

14-760:
**ADVANCED REAL-WORLD
NETWORKS**

LECTURE 17 * SPRING 2019 * KESDEN

SERIAL COMMUNICATION

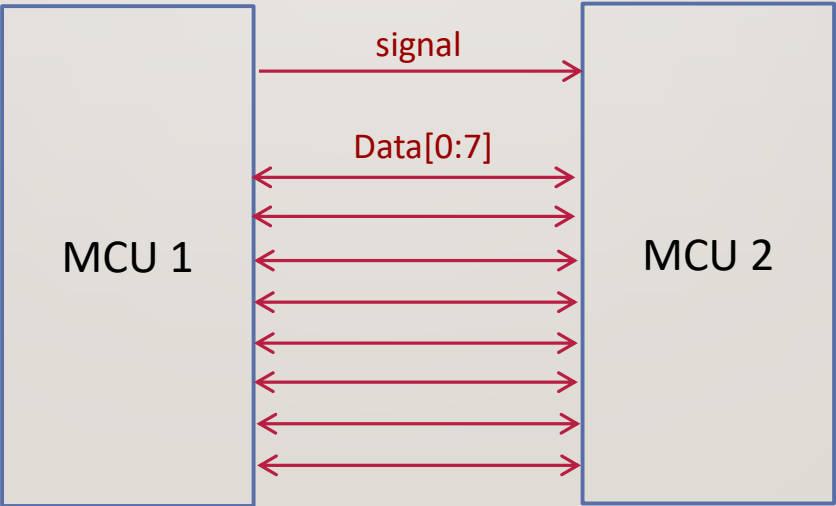
Courtesy 18-349

SERIAL VS. PARALLEL

Serial



Parallel

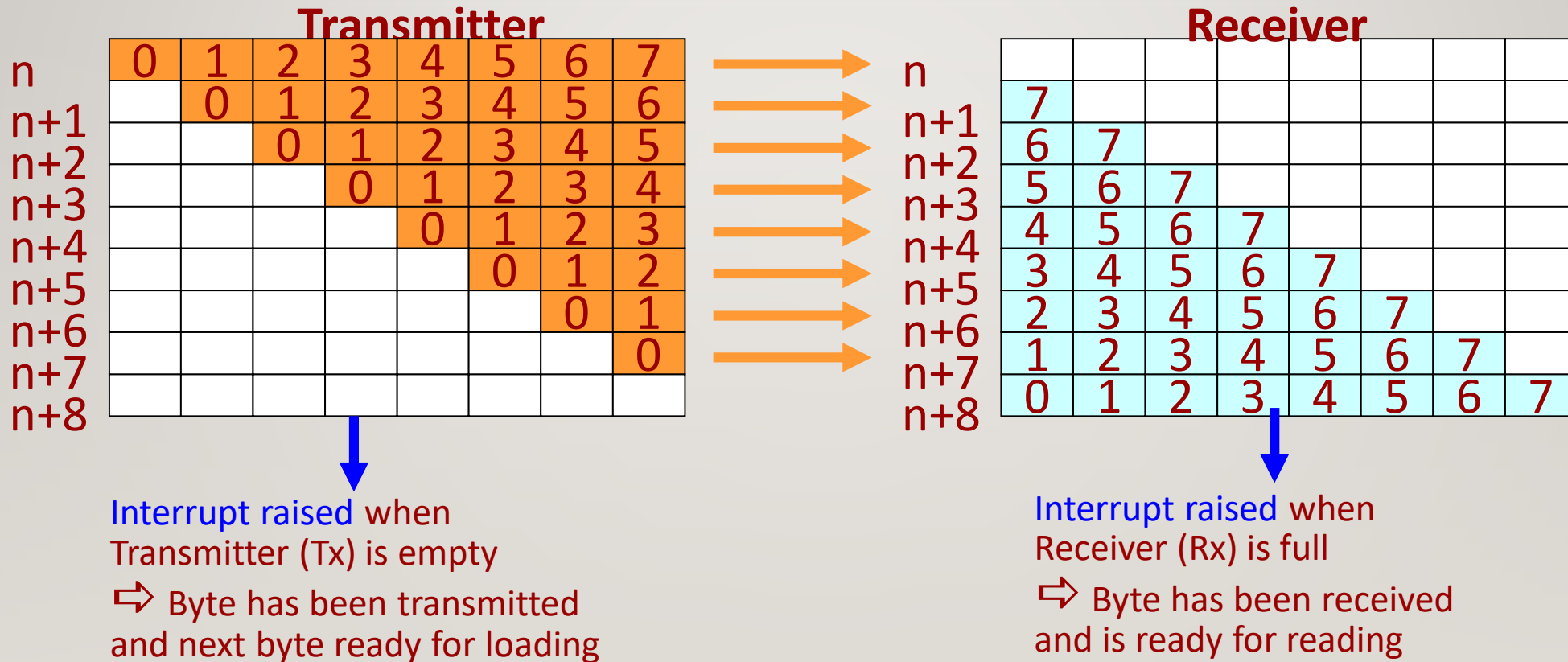


WHY SERIAL COMMUNICATION?

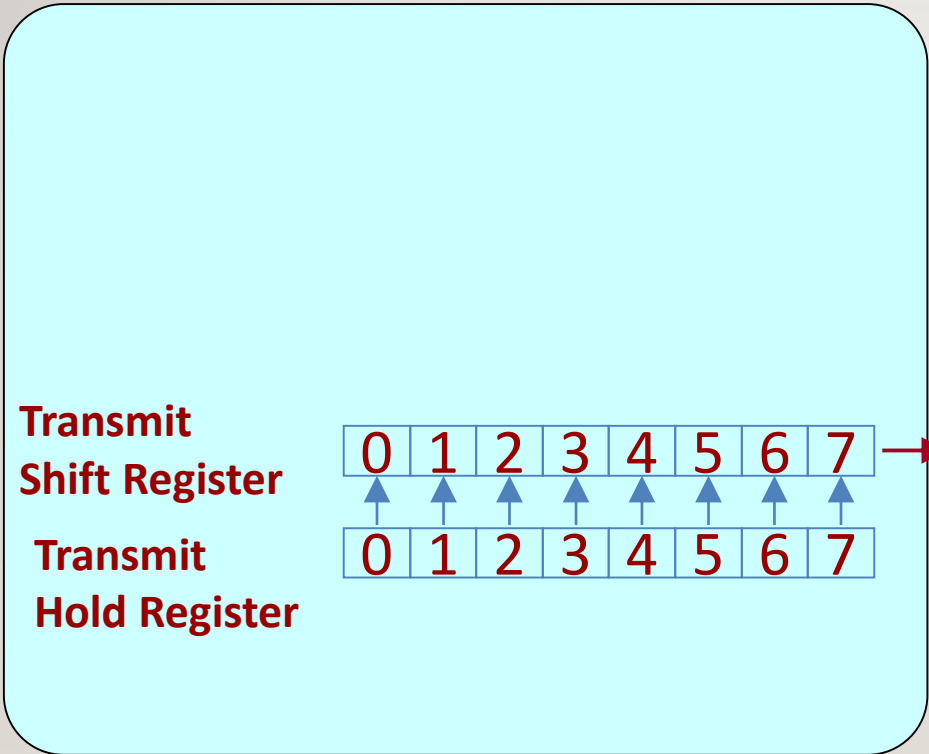
4

-
- Serial communication is a **pin-efficient** way of sending and receiving bits of data
 - Sends and receives data one bit at a time over one wire
 - While it takes eight times as long to transfer each byte of data this way (as compared to parallel communication), only a few wires are required
 - Typically one to send, one to receive (for full-duplex), and a common signal ground wire
 - **Simplistic** way to visualize serial port
 - Two 8-bit shift registers connected together
 - Output of one shift register (transmitter) connected to the input of the other shift register (receiver)
 - Common clock so that as a bit exits the transmitting shift register, the bit enters the receiving shift register
 - Data rate depends on clock frequency

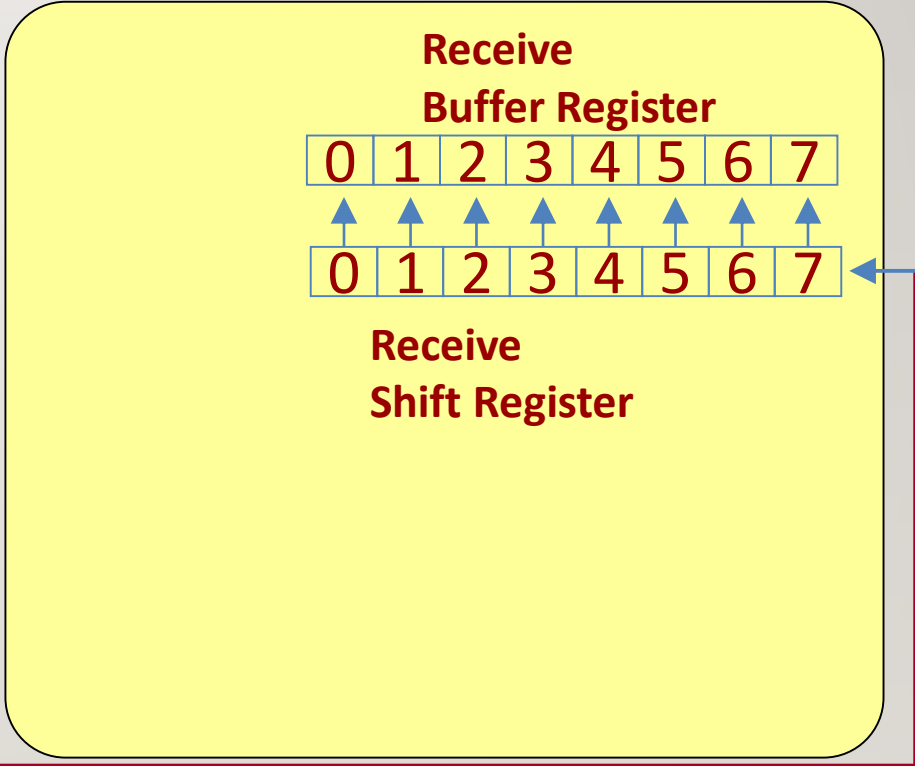
SIMPLISTIC VIEW OF SERIAL PORT OPERATION



SIMPLE SERIAL PORT



Processor



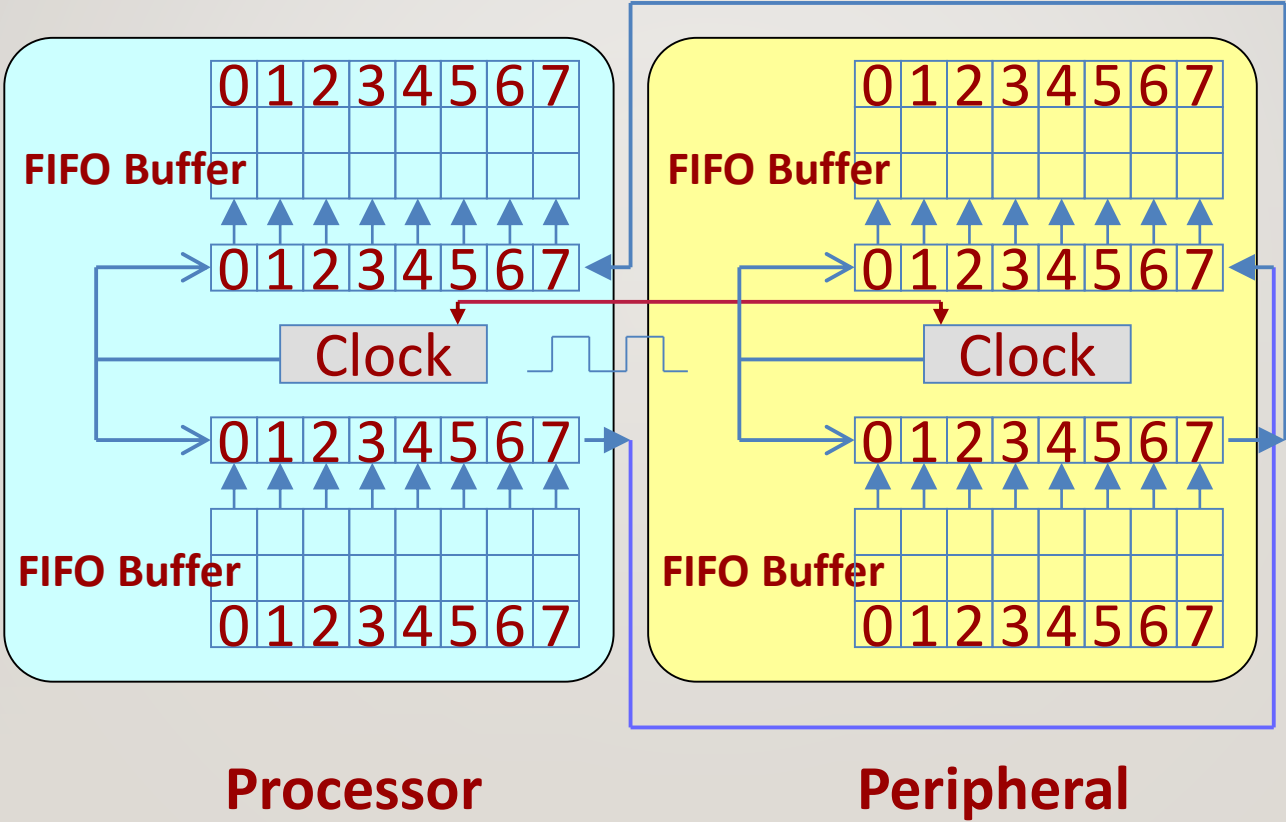
Peripheral

PROTECTING AGAINST DATA LOSS

7

-
- How can data be lost?
 - If the transmitter starts to send the next byte before the receiver has had a chance to process/read the current byte
 - If the next byte is loaded at the transmitter end before the current byte has been completely transmitted
 - Most serial ports use FIFO buffers so that data is not lost
 - Buffering of received bytes at receiver end for later processing
 - Buffering of loaded bytes at transmitter end for later transmission
 - Shift registers free to transmit and receive data without worrying about data loss
 - Why does the size of the FIFO buffers matter?

SERIAL PORT



TYPES OF SERIAL COMMUNICATIONS

9

-
- **Synchronous communication**
 - Data transmitted as a steady stream at regular intervals
 - All transmitted bits are synchronized to a common clock signal
 - The two devices initially synchronize themselves to each other, and then continually send characters to stay synchronized
 - Faster data transfer rates than asynchronous methods, because it does not require additional bits to mark the beginning and end of each data byte
 - **Asynchronous communication**
 - Data transmitted intermittently at irregular intervals
 - Each device uses its own internal clock resulting in bytes that are transferred at arbitrary times
 - Instead of using time as a way to synchronize the bits, the data format is used
 - Data transmission is synchronized using the start bit of the word, while one or more stop bits indicate the end of the word
 - Asynchronous communications slightly slower than synchronous

SYNC VS. ASYNC

- Synchronous communications
 - Requires common clock (SPI)
 - Whoever controls the clock controls communication speed
- Asynchronous communications
 - Has no clock (UART)
 - Speed must be agreed upon beforehand (the baud-rate configuration accomplishes that)

WHAT IS RS-232?

11

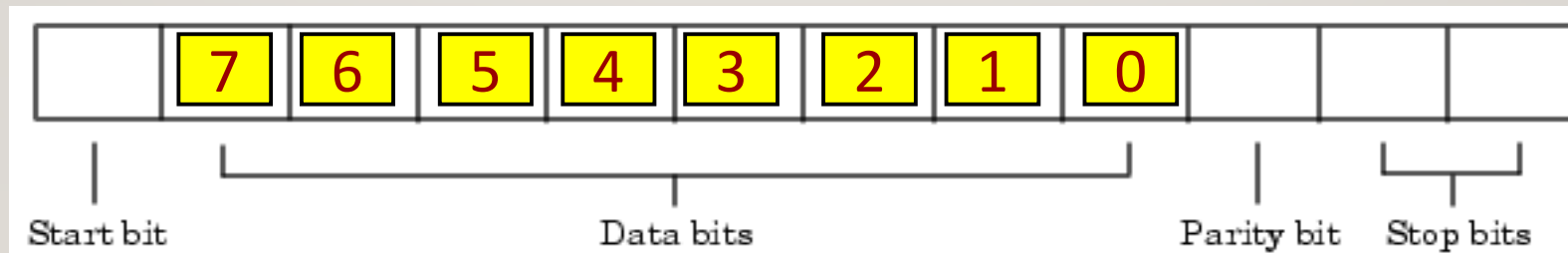
-
- So far, we've talked about clocks being synchronized and using the clock as a reference for data transmission
 - Fine for short distances (e.g., within chips on the same board)
 - When data is transmitted over longer distances (off-chip), voltage levels can be affected by cable capacitance
 - A logic "1" might appear as an indeterminate voltage at the receiver
 - Wrong data might be accepted when clock edges become skewed
 - Enter RS232: Recommended Standard number 232
 - Serial ports for longer distances, typically, between PC and peripheral
 - Data transmitted asynchronously, i.e., no reference clock
 - Data provides its own reference clock

RS-232

Courtesy 18-349


RS232 – BITS AND SERIAL BYTES

- Serial ports on IBM-style PCs support asynchronous communication only
- A “serial byte” usually consists of
 - *Characters*: 5-8 data bits
 - *Framing bits*: 1 start bit, 1 parity bit (optional), 1-2 stop bits
 - When serial data is stored on your computer, framing bits are removed, and this looks like a real 8-bit byte
- Specified as number of data bits - parity type - number of stop bits
 - 8-N-1 a eight data bits, no parity bit, and one stop bit
 - 7-E-2 a seven data bits, even parity, and two stop bits



PARITY BITS

14

-
- Simple error checking for the transmitted data
 - Even parity
 - The data bits produce an even number of 1s
 - Odd parity
 - The data bits produce an odd number of 1s
 - Parity checking process
 1. The transmitting device sets the parity bit to 0 or to 1 depending on the data bit values and the type of parity checking selected.
 2. The receiving device checks if the parity bit is consistent with the transmitted data; depending on the result, error/success is returned
 - Disadvantage
 - Parity checking can detect only **an odd number of bit-flip errors**
 - Multiple-bit errors can appear as valid data
- 

PARITY EXAMPLE

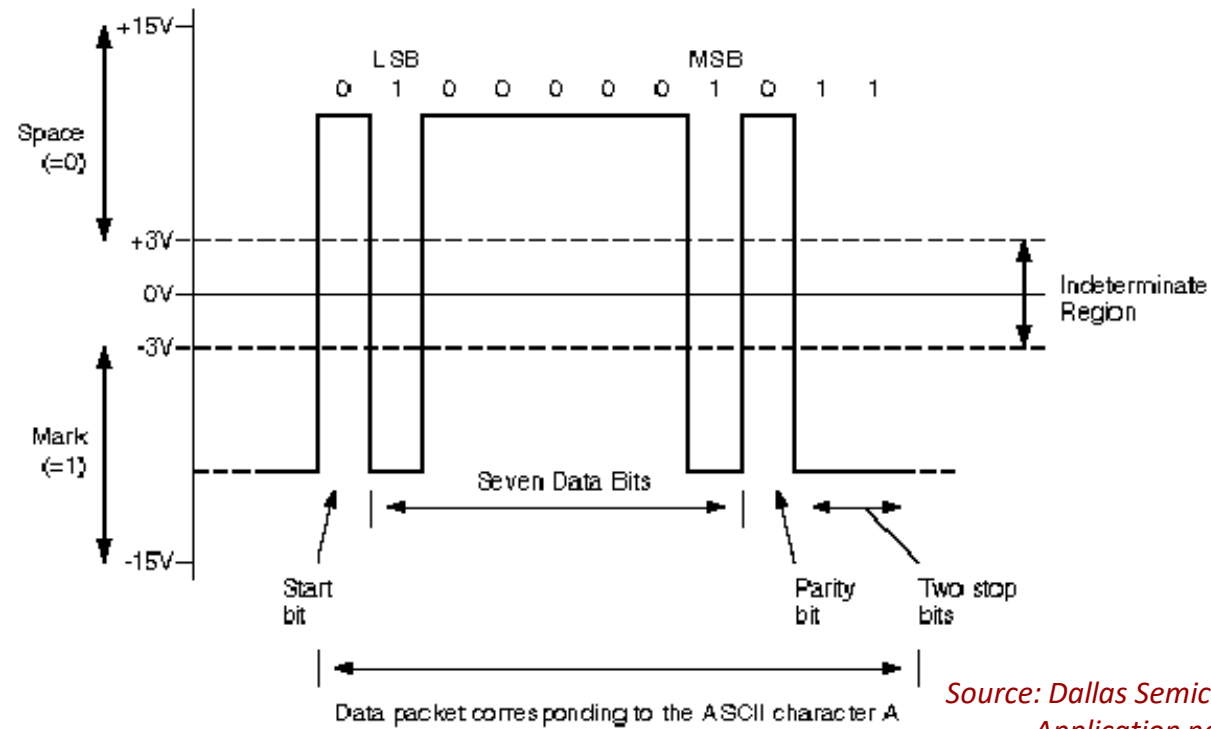
7 bits of data	(count of 1 bits)	8 bits including parity	
		even	odd
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

DATA MODULATION

16

-
- When sending data over serial lines, logic signals are converted into a form the physical media (wires) can support
 - **RS232C** uses bipolar pulses
 - Any signal greater than +3 volts is considered a space (0)
 - Any signal less than -3 volts is considered a mark (1)
 - Conventions
 - Idle line is assumed to be in high (1) state
 - Each character begins with a zero (0) bit, followed by 5-8 data bits and then 1, 1 1/2, or 2 closing stop bits
 - Bits are usually encoded using ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)

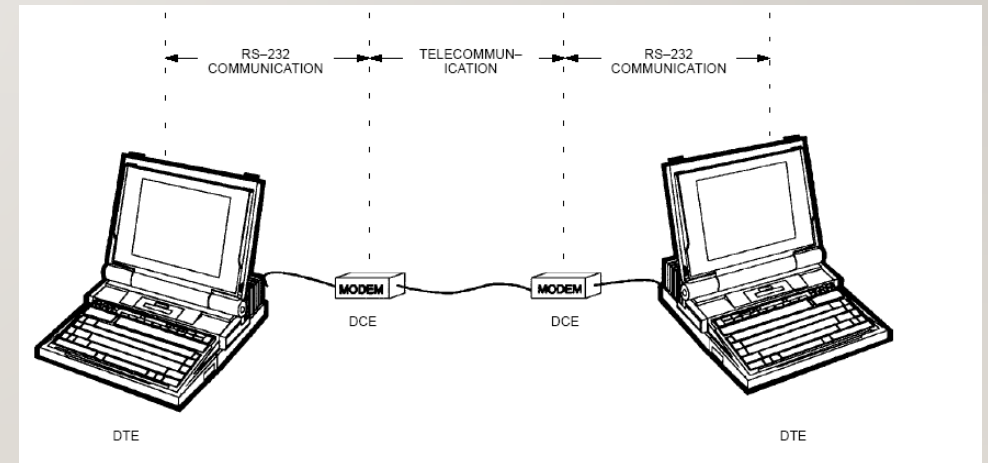
RS-232 SIGNAL LEVELS



TERMINOLOGY

18

- DTE: Data terminal equipment, e.g., PC
- DCE: Data communication equipment, e.g., modem, remote device
- Baud Rate
 - Maximum number of times per second that a line changes state
 - Not always the same as bits per second



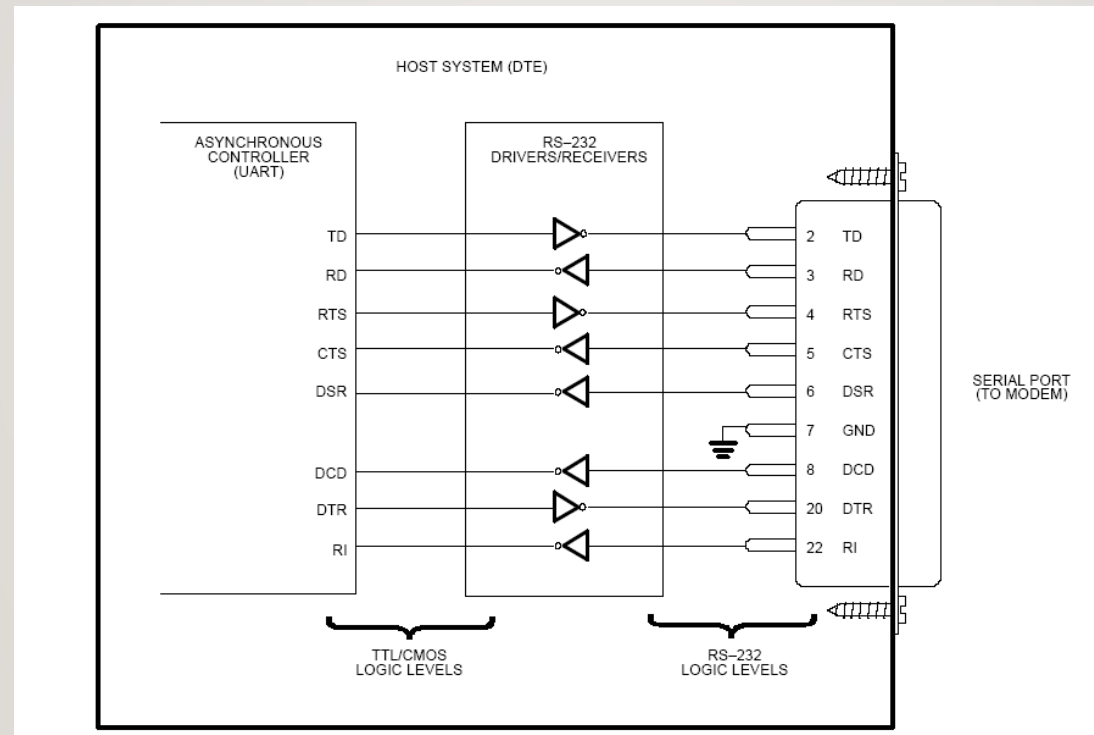
SERIAL PORT CONNECTOR

19

- 9-pin (DB-9) or 25-pin (DB-25) connector
- Inside a 9-pin connector
 - **Carrier Detect** - Determines if the DCE is connected to a working phone line
 - **Receive Data** - Computer receives information sent from the DCE
 - **Transmit Data** - Computer sends information to the DCE
 - **Data Terminal Ready** - Computer tells the DCE that it is ready to talk
 - **Signal Ground** - Pin is grounded
 - **Data Set Ready** - DCE tells the computer that it is ready to talk
 - **Request To Send** - Computer asks the DCE if it can send information
 - **Clear To Send** - DCE tells the computer that it can send information
 - **Ring Indicator** – Asserted when a connected modem has detected an incoming call
- What's a null modem cable?




RS-232 PIN CONNECTIONS



HANDSHAKING

21

-
- Some RS232 connections using handshaking lines between DCE and DTE
 - RTS (ReadyToSend)
 - Sent by the DTE to signal the DCE it is Ready To Send
 - CTS (ClearToSend)
 - Sent by the DCE to signal the DTE that it is Ready to Receive
 - DTR (DataTerminalReady)
 - Sent to DTE to inform the DCE that it is ready to connect
 - DSR (DataSetRead)
 - Sent to DCE to inform the DTE that it is ready to connect
 - Handshaking lines can make it difficult to set up the serial communications, but seamless after set-up.
 - Also, software handshaking (XON/XOFF)
- 

SERIAL DATA COMMUNICATION MODES

22



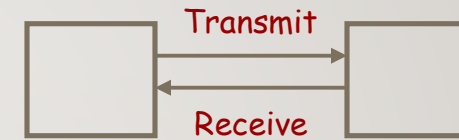
Simplex Mode

Transmission is possible only in one direction.



Half-duplex Mode

Data is transmitted in one direction at a time but the direction can be changed.



Full-duplex Mode

Data may be transmitted simultaneously in both directions.

FLOW CONTROL

23

-
- Necessary to prevent terminal from sending more data than the peripheral can consume (and vice-versa)
 - Higher data rates can result in missing characters (data-overflow errors)
 - Hardware handshaking
 - Hardware in UART detects a potential overrun and asserts a handshake line to prevent the other side from transmitting
 - When receiving side can take more data, it releases the handshake line
 - Software flow-control
 - Special characters XON and XOFF
 - XOFF stops a data transfer (control-S or ASCII code 13)
 - XON restarts the data transfer (control-Q or ASCII code 11)
 - Assumption is made that the flow-control becomes effective before data loss happens

LIMITATIONS

- Not useful for long distances
 - Cable capacitance is an issue, e.g. 50pF/ft
- Ground reference is poor
 - Cross-talk between parallel wires
 - Biasing
 - Little ability to reject outside noise
- Point-to-point
 - Not standard for multi-drop receivers
 - No standard for multiple senders
- Multi-point work-arounds (Non-standard)
 - Daisy chain receivers
 - Buffered multiplexer for multiple senders

SEATALK



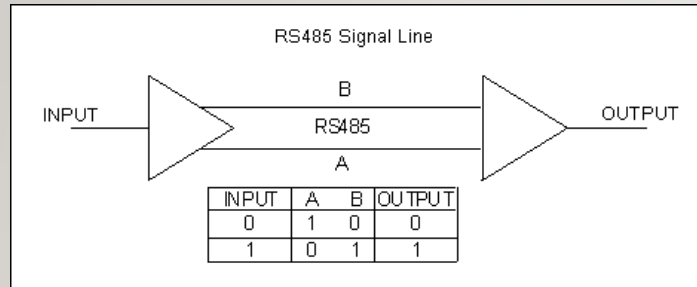
SEATALK-I

- Example proprietary serial line solution
- Not compatible with RS-232
- 4800,N,8,1
 - Parity bit is actually used to indicate first byte of each datagram, not parity
- All participants are symmetric
 - Listen for collision, if quiet, transmit
 - Listen while transmitting.
 - If garbled stop and retransmit after delay.
 - Delay is random, but greater for lower priority devices.
 - If a datagram is short, it is garbled.
 - Receiver drops it.

RS-422

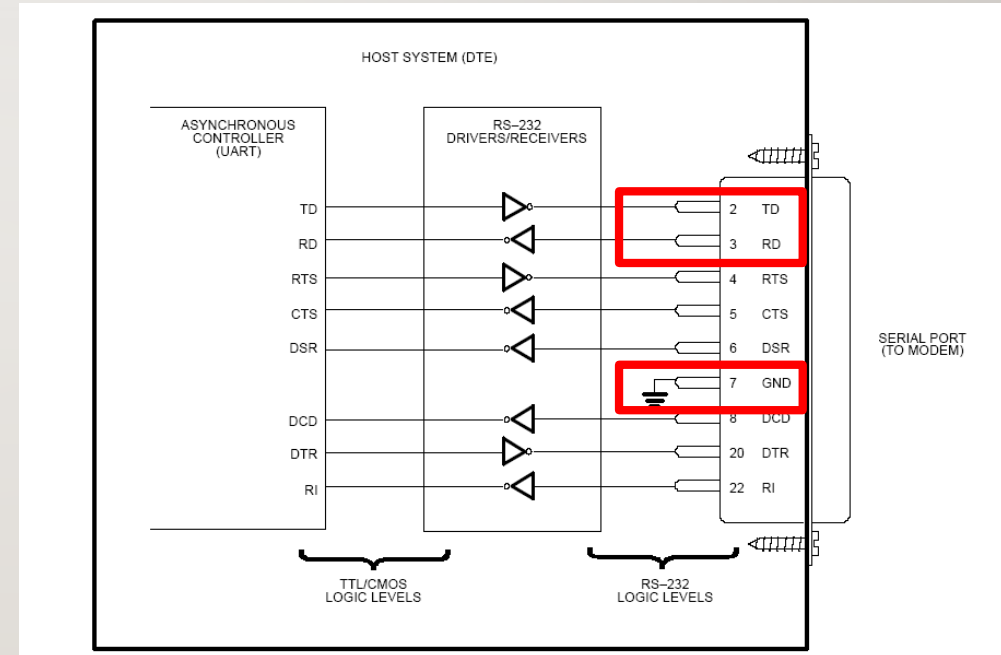


DIFFERENTIAL ENCODING: NOT POSITIVE/NEGATIVE



<http://www.windmill.co.uk/rs485.html>

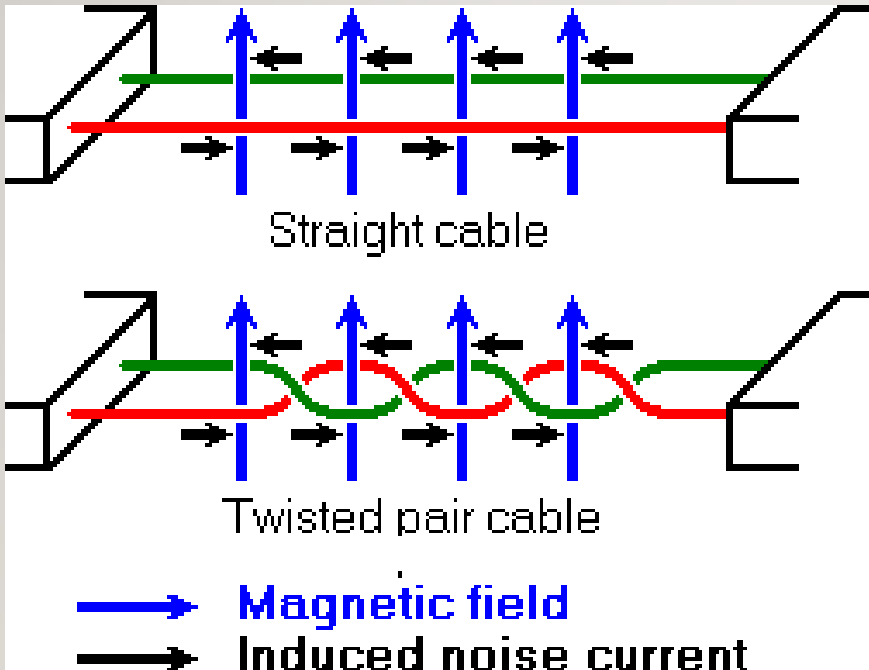
- Notice that in RS-232 TX and RX are relative to the ground
- Notice that in RS-422:
 - There are not TD and RD, just A and B
 - A and B are relative to each other



DIFFERENTIAL ENCODING: WHY BETTER

- Not dependent upon quality of ground
- Can't get a ground loop due to differences
 - Current flowing through ground
- Both A and B are driven, not floating

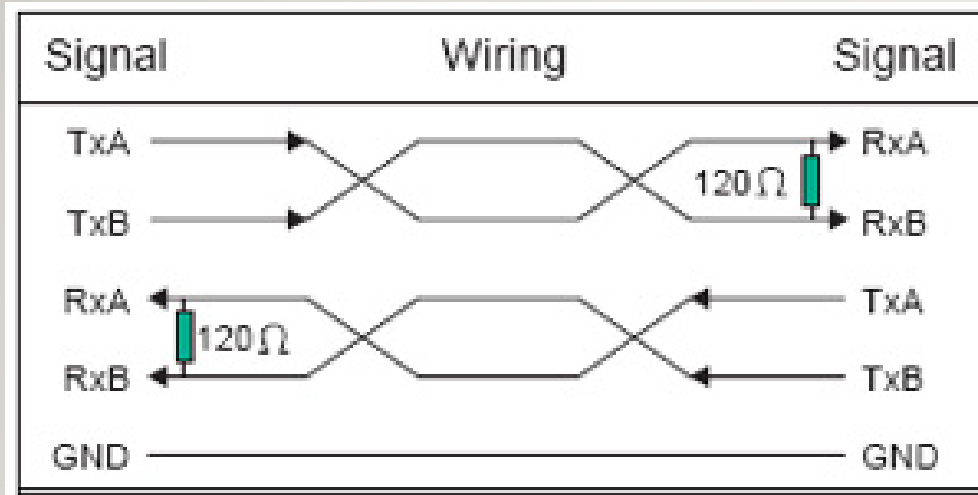
WIRING: STRAIGHT CABLE VS TWISTED PAIR



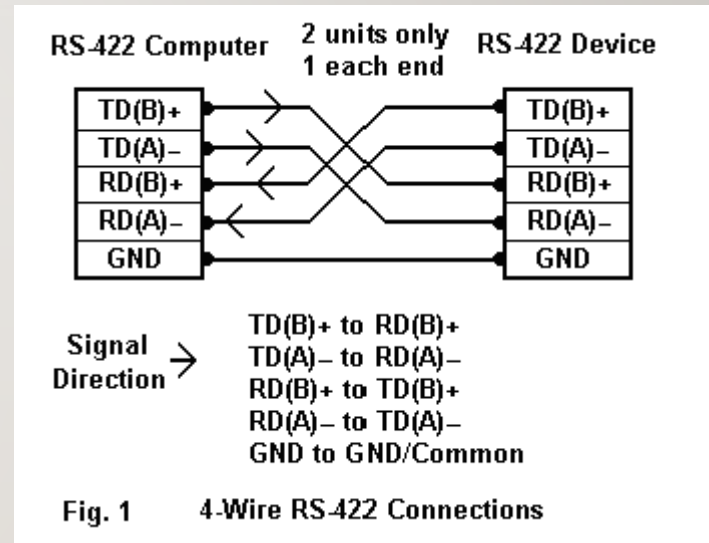
Twisted pair (ex, RS-422) allows much higher data rates than straight run (ex., RS-232).

<https://www.root.cz/clanky/sbernice-rs-422-rs-423-a-rs-485/>

SINGLE DROP WIRING

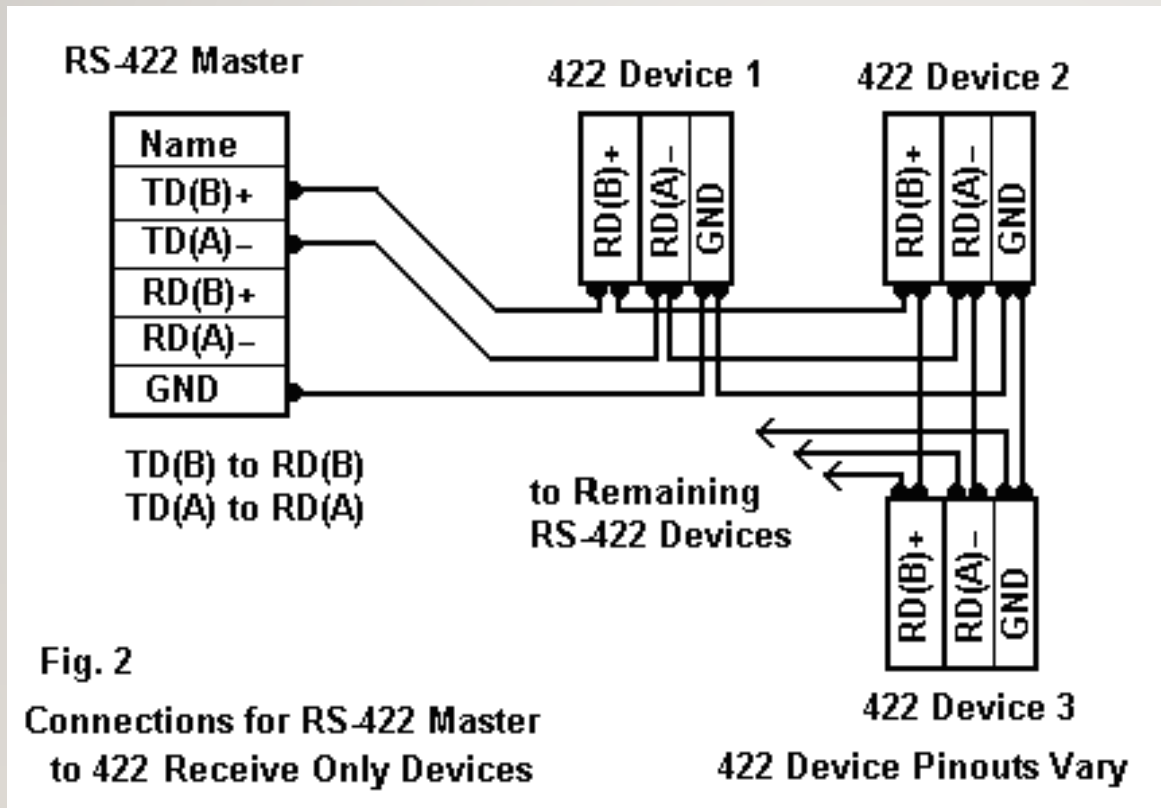


<https://www.root.cz/clanky/sbernice-rs-422-rs-423-a-rs-485/>



<http://www.bb-elec.com/Learning-Center/All-White-Papers/Serial/FAQ-Connect-RS-422-Devices.aspx>

MULTI-DROP WIRING

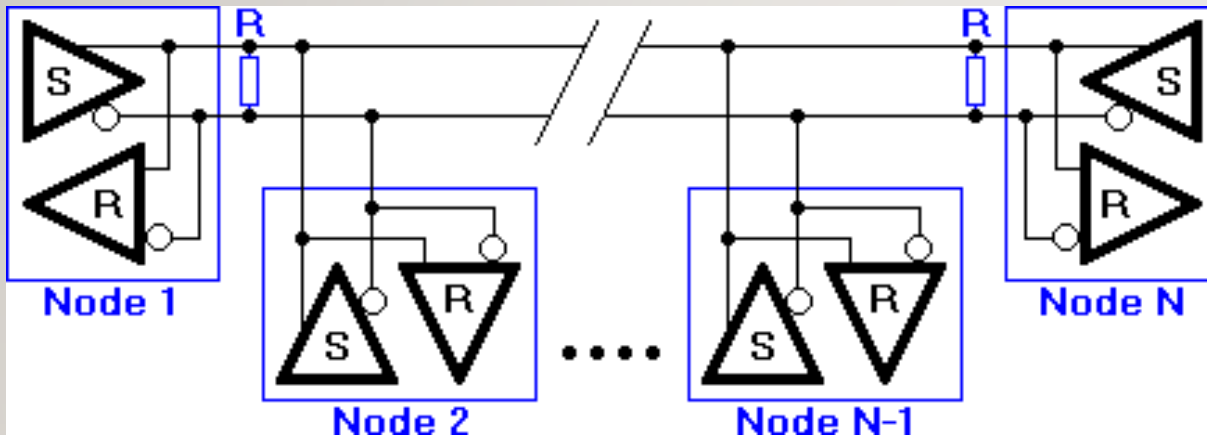


- Multi-**Drop**?
 - What Seems to be missing here?
 - (Hint: One way, master to slaves)
- Addressing
 - Can be added at higher level
 - Ignore message if not intended recipient

RS-485



MULTI-DROP



<https://www.lammertbies.nl/comm/info/RS-485.html#topo>

- Like a multi-point RS-422
- Differential signaling
- Twisted Pair
- 32-256 devices
 - Depends on impedance and value of R (ohms)
- RS-422 doesn't describe addressing, collision management, etc.
 - Left for a higher layer

MODBUS

<https://support.automationdirect.com/docs/an-misc-027.ppt>

ASCII VS MTU

The MODBUS protocol comes in 2 versions :

- ASCII transmission mode

Each eight-bit byte in a message is sent as 2 ASCII characters.

- RTU transmission mode

Each eight-bit byte in a message is sent as two four-bit hexadecimal characters.

The main advantage of the RTU mode is that it achieves higher throughput.

ASCII mode allows time intervals of up to 1 second to occur between characters without causing an error.

FRAME STRUCTURE

The Modbus frame structure is the same for requests (master to slave messages) and responses (slave to master messages).

Modbus RTU



Silence \geq 3.5 characters

Modbus ASCII

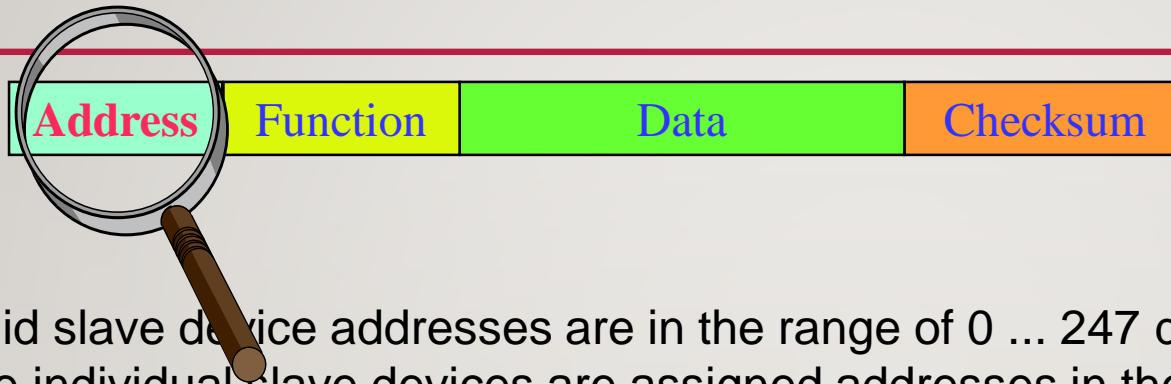


3A Hex

0D Hex

0A Hex

FRAME ADDRESS



Valid slave device addresses are in the range of 0 ... 247 decimal.

The individual slave devices are assigned addresses in the range of 1 ... 247.

Value 0 is reserved for broadcast messages (no response).

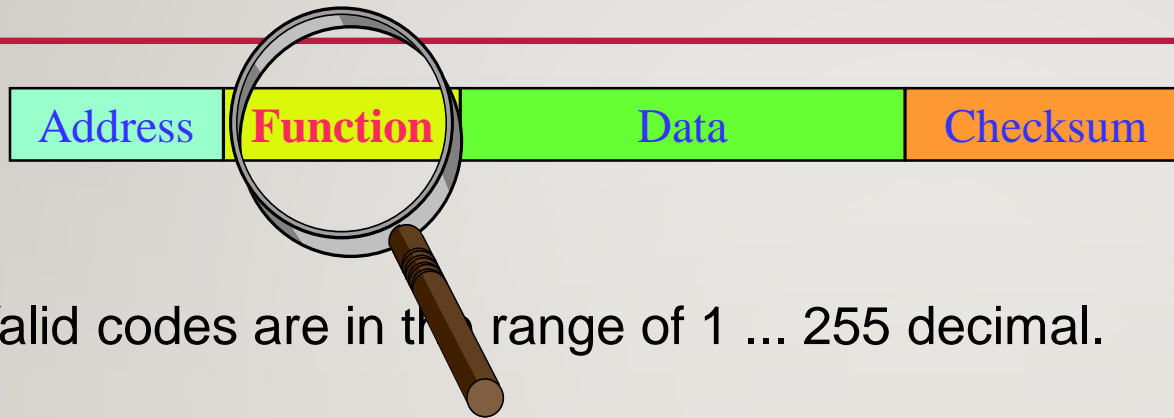
Request :

A master addresses a slave by placing the slave address in the address field of the message.

Response :

When the slave sends its response, it places its own address in this address field of the response to let the master know which slave is responding.

FRAME FUNCTION FIELD



Valid codes are in the range of 1 ... 255 decimal.

Request :

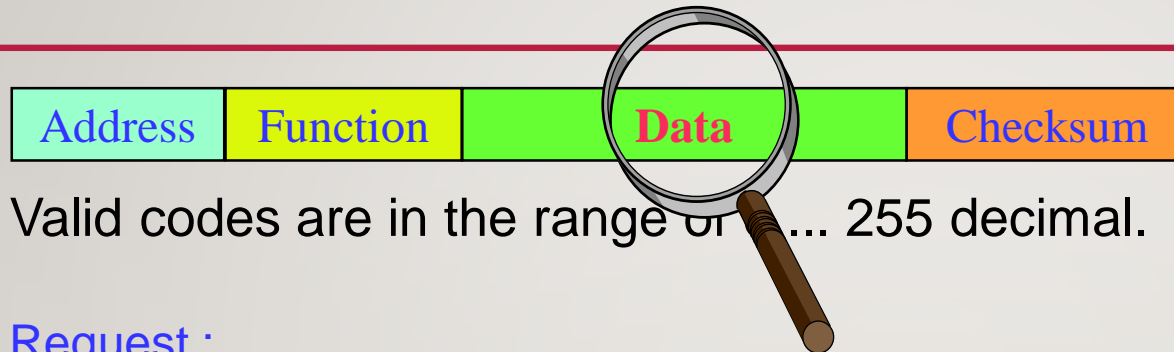
The function code field tells the slave what kind of action to perform.

Response :

For a normal response, the slave simply echoes the original function code.

For an exception response, the slave returns a code that is equivalent to the original function code with its most significant bit set to a logic 1.

DATA FIELD



Valid codes are in the range of 0 ... 255 decimal.

Request :

The data field contains additional information which the slave must use to take the action defined by the function code. This can include items like register addresses, quantity of items to be handled, etc...

Response :

If no error occurs, the data field contains the data requested.

If an error occurs, the field contains an exception code that the master application can use to determine the next action to be taken.

CHECKSUM FIELD



Valid codes are in the range of 0 ... 255 decimal.

Modbus RTU uses CRC : Cyclycal Reduncy Check (2 byte)
Modbus ASCII uses LRC : Longitudinal Redundancy Check (1 bytes)

Request :

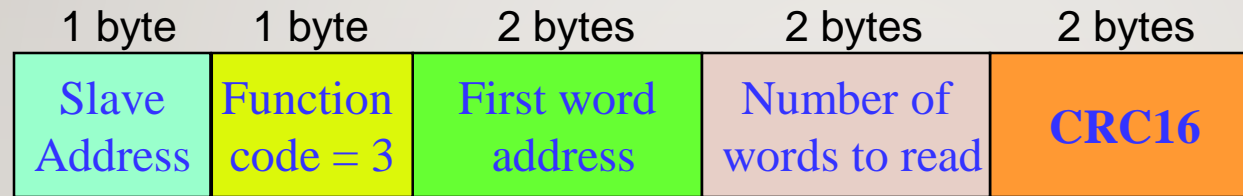
The checksum is calculated by the master and sends to the slave.

Response :

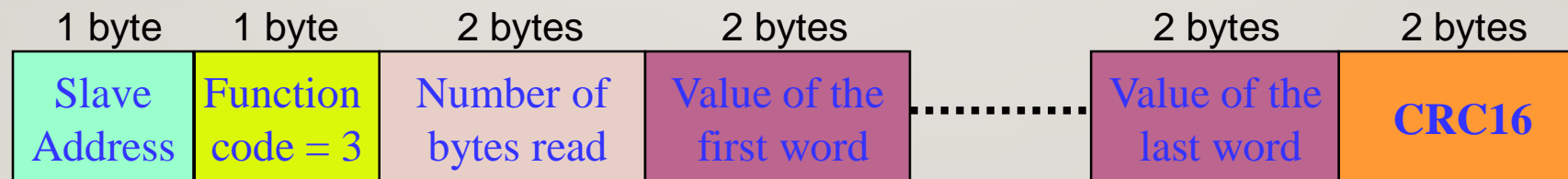
The checksum is re-calculated by the slave and compared to the value sent by the master. If a difference is detected, the slave will not construct a response to the master.

RTU FRAME EXAMPLE

Request :



Response : Function code = 3 : Read n words



EXAMPLE FUNCTION CODES

Code	Type
01	Read n consecutive output bits
02	Read n consecutive input bits
03	Read n consecutive output words
04	Read n consecutive input words
05	Write 1 output bit
06	Write 1 output word
07	Read exception status
08	Access diagnostic counters
11	Read event counter
12	Read connection events
15	Write n output bits
16	Write n output words
17	Read identification

NMEA 0183



OVERVIEW

- National Marine Electronics Association (NMEA)
- RS-422 Based Protocol for interconnecting devices on boats
 - Originally RS-232
 - Changed to RS-422 for multi-drop and length of cable
- What types of data?
 - Depth sounder
 - GPS
 - Speed sensor
 - Temp sensor
 - AIS, etc
- Common applications
 - Collect from devices, unify on single display
 - Unify toward a goal, e.g. GPS, speed and wind to autopilot
- ASCII based data

SERIAL CONFIGURATION

- Commonly
 - 4800, N, 8, 1
- High Speed
 - 38400, N, 8, 1
 - Used for time-sensitive, high-density updates, most commonly AIS
- Remember
 - Multi-**drop**, not multi-point

MULTIPLEXERS



<https://www.navstore.com/actisense-pro-mux-1-nmea-0183-multiplexer.html>

MESSAGE STRUCTURE

ASCII	Hex	Dec	Use
<CR>	0x0d	13	Carriage return
<LF>	0x0a	10	Line feed, end delimiter
!	0x21	33	Start of encapsulation sentence delimiter
\$	0x24	36	Start delimiter
*	0x2a	42	Checksum delimiter
,	0x2c	44	Field delimiter
\	0x5c	92	TAG block delimiter
^	0x5e	94	Code delimiter for HEX representation of ISO/IEC 8859-1 (ASCII) characters
~	0x7e	126	Reserved

[https://en.wikipedia.org/wiki/NMEA_0183#Serial_configuration_\(data_link_layer\)](https://en.wikipedia.org/wiki/NMEA_0183#Serial_configuration_(data_link_layer))

MESSAGE EXAMPLES

```
$GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,19,13,28,070,17,26,23,252,,04,14,186,14*79
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092750.000,A,5321.6802,N,00630.3372,W,0.02,31.66,280511,,,A*43
$GPGGA,092751.000,5321.6802,N,00630.3371,W,1,8,1.03,61.7,M,55.3,M,,*75
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,16,13,28,070,17,26,23,252,,04,14,186,15*77
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092751.000,A,5321.6802,N,00630.3371,W,0.06,31.66,280511,,,A*45
```

[https://en.wikipedia.org/wiki/NMEA_0183#Serial_configuration_\(data_link_layer\)](https://en.wikipedia.org/wiki/NMEA_0183#Serial_configuration_(data_link_layer))

DETAILED MESSAGE EXAMPLE

Global Positioning System Fix Data. Time, Position and fix related data for a GPS receiver

```
.
      1      2      3 4      5 6 7 8 9 10 | 11 12 13 14 15
      |      |      ||      ||| | | | | | | | | |
$--GGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh
1) Time (UTC)
2) Latitude
3) N or S (North or South)
4) Longitude
5) E or W (East or West)
6) GPS Quality Indicator,
0 - fix not available,
1 - GPS fix,
2 - Differential GPS fix
7) Number of satellites in view, 00 - 12
8) Horizontal Dilution of precision
9) Antenna Altitude above/below mean-sea-level (geoid)
10) Units of antenna altitude, meters
11) Geoidal separation, the difference between the WGS-84 earth ellipsoid and mean-sea-level (geoid), "-" means mean-sea-level below
12) Units of geoidal separation, meters
13) Age of differential GPS data, time in seconds since last SC104 type 1 or 9 update, null field when DGPS is not used
14) Differential reference station ID, 0000-1023
15) Checksum
```

<http://freenmea.net/docs>

LIMITATIONS

- Multi-drop, but not multi-point
 - Many devices exchange data versus strictly send or receive it
 - AIS needs to send other boats' info, but receive own boat info to send it out
 - Autopilot needs boat parameters to calculate course, but sends out information about deviation
 - Etc
- Total rats nest of wiring
 - No common bus.
 - It is hard to build a “star” around a multiplexer snaking all sorts of wires through a boat

CAN BUS

<https://teaching.shu.ac.uk/aces/agl/files/ml67/CAN%20BUS.ppt>

OVERVIEW

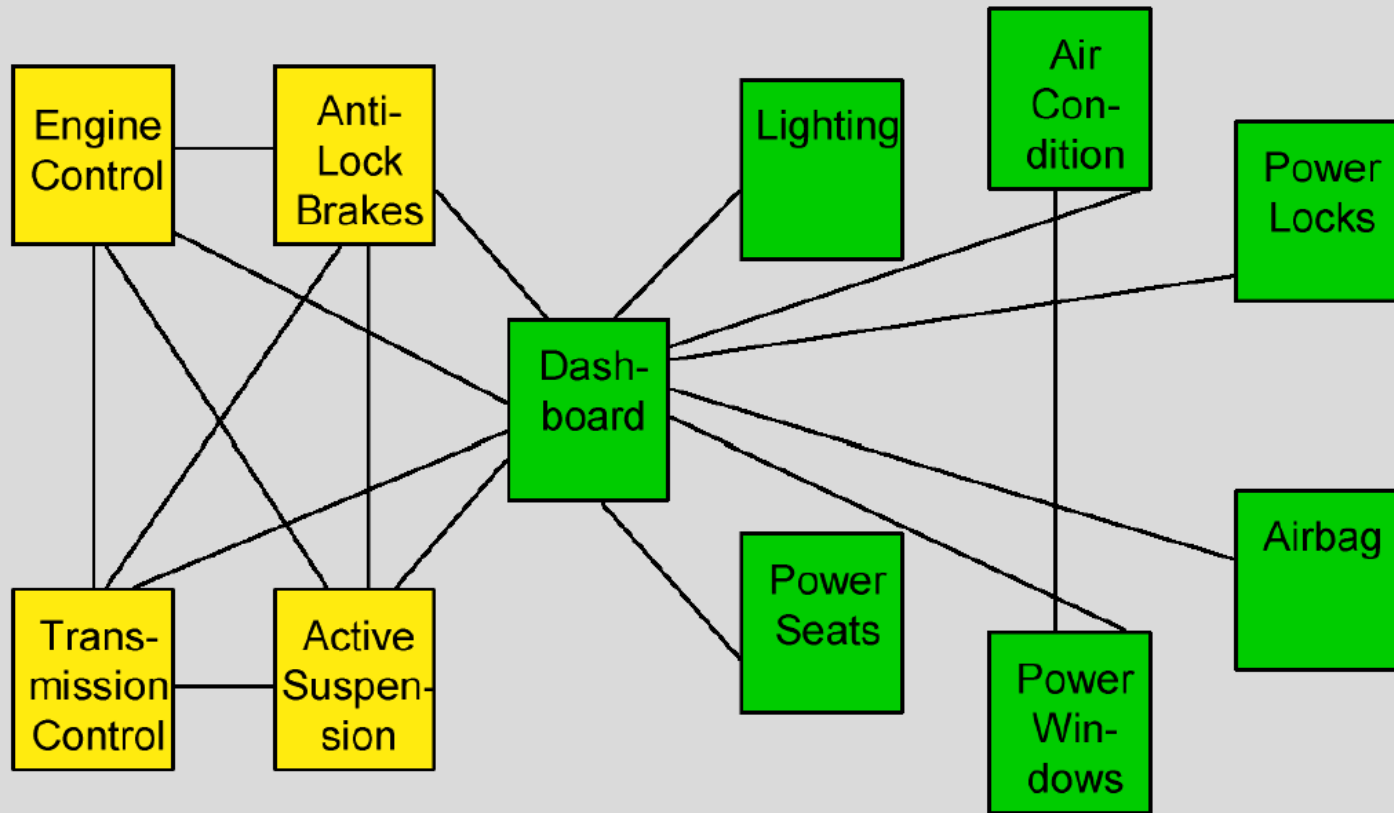
- CAN is an important embedded protocol
- Primarily automotive, but used in many other places
- CAN specifies:
 - Physical layer
 - Protocol layer
 - Message filtering layer (with add-on protocols)
- Note
 - How message prioritization achieved
 - How “small” nodes can be kept from overloading with received messages

THE DEVELOPMENT OF CAN

The development of CAN began when more and more electronic devices were implemented into modern motor vehicles. Examples of such devices include engine management systems, active suspension, ABS, gear control, lighting control, air conditioning, airbags and central locking. All this means more safety and more comfort for the driver and of course a reduction of fuel consumption and exhaust emissions.

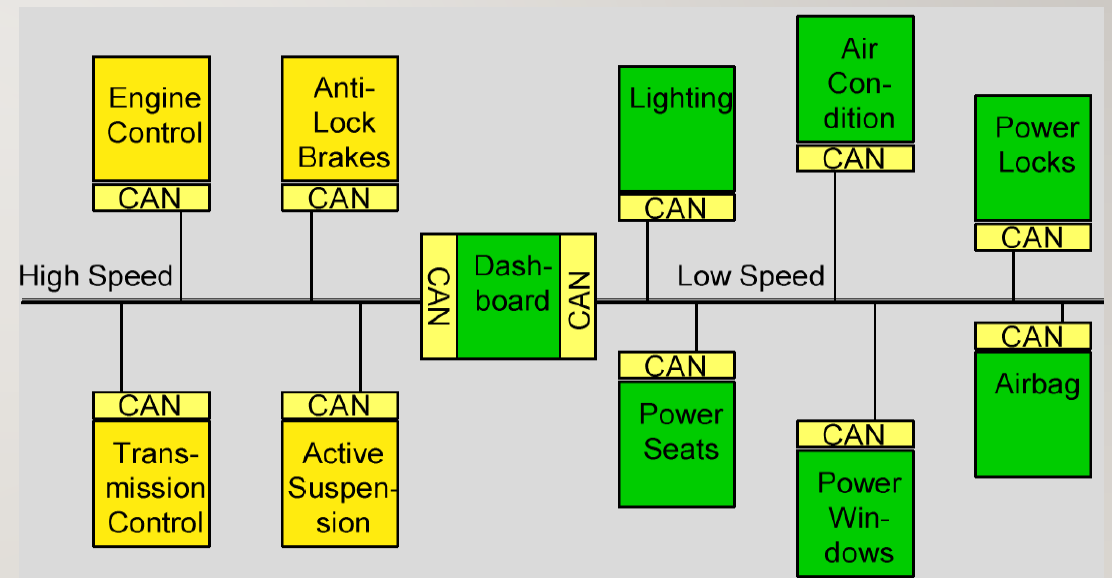
To improve the behavior of the vehicle even further, it was necessary for the different control systems (and their sensors) to exchange information. This was usually done by discrete interconnection of the different systems (i.e. point to point wiring). The requirement for information exchange has then grown to such an extent that a cable network with a length of up to several miles and many connectors was required. This produced growing problems concerning material cost, production time and reliability.

BEFORE CAN



WITH CAN

The solution to this problem was the connection of the control systems via a serial bus system. This bus had to fulfill some special requirements due to its usage in a vehicle. With the use of CAN, point-to-point wiring is replaced by one serial bus connecting all control systems. This is accomplished by adding some CAN-specific hardware to each control unit that provides the "rules" or the protocol for transmitting and receiving information via the bus.



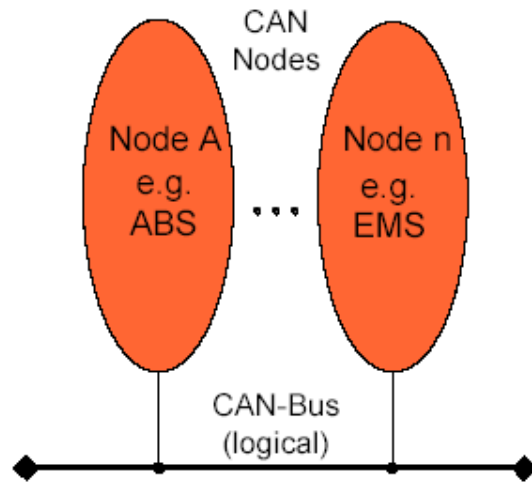
THE CAN BUS

Basic Concepts

- Multimaster Concept

- Number of nodes not limited by protocol

- No node addressing, Message identifier specifies contents & priority

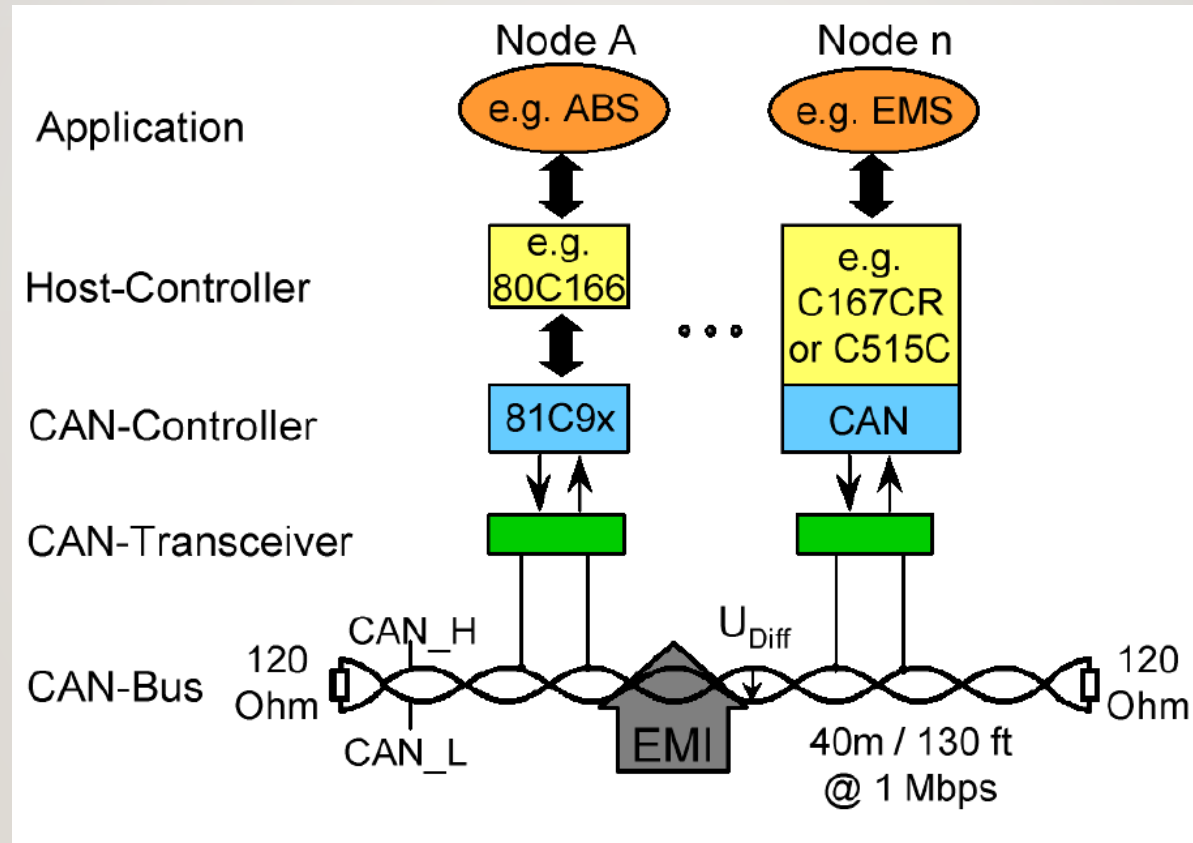


- Easy connection/disconnection of nodes

- Broadcast/Multicast capability

- CAN is a broadcast type of bus.
 - This means that all nodes can "hear" all transmissions. There is no way to send a message to just a specific node; all nodes will invariably pick up all traffic. The CAN hardware, however, provides local filtering so that each node may react only on the "interesting" messages.

BASIC CONFIGURATION



CAN BUS OVERVIEW

- The physical layer uses differential transmission on a twisted pair wire. The bus uses Non-Return To Zero (NRZ) with bit-stuffing.
- The nodes are connected to the bus in a *wired-and* fashion: if just one node is driving the bus to a logical 0, then the whole bus is in that state regardless of the number of nodes transmitting a logical 1.
- Max. transfer rate of 1000 kilobits per second at a maximum bus length of 40 meters or 130 feet when using a twisted wire pair which is the most common bus medium used for CAN.
- Message length is short with a maximum of 8 data bytes per message and there is a low latency between transmission request and start of transmission. The messages are protected by a CRC type checksum

CAN BUS OVERVIEW

- The bus access is handled via the advanced serial communications protocol Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration. This means that collision of messages is avoided by bitwise arbitration without loss of time.
- There is no explicit address in the messages, instead, each message carries a numeric value which controls its priority on the bus, and may also serve as an identification of the *contents* of the message.
- An elaborate error handling scheme that results in retransmitted messages when they are not properly received.
- There are effective means for isolating faults and removing faulty nodes from the bus.

BASIC BIT ENCODING

NRZ = Non-Return-To Zero

- Fewer transitions (on average) = less EMI, but requires less oscillator drift

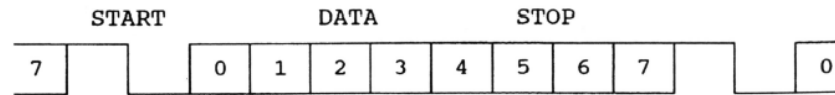
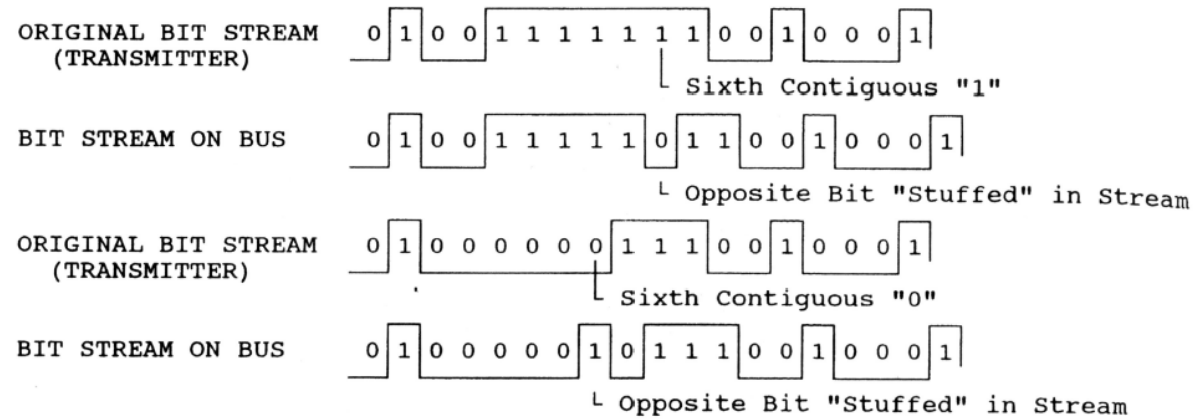


FIGURE 26.21 A 10-bit NRZ waveform (LSB first).

- Bit stuffing relaxes oscillator drift requirements



NRZ BIT STUFFING (ADDED SLIDE)

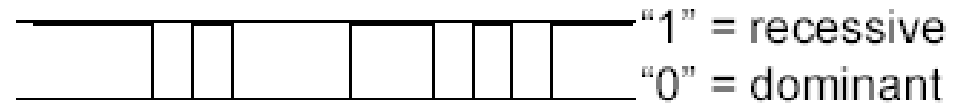


https://commons.wikimedia.org/wiki/File:Bitstuffing_en.svg

- 2 bits stuffed after 5 bits at same level
- Generates transitions for syncing

CAN BUS CHARACTERISTICS

Two logic states
possible on the bus:
"1" = recessive
"0" = dominant

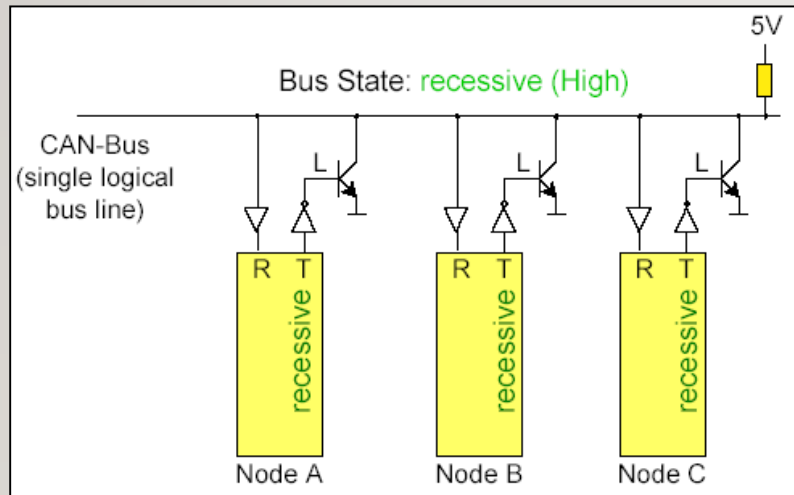


A	B	C	BUS
D	D	D	D
D	D	R	D
D	R	D	D
D	R	R	D
R	D	D	D
R	D	R	D
R	R	D	D
R	R	R	R

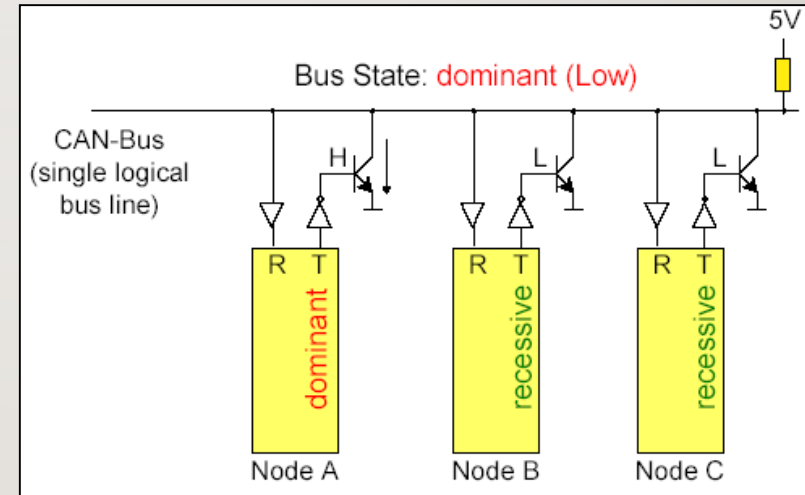
As soon as one node transmits
a dominant bit (zero):
Bus is in the dominant state.

Only if all nodes transmit
recessive bits (ones):
Bus is in the recessive state.

BUS CHARACTERISTICS – WIRED AND



Only if all nodes transmit recessive bits (ones), the Bus is in the recessive state.

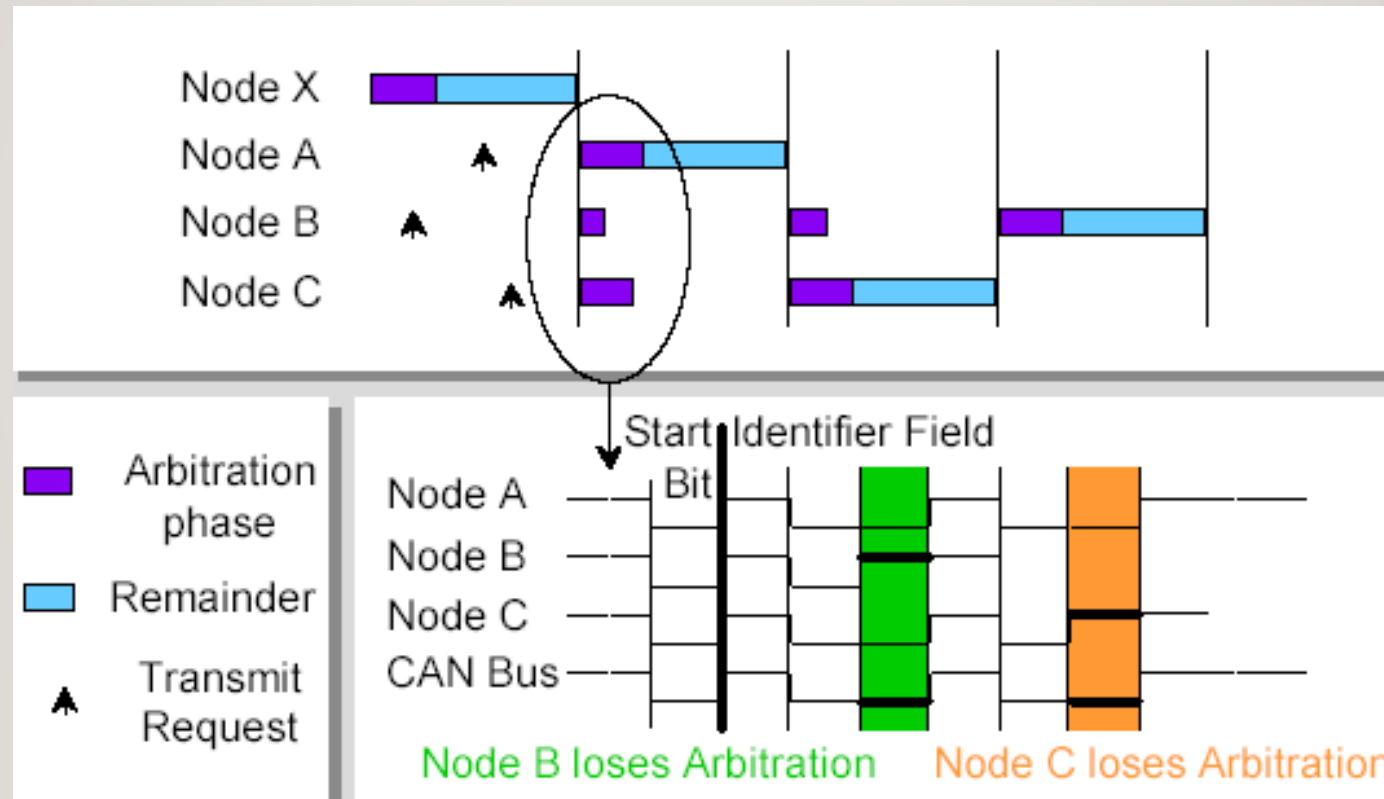


If any one node transmits a dominant bit (zero), the bus is in the dominant state.

T is Transmitter, R is receiver. Note nodes can therefore check the line while transmitting. This is important particularly during arbitration.

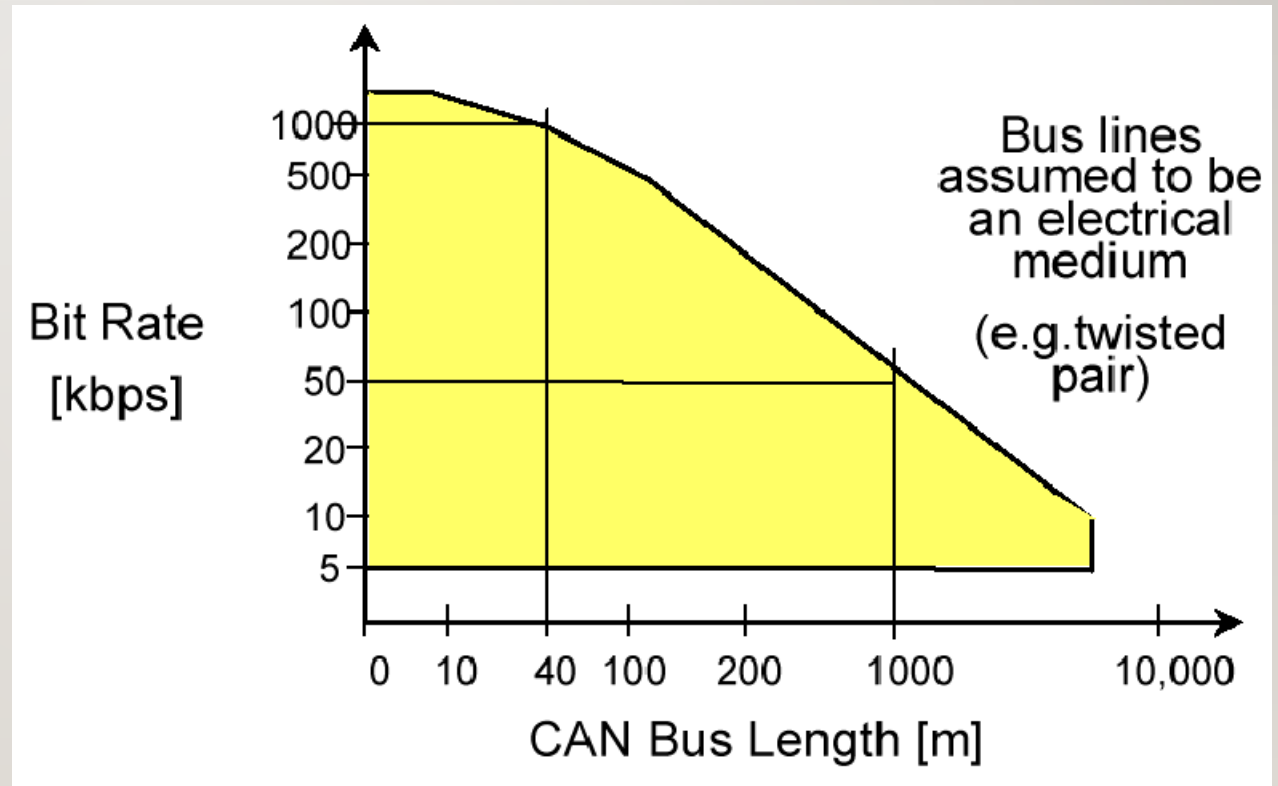
BUS ACCESS AND ARBITRATION – CSMA/CD NDA

CSMA/CD NDA – Carrier Sense Multiple Access/Collision avoidance by Non Destructive arbitration



BUS TRANSMISSION SPEED

- Arbitration limits bus speed.
- Maximum speed = $2 \times \tau_{pd}$
- τ_{pd} = propagation delay of medium



THE CAN PROTOCOL

- Specifies how small packets of data may be transported from point A to point B using a shared communications medium.
- It (quite naturally) contains nothing on topics such as
 - flow control
 - transportation of data larger than can fit in a 8-byte message
 - node addresses
 - establishment of communication, etc.

HIGHER LAYER PROTOCOLS

- Higher layer protocols are used in order to
 - standardize startup procedures including bit rate setting
 - distribute addresses among participating nodes or kinds of messages
 - determine the layout of the messages
 - provide routines for error handling at the system level
- Some high layer protocols
 - Device net
 - CANKingdom
 - CANopen

THE CAN STANDARD

- The CAN standard defines four message types
 - Data Frame – the predominantly used message type
 - Remote Frame
 - Error Frame
 - Overload Frame
- The messages uses a clever scheme of bit-wise arbitration to control access to the bus, and each message is tagged with a priority.
- The CAN standard also defines an elaborate scheme for error handling and confinement.
- CAN may implemented using different physical layers, and there are also a number of different connector types in use.

I. THE DATA FRAME

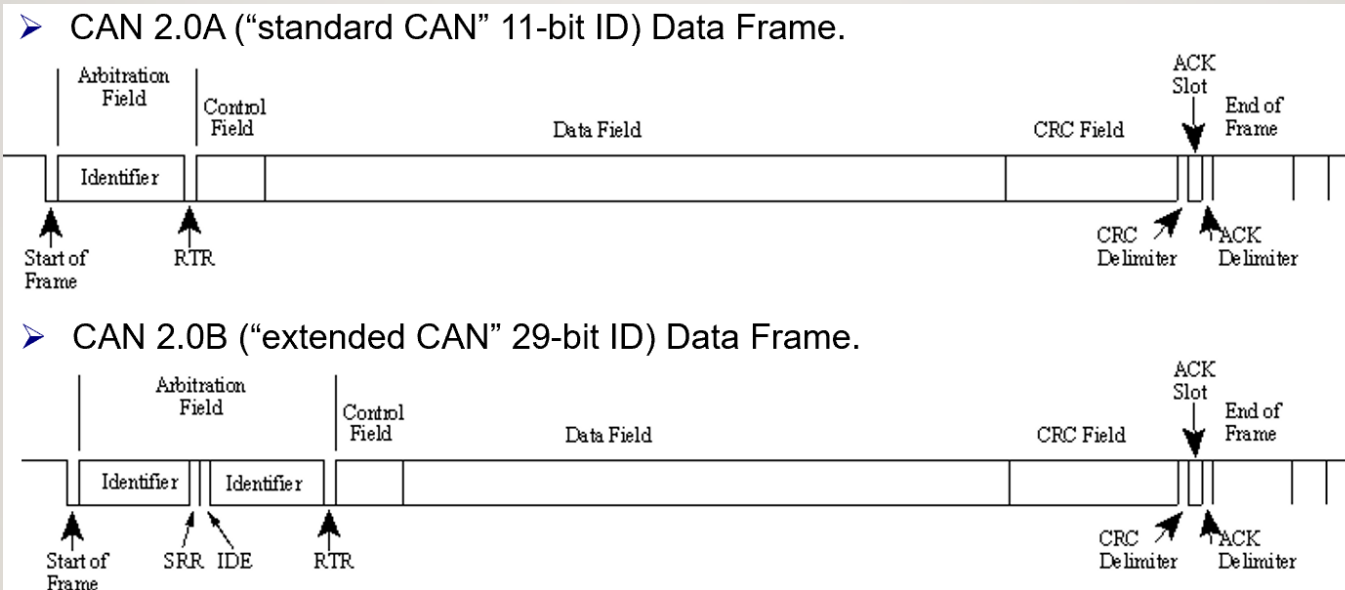
Summary: "Hello everyone, here's some data labeled X, hope you like it!"

- The Data Frame is the most common message type. It comprises the following major parts (a few details are omitted for the sake of brevity):
 - the Arbitration Field, which determines the priority of the message when two or more nodes are contending for the bus. The Arbitration Field contains:
 - For CAN 2.0A, an 11-bit Identifier and one bit, the RTR bit, which is dominant for data frames.
 - For CAN 2.0B, a 29-bit Identifier (which also contains two recessive bits: SRR and IDE) and the RTR bit.
 - the Data Field, which contains zero to eight bytes of data.
 - the CRC Field, which contains a 15-bit checksum calculated on most parts of the message. This checksum is used for error detection.
 - an Acknowledgement Slot; **any** CAN controller that has been able to correctly receive the message sends an Acknowledgement bit at the end of each message. The transmitter checks for the presence of the Acknowledge bit and retransmits the message if no acknowledge was detected.

CAN DATA FRAMES

Note 1: It is worth noting that the presence of an Acknowledgement Bit on the bus does not mean that any of the *intended* addressees has received the message. The only thing we know is that *one or more* nodes on the bus has received it correctly

Note 2: The Identifier in the Arbitration Field is not, despite of its name, necessarily identifying the contents of the message.

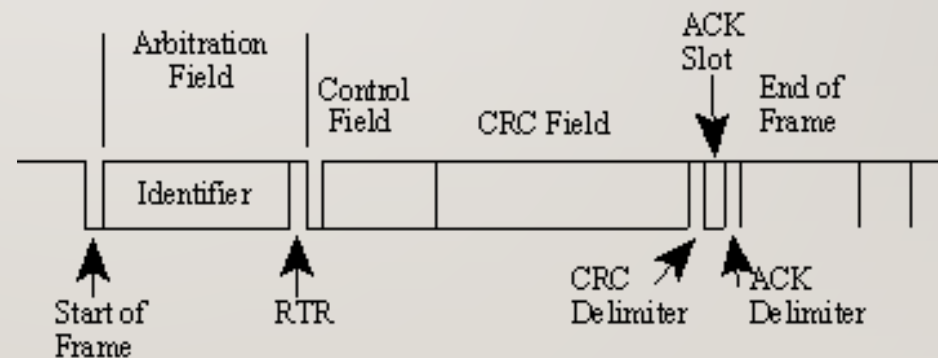


THE REMOTE FRAME

- **Summary:** *"Hello everyone, can somebody please produce the data labeled X?"*
- The Remote Frame is just like the Data Frame, with two important differences:
 - It is explicitly marked as a Remote Frame (the RTR bit in the Arbitration Field is recessive), and
 - there is no Data Field.
- The intended purpose of the Remote Frame is to solicit the transmission of the corresponding Data Frame. If, say, node A transmits a Remote Frame with the Arbitration Field set to 234, then node B, if properly initialized, might respond with a Data Frame with the Arbitration Field also set to 234.
- Remote Frames can be used to implement a type of request-response type of bus traffic management. In practice, however, the Remote Frame is little used. It is also worth noting that the CAN standard does not *prescribe* the behaviour outlined here. Most CAN controllers can be programmed either to automatically respond to a Remote Frame, or to notify the local CPU instead.

REMOTE FRAME (CONTD.)

- There's one catch with the Remote Frame: the Data Length Code *must be set to the length of the expected response message*. Otherwise the arbitration will not work.
- Sometimes it is claimed that the node responding to the Remote Frame is starting its transmission as soon as the identifier is recognized, thereby "filling up" the empty Remote Frame. ***This is not the case.***
- A Remote Frame (2.0A type):

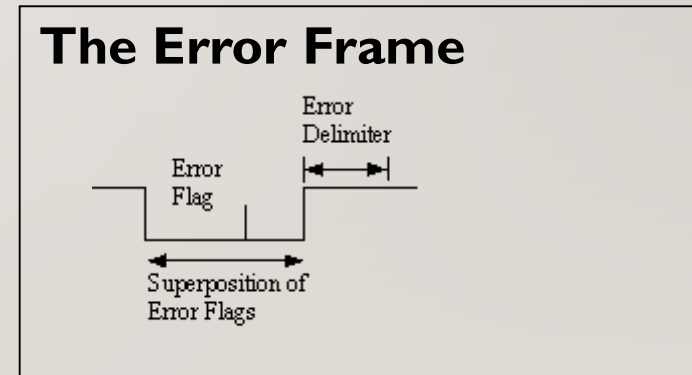


3. THE ERROR FRAME

Summary: (everyone, aloud) "OH DEAR, LET'S TRY AGAIN"

Simply put, the Error Frame is a special message that violates the framing rules of a CAN message. It is transmitted when a node detects a fault and will cause all other nodes to detect a fault - so they will send Error Frames, too. The transmitter will then automatically try to retransmit the message. There is an elaborate scheme of error counters that ensures that a node can't destroy the bus traffic by repeatedly transmitting Error Frames.

The Error Frame consists of an Error Flag, which is 6 bits of the same value (thus violating the bit-stuffing rule) and an Error Delimiter, which is 8 recessive bits. The Error Delimiter provides some space in which the other nodes on the bus can send their Error Flags when they detect the first Error Flag.



4 THE OVERLOAD FRAME

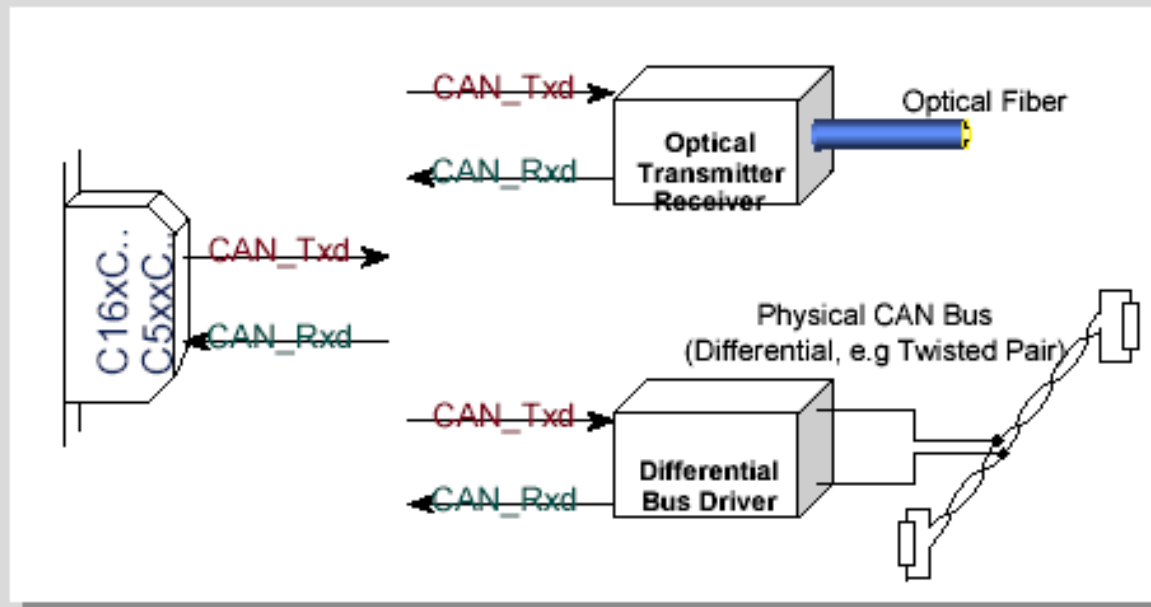
Summary: "I'm a very busy little 82526 device, could you please wait for a moment?"

- The Overload Frame is mentioned here just for completeness. It is very similar to the Error Frame with regard to the format and it is transmitted by a node that becomes too busy. The Overload Frame is not used very often, as today's CAN controllers are clever enough not to use it. In fact, the only controller that will generate Overload Frames is the now obsolete 82526

ISO PHYSICAL LAYER

□ Usual ISO Physical Layer :-

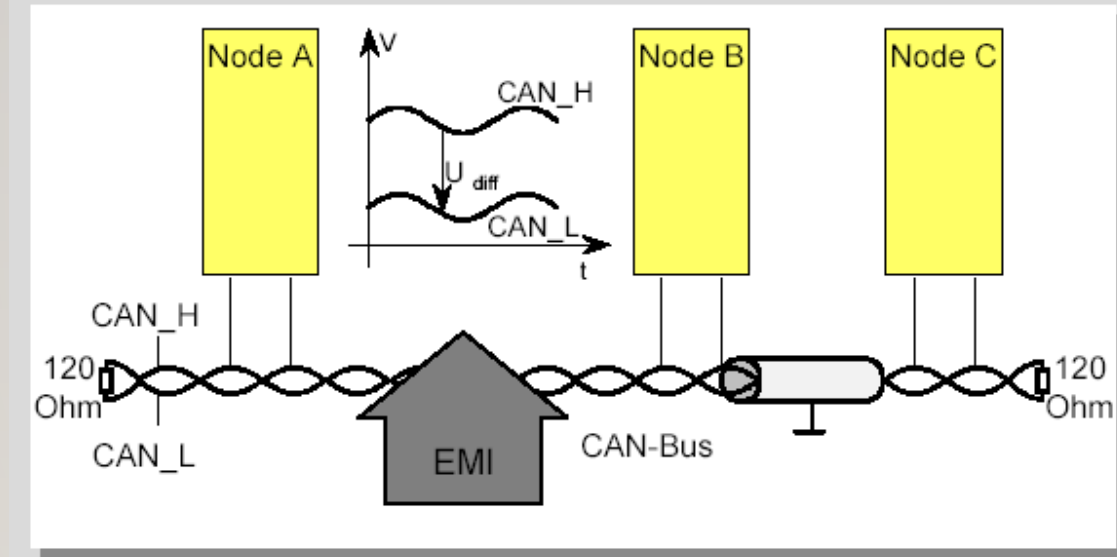
- Bus wires twisted pair, 120R Termination at each end
- 2 wires driven with differential signal (CAN_H, CAN_L)



One of the most common and cheapest implementations is to use a twisted wire pair. The bus lines are then called "CAN_H" and "CAN_L". The two bus lines CAN_H and CAN_L are driven by the nodes with a differential signal. The twisted wire pair is terminated by terminating resistors at each end of bus line, typically 120 ohms.

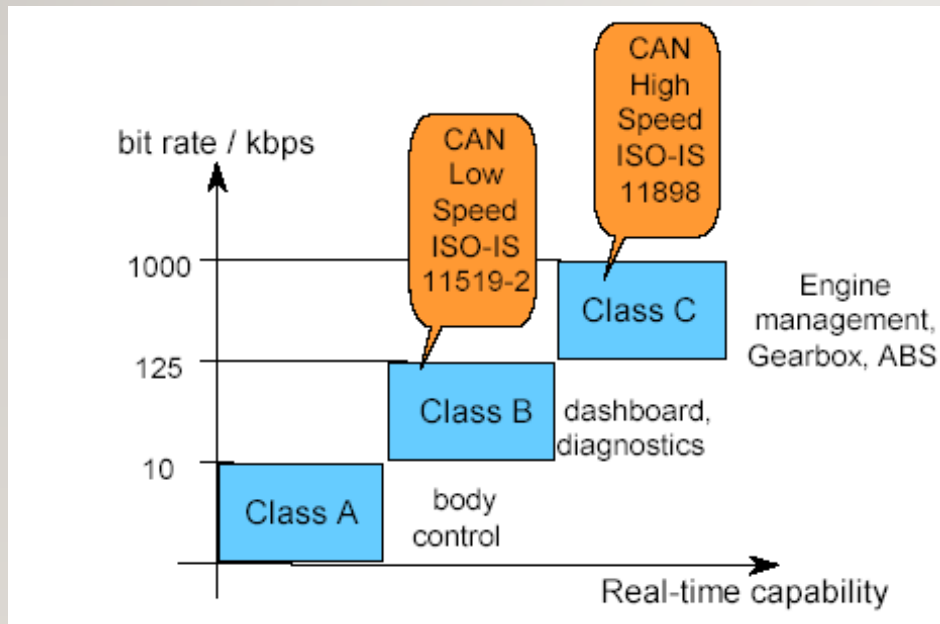
CAN AND EMI

□ CAN is insensitive to electromagnetic interference



Due to the differential nature of transmission CAN is insensitive to electromagnetic interference, because both bus lines are affected in the same way which leaves the differential signal unaffected. To reduce the sensitivity against electromagnetic interference even more, the bus lines can additionally be shielded. This also reduces the electromagnetic emission of the bus itself, especially at high baud rates.

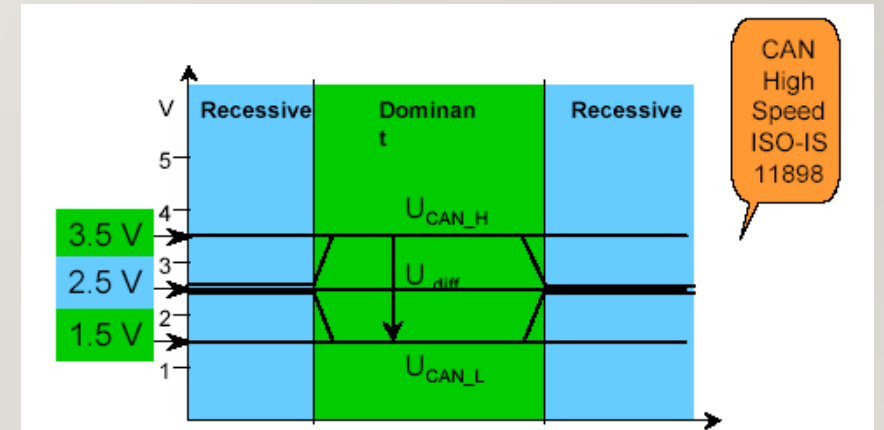
STANDARDISATION



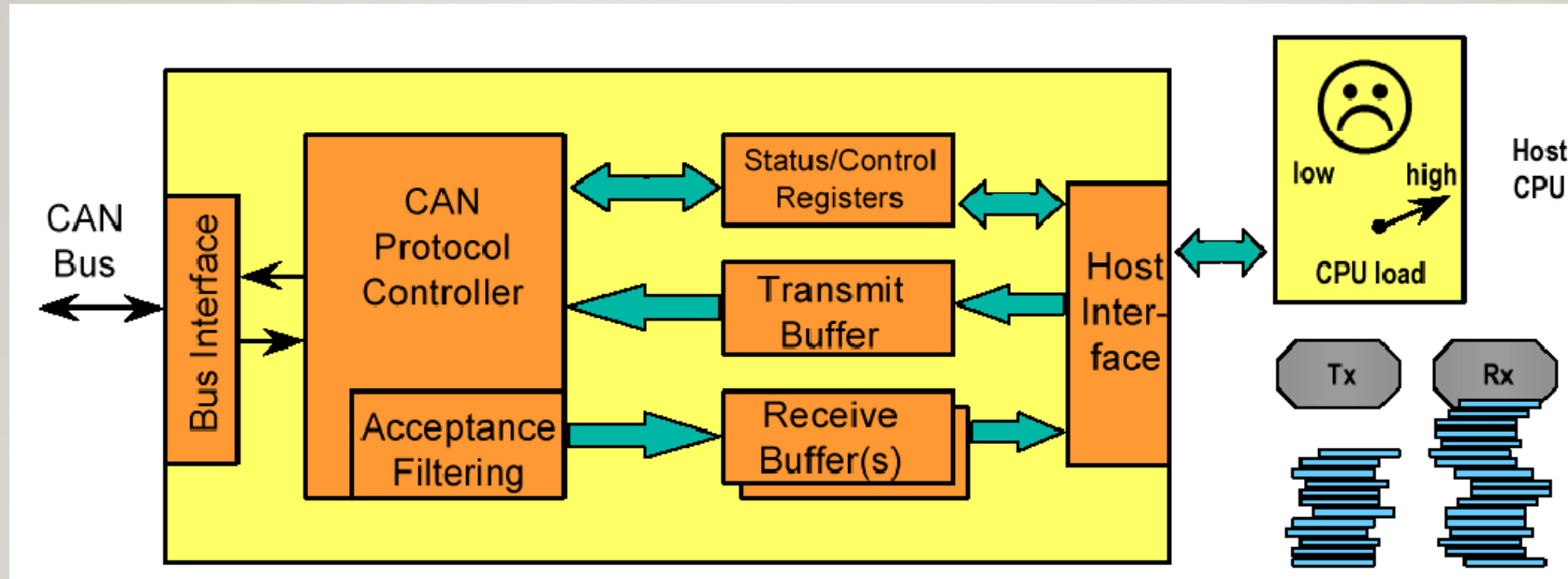
- Vehicle bus system applications can be separated in three different categories according to their real-time capabilities.
 - Class A for a low speed bus with bit rates up to 10 kbps, e.g. for body control applications,
 - Class B for a low speed bus with bit rates from 10 kbps to 125 kbps, e.g. for dashboard and diagnostics,
 - Class C for a high speed bus with bit rates from 125 kbps to 1 Mbps for real time applications like engine management, Gearbox, ABS etc.

BUS LEVELS ACCORDING TO ISO-11898

- These are the bus levels according to ISO-11898. A recessive bit is represented by both CAN bus lines driven to a level of about 2.5 V so that the differential voltage between CAN_H and CAN_L is around 0V.
- A dominant bit is represented by CAN_H going to about 3.5 V and CAN_L going to about 1.5 V. This results in a differential voltage for a dominant bit of about 2V.



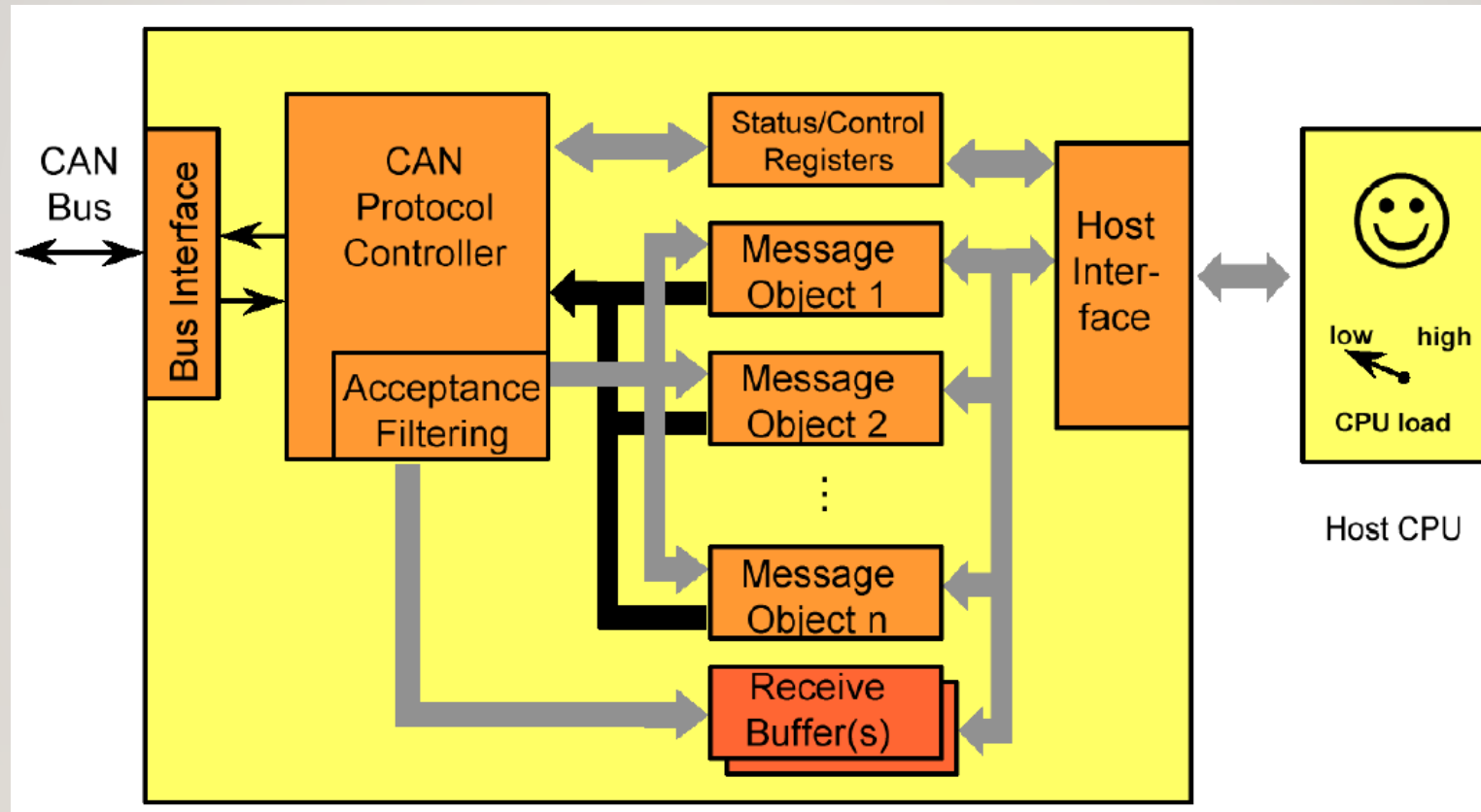
A BASIC CAN CONTROLLER



- Cheap CAN controller – CPU could get overrun with messages even if it didn't need them.

FULL CAN CONTROLLER

- **Hardware message filters sort & filter messages without interrupting CPU**



NMEA 2000



OVERVIEW

- Updated NMEA standard
- Addresses most limitations of older standard
- Essentially based upon CAN
 - But they didn't originally adopt a specific standard for the termination or wire types
 - Otherwise compatible physical layers have incompatible connectors
 - Raymarine Seatalk-NG
 - Simrad Simnet
 - CAN
 - Updated NMEA 2000 standard uses DeviceNet MicroC connectors
 - But, too late! All the proprietary standards are well rooted in the wild

PARAMETER GROUP NUMBERS (PGNS)

NMEA 0183 Sentence RMB – Recommended Min. Nav. Info.	
Field #	Description
1	Data Status
2	Cross Track Error
3	Direction to Steer
4	Origin Waypoint ID
5	Destination Waypoint ID
6	Destination WP Latitude
7	Destination WP Lat. M/S
8	Destination WP Longitude
9	Destination WP Long. E/W
10	Range to Destination
11	Bearing to Destination
12	Dest. Closing Velocity
13	Arrival Status
14	Mode Indicator

NMEA 0183 Sentence BOD – Bearing – Origin to Destination	
Field #	Description
1	Bearing True
2	True
3	Bearing Magnetic
4	Magnetic
5	Destination Waypoint ID
6	Origin Waypoint ID

NMEA 2000 PGN 129284 – Navigation Data	
Field #	Description
1	SID
2	Distance to Dest. WP
3	Course/Bearing Ref.
4	Perpendicular Crossed
5	Arrival Circle Entered
6	Calculation Type
7	ETA Time
8	ETA Date
9	Bearing, Origin to Dest.
10	Bearing, Position to Dest.
11	Origin WP Number
12	Destination WP Number
13	Destination WP Latitude
14	Destination WP Longitude
15	WP Closing Velocity

PGNs are rough approximation of NMEA sentences

https://www.maretron.com/support/manuals/USB100UM_1.6.html

ETHERNET, WI-FI, AND BEYOND



MARINE NETWORKS TODAY

- Mix of standards
 - Old stuff that still works, e.g. SeaTalk-I
 - NMEA-0183 won't die
 - It is still low cost, highly reliable, sufficient
 - Perfect for DSC radios, AIS, etc
 - USB
 - Limited use, mostly via bridging from RS-485
 - Still point-to-point, not common bus
 - Limited value of RS-485 given data rates
 - NMEA-2000 variants
 - Usually one of these is used as a common bus
 - Everything else gets bridged to this
 - Ethernet and Wi-Fi
 - Ethernet can be used as common bus in place of CAN standard
 - Virtual serial ports convert network data to familiar software interface

MIXING DATA TYPES

- NMEA-0183 is good enough for most individual uses and often times many uses combined
 - And it is 4800,n,8,l
- NMEA-0183 isn't good enough for some things
 - AIS, for example, requires 38400,n,8,l
- CAN, Ethernet and Wi-Fi have a ton more bandwidth
 - They don't really have the collision/buffering problem that some NMEA-0183 multiplexers have
 - They can basically handle a ton of NMEA data
- Radar data, sonar data, etc can be high resolution and are timely
 - There can be problems mixing these with other types of data.

MIXING TYPES OF DATA

- Radar has been using Ethernet (or specialized high-bandwidth connections) since inception
 - When using Ethernet, it was originally the only thing
 - Now it often has to share with other devices
 - And, modern radars are even higher resolution and faster scanning
 - Collision causes missing Radar spokes

THE HOLDING POWER OF LEGACY

- Old interfaces live long lives
 - Never underestimate the staying power of “legacy”
- Standard interface is an old-school “COM Port”
- Common bus is necessary for modern data distribution
 - CAN bus, ethernet, etc
- Options
 - Use “adapter box” to bridge common bus to USB (which is natively mapped as a COM port)
 - Map network traffic to “COM Port” as “Virtual COM Port”

NETWORK VIRTUAL COM PORTS

- Abstract network communication as a virtual serial port
 - Enables continued use of legacy interface
 - COM settings are basically imaginary
 - Software just reads and writes from COM port as usual
- UDP
 - Unicast or broadcast
 - Unreliable
 - Map traffic from <IP,Port> to a COM Port.
- TCP
 - Unicast only, broadcast isn't an option
 - Reliable
 - Requires a connection
 - End-User device can act as client or server
 - Often times (but not always) TCP server device limits to one client
 - Otherwise, how long to buffer data? How much buffer needed

EXAMPLE

“SEAS THE BAY”, 42’ HARDIN EUROPA TRAWLER



- Seataalk Legacy devices
 - MFD, GPS, Wireless AP Controller
- NMEA-0183
 - 2x VHF Radios (GPS, AIS, DSC)
 - AIS Transceiver (In: GPS, Heading, etc; Out: AIS)
 - Depth, temperature, and speed
- NMEA-2000 (CAN + Simnet + Seataalk-NG)
 - Common Seataalk-NG bus
 - AP, Sensor core, Mini-displays, AP control head
 - Some radar data (ARPA contacts + more)
- Ethernet
 - Radar data and most control
 - Upper and lower primary displays
- Wi-Fi
 - Laptop
 - Cellphone