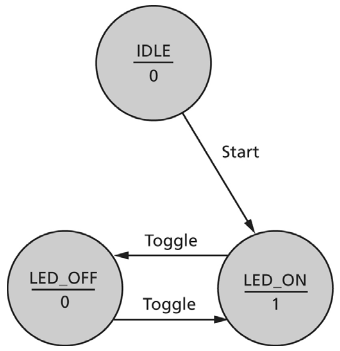
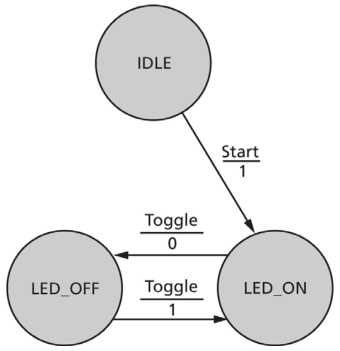
Lecture 4: Interfaces

By: Emad Samuel Malki Ebeid

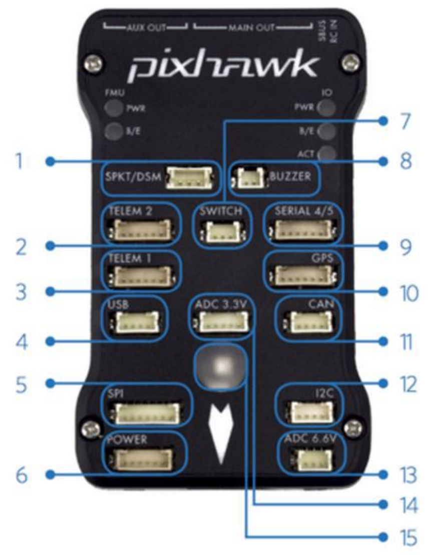
Summary of lecture 3

- Latches & Flip flops
- Parallel & shift registers
- FSM
- J1





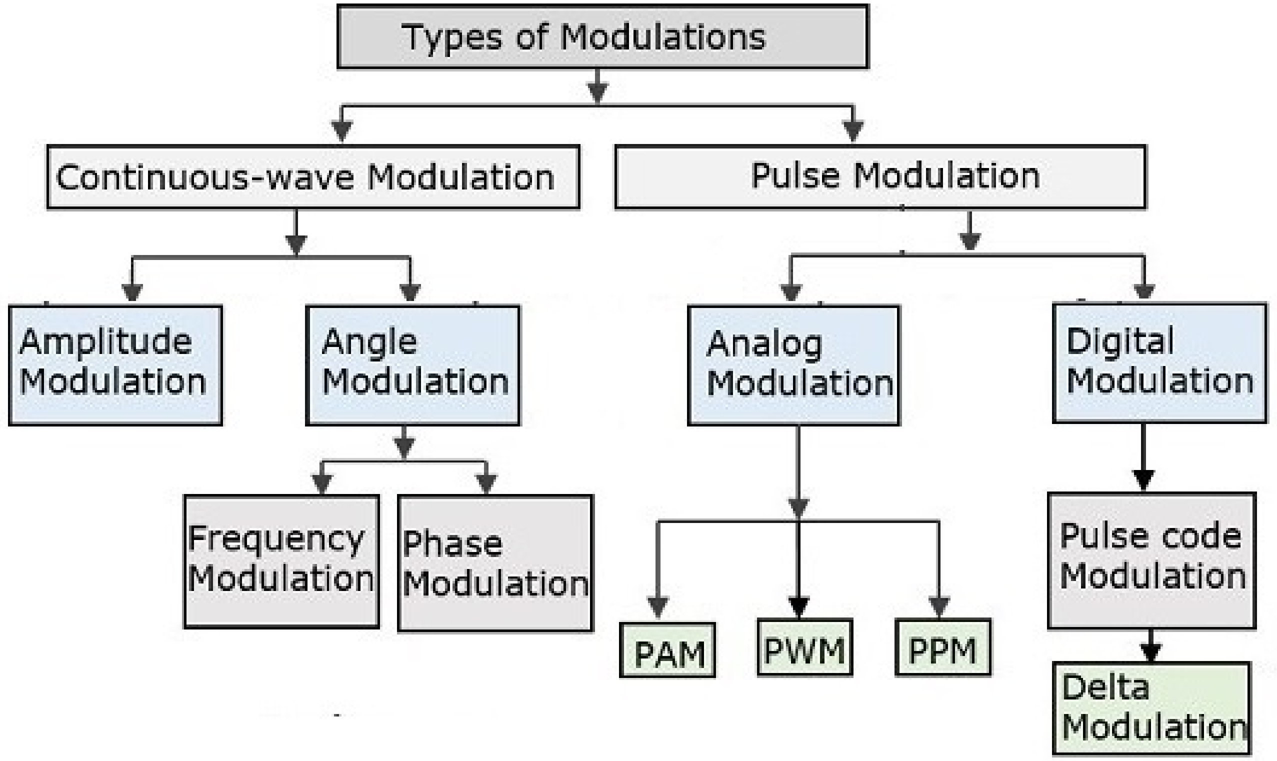
Communication interfaces



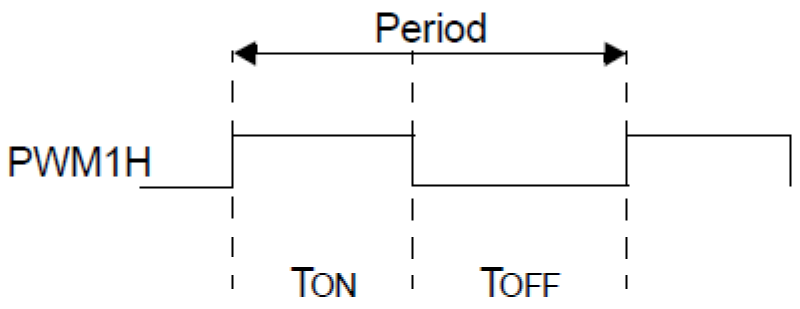
1	Spektrum DSM receiver
2	Telemetry (radio telemetry)
3	Telemetry (on-screen display)
4	USB
5	SPI (serial peripheral interface) bus
6	Power module
7	Safety switch button
8	Buzzer
9	Serial
10	GPS module
11	CAN (controller area network) bus
12	I^2C splitter or compass module
13	Analog to digital converter 6.6 V
14	Analog to digital converter 3.3 V
15	LED indicator

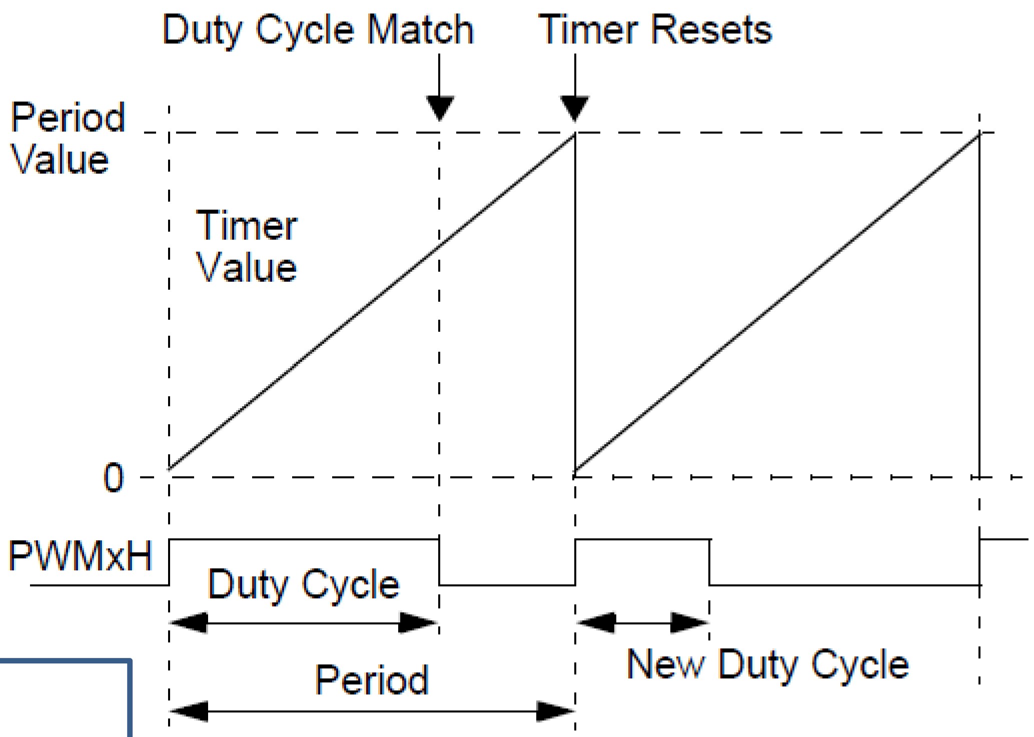
Communication technologies

- Analog and digital communication



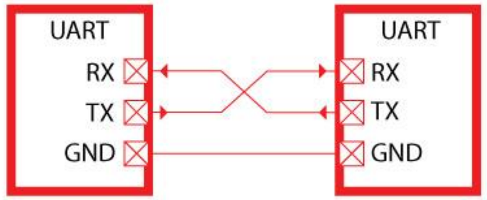
PWM





10 minutes to think about how to build PWM circuit.

Serial interface: Universal Asynchronous Receiver/Transmitter

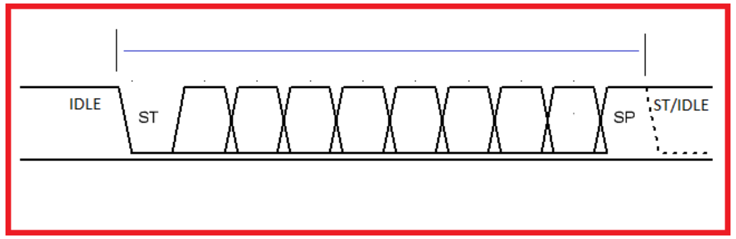


Serial communication introduction

- Serial port is a serial communication physical interface through which information transfers in or out one bit at a time. Data transfer through serial ports connected the computer to devices such as terminals and various peripherals.

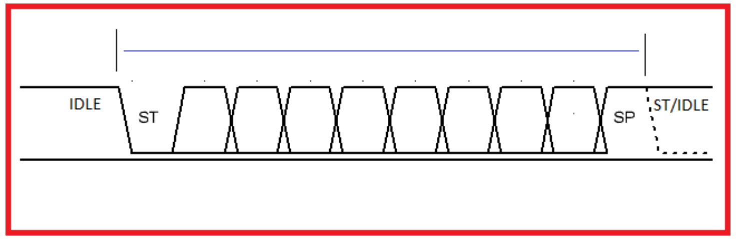
- Some computers, such as the IBM PC, used an integrated circuit called a UART, that converted characters to (and from) asynchronous serial form, and automatically looked after the timing and framing of data. A universal asynchronous receiver/transmitter (usually abbreviated UART) is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms

Baudrate



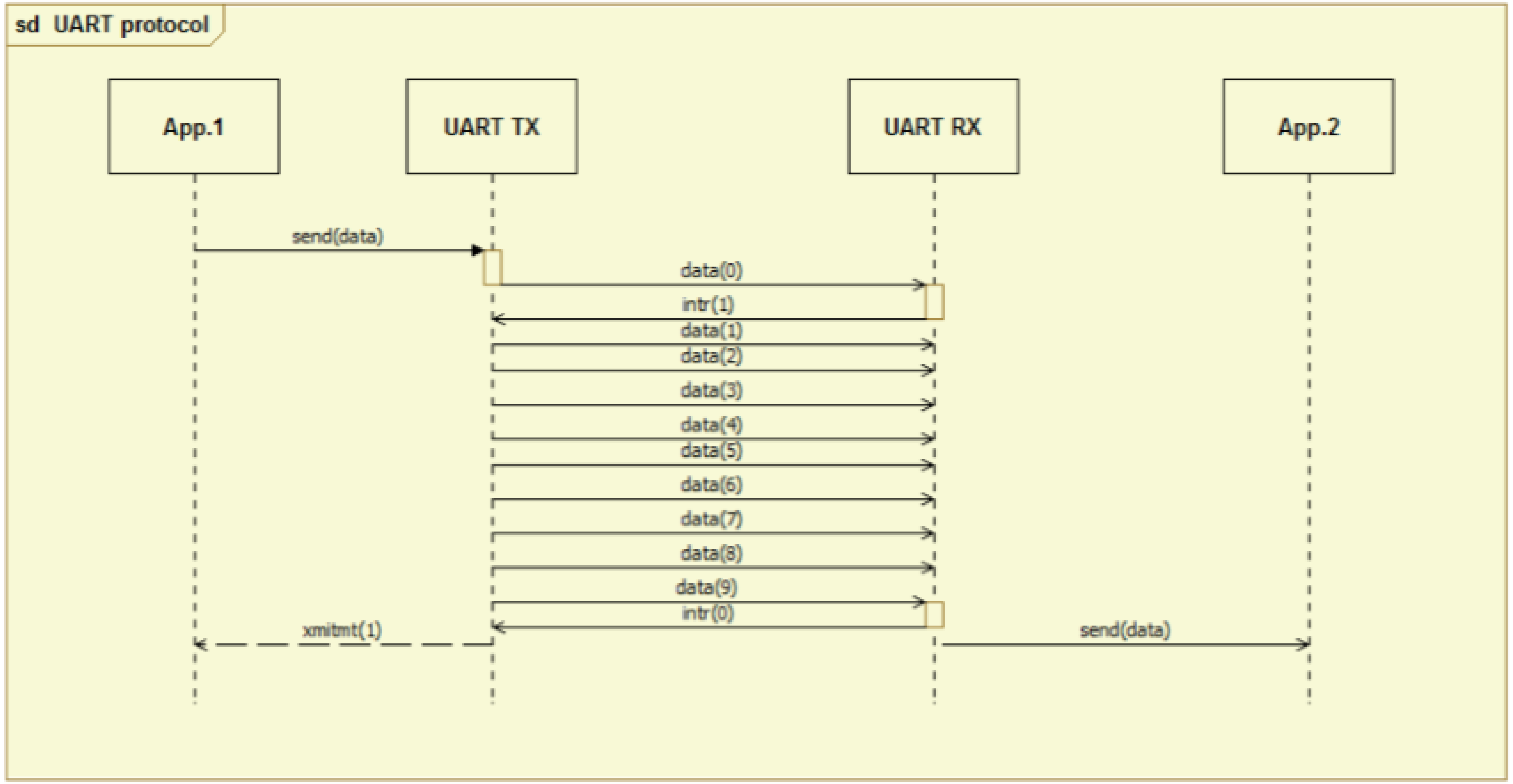
- **Baudrate**: In embedded designs, it is necessary to choose a proper oscillator to get the correct baud rate with little or no error. Some examples of common crystal frequencies and baud rates with no errors are: 300, 600, 1200, 1800, 2400, 4800, 7200, 9600, 14400, 19200, 38400, 57600, 115200 Bd
- **Data bits:** The number of data bits in each character can be 5 (for Baudot code), 6 (rarely used), 7 (for true ASCII), 8 (for any kind of data, as this matches the size of a byte), or 9 (rarely used). 8 data bits are almost universally used in newer applications. 5 or 7 bits generally only make sense with older equipment such as teleprinters. Most serial communications designs send the data bits within each byte LSB (Least Significant Bit) first. This standard is also referred to as "little endian". Also, possible, but rarely used, is "big endian" or MSB (Most Significant Bit) first serial communications. The order of bits is not usually configurable, but data can be byte-swapped only before sending.

Parity and stop bits

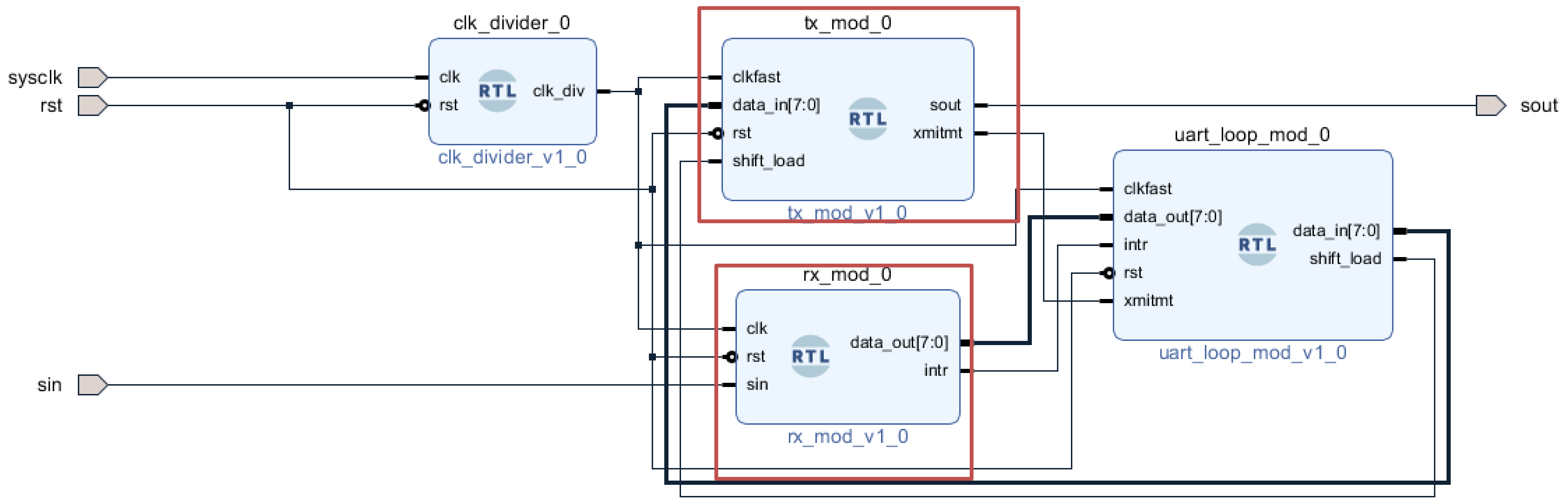


- **Parity:** Parity is a method of detecting errors in transmission. When parity is used with a serial port, an extra data bit is sent with each data character, arranged so that the number of 1 bits in each character, including the parity bit, is always odd or always even. If a byte is received with the wrong number of 1's, then it must have been corrupted. However, an even number of errors can pass the parity check.
- **Stop bits:** Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit. If slow electromechanical teleprinters are used, one-and-one half or two stop bits are required.

UART protocol



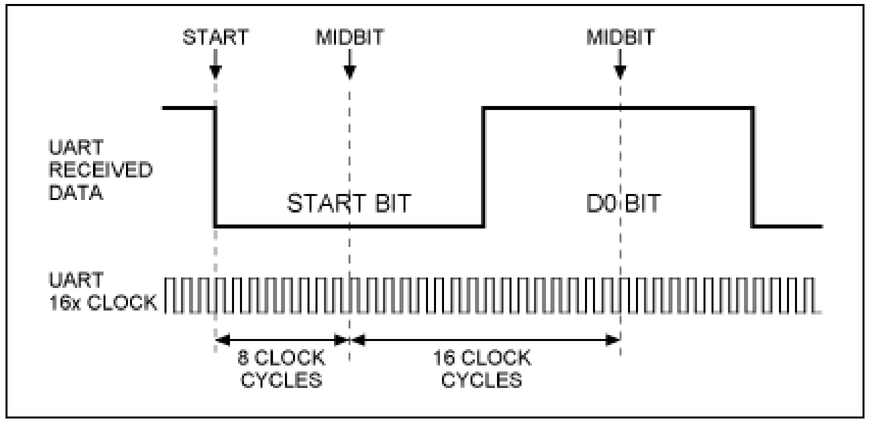
Overall system



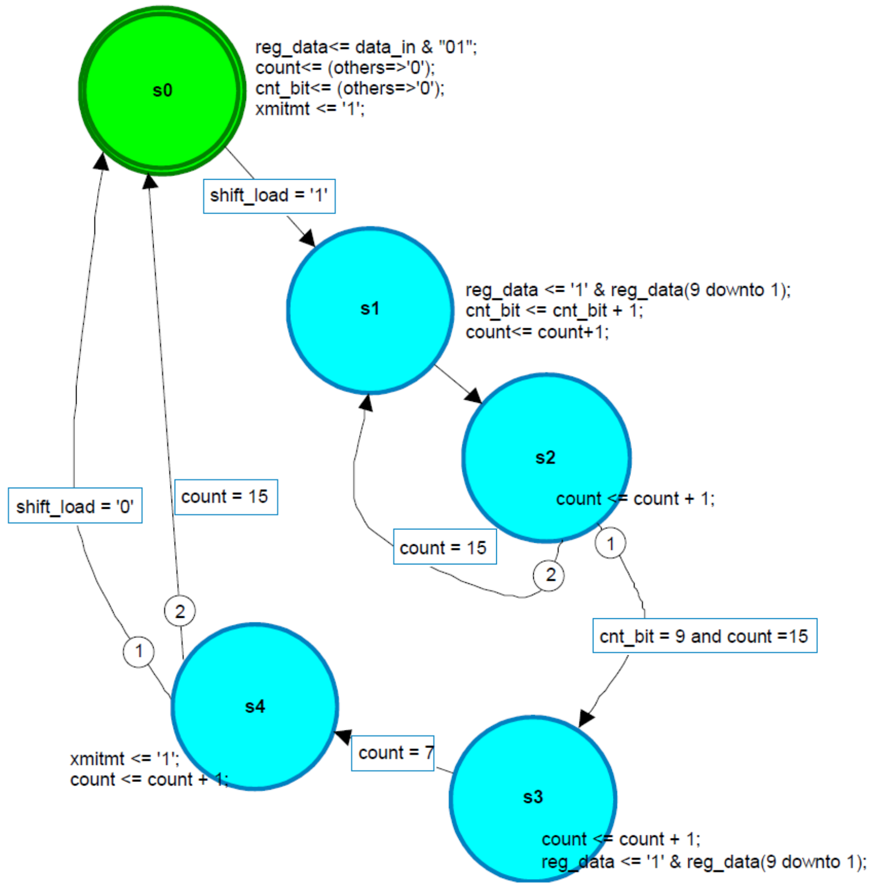
- The code is in BB

Receiver

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;


ENTITY rx_mod IS
   PORT(
      clk      : IN     std_logic;
      rst      : IN     std_logic;
      sin      : IN     std_logic;
      data_out : OUT    std_logic_vector (7 DOWNTO 0);
      intr     : OUT    std_logic);

END rx_mod ;

-- hds interface_end
ARCHITECTURE rtl OF rx_mod IS
    signal rxreg: std_logic_vector(9 downto 0);
    signal count: unsigned (3 downto 0);
    signal rxmt: std_logic;
    signal rxin,start_flag: std_logic;
    begin
        process (clk, rst)
            begin
                if (rst = '0') then
                    count <= (others => '0');
                    rxmt <= '1';
                    rxreg <= (others => '1');
                    intr <= '0';
                    rxin <= '1';
                    start_flag<='0';
                elsif (rising_edge(clk)) then
                    rxin<=sin;
                    if (rxmt = '1' and rxin = '0') then
                        count <= (others => '0');
                        rxmt <= '0';
                        rxreg <= (others => '1');
                        start_flag<='0';
                    elsif (count = 7 and rxmt = '0' and rxin = '0' and start_flag='0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= (others => '0');
                        start_flag<='1';
                    elsif (count = 15 and rxmt = '0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= count + 1;
                    else
                        count <= count + 1;
                    end if;
                    if (rxmt = '0' and rxreg(9) = '1' and rxreg(0) = '0') then
                        intr <= '1';
                        rxmt <= '1';
                    else
                        intr <= '0';
                    end if;
                end if;
        end process;
        data_out <= rxreg(8 downto 1);
END rtl;
```



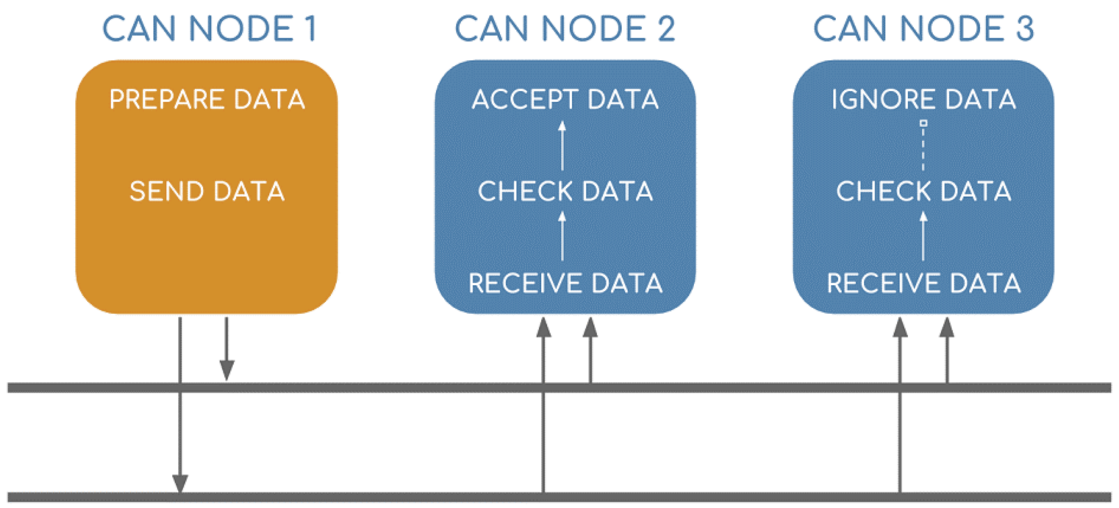
Transmitter



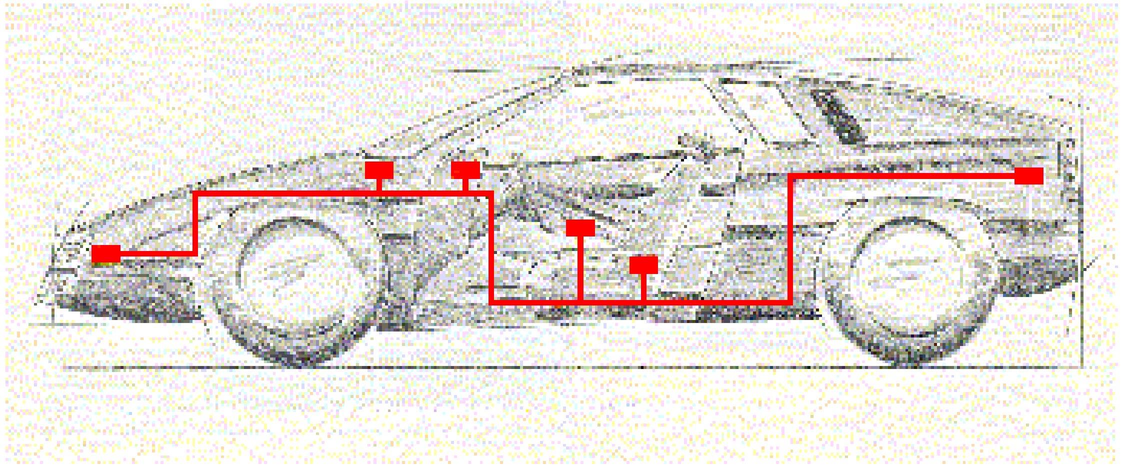
Lab work

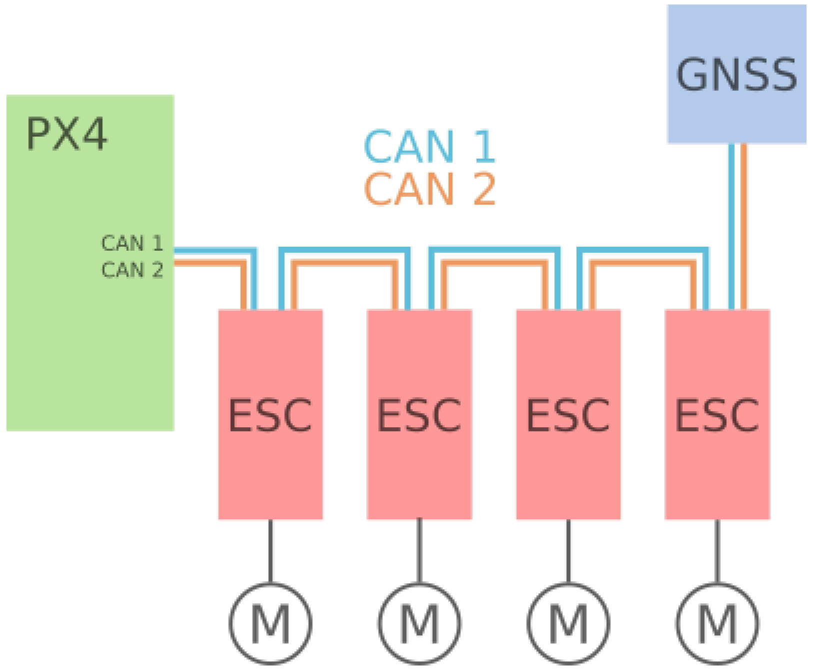
- PWM and UART implementations on the FPGA

Controller Area Network (CAN bus)



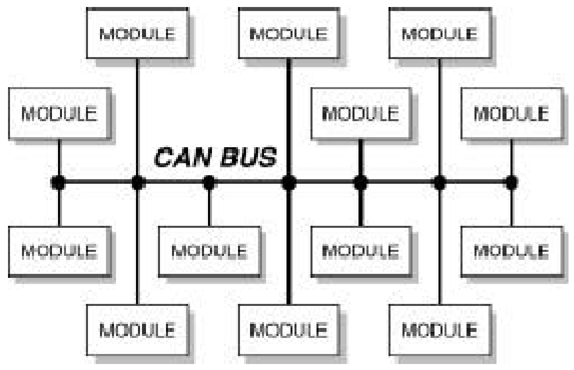
CAN-BUS Introduction



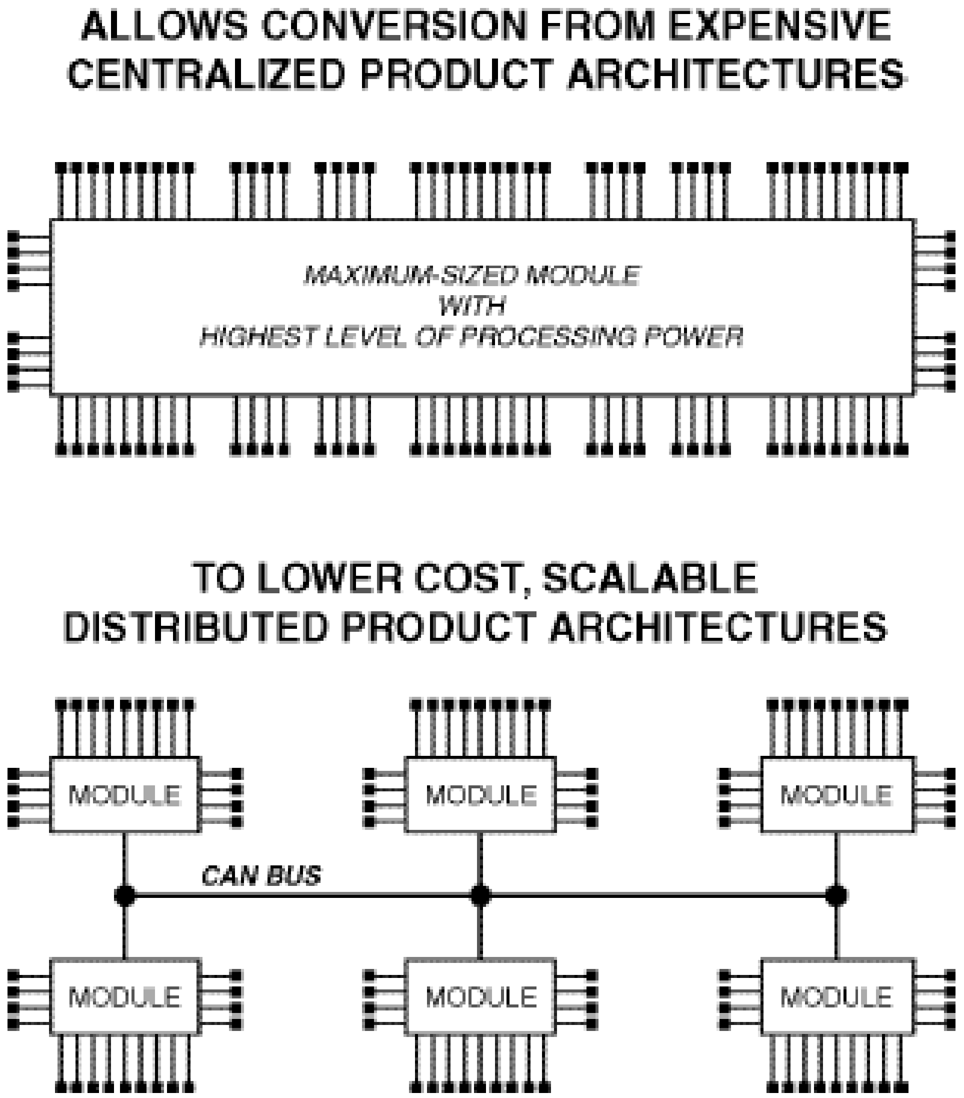


What is CAN?

- **Controller Area Network (CAN)** is a common, small area network solution that supports distributed product and **distributed system architectures**
- The CAN bus is used to interconnect a network of electronic nodes or modules
- Typically, **a two wire, twisted pair cable** is used for the network interconnection



Why use CAN?


ALLOWS CONVERSION FROM EXPENSIVE
CENTRALIZED PRODUCT ARCHITECTURES
MAXIMUM-SIZED MODULE
WITH
HIGHEST LEVEL OF PROCESSING POWER
TO LOWER COST, SCALABLE
DISTRIBUTED PRODUCT ARCHITECTURES
MODULE
MODULE
MODULE
CAN BUS
MODULE
MODULE
MODULE

From
Centralized

To
Distributed
Architectures

Three Development Scenarios for CAN

1. Development of a **controller** (SW/HW) to be used as a **node** in a CAN-BUS system
   - To be sold as a CAN enabled system
2. Development of a **distributed system** with several distributed nodes communicating via a CAN-BUS
3. Development of a **product** (e.g. an apparatus), where CAN is used as an **internal communication bus and protocol** for distributed internal communication

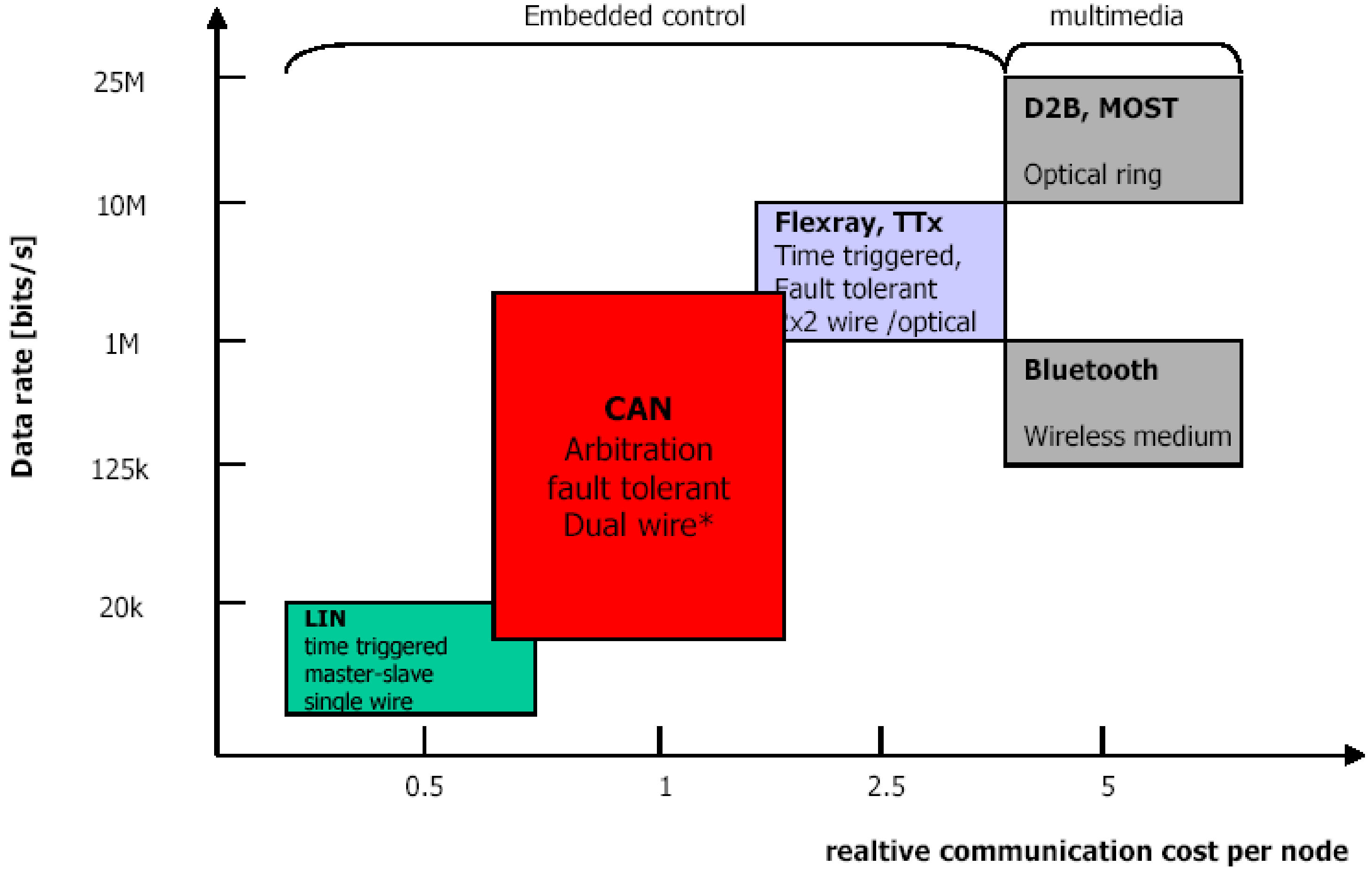
Key Reasons to Use CAN

- Low connection cost
- Low cost components
- Growing number of CAN chips & µControllers
- Increasing knowledge base
- Increasing integration service base
- Wide variety of CAN-based products
- Wide variety of Off-the-Shelf tools available
- Potential lower wiring costs
- Lower weight

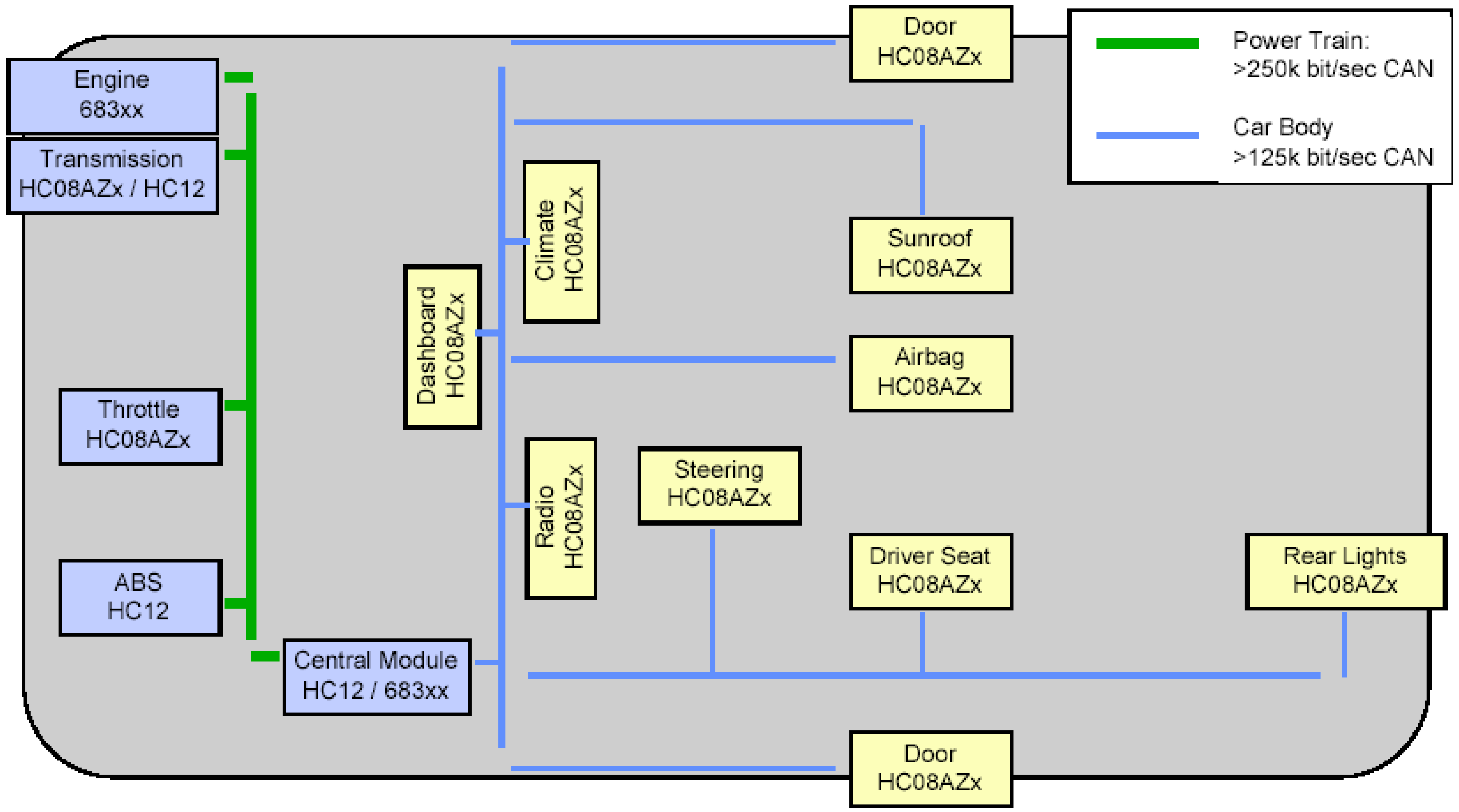
CAN History

- CAN first introduced by Bosch in **1986**
- Bosch published CAN specification version 2.0 in 1991
- Official **ISO CAN standard in 1993**: **ISO 11898**
- In 1999 57 million CAN controller chips sold
- Estimated to 300 million CAN chips in 2003
- **TTCAN:** Time Triggered CAN protocol: **ISO 11898-4** standard in **2004**.
- More History on the CiA webpage:
  http://www.can-cia.de/can-knowledge/can/can-history/

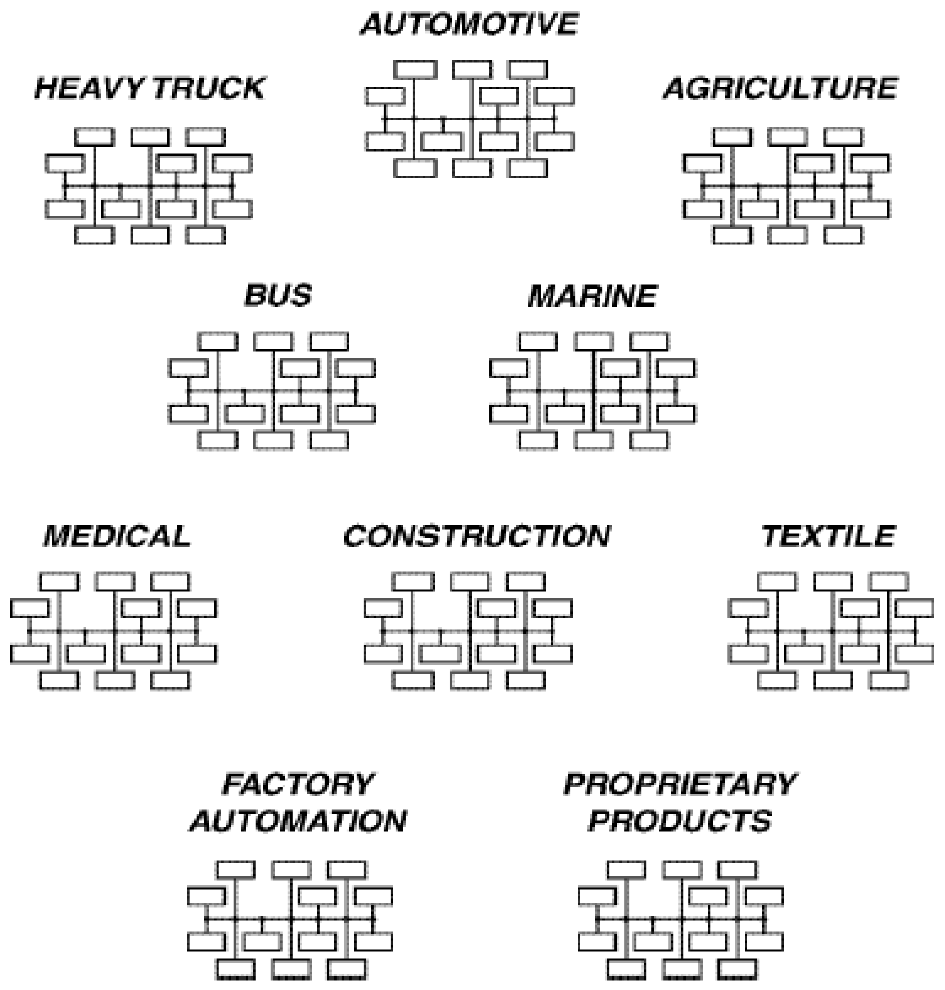
Field Buses in the Automotive Industry



Typical CAN Network



What Industries are using CAN?



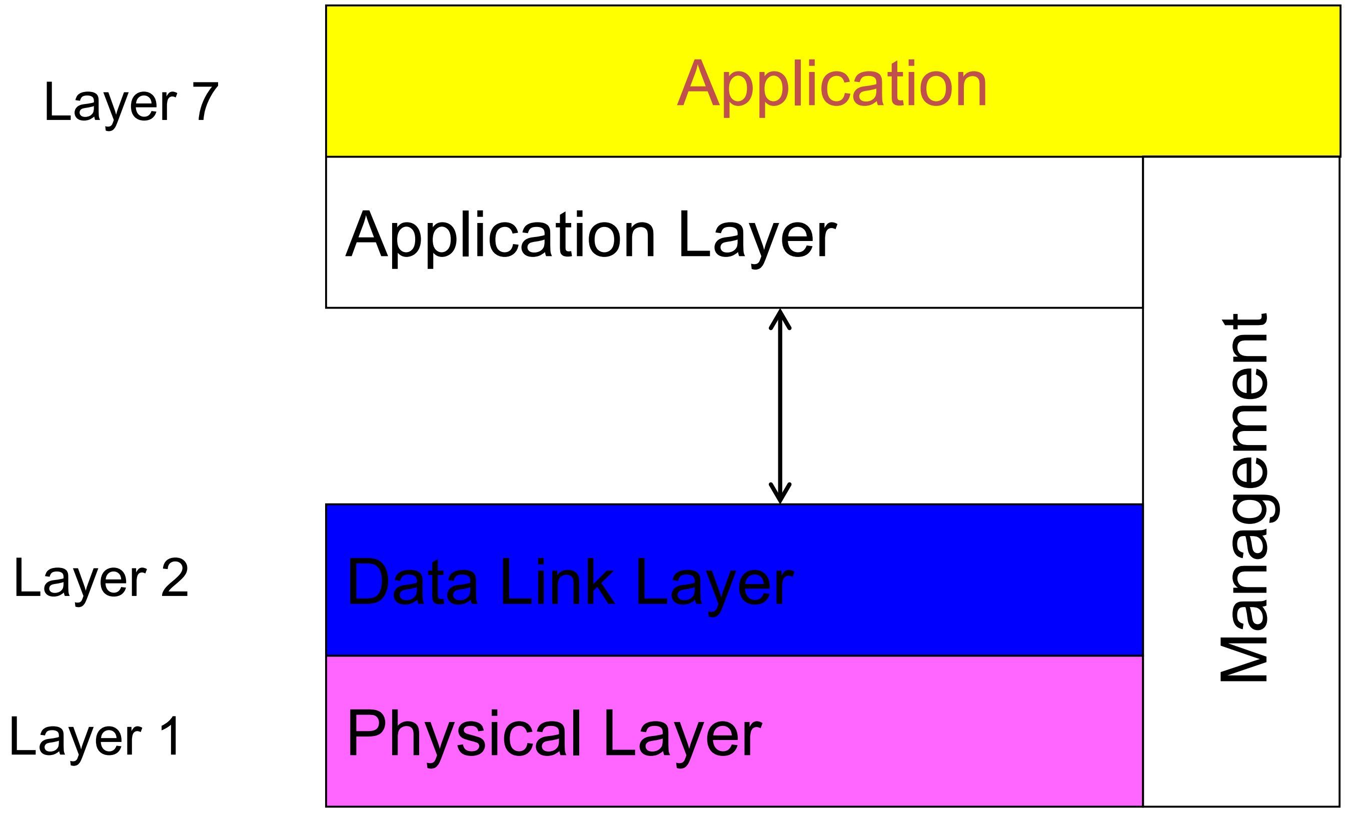
CAN - Highlights

- Is a high-integrity serial data communications bus for real-time applications
- Is an **event driven protocol**
- Is a **CSMA** (Carrier Sense Multiple Access) / **CA** (Collision Avoidance) protocol
- Operates at data rates of **up to 1 Mbits/s**
- Has excellent error detection capabilities
- Was originally developed by **Bosch** for use in cars (1986)
- Is now being used in many other industrial automation and control applications

CAN Standards

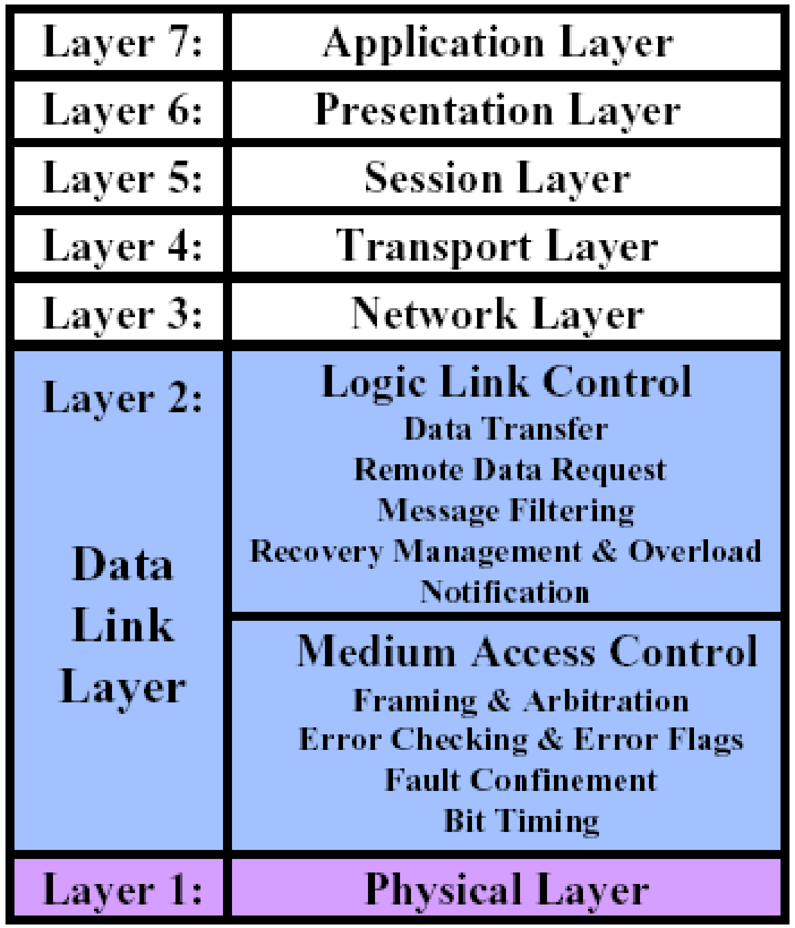
- Although CAN was originally **developed in Europe** by Robert Bosch for automotive applications, the protocol has gained wide acceptance and has become **an open, international ISO standard**
- As a result, the Bosch **CAN 2.0B** specification has become the de facto standard that new CAN chip designs follow
- **CAN ISO Standardization:**
  - ISO PRF 16845 : CAN Conformance Test Plan
  - ISO PRF 11898-1: CAN Transfer Layer
  - ISO PRF 11898-2: CAN High Speed Physical Layer
  - ISO DIS 11898-3: CAN Fault Tolerant Physical Layer
  - ISO DIS 11898-4: **TTCAN Time Triggered CAN**

Three-Layered Reference Model



CAN 2.0 Structure of a CAN Node

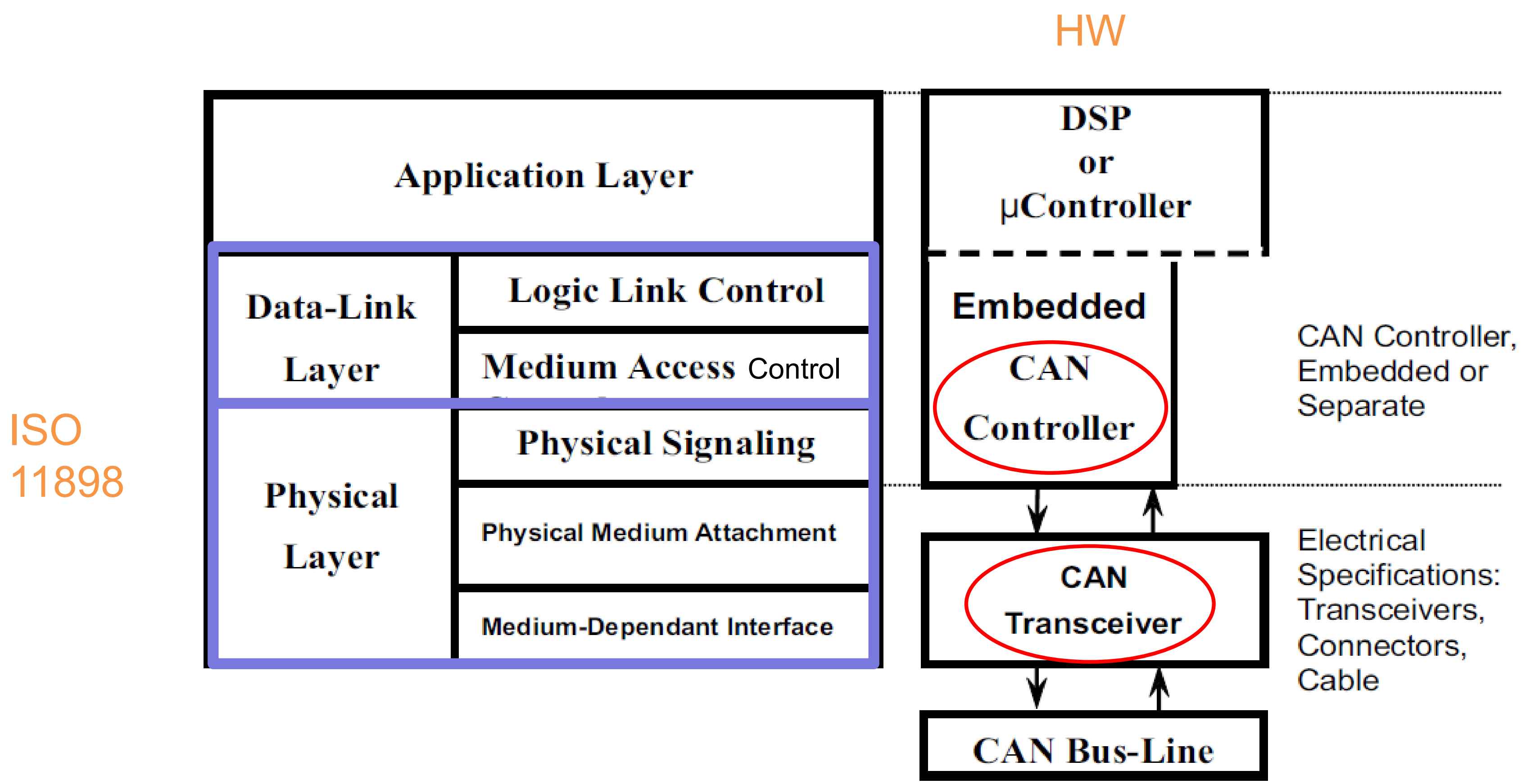
ISO/OSI Reference Model



Layer 7:	Application Layer
Layer 6:	Presentation Layer
Layer 5:	Session Layer
Layer 4:	Transport Layer
Layer 3:	Network Layer
Layer 2: Data Link Layer	**Logic Link Control** Data Transfer, Remote Data Request, Message Filtering, Recovery Management & Overload Notification
	Medium Access Control Framing & Arbitration, Error Checking & Error Flags, Fault Confinement, Bit Timing
Layer 1:	Physical Layer

CAN Specification **ISO 11898**, deals only with the **Physical** & **Data Link** Layers for a CAN network

ISO 11898 Architecture and CAN HW



Ref. Texas Instrument App. Report

How does CAN operate?

- CAN is a multiplexed serial communication channel
- CAN is a message-oriented protocol based on a **message identifier**
- CAN supports data transfers to multiple peers
- **No master controller** is needed to supervise the network conversation
- Can transfer up to **8 data bytes** within a single message
- For larger amounts of data, multiple messages are commonly used
- Most CAN based networks select a single bit rate
  - While communication bit rates may be as high as **1 MBit/s**, most implementations are **500Kbit/s** or less

Bit Rate versus Bus Length

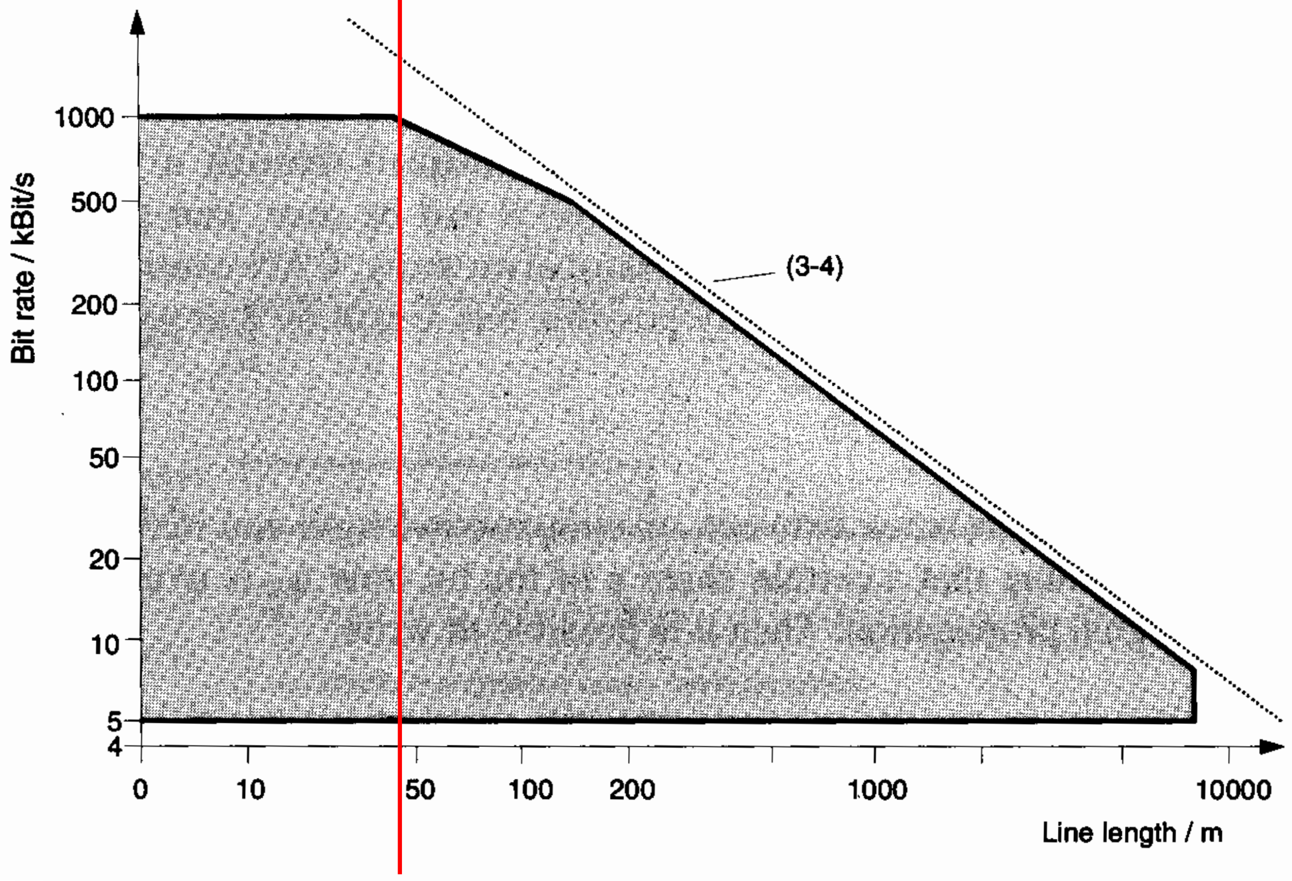
Bit Rate (kBits/s)	Maximum Bus length (m)
1000	**50**
500	110
135	620
100	790
50	1640

A Rule of Thumb for bus length > 100 m:

Bit Rate (Mbit/s) * Lmax (m) <=60

[Ref: Etschberger]

Bit Rate versus Bus Length



[Ref: Etschberger]

CAN Identifiers

- Labels the **content (type)** of a message used by receivers to select a message
- A system wide **unique identifier** for each message
- Used for **bus arbitration** & **determines the priority** of the message
  - Low id.number = high priority
- Two formats: **11 bit (A)** or **29 bit (B) identifiers**

CAN 2.0A vs CAN 2.0B

CAN 2.0A:

11 Bit Identifier

M68HC05X Family

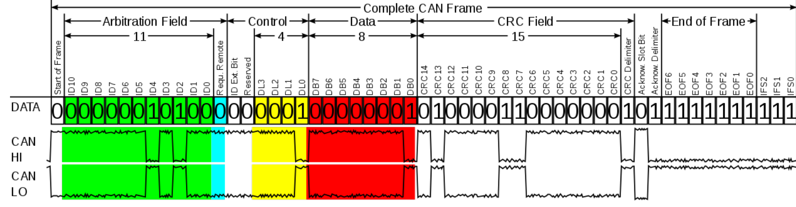
- Used by vast majority of current applications.
- **Greater message throughput** and **improved latency times**
- *Less silicon overhead !*

CAN 2.0B

29 Bit Identifier

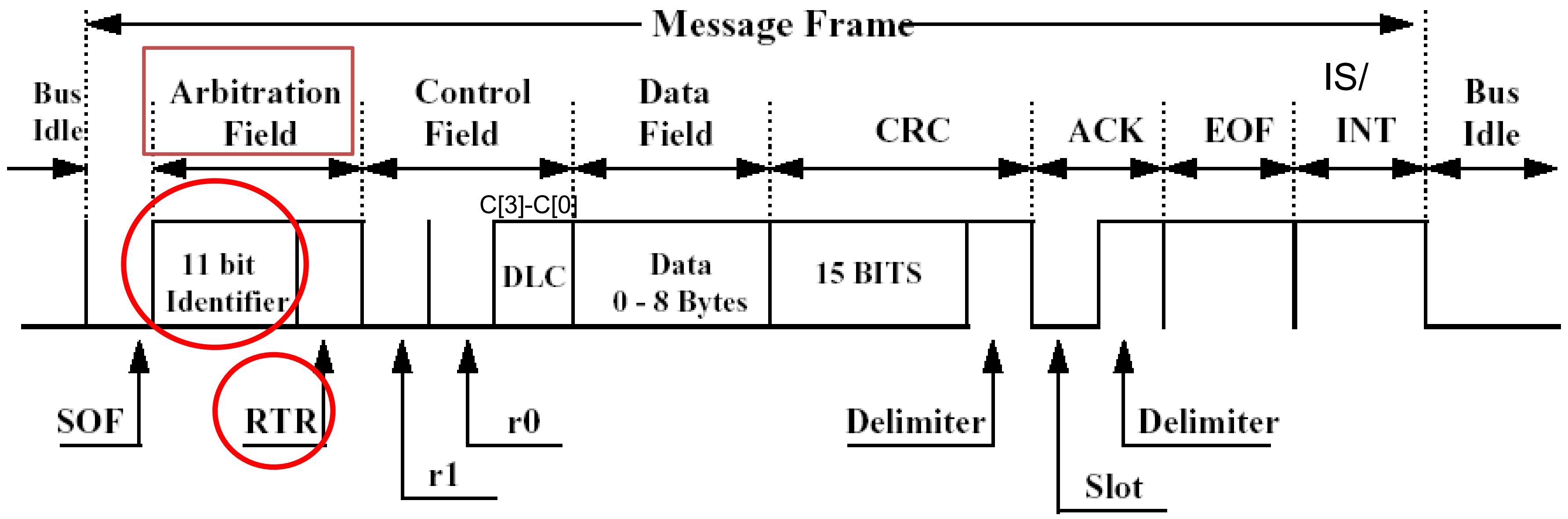
HC08 / HC11 + MSCAN

- Originally defined for USA Passenger Cars but now their Taskforce decree that it is **not necessary**.
- Allows more information in message but requires **more bus bandwidth**
- ***More silicon cost and less efficient use of bus !***



CAN 2.0A Message Frame

- CAN 2.0A (Standard Format)
  - 11 bit message identifier (2048 different frames)
  - Transmits and receives only standard format messages



RTR: Remote Transmission Request bit

CAN Message Frame

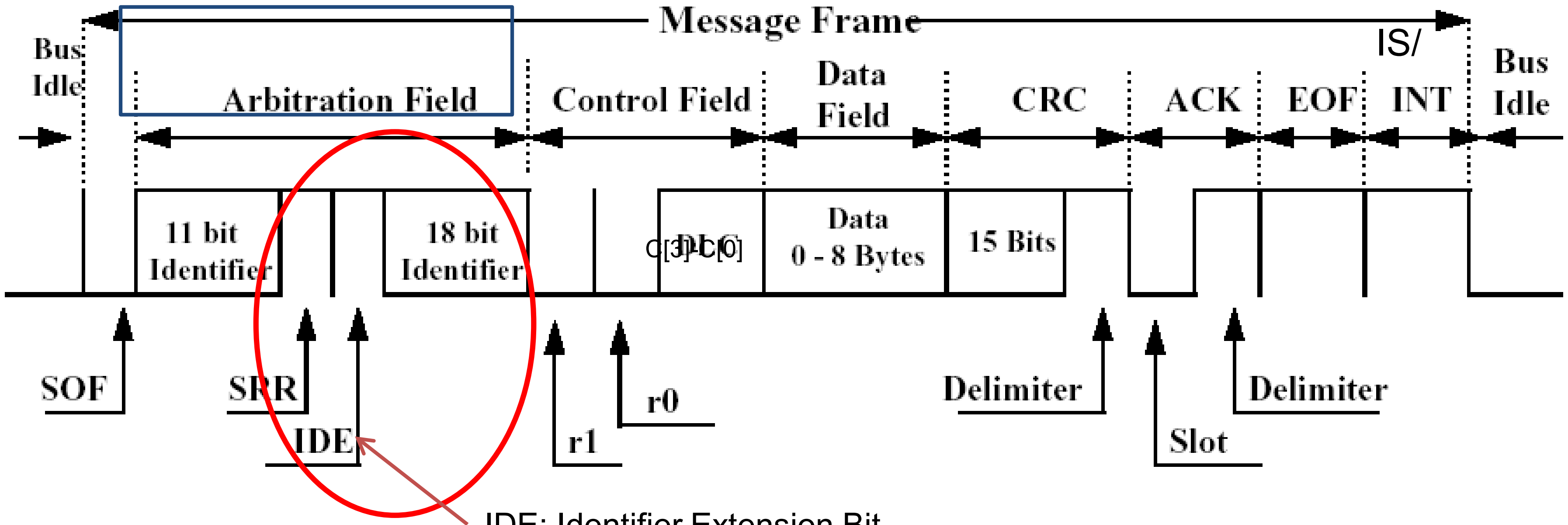
DATA FRAME:

- IS: Interframe space (INT)
- SOF: Start of frame, one single D-bit, start only if the bus is IDLE, all devices have to synchronize to the leading edge caused by START OF FRAME.
- ID: Identifier (CAN 2.0A (standard) = 11 bit, CAN 2.0B (extended) = 29 bit)
- RTR: Remote transmission request
  - D-bit: data follows = DATA FRAME
  - R-bit: transmission request to receiver = REMOTE RAME
- DLC: Data Length Code = 6 bit, C[3] - C[0] length of data array, MSB first
  - REMOTE FRAME: number of requested data bytes
  - C[5], C[4] are used for indicating extended IDs (2.0B)
- CRC: Cyclic redundancy checksum; 15 bit and a leading 0, sum and a R-bit delimiter bit
- ACK: Acknowledge (2 bits: ACK slot a and ACK delimiter)
  - The bit in ACK slot is sent as a R-bit and overwritten as a D-bit by those transducers which have received the message correctly.
- EOF: End of frame (7 R-bits)

Bit	>3	1	11,1	6	0...64	16	2	7
	IS	SOF	ID, RTR	DLC	DATA	CRC	ACK	EOF

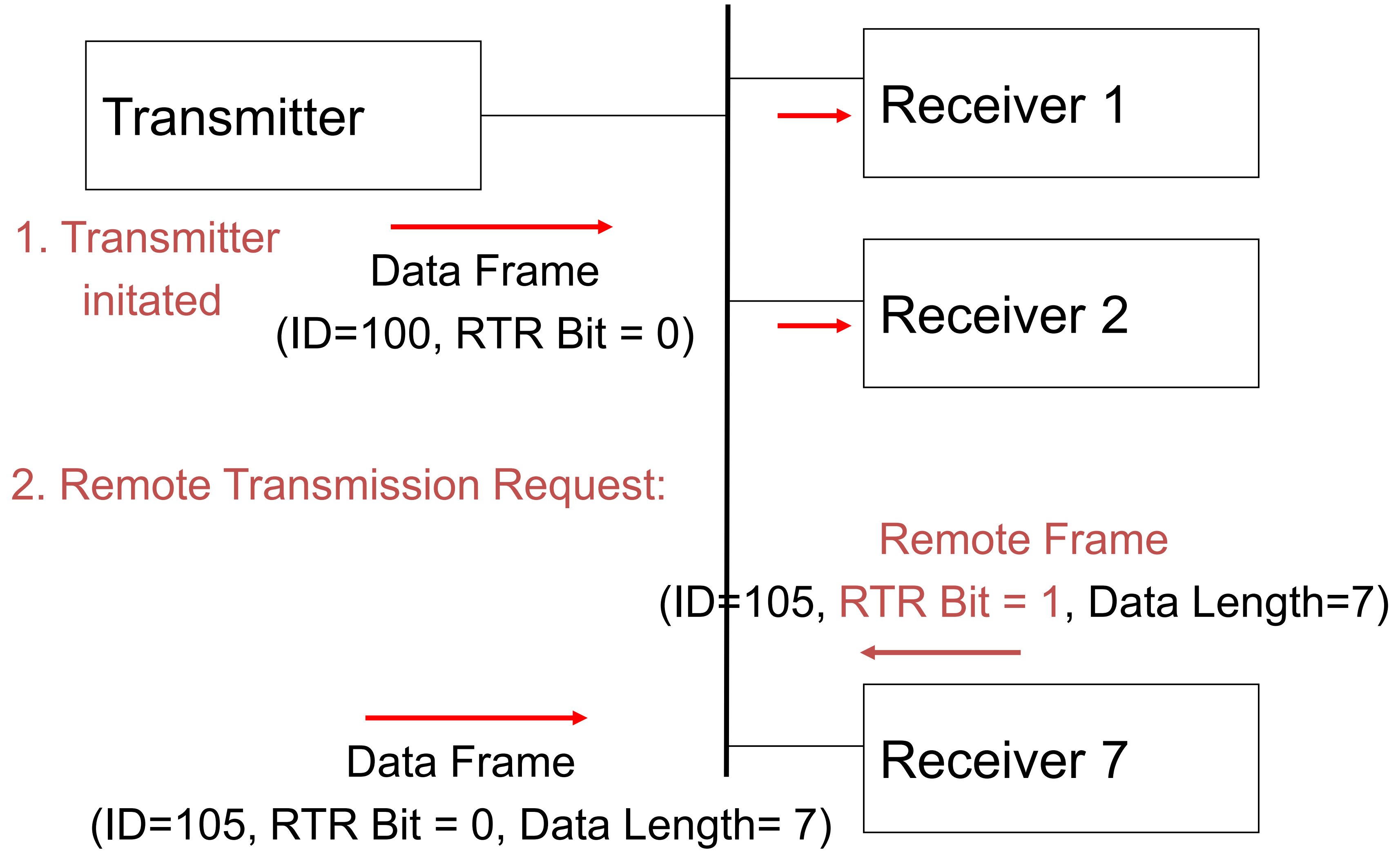
CAN 2.0B Message Frame

- CAN 2.0B (Extended Format)
  - Capable of receiving CAN 2.0A messages
  - 29 bit message identifier (512 million frames)
  - 11 bits for a CAN 2.0A message + 18 bits for a CAN 2.0B message



IDE: Identifier Extension Bit

Two Communcation Types



Arbitration (1)

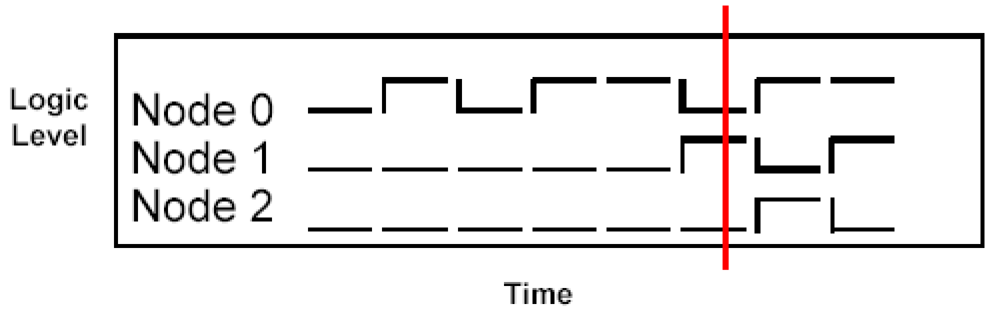
- Carrier Sense, Multiple Access **with Collision Avoidance (CSMA/CA)**
- Method used to arbitrate and determine the priority of messages
- Uses enhanced capability of non-destructive bitwise arbitration to provide collision resolution

Arbitration (2)

- A station may send if the bus is free (carrier sense)
- Any message begins with a field for unique bus arbitration containing the **message ID**
- The station with the **lowest ID is dominant**
- A dominant bit (D-Bit) is 0 and recessive bit R-Bit is high (1)
- The **lowest identifier** has the **highest priority**

Bitwise Arbitration

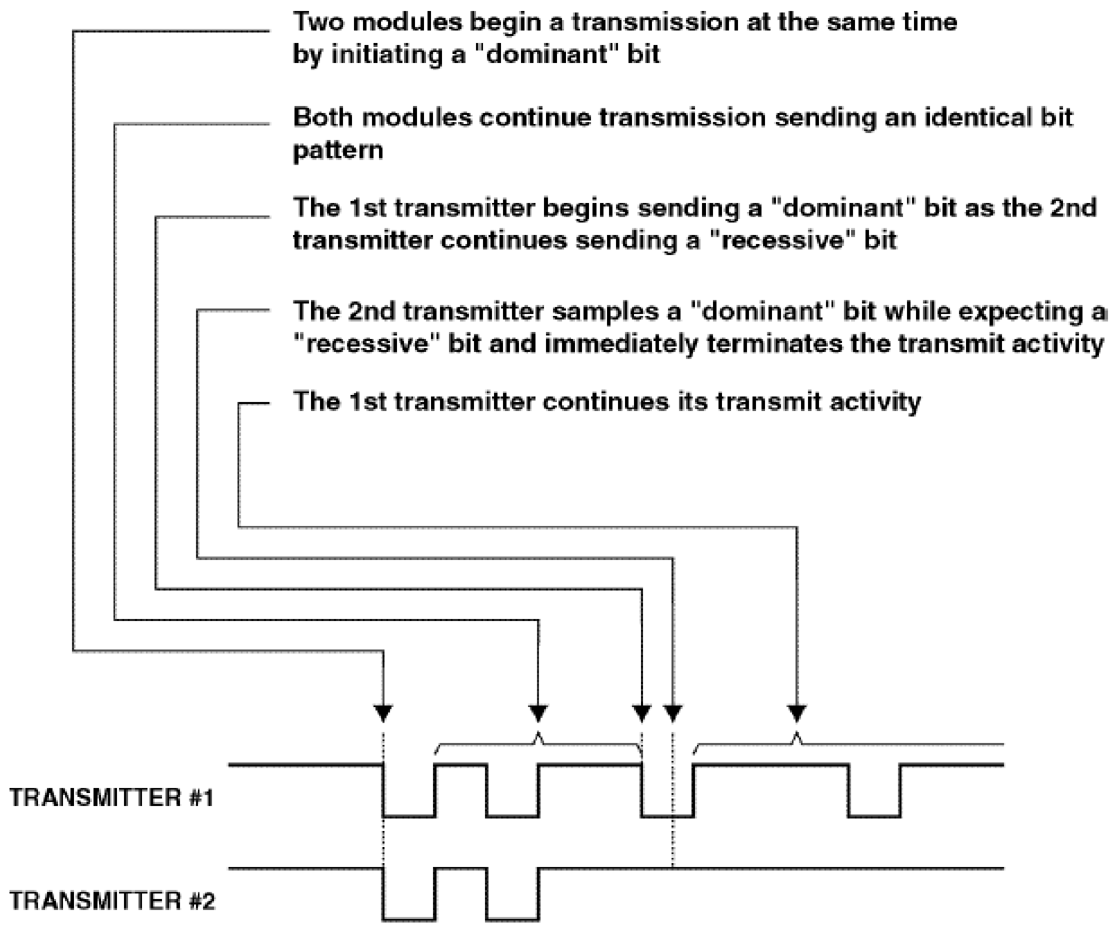
- Any potential bus conflicts are resolved by **bitwise arbitration**
- **Dominant state (logic 0)** has **precedence** over a **recessive state (logic 1)**



Competition for the bus is won by node 2

- Nodes 0 and 1 automatically become receivers of the message
- Nodes 0 and 1 will re-transmit their messages when the bus becomes available again

Example of Bitwise Arbitration



Two modules begin a transmission at the same time by initiating a "dominant" bit
Both modules continue transmission sending an identical bit pattern
The 1st transmitter begins sending a "dominant" bit as the 2nd transmitter continues sending a "recessive" bit
The 2nd transmitter samples a "dominant" bit while expecting a "recessive" bit and immediately terminates the transmit activity
The 1st transmitter continues its transmit activity
TRANSMITTER #1
TRANSMITTER #2

Qualities: Safe Collision and TX-feedback

The CAN-controller has 2 important features:

- A **collision** do not destroy any message on the bus
  - All Tx's with recessive levels stops immediately and changes to Rx's.
  - The Tx-node with highest priority wins the bus and sends data
- Every **transmitted** message is **evaluated** by each receiving node Rx, and if the received message is damaged the Tx-node is alerted with a **feedback** at dominant level, sent from the Rx-node (by **sending an error frame**)

Error Detection

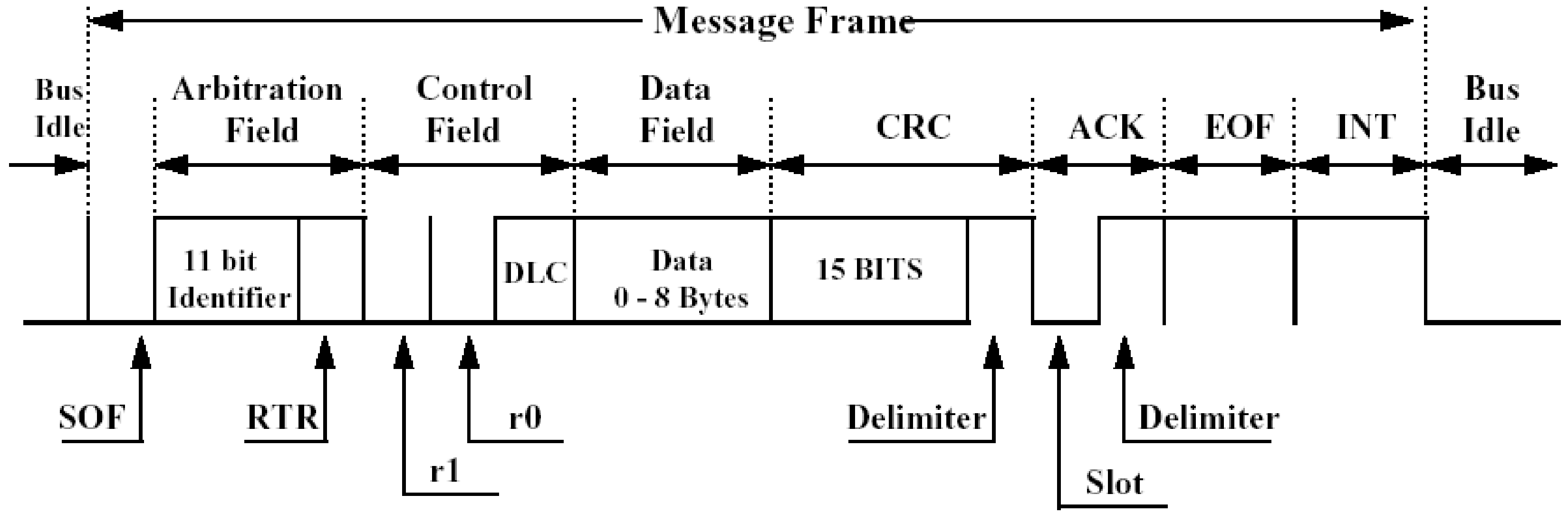
- CAN implements five error detection mechanisms
- Three at the **message level**
  - Cyclic Redundancy Checks (CRC)
  - Frame Checks
  - Acknowledgment Error Checks
- Two at **the bit level**
  - Bit Monitoring
  - Bit Stuffing

Cyclic Redundancy Check (CRC) (Message Level)

- The 15 bit CRC is computed by the **transmitter** based on the message content
- All **receivers** that accept the message, recalculates the CRC and compares against the received CRC
- If the two values do not match a **CRC error** is flagged by sending an **Error frame**

Frame Check (Message Level)

- If a receiver detects an invalid bit in one of these positions a **Form Error** (or Format Error) will be flagged:
  - CRC Delimiter
  - ACK Delimiter
  - End of Frame Bit Field
  - Interframe Space (the 3 bit INTermission field and a possible Bus Idle time)



ACK Error Check (Message Level)

- Each receiving node writes a dominant bit into the **ACK slot**
- If a transmitter determines that a message has not been ACKnowledged then an **ACK Error** is flagged.
- **ACK errors** may occur because of transmission errors because the ACK field has been corrupted or there is no operational receivers

Bit Monitoring (Bit Level)

- Each bit level (dominant or recessive) on the bus **is monitored by the transmitting node**
  - Bit monitoring **is not performed during arbitration** or in **the ACK Slot**

Bit Stuffing (Bit Level)

- Bit stuffing is used to guarantee enough edges in the **NRZ** bit stream to maintain synchronization:
  - After five identical and consecutive bit levels have been transmitted, **the transmitter will automatically inject (stuff) a bit** of the opposite polarity into the bit stream
  - Receivers of the message will automatically **delete (destuff) such bits**
  - If any node detects six consecutive bits of the same level, **a stuff error is flagged**

Error Flag/Frame

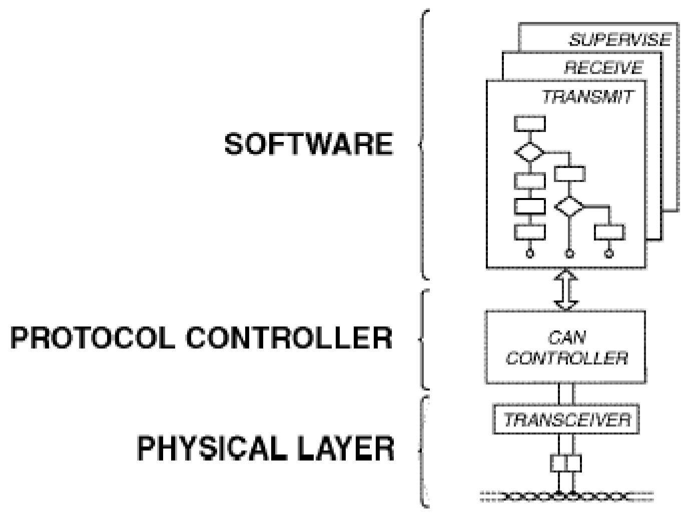
- If an error is detected by at least one node
  - The node that detects the error will immediately abort the transmission by sending an **Error Frame**
- An **Error Flag** consists of **six dominant bits**
  - This violates the bit stuffing rule and all other nodes respond by also transmitting Error Frames

Fault Confinement

- Every node can be in one of three states
  - Error Active
  - Error Passive
  - Bus-off
- An **Error active** node takes part in bus communication and send active error flags, when it detects an error
- An **Error passive** node can’t send error flags
- A **Bus-off** node is not allowed to have any information send on the bus

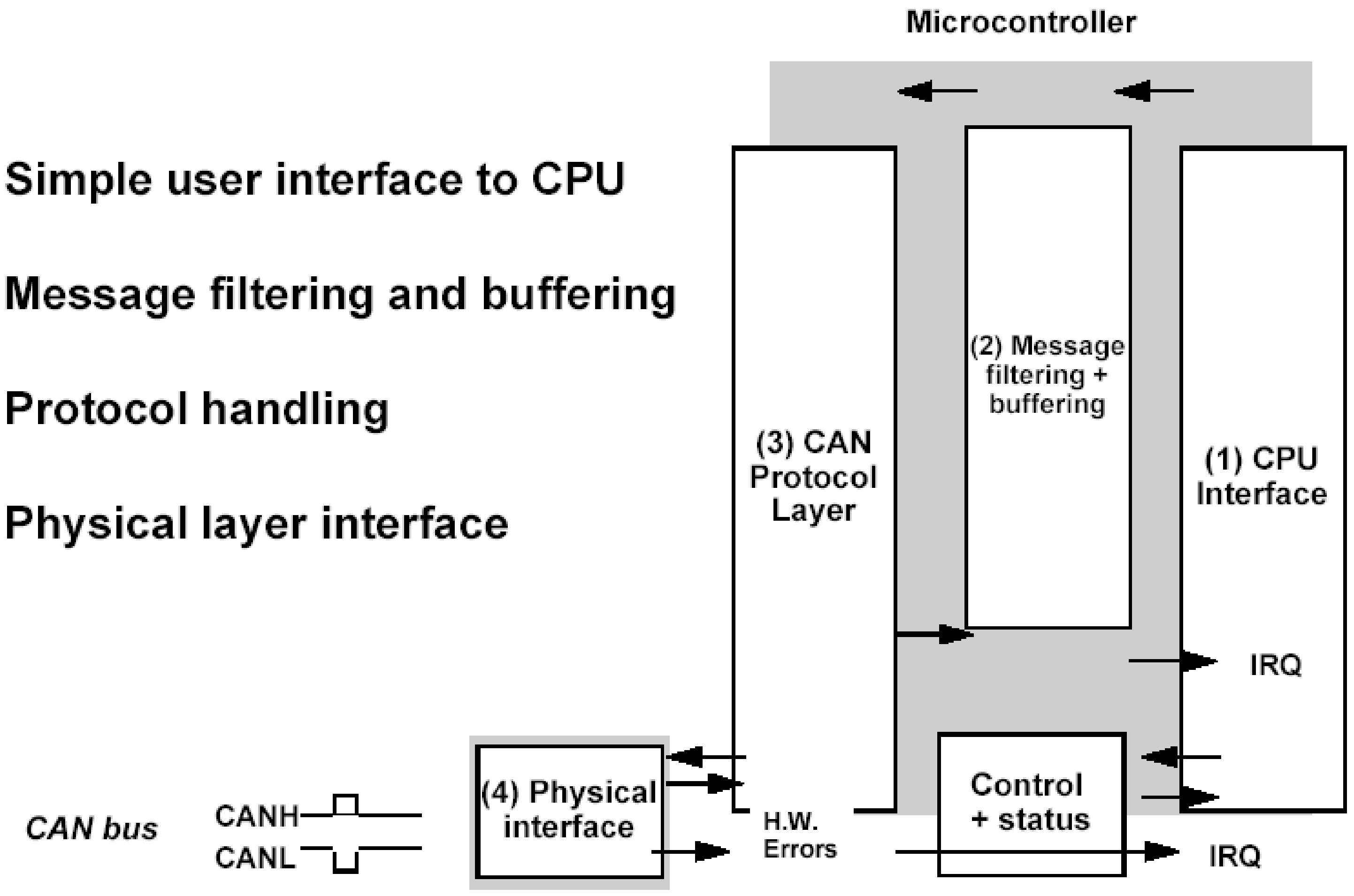
What is needed to implement CAN?

To implement CAN, three components are required - software, a CAN controller, and a physical layer



Requirements for a CAN Controller

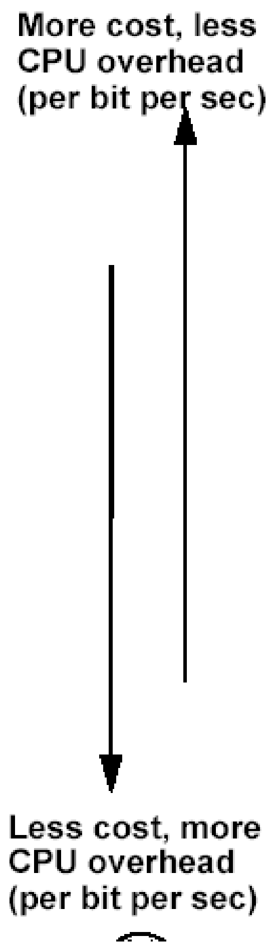
- Simple user interface to CPU
- Message filtering and buffering
- Protocol handling
- Physical layer interface



FullCAN vs BasicCAN Controller

- **FullCAN Controller**:
  - Typically 16 message buffers, sometimes more
  - Global and Dedicated Message Filtering Masks
  - Dedicated H/W for Reducing CPU Workload
  - More Silicon => more cost
    - e.g. Powertrain

- **BasicCAN Controller**:
  - 1 or 2 Tx and Rx buffers
  - Minimal Filtering
  - More Software Intervention
  - Low cost
    - e.g. Car Body


More cost, less CPU overhead (per bit per sec)
Less cost, more CPU overhead (per bit per sec)

CAN Summary

- CAN is designed for **asynchronous** communication (**event communication**) with little information contents (**8 bytes**)
- Max **1MBit/s**
- Useful for soft real-time systems
- Many microcontrollers comes with an integrated CAN controller
- A low-cost solution
- New invention:
  - TTCAN – a Time-Triggered CAN protocol

References

- [Etschberger]:"**Controller Area Network** – basics, protocols, chips and applications",
  by Konrad Etschberger, IXXAT Press, 2001
- CAN Specification 2.0 (2.0b) – Bosch 1991, 1997.
- www.can.bosch.com
  - Contains specification documents
  - References
  - Links
- ODVA: Open DeviceNet Vendors Association www.odva.org
- CiA: CAN in Automation: http://www.can-cia.org/