

# B35APO: Computer Architectures

## Lecture 07. Input and Output

Pavel Píša

pisa@fel.cvut.cz

Petr Štěpán

stepan@fel.cvut.cz



17. června, 2025

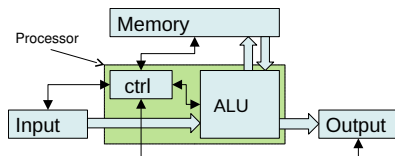
# Outline

- 1 Input and Output
- 2 QtRvSim Peripherals
- 3 Internal Interconnection Buses

# Today's Lecture Objective

- Review what are the input and output options in a computer
- Memory-mapped peripherals
- Examples in QtRvSim
- PCI and PCIe buses

# Computer Architecture – John von Neumann



- 5 basic units – control unit, arithmetic-logic unit, memory, input (input device), output (output device)
- The architecture of a computer should not depend on the task being solved, it should be able to execute a program stored in memory. The program controls which sequence of instructions computer executes and thus what results are computed.
- The program and data are stored in the same memory, composed of cells (units) of the same size. In contrast, the Harvard architecture had one type of memory for the program and another type of memory for data.
- The next instruction to be executed is stored in the next memory location (excluding program jumps)
- Instructions perform arithmetic and logical operations, data transfers to/from memory, program jumps and branches, and special control instructions.

# Classification of Input/Output Devices/Peripherals

By behavior:

- Input (read only)
- Output (write only, cannot be read)
- Input and output (currently, most devices, including keyboards – they have an LED output)

By connection:

- Direct connection between CPU and peripherals
- Hierarchical – connection via other peripherals (bridge, switch)

By partner kind:

- Human – other communication parameters
- Computer – usually faster communication
- Environment – sensors and actuators

By communication link/bus parameters:

- Capacity/bandwidth of the link – maximum data transfer capabilities
- Latency – time in which data transfer is performed

# Classification Peripherals – Continues

Examples of human-machine peripherals:

- keyboard – only input, but often output on LED diodes, very small transmission speed, latency up to 200ms (except games playing)
- microphone/speakers – transfer speed up to 8Mb/s, latency depends on application, for interactive communication (i.e. calls) requires latency of less than 500 ms, optimally 150 – 300 ms
- printer/scanner – transfer speed according to connection, latency does not matter (in seconds / minutes)

Examples of peripherals for communication between computers

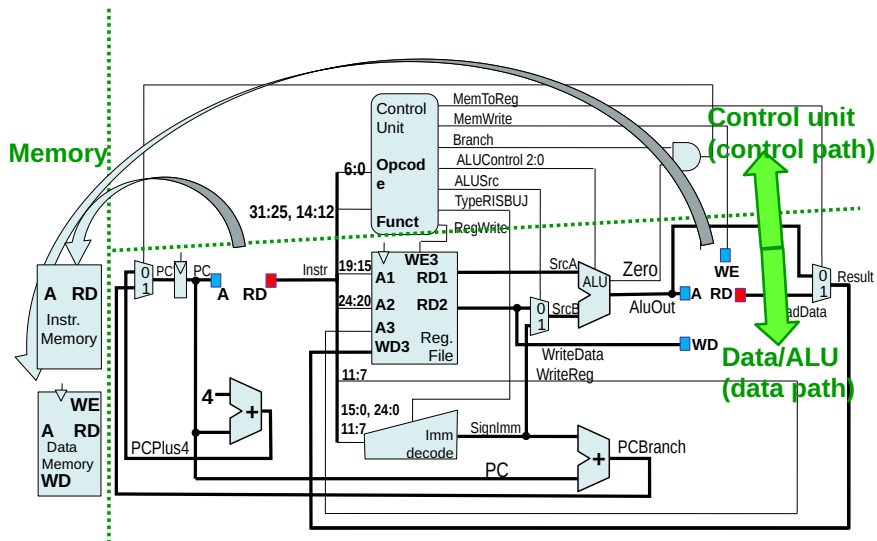
- modem – modems 115.2 kb/s (the first 200 b/s), LTE max 300 Mb/s, 5G to 500 Mb/s
- network/WLAN – from 10 Mb/s to 1 Gb/s to 1 Tb/s
- data storage – HDD, SSD, magneto-tape units, communication speed according to connection (later today), SSD latency best, HDD worse, magneto-tape units – only sequential writing possible

# Classification Peripherals – Continues

Examples of sensors and actuators:

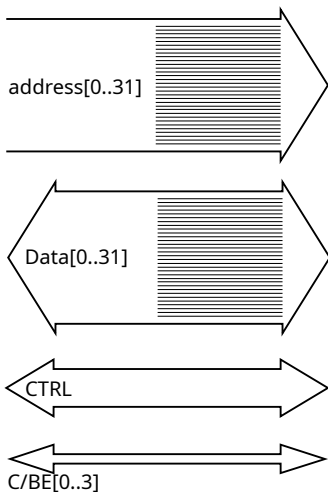
- cameras, laser range finders – communication speed by type of connection
  - USB 2.0 max 480 Mb/s,
  - USB 3.1 max 5 Gb/s,
  - WLAN up to 10 Gb/s
- actuators – DC/PMSM motors
  - transfer speed not so important, but latency
  - latency is the most important parameter for control
  - DC – latency 0.5–0.05 ms,
  - PMSM – latency 0.05–0.01 ms

## CPU Design from Lecture 5





# CPU Connection with Memory and Peripherals



- The address bus (A0..A31) can be separated or multiplexed, or share the same signals as the data part
- Data bus (D0.. D31) can be bidirectional or separated for each direction, parallel or serial
  - Example in the picture – parallel 32-bit bus, the half-duplex data path using same signals for both directions
- Control bus signals
  - It controls the communication on the bus, direction, when transfer starts, ends, if the delay is required
- BE0 to 3 – controls write (even read sometimes) of individual bytes on a bus wider than 8 bits.

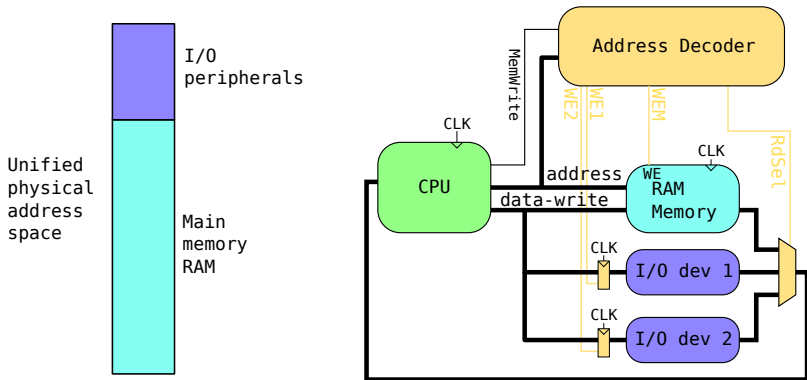
# CPU Peripherals Access

Two different approaches used:

- Special instructions for input/output
  - "x86" uses the instructions `in`, `out`.
  - These instructions are similar to memory access ones, but data are read and written on the bus where peripherals are connected and or with special control signals.
  - The modern peripherals need often block access and larger addressing ranges for which memory access oriented instructions serve better and separated signalling for I/O access only complicates hardware and CPU.
- Part of common (memory) address space reserved for input/output
  - The RISC (including RISC-V) and even lot of CISC CPUs do not have special instructions for communication with peripherals, and therefore use same method and instructions as are used for reading and writing into data memory.
  - This methods is often runtime configurable, the peripherals are mapped into reserved address range that serves to move data between CPU and peripherals.

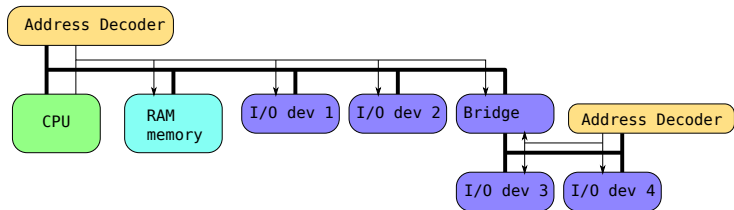
# Memory Mapped Peripherals – RISC-V

- There are no special instructions to access peripherals on RISC-V
- The same instructions are used for peripheral access as for load and store into data memory.
- Address Decoder – controls where are data sent/which device is read

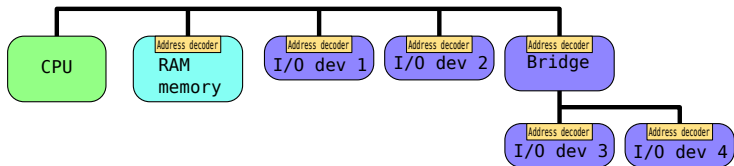


# Address Decoders Realizations

Central decoder (one per system or bus)



Autonomous – peripheral local/decentralized decoders



# Options to Exchange Data and Wait for Peripheral

Software active (busy) polling:

- The device waits for CPU access and sends data to output, or provides already received input
- If the data sending or availability is slower than CPU then CPU has to poll/read device status register (bits data ready/space available)

Interrupt driven/timed access to peripheral:

- If the state changes (data became ready, space is available) hardware signals interrupt (lecture 9)
- This activates interrupt service handler and CPU then reads or writes data under SW control

Peripheral uses direct memory access:

- Uses interrupt for availability signalling as well
- The CPU sets only from/to which address in the memory data will be read/written and the periphery itself controls data transfer
- Peripheral signals by interrupt that all data/packet is ready to be processed by CPU or next one should be prepared by CPU

Another option is direct access from the user application by mapping peripheral into process space – the next topics of today's lecture. Low level kernel driver works similar way.



# System Calls and Services

## System calls:

- system calls are wrapped as regular C functions in libc library functions and offered to user applications – POSIX API
- open function/system call
  - for each periphery can be created handle same as for file
  - this "file" handle is used to communicate with the peripheral
- read function/system call
  - reads the data from the periphery same as data are read from the file
  - blocking operation
    - if no data are available, the function waits for at least one byte or packed arrival
    - the process execution is suspended by operating system and does not block CPU
  - non-blocking operation
    - if no byte/char is available, function return -1 and errno EAGAIN/EWOULDBLOCK
    - the process is responsible for waiting (i.e. by poll or select calls)
    - received data are stored into internal buffers by the driver up to allocated buffers capacity

# Quiz

the scanf function (read formatted input) behavior if data are not currently available to fill/parse into all specified fields:

- A actively repeats call to check wheather data are available
- B the process is suspended and it is necessary to restart it
- C the process is suspended and it is woken u by operating system when data arrives
- D the function returns -1



# Outline

- 1 Input and Output
- 2 QtRvSim Peripherals**
- 3 Internal Interconnection Buses

# QtRvSim – Rotary Knobs and RGB LEDs

- the same data format for RGB LEDs as for reads of the rotary knobs state
  - only bits 24 – 31 are not used for RGB LEDs

Bits	31 ... 27	26	25	24	23 ... 16	15 ... 8	7 ... 0
Meaning	not used	red push.	green push.	blue push.	red value	green value	blue value

- one word sized register on appropriate address for each RGB LED color value store, all three knobs state is read from the single 32-bit word size register/address
- the write of the value to RGB LED register changes its color and intensity to written value immediately
- the read of the register at rotary knobs representing address returns state of the knobs at the current time

# QtRvSim – Rotary Knobs and RGB LEDs

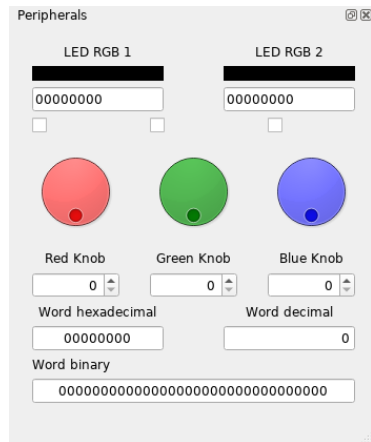
```
# base of SPILED port region
.equ SPILED_REG_BASE,      0xffffc100

# RGB LED 1 - color components, 8 bits each
.equ SPILED_REG_LED_RGB1,  0xffffc110
.equ SPILED_REG_LED_RGB1_o, 0x0010

# RGB LED 2 - color components, 8 bits each
.equ SPILED_REG_LED_RGB2,  0xffffc114
.equ SPILED_REG_LED_RGB2_o, 0x0014

# read 8-bit color component value for each
# knob and knob push states in the MSB
.equ SPILED_REG_KNOBS_8BIT, 0xffffc124
.equ SPILED_REG_KNOBS_8BIT_o, 0x0024

# 32 LEDs - each of 32 bits controls one LED
.equ SPILED_REG_LED_LINE,   0xffffc104
.equ SPILED_REG_LED_LINE_o, 0x0004
```



# Example of Using Rotary Knobs Value to Control RGB

```

# a0 set to provide base for SPILED I/O memory mapped region
li    a0, SPILED_REG_BASE
ori   t2, t2, -1

loop:
# read values from rotary knobs
lw    t0, SPILED_REG_KNOBS_8BIT_o(a0)
# set RGB LED 1 to corresponding color
sw    t0, SPILED_REG_LED_RGB1_o(a0)
xor   t1, t0, t2
# set RGB LED 2 to complementary color
sw    t1, SPILED_REG_LED_RGB2_o(a0)
srli  t0, t0, 24
andi  t0, t0, 4
beq   t0, zero, loop          # repeat until red knob is pressed

ebreak                          # stop/finish the program

```

## Quiz – Rotary Knobs

Choose how to obtain value of the green knob if the 32-bit/word value representing position of the knobs is read from SPILED\_REG\_BASE+SPILED\_REG\_KNOBS\_8BIT\_o register and stored into variable `unsigned int v`;. Available solutions:

- A `((v<<24) & 0x00ff00)`
- B `((v>>8) & 0xff)`
- C `(v & 0x30303030)`
- D `((v>>24) & 0xf0)`

# Asynchronous and Synchronous Buses

## Asynchronous bus:

- two basic variants:
  - The start and end of each bit is detectable by the other side
  - The duration of a single bit is agreed upon and the individual bytes have the start and end detectable by the other side, start of byte/character and or whole frame is denoted by start bit or longer synchronization mark
- An example of asynchronous communication is serial port, USB, SATA drives

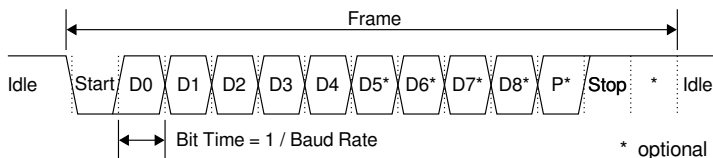
## Synchronous bus:

- The easiest way is to reserve a separate signal to to connect clock signal of transmitter to the receiver
- The data bit or parallel word is synchronized by a clock, either by rising edge or falling edge of clock signal (sometimes by both – DDR)
- An example of synchronous communication is DDR memory, PCI, PCI Express

# Asynchronous Serial Communication

Serial link (serial port) is one of the oldest methods of digital communication used even today.

- Asynchronous transfer without a dedicated clock signal.
- Both sides are set to the same speed, which defines the length of a single bit sent
  - Transfer begins with a start bit sent (starts by transition from 1  $\rightarrow$  0)
    - Sending and receiving a start bit synchronizes the local clock of all devices
  - Then the individual data bits of a single character/byte are sent
  - Data bits can then be followed by parity bit to check for transmission errors
  - Last stop (0) bit sent (followed by 0  $\rightarrow$  1 transition)
- Sending a single byte therefore contains a 10-11 bit sent
- Normal speeds, formerly 9600 Bd to 115200 Bd, now up to 921600 Bd (Bd – Baud = bit per second)



# Serial Line

Basic RS 232 specification:

- Designed to connect two devices only
- Both devices are connected by a signal ground
- 0 represented by +3 – +15V, 1 represented by -3 – -15V
- Full duplex, i.e. separate signals for each transmit direction (Rx and Tx signals crossed between ends)
- Optional handshake signals to stop transmitting when receive buffer of one or other side is getting full

Basic RS 422 specification:

- Differential signals, Rx+, Rx-, Tx+, Tx- – the logical value represented by voltage difference (+/-) between two conductors, can be used up to 1200m distance
- Full duplex, i.e. separate signals for each direction
- Multiple listeners for one transmitter possible

Basic RS 485 specification:

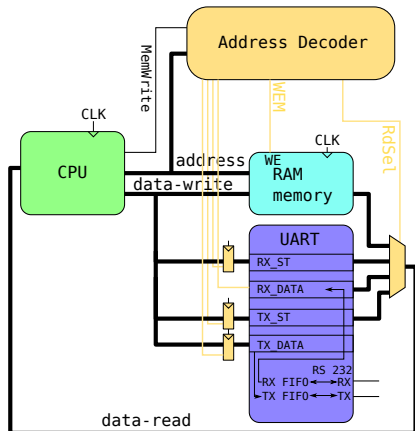
- Differential signaling same as RS 422
- It is half-duplex - i.e. only two conductors, it is necessary disable transmitter output after sending the data and listen for other node answer
- Multiple devices can be interconnected, one initiator and others respond according to the address or multi-master with bus access arbitration



# UART – Universal Asynchronous Receiver-Transmitter

UART – a device to receive and transmit characters/bytes over a serial line

- **RX\_ST** receiver status register
  - bit 0 ready – received data available
- **RX\_DATA** received data register
  - Reading from RX\_DATA removes data from UART FIFO and clears the ready flag if FIFO is empty
- **TX\_ST** transmitter status register
  - bit 0 ready – ready to accept data to transmit
- **TX\_DATA** data to transmit
  - UART starts transmit immediately after write to TX\_DATA



# QtRvSim Serial Port – Terminal

```
.equ SERIAL_PORT_BASE,    0xffffc000
#base address of QtRVSim serial port

.equ SERP_RX_ST_REG,      0xffffc000 #Receiver status register
.equ SERP_RX_ST_REG_o,    0x0000     #Offset of RX_ST_REG
.equ SERP_RX_ST_REG_READY_m, 0x1 #Data byte is ready to be read
.equ SERP_RX_ST_REG_IE_m,   0x2 #Enable Rx ready interrupt

.equ SERP_RX_DATA_REG,    0xffffc004 #Received data byte in 8 LSB bits
.equ SERP_RX_DATA_REG_o,   0x0004     #Offset of RX_DATA_REG

.equ SERP_TX_ST_REG,      0xffffc008 #Transmitter status register
.equ SERP_TX_ST_REG_o,    0x0008     #Offset of TX_ST_REG
.equ SERP_TX_ST_REG_READY_m, 0x1 #Transmitter can accept next byte
.equ SERP_TX_ST_REG_IE_m,   0x2 #Enable Tx ready interrupt

.equ SERP_TX_DATA_REG,    0xffffc00c #Write word to send 8 LSB bits
.equ SERP_TX_DATA_REG_o,   0x000c     #Offset of TX_DATA_REG
```

# QtRvSim – Send Character/Text String Example

write:

```
li    a0, SERIAL_PORT_BASE # a0 set to point o UART mapping base addr.
la    a1, text_1           # setup a1 to point to text start address
```

next\_char:

```
lb    t1, 0(a1)            # load chracter/byte from memory
beq   t1, zero, end_char   # is this null/zero terminating character
addi  a1, a1, 1            # move pointer to next character
```

tx\_busy:

```
lw    t0, SERP_TX_ST_REG_o(a0) # read status of transmitter
andi  t0, t0, SERP_TX_ST_REG_READY_m # mask other bits except READY
beq   t0, zero, tx_busy      # wait/repeat if no space in UART Tx buffer
sw    t1, SERP_TX_DATA_REG_o(a0) # transmitter is ready - write character
j     next_char              # process next character from the string
```

end\_char:

```
ebreak # stop/finish the program
```

.data

text\_1:

```
.asciz "Hello world.\n" # null-character terminated text string
```

# QtRvSim – Character Receive Example

```

gets: li    a0, SERIAL_PORT_BASE # a0 set to point o UART mapping base
      la    a1, text_1           # set a1 to point to start of receive buffer
      addi  t2, zero, 40         # caoacity of the receive buffer

next_char:

rx_not_ready:
      lw    t0, SERP_RX_ST_REG_o(a0)      # load state of the receiver
      andi  t0, t0, SERP_RX_ST_REG_READY_m # mask other bits except READY
      beq   t0, zero, rx_not_ready        # wait/repeat if no character is ready
      lw    t1, SERP_RX_DATA_REG_o(a0)    # read char., it removes it from FIFO
      sb    t1, 0(a1)                    # store character into buffer at a1 address
      addi  t1, t1, -13                 # is this new line character?
      beq   t1, zero, end_char           # if yes, branch out of the loop
      addi  a1, a1, 1                   # move pointer to next/free location
      addi  t2, t2, -1                  # subtract one from available capacity
      bne   t2, zero, next_char          # if there is space still reapar receive

end_char:
      ebreak                           # stop/finish the program
      .data

text_1:
      .skip 40

```

# QtRvSim Terminal – Serial Port

QtRvSim

File Machine Windows Help

1x 2x 5x 10x Unlimited Max

Registers

x0/zero 0x0	x1/ra 0x0	x2/sp 0xbffff00	x3/gp 0x0	x4/tp 0x0	x5/t0 0x1	x6/t1 0x6c	x7/t2 0x0	x8/t0 0x0	x9/t1 0x0
x10/a0 0xffff000	x11/a1 0x238	x12/a2 0x0	x13/a3 0x0	x14/a4 0x0	x15/a5 0x0	x16/a6 0x0	x17/a7 0x0	x18/a2 0x0	x19/a3 0x0
x20/a4 0x0	x21/a5 0x0	x22/a6 0x0	x23/a7 0x0	x24/a8 0x0	x25/a9 0x0	x26/a10 0x0	x27/a11 0x0	x28/t3 0x0	x29/t4 0x0
x30/t5 0x0	x31/t6 0x0	pc 0x228							

Program

Follow fetch

Bp	Address	Instruction
	0x0000200	lui x10, 0xffffc
	0x0000204	addi x10, x10, 0
	0x0000208	addi x11, x0, 564
	0x000020c	lb x6, 0(x11)
	0x0000210	beq x6, x0, 0x22c
	0x0000214	addi x11, x11, 1
	0x0000218	lw x5, 0(x10)
	0x000021c	andi x5, x5, 1
	0x0000220	beq x5, x0, 0x218
	0x0000224	sw x6, 12(x10)
	0x0000228	jal x0, 0x20c
	0x000022c	ebreak
	0x0000230	jal x0, 0x22c
	0x0000234	unknown
	0x0000238	jal x0, 0x7312e
	0x000023c	unknown
	0x0000240	unknown
	0x0000244	unknown
	0x0000248	unknown
	0x000024c	unknown

Core template.S

Diagram showing the internal architecture of the processor, including the Control Unit, Registers, Program Memory, Instruction Memory, ALU, Branch/Shift Unit, and Peripherals (Cache, Memory, Terminal).

Memory

Word	Direct
Address	+0
0xffffc00	0000000b
0xffffc04	00000000
0xffffc08	00000001
0xffffc0c	00000000
0xffffc10	00000000
0xffffc14	00000000
0xffffc18	00000000
0xffffc1c	00000000
0xffffc20	00000000
0xffffc24	00000000
0xffffc28	00000000
0xffffc2c	00000000
0xffffc30	00000000
0xffffc34	00000000
0xffffc38	00000000
0xffffc3c	00000000
0xffffc40	00000000
0xffffc44	00000000
0xffffc48	00000000
0xffffc4c	00000000
0xffffc50	00000000
0xffffc54	00000000
0xffffc58	00000000
0xffffc5c	00000000
0xffffc60	00000000
0xffffc64	00000000
0xffffc68	00000000
0xffffc6c	00000000
0xffffc70	00000000
0xffffc74	00000000
0xffffc78	00000000
0xffffc7c	00000000
0xffffc80	00000000
0xffffc84	00000000
0xffffc88	00000000
0xffffc8c	00000000
0xffffc90	00000000
0xffffc94	00000000
0xffffc98	00000000
0xffffc9c	00000000
0xffffca0	00000000
0xffffca4	00000000
0xffffca8	00000000
0xffffcac	00000000
0xffffcad	00000000
0xffffcae	00000000
0xffffcaf	00000000
0xffffcb0	00000000
0xffffcb4	00000000
0xffffcb8	00000000
0xffffcbc	00000000
0xffffcbd	00000000
0xffffcbe	00000000
0xffffcbf	00000000
0xffffcc0	00000000
0xffffcc4	00000000
0xffffcc8	00000000
0xffffccc	00000000
0xffffccd	00000000
0xffffcce	00000000
0xffffccf	00000000
0xffffcd0	00000000
0xffffcd4	00000000
0xffffcd8	00000000
0xffffcdc	00000000
0xffffcde	00000000
0xffffcdf	00000000
0xffffce0	00000000
0xffffce4	00000000
0xffffce8	00000000
0xffffcec	00000000
0xffffced	00000000
0xffffcee	00000000
0xffffcef	00000000
0xffffcf0	00000000
0xffffcf4	00000000
0xffffcf8	00000000
0xffffcfc	00000000
0xffffcfd	00000000
0xffffcfe	00000000
0xffffcff	00000000
0xffffd00	00000000
0xffffd04	00000000
0xffffd08	00000000
0xffffd0c	00000000
0xffffd0d	00000000
0xffffd0e	00000000
0xffffd0f	00000000
0xffffd10	00000000
0xffffd14	00000000
0xffffd18	00000000
0xffffd1c	00000000
0xffffd1d	00000000
0xffffd1e	00000000
0xffffd1f	00000000
0xffffd20	00000000
0xffffd24	00000000
0xffffd28	00000000
0xffffd2c	00000000
0xffffd2d	00000000
0xffffd2e	00000000
0xffffd2f	00000000
0xffffd30	00000000
0xffffd34	00000000
0xffffd38	00000000
0xffffd3c	00000000
0xffffd3d	00000000
0xffffd3e	00000000
0xffffd3f	00000000
0xffffd40	00000000
0xffffd44	00000000
0xffffd48	00000000
0xffffd4c	00000000
0xffffd4d	00000000
0xffffd4e	00000000
0xffffd4f	00000000
0xffffd50	00000000
0xffffd54	00000000
0xffffd58	00000000
0xffffd5c	00000000
0xffffd5d	00000000
0xffffd5e	00000000
0xffffd5f	00000000
0xffffd60	00000000
0xffffd64	00000000
0xffffd68	00000000
0xffffd6c	00000000
0xffffd6d	00000000
0xffffd6e	00000000
0xffffd6f	00000000
0xffffd70	00000000
0xffffd74	00000000
0xffffd78	00000000
0xffffd7c	00000000
0xffffd7d	00000000
0xffffd7e	00000000
0xffffd7f	00000000
0xffffd80	00000000
0xffffd84	00000000
0xffffd88	00000000
0xffffd8c	00000000
0xffffd8d	00000000
0xffffd8e	00000000
0xffffd8f	00000000
0xffffd90	00000000
0xffffd94	00000000
0xffffd98	00000000
0xffffd9c	00000000
0xffffd9d	00000000
0xffffd9e	00000000
0xffffd9f	00000000
0xffffda0	00000000
0xffffda4	00000000
0xffffda8	00000000
0xffffdac	00000000
0xffffdad	00000000
0xffffdae	00000000
0xffffdaf	00000000
0xffffdb0	00000000
0xffffdb4	00000000
0xffffdb8	00000000
0xffffdbc	00000000
0xffffdbd	00000000
0xffffdbe	00000000
0xffffdbf	00000000
0xffffdc0	00000000
0xffffdc4	00000000
0xffffdc8	00000000
0xffffdcc	00000000
0xffffdcd	00000000
0xffffdce	00000000
0xffffdcf	00000000
0xffffdd0	00000000
0xffffdd4	00000000
0xffffdd8	00000000
0xffffddc	00000000
0xffffddd	00000000
0xffffdde	00000000
0xffffddf	00000000
0xffffde0	00000000
0xffffde4	00000000
0xffffde8	00000000
0xffffdec	00000000
0xffffded	00000000
0xffffdee	00000000
0xffffdef	00000000
0xffffdf0	00000000
0xffffdf4	00000000
0xffffdf8	00000000
0xffffdfc	00000000
0xffffdfd	00000000
0xffffdfe	00000000
0xffffdff	00000000
0xffffe00	00000000
0xffffe04	00000000
0xffffe08	00000000
0xffffe0c	00000000
0xffffe0d	00000000
0xffffe0e	00000000
0xffffe0f	00000000
0xffffe10	00000000
0xffffe14	00000000
0xffffe18	00000000
0xffffe1c	00000000
0xffffe1d	00000000
0xffffe1e	00000000
0xffffe1f	00000000
0xffffe20	00000000
0xffffe24	00000000
0xffffe28	00000000
0xffffe2c	00000000
0xffffe2d	00000000
0xffffe2e	00000000
0xffffe2f	00000000
0xffffe30	00000000
0xffffe34	00000000
0xffffe38	00000000
0xffffe3c	00000000
0xffffe3d	00000000
0xffffe3e	00000000
0xffffe3f	00000000
0xffffe40	00000000
0xffffe44	00000000
0xffffe48	00000000
0xffffe4c	00000000
0xffffe4d	00000000
0xffffe4e	00000000
0xffffe4f	00000000
0xffffe50	00000000
0xffffe54	00000000
0xffffe58	00000000
0xffffe5c	00000000
0xffffe5d	00000000
0xffffe5e	00000000
0xffffe5f	00000000
0xffffe60	00000000
0xffffe64	00000000
0xffffe68	00000000
0xffffe6c	00000000
0xffffe6d	00000000
0xffffe6e	00000000
0xffffe6f	00000000
0xffffe70	00000000
0xffffe74	00000000
0xffffe78	00000000
0xffffe7c	00000000
0xffffe7d	00000000
0xffffe7e	00000000
0xffffe7f	00000000
0xffffe80	00000000
0xffffe84	00000000
0xffffe88	00000000
0xffffe8c	00000000
0xffffe8d	00000000
0xffffe8e	00000000
0xffffe8f	00000000
0xffffe90	00000000
0xffffe94	00000000
0xffffe98	00000000
0xffffe9c	00000000
0xffffe9d	00000000
0xffffe9e	00000000
0xffffe9f	00000000
0xffffea0	00000000
0xffffea4	00000000
0xffffea8	00000000
0xffffeac	00000000
0xffffead	00000000
0xffffeae	00000000
0xffffeaf	00000000
0xffffeb0	00000000
0xffffeb4	00000000
0xffffeb8	00000000
0xffffebc	00000000
0xffffebd	00000000
0xffffebe	00000000
0xffffebf	00000000
0xffffec0	00000000
0xffffec4	00000000
0xffffec8	00000000
0xffffecc	00000000
0xffffecd	00000000
0xffffece	00000000
0xffffecf	00000000
0xffffed0	00000000
0xffffed4	00000000
0xffffed8	00000000
0xffffedc	00000000
0xffffedd	00000000
0xffffede	00000000
0xffffedf	00000000
0xffffee0	00000000
0xffffee4	00000000
0xffffee8	00000000
0xffffeec	00000000
0xffffeed	00000000
0xffffeee	00000000
0xffffeef	00000000
0xffffef0	00000000
0xffffef4	00000000
0xffffef8	00000000
0xffffefc	00000000
0xffffefd	00000000
0xffffefe	00000000
0xffffeff	00000000
0xfffff00	00000000
0xfffff04	00000000
0xfffff08	00000000
0xfffff0c	00000000
0xfffff0d	00000000
0xfffff0e	00000000
0xfffff0f	00000000
0xfffff10	00000000
0xfffff14	00000000
0xfffff18	00000000
0xfffff1c	00000000
0xfffff1d	00000000
0xfffff1e	00000000
0xfffff1f	00000000
0xfffff20	00000000
0xfffff24	00000000
0xfffff28	00000000
0xfffff2c	00000000
0xfffff2d	00000000
0xfffff2e	00000000
0xfffff2f	00000000
0xfffff30	00000000
0xfffff34	00000000
0xfffff38	00000000
0xfffff3c	00000000
0xfffff3d	00000000
0xfffff3e	00000000
0xfffff3f	00000000
0xfffff40	00000000
0xfffff44	00000000
0xfffff48	00000000
0xfffff4c	00000000
0xfffff4d	00000000
0xfffff4e	00000000
0xfffff4f	00

# QtRvSim Terminal – Serial Port

Pipelined processor – peripheral access takes place in MEM stage/phase

The screenshot displays the QtRvSim software interface, which is used for simulating a pipelined processor. The interface is divided into several panels:

- Registers:** A table showing the current state of various registers. For example, `x0/zero` is `0x0`, `x1/ra` is `0x0`, and `pc` is `0x230`.
- Program:** A list of assembly instructions with their addresses. The current instruction being executed is `jal x0, 0x22c` at address `0x0000230`.
- Core:** A block diagram of the processor core, showing the flow of data and control signals between various components like the ALU, Registers, and Memory.
- Memory:** A window showing the memory contents at various addresses. The current address is `0x0000000`.
- Terminal:** A window for serial port communication, currently showing the text "Hell".

The processor is currently in the MEM stage/phase, as indicated by the highlighted instruction `jal x0, 0x22c` in the Program window.

# Peripheral Access Summary

- the above method of communication with busy waiting is called polling
  - the program constantly asks if something has changed, a character has been received/available or there is space in transmit queue
  - this is very inefficient, it wastes CPU time, which could be doing something useful
- in lecture 9 we will introduce interrupts as method to notify CPU by peripherals
  - the program can do something else, the interrupt occurs if it is enabled and the state of the peripheral changes
  - when an interrupt occurs another program/handler function starts to execute, which checks peripheral state to find which event has happened and processes it
  - information about what happened and corresponding data are passed to the program using synchronization mechanism implemented by operating system (will be discussed in detail in the OSY subject)

# Outline

- 1 Input and Output
- 2 QtRvSim Peripherals
- 3 Internal Interconnection Buses**

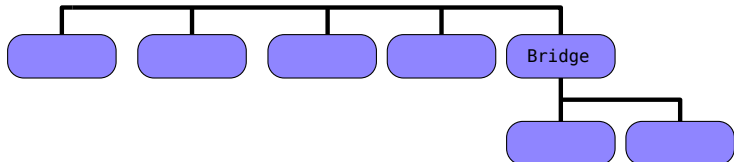


# A Brief History of Internal Buses in Personal Computers

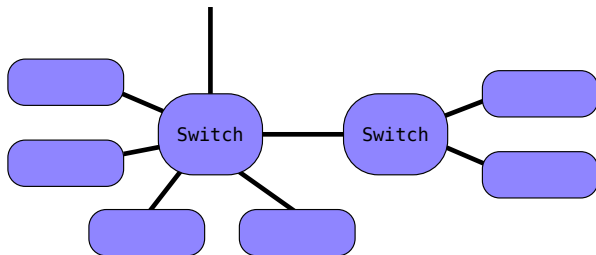
- ISA – an older type of passive bus, 8 or 16 bits wide, maximum transfer rate of 8 MB/s
- PCI – a newer type of “smart” bus, 32 or 64 bits wide, burst mode, transfer rate of up to 530 MB/s, topological enumeration, Plug and Play support and programmable mapping of devices into I/O and memory address space
- AGP – a dedicated bus designed to connect a graphics card via the northbridge to the CPU, transfer rate of 260 MB/s – 2 GB/s
- PCI-Express (PCIe) – a new serial implementation of the PCI bus

# Bus Topology

Shared bus (PCI for example) – data/address/control signals to multiple card slots



Peer-to-peer connection using a switches/hubs (e.g. PCIe, USB)



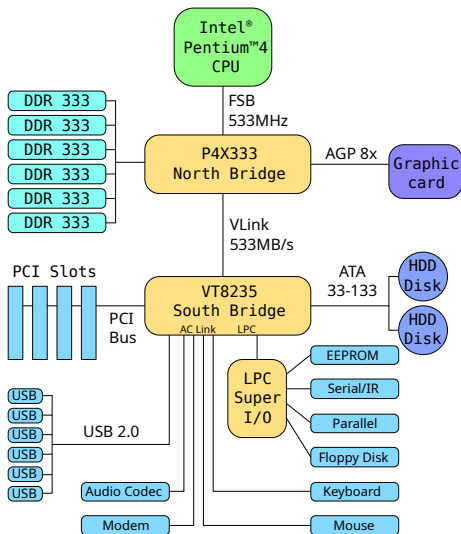
# Buses in an Older PC Computers

## Old Pentium 4 architecture (1990s)

The northbridge is connected directly to the CPU and the fastest peripherals – memory and graphics card

The southbridge communicates with the northbridge and integrates or connects network cards, HDD, PCI slots.

The slowest peripherals like Floppy Disk, or serial and parallel ports (printers) are usually connected via other bridges.



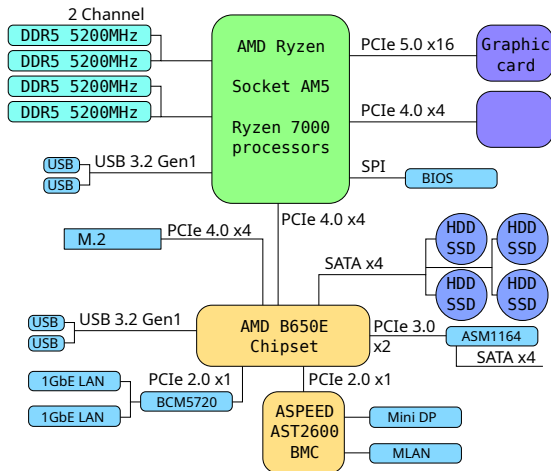
# Buses in a Newer PC Computers

Modern with memory controllers on processor chip (package).

The northbridge has become part of the processor.

The southbridge communicates directly with the processor.

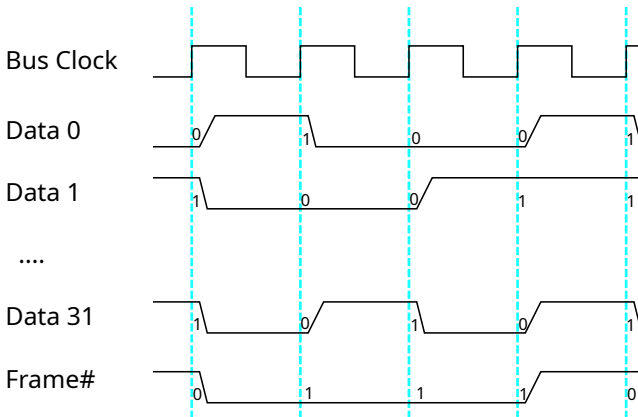
Most peripherals are connected via PCI-Express and USB.



# Peripheral Component Interconnect Standard Bus – PCI

The all state changes and signal strobes are controlled by clock edges. For proper operation, it needs to be synchronized as precisely as possible to the transmitted clock.

Signals marked with # are negated because the falling edge is faster.



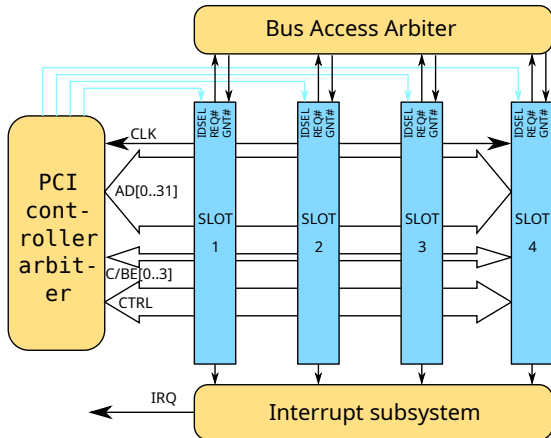
# PCI (Original Parallel) Bus Architecture

The card slot specific IDSEL signal is only for initialization, to find out what device is connected in which slot.

AD is the 32 (64 for PCI-x) signals used for multiplexed address and data

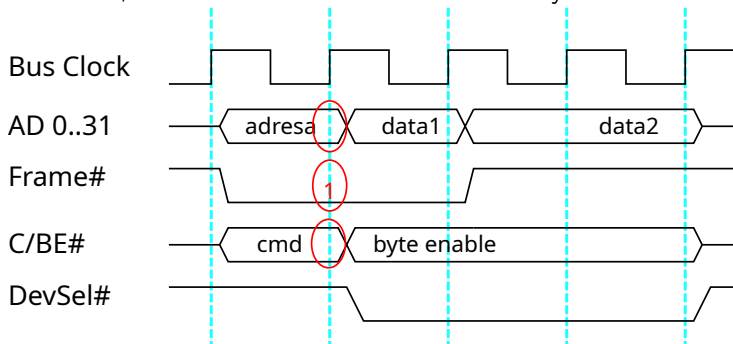
C/BE signals provide 4 command (transfer type) and byte enable signals

CTRL are signals for bus transaction control (i.e. FRAME)



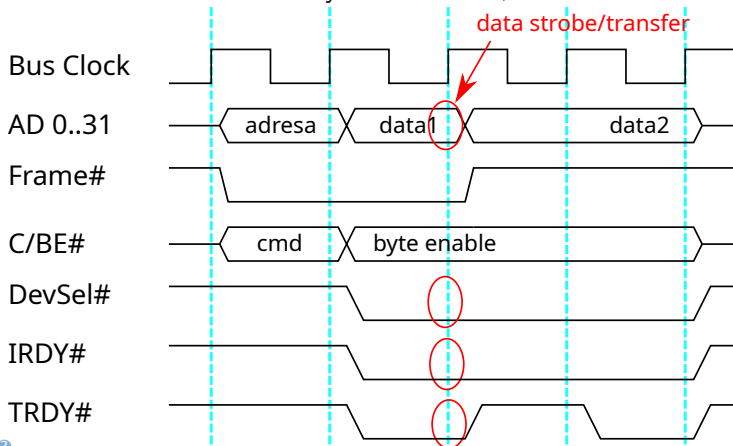
# PCI Data Transfers – Write Transaction

- The initiator begins the transfer by request for bus control to arbiter
  - If multiple devices request the bus at the same time, the arbiter must queue their requests and allow only one transfer at a time
- The initiator begins the transmission by setting the address of the target peripheral register on the AD bus and asserting (active low) the FRAME signal; the first clock rising edge address is strobed, next is data, the last data transfer is denoted by the deassertion of FRAME



# PCI Data Transfers – Write Transaction – Data

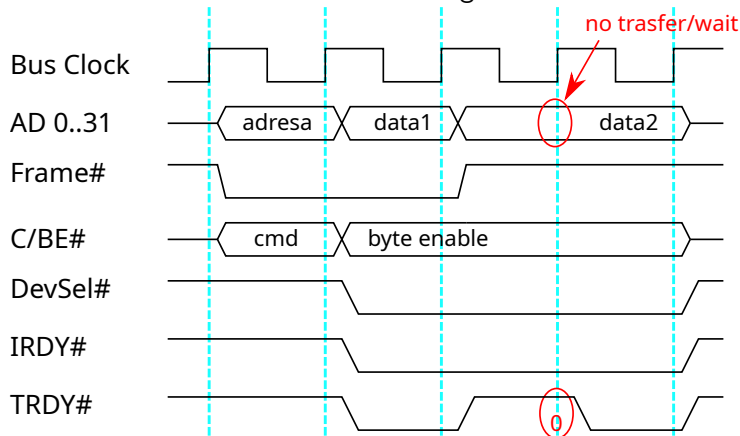
- A peripheral that recognizes its address asserts DevSel
- If the target peripheral (Target) is ready to receive data, it asserts TRDY.
- If the initiator is ready to send data, it asserts IRDY.





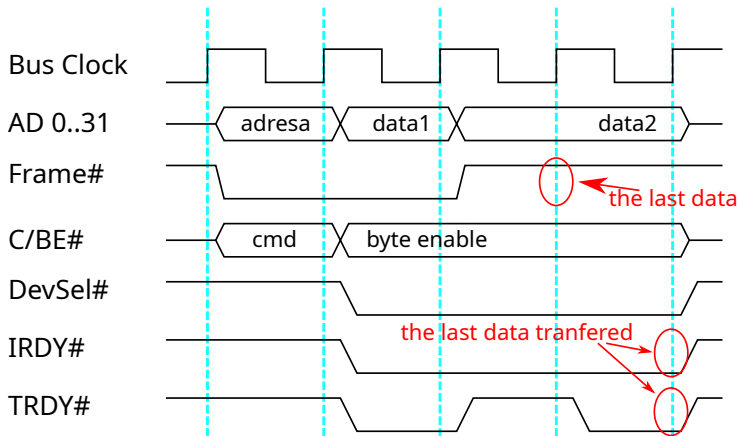
# PCI Data Transfers – Write Transaction – Wait

- If the target peripheral is not ready, it deasserts TRDY
- If the initiator is not ready to put data on the bus, it deasserts IRDY
- If TRDY or IRDY is not asserted, then the data transfer is suspended – wait for state at next clock edge



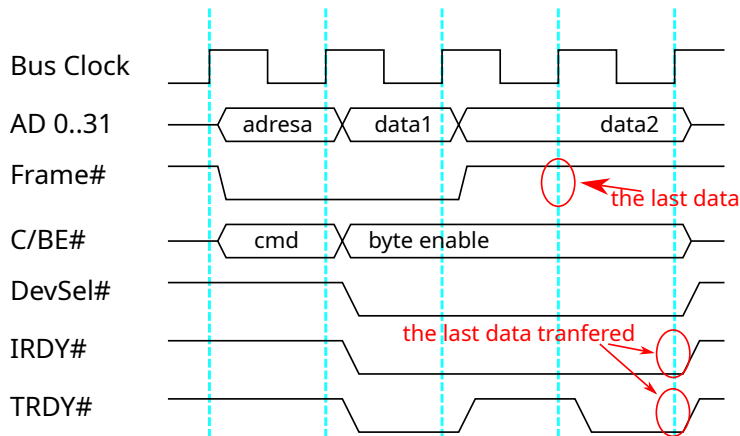
# PCI Data Transfers – Write Transaction – the Last Data

- Deasserts the FRAME signal to inform that the last data will be sent
- In the shown case, the data transfer was suspended, so the transfer of the last data is postponed to the next clock cycle.



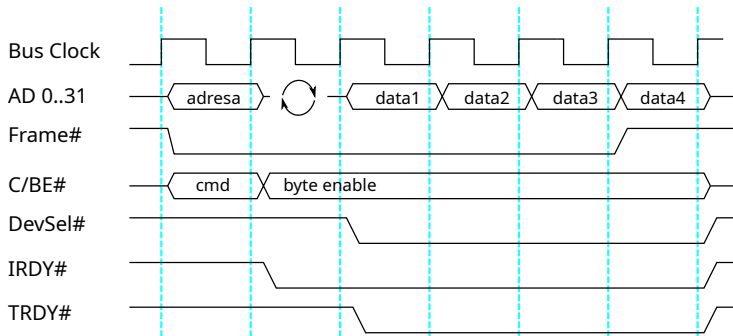
# PCI Data Transfers – Write Transaction – Release Bus

- After the transfer is completed (the last data sent and accepted), the IRDY#, TRDY# and DEVSEL signals are deasserted and the bus is released for the next transfer.



# PCI Data Transfers – Read Transaction

- The initiator requests data from the target peripheral.
- Data transfer is similar, but cannot start on the next clock cycle because the initiator must disconnect from the AD bus and the target device must connect its output buffer to the bus.



# Classical Parallel PCI Bus – Summary

Disadvantages of the PCI bus:

- Half-duplex data cannot be sent in both directions at the same time, data transferred in only one direction at time
- Multiple devices on the shared bus – slow peripherals slow down fast peripherals, increases the latency of all other peripherals
- PCI bus only allows clocks with 33 MHz, or 66 MHz
  - This corresponds to 132 MB/s or 264 MB/s for the 32-bit variant
  - This corresponds to 264 MB/s or 528 MB/s for the 64-bit variant
- PCI eXtended (PCI-X) bus allows clocks up to 133 MHz and later a maximum of 533 MHz
  - This corresponds to transfer speeds of 532MB/s to a maximum of 4266 MB/s for the 64-bit variant variant, very hard to route on PCB
  - PCI-X version 2.0 with speeds above 133MHz were not very widespread
- The connector for the 32-bit version has 62 pins – i.e. 124 signals, for the 64-bit version it is even 188 signals

# PCI Express – PCIe

The main disadvantage of parallel busses is the required precise mutual matching of the signals delays and routing:

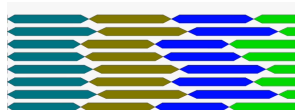
- Even small inaccuracies in the length of the conductors and the quality of the connections lead to different propagation speed/delay of the electrical signal
- No problem for low frequencies but even small mutual and or clocks timing shift prevents consistent data strobing over multiple wires for high ones.
- It is demonstrated in animation at <https://cw.fel.cvut.cz/wiki/courses/b35apo/en/lectures/07/start>

frequency  $f$



transfer without  
problems

frequency  $2 \cdot f$



transfer at reliability  
limit

frequency  $4 \cdot f$



no instant to strobe  
parallel data by receiver

# PCI-Express (PCIe) – Upgrades to Parallel PCI

- PCIe is peer-to-peer – signals are only routed between two devices.
- PCIe is full-duplex – data can be transferred in both directions at the same time.
- For one-way transmission, a serial method with differential signal pair (per lane) is used, rejects common mode voltage shifts and noises
  - This method of transmission is less susceptible to interference than a single ended wire to ground.
- PCIe can contain multiple links, but the transmission between the links is not synchronized at the bit level.
- In the simplest version, PCIe connectors have only 18 pins, 36 signals, of which 18 are ground and power.

# PCIe Serial Transmission

- PCIe can use different speeds for transmission
- It is necessary that the receiving side can detect the transmission speed.
- The problem is that if a byte contains only 0s or only 1s, the signal does not change.
- The solution is to encode a byte (8 bits) into 10 bits so that the total number of 0s and 1s transmitted is the same.

Quiz: How many different 10-bit numbers are there that have five 0s and five 1s?

- A  $2^5 \cdot 2^5 = 64$
- B  $5! \cdot 5! = 14400$
- C  $\binom{10}{5} = 252$
- D  $5! + 5! = 240$



# PCIe 8b/10b Encoding

- 8 bits, or 256 different values, are encoded into a 10-bit number that has at least four 0s and at least four 1s
  - This extends to  $\binom{10}{5} + 2 \cdot \binom{10}{6} = 672$  of such 10-bit numbers
  - We choose those codes where there are more  $1 \rightarrow 0$  and  $0 \rightarrow 1$  transitions.
- For codes where count of 0s and 1s differs (by one only), there is freedom whether code with more 1s or matching complement with more 0s is chosen

table to code 3b by 4b

Input		RD = -1	RD = +1
Code	HGF	f g h j	
D.x.0	000	1011	0100
D.x.1	001	1001	
D.x.2	010	0101	
D.x.A3	011	1100	
D.x.B3		0011	
D.x.4	100	1101	0010
D.x.5	101	1010	
D.x.6	110	0110	
D.x.P7	111	1110	0001
D.x.A7		0111	1000

# PCIe Versions 1.x and 2.x

## Ver 1.x

- The transfer rate is 2.5 GT/s (transfers per second, number of symbols per second on one lane)
- 10 transfers are required for one byte of 8 bits
- The maximum bandwidth (transfer capacity) is therefore  $250 \text{ MB/s} = (2500 \cdot \frac{8}{10}) \text{ Mb/s} = (2500 \cdot \frac{1}{10}) \text{ MB/s}$ , practical with headers  
200 MB/s per lane
- PCIe allows up to 16 independent links (lanes) for one peripheral connection, data bytes transferred independently in parallel
  - The maximum bandwidth is  $(16 \cdot 250) \text{ MB/s} = 4 \text{ GB/s}$

## Ver 2.x

- The transfer rate is 5 GT/s (transfers per second)
- 10 transfers are required for one byte of 8 bits
- The maximum bandwidth of one line (x1) is 500 MB/s
- The maximum bandwidth for 16 links (x16) is  $(16 \cdot 500) \text{ MB/s} = 8 \text{ GB/s}$

# PCIe Version. 3.x, 4.x and 5.x

8b/10b encoding is unnecessarily inefficient, 128b/130b encoding with similar parameters was chosen.

## ■ Ver 3.x

- The transfer rate is 8 GT/s (transfers per second)
- The maximum transfer capacity is therefore almost  $985 \text{ MB/s} = (8000 \cdot \frac{128}{130}) \text{ Mb/s} = (8000 \cdot \frac{16}{130}) \text{ MB/s}$
- The maximum bandwidth for x16 is  $(16 \cdot 985) \text{ MB/s} = 15.75 \text{ GB/s}$

## ■ Ver 4.x

- The transfer rate is 16 GT/s (transfers per second)
- The maximum bandwidth is therefore almost  $1.97 \text{ GB/s} = (16000 \cdot \frac{16}{130}) \text{ MB/s}$
- The maximum bandwidth for x16 is  $(16 \cdot 1.97) \text{ GB/s} = 31.5 \text{ GB/s}$

## ■ Ver 5.x

- The transfer rate is 32 GT/s (transfers per second)
- The maximum bandwidth is therefore almost  $3.94 \text{ GB/s} = (32000 \cdot \frac{16}{130}) \text{ MB/s}$
- The maximum bandwidth for x16 is  $(16 \cdot 3.94) \text{ GB/s} = 63 \text{ GB/s}$

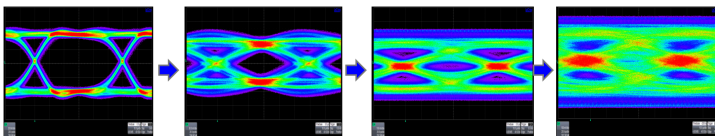
# PCIe Topology

Communication over the PCIe bus is similar to communication over a switched network (i.e. Ethernet).

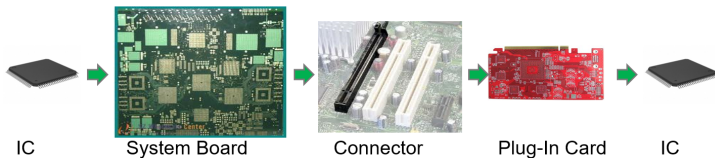
- Communication takes place in packets
  - ATTENTION – packet overhead is not included in the maximum transmission capacity.
  - Each packet has a synchronization header, address, data, crc – similar to the Ethernet protocol.
- The use of switches is similar to that in a network
  - Switches allow direct communication only between two devices
  - Switches can prioritize packets – advantageous for reducing latency (using packets, on the other hand, increases latency)
  - Switches can be used to ensure automatic detection and configuration of connected devices on a similar principle to that of the PCI bus

# The Reality of Serial Bus Signals

High-speed communication presents many different problems.  
Signal Appearance over Distance



Signal degrades over long transmission path and connectors



# Hard Drives and SSD Storage

- A similar development to the change from PCI to PCIe can be observed in drives.
- PATA or Parallel ATA is a parallel drive connection since 1984 for the first IBM PC/AT
  - The name ATA actually stands for AT Attachment, AT is an abbreviation for Advance Technology.
  - Also referred to as IDE, later Extended IDE (EIDE) Ultra ATA (UATA)
  - PATA is a 16-bit parallel data transfer between the CPU and the drive
  - In its fastest version, it could transfer up to 133 MB/s
- SATA is a serial version of disk communication.
  - In the minimum version, it only needs 7 wires, A+, A-, B+, B- and 3x ground.
    - SATA 1.0: 150MB/s (PATA:130MB/s)
    - SATA 2.0: 300 MB/s
    - SATA 3.0: 600 MB/s
    - SATA 3.2: about 2 GB/s