B35APO: Computer Architectures

Lecture 07. Input and Output

Pavel Píša
pisa@fel.cvut.cz

Petr Štěpán
stepan@fel.cvut.cz

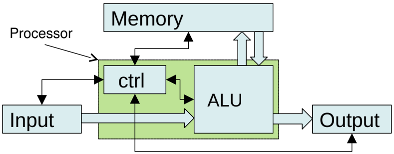
17. června, 2025

Outline

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/badwidth 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

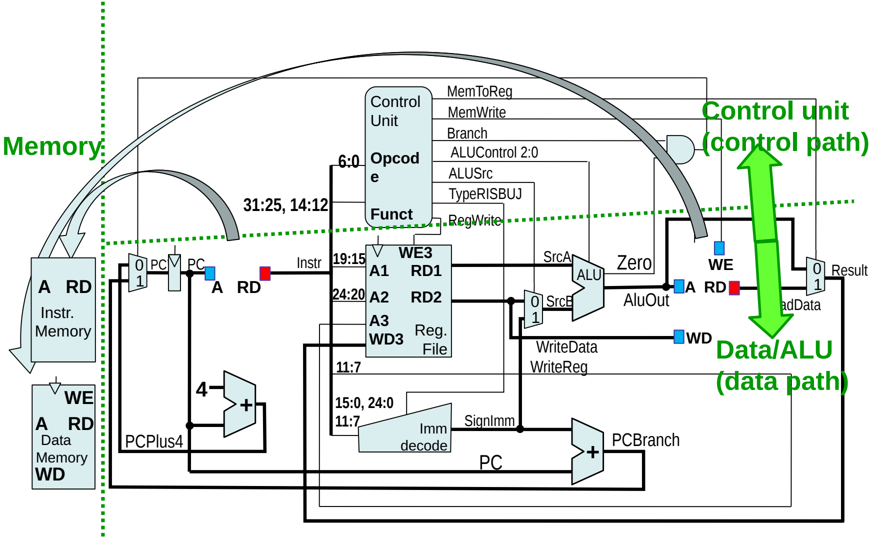
- modem – modems 115.2 kb/s (the first 200 b/s), LTE max 300 Mb/s, 5G to 500 Mb/s
- network/WLAN – from 10 Mb/s to 1 Gb/s to 1 Tb/s
- data storage – HDD, SSD, magneto-tape units, communication speed according to connection (later today), SSD latency best, HDD worse, magneto-tape units – only sequential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

Classification Peripherals – Continues

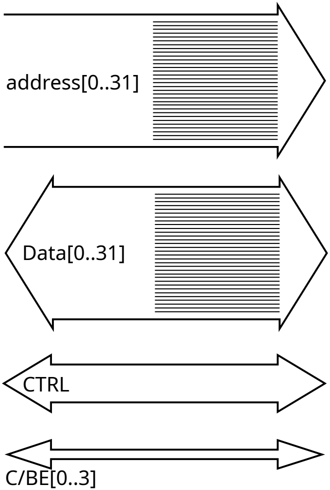
Examples of sensors and actuators:

- cameras, laser rage 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.

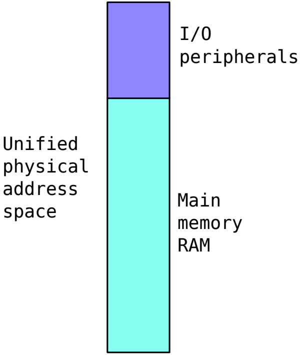
CPU Peripherals Access

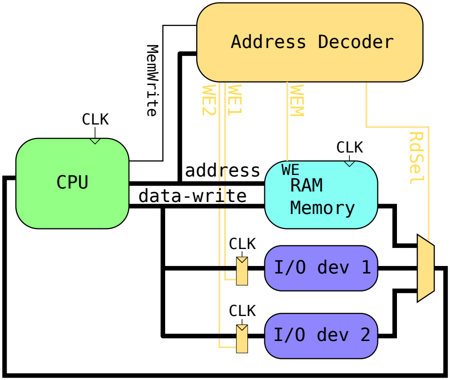
Two different approaches used:

- 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

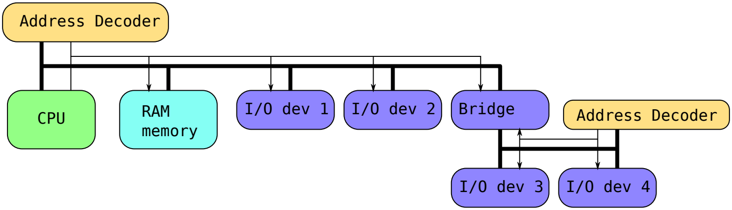
- There are no special instructions to access peripherals on RISC-V
- The same instruction are used for peripheral access as for load and store into data memory.
- Address Decoder – controls where are data sent/which device is read



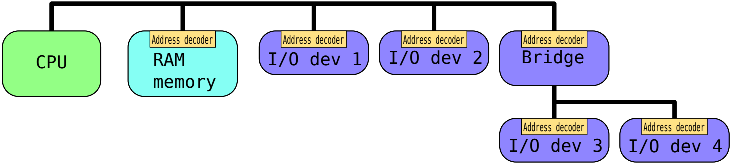


Address Decoders Realizations

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

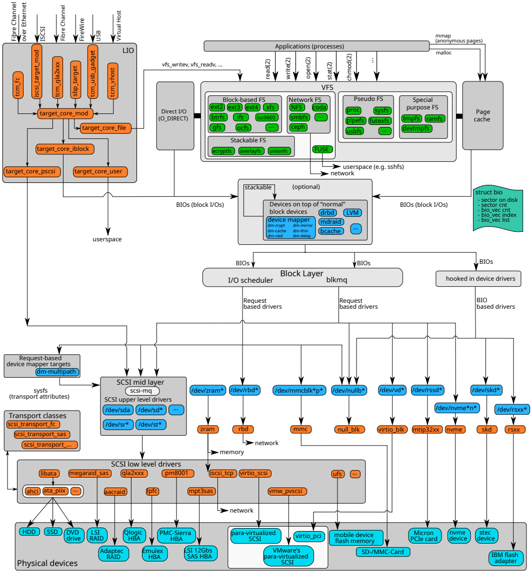
Peripheral uses direct memory access:

- 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

Input/Output and Drivers in Linux Kernel (simplified)

The programs communicate with the peripherals using the operating system and system call and periphery drivers (overview in lecture 10, detailed in the OSY course – Operating Systems).

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 errrno 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

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

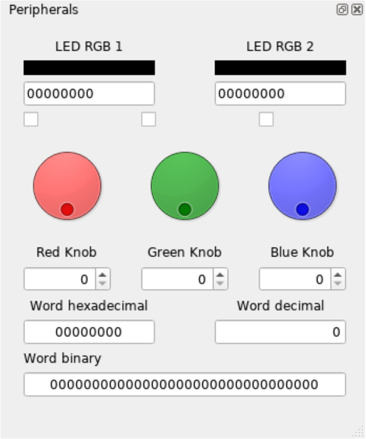
```
# base of SPILED port region
.equ SPILED_REG_BASE,       0xffffc100

# 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

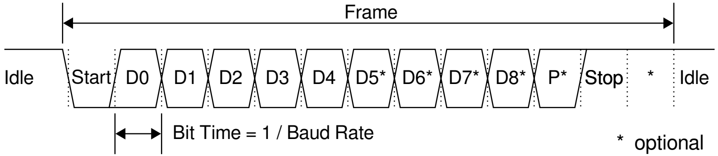
Synchronous bus:

- 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 $1 \to 0$)
    - 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 (folowed by $0 \to 1$ 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 represnetd 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

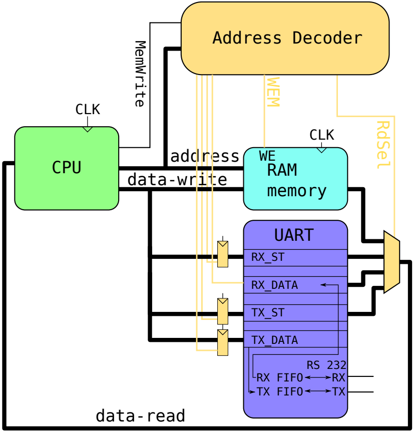
Basic RS 485 specification:

- Diferential 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 othe 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-Transmiter

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 transmiter status register
  - bit 0 ready – ready to accept data to transmit
- TX_DATA data to transmit
  - UART starts transmit imediatelly 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

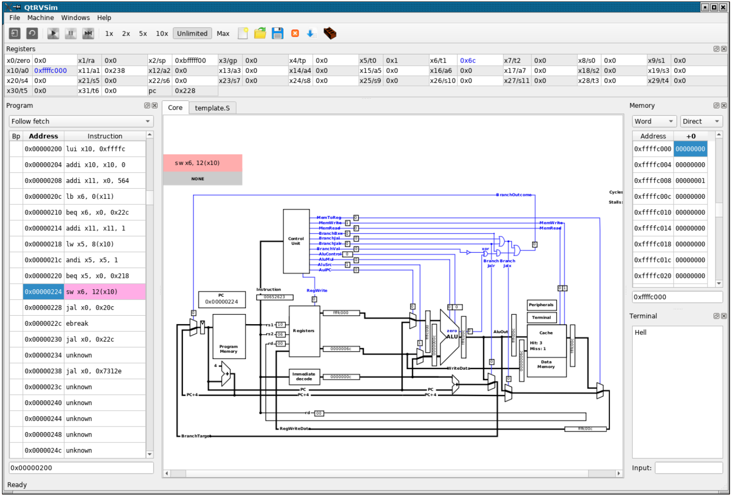
```
write:
    li   a0, SERIAL_PORT_BASE # a0 set to point o UART mapping base addr.
    la   a1, text_1           # setup a1 to point to text start address
next_char:
    lb   t1, 0(a1)            # load chracter/byte from memory
    beq  t1, zero, end_char   # is this null/zero terminating character
    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) # tranmitter 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

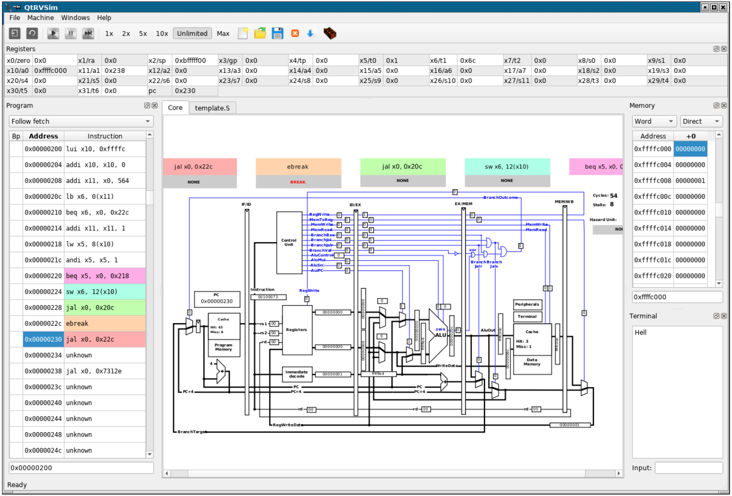
```
gets: li   a0, SERIAL_PORT_BASE # a0 set to point o UART mapping base
    la   a1, text_1             # set a1 to point to start of receive buffer
    addi t2, zero, 40           # caoacity of the receive buffer
next_char:
rx_not_ready:
    lw   t0, SERP_RX_ST_REG_o(a0)         # load state of the 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 repear receive
end_char:
    ebreak                      # stop/finish the program
    .data
text_1:
    .skip 40
```

QtRvSim Terminal – Serial Port



QtRvSim Terminal – Serial Port

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



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 wates CPU tie, 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 peripheal 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)

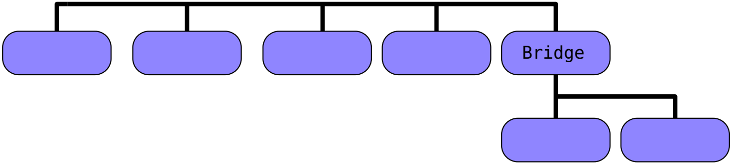
Outline

A Brief History of Internal Buses in Personal Computers

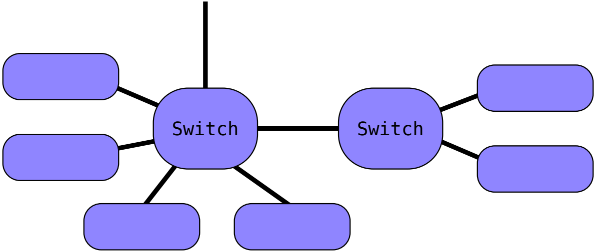
- ISA – an older type of passive bus, 8 or 16 bits wide, maximum transfer rate of 8 MB/s
- PCI – a newer type of “smart” bus, 32 or 64 bits wide, burst mode, transfer rate of up to 530 MB/s, topological enumeration, Plug and Play support and programmable mapping of devices into I/O and memory address 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)



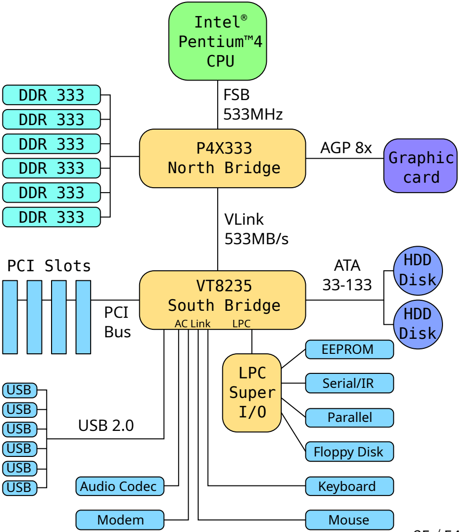
Buses in an Older PC Computers

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.



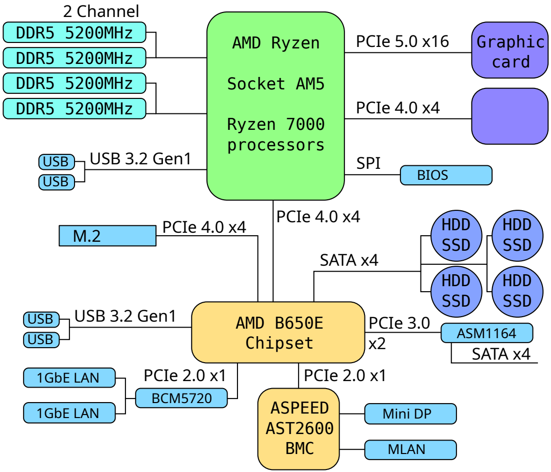
Buses in a Newer PC Computers

Modern with memory controllers on processor chip (package).

The northbridge has become part of the processor.

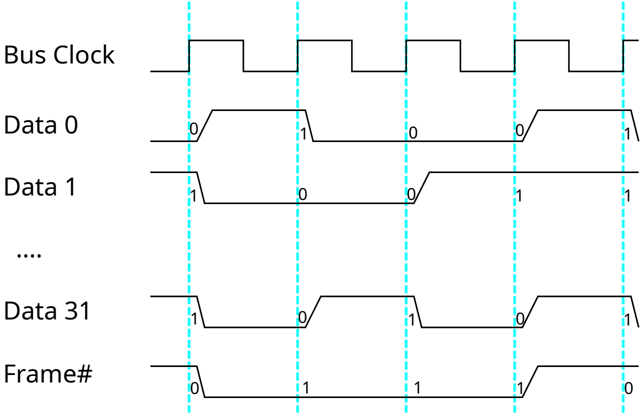
The southbridge communicates directly with the processor.

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.



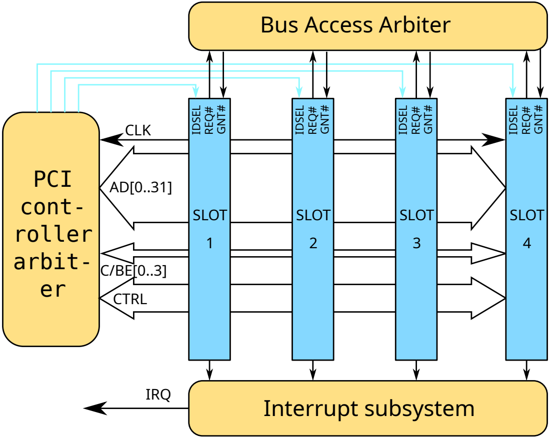
PCI (Original Parallel) Bus Architecture

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

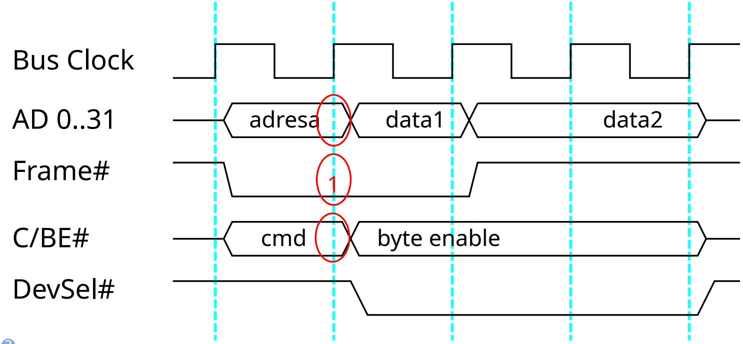
C/BE signals provide 4 command (transfer type) and byte enable signals

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



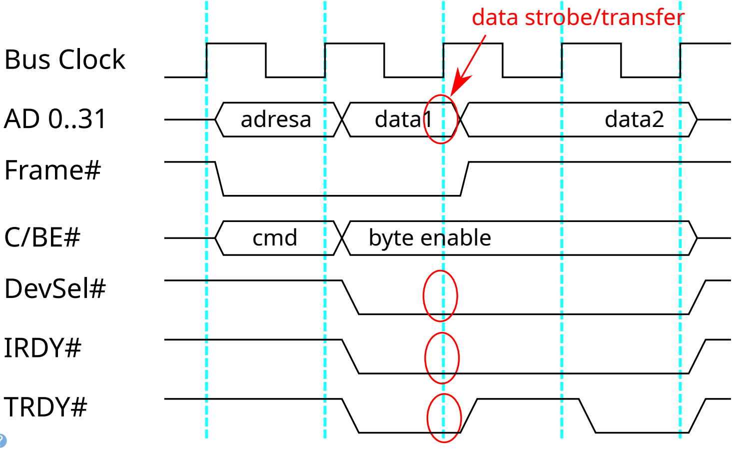
PCI Data Transfers – Write Transaction

- 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



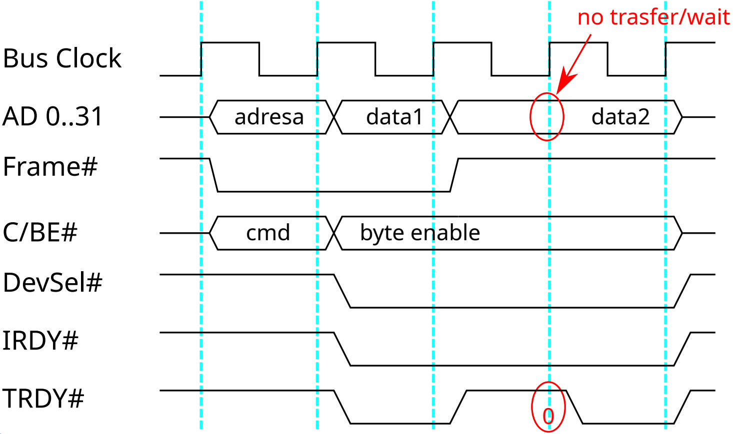
PCI Data Transfers – Write Transaction – Data

- 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.



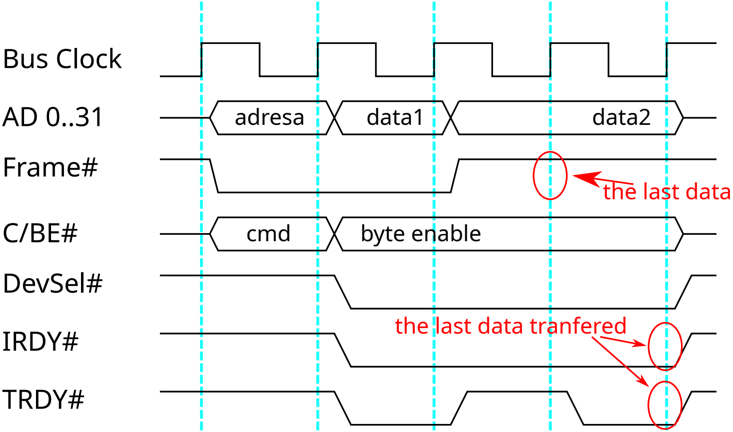
PCI Data Transfers – Write Transaction – Wait

- 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



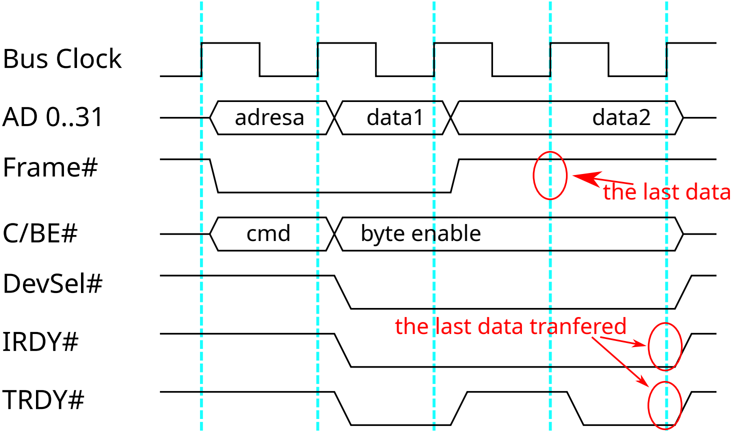
PCI Data Transfers – Write Transaction – the Last Data

- 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.



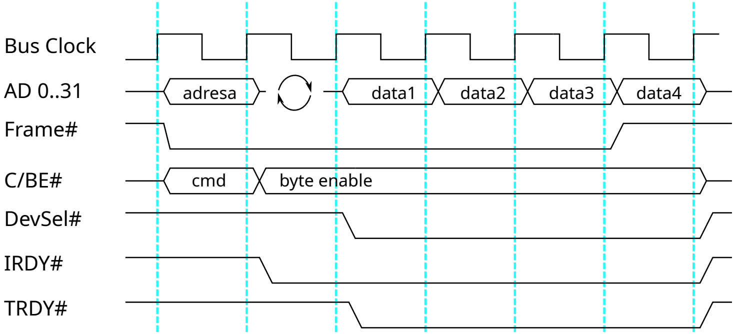
PCI Data Transfers – Write Transaction – Release Bus

- 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:

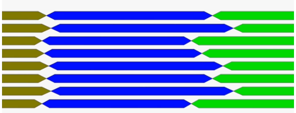
- Half-duplex data cannot be sent in both directions at the same time, data transferred in only one direction at time
- Multiple devices on the shared bus – slow peripherals slow down fast peripherals, increases the latency of all other peripherals
- PCI bus only allows clocks with 33 MHz, or 66 MHz
  - 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 Expres – PCIe

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

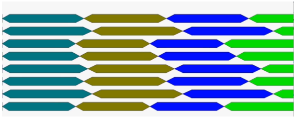
- Even small inaccuracies in the length of the conductors and the quality of the connections lead to different propagation speed/delay of the electrical signal
- No problem for low frequencies but even small mutual and or clocks timing shift prevents consistent data strobing over multiple wires for high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

frequency f



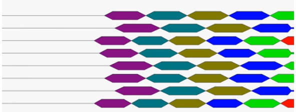
transfer without problems

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 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 D.x.B3	011	1100 0011	
D.x.4	100	1101	0010
D.x.5	101	1010	
D.x.6	110	0110	
D.x.P7 D.x.A7	111	1110 0111	0001 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 $250\,\mathrm{MB/s} = (2500 \cdot \frac{8}{10})\ \mathrm{Mb/s} = (2500 \cdot \frac{1}{10})\,\mathrm{MB/s}$, 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 $(16 \cdot 250)\,\mathrm{MB/s} = 4\,\mathrm{GB/s}$

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 $(16 \cdot 500)\ \mathrm{MB/s} = 8$ GB/s

PCIe Vresion. 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 MB/s $= (8000 \cdot \frac{128}{130})$ Mb/s $= (8000 \cdot \frac{16}{130})$ MB/s
  - The maximum bandwidth for x16 is $(16 \cdot 985)$ MB/s = 15.75 GB/s
- Ver 4.x
  - The transfer rate is 16 GT/s (transfers per second)
  - The maximum bandwidth is therefore almost 1.97 GB/s $= (16000 \cdot \frac{16}{130})$ MB/s
  - The maximum bandwidth for x16 is $(16 \cdot 1.97)$ GB/s = 31.5 GB/s
- Ver 5.x
  - The transfer rate is 32 GT/s (transfers per second)
  - The maximum bandwidth is therefore almost 3.94 GB/s $= (32000 \cdot \frac{16}{130})$ MB/s
  - The maximum bandwidth for x16 is $(16 \cdot 3.94)$ GB/s = 63 GB/s

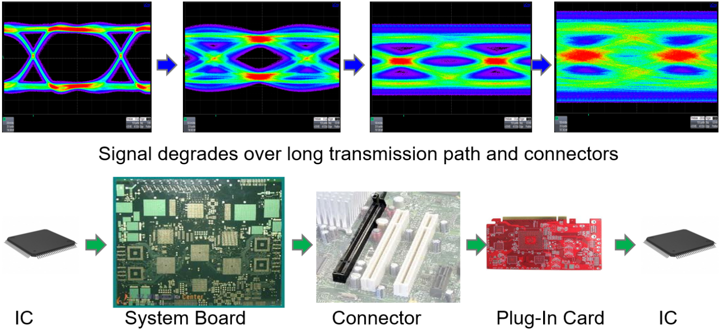
PCIe Topology

Communication over the PCIe bus is similar to communication over a swiched 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



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) Utra 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