DISTRIBUTED EMBEDDED SYSTEMS; NETWORKS FOR IOT

Introduction

Distributed Embedded System:

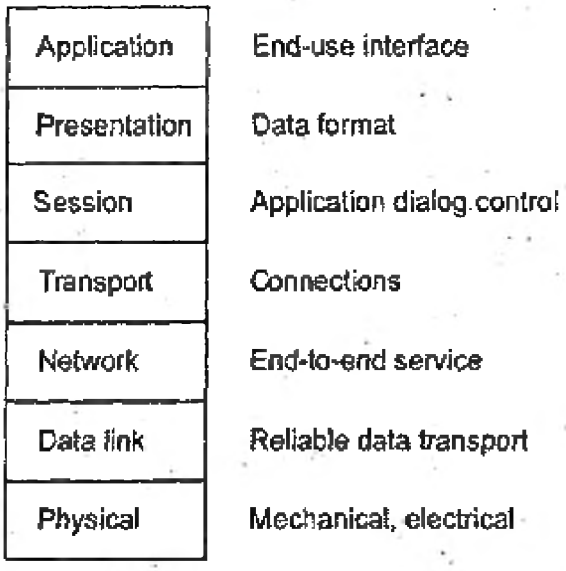
In a distributed embedded system, several Processing Elements (PEs) are connected by a network that allows them to communicate. Processing elements may includes DSP, CPU or microcontroller. Nonprogrammable unit such as the ASICs is also used to implement as PE.

It differs from parallel computing or multiprocessor with the reason that its individual nodes have much higher independence for passing the messages.

For analyzing the performance of a distributed embedded system, it is required to know many basic terms which are associated with it along with its problems and the ways to solve, it. The approaches to solve these problems include simulation technique and also scheduling technique such as holistic scheduling technique.

Network Abstractions: The OSI Model

Networks are complex systems which ideally provide high-level services while hiding many of the details of data transmission from the other components in the system.



The International Standards Organization (ISO) has developed a seven-layer model for networks known as Open System Interconnection (OSI) models. This OSI layers will help us to understand the details of real networks.

The seven layers of the OSI model is shown in Fig which are proposed to cover a broad spectrum of networks and their uses. Some networks may not need the services of one or more layers because the higher layers may be totally missing or an intermediate layer may not be necessary.

Any data network should fit into the OSI model which includes seven levels of abstraction known as layers:

(i) Physical Layer:

This layer defines the basic properties of the interface between systems which includes the physical connections (plugs and wires), electrical properties, basic functions of the electrical and physical components, and the basic procedures for exchanging bits.

(ii) Data link Layer:

The primary purpose of this layer is error detection and control across a single link.

(iii) Network Layer:

This layer defines the basic end-to-end data transmission service which is particularly important in the ihulti-hop networks.

(iv) Transport Layer

This layer defines connection-oriented services which ensures that data are delivered in the proper order and without errors across multiple links. This layer may also try to optimize network resource utilization.

(v) Session Layer

A session layer provides mechanisms for controlling the interaction of end-user services across a network, such as data grouping and check pointing.

(vi) Presentation Layer

This layer defines data exchange formats and provides transformation utilities to an application programs.

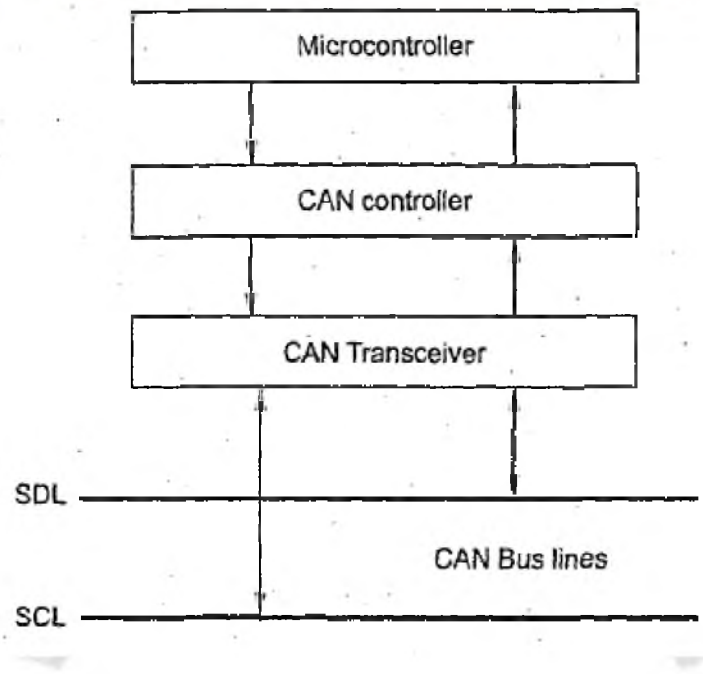
(vii) Application Layer

The application layer provides the application interface between the network and end-user programs.

CAN Bus

Definition:

A Controller Area Network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each, other'-, applications without a host computer.

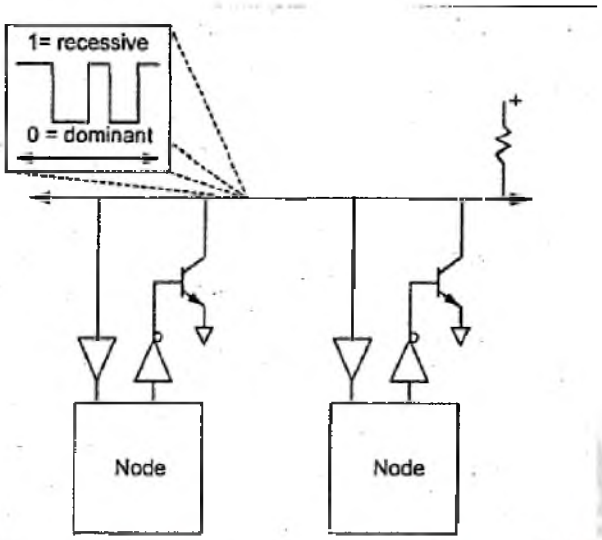


The CAN bus was designed for automotive electronics and first used in production cars in 1991. It is well suited to the requirements of automotive electronics such as reliability, low power consumption, low weight, and low cost.

A CAN uses bit-serial communication and runs at rates of 1 Mb/s over a twisted pair connection of 40 m. An optical link can also be used. This bus protocol supports multiple masters on the bus.

(1) Physical Layer

Each node in the CAN bus has its own electrical drivers and receivers that connect the node to the bus in wired-AND fashion. In CAN terminology, a logical 1 on the bus is called recessive and a logical 0 is dominant.



When all nodes are transmitting Is, the bus is said to be in the recessive state-, when a node transmits a0, the bus is in the dominant state. Data are sent on the network in packets known as data frames.

CAN is a synchronous bus because all transmitters must send at the same time for bus arbitration to work.

(2) Data Frame

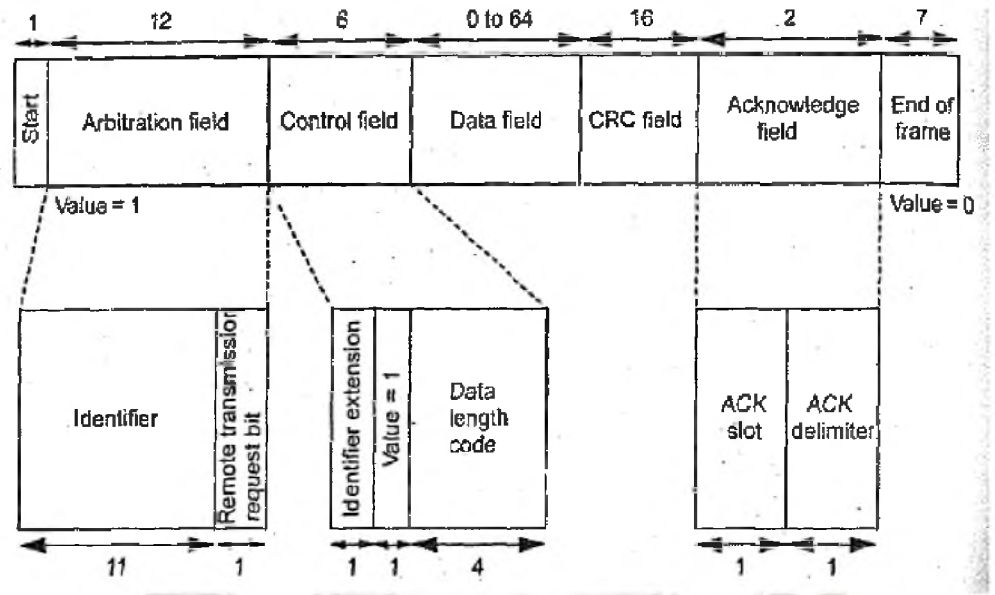
The format of a CAN data frame is shown in Fig . A data frame starts with a 1 and ends with a string of seven zeroes. The first field in the packet contains the packet's destination address which is known as the arbitration field.

The destination identifier is 11 bits long. The trailing Remote Transmission Request (RTR) bit is set to 0 if the data frame is used to request data from the device specified by the identifier.

When RTR = 1, the packet is used to write data to the destination identifier. The control field provides an identifier extension and a 4-bit length for the data field with a 1 in between them.

The data field is from 0 to 64 bytes, depending on the value given in the control field. A Cyclic Redundancy Check (CRC) is used for error detection.

The acknowledge field is used to check whether the frame was correctly received.



(3) Arbitration

Control of the CAN bus is arbitrated using a technique known as Carrier Sense Multiple Access with Arbitration on Message Priority (CSMA/AMP).

CAN encourages a data-push programming style. Network nodes transmit synchronously, so that they all start sending their identifier fields at the same time.

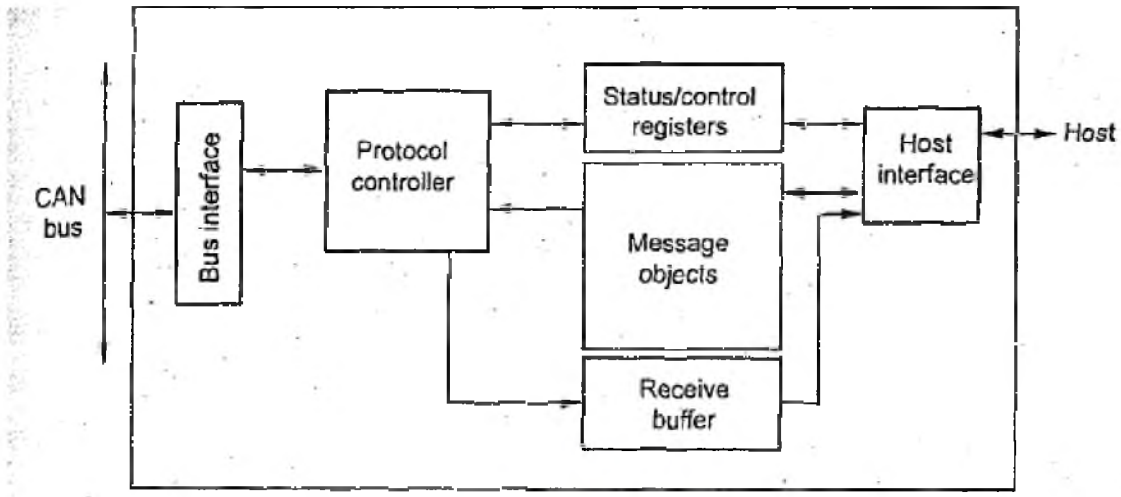
When a node hears a dominant bit in the identifier if it tries to send a recessive bit, it stops transmitting.

Remote Frames:

A remote frame is used to request data from another node.

(4) Architecture

The basic architecture of a typical CAN controller which implements the physical and data link layers; because CAN is a bus, it does not need network layer services to establish end-to-end connections.



The protocol control block is responsible for determining when to send messages, when a message must be resent due to arbitration losses, and when a message should be received.