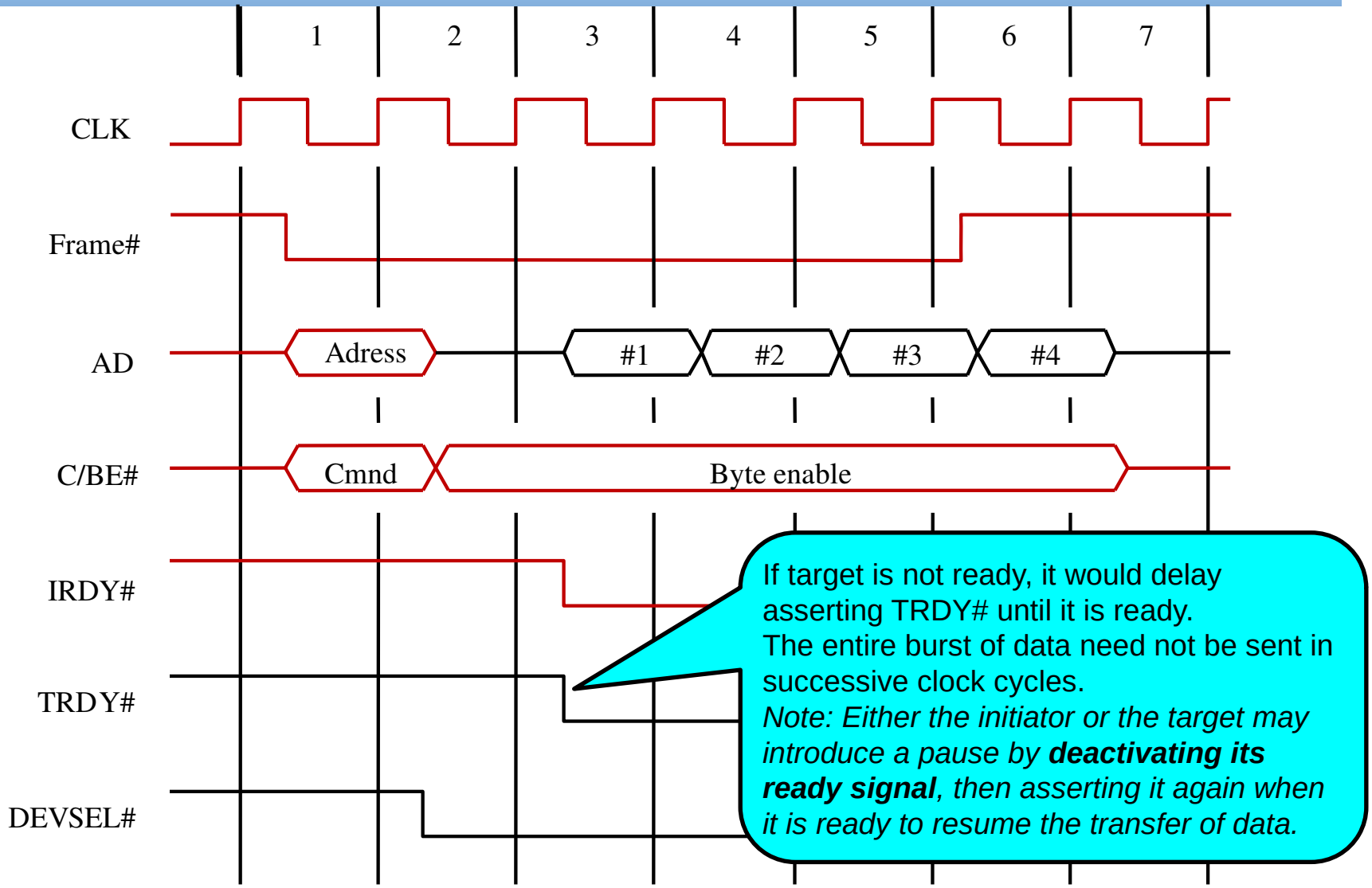
A Read Operation on the PCI Bus



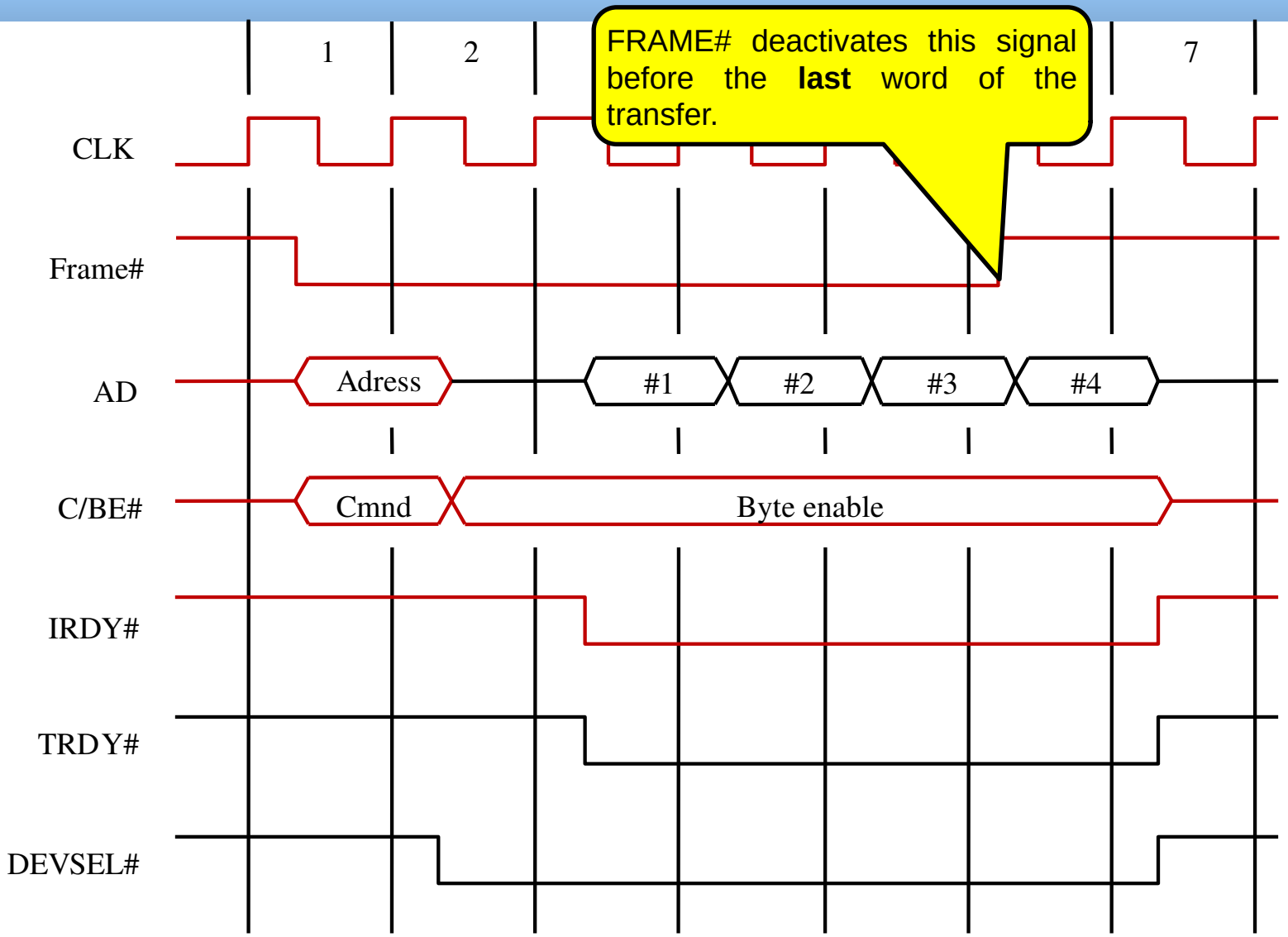
A Read Operation on the PCI Bus



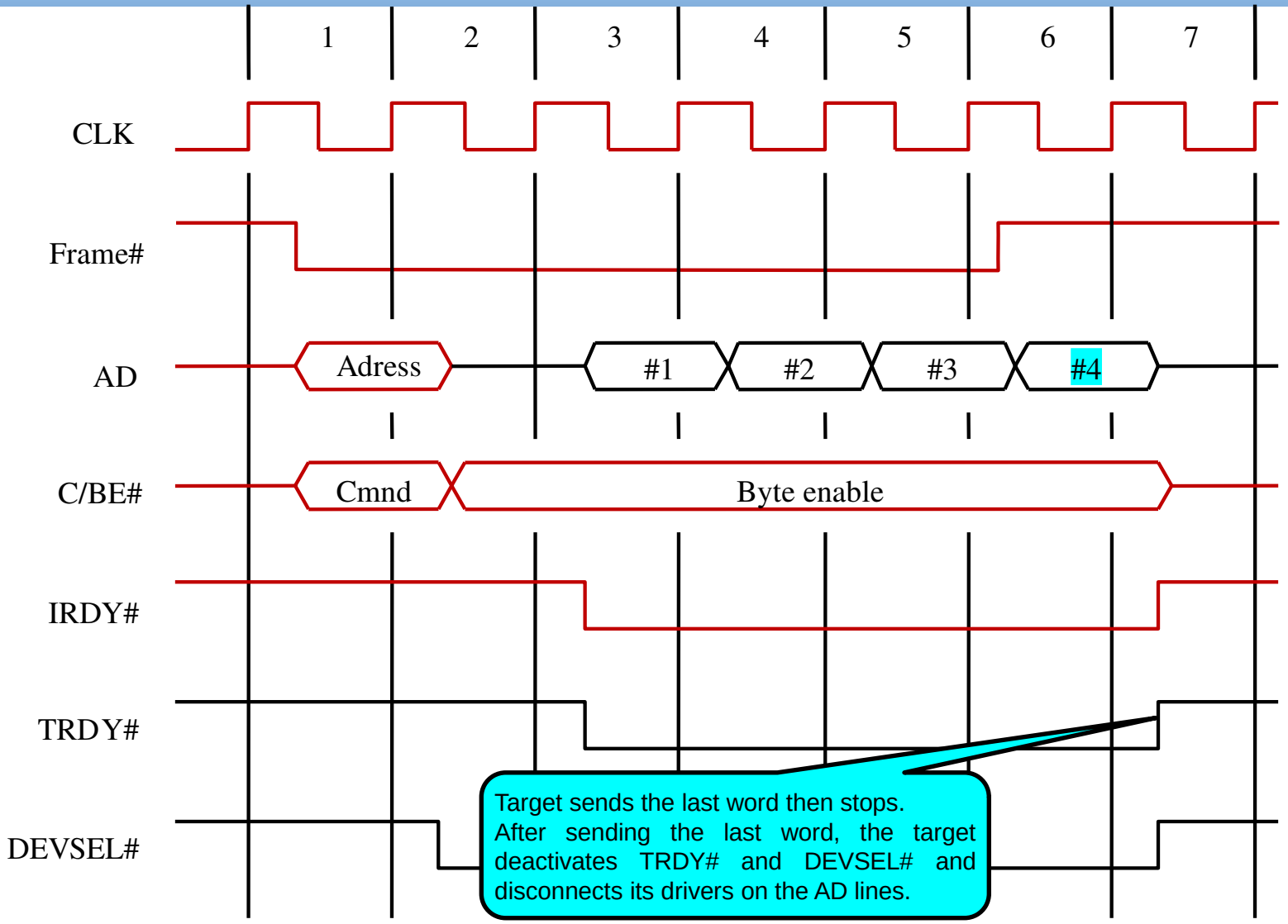
A Read Operation on the PCI Bus



A Read Operation on the PCI Bus



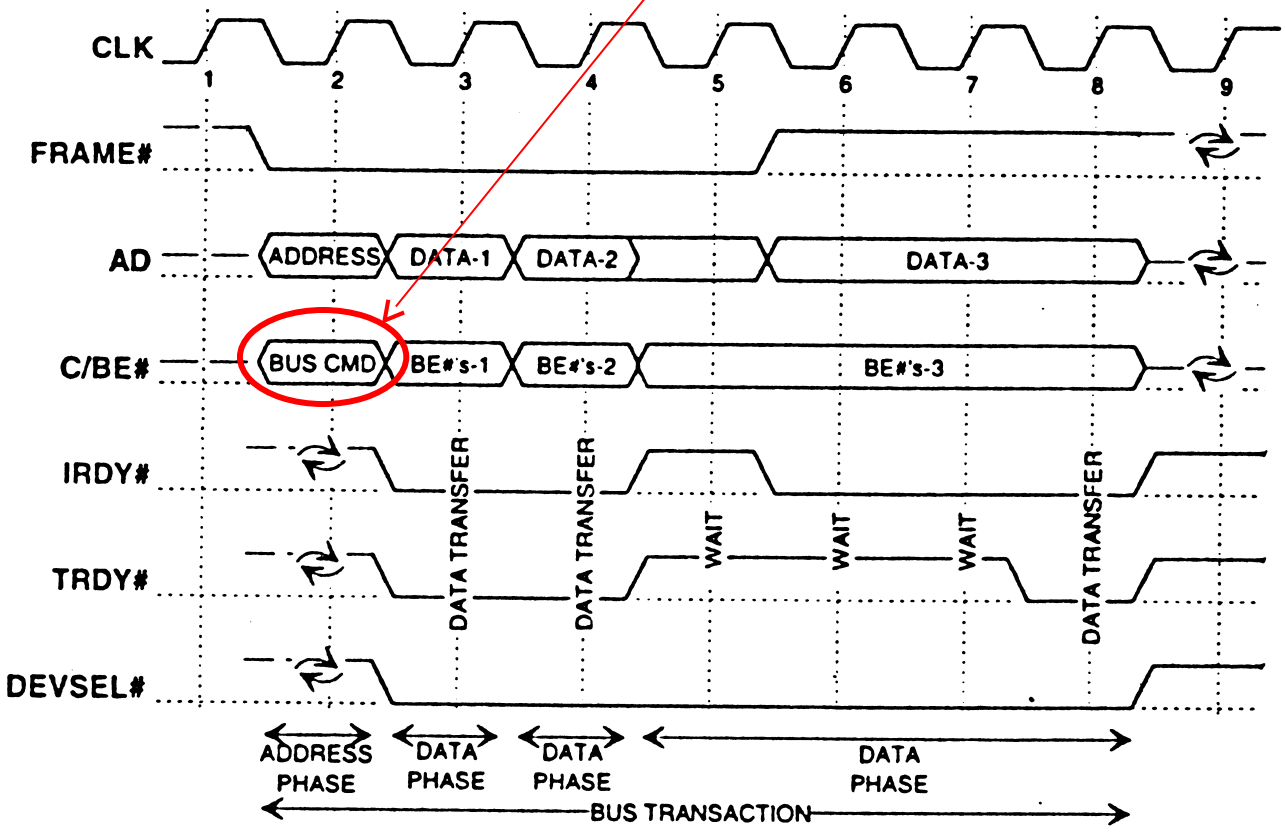
A Read Operation on the PCI Bus



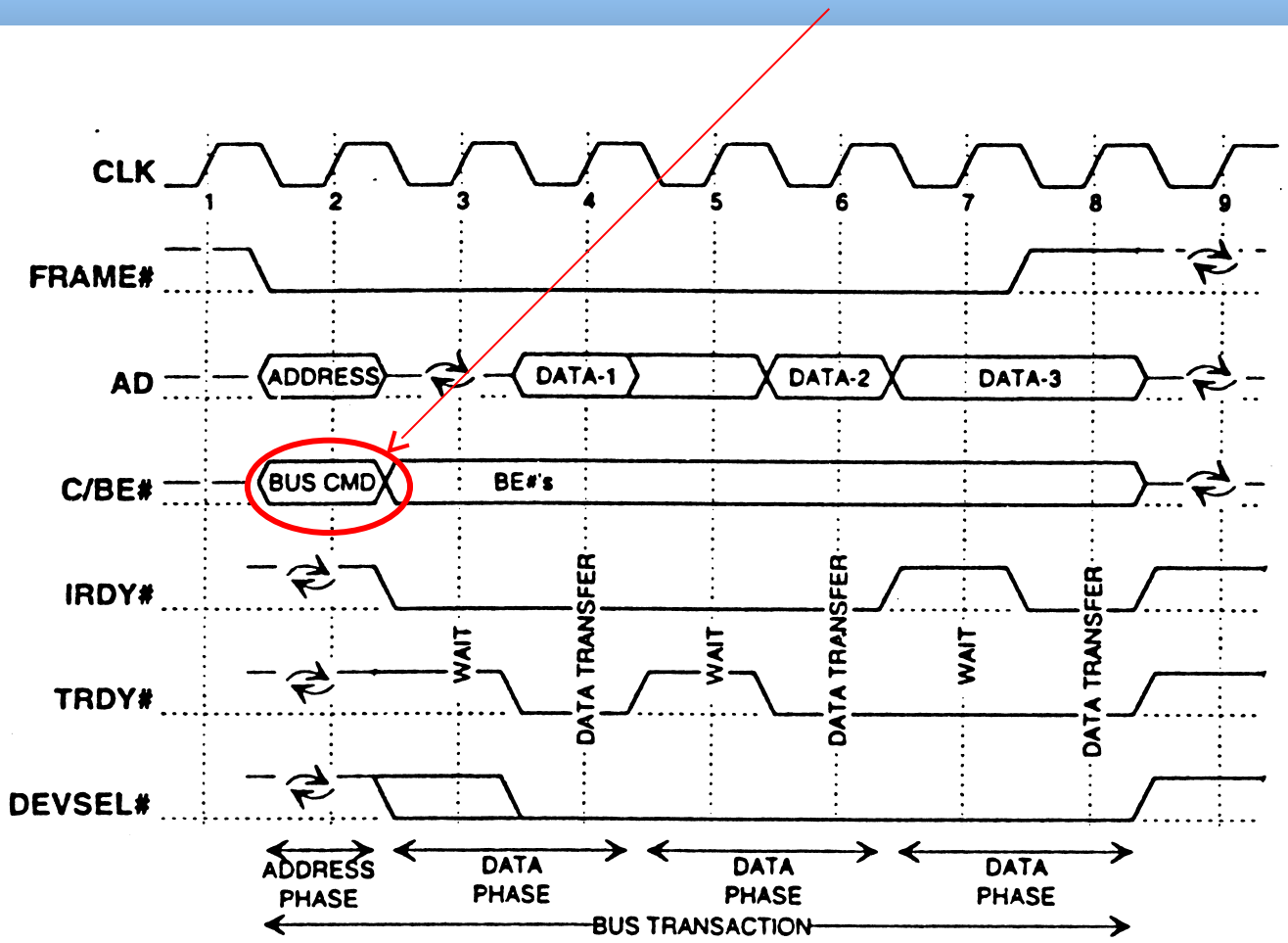
Some Remarks and Observations

- The length of the transferred data block is controlled by the FRAME signal. It is negated (de-asserted) before last word transfer by initiator.
- Single world or **burst** transfer can be delayed by inserting wait cycles (pseudo-synchronous synchronization)!

PCI Bus Memory **Write** Timing



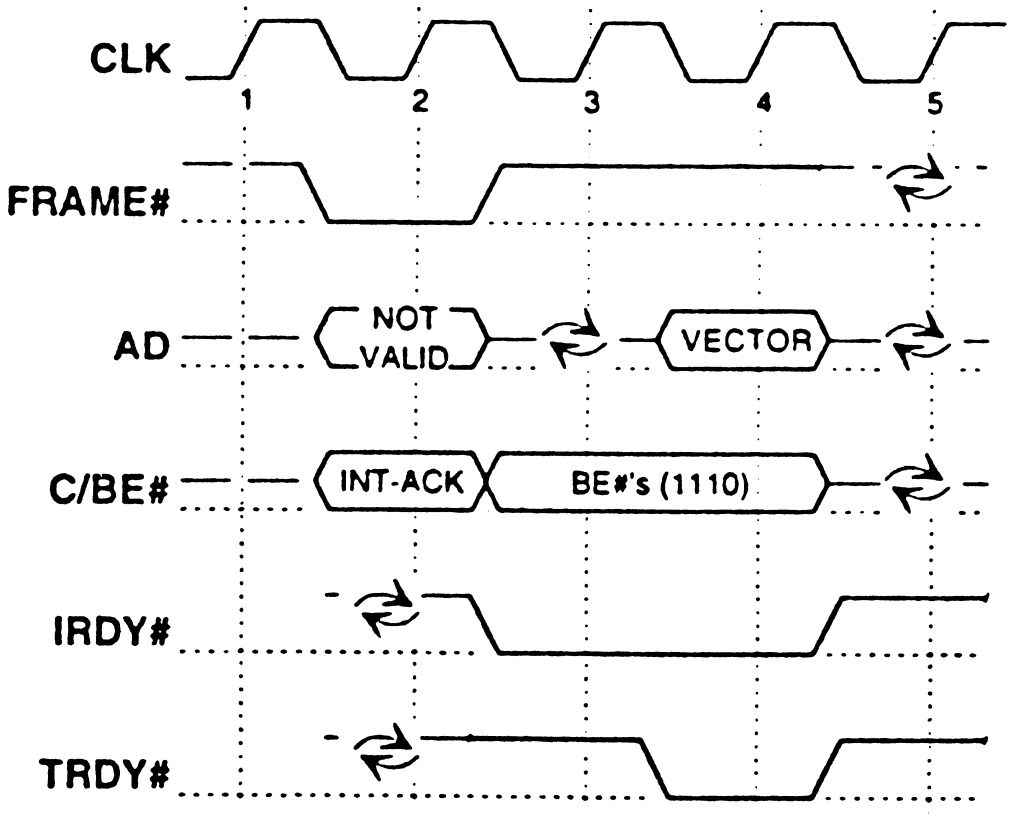
PCI Bus Memory **Read** Timing



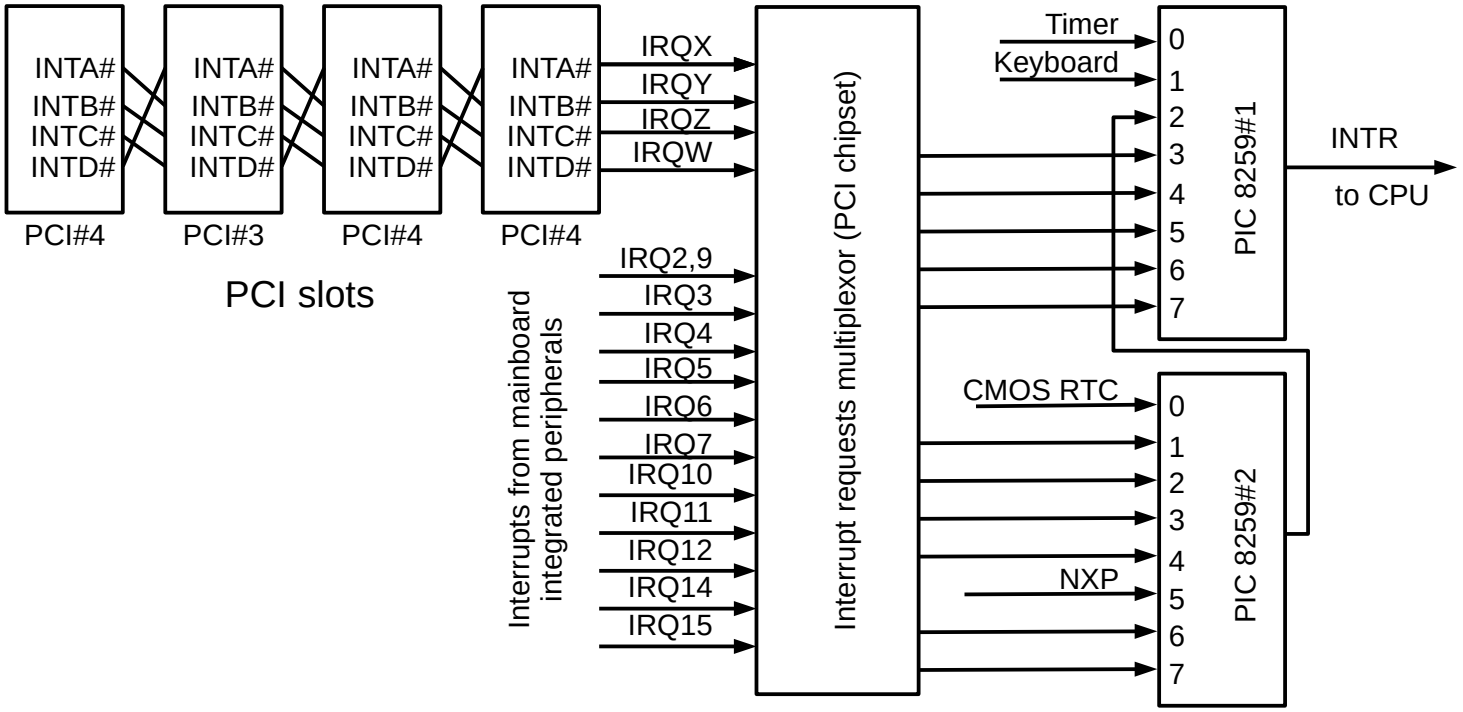
Interrupt Acknowledge Cycle

- Processor needs time to save interrupted execution state to allow state restoration after return from service routine
- Interrupt controller provides vector number (assigned to the asynchronous event) and CPU is required to translate it to the interrupt service routine start address
- All these activities require some time

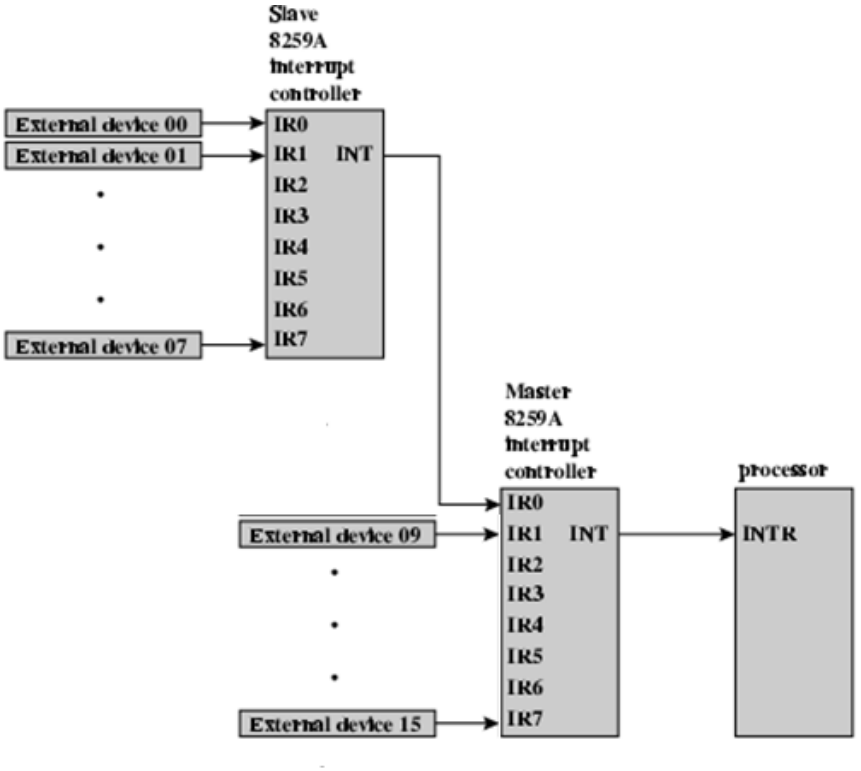
Interrupt Acknowledge Cycle Timing



Some More Details About Standard PC PIC IRQ Routing



Standard PC PIC Interrupt Vectors Assignment



Common interrupt numbers for PC category computers:

IRQ	
0	Timer (for scheduler, timers)
1	Keyboard
2	i8259 cascade interrupt
8	Real-time clocks (CMOS wall time)
9	Available or SCSI controller
10,11	Available
12	Available or PS/2 mouse
13	Available or arithmetics co-processor
14	1-st IDE controller
15	2-nd IDE controller
3	COM2
4	COM1
5	LPT2 or available
6	Floppy disc controller
7	LPT1

Message Signaled Interrupt

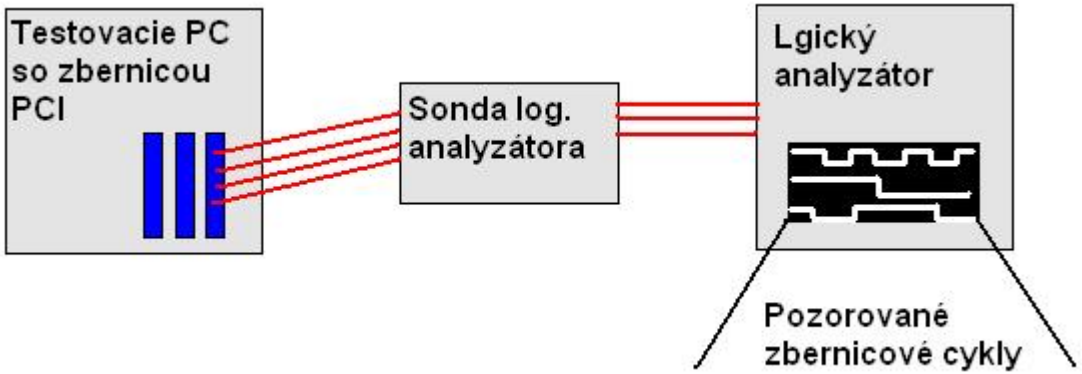
- Memory address space write by device to a special address → interrupt to CPU
- MSI from PCI 2.2 (single interrupt per device)
- MSI-X from PCI 3.0 (up to 2048 can be allocated)
- Reasons
  - Pin-based PCI interrupts often shared
  - Pin-based can arrive before data reach memory
  - PCI devices supports only single pin-based per function.

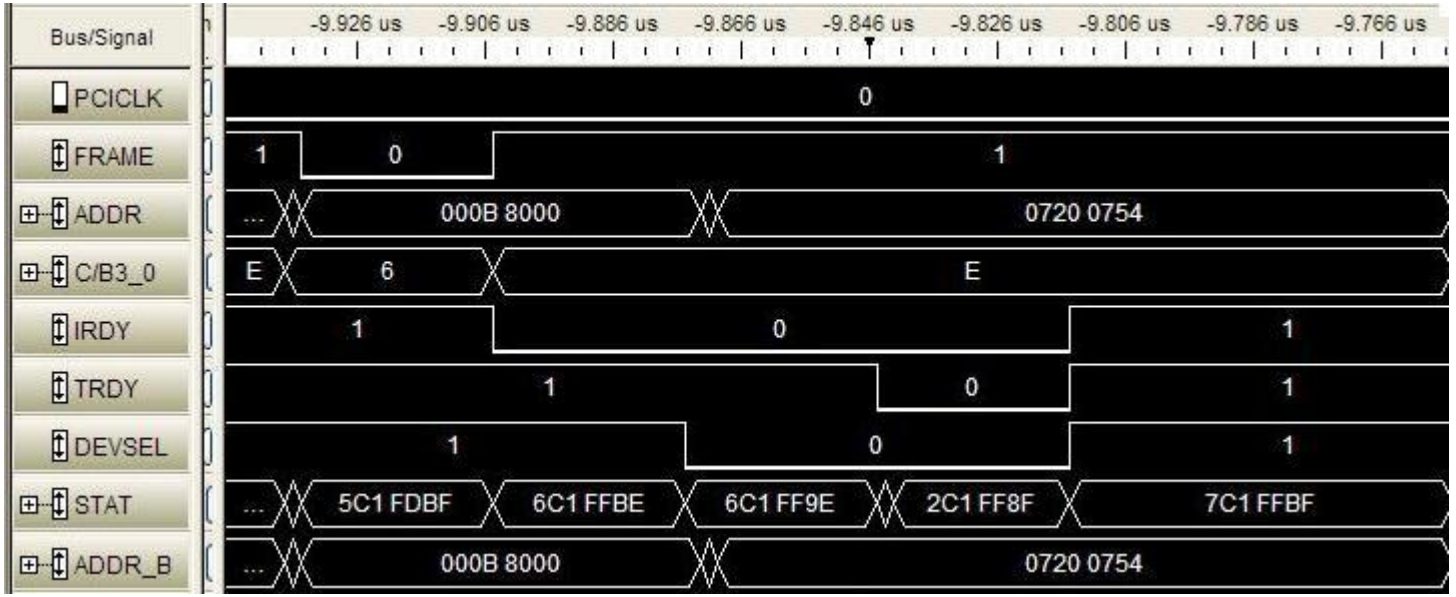
Recapitulation of the Bus Description Steps

- Notice: we started PCI description by bus topology analysis, PCI signals and then we moved to timing diagrams
  - simpler cases first (read and write)
  - then how bus access is arbitrated
  - the special functions (interrupt acknowledge) last

- Doc. Šnorek recommends: always follow these steps when trying to learn new bus technology.

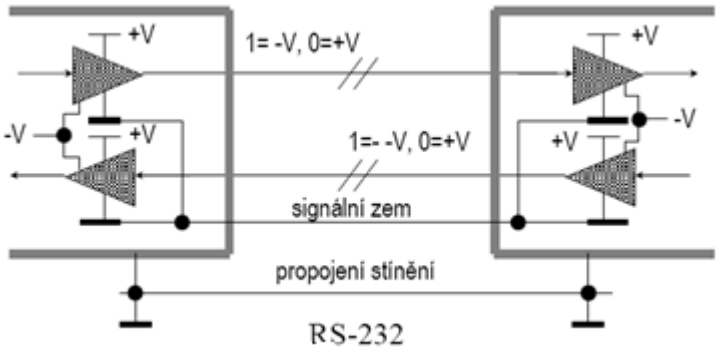
PCI Bus Timing Laboratory Exercise





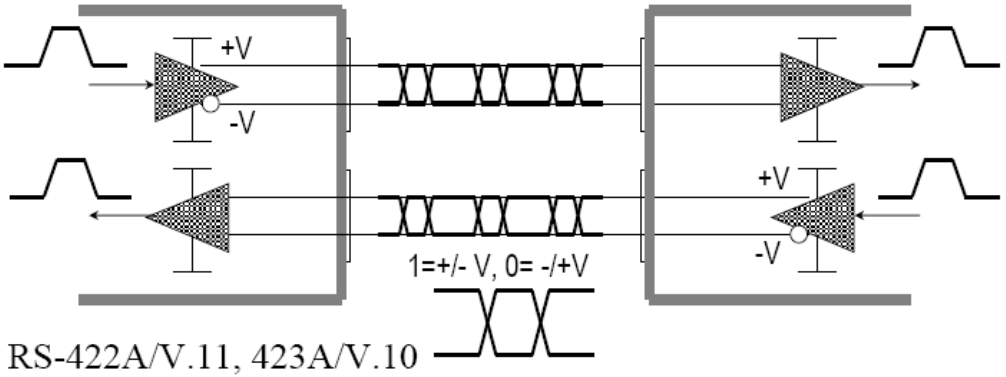
Some more notes regarding the bus hardware realization

How Signals are Transferred



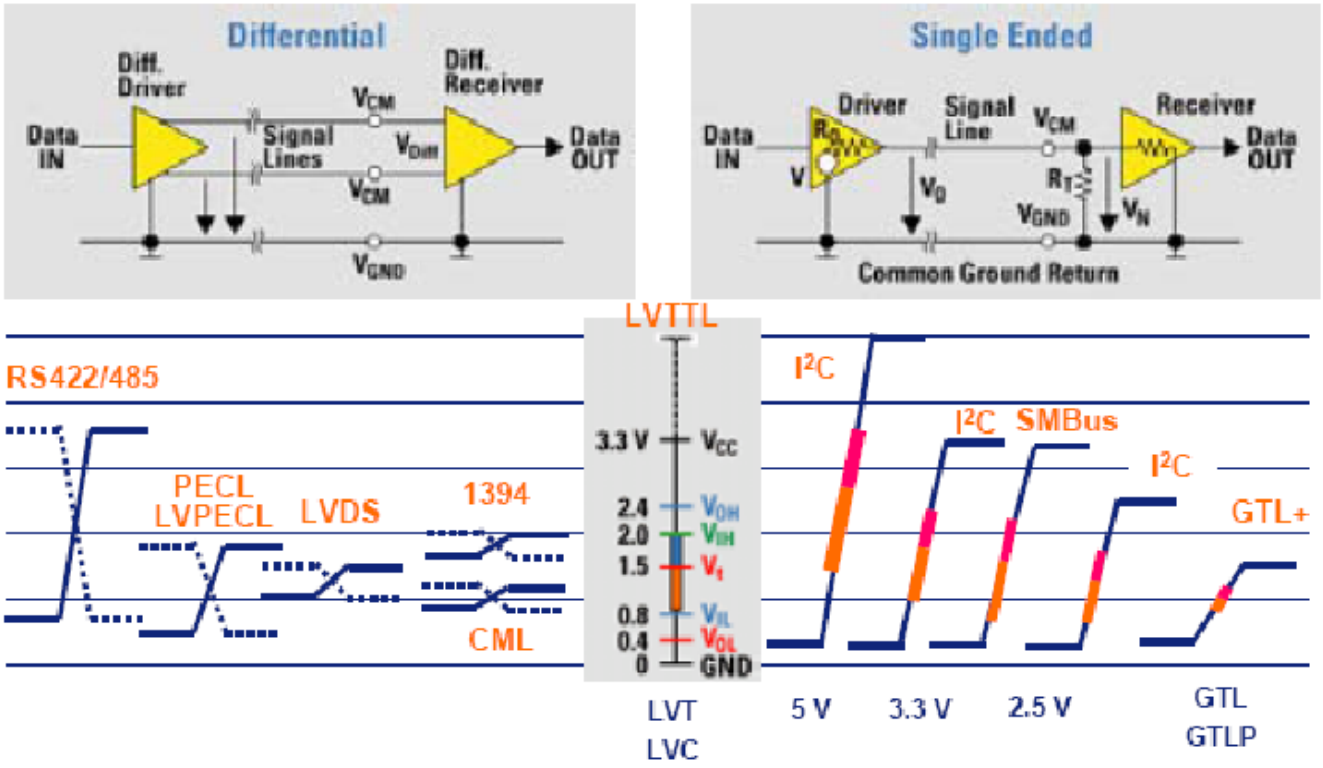
English term
Signalling

Single ended (asymmetric) versus.
differential (symmetric)



Typical Signaling Levels and Some Speed Considerations

DesignCon 2003 TecForum I²C Bus Overview

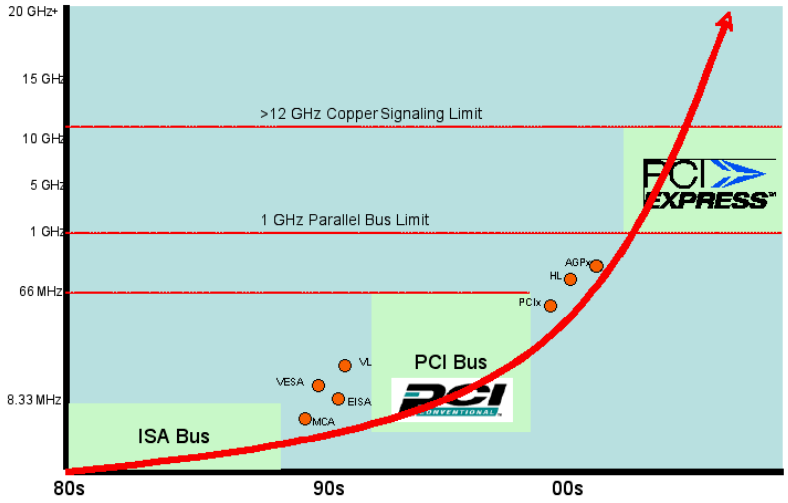


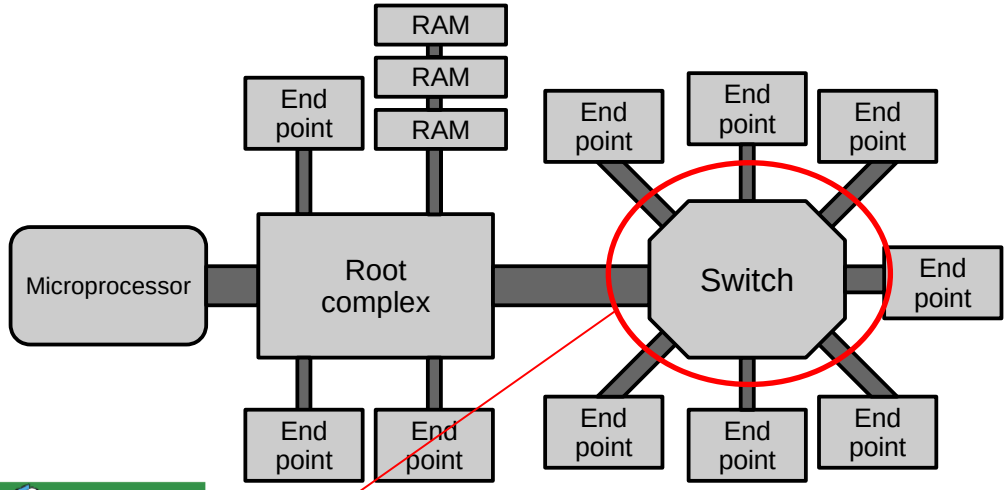
Differential signaling

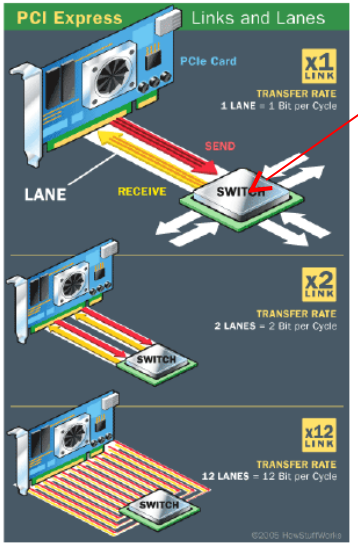
Single ended signaling

PCIe – Peripheral Component Interconnect Express

PCIe Architecture

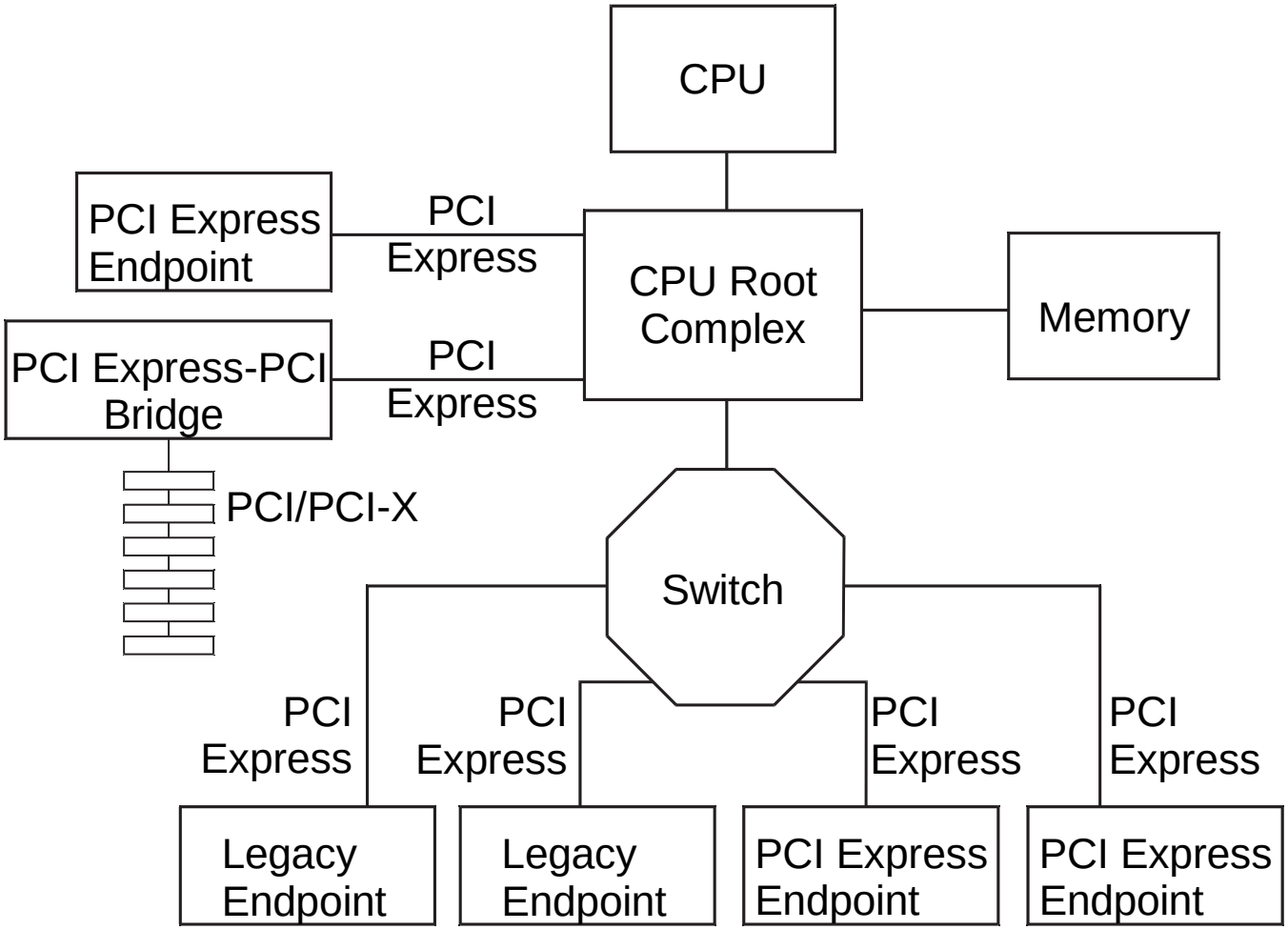




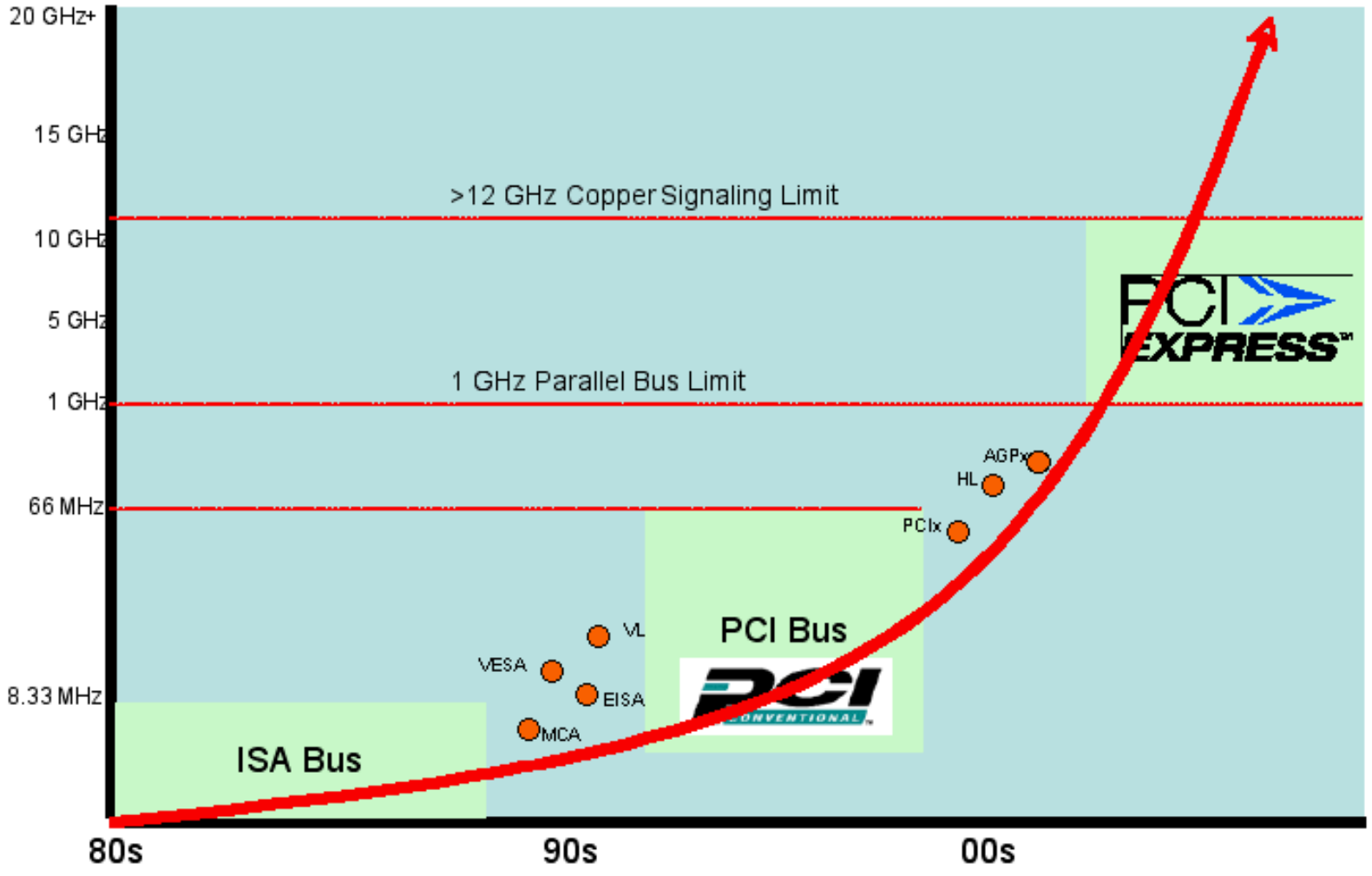


Source:
http://computer.howstuffworks.com/pci-express.htm

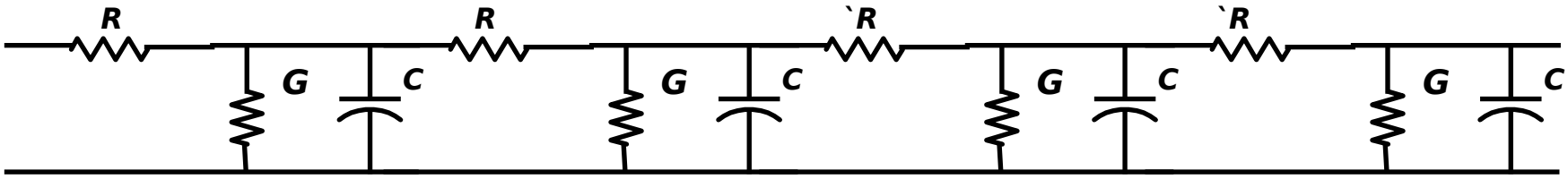
PCIe Topology and Components



Why Switch to Serial Data Transfer



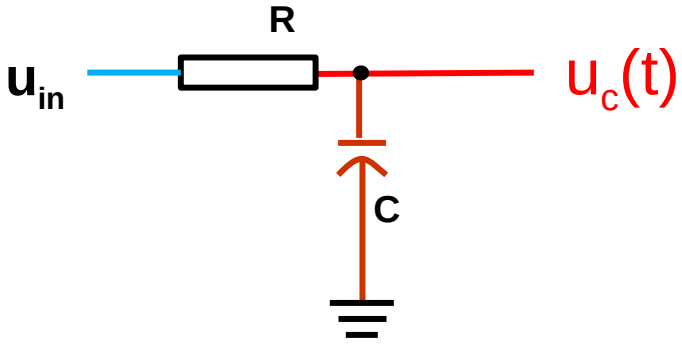
RC Delay in Wire

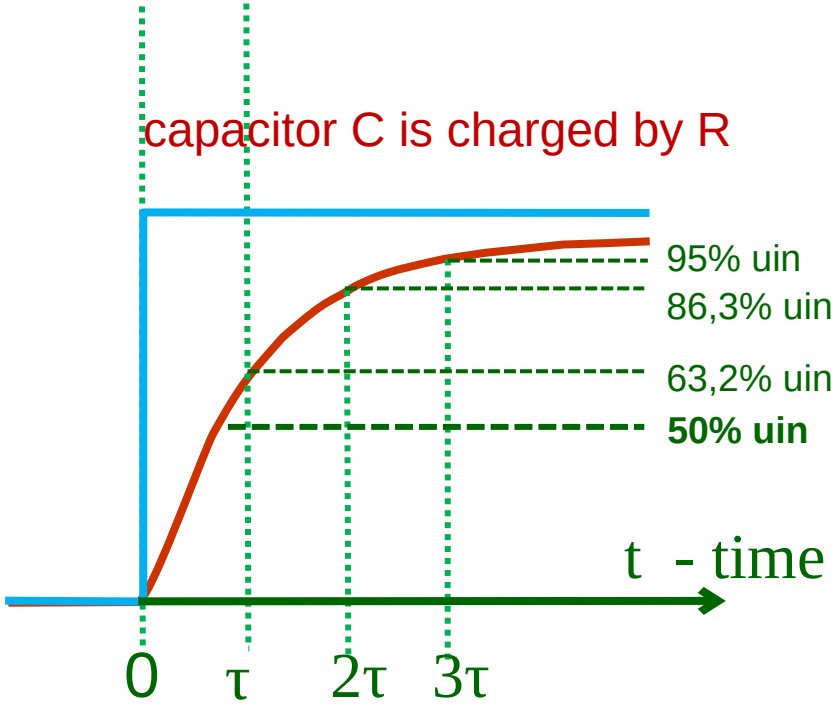


Wire: The length of one wire element → 0 and the number of elements → ∞

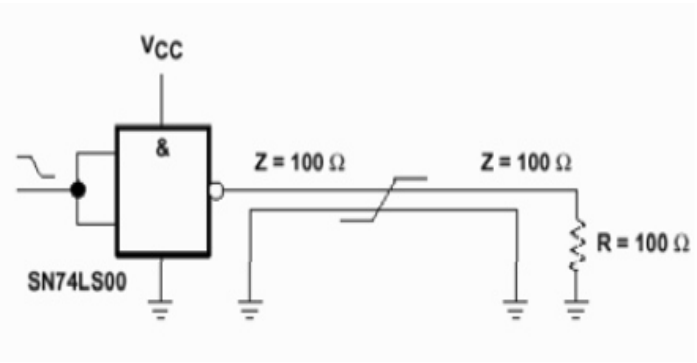
$u_c(t)=u_{in}(1-e^{-t/\tau})$, $\tau=R.C$

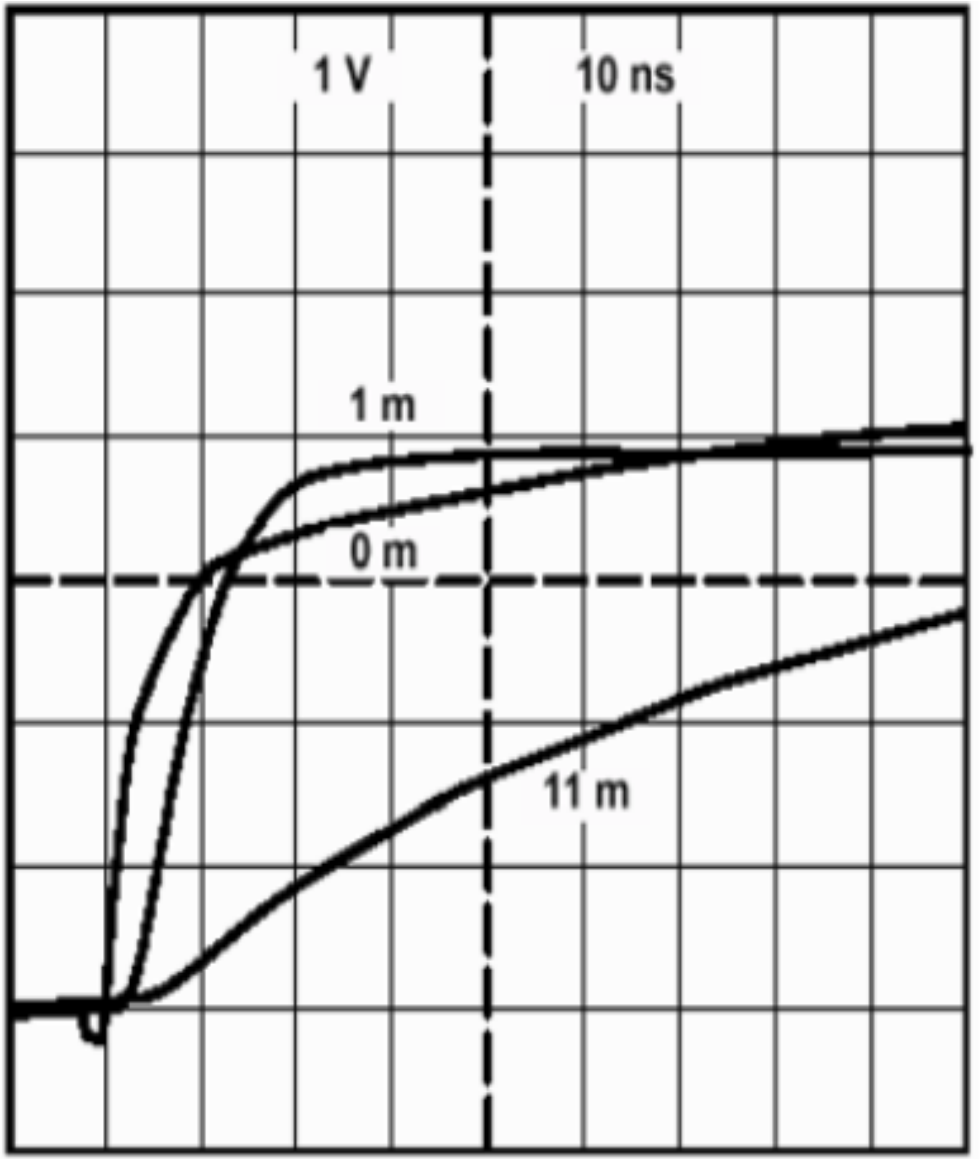
50 % = 0.69 RC





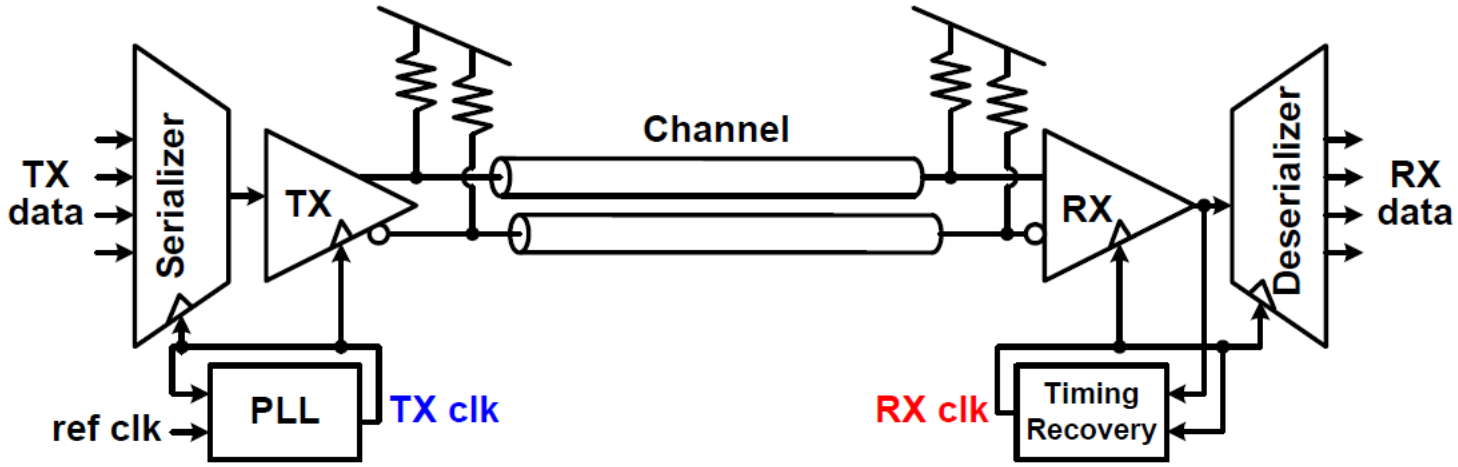
Deformation of Signals



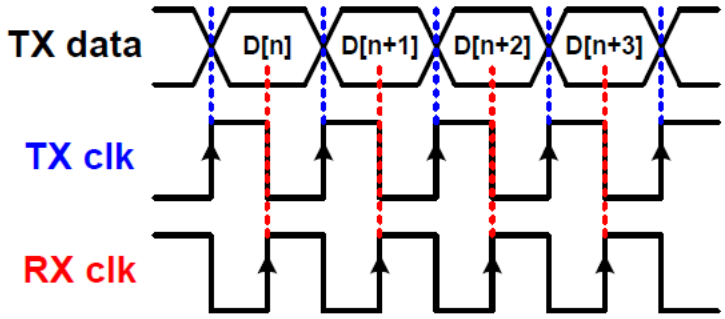


Source:

High Speed Serial Link



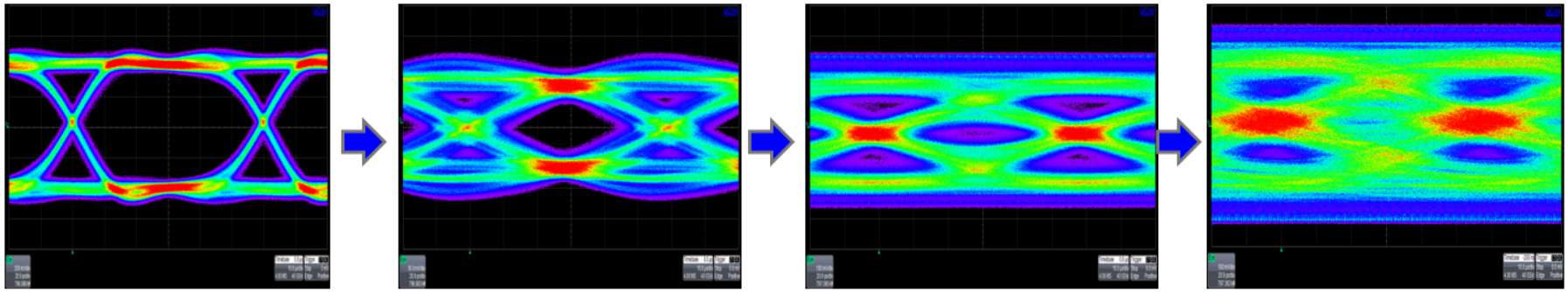
PLL
= Phase-locked Loop
(cz: fázový závěs)
generates frequency with phase related to input reference signal ref clk.



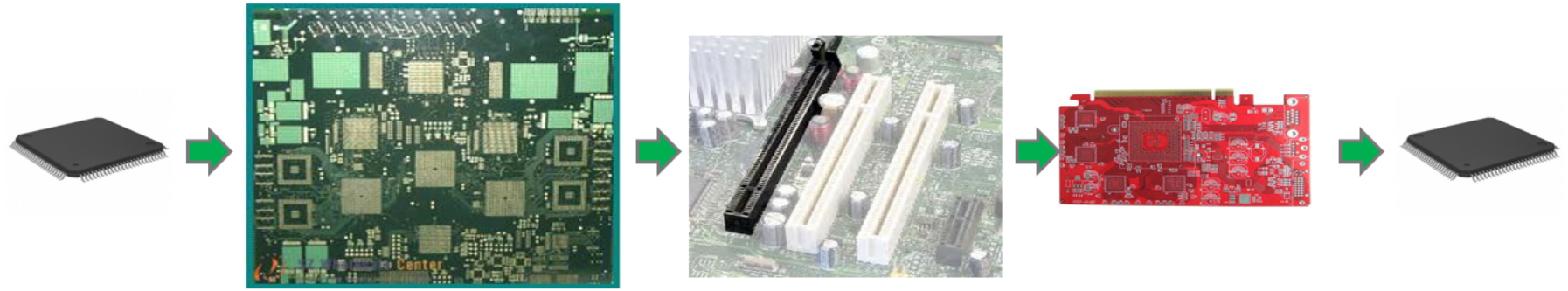
Only ideal theoretical signals:-)

Source: S. Palermo: **High-Speed Serial I/O Design for Channel-Limited and Power-Constrained Systems ,** Texas A&M University 2010

High Speed Serial Link Reality



Signal degrades over long transmission path and connectors

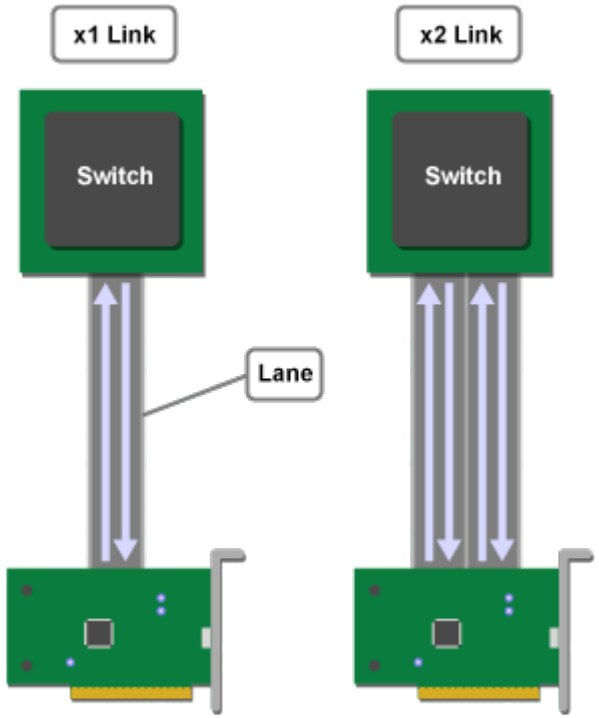


IC System Board Connector Plug-In Card IC

Source: TELEDYNE LECROY, 2018

PCIe Transfers Signaling, PCIe Lanes

- Link interconnect switch with exactly one device
- Differential AC signaling is used – two wires for single direction
- Each link consists of one or more lanes
- Lane consists of two pair of wires
- One pair for Tx and other one for Rx
- Data are serialized by 8/10 code
- The separate pairs allow full-duplex operation/transfers



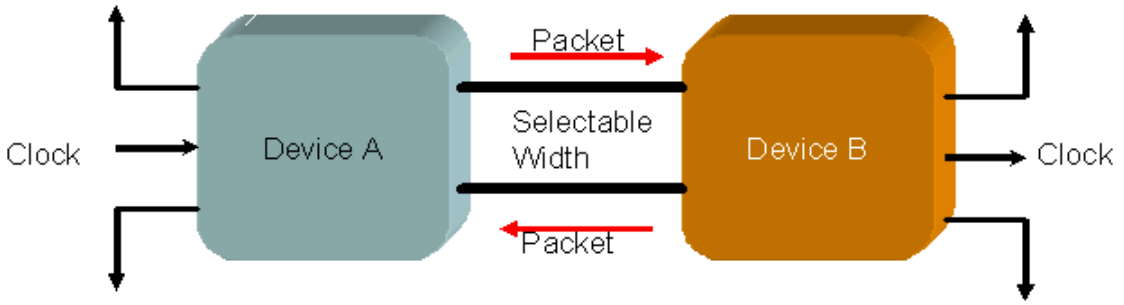
Up to x16 lanes scalable

PCIe Slot Signals

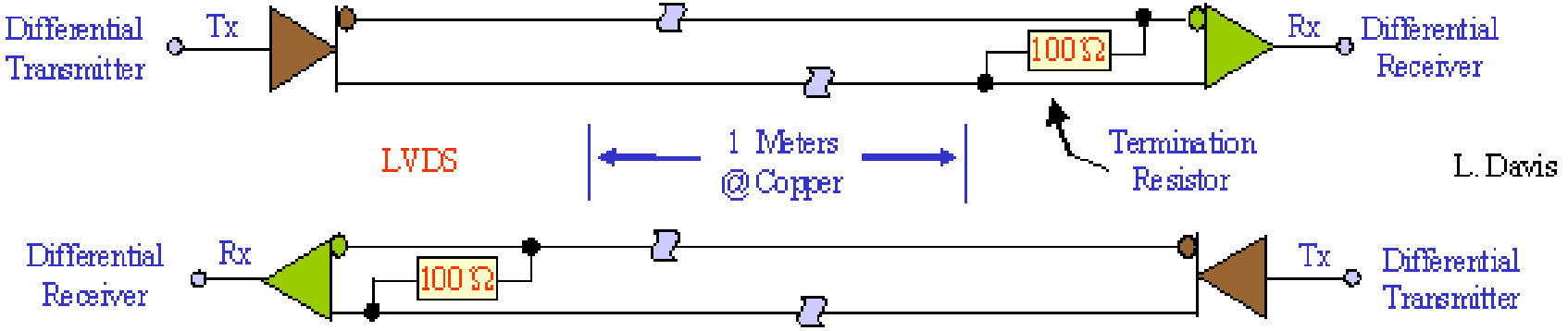
Pin	Side B Connector		Side A Connector	
#	Name	Description	Name	Description
1	+12v	+12 volt power	PRSNT#1	Hot plug presence detect
2	+12v	+12 volt power	+12v	+12 volt power
3	RSVD	Reserved	+12v	+12 volt power
4	GND	Ground	GND	Ground
5	SMCLK	SMBus clock	JTAG2	TCK
6	SMDAT	SMBus data	JTAG3	TDI
7	GND	Ground	JTAG4	TDO
8	+3.3v	+3.3 volt power	JTAG5	TMS
9	JTAG1	+TRST#	+3.3v	+3.3 volt power
10	3.3Vaux	3.3v volt power	+3.3v	+3.3 volt power
11	WAKE#	Link Reactivation	PWRGD	Power Good
Mechanical Key				
12	RSVD	Reserved	GND	Ground
13	GND	Ground	REFCLK+	Reference Clock Differential pair
14	HSOp(0)	Transmitter Lane 0, Differential pair	REFCLK-	
15	HSOn(0)		GND	Ground
16	GND	Ground	HSIp(0)	Receiver Lane 0, Differential pair
17	PRSNT#2	Hotplug detect	HSIn(0)	
18	GND	Ground	GND	Ground

PCIe physical link layer

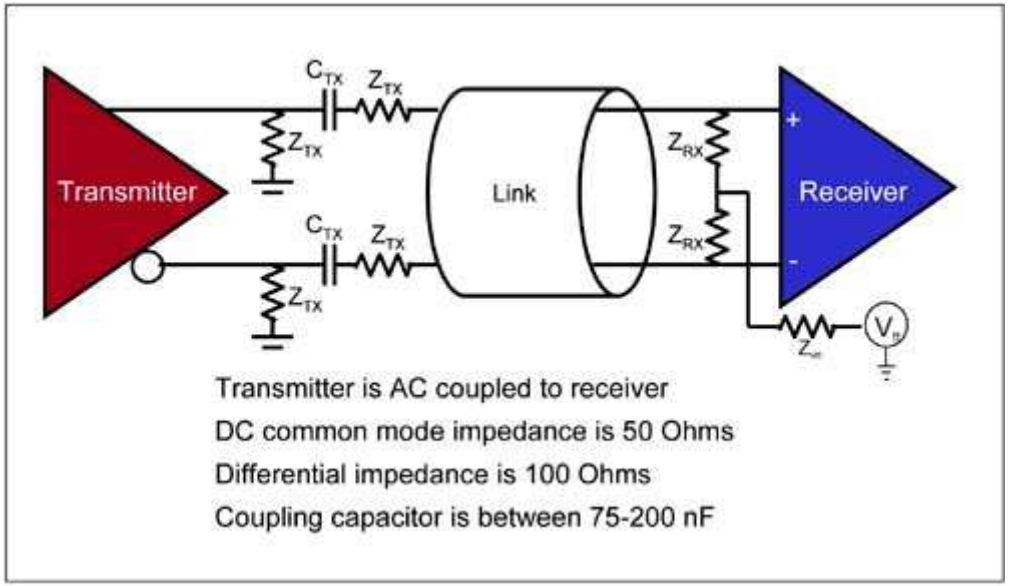
Differential full-duplex physical layer



8b/10b encoding provides enough edges for clock signal reconstruction/synchronization and balanced number of ones and zeros. This ensures zero common signal (DC) and AC (only) coupling is possible



PCIe Physical Layer Model

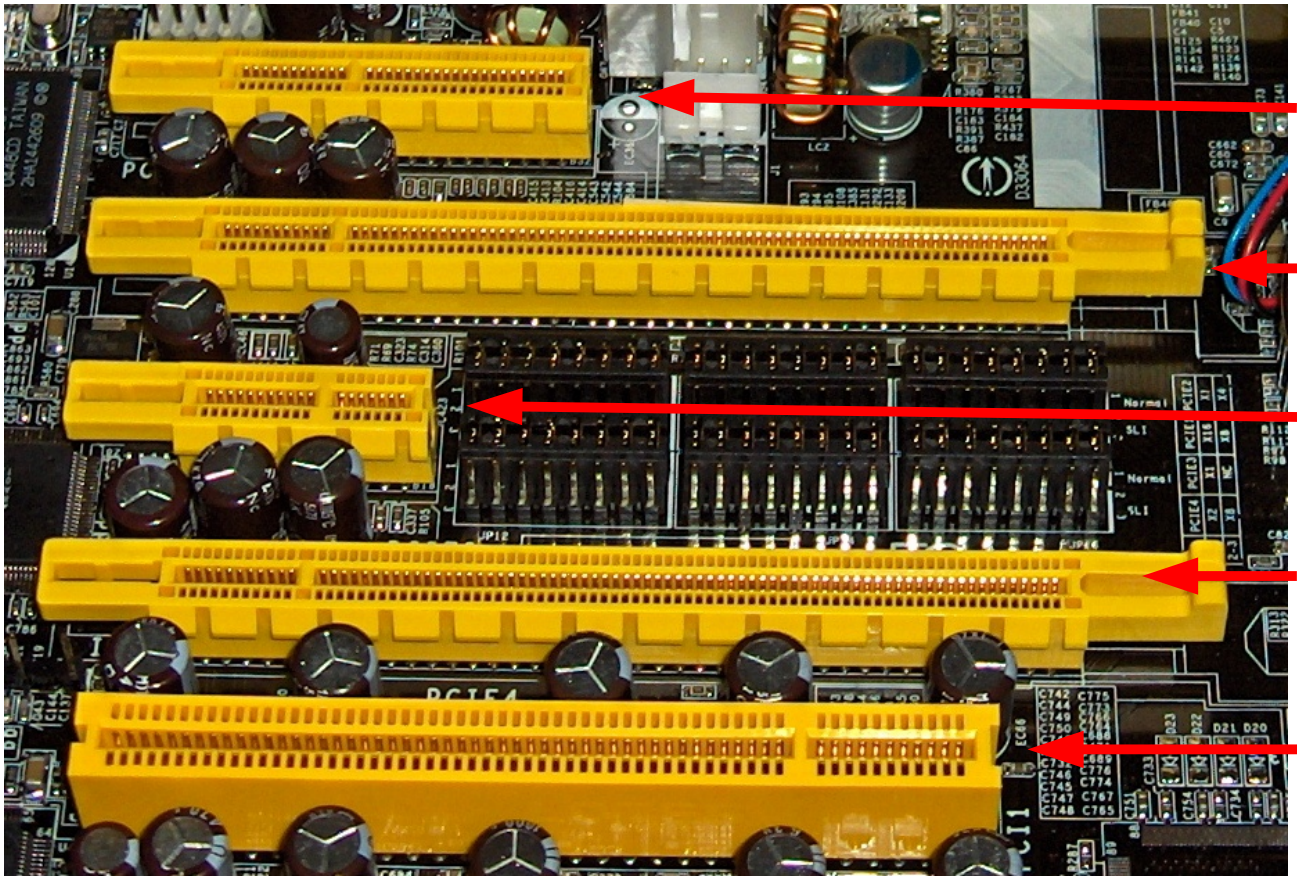


Source: Budruk, R., at all: PCI Express System Architecture

PCIe Characteristics

- 2.5 GHz clock frequency (1 GT/s), raw single link single direction bandwidth 250 MB/s (can be multiplied by parallel lanes – 2×, 4×, 8×)
- Single link efficient data rate is 200 MB/s, this is 2× … 4× more than for classic PCI
- The bandwidth is not shared, point to point interconnection
- Two pairs of wires, differential signaling
- Data are encoded (modulated) using 8b/10b code
- Expected up to 10 GHz clocks due technology advances
- PCI Express 2.x (2007) allows 5 GT/s (5 GHz clock)
- PCI Express 3.x (started at 2010) increases it to 8 GT/s
- Encoding changed from 8b/10b (20% bandwidth used by encoding) to "scrambling" and 128b/130b encoding (takes only 1.5% of the bandwidth)

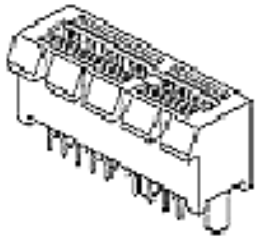
PCI and PCIe Slots on the PC Motherboard



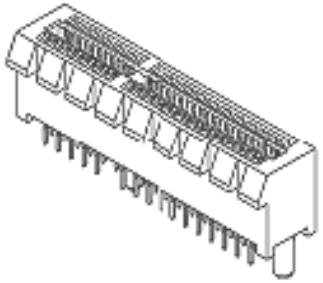
Picture source: Wikipedia

LanParty nF4 Ultra-D mainboard from DFI

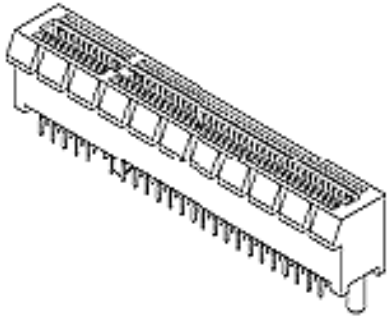
ExpressCard



PCIe 1×

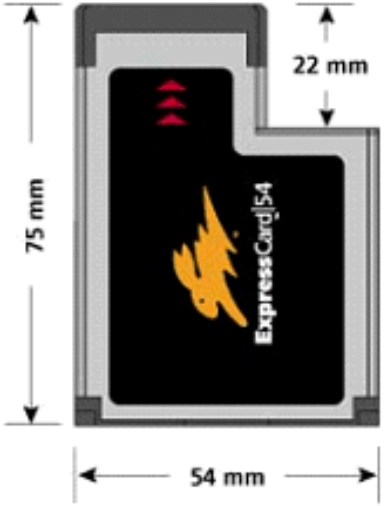


PCIe 4×



PCIe 8×

CardBus
PCMCI



ExpressCard/54

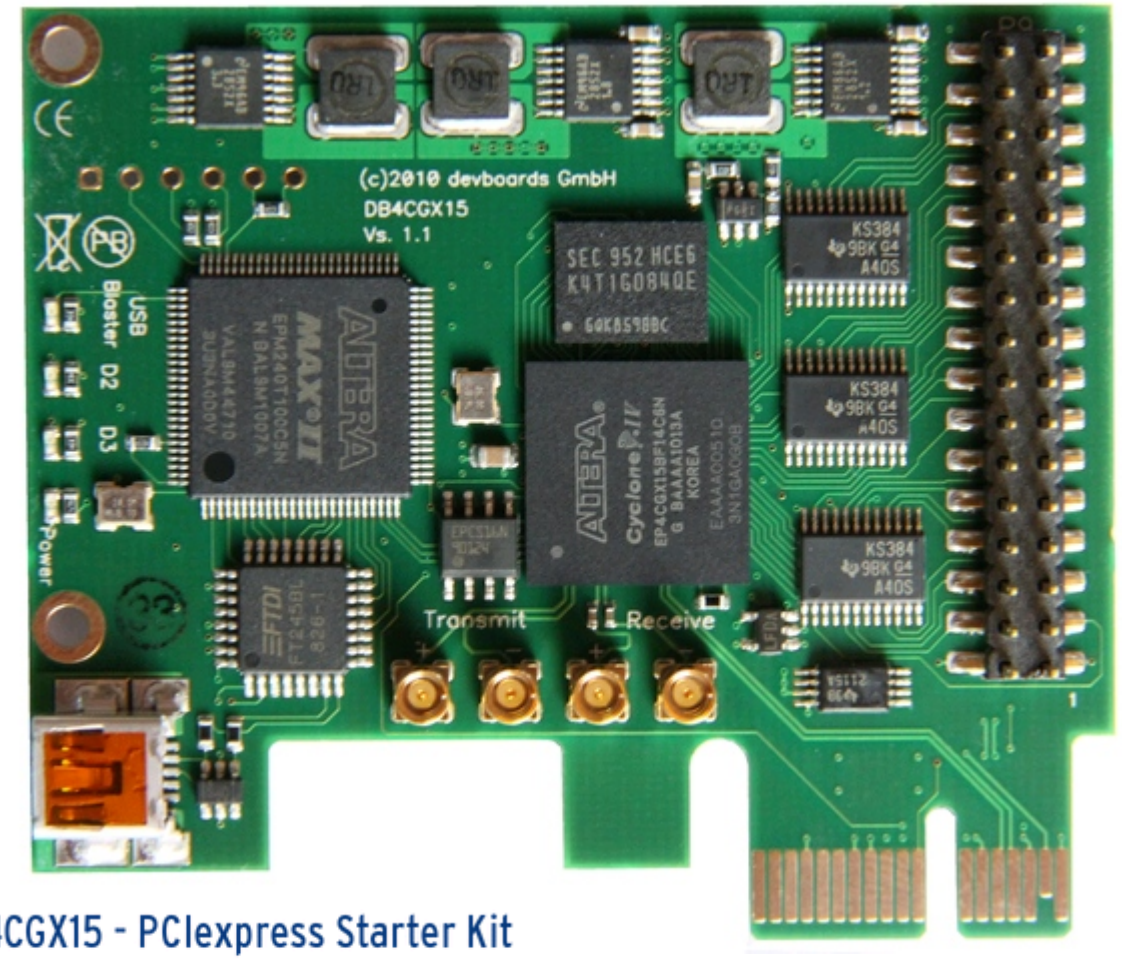
ExpressCard/34

PCIe 1×

The PCIe board DB4CGX15 Example

- PCIe x1
- Altera

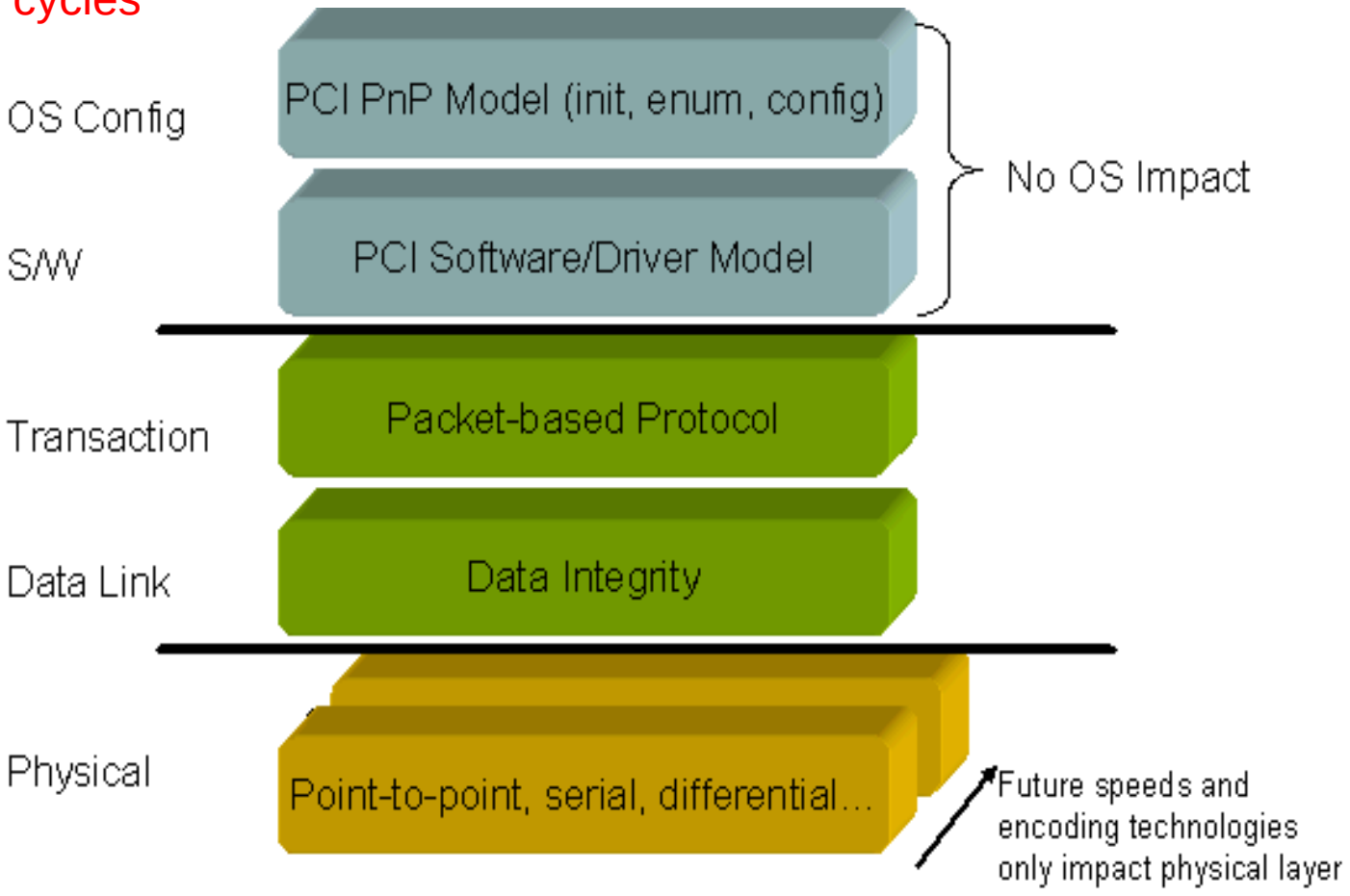
  EP4CGX15
  BF14C6N FPGA
- EPCS16 Configuration device
- 32Mbyte DDR2 SDRAM
- 20 I/O Pins
- 4 Input Pins
- 2 User LEDs



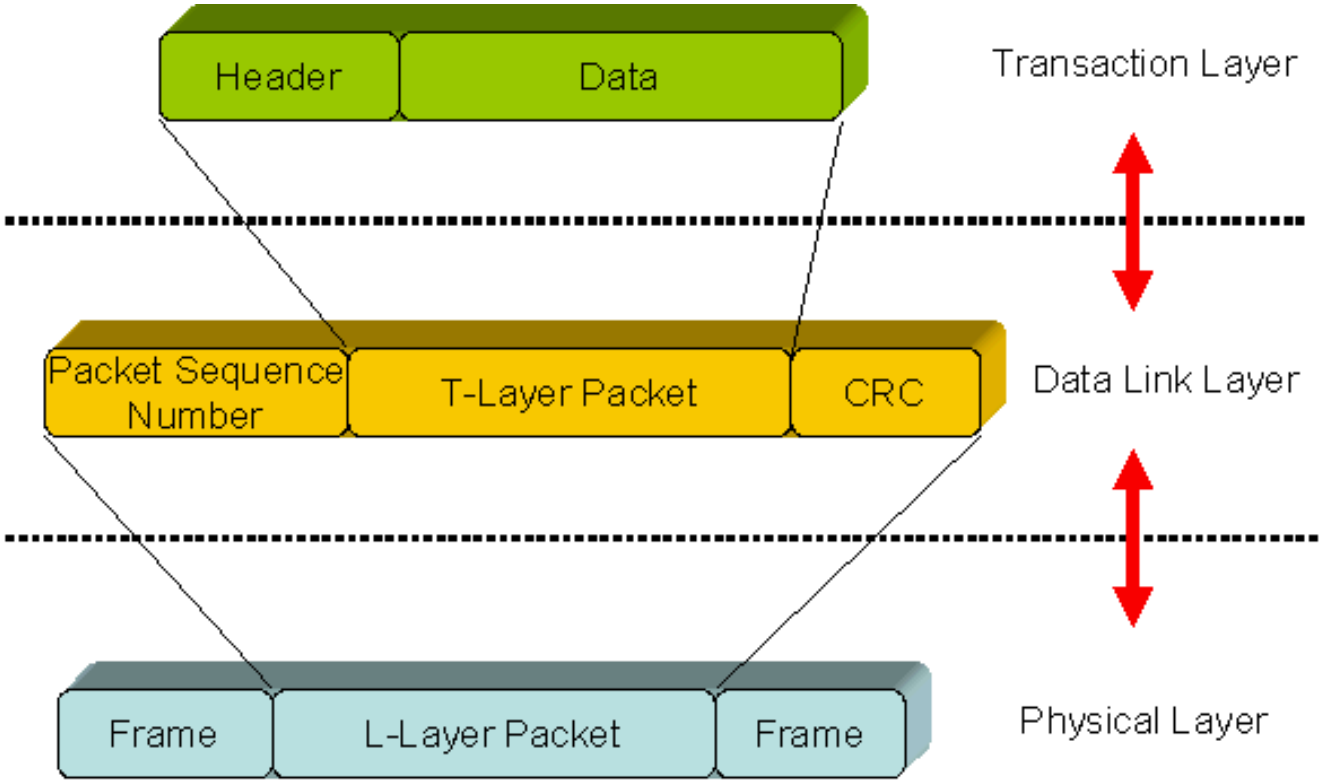
DB4CGX15 - PCIexpress Starter Kit
Development board for PCIexpress applications

PCIe Communication Protocol – HW and SW Layers

PCIe physical topology is serial, point-to-point, packet oriented, but its logical view and behavior is the same as PCI – that is multi-master bus, same enumeration, read/write cycles

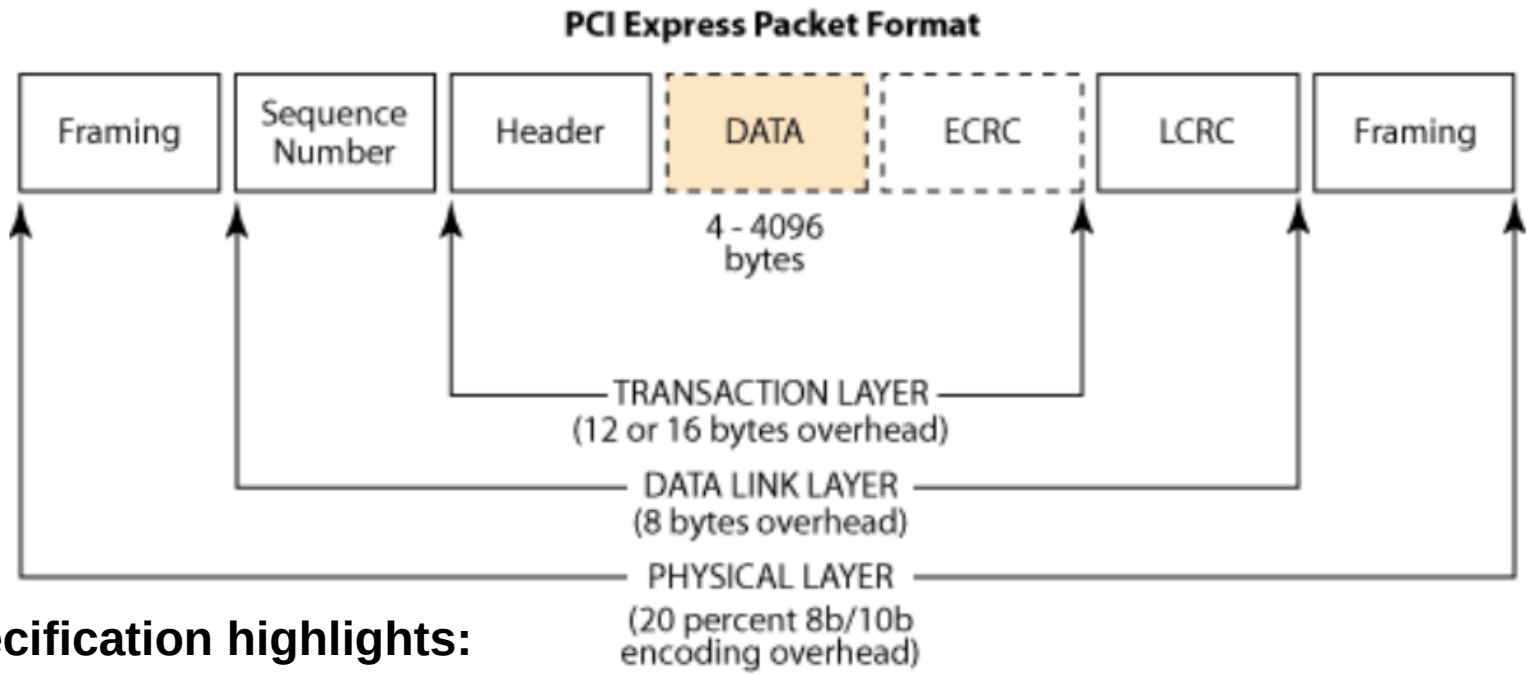


Data Packet Structure and Transport Layers



Source: http://zone.ni.com/devzone/cda/tut/p/id/3767#toc0

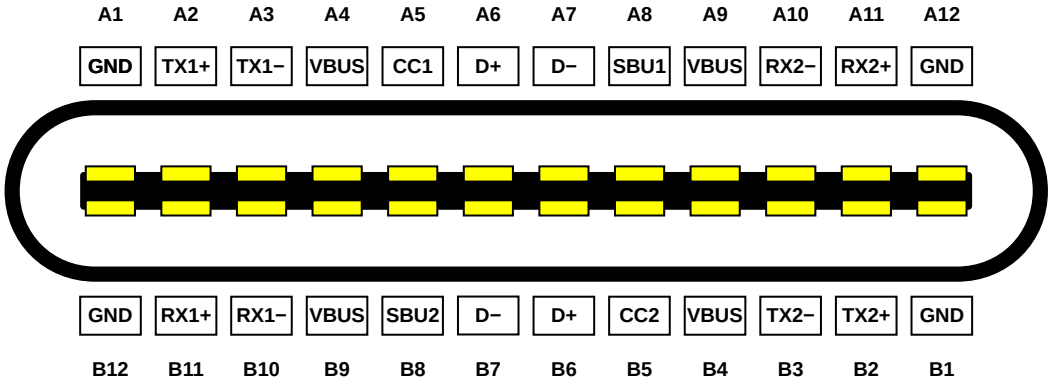
PCIe Packet Format



Specification highlights:

- Packet length from 4B to 4096B
- PCIe packed has o be exchanged as the whole (no option to preempt when higher priority transfer is requested)
- Long packed can cause increase of latencies in the system
- On the other hand, short packets overhead (frame/data) is considerable

USB-C, Alternative Modes, PCIe and Thunderbolt



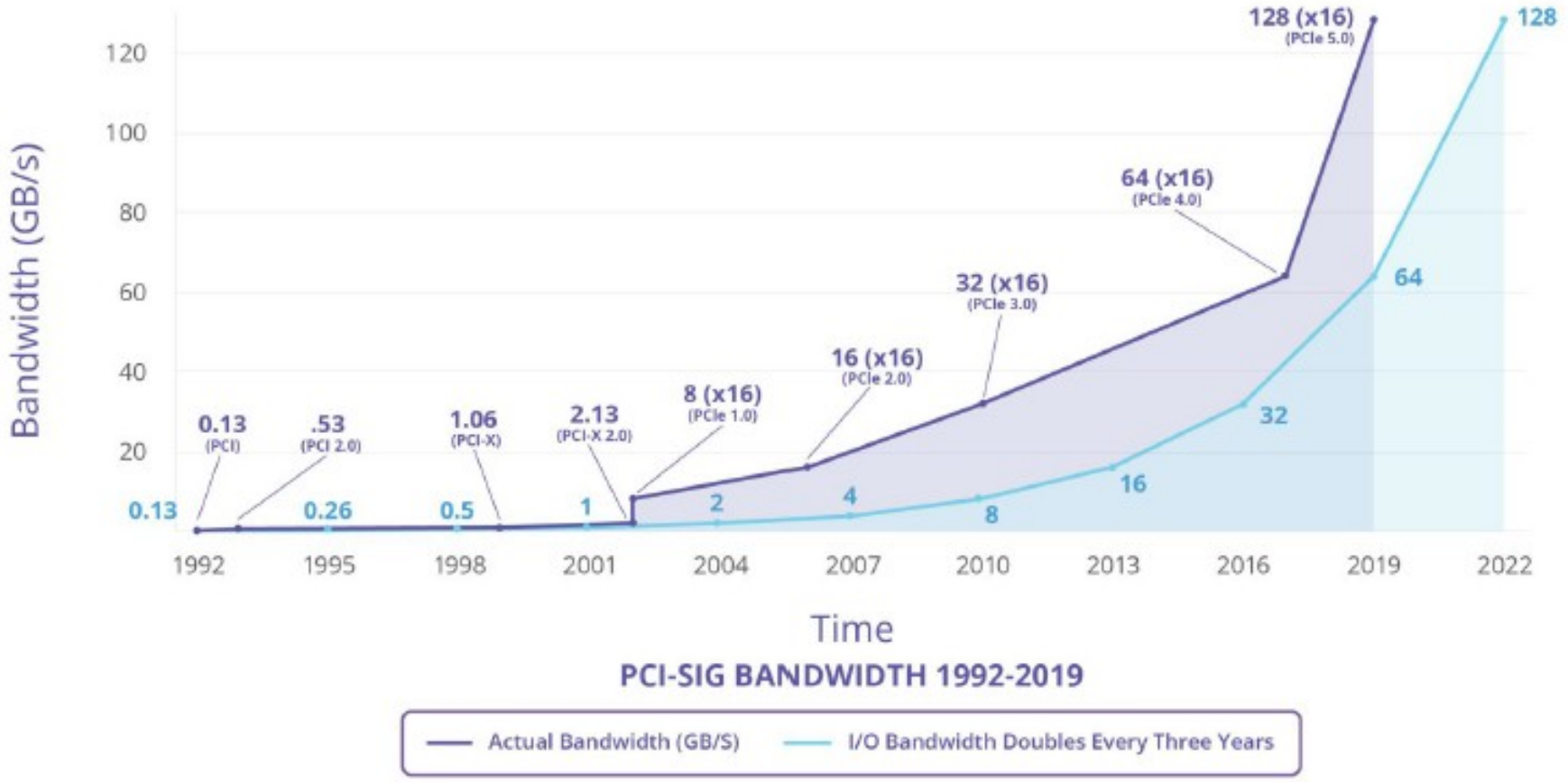
- SuperSpeed+ / USB 2.0 with up to 480 Mbit/s data rate
- 10 and 20 Gbit/s (1 and ~2.4 GB/s) when two lanes used
- USB Power Delivery 2.0 3 A current (@ 20 V, 60 W), high-power 5 A current (@ 20 V, 100 W).
- Cable electronic ID chip
- Alternate Mode partner specifications: DisplayPort, Mobile High-Definition Link, Thunderbolt, HDMI

USB-C Signals

Pin	Name	Description	Pin	Name	Description
A1	GND	Ground return	**B12**	GND	Ground return
A2	SSTXp1	SuperSpeed diff. pair #1, TX, pos.	**B11**	SSRXp1	SuperSpeed diff. pair #2, RX, pos.
A3	SSTXn1	SuperSpeed diff. pair #1, TX, neg.	**B10**	SSRXn1	SuperSpeed diff. pair #2, RX, neg.
A4	VBUS	Bus power	**B9**	VBUS	Bus power
A5	CC1	Configuration channel	**B8**	SBU2	Sideband use (SBU)
A6	Dp1	USB 2.0 diff. pair, position 1, pos.	**B7**	Dn2	USB 2.0 diff. pair, position 2, neg.
A7	Dn1	USB 2.0 diff. pair, position 1, neg.	**B6**	Dp2	USB 2.0 diff. pair, position 2, pos.
A8	SBU1	Sideband use (SBU)	**B5**	CC2	Configuration channel
A9	VBUS	Bus power	**B4**	VBUS	Bus power
A10	SSRXn2	SuperSpeed diff. pair #4, RX, neg.	**B3**	SSTXn2	SuperSpeed diff. pair #3, TX, neg.
A11	SSRXp2	SuperSpeed diff. pair #4, RX, pos.	**B2**	SSTXp2	SuperSpeed diff. pair #3, TX, pos.
A12	GND	Ground return	**B1**	GND	Ground return

Source: https://en.wikipedia.org/wiki/USB-C

PCI(e) I/O Bandwidth Expectations and Reality



PCIe buses overcome theoretical expectations

Source: Nextpatform

PCI-signal Bandwidth in Numbers

Year	Bandwidth	Frequency/Speed
1992	133MB/s (32 bit simplex)	33 Mhz (PCI)
1993	533MB/s (64 bit simplex)	66 Mhz (PCI 2.0)
1999	1.06GB/s (64 bit simplex)	133 Mhz (PCI-X)
2002	2.13GB/s (64 bit simplex)	266 Mhz (PCI-X 2.0)
2002	8GB/s (x16 duplex)	2.5 GHz (PCIe 1.x)
2006	16GB/s (x16 duplex)	5.0 GHz (PCIe 2.x)
2010	32GB/s (x16 duplex)	8.0 GHz (PCIe 3.x)
2017	64GB/s (x16 duplex)	16.0 GHz (PCIe 4.0)
2019	128GB/s (x16 duplex)	32.0 GHz (PCIe 5.0)

Source: Nextpatform

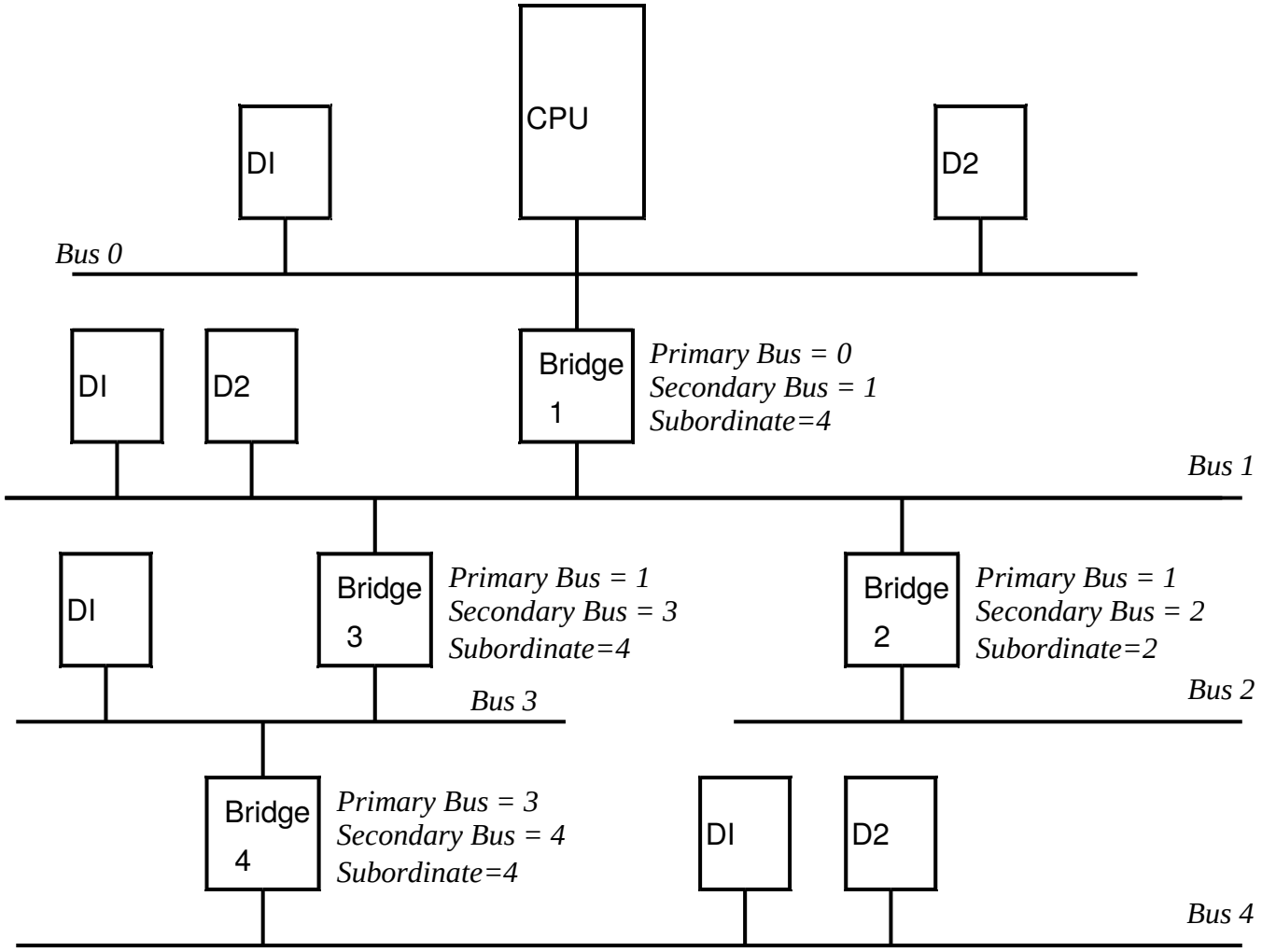
PCI/PCIe Programmer/Operating System Model

Computer Startup Procedure (from PCI perspective)

1. CPU is directed by BIOS code to retrieve device identification for each PCI slot. This is done by read cycle from PCI **configuration space**. The read (topological) address decodes to **IDSEL** (**Initialization Device Select**) signal to the corresponding PCI slot (bus/device/function) + register number
2. Each device identification (Vendor ID, Device ID) and request for I/O **resources** (sizes of I/O ports and memory ranges and interrupt link (A/B/C/D) use by function) are read. All this information is available in card/slot configuration space. This search is done together with bus numbers assignment when bridge is found.
3. BIOS allocates non-overlapping ranges to the devices. It ensures that there is no collision with system memory and I/O. Interrupts can be, and are, shared but sharing level can be balanced. Allocated ranges/resources are written to the corresponding device/function **Base Address Register** (**BAR**). They usually stay constant till computer power off but OS can reconfigure them under certain circumstances.
4. Operating System is loaded and given control. OS reads devices identifications again from PCI configuration space and locates device drivers according to VID:PID (+class,+subsystem IDs).

- This process of device “searching” is called **enumeration** and is used in some form by each PnP aware bus (PCI, USB, etc.).

PCI Bus Hierarchy

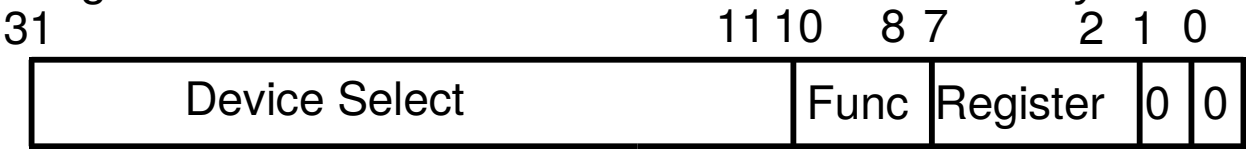


PCI BUS Address Space(s)

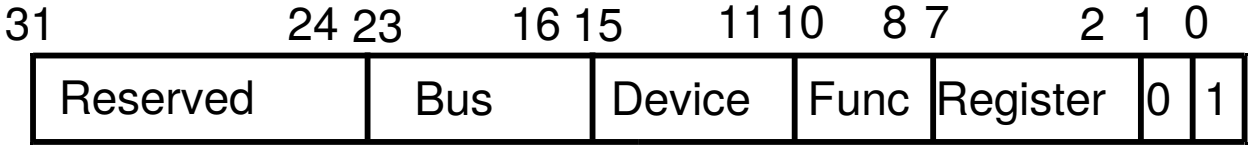
- PCI bus recognizes three address spaces:
  - **memory** – address is 32 or 64-bit
  - **I/O** – exists mainly for compatibility with x86 specific I/O ports and I/O instructions concept
  - **configuration space** – 256 bytes are assigned to each device function in the basic PCI bus variant, 8 functions per device/slot/card and 32 devices per bus can exist in maximum.
- Each end-point device can implement up to 6 Base Address Registers (BARs) which can define up to 6 independent regions (address ranges) – each for I/O or memory mapped access. For 64-bit ranges BARs are used in pairs. The requested size is obtained by writing ones into BAR bits and reading back where BAR's bits corresponding to the range size are fixed on zero. LSB bits then informs about address space type. Then BIOS/OS writes allocated region start address back to the BAR.

PCI Configuration Space Address and Access

1 from n original IDSEL activation address – not used today



Topological/geographical BDF address

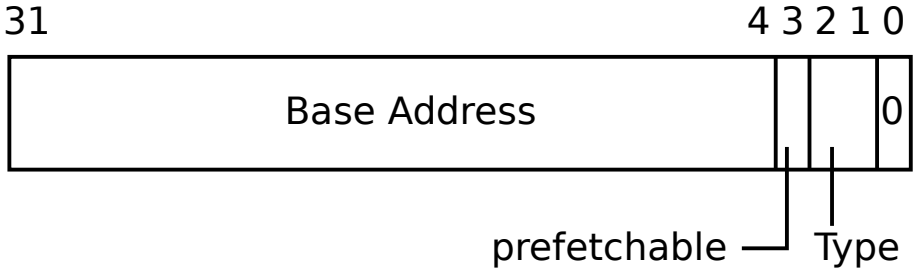


- There are two mechanisms of accessing configuration space on x86 PC:
  - Through well known I/O ports

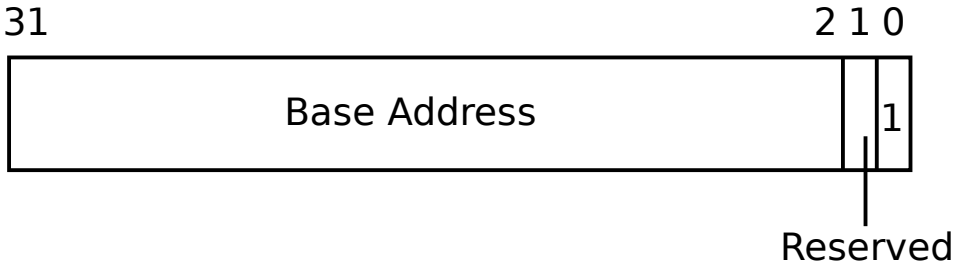
    `0xCF8` – PCI CONFIG_ADDRESS (write address first, A0:1=0)

    `0xCFC` – PCI CONFIG_DATA (read/write corresponding byte, 16-bit or 32-bit entity, address bits 0 and 1 added to **`0xCFC`**)
  - Enhanced Configuration Access Mechanism (ECAM) – required for PCI express – 4kB per slot, memory mapped

PCI Device Header

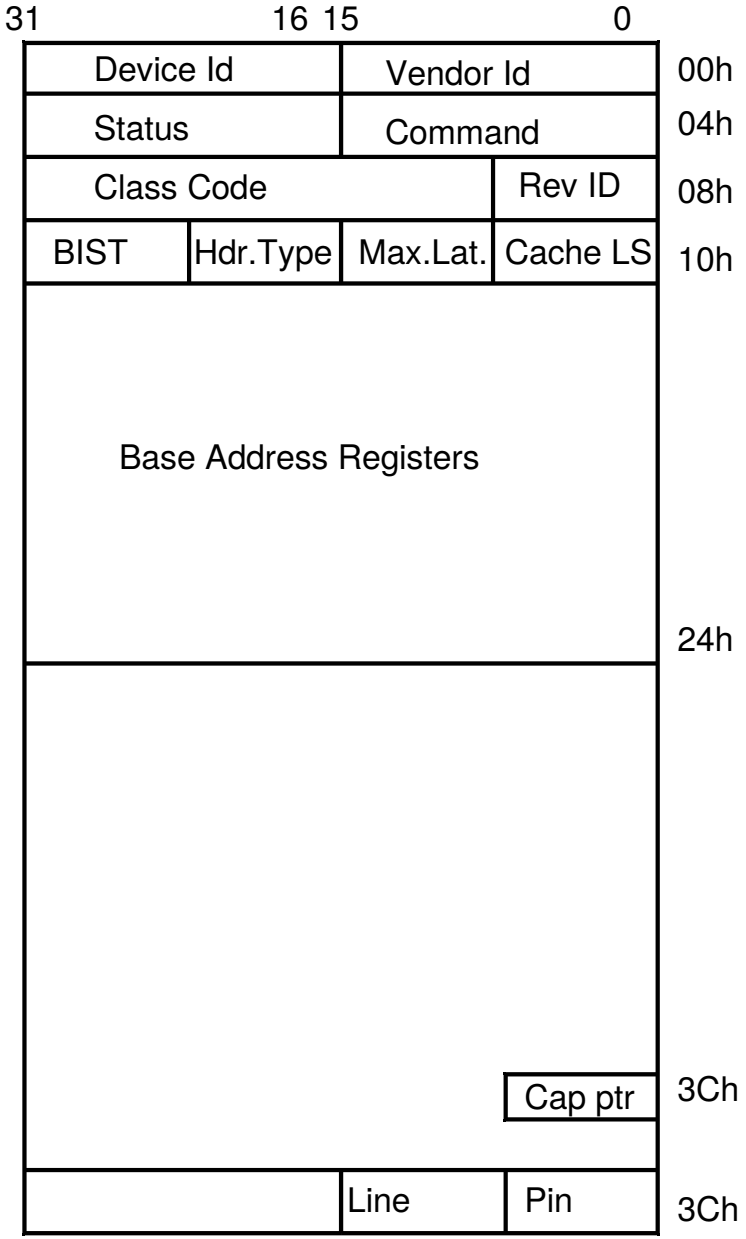


Base Address for PCI Memory Space

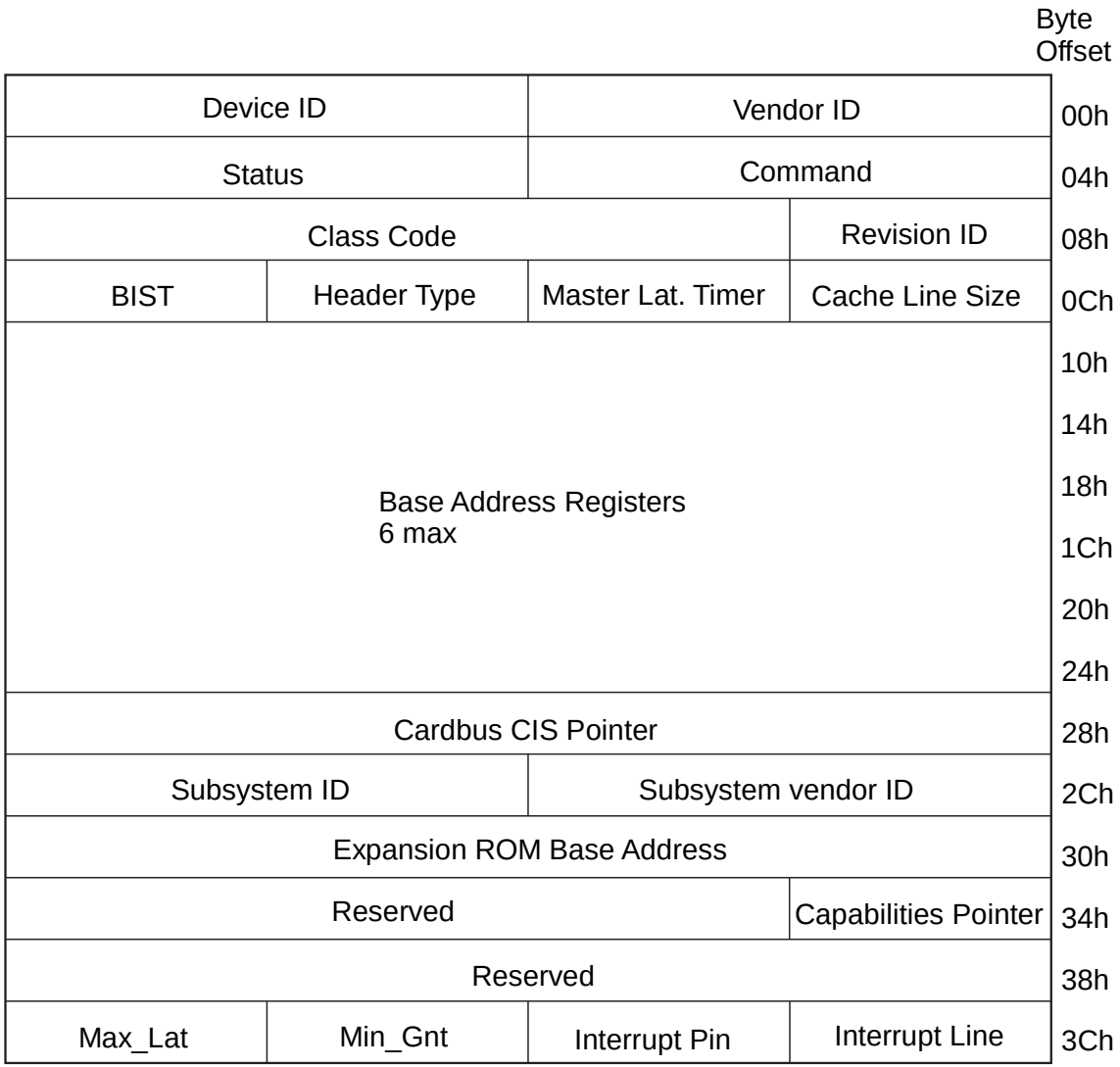


Base Address for PCI I/O Space

Device's PCI header is located in PCI bus configuration space

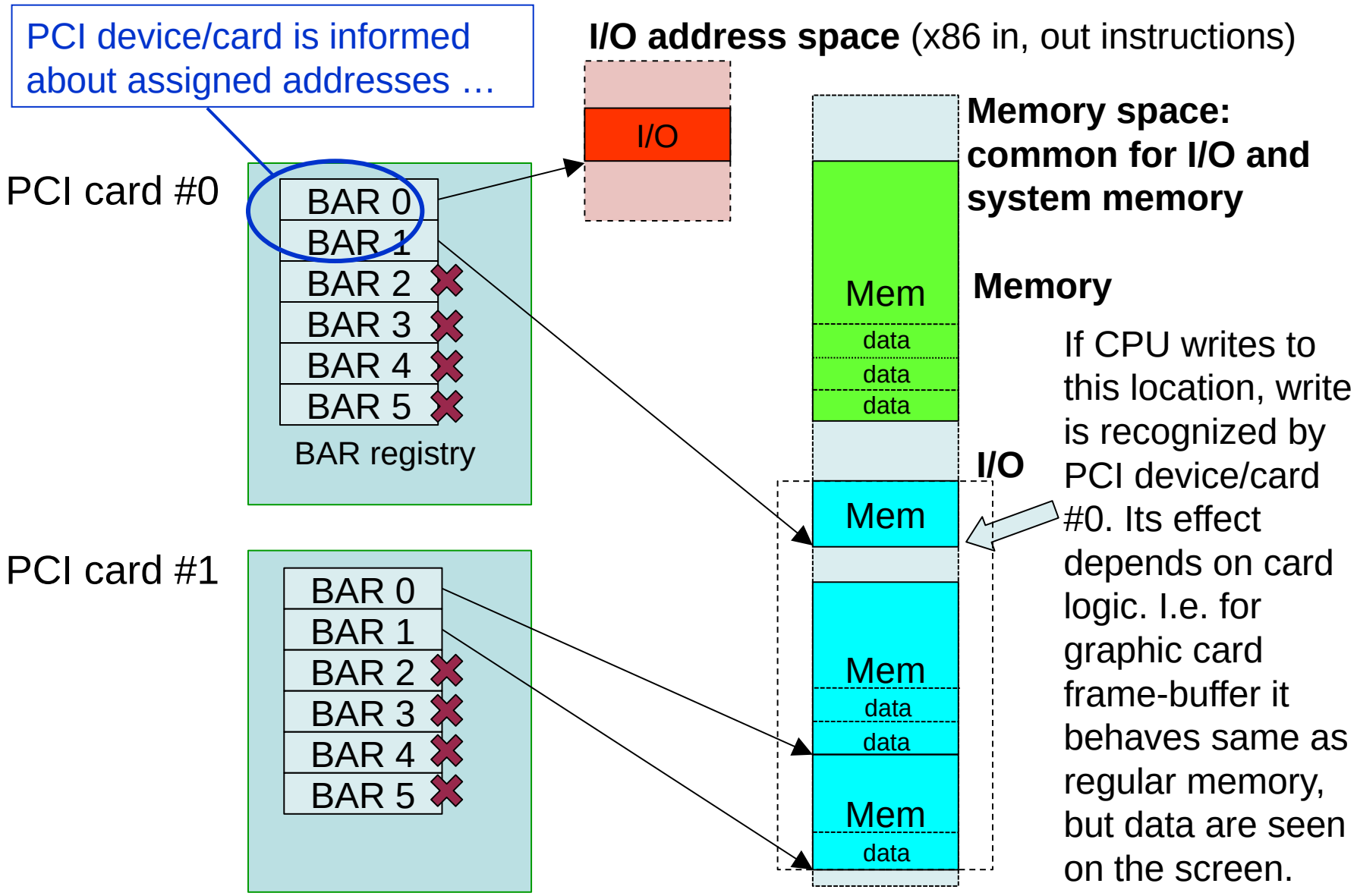


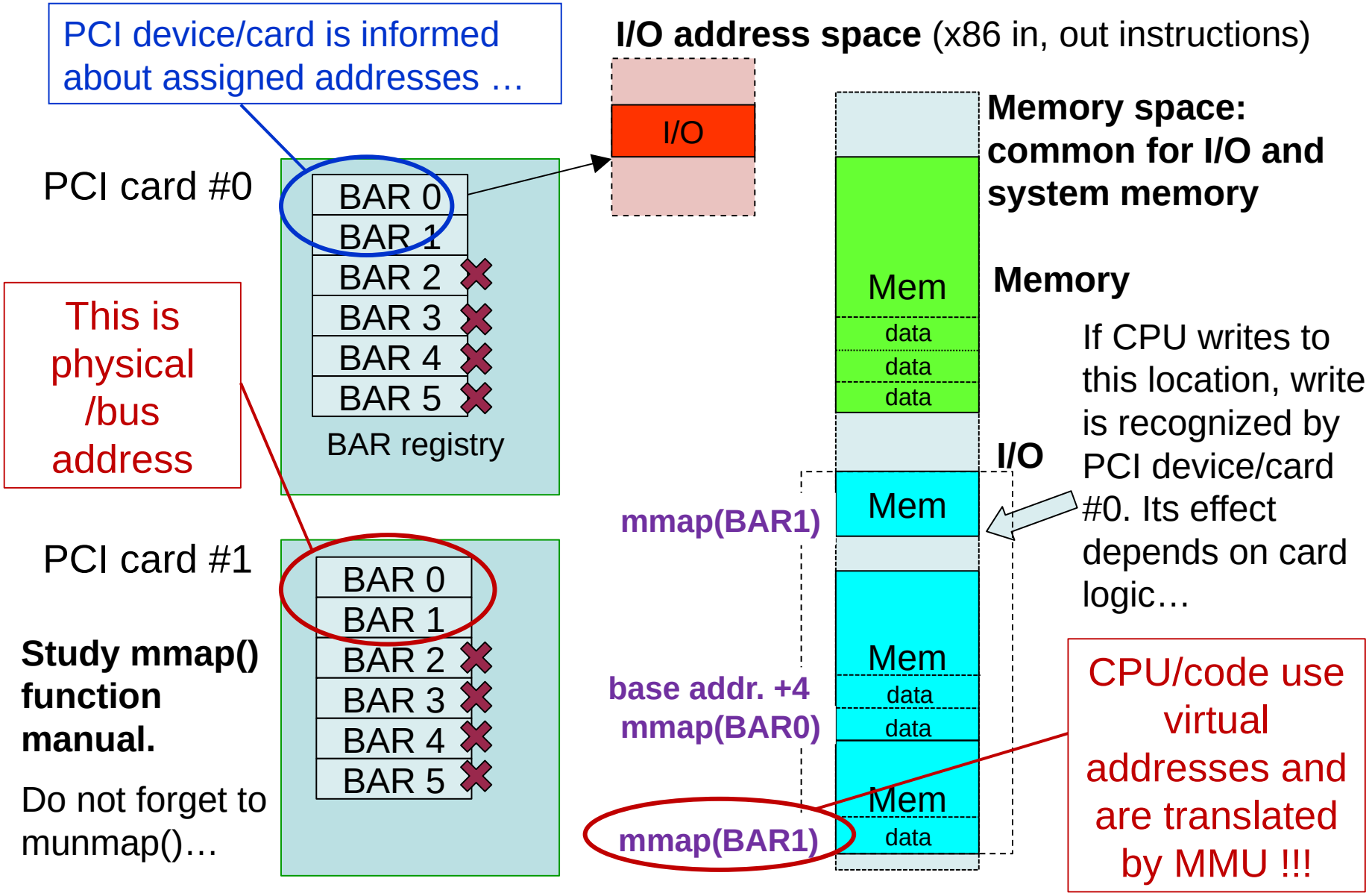
PCI Device Header Type 0 – End-point device



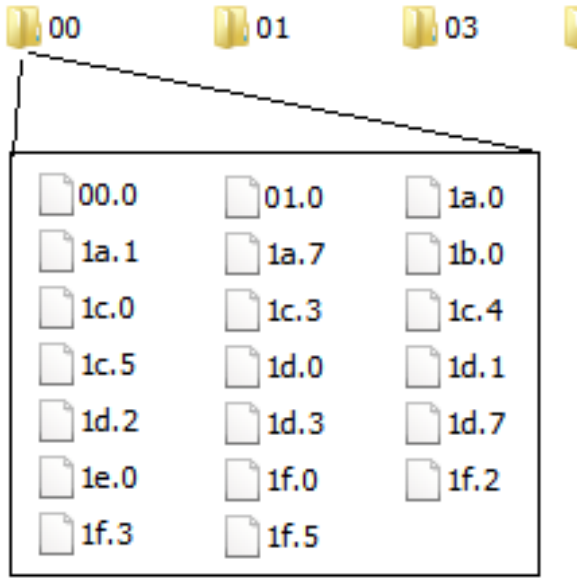
PCI Device Header Type 1 – Bus Bridges

				Byte Offset
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Master Lat. Timer	Cache Line Size	0Ch
Base Address Register 0				10h
Base Address Register 1				14h
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18h
Secondary Status		I/O Limit	I/O Base	1Ch
Memory Limit		Memory Base		20h
Prefetchable Memory Limit		Prefetchable Memory Base		24h
Prefetchable Base Upper 32 Bits				28h
Prefetchable Limit Upper 32 Bits				2Ch
I/O Limit Upper 16 Bits		I/O Limit Base Upper 16 Bits		30h
Reserved			Capabilities Pointer	34h
Expansion ROM Base Address				38h
Bridge Control		Interrupt Pin	Interrupt Line	3Ch


PCI device/card is informed about assigned addresses …
I/O address space (x86 in, out instructions)
I/O
Memory space: common for I/O and system memory
PCI card #0
BAR 0
BAR 1
BAR 2
BAR 3
BAR 4
BAR 5
BAR registry
Mem
data
data
data
Memory
I/O
Mem
PCI card #1
BAR 0
BAR 1
BAR 2
BAR 3
BAR 4
BAR 5
Mem
data
data
Mem
data
If CPU writes to this location, write is recognized by PCI device/card #0. Its effect depends on card logic. I.e. for graphic card frame-buffer it behaves same as regular memory, but data are seen on the screen.


PCI device/card is informed about assigned addresses …
I/O address space (x86 in, out instructions)
I/O
Memory space: common for I/O and system memory
PCI card #0
BAR 0
BAR 1
BAR 2
BAR 3
BAR 4
BAR 5
BAR registry
Mem
data
data
data
Memory
This is physical /bus address
I/O
Mem
mmap(BAR1)
If CPU writes to this location, write is recognized by PCI device/card #0. Its effect depends on card logic…
PCI card #1
BAR 0
BAR 1
BAR 2
BAR 3
BAR 4
BAR 5
Mem
data
data
base addr. +4
mmap(BAR0)
Mem
data
mmap(BAR1)
CPU/code use virtual addresses and are translated by MMU !!!
Study mmap() function manual.
Do not forget to munmap()…

Linux **/proc/bus/pci** Directory




```
          00 01 02 03  04 05 06 07  08 09 0a 0b  0c 0d 0e 0f
00000000  72 11 32 1f  07 00 10 00  01 00 00 ff  08 00 00 00  r.2.....'....
00000010  00 00 8f fe  00 00 00 00  00 00 00 00  00 00 00 00  ..Źţ............
00000020  00 00 00 00  00 00 00 00  00 00 00 00  72 11 32 1f  ............r.2.
00000030  00 00 00 00  50 00 00 00  00 00 00 00  0b 01 00 00  ....P...........
00000040  .. .. .. ..  .. .. .. ..  .. .. .. ..  .. .. .. ..  ................
00000050  .. .. .. ..  .. .. .. ..  .. .. .. ..  .. .. .. ..  ................
```

- Each directory represents one PCI bus (with its number assigned) and each file mirrors one PCI device function PCI header (the first 64 bytes)
- Homework: Write C/C++ language program that can traverse and open files in given directory and its subdirectory and searches for given sequence of four characters (4B) at each file start. The full path of the first matching file is printed.

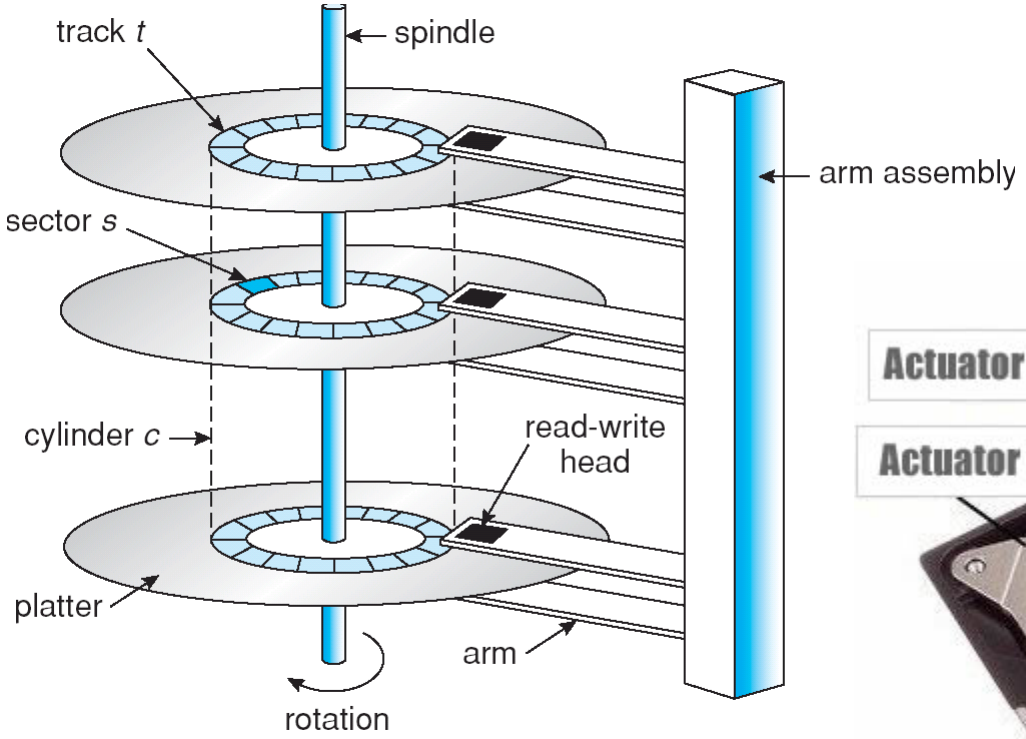
Linux **/proc/bus/pci** Directory – Command **lspci -vb**

```
00:00.0 Host bridge: Intel Corporation 82X38/X48 Express DRAM Controller
        Subsystem: Hewlett-Packard Company Device 1308
        Flags: bus master, fast devsel, latency 0
        Capabilities: [e0] Vendor Specific Information <?>
        Kernel driver in use: x38_edac
        Kernel modules: x38_edac

00:01.0 PCI bridge: Intel Corporation 82X38/X48 Express Host-Primary PCI
Express Bridge
        Flags: bus master, fast devsel, latency 0
        Bus: primary=00, secondary=01, subordinate=01, sec-latency=0
        I/O behind bridge: 00001000-00001fff
        Memory behind bridge: f0000000-f2ffffff
        Kernel driver in use: pcieport
        Kernel modules: shpchp

00:1a.0 USB Controller: Intel Corporation 82801I (ICH9 Family) USB UHCI
Controller #4 (rev 02)
        Subsystem: Hewlett-Packard Company Device 1308
        Flags: bus master, medium devsel, latency 0, IRQ 5
        I/O ports at 2100
        Capabilities: [50] PCI Advanced Features
        Kernel driver in use: uhci_hcd
```

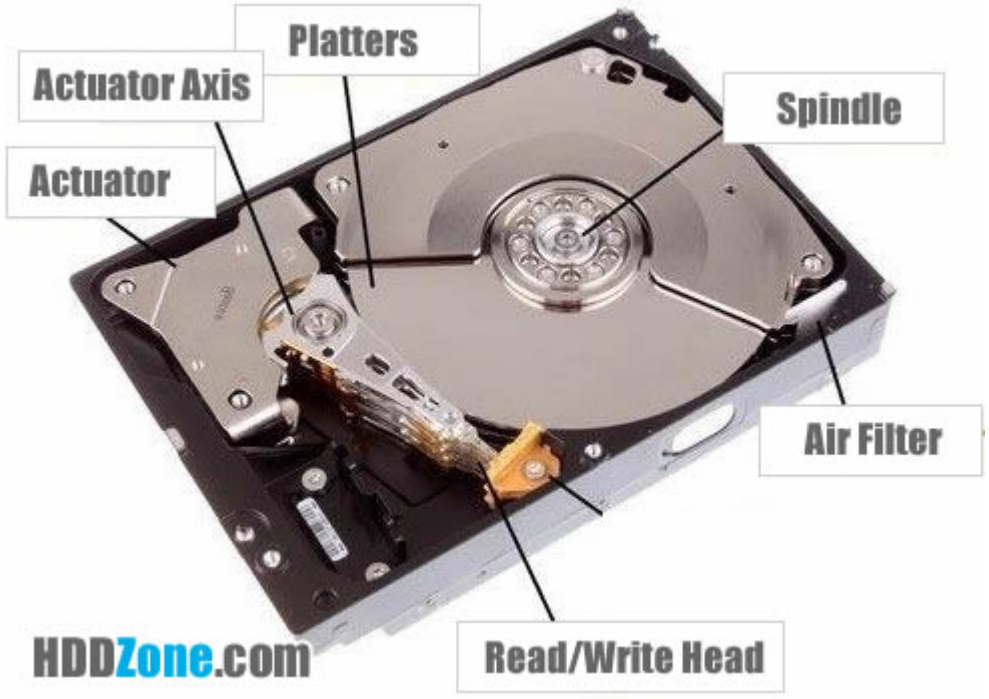
Disk – Another Critical Memory Hierarchy Component

- Enhancement required
  - Speedup
  - Reliability

Hard Drive

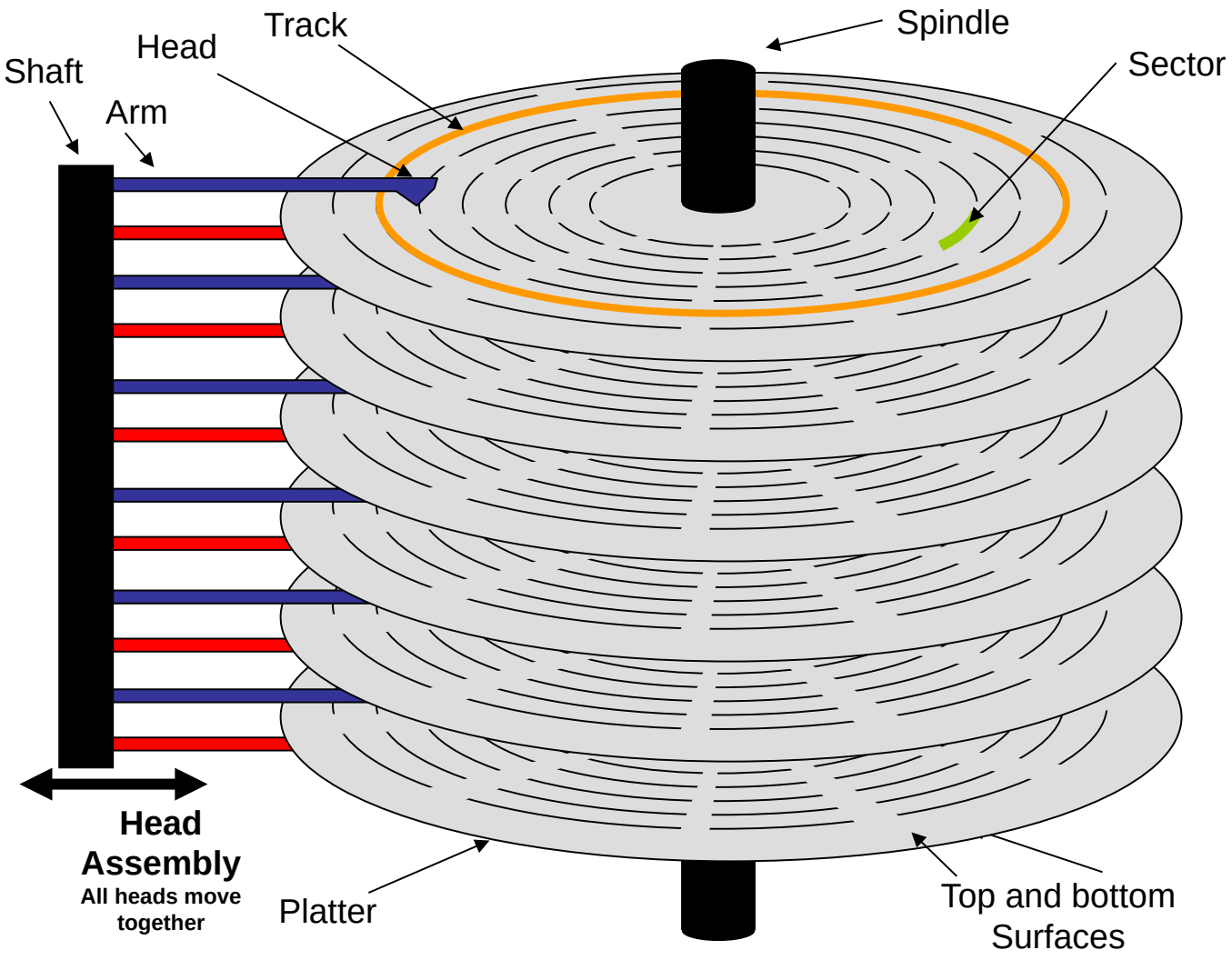


Drawing source: Junfeng Yang



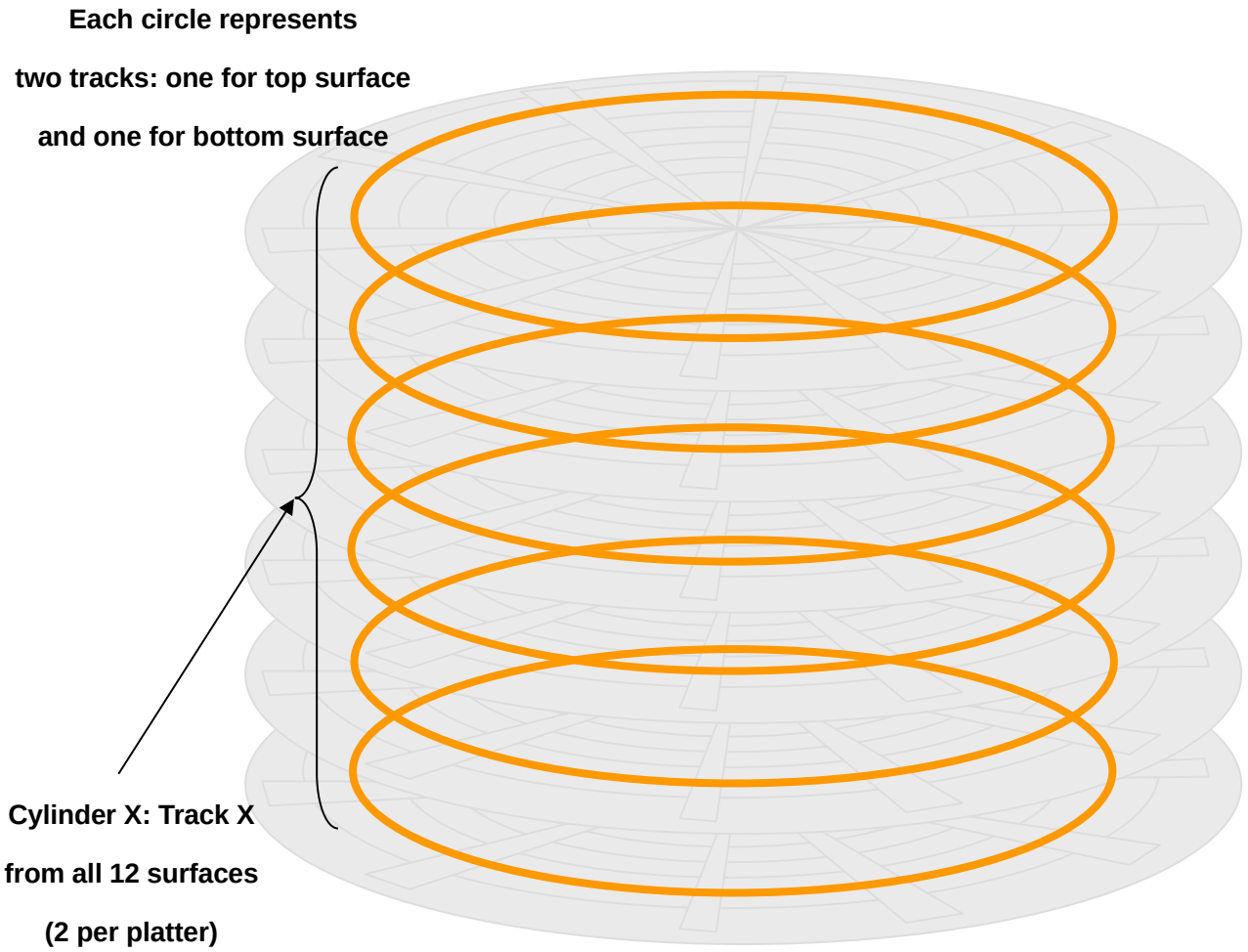
Picture source: https://www.hddzone.com/hard_disk_drive_compontents.html

Disk Storage



Source: U. Ramachandran and W. D. Leahy Jr.: An Integrated Approach to Architecture and Operating Systems , 2008

Disk Storage



Source: U. Ramachandran and W. D. Leahy Jr.: An Integrated Approach to Architecture and Operating Systems , 2008

Sequential v.s. Random

A sequential access

- Seek to the right track
- Rotate to the right sector
- Transfer

A random access of the same amount of data

- Seek to the right track
- Rotate to the right sector
- Transfer
- Repeat

Since seek and rotate are slow, sequential access is much faster than random access

Examples of HD Parameters

	Barracuda 180	**Cheetah X15 36LP**
Capacity	181GB	36.7GB
Disk/Heads	12/24	4/8
Cylinders	24247	18479
Sectors/track	~609	~485
Speed	**7200 RPM**	**15000 RPM**
Rotational latency (ms)	4.17	2.0
Avg seek (ms)	7.4	3.6
Track-2-track(ms)	0.8	0.3

Table source: Junfeng Yang

Maximum RPM

Sonar boom - source: U.S. Navy/Travis K. Mendoza

The sound velocity at the HDD circuit is reached at RPM

Diameter	RPM
3.5 inch	73105
2.5 inch	102347

But RPM also increases spinning and centrifugal power

IOPS

IOPS stands for **input/output operations per second**

IOPS=1 / (Average Seek Time + Average Latency)

Pameter	HD	SSD
IOPS	55-180	3000-40000
IOPS/Watt	10-30	2000-60000
$/GB	$0.02 - $0.04	$0.25-$0.50
Data retention if unplugged	10 - 60 years	1 - ?100? years / 25 °C *SSD are used for short time, reliable data are unknown yet*

Backblaze Hard Drive Stats
https://www.backblaze.com/b2/hard-drive-test-data.html

SSD vs HDD

Google test of SSD:

The **annual** replacement rates of hard disk drives have previously been reported to be **2-9%**, which is high compared to the **4-10%** of flash drives we see being replaced in a ***4 year period***.

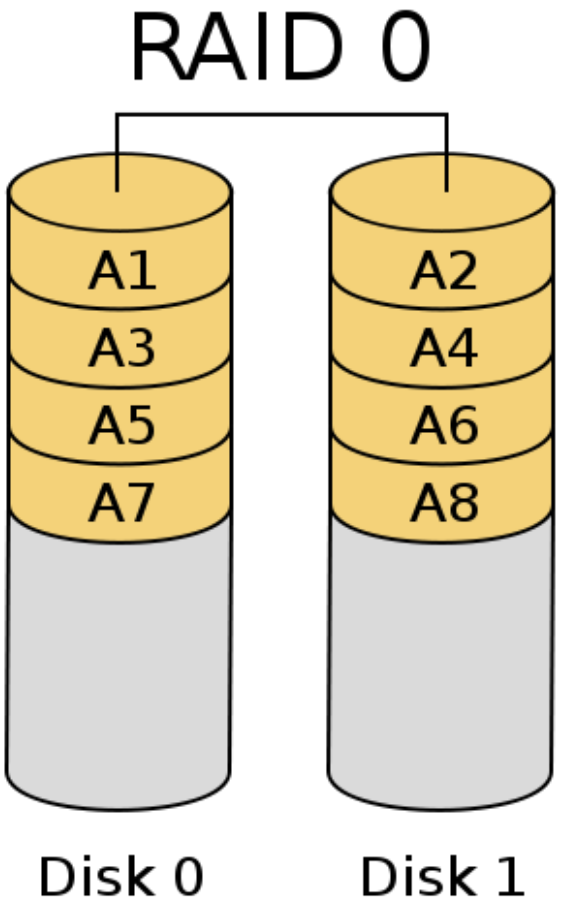
However, flash drives are less attractive when it comes to their error rates. More than **20%** of flash drives develop uncorrectable errors in a **four year** period, **30-80%** develop bad blocks and **2-7%** of them develop bad chips.

In comparison, previous work [1] on HDDs reports that only **3.5%** of disks in a large population developed bad sectors in a **32 months period** – a low number when taking into account that the number of sectors on a HDD is orders of magnitudes larger than the number of either blocks or chips on a SSD, and that sectors are smaller than blocks, so a failure is less severe.

[1] **An analysis of latent sector errors in disk drives.** BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, JIn *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, ACM, pp. 289–300.

RAID 0

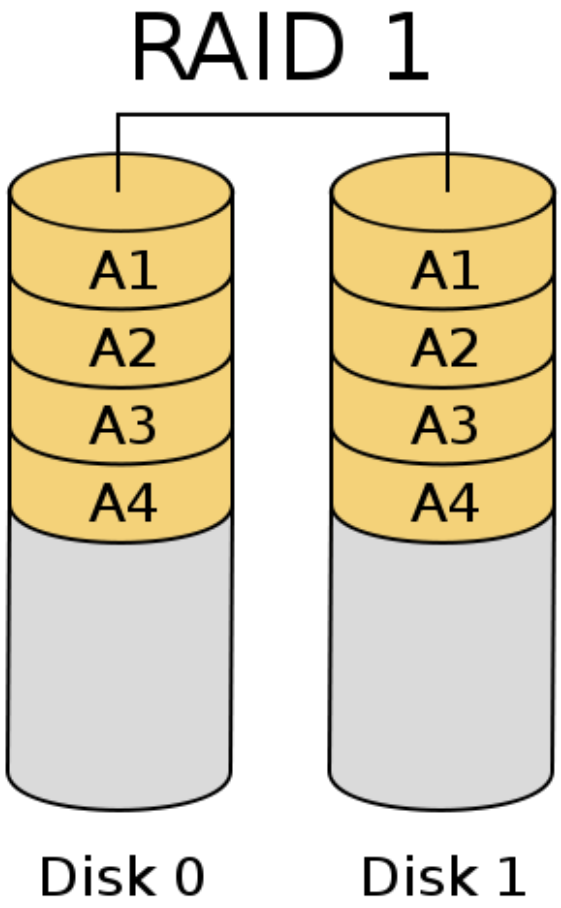
- RAID – Redundant Array of Inexpensive/Independent Disks
- Can be used to achieve higher performance/throughput of the hard disks
- Method called **stripping**
- Raw bandwidth up to two times higher
- Capacity is sum of the both devices



Images source: Wikipedia

RAID 1

- Each data block exists in two copies, each on one of two independent disks
- The total capacity is same as of a single disk
- Data reliability is much higher, probability of coincidence of two independent events (disk failures) is much much lower than for single device
- Method is called **mirroring**
- Write has some overhead against single device. Reads can be optimized for less head movement



RAID 10

- It is combination of both previous techniques
- RAID 0 is created first on two (or more) devices and all data are copied on the second set of devices (same as for RAID1)
- RAID 10 contributes to both – reliability and performance
- Disadvantage – at least 4 drives with same capacity are required.
- Total capacity T, disk capacity D, number of drives n

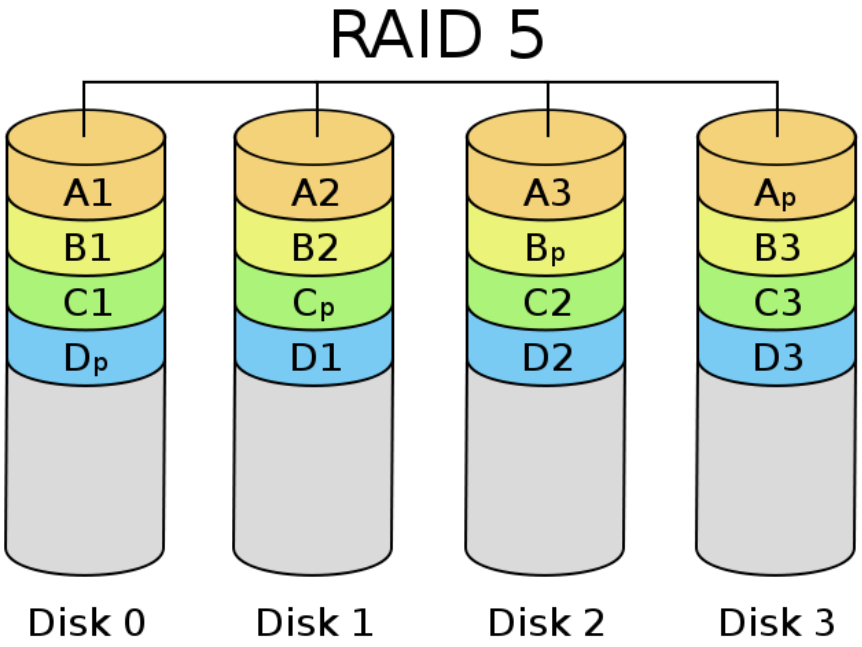
RAID 0 $n = 2 \cdot ceil\left(\frac{T}{2 \cdot D}\right)$ **RAID 1** $n = 2 \cdot ceil\left(\frac{T}{D}\right)$

RAID 10 $n = 4 \cdot ceil\left(\frac{T}{2 \cdot D}\right)$

RAID 5

- The data blocks are distributed over n-1 drives (for each disk LBA) and last block represents parity (XOR for example) of previous blocks
- But disk used for parity is chosen sequentially for each disk LBA – it balances number of rewrites and speed gain for degraded mode
- It speeds-up reads, single block write is slower because of checksum computation overhead

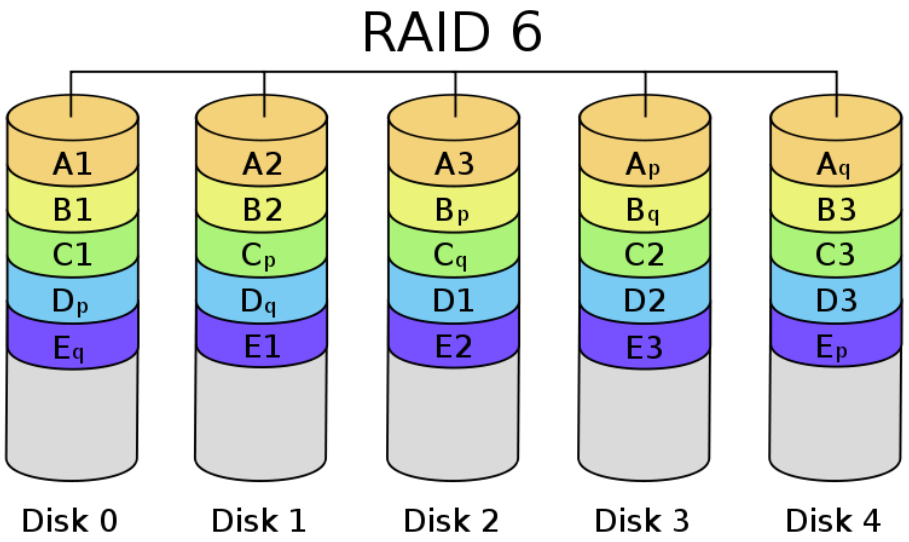
$$n = ceil\left(\frac{T}{D}\right) + 1$$



RAID 6

- Uses two parity blocks on different disks for given disk LBA. Each parity is computed different way.
- It is resistant to two concurrent disk failures
- The read is speed similar to RAID 5, write is more demanding/complex

$$n = ceil\left(\frac{T}{D}\right) + 2$$



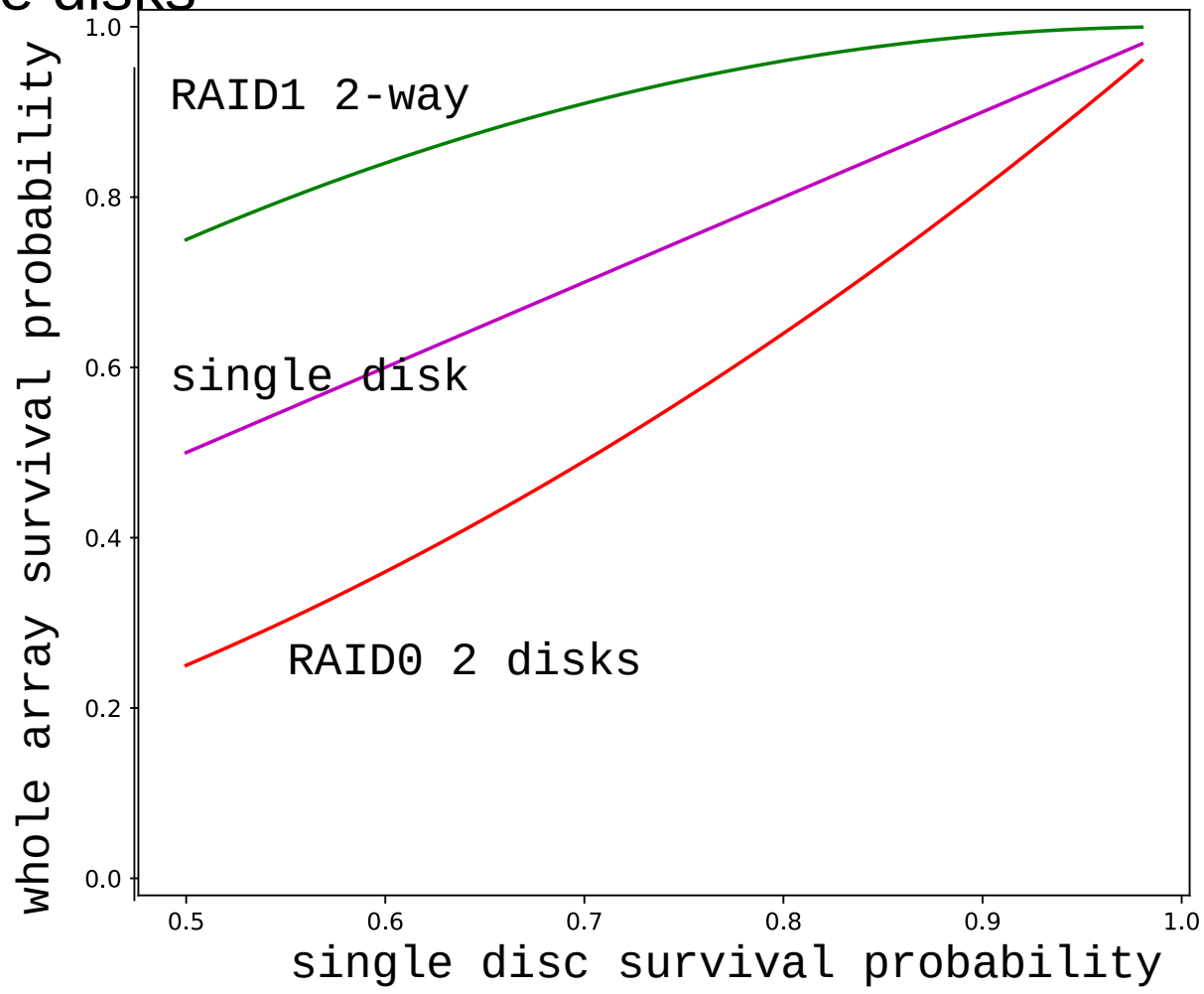
RAID Discs Failure Probability for 2 Disks

Use of 2 same disks

- RAID0

  T = 2D
- RAID1

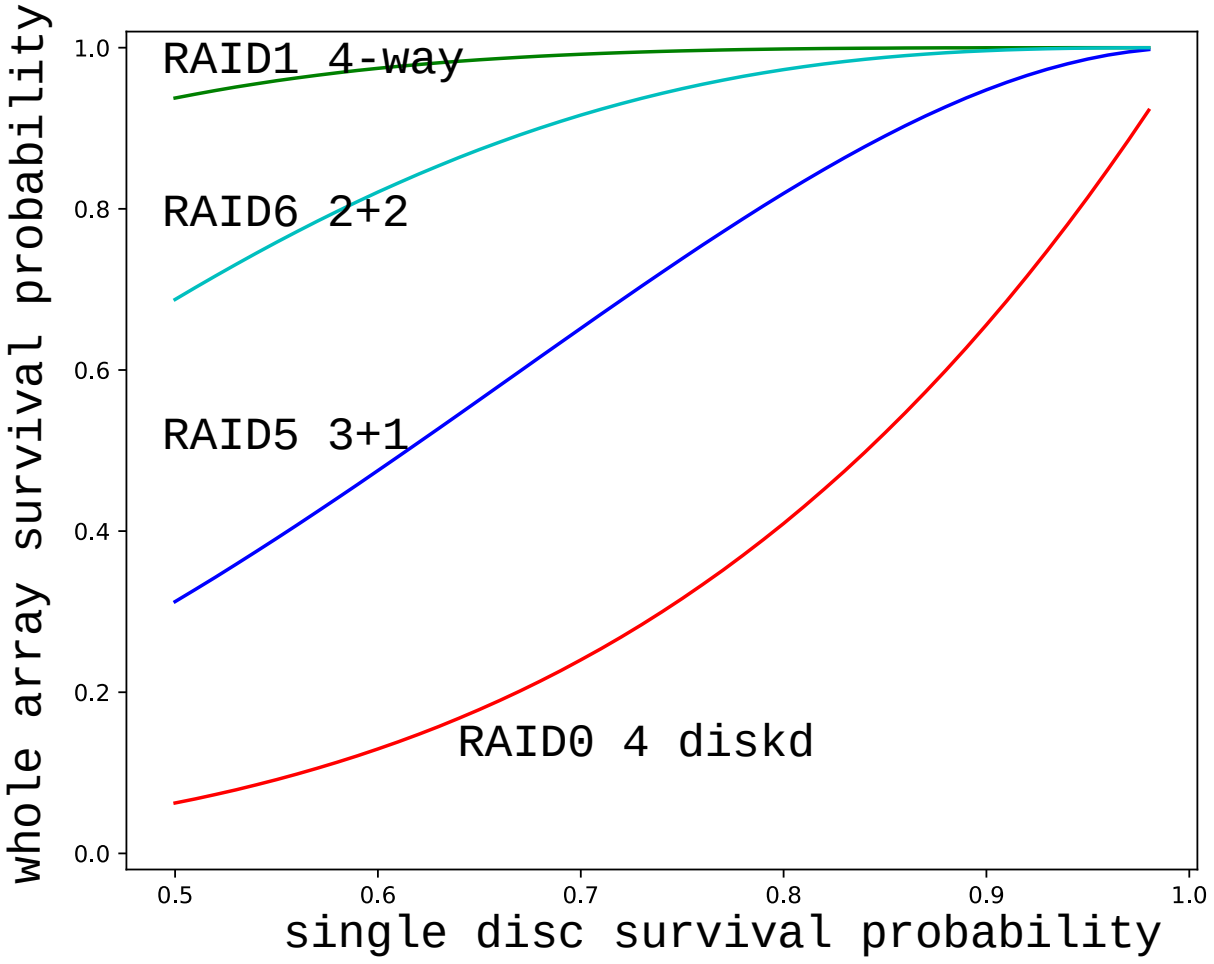
  T = D



RAID Discs Failure Probability for 4 Disks

Use of 4 same disks

- RAID0
  T = 4D
- RAID1
  T = D
- RAID5
  T = 3D
- RAID6
  T = 2D



RAID Discs Failure Probability for 6 Disks

Use of 6 same disks

- RAID0
  T = 6D
- RAID1
  T = D
- RAID5
  T = 5D
- RAID6
  T = 4D