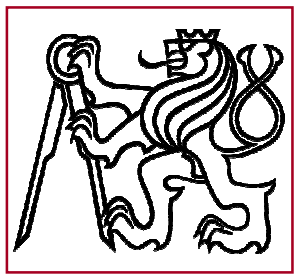
Y35PES

Programming for Embedded Systems

Ondřej Špinka, Pavel Němeček
spinkao@fel.cvut.cz nemecp1@fel.cvut.cz
http://dce.felk.cvut.cz/pes

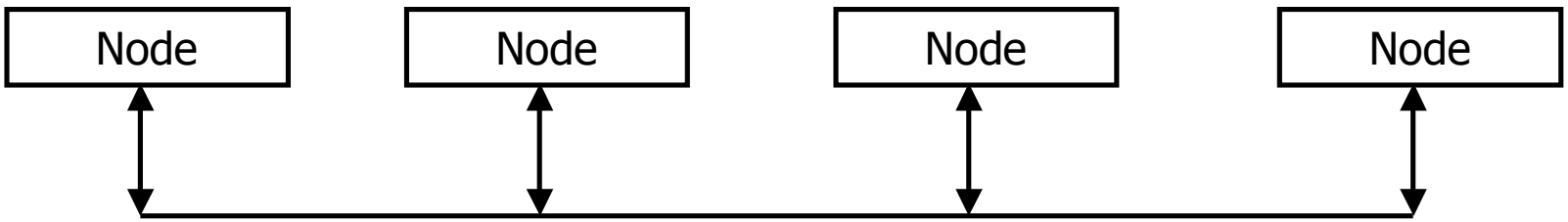


12/7/2010

© Ondřej Špinka 2010

Distributed Embedded Systems (1)

- A distributed embedded system consists of nodes, connected usually by a bus
- Usually, the system is **hierarchical**, consisting of one (or possibly several) **master** and **slave** nodes
- Distribution of the system enhances its serviceability, improves HW reusability and simplifies testing, although care must be taken to not to "overengineer" things
- Distribution can also introduce unwanted interactions into the system



© Ondřej Špinka 2010

Distributed Embedded Systems (2)

- Naturally, distributed embedded systems are highly reliant on **communications**. Correct interactions between nodes are of crucial importance.
- Communications must be taken into account especially when dealing with real-time. Designing correct communication protocol is often not an easy task, especially for complex systems. There are many pre-engineered solutions, like CANopen or Profibus.
- Collisions on the bus have major impact on real-time communications. Solving collisions is not an easy task.
- Basically, there are two modes of failure of a node from a communication point of view; it can become **silent**, or it can turn into a **bubbling idiot**. Bubbling idiots are much more dangerous than silent nodes, since they can seriously disturb communication among other nodes, or even render the bus unusable.
- Protocols in safety-critical ES must be prepared to handle bubbling idiots.

© Ondřej Špinka 2010

Solutions of Collisions

- There are several approaches to handle access collisions on a bus. There can be a **bus master**, a single node, which is the only one allowed to initialize a communication; other nodes (**slaves**) are only allowed to transmit when asked by the bus master. Naturally, a failure of a bus master renders the whole system dead. These so-called **master-slave** protocols are convenient mainly because they are **strictly deterministic**, latencies can be easily evaluated and collisions are not possible under normal circumstances.
- Multi-master protocols must either handle bus access collisions or avoid them. To avoid collisions in multi-master systems, **time-sharing** of the bus is often used; A **token** can be passed from one node to the other, allowing only the node handling the token to transmit. Alternatively, **time frames** can be associated to each node, allowing it to transmit only within its time frame. Clocks of the nodes must be synchronized in this case.
- If bus access collisions are permitted, there must be ability to solve them deterministically in real-time systems. Therefore, **non-destructive bus arbitration**, with **priority** assigned to each node, is required. Destructive **CSMA/CD** (**Carrier Sense Multiple Access / Collision Detection**), as on Ethernet, is not feasible for real-time systems.

© Ondřej Špinka 2010

Serial Communication (1)

Serial buses have many advantages over parallel ones

- Less wires needed, hence smaller, less complex, cheaper, lighter
- Faster, thanks to little or no crosstalk
- Modern processors often use internal serial busses

Serial lines can be

- **Asynchronous** (UART – Universal Asynchronous Receiver/Transmitter) (without a clock line) – RS-232, USB, RS-485, CAN…
- **Synchronous** (USRT) (synchronized by a separate clock line) – SPI, I2C…

Point to point communication can be

- **Half-duplex** (only one node can transmit at a given time)
- **Full-duplex** (both nodes can transmit simultaneously)

© Ondřej Špinka 2010

Serial Communication (2)

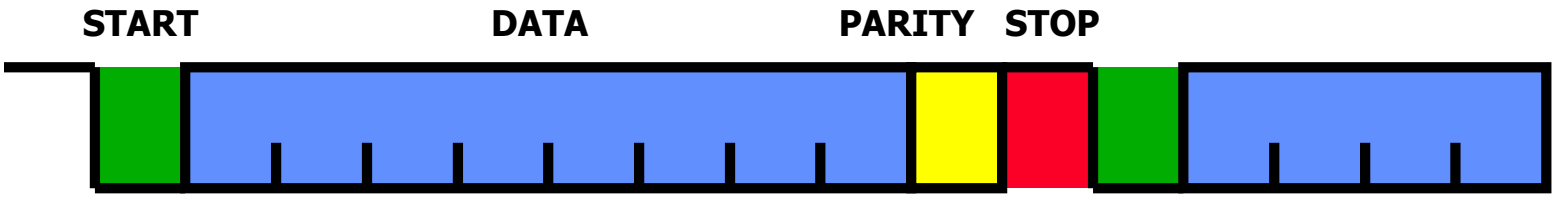
Physical layer properties

- Using higher voltage levels improves the noise tolerance, but makes the bit rate slower and increases the power consumption
- Differential voltage also improves noise tolerance, but makes the hardware more complex
- The same holds for optical decoupling

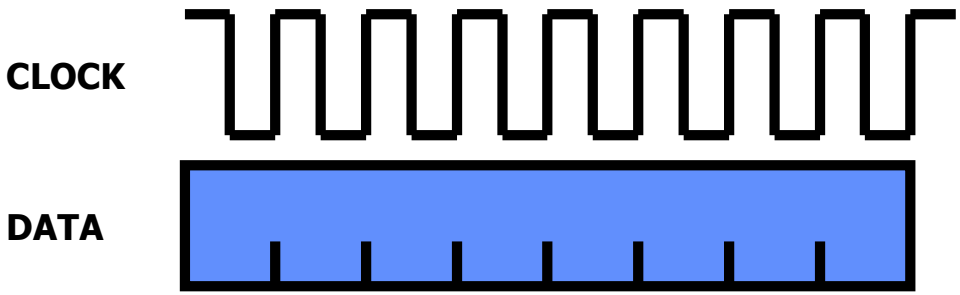
© Ondřej Špinka 2010

Serial Communication (3)

UART data frame format



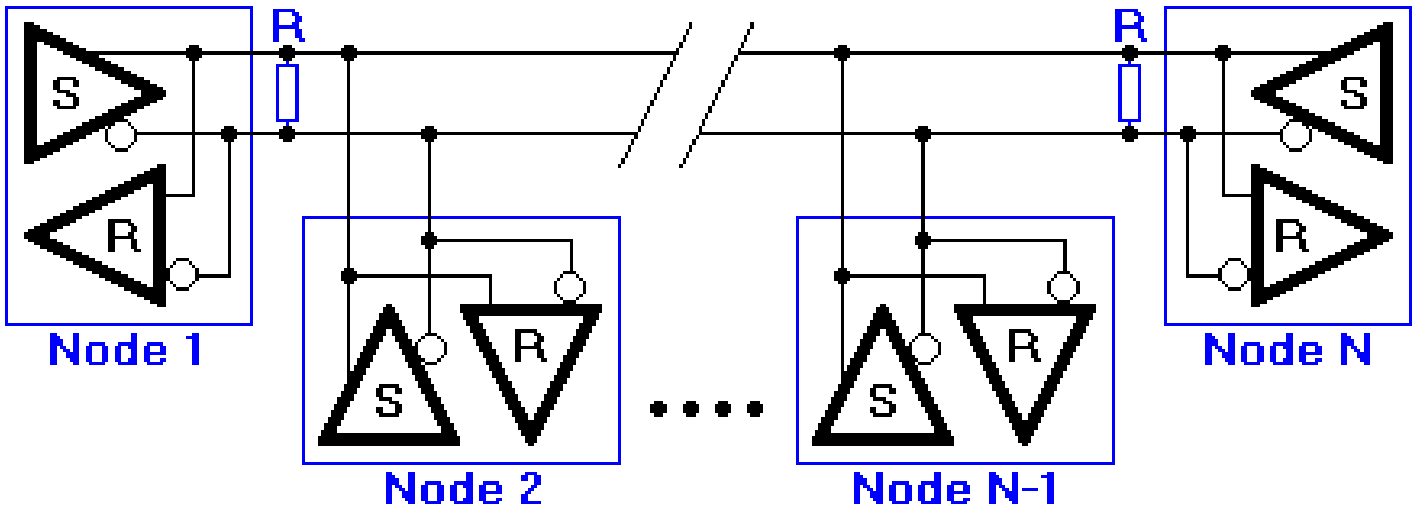
USRT data frame format



© Ondřej Špinka 2010

RS-485

- Master-slave communicating protocols are mostly used to avoid collisions
- Very robust and reliable, uses differential voltage
- Bandwidth depends on bus length – can be up to 35Mb/s (12m) to 100kb/s (1200m)



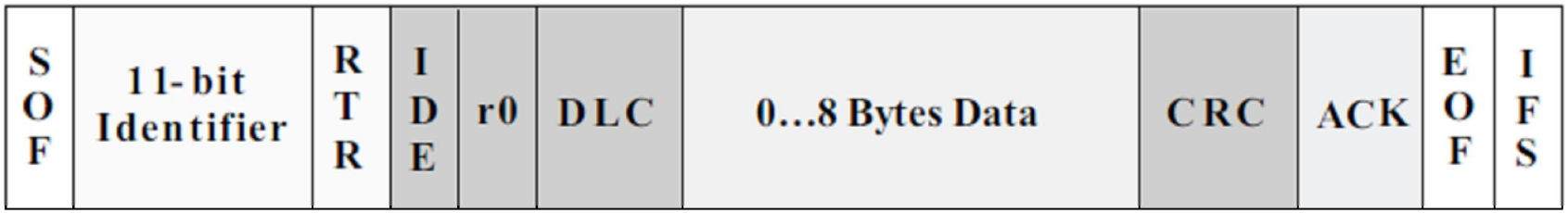
© Ondřej Špinka 2010

Controller Area Network (CAN) (1)

- Very reliable industrial serial bus, **suitable for real-time communications** due to **non-destructive arbitration**, mainly used in automotive industry
- Up to 1Mb/s data rate
- Supports packet priorities
- Works on various physical layers, electrical and optical, but the physical layer must support **dominant** (0) and **recessive** (1) logic levels
- Standard 11-bit or extended 29-bit packet IDs
- Uses **bit stuffing** to prevent out-syncing during long data frames, since **NRZ (Non-Return to Zero)** encoding is used. Therefore, a maximum of 5 consecutive bits of the same logical value must be followed by an opposite "stuffed" bit (which is not considered as data-carrying).
- 15 bit **CRC (Cyclic Redundancy Check)** is used to secure the data

© Ondřej Špinka 2010

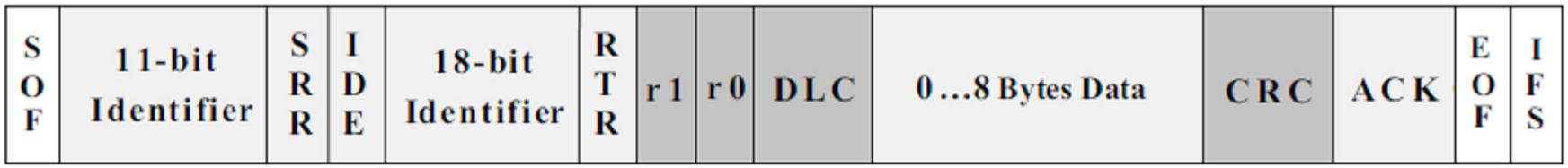
Controller Area Network (CAN) (2)



- SOF – Start Of Frame bit (dominant)
- RTR – Remote Transmit Request
- IDE – IDentificator Extension – always dominant in standard frame
- r0 – reserve bit
- DLC – Data Length Code
- CRC – Cyclic redundancy check (15 bits + 1 bit delimiter)
- ACK – Acknowledge (dominant, must be transmitted by other nodes)
- EOF – End Of Frame (7 consecutive recessive bits **without bit stuffing**)
- IFS – Inter-Frame Space (7 consecutive recessive bits **without bit stuffing**)

© Ondřej Špinka 2010

Controller Area Network (CAN) (3)



- IDE bit is recessive in this case (Extended ID field)
- SRR (Substitude Remote Request) is used as a placeholder for standard RTR
- r1 – additional reserved bit

© Ondřej Špinka 2010