Understanding and Using Controller Area Network (CAN)

Introduction: The CAN bus

- The CAN bus was developed by Robert Bosch GmbH
- Multi-master with Maximum signalling rate of 1 Mbps.
- Message broadcast system.
  - 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.

Introduction: The CAN bus

- The bus uses Non-Return To Zero (NRZ) with bit-stuffing
- The modules 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.
- CAN is ideally suited in applications requiring a large number of short messages with high reliability in rugged operating environments.
  - high immunity to electrical interference and the ability to self-diagnose and repair data errors.

The CAN Standard

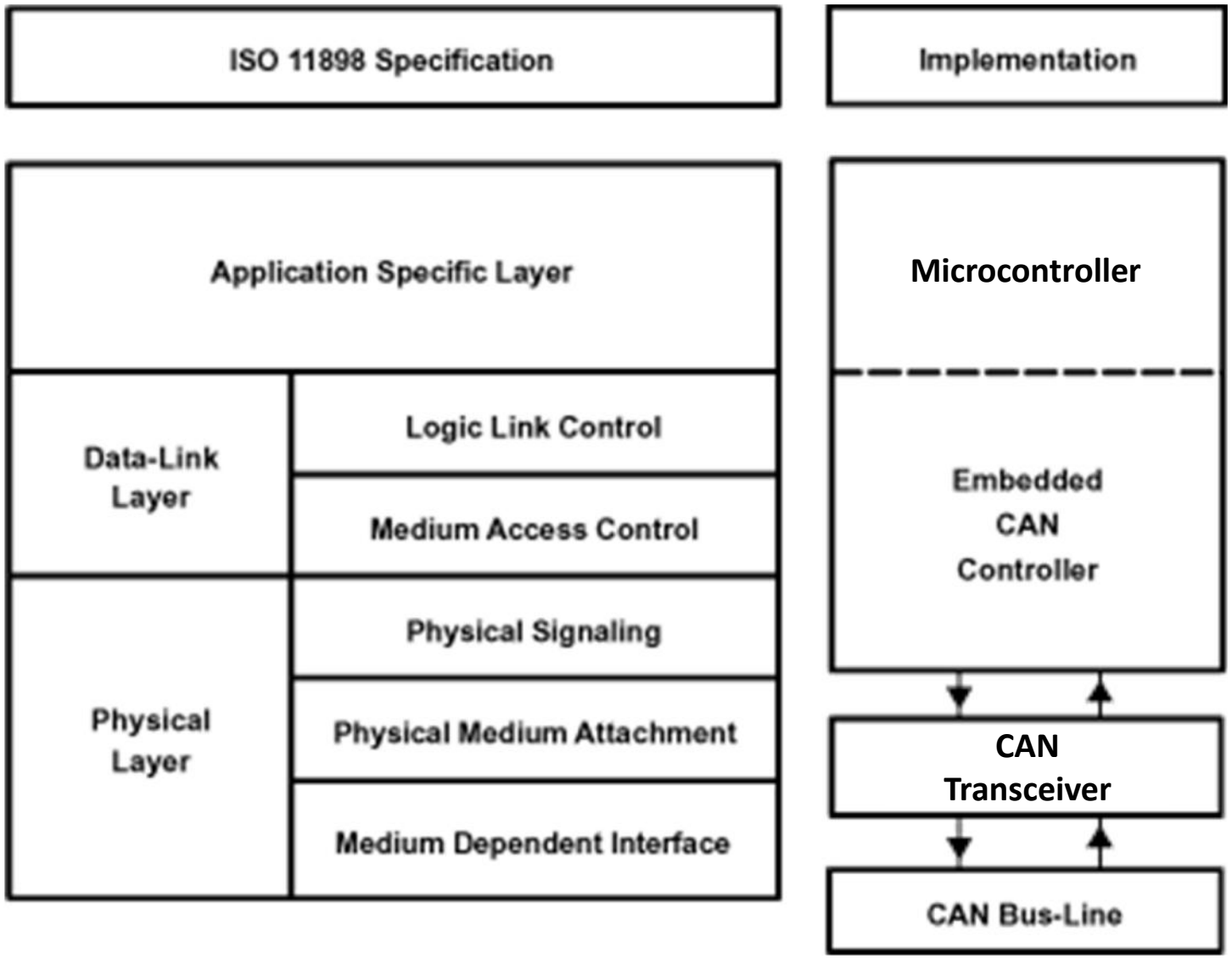
- CAN is an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus.
- Application Areas: Automotive, Automation, and Medical.
- The CAN communications protocol, ISO-11898, describes how information is passed between devices on a network and conforms to the Open Systems Interconnection (OSI) model that is defined in terms of layers.

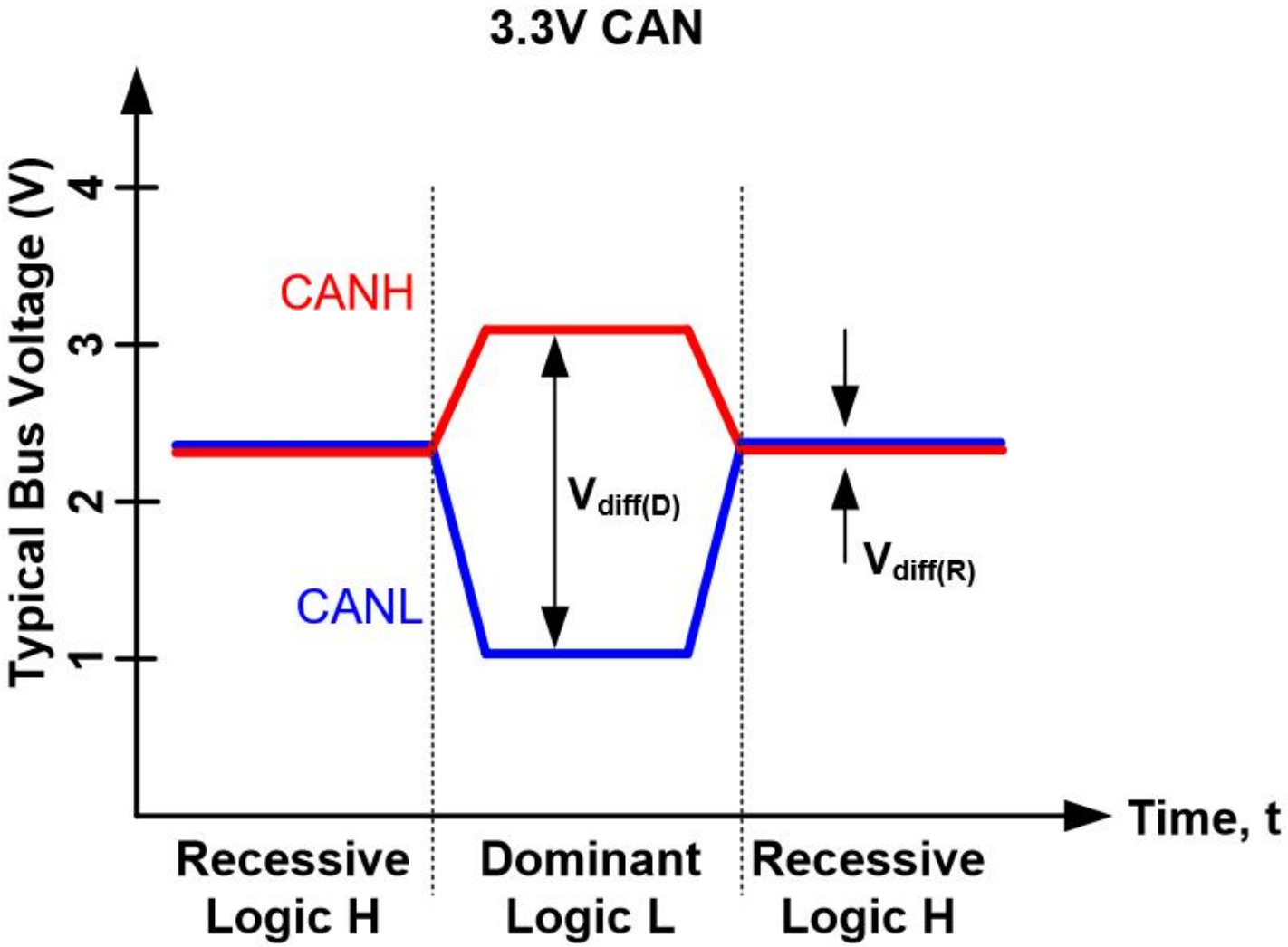
The CAN Standard

- The **ISO 11898** architecture defines the lowest two layers of the seven layer OSI/ISO model as the data-link layer and physical layer.

Version Number	Year	Additions
1.0	1995	Original CAN standard
1.1	1987	Respecified bit timing requirements
1.2	1990	Increased oscillator tolerance
2.0A	1991	Same as version 1.2
2.0B	1991	Introduced optional extended frame

The Layered ISO 11898 Standard Architecture

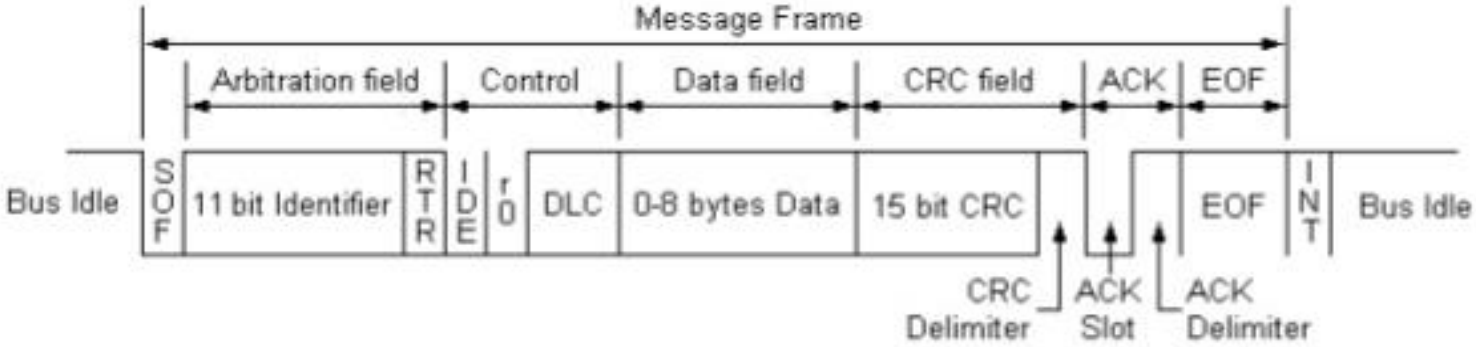



3.3V CAN
Typical Bus Voltage (V)
4
3
2
1
CANH
CANL
$V_{diff(D)}$
$V_{diff(R)}$
Time, t
Recessive
Logic H
Dominant
Logic L
Recessive
Logic H

CAN Messages

- CAN uses short messages
  - the maximum utility load is 94 bits.
- There is no explicit address in the messages
  - instead, the messages can be said to be contents-addressed, that is, their contents implicitly determines their address.
- Message Types / Frames
  1. The Data Frame (*used by CAN to send data*)
  2. The Remote Frame (*used by CAN to request data*)
  3. The Error Frame (*used by CAN to indicate error*)
  4. The Overload Frame (*used by CAN to insert a delay*)

CAN 2.0A Format



SOF: Start of frame (start bit)
ID: Message identifier (indicates msg priority)
RTR: Remote transmission request
IDE: Identifier extension bit (2.0A or 2.0B)
r0: Reserved bit. Sent as dominant.
DLC: Data length code. Valid range 0 – 8.
CRC D: CRC delimiter. Marks end of CRC field.
ACK S: Used for receiver to ACK msg. Sent as recessive.
ACK D: Marks end of ACK field.
EOF: End of frame. (stop bit). Sent as 7 recessive bits.
INT: Intermission. Sent as 3 recessive bits.

Standard vs. Extended CAN

- CAN 2.0 A / Basic or Standard CAN
  - Specifies standard message format only (extended messages are treated as an error).

- CAN 2.0 B / Full CAN
  - CAN 2.0 B passive
    - Handles standard messages and ignores extended messages.
  - CAN 2.0 B active
    - Handles both standard and extended message formats.

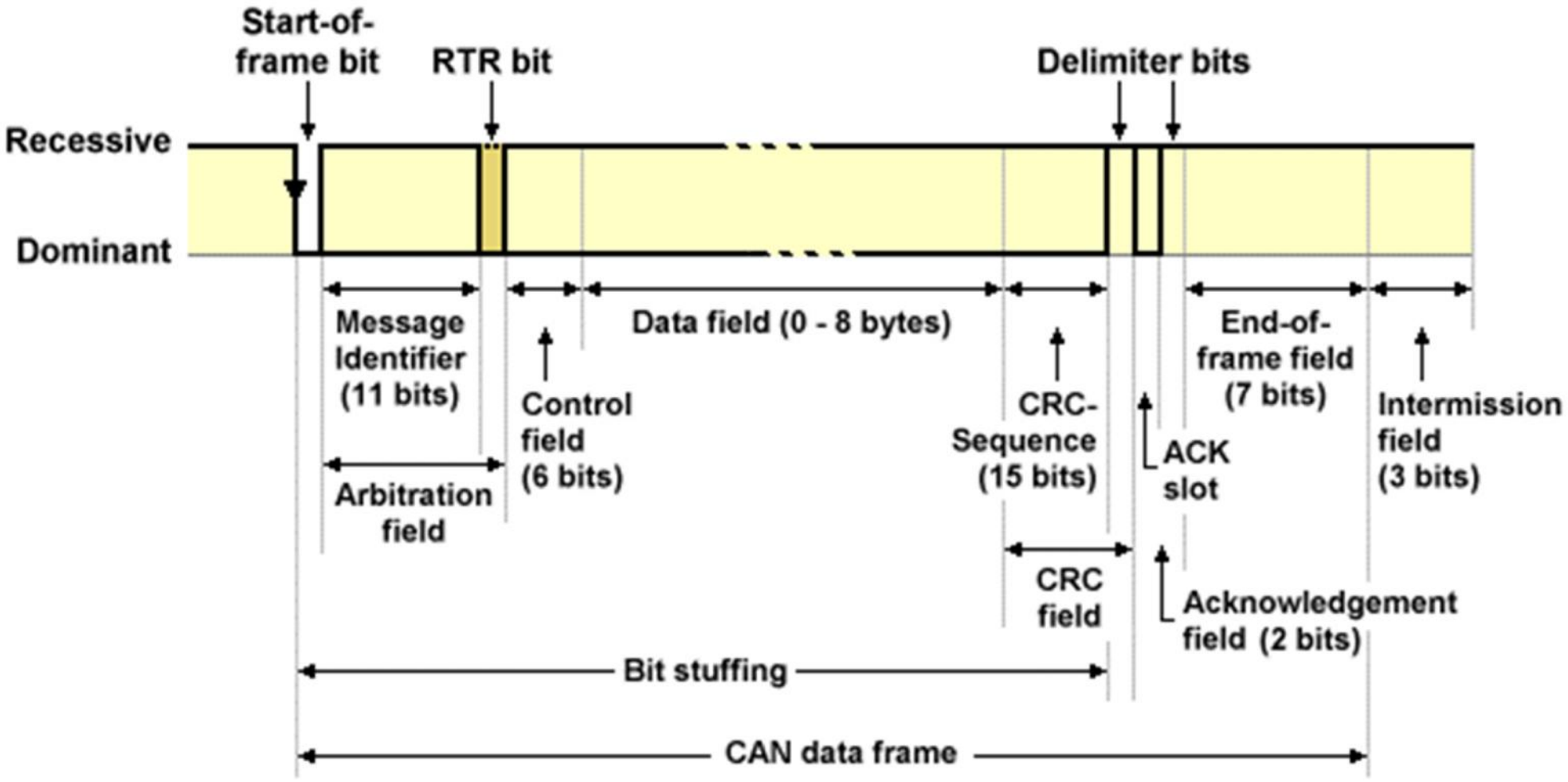
Basic CAN vs. Full CAN

- Once upon a time there was the Intel 82526 CAN controller which provided a DPRAM-style interface to the programmer – **Full CAN**

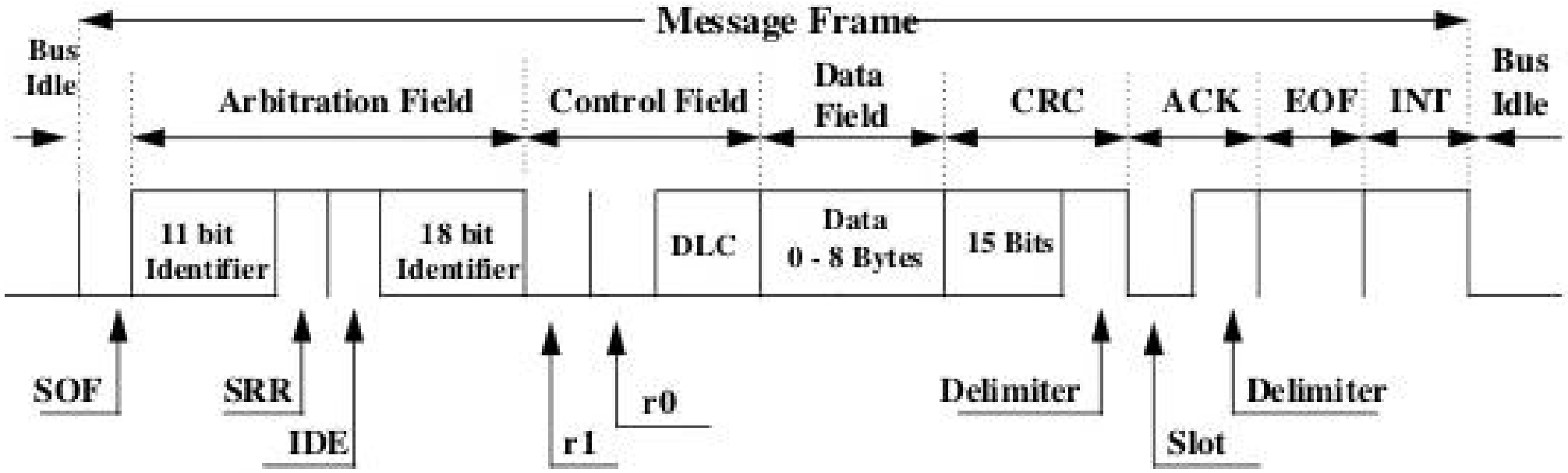
- Then came along Philips with the 82C200 which used a FIFO oriented programming model and limited filtering abilities - **Basic CAN**

- **These terms can cause confusion and should be avoided**
- Of course, a “Full CAN” controller can communicate with a “Basic CAN” controller and vice versa. There are no compatibility problems.

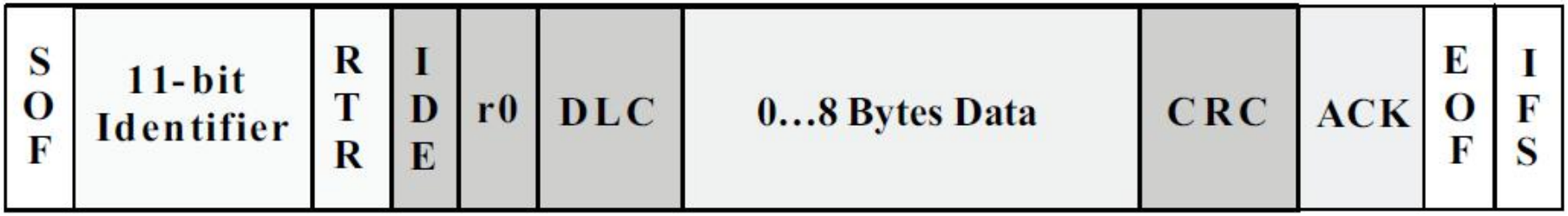
CAN Frame (2.0 A)



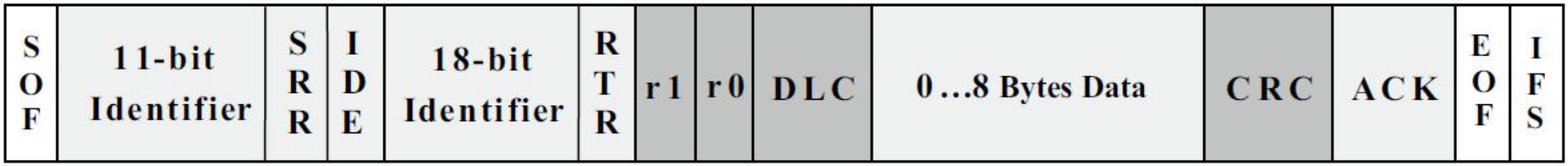
CAN Frame (2.0 B)

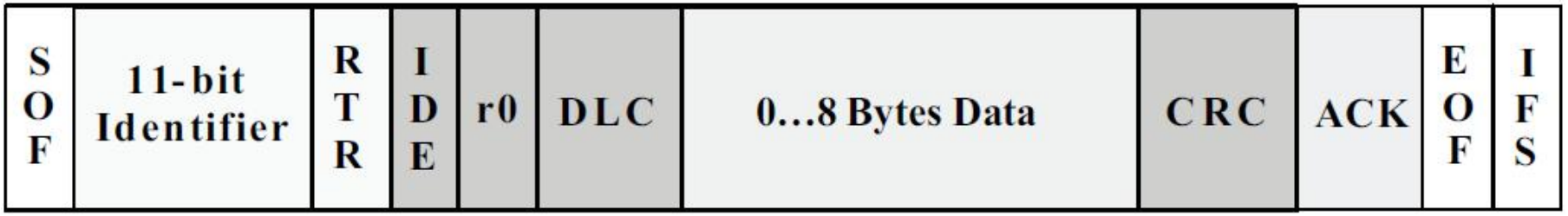


Standard CAN: 11-Bit Identifier



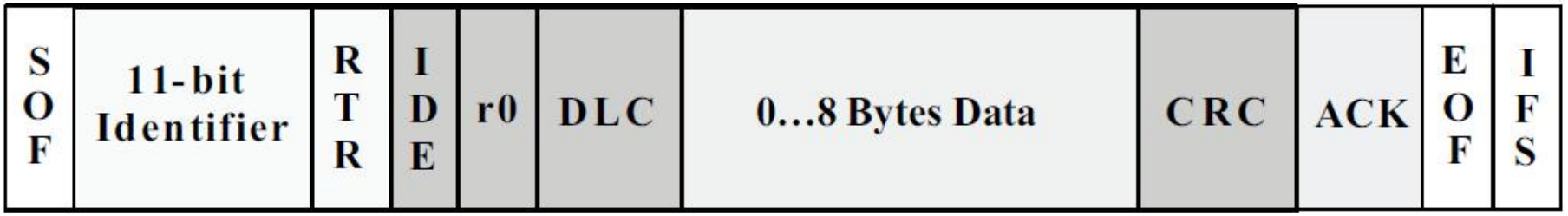
Extended CAN: 29-Bit Identifier





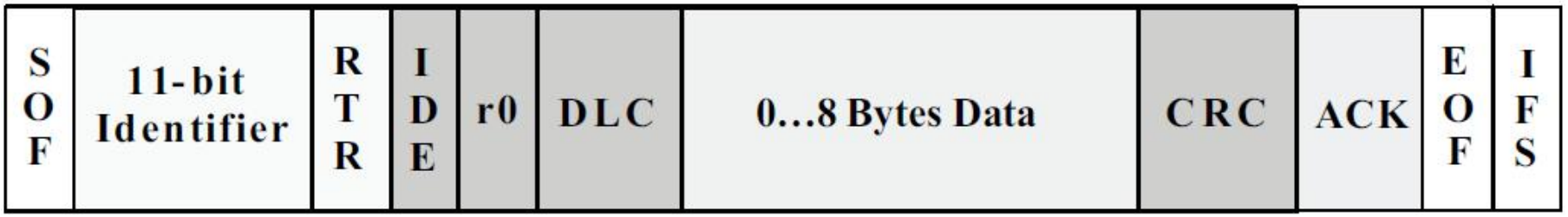
- SOF
  - The single dominant start of frame (SOF) bit marks the start of a message, and is used to synchronize the nodes on a bus after being idle.

- Arbitration Field
  - Identifier
    - The Standard CAN 11-bit identifier establishes the priority of the message.
    - The lower the binary value, the higher its priority.
  - RTR
    - The single remote transmission request (RTR) bit is dominant when information is sent to another node.

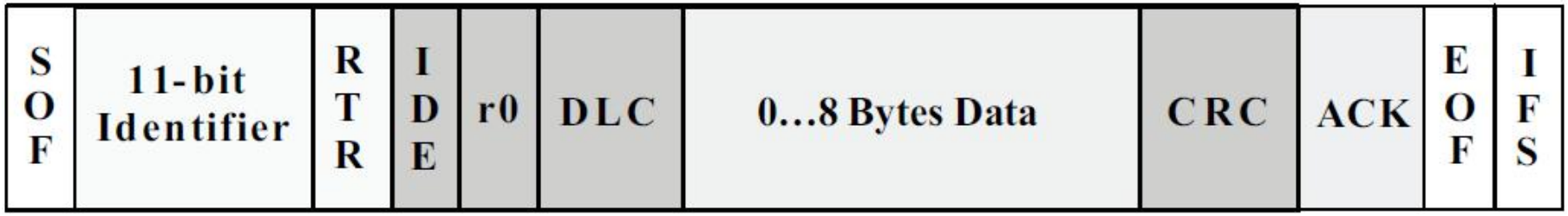


- Control Field
  - IDE
    - A dominant single identifier extension (IDE) bit means that a standard CAN identifier with no extension is being transmitted
  - r0
    - Reserved bit (for possible use by future standard amendment).
  - DLC
    - The 4-bit data length code (DLC) contains the number of bytes of data being transmitted.

- Data Field
  - Up to 64 bits of application data may be transmitted.



- CRC
  - The 16-bit (15 bits plus delimiter) cyclic redundancy check (CRC) contains the checksum (number of bits transmitted) of the preceding application data for error detection.
- ACK
  - Every node receiving an accurate message overwrites this recessive bit in the original message with a dominate bit, indicating an error-free message has been sent.
  - Should a receiving node detect an error and leave this bit recessive, it discards the message and the sending node repeats the message after re-arbitration. In this way, each node acknowledges (ACK) the integrity of its data.
  - ACK is 2 bits, one is the acknowledgment bit and the second is a delimiter.



- EOF
  - This end-of-frame (EOF)
  - 7-bit field marks the end of a CAN frame (message) and disables bit-stuffing, indicating a stuffing error when dominant.
  - When 5 bits of the same logic level occur in succession during normal operation, a bit of the opposite logic level is *stuffed* into the data.

The Data Frame

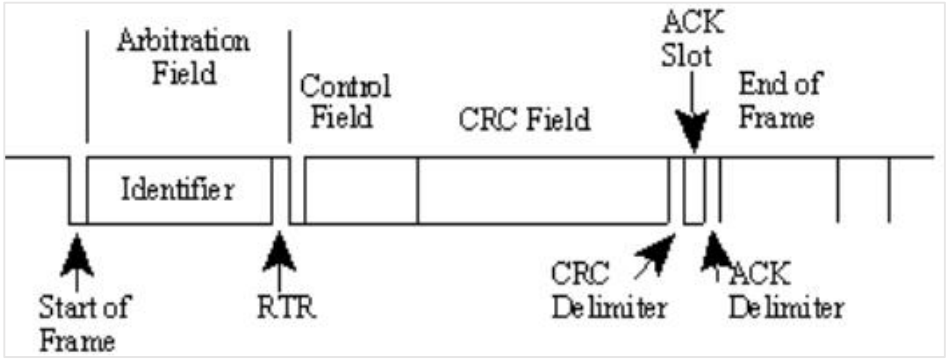
Summary: “Hello everyone, here’s some data labelled X, hope you like it!”

- The most common message type.
- 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 Remote Frame

Summary: “Hello everyone, can somebody please produce the data labelled 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.



- 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.

The Error Frame

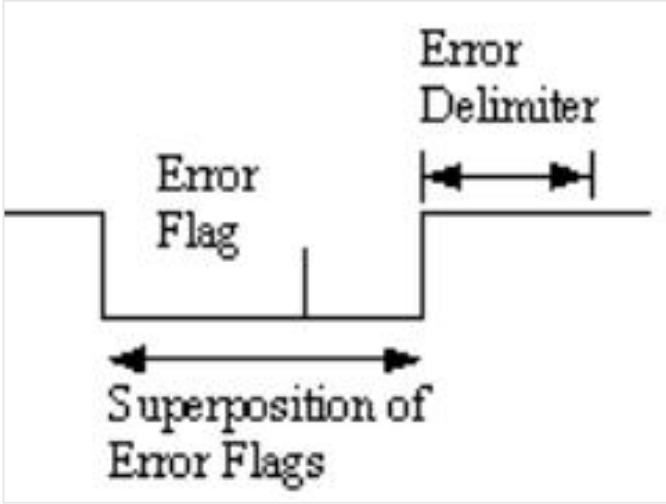
Summary: (everyone, aloud) “OH DEAR, LET’S TRY AGAIN”

- 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…

Summary: (everyone, aloud) “OH DEAR, LET’S TRY AGAIN”

- 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.



The Overload Frame

Summary: “I’m a very busy little 82526, could you please wait for a moment?”

- 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.

CAN Error Handling

How CAN Handles Errors

- Every CAN controller along a bus will try to detect errors within a message. If an error is found, the discovering node will transmit an Error Flag, thus destroying the bus traffic.

- Each node maintains two error counters:
  - the Transmit Error Counter and
  - the Receive Error Counter.

- Using the error counters, a CAN node can not only detect faults but also perform error confinement.

Error Detection Mechanisms

- The CAN protocol defines no less than five different ways of detecting errors. Two of these works at the bit level, and the other three at the message level.
  1. **Bit Monitoring. (Tx, Bit)**
  2. **Bit Stuffing. (Rx, Bit)**
  3. Frame Check. (Rx, Message Level)
  4. Acknowledgement Check. (Tx, Msg Level)
  5. Cyclic Redundancy Check. (Rx, Msg Level)

Bit Monitoring

- Each transmitter on the CAN bus monitors (i.e. reads back) the transmitted signal level. If the bit level actually read differs from the one transmitted, a Bit Error is signalled.
- No bit error is raised during the arbitration process.

Bit Stuffing

- When five consecutive bits of the same level have been transmitted by a node, it will add a sixth bit of the opposite level to the outgoing bit stream. The receivers will remove this extra bit.
- This is done to avoid excessive DC components on the bus, but it also gives the receivers an extra opportunity to detect errors: if more than five consecutive bits of the same level occurs on the bus, a Stuff Error is signalled.

Frame check

- Some parts of the CAN message have a fixed format, i.e. the standard defines exactly what levels must occur and when.
  - Those parts are the CRC Delimiter, ACK Delimiter, End of Frame, and also the Intermission, but there are some extra special error checking rules for that.
- If a CAN controller detects an invalid value in one of these fixed fields, a Form Error is signalled.

Acknowledgement Check

- All nodes on the bus that correctly receives a message (regardless of their being “interested” of its contents or not) are expected to send a dominant level in the so-called Acknowledgement Slot in the message.
  - The transmitter will transmit a recessive level here.
- If the transmitter can’t detect a dominant level in the ACK slot, an Acknowledgement Error is signalled.

Cyclic Redundancy Check

- Each message features a 15-bit Cyclic Redundancy Checksum (CRC), and any node that detects a different CRC in the message than what it has calculated itself will signal an *CRC Error*.

Error Confinement Mechanisms

- If an error is found, the discovering node will transmit an Error Flag, thus destroying the bus traffic.
- The other nodes will detect the error caused by the Error Flag (if they haven’t already detected the original error) and take appropriate action, i.e. discard the current message.

- Each node maintains two error counters:
  - the Transmit Error Counter and
  - the Receive Error Counter.

- There are several rules governing how these counters are incremented and/or decremented.
  - In essence, a transmitter detecting a fault increments its Transmit Error Counter faster than the listening nodes will increment their Receive Error Counter. This is because there is a good chance that it is the transmitter who is at fault!

- A node starts out in Error Active mode.
- When any one of the two Error Counters raises above 127, the node will enter a state known as Error Passive
- When the Transmit Error Counter raises above 255, the node will enter the Bus Off state.

  - An Error Active node will transmit Active Error Flags when it detects errors.
  - An Error Passive node will transmit Passive Error Flags when it detects errors.
  - A node which is Bus Off will not transmit anything on the bus at all.

- The rules for increasing and decreasing the error counters are somewhat complex, but the principle is simple:
  - transmit errors give 8 error points, and
  - receive errors give 1 error point.
  - Correctly transmitted and/or received messages causes the counter(s) to decrease.

Example

- Example (slightly simplified): Let's assume that node A on a bus has a bad day. Whenever A tries to transmit a message, it fails (for whatever reason).
  - Each time this happens, it increases its Transmit Error Counter by 8 and transmits an Active Error Flag.
  - Then it will attempt to retransmit the message.. and the same thing happens.
  - When the Transmit Error Counter raises above 127 (i.e. after 16 attempts), node A goes Error Passive.
  - The difference is that it will now transmit Passive Error Flags on the bus.
  - A Passive Error Flag comprises 6 recessive bits, and will not destroy other bus traffic – so the other nodes will not hear A complaining about bus errors.
  - However, A continues to increase its Transmit Error Counter. When it raises above 255, node A finally gives in and goes Bus Off.

Higher Layer Protocols

- The CAN standard defines the hardware (“the physical layer” – there are several) and the communication on a basic level (“the data link layer”).
  - The CAN protocol itself just specifies how to transport small packets of data from point A to point B using a shared communications medium. It 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.
- In order to manage the communication within a system, a **higher layer protocol (HLP)** is required. The term HLP is derived from the OSI model and its seven layers.
- The HLP typically specifies things like:
  - Start-up behaviour
  - How to distribute message identifiers among the different nodes in a system
  - How to translate the contents of the data frames
  - Status reporting within the system

Different Higher Layer Protocols

- CanKingdom
- CANopen
- CCP/XCP
- DeviceNet
- J1939: Introduction and Standards overview
- MilCAN
- NMEA 2000®
- OSEK/VDX
- SDS
- EnergyBus

Other Related Protocols and Standards

- There are a number of protocols and standards closely related to CAN, but which are not higher layer protocols. Read about some of them below:
- LIN bus
- J1587
- J1708
- J2534
- RP1210A, RP1210 B