# Serial Buses in Industrial and Automotive Applications

**Presented by**
**Neelima Chaurasia**

**Class: #368**

## Overview

As consumer electronics, computer peripherals, vehicles and industrial applications add embedded functionality, demand is growing for inexpensive, fast and reliable communication media to serve these applications. As a result, increasing numbers of processors and controllers are now integrated with different kinds of buses. These buses are used for communication with PC software, development systems such as emulators, or other devices within the network. Communication usually occurs in either serial or parallel mode, with serial mode much more prevalent at this time.

The common integrated serial buses in microcontrollers are Universal Asynchronous Receiver Transmission (UART), Serial Communication Interface (SCI), Synchronous Peripheral Interface (SPI), Inter-integrated Circuit ($I^2C$), and Universal Serial Bus (USB), along with the automotive serial buses: the Controller Area Network (CAN) and Local Interconnect Network (LIN). These buses differ in speed, physical-interface requirements and communications methodology. This paper offers an overview of serial buses, drivers and physical-interface requirements for embedded system design. Guidelines for selecting the optimal buses and a comparison chart are provided. For illustration purposes, a microcontroller-based design is used.

## Serial vs. Parallel

The main advantage of a serial over a parallel bus is the small number of wires required for communication. For instance, the LIN serial bus used in the automotive industry needs only one wire to communicate with slave devices; a Dallas 1-Wire® bus uses one wire to carry both signal and power.

Fewer wires mean fewer pins in the controller. A parallel bus integrated in a microcontroller typically requires eight or more lines, depending on the address and data bus size designed. Therefore, a chip that integrates a parallel bus needs at least eight pins to interface with external devices. This increases the overall chip size. However, using the serial bus it is possible to integrate the same chips into a smaller package.

Another factor to consider is that during the PCB board design, the parallel bus requires more lines to interface with other peripherals. This leads to a bigger, more complex PCB board, increasing the cost of the hardware.

It is easy to add a new device on a serial network without affecting the existing devices on the network For example, obsolete devices on the bus can simply be removed and replaced with the new IC.

Fault diagnostics and debugging are also very simple for serial buses. It is easy to trace a defective device on the network and replace it without disturbing the network.

On the other hand, parallel buses are fast compared to serial. A parallel processor bus, "Redwood" from Rambus, can run as fast as 6.4GHz, whereas the top serial speeds never exceed a few Megahertz.

## Common Serial Protocols in Industrial and Automotive Applications

### UART

The Universal Asynchronous Receiver Transmission (UART) is a general-purpose serial data bus for asynchronous communication. The bus is bi-directional and capable of full-duplex transmission and reception between two receive and transmit pairs.

In embedded designs UART is used for communication with PC software such as monitor debugger or other peripheral devices such as EEPROM memory and RTC.

#### *UART Communication*

UART first converts the received parallel data into serial data for transmission. The message frame starts with a low-level start bit followed by seven or eight data bits, an optional parity bit and one or more high-level stop bits. When a receiver sees the start bit, it recognizes that the data is ready to be sent and tries to synchronize with the transmitter clock frequency. If parity is selected, the UART adds the parity bit at the end of the data bits. The parity bit helps in checking the error of the data received.

In the reception process, the UART removes the start and stop bits from the message frame, checks the parity for incoming bytes, and converts the data bytes from serial to parallel. The UART also generates additional signals to indicate the status of the transmission and reception. For example, if a parity error occurs, the UART sets the parity flag.

The diagram below shows a data frame transmitted by UART.


ST: Start bit
D0-D7: Data bit
P: Parity bit
SP: Stop bit

**Figure 1: UART Data Frame**

#### *Data Direction and Communication Speed*

Typically data will be transmitted with least significant bit (LSB) first. However, some UARTs allow the flexibility of the LSB or most significant bit (MSB) first option.

UARTs integrated in microcontrollers can transfer data at speeds ranging from a few hundred bits per second up to 1.5 Mbps. For example, the high-speed UART embedded in the ÉlanSC520 microcontroller can communicate at a speed of up to 1.1152 Mbit/s. The UART baud rate is also limited by the distance (length of wires) between the transmit and receive pair.

### *Types of UARTs*

There are two kinds of hardware solution available for UARTs:

- **Hardware that supports only asynchronous communication.** Typically this is referred by the name UART. In Motorola microcontrollers, it is referred as the serial communication interface (SCI).

- **Hardware that supports both synchronous and asynchronous communication.**
  The Universal Synchronous Asynchronous Receiver Transmitter (USART) in Microchip microcontroller and the UART in Fujitsu's microcontroller are good examples of this kind of hardware.

### *UARTs in Computers*

The UART is the key component of the serial communication port on a computer. In the computer the UART is connected to a circuitry that generates a signal compliant with the RS232 specification. The RS232 standard defines the logic "1" signal to be between 3 and 25 volts, and the logic "0" signal to be between -3 and -25 volts with respect to the ground. Therefore, when UART in a microcontroller is connected with the PC, it requires a RS232 driver to convert the voltage levels. The MAX232 family from Maxim is an example of a driver IC used for converting the voltage level and for connecting the UART to the serial port of the computer.

## SPI

The synchronous peripheral interface (SPI) is a full-duplex, synchronous serial bus developed by Motorola. The bus is used frequently for communication with slow peripheral devices such as EEPROM, ADC, FRAM and the display driver.

### *SPI Communication*

The bus communication is based on a master-slave configuration. There are four signals:

- MOSI: Master Out/Slave In
- MISO: Master In/Slave Out
- SCK: Serial Clock
- SS: Slave Select

The number of slave-select pins on the chip determines the number of devices to be connected on the bus. The figure below shows the master-slave connection for two slave-select pins.

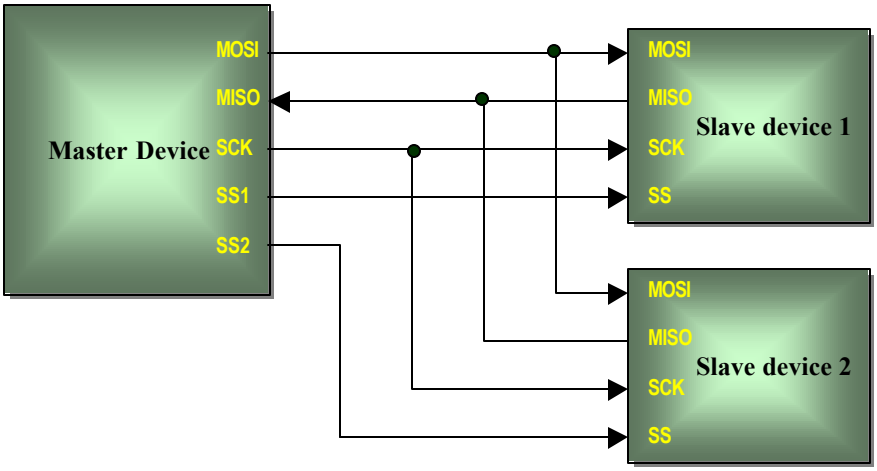



**Figure 2: Single Master, Two Slaves Implementation**

In an SPI transfer, data is simultaneously transmitted and received. The data is clocked based on clock pulses from the master. Motorola has not defined any common SPI clock specifications. However, the most common clock settings are based on two parameters: clock polarity (CPOL) and clock phase (CPHA). CPOL defines the active state of the SPI serial clock; CPHA defines the clock phase in respect of the SO-data bit. The settings of CPOL and CPHA determine the edge of the clock at which data will be sampled.

The figure below shows the data latch for different combinations of CPOL and CPHA.

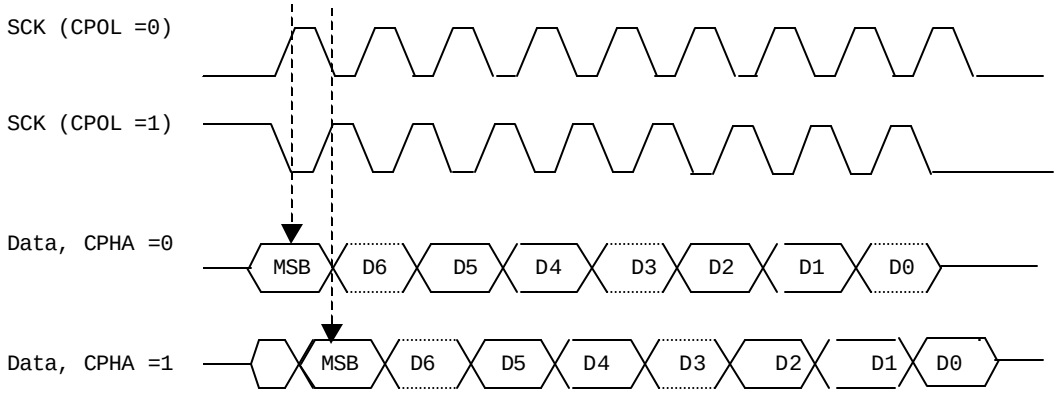



**Figure 3: SPI data transfer for different clock**

### *Data Direction and Communication Speed*

SPI transfers the serial data as the MSB first. The baud rate can be as fast as 5Mbit/s depending upon the SPI hardware. For example, Xicor's SPI serial devices can transfer data as fast as 5Mbit/s.

### *SPI vs. UART*

SPI communication is faster than UART communication. Both can be used for moderate-speed peripheral communication such as non-volatile EEPROM memory. However, SPI is more commonly used with EEPROM or Digital to Analog Converter (DAC) communications.

Some UARTs can support SPI communications. In those cases, a general-purpose IO pin will serve as a slave-select pin.

## $I^2C$

The Inter-Integrated Circuit ($I^2C$) is a two-wire synchronous bus developed by Philips. Like SPI, this bus is useful for communication with slow peripheral devices such as EEPROM, ADC, DAC and LCD.

### *$I^2C$ Communication*

$I^2C$ is a half duplex, multi-master bus. The $I^2C$ network has one or more master devices and many slave devices. The information is carried by two serial wires: serial data (SDA) and serial clock (SCL). The figure below shows an example of a connection using two masters and three slaves.

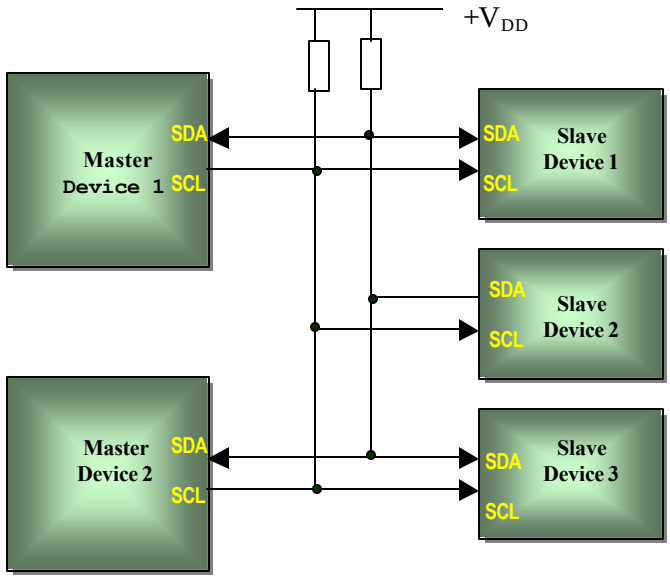



**Figure 4: $I^2C$ Multi Master Multi Slave Implementation**

Each device on the network is pre-assigned a unique 7- or 10-bit address. Philips assigns the address to the manufacturer of the device. There is also a special address for high-speed communication, and a general call address for communication with all the devices on the network. The advantage of 10-bits addressing is that it allows more devices (up to

1024) to be on the network. However, the number of device on the bus depends on the bus capacitance, which is limited to 400pF.

The master device initiates the data transfer and is responsible for supplying the clock signal for communication. The communication starts with the high to low transition of the data line (SDA), while the clock signal (SCL) is high. This is followed by a 7 or 10 bit slave address, a data direction bit (R/W), an acknowledgement bit and stop condition. The stop condition is defined as low to high transition of the data line when the clock signal is high. Every data byte is 8 bits long. However, there is no restriction on the number of bytes transmitted per transfer.

Since $I^2C$ is a multi-master bus, it is possible that two or more masters might try to access the bus simultaneously. The master that puts a "1" on the bus while the clock signal is high wins the arbitration.

$I^2C$ has three different modes of operation: standard, fast and high-speed (Hs) mode. When using the fast and high-speed modes, it is possible that a slave device may be unable to process the data as fast as a master. In this situation, the slave holds the bus by pulling the SCL line low. This forces the master device into the wait state until the slave is ready.

The diagram below shows a complete data transfer on the I2C bus.

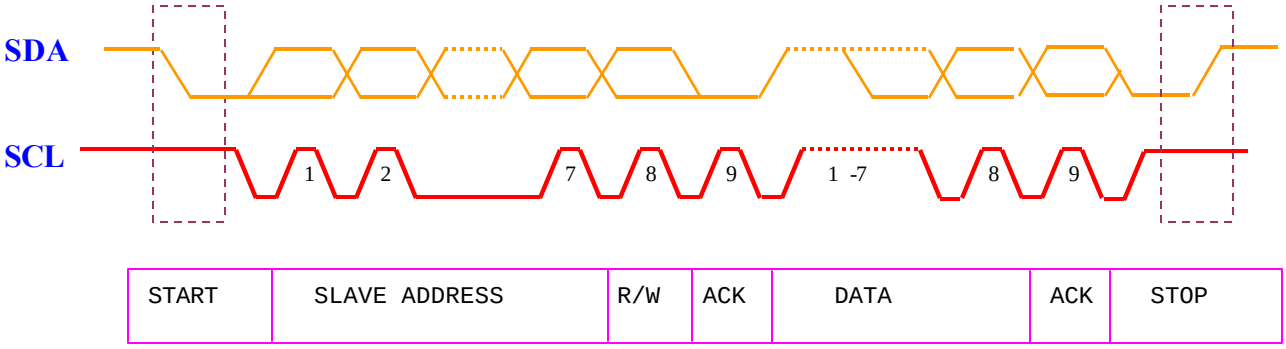



**Figure 5: $I^2C$ DATA Transfer**

***Data Direction and Communication Speed***

Data is transmitted with the MSB first. The $I^2C$ bus is designed for the three-data transmission speed, each with downward compatibility:

Low Speed: The data can be transmitted from 0 to 100 kbps.
Fast Speed: The data can be transmitted up to 400 kbps.
High-Speed: The data can be transmitted up to 3.4 Mbps.

### *$I^2C$ vs. SPI*

Both $I^2C$ and SPI are useful for communication with slow devices. However, because of its high data rate, SPI is preferred over the $I^2C$ bus. In addition, SPI has a built-in address capability, this reduces the overhead in the software and hardware for designing an additional register to test the address.

## Controller Area Network

The Controller Area Network (CAN) is a multi-master asynchronous serial bus. Because of its excellent error-handling mechanism and the reliability of its data transmission, this bus has become very popular in the automotive industry and is gaining popularity in the healthcare industry for safety-critical equipment.

CAN was originally developed by the Robert Bosch Company in Germany to provide a cost-effective communications bus for in-car electronics. Now it is an international standard and is documented in ISO 11898 for high-speed applications and in ISO 11519 for lower-speed applications.

### *CAN Communication*

When the bus is free any CAN node can start the transmission. If two or more nodes start the transmission simultaneously, then the access conflict is resolved by the bit-wise arbitration using the identifier. The CAN bus is a broadcast type of bus. Therefore, all nodes receive the data on the bus. The filtering mechanism on the hardware decides whether the message is intended for the node or not.

### *Types of Message Frames*

There are four types of message frames.

- **`Data Frame:`** This frame carries data from a transmitter to a receiver. There are two data-frame formats based on the CAN specification. The only essential difference is in the length of the identifier. The CAN standard frame, also known as CAN 2.0A, supports a length of 11 bits for the identifier. The CAN extended frame, also known as CAN 2.0B, supports a length of 29 bits for the identifier.

The figure below shows the CAN data frame for both specifications.

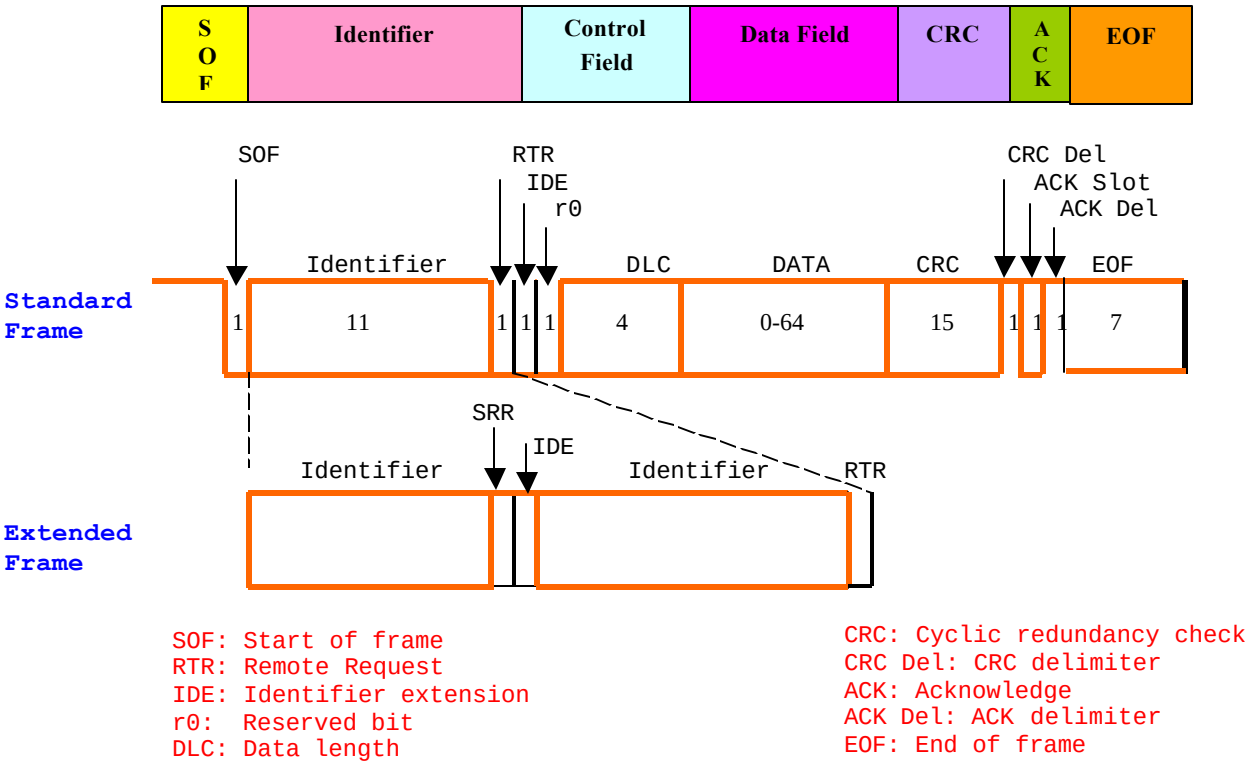
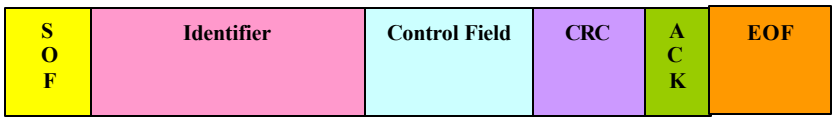



**Figure 6: CAN Standard and Extended Data Frame**

- **Remote Frame:** This frame is transmitted by a receive CAN node to request the data frame with the identifier specified in the remote frame.




- **Error Frame:** This frame notifies other units about any bus error. On reception of this frame the transmitter automatically tries to retransmit the message.


- **Overload Frame:** The overload frame is sent by a busy CAN node to request an extra delay between the preceding and succeeding data frame.

*CAN Hardware Terminology*

- **Basic CAN:** This inexpensive CAN controller has limited transmit/receive message buffers, and limited filtering mechanisms for CAN messages.

- **Full CAN:** The full CAN is an expensive, high-performance CAN controller with eight or more message buffers for transmit and receive. For example, Fujitsu's integrated CAN microcontroller offers 16 message buffers for transmit and receive. In addition, Fujitsu's MB90443 triple-CAN microcontroller offers the flexibility to combine the message buffer of two CAN controllers into one to form a 32-message buffer.

- **Standard CAN:** This CAN controller is capable of handling messages with only an 11-bit identifier.

- **Extended CAN:** This controller can handle messages with both 11- and 29-bit identifiers.

- **Time triggered CAN (TTCAN):** This CAN controller schedules the CAN messages based on time or event triggering. The scheduling increases the total performance and deterministic behavior of a CAN network.

*Data Direction and Communication Speed*

The data bytes are transferred with MSB first. A 0-to-8 data byte can be transferred in one transmission. The maximum CAN bus speed is 1 Mbit/s.

*CAN in Vehicles*

The example below shows how the CAN network is connected in the car using Fujitsu's 16-bit CAN microcontroller.

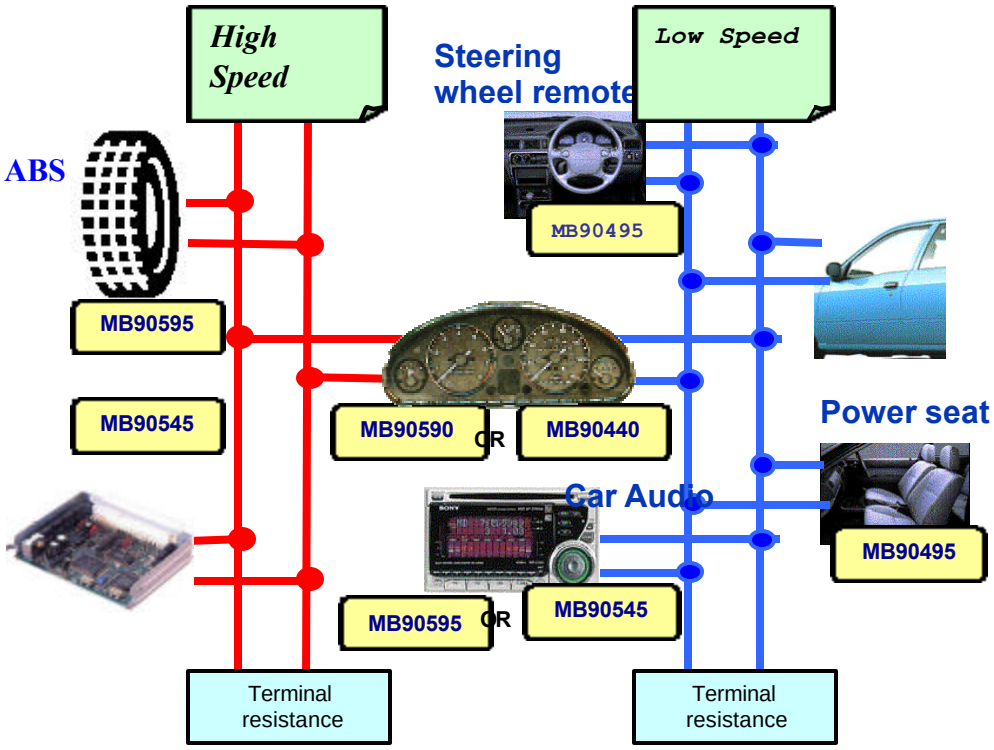



Figure 7: CAN in Vehicles

### *Physical Interface*

Most CAN microcontrollers require an external transceiver to connect to the physical CAN bus. Here is the list of some of the transceivers available today:

**For High-speed CAN**

- Philips 82C251
- Texas Instrument SN65/75LBC031
- Bosch CF150, C250
- Unitrode UC5350

**For Low-speed CAN**

- Philips 82C252, TJA1053
- Siemens TLE 6252G

**For Single-wire CAN**

- Philips AU5790
- Infineon TLE 6255
- Delphi DK166153

## LIN

A Local Interconnect Network (LIN) is a low-cost, one-wire serial bus capable of performing full-duplex serial communication. LIN is used in distributed electronics systems in cars such as communication with smart sensors and actuators. The LIN protocol can be implemented on the low-cost common UART/SCI interface, which is available in almost all microcontrollers.

### *LIN Communication*

The LIN network consists of one master and multiple slave devices. The master device initiates all the communication.

All nodes perform the slave communication task that includes the transmit and receive task. In addition, the master node performs the master transmit task. The master task decides when and which frame will be transferred on the bus. In this way there is no bus arbitration and the worst-case time for each message can easily be calculated. When a message frame is transmitted, only one slave device gets activated after reception and filtering of the identifier.

All messages on the bus are sent as a frame. A frame consists of a header and response field. The master always sends the header on the bus. The header consists of at least a 13-bit break field, a sync byte and a 6-bits identifier that is in the range of 0 to 63. The response field consists of two, four and eight data bytes, and the checksum field with inverted 8 bit sums with carry of all data bytes and identifier.

The figure below shows the mapping of all fields on a message frame.

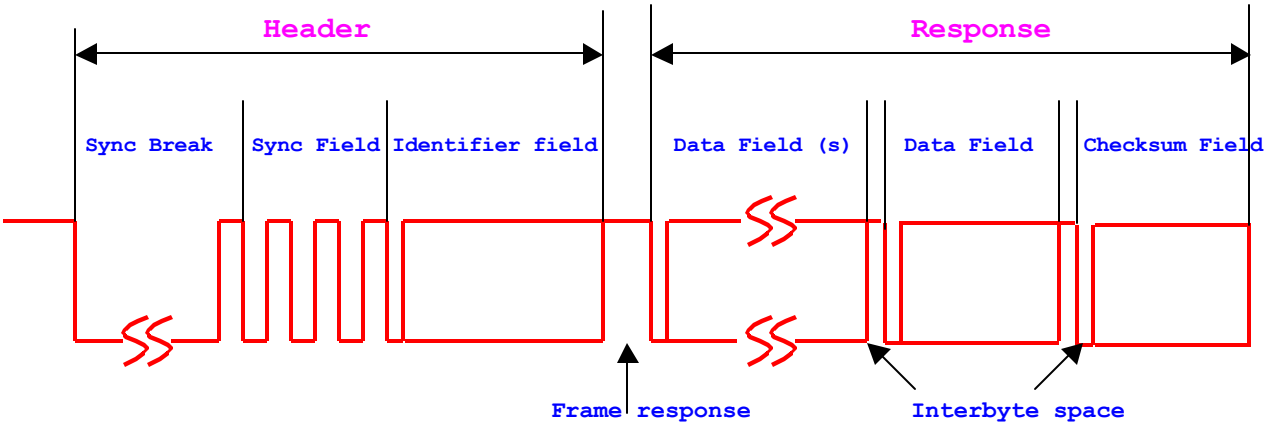



**Figure 8: LIN Data Frame**

***Data Direction and Communication Speed***

The data bytes are transferred with LSB first. The maximum speed of a LIN bus is 20kbit/s.

***Physical Interface***

Very few microcontrollers integrate the dedicated LIN hardware. Most vendors provide support using either SCI or UART. Since the LIN physical layer is a single-wire 12V bus derived from the ISO9141 standard for automotive diagnostics, an external LIN transceiver is required to convert the voltage level. Motorola's IMC33689 LIN and Infineon's TLE 6259-2G are examples of the transceivers available in the market.

***CAN vs. LIN***

Both CAN and LIN are used in the automotive industry. CAN is used for high- and low-speed networks in a vehicle, and LIN is used only for low-speed networks, such as a door-control unit. In most respects, CAN is more expensive and reliable than LIN. So there is a trade-off between the reliability and the additional cost to design the hardware and software-using CAN. Since LIN costs less, and easy to implement on an UART, it is expected that it will replace the low-speed CAN network applications.

## Other Common Serial Buses

Other buses used in the microcontroller industry are RS422, RS485, USB and Microwire.

RS422 and RS485 communication can be performed using an UART. Therefore, no additional hardware is implemented in microcontrollers for these buses.

The popularity of the USB bus has inspired microcontroller vendors to integrate the USB controller into their microcontrollers. It is easy to add peripheral devices on the USB bus

without needing to reboot the system. Cypress Semiconductor and other industry leaders have a range of USB chips to meet market demand.

Microwire, a one-wire bus developed by National Semiconductor, is available in many microcontrollers and non-volatile memory such as EEPROM and ADC. The bus offers synchronous communication like SPI and can be used in place of SPI. Some microcontroller vendors support the Microwire bus by using the UART.

A new upcoming bus in the automotive industry, FlexRay, can be used in place of a CAN bus and will support data rates up to 10Mbit/sec, which is 10 times the speed of the CAN bus. Currently microcontroller vendors are working on developing devices with FlexRay.

## Guidelines for Selecting the Right Bus

The criteria for selecting a bus vary from application to application. Here are some guidelines for making the best selection:

- Evaluate the system cost for connecting the various devices on the network using the different serial buses. For instance, a system that requires only the control function can be managed with a low-cost serial bus such as LIN in an automotive application.

- Identify what is most important to you, e.g. efficiency, speed or reliability. For example, a safety-critical system where reliability is extremely important, CAN is a good choice.

- Define how many devices will be on the network and what the bus capacitance will be. Some serial buses have limitations on the number of devices that can be connected on the network.

- Be careful about the distance between the devices. Some serial buses only support short-distance communication.

- If the application is automotive, it is a good idea to use CAN or LIN because of their robustness, fault tolerance and reliability of transmission.

## Summary Table

The table below summarizes some key parameters for the above buses.

**Table 1: Bus Summary Table**

| Name of the bus | Number of wires* | Type of communication | Multi master support | Data rate | Number of devices on the bus | Cable length (meters) |
|---|---|---|---|---|---|---|
| UART | 2 | Async | No | 3kbps to 4Mbps | 2 | 1.5m @128kbps |
| SPI | 3 | Sync | No | >1Mbps | <10 | <3m |
| $I^2C$ | 2 | Sync | Yes | 3.4Mbps max | <10 | <3m |
| CAN | 2 or 1 | Async | Yes | 20kbps to 1 Mbps | 128 | 40m @1mbps |
| LIN | 1 | Async | No | 20kbps max | 16 | 40m |

**Notes:**

***** Not including ground signal.

## References

### Books


1. Byron Putnam and Byron W. Putman, *"Rs-232 Simplified: Everything You Need to Know About Connecting, Interfacing & Troubleshooting Peripheral Devices".*
2. Konrad Etschberger, "*Controller Area network*"


### Internet


1. http://www.mct.net/faq/spi.html
2. www.can-cia.org
3. http://www.lin-subbus.org/
4. http://www.vector-cantech.com
5. http://www.kvaser.com/can/index.htm
6. http://www.flexray.com


### Bus Specification


1. "The *$I^2C$-bus specification,"* version 2.1, Philips Semiconductors.
2. "*CAN specification 2.0,"* Robert Bosch GmbH