     SOC BUSES-A SURVEY AND COMPAISION

Hagar Mostafa

Mohamed Soliman

Alaa Ahmed

Samar Mohamed

The comparative analysis of multiple System On Chip buses architecture is given in this paper. A brief introduction to the SOC bus is discussed then a comparative performance analysis is covered including the main characteristics of the buses and interfaces with respect to topology, arbitration method, bus-width, types of data transfers and applications.

# I.       INTRODUCTION

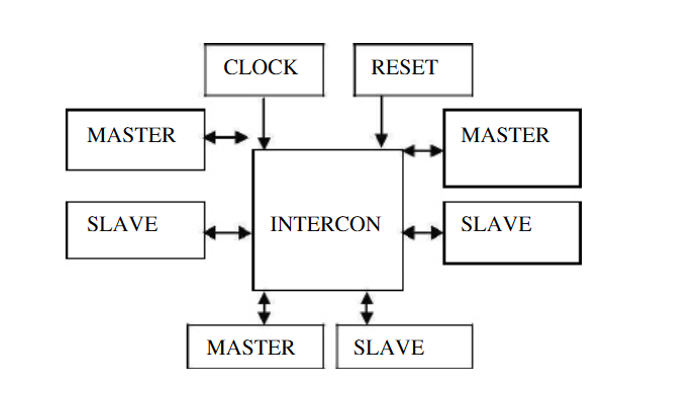
Shrinking process technologies and increasing design sizes have led to highly complex billion-transistor integrated circuits (ICs). As a consequence, manufacturers are trying to integrate large numbers of components on a chip. A heterogeneous system-on-a-chip (SoC) might include more no of programmable components such as general purpose processors cores, digital signal processor cores, or application specific intellectual property (IP) cores, as well as an analog front end, on-chip memory, I/O devices, and other application specific circuits.  [1]

In CMOS SoC technology, due to rapidly increasing frequencies of operation and large chip size, On-chip bus based on Industry standards is among the top challenges. Usually, IP cores are designed with communication protocols and many different interfaces. Integration of such cores in a SoC often requires optimal logic. To avoid this problem, Standards of on-chip bus structures were developed [1]. Currently there are publicly available bus architectures from leading manufacturers, such as CoreConnect from IBM, AMBA from ARM, STBus from STMicroelectronics, and others. This paper focuses on multiple SoC protocols providing a comparative analysis of the main characteristics of the buses.

# II.      SYSTEM ON CHIP BUSES; COMPUTER DOMAIN

SoC Buses are classified into many categories based on the desired application. Every bus defines a set of signals, which utilizes “Master” and “Slave” architectures, connect to each other by interface called “INTERCON”. This “INTERCON” basically handle data transfer between Master and Slave IP Cores. Master start data transmission generating bus cycle, and sending address and control signal to Slave IP Core. Slave in turn receive bus cycle, so respond with specified address. SoC buses support different types of interconnection to interface Master and Slave. The interconnection includes Point to Point, Shared bus, Crossbar Switch, and Data Flow.  In this paper we are going to focus on three main categories, each one will include a group of buses.

**A. Wishbone**is a SoC Bus which is commonly used in FPGA. Wishbone is a portable System on Chip design methodology. SoC design methodology interface wishbone bus, elevating integration problems and making SoC highly custom. Wishbone provide a portable interface to the IP Cores and use as an internal bus for SoC. The objective behind standard interface is to provide common interface between IP Cores improving portability, reliability of the system and faster time to market by end user. The wishbone bus accomplishes these objectives at single platform, Wishbone bus defines a set of signals and buses. Wishbone utilize “Master” and “Slave” architectures, connect to each other by interface called “INTERCON”. This “INTERCON” basically handle data transfer between Master and Slave IP Cores. Master start data transmission generating bus cycle, and sending address and control signal to Slave IP Core. Slave in turn receive bus cycle, so respond with specified address. Wishbone support four interconnection to interface Master and Slave. The interconnection includes Point to Point, Shared bus, Crossbar Switch, and Data Flow[2][3].



Wishbone Bus.

WISHBONE MASTER and SLAVE interfaces can be connected together in a number of ways. This requires that WISHBONE interface signals and bus cycles be designed in a very flexible and reusable manner. The signals were defined with the following requirements:

• The signals allow MASTER and SLAVE interfaces to support point-to-point, data flow, shared bus and crossbar switch interconnections.

• The signals allow three basic types of bus cycle. These include SINGLE READ/WRITE, BLOCK READ/WRITE and RMW (read-modify-write) bus cycles. The operation of these bus cycles is described below.

• A handshaking mechanism is used so that either the MASTER or the participating

SLAVE interface can adjust the data transfer rate during a bus cycle. This allows the

speed of each bus cycle (or phase) to be adjusted by either the MASTER or the SLAVE

interface. This means that all WISHBONE bus cycles run at the speed of the slowest interface.

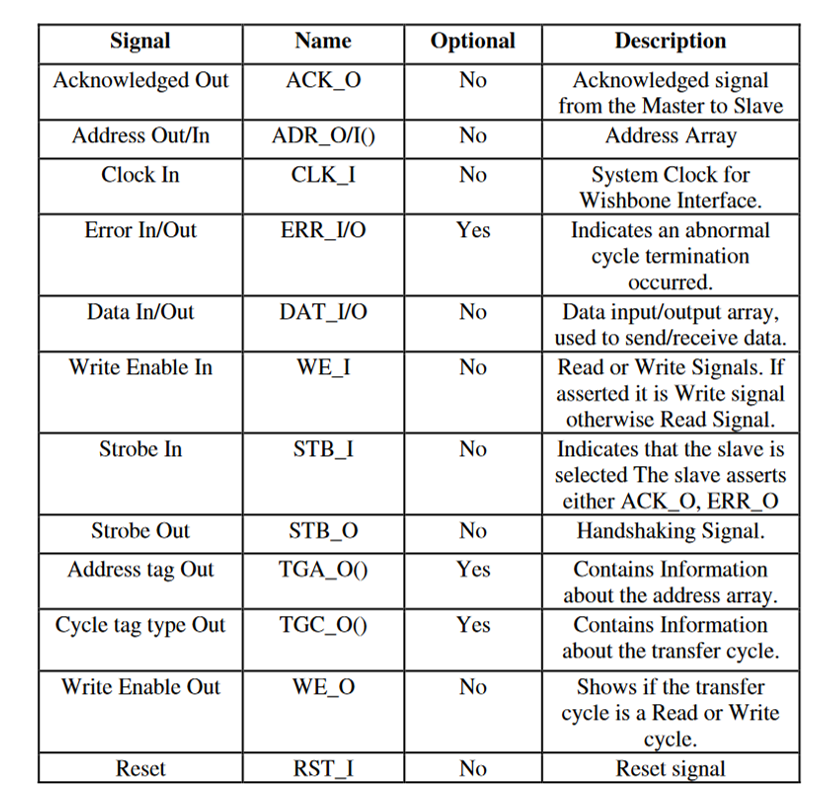
• The handshaking mechanism allows a participating SLAVE to accept a data transfer, reject a data transfer with an error or ask the MASTER to retry a bus cycle. The SLAVE does this by generating the [ACK\_O], [ERR\_O] or [RTY\_O] signals respectively. Every interface must support the [ACK\_O] signal, but the error and retry acknowledgement signals are optional.

• All signals on MASTER and SLAVE interfaces are either inputs or outputs, but are never bi-directional (i.e. three-state). This is because some FPGA and ASIC devices do not support bi-directional signals. However, it is permissible (and sometimes advantageous) to use bi-directional signals in the interconnection logic if the target device supports it.

• Address and data bus widths can be altered to fit the application. 8, 16, 32 and 64-bit data buses, and 0-64-bit address buses are defined.

• As shown in figure (2) , all signals are arranged so that MASTER and SLAVE interfaces can be connected directly together to form a simple point-to-point interface. This allows very compact and efficient WISHBONE interfaces to be built. For example, WISHBONE could be used as the external system bus on a microprocessor IP Core. However, it’s efficient enough so that it can be used for internal buses inside of the microprocessor.

• User defined signals in the form of ‘tags’ are allowed. This allows the system integrator to add special purpose signals to each WISHBONE interface. For example, the system integrator could add a parity bit to the address or data buses.



Wishbone Signals.

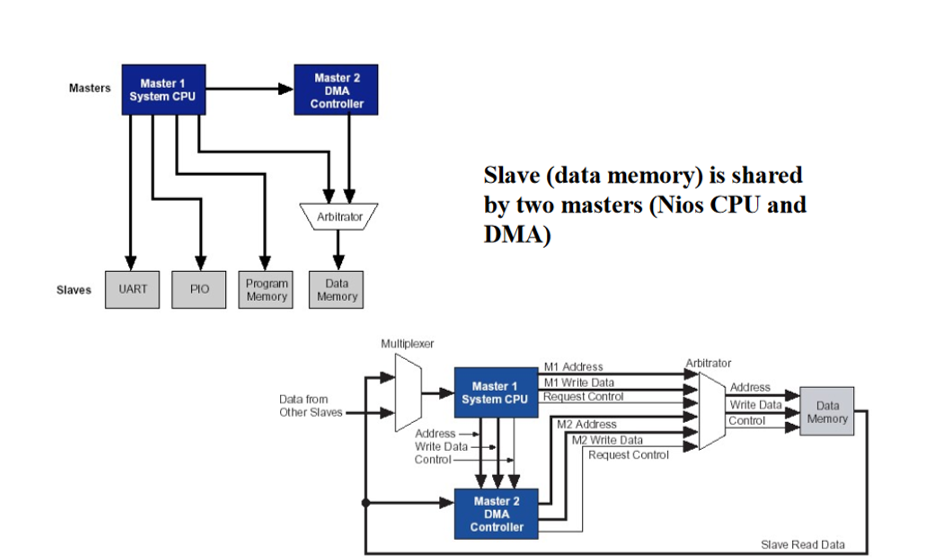
**B. Avalon**is a SoC Bus which is used in FPGA and SOPC. It has a set of predeﬁned signal types with which a user can connect IP blocks. Avalon speciﬁes the port connections between master and slave components and speciﬁes the timing by which these components communicate.  Basic Avalon bus transactions transfer one data item 8-, 16-, 32-, 64-, or 128-bits wide.  This bus supports multiple bus masters. Masters and slaves interact with each other based on a technique called slave-side (distributed) arbitration.

Avalon uses separate address, data and control lines. The Avalon bus model (switch fabric) provides the following services to Avalon peripherals connected to the bus: data-path multiplexing, address decoding, wait state generation, dynamic bus sizing, interrupt priority assignment, latent transfer capabilities, and a streaming Read and Write capabilities. AlteraXs SOPC Builder, as a system development tool, automatically generates the switch fabric logic that supports each type of transfer supported by the Avalon interface. Avalon bus module contains one slave arbitrator for each shared slave ports. Slave arbitrator performs the following:

• Defines control, address, and data paths from multiple master ports to the slave port and specifies the arbitration mechanism to use when multiple masters contend for a slave at the same time.

• At any given time, selects which master port has access to the slave port and forces all other contending masters (if any) to wait, based on the arbitration assignments.

• Controls the slave port, based on the address, data, and control signals presented by the currently selected master port.



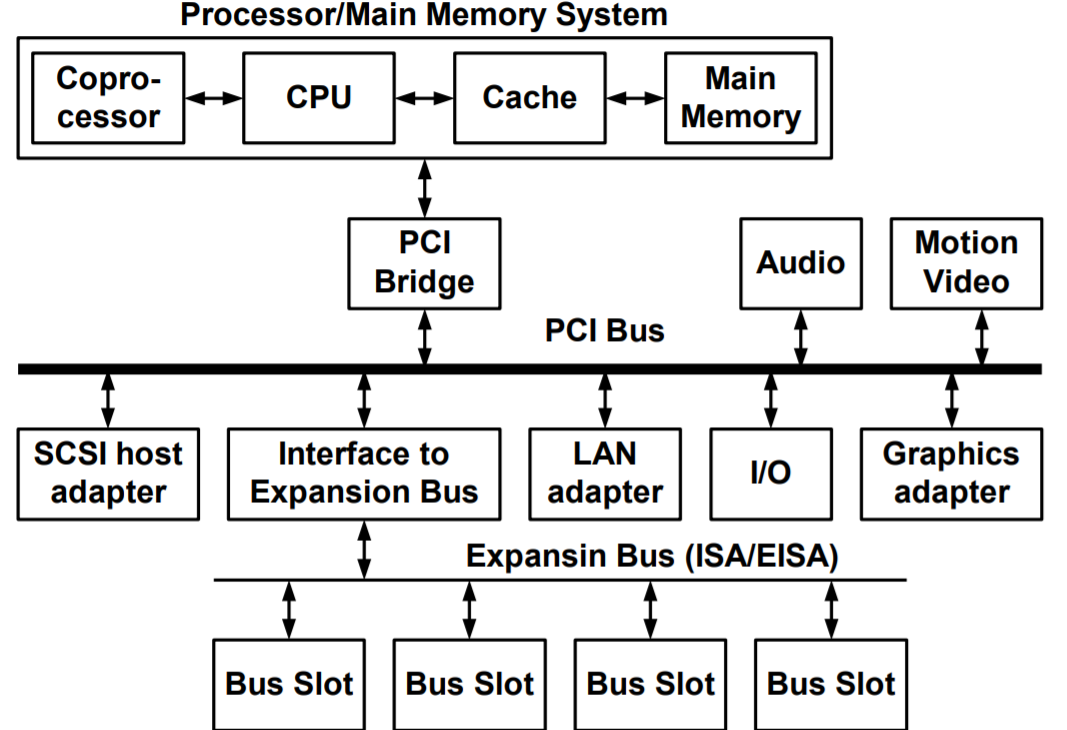
Avalon Bus Architecture.

* **Avalon interfaces** simplify system design by allowing you to easily connect components in Intel FPGA. The Avalon interface family defines interfaces appropriate for streaming high-speed data, reading and writing registers and memory, and controlling off-chip devices. Components available in Platform Designer incorporate these standard interfaces. Additionally, you can incorporate Avalon interfaces in custom components, enhancing the interoperability of designs. The specification defines the following seven interfaces:
* Avalon Streaming Interface (Avalon-ST)—an interface that supports the unidirectional flow of data, including multiplexed streams, packets, and DSP data.
* Avalon Memory Mapped Interface (Avalon-MM)—an address-based read/write interface typical of master–slave connections.
* Avalon Conduit Interface— an interface type that accommodates individual signals or groups of signals that do not fit into any of the other Avalon types. You can connect conduit interfaces inside a Platform Designer system. Alternatively, you can export them to connect to other modules in the design or to FPGA pins.
* Avalon Tri-State Conduit Interface (Avalon-TC) —an interface to support connections to off-chip peripherals. Multiple peripherals can share pins through signal multiplexing, reducing the pin count of the FPGA and the number of traces on the PCB.
* Avalon Interrupt Interface—an interface that allows components to signal events to other components.
* Avalon Clock Interface—an interface that drives or receives clocks.
* Avalon Reset Interface—an interface that provides reset connectivity.

 A single component can include any number of these interfaces and can also include multiple instances of the same interface type[4].

**C. PCI protocol**standing for  Peripheral Component Interconnect. ItisUsed in connecting the elements of modern, high performance computer systems. It evolved when ISA bus failed to keep up with speed requirement  and the need to connect more than two devices to the VL-Bus which introduced the possibility of interference with the performance of the CPU , then PCI evolved which presents a hybrid of sorts between ISA and VL-Bus.

PCI provided direct access to system memory for connected devices and offers a number of significant advantages like speed and configurability . It is a synchronous bus with clock rate of 33 MHz and nowadays it is extended to support operation at 66 MHz and  implements a 32-bit multiplexed Address and Data bus where it could support a 64-bit data bus through a longer connector slot. The PCI bus speed is  independent of the CPU’s speed and expansion of the bus could be achieved by means of a bridge. A PCI bus transfer is achieved through a set of data and address signals where it consists of one address phase and any number of data phases.



PCI Architecture

Now,How data is transferred using in PCI protocol?

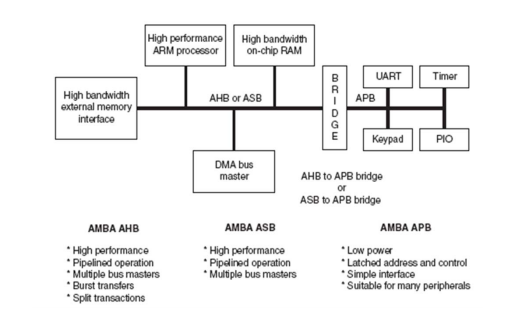
* PCI signals are active low, where PCI transaction is initiated when FRAME# signal is asserted low and hence the address phase is signaled .The next clock edge begins the first of one or more data phases in which data is transferred over the AD[31:0] signals.
* The initiator drives the C/BE[3:0]#  which  defines the type of transfer to be performed during  the - address phase and during the data phase of a transaction these signals carry byte enable information.

IRDY#  (Initiator Ready) is driven low by the initiator as an indication it is ready to complete the current data phase of the transaction , TRDY# (Target Ready )is driven low by the target as an indication it is read to complete the current data phase of the transaction. Only when  both IRDY# and TRDY# are low transaction takes place.

* The initiator signals completion of the bus transfer by de-asserting the FRAME# signal during the last data phase.
* A target may terminate a bus transfer by asserting the STOP# signal. When the initiator detects an active STOP# signal, it must terminate the current bus transfer and re-arbitrate for the bus before continuing. If STOP# is asserted without any data phases completing, the target has issued a retry. If STOP# is asserted after one or more data phases have successfully completed, the target has issued a dis-connect.
* Initiators arbitrate for ownership of the bus by asserting a REQ# signal to a central arbiter. The arbiter grants ownership of the bus by asserting the GNT# signal. Arbitration in PCI is “hidden” in the sense that it does not consume clock cycles.

**D. SATA** stands for Serial Advanced Technology Attachment, its main application is to connect the host with the hard disk driver (HDD) or the solid state disk (SSD), The SATA environment is a point-to-point connection scheme (half-duplex). SATA uses 8b/10b encoding for clock recovery and DC balance and can transmit and recieve up to 6 Gb/sec differnetial NRZ serial stream. SATA standards do not define switches or other hardware mechanisms that allow the architecture to go beyond a single device to host connection. To overcome this restriction, the SATA community has defined a hardware and software scheme that will allow more than one (or two for ATA emulation) device to be attached to a host system. This is accomplished through a mechanism known as a Port Multiplier [9].

**E. AMBA** , *Advanced Microcontroller Bus Architecture* specification by ARM defines an on-chip communications standard which is required for designing high-performance embedded microcontrollers. AMBA comprises of three buses which are the Advanced High-performance Bus (AHB), the Advanced System Bus (ASB), the Advanced Peripheral Bus (APB), in addition to Advanced extensible Interface (AXI). 



AMBA General Architecture

The architecture consists of a backbone bus which has the capability to sustain the external memory bandwidth where the on-chip memory, CPU and other Direct Memory

AMBA is hierarchically organized into two bus segments, system and peripheral bus, mutually connected via bridge that buffers data and operations between them.   AMBA does not define method of arbitration. Instead it allows the arbiter to be designed to suit the applications needs, the best.[19]

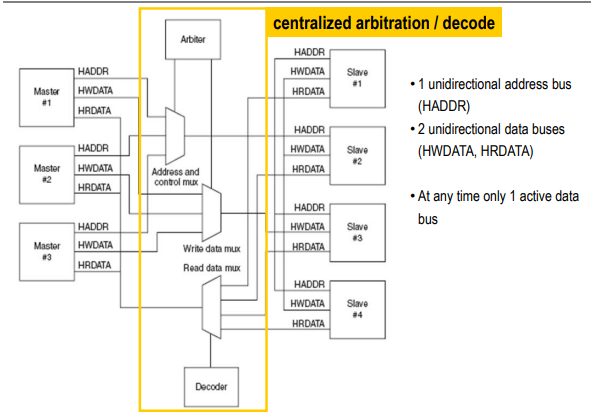
**E.I. AMBA AHB**

### 

   The AHB is mainly used for high-performance, high clock frequency system modules. It acts as the high-performance system backbone bus. It supports multiple bus masters, provides high-bandwidth operation and supports efficient connection of processors, off-chip external memory interfaces and on-chip memories with low-power peripheral macrocell functions. AHB Master, AHB Slave, AHB Arbiter, and AHB Decoder are the main components of the AHB.

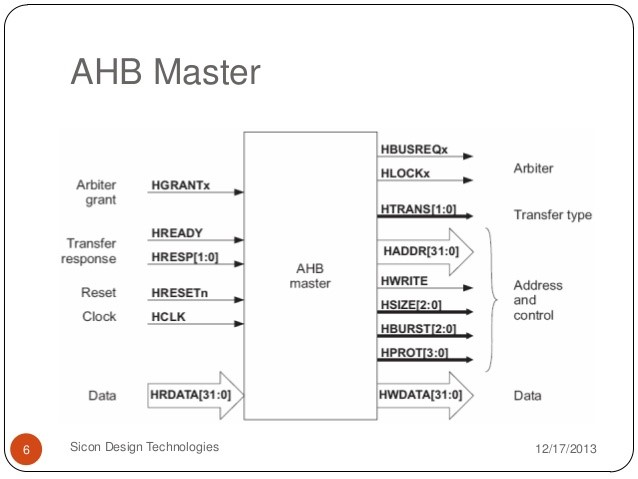
AMBA AHB is a new level of bus which sits above the APB and implements the features required for high-performance, high clock frequency systems including:

* Burst transfers.
* Split transactions.
* Wider data bus configurations (64/128 bits).



AHB Architecture

  In normal operation a master is allowed to complete all the transfers in a particular burst before the arbiter grants another master access to the bus. However, in order to avoid excessive arbitration latencies it is possible for the arbiter to break up a burst and in such cases the master must re-arbitrate for the bus in order to complete the remaining transfers in the burst.



AHB Master

* **Write data**

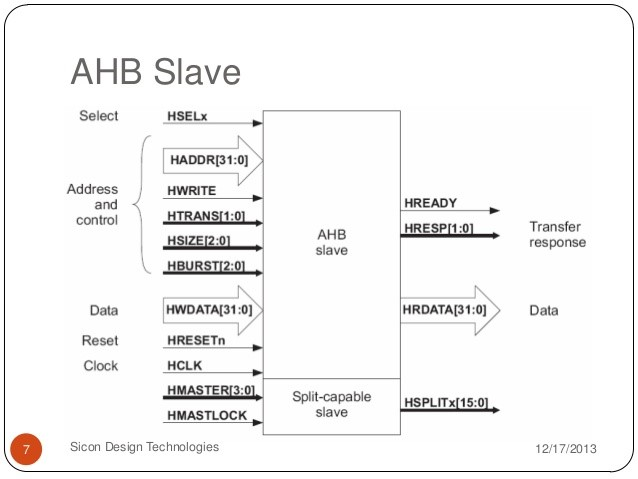
A write data bus is used to move data from the master to a slave, while a read data bus

is used to move data from a slave to the master where every transfer consists of:

• An address and control cycle.

• One or more cycles for the data.

 The address cannot be extended and therefore all slaves must sample the address during this time. The data, however, can be extended using the **HREADY** signal. When LOW this signal causes wait states to be inserted into the transfer and allows extra time for the slave to provide or sample data.



AHB Slave

**II.  AMBA ASB**

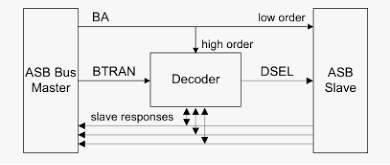
The ASB is a high-performance pipelined bus, which supports multiple bus masters.  The basic flow of the bus operation is:

1. The arbiter determines which master is granted access to the bus.

2. When granted, a master initiates transfers on the bus.

3. The decoder uses the high order address lines to select a bus slave.

4. The slave provides a transfer response back to the bus master and data is  transferred between the master and slave[21] .



ASB Architecture

**1. Decoder**

The decoder manages all transfers on the ASB bus. Each bus transfer requires three components to act:

* A bus master to start the transfer.
* The decoder to control the operation of the transfer.
* A bus slave to accept a write transfer or control a read transfer.
* **BTRAN** is used to determine how the transfer should proceed, whether a slave should be selected, if and in which state slave responses (**BWAIT**, **BERROR** and **BLAST**) should be driven. The high order bits of **BA** are used to generate the corresponding slave select line (**DSEL**).[21]

**2. Bus Master**

     This initiates a read or write transfer by driving **BTRAN[1:0]** (transfer type) and **BA[31:0]**(AMBA    address bus) and control signals. This component drives **BD[31:0] (bidirectional system data bus)** for a write transfer.



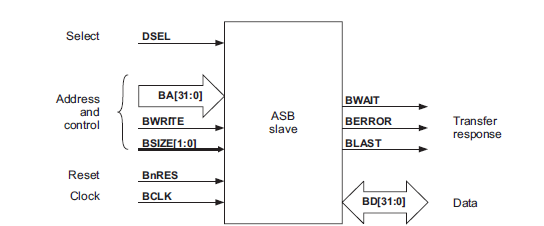
ASB Master

**3. Bus Slave**

  An ASB bus slave responds to transfers initiated by bus masters within the system. The slave uses a **DSEL** select signal from the decoder to determine when it should respond to a bus transfer. All other signals required for the transfer, such as the address and control information, will be generated by the bus master. The decoder greatly simplifies the slave interface and removes the need for the slave to understand the different types of transfer.

**4. Transfer response**

A slave must provide a transfer response in the LOW phase of **BCLK** when **DSEL** is asserted. Using the **BWAIT, BERROR** and **BLAST** signals one of the following responses must be generated.



ASB Slave

**5. Arbiter**

  The arbiter functions as follows:

1. Bus masters assert **AREQx** during the HIGH phase of **BCLK.**

2. The arbiter samples all **AREQx** signals on the falling edge of **BCLK**.

3. During the LOW phase of **BCLK** the arbiter also samples the **BLOK (**lock signal) and then asserts the appropriate **AGNTx** signal:

* If **BLOK** is LOW, then the arbiter will grant the highest priority bus master
* If **BLOK**is HIGH, then the arbiter will keep the same bus master granted. [21]

   The arbiter can update the grant signals every bus cycle. However, a new bus master can only become granted and start driving the bus when the current transfer completes, as indicated by **BWAIT** being LOW. Therefore, it is possible for the potential next bus master to change during waited transfers.

**III. AMBA APB**

The AMBA **APB**acts as local secondary bus which is used for low-power peripherals and is enclosed as a single AHB or ASB slave device. AMBA APB is optimized for minimum power consumption and reduced interface complexity in order to support peripheral functions. The AMBA APB can be used to interface to any peripherals which are having low bandwidth and they do not require the high performance of a pipelined bus interface. [19]

**1. APB bridge**

The bridge unit converts system bus transfers into APB transfers and performs the following functions:

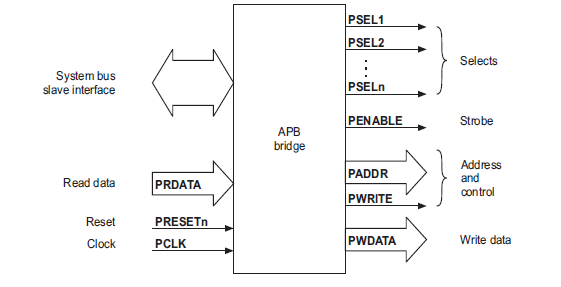
• Latches the address and holds it valid throughout the transfer.

• Decodes the address and generates a peripheral select, **PSELx**. Only one select signal can be active during a transfer.

• Drives the data onto the APB for a write transfer.

• Drives the APB data onto the system bus for a read transfer.

• Generates a timing strobe, **PENABLE**, for the transfer.



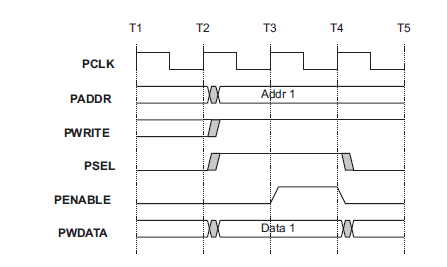
APB bridge

**2. Read and Write Transfer :**

The write transfer starts with the address, write data, write signal and select signal all changing after the rising edge of the clock. The first clock cycle of the transfer is called the SETUP cycle. After the following clock edge the enable signal **PENABLE** is asserted, and this indicates that the ENABLE cycle is taking place. The address, data and control signals all remain valid throughout the ENABLE cycle. The transfer completes at the end of this cycle.

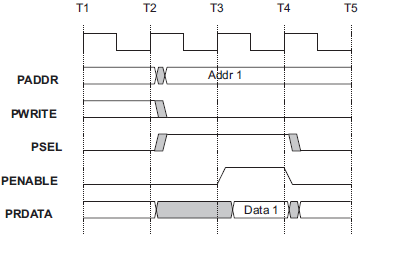
 The enable signal, **PENABLE**, will be de-asserted at the end of the transfer. The select signal will also go LOW, unless the transfer is to be immediately followed by another transfer to the same peripheral.

In order to reduce power consumption the address signal and the write signal will not change after a transfer until the next access occurs.



Write transfer

The timing of the address, write, select and strobe signals are all the same as for the write transfer. In the case of a read, the slave must provide the data during the ENABLE cycle. The data is sampled on the rising edge of clock at the end of the ENABLE cycle.



Read Transfer

**IV.  AMBA AXI**

The AMBA AXI protocol supports high-performance, high-frequency system designs in addition of being suitable for high-bandwidth and low-latency designs with backward-compatible with existing AHB and APB interfaces. [20]

* **The key features of the AXI protocol are:**

• Separate address/control and data phases .

• Support for unaligned data transfers, using byte strobes .

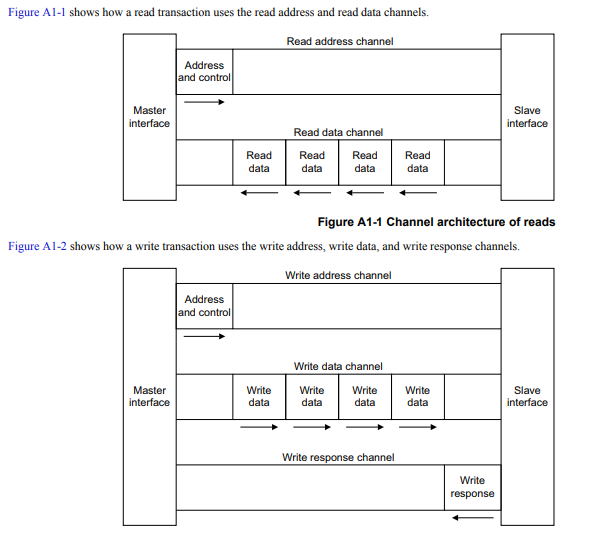
• Separate read and write data channels, that can provide low-cost Direct Memory Access   (DMA), 5 channels each transfer in one direction.

• Support for out-of-order transaction completion .

• The AXI protocol includes the optional extensions that cover signaling for low-power  operation. [20]

The AXI protocol is **burst-based** and defines the following independent transaction channels:

* Read address
* Read data
* Write address
* Write data
* Write response



Read and write Transactions

An address channel carries control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either:

* A write data channel to transfer data from the master to the slave. In a write    transaction, the slave uses the write response channel to signal the completion of the transfer to the master.
* A read data channel to transfer data from the slave to the master. [20]

* **Channel definition**

    Each of the independent channels consists of a set of information signals and **VALID and READY** signals that provide a two-way handshake mechanism. The information source uses the VALID signal to show when valid address, data or control information is available on the channel. The destination uses the READY signal to show when it can accept the information. Both the read data channel and the write data channel also include **a LAST signal** to indicate the transfer of the final data item in a transaction. The appropriate address channel carries all of the required address and control information for a transaction.

All five transaction channels use the same VALID/READY handshake process to transfer address, data, and control information. This two-way flow control mechanism means both the master and slave can control the rate at which the information moves between master and slave. Transfer occurs only when both the VALID and READY signals are HIGH. [20]

**1.** **Read data channel**

The read data channel carries both the read data and the read response information from the slave to the master, and includes:

• The data bus, that can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide

• A read response signal indicating the completion status of the read transaction. [20]

**2. Write data channel**

    The write data channel carries the write data from the master to the slave and includes:

    • the data bus, that can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide

    • a byte lane strobe signal for every eight data bits, indicating which bytes of the   data are valid.

    Write data channel information is always treated as buffered, so that the master can perform write    transactions without slave acknowledgement of previous write transactions. [20]

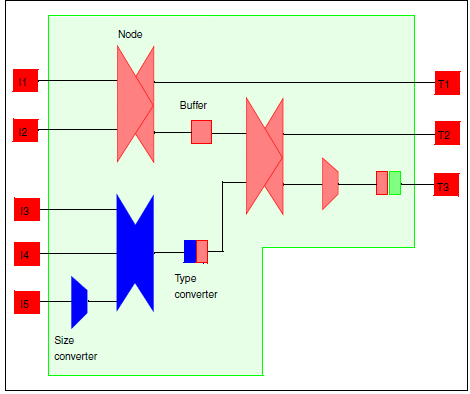
# III.      SYSTEM ON CHIP BUSES; CONSUMER DOMAIN

**A. STBus**is an on-chip bus protocol developed by STMicroelectronics, its main applications are set top boxes, ATM networks and digital still cameras. Three different types of the STBus protocol exist, each having a different level of complexity in terms of both performance and implementation. The arbiter in the STBus is called node while the master and slave are called initiator and target respectively. Type (I) is a simple synchronous handshake protocol with limited set of available command types, no pipelining is applied.

Type (I) acts as a request-grant protocol. Only limited operation code and length are supported.

Type (II) This protocol is more efficient than type I as it supports split transactions and adds pipelining features. The transaction set include read/write operation with different sizes (up to 64 bytes) and also specific operations like read-modify-write and swap.

Type (III) This is the most efficient protocol, as it adds support for split transactions, out-of-order executions, and asymmetric communications. The switch or node is a block that arbitrates requests and responses, different kinds of arbitration are possible, including fixed priorities and variable priorities. STBus can be implemented using different bus topologies as single shared bus, full crossbar and partial crossbar.  [8]



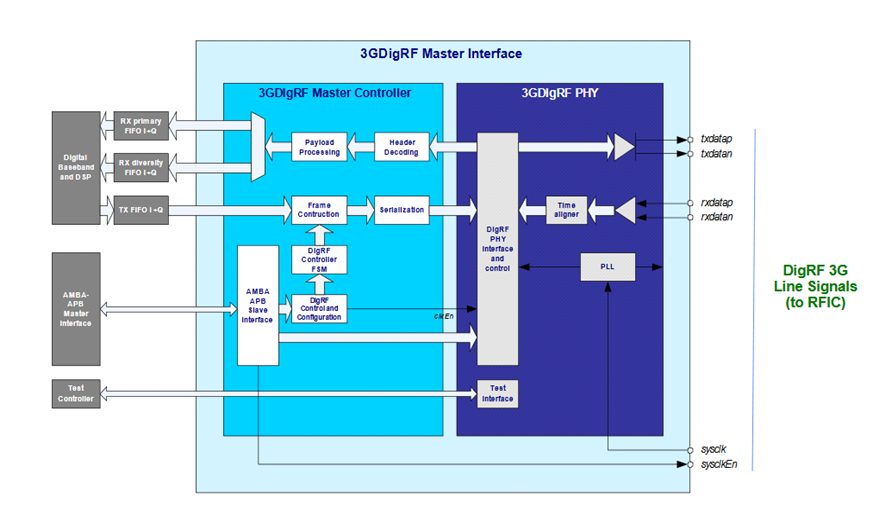
Example of STBus interconnect system

**B. MIPI**stands for Mobile Industry Processor Interface developed by MIPI Alliance. MIPI is the main    interface for connecting IP blocks in mobile phone, A broad portfolio of interface specifications from the MIPI Alliance enables design engineers to efficiently interconnect essential components in a mobile device, from the modem and antenna to the peripherals and application processor. MIPI specifications have enabled manufacturers to simplify the design process, reduce design costs, create economies of scale that lower price points, and shorten time-to-market for components, features, and services. Fundamentally, every MIPI specification addresses the industry’s needs for three key characteristics that are essential for any successful mobile design: low power consumption, high-performance operations, and low electromagnetic interference (EMI). MIPI currently has a pair of high-speed physical-layer (PHY) specifications, M-PHY and D-PHY, to support a full range of application requirements in mobile terminals[4]. Application area for MIPI includes many domains, In this paper we focused on Chip-to-Chip Inter Process Communications which includes two main protocols LLI and DigRF.



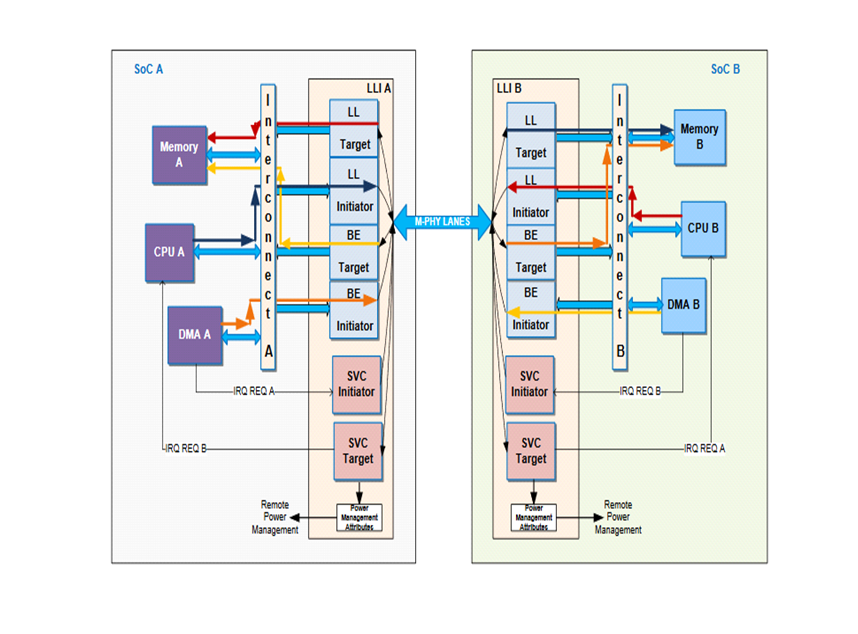
MIPI different domians.

**I. DigRF**is a high-speed digital interface standard defined and supported by the Mobile Industry Processor Interface (MIPI) Alliance. It is used primarily in between the RF transceiver IC and the baseband (BB) IC in a mobile handset. It provides a standard way to transmit I/Q data between the RF transceiver and the BB chip. The receiver circuits generate the I and Q signals from the radio signal. These are digitized and otherwise processed and sent to the BB circuits for further processing. In the transmit mode, the BB chip creates the digital versions of the I and Q signals and sends them to the RF chip over the interface. The RF chip performs digital-to-analog conversion to create the analog signals that become the radio signal to be transmitted. In the past, a variety of RF and BB chips generated both digital and analog I/Q signals. There was no standard approach, meaning designers had to add extra circuits to make conversions as required to allow the two chips to talk to one another. The DigRF interface standardizes digital transfers. The digital interface is basically a serial data bus with six paths—one for receive (Rx) I/Q data, one for transmit (Tx) data, one for a clock signal, and another for a clock enable. The Rx and Tx paths are differential low-voltage differential signalling (LVDS). In the new version 4, the data rate is 1.5 Gbits/s. That speed is needed to support the newer Long-Term Evolution and WiMAX standards that also use multiple-input multiple-output (MIMO). The older version 3 transmitted at a 312-Mbit/s rate. MIPI defines a packet-based protocol that transmits the data as well as control information. Control packets send control signals between the RF and BB chips. Data packets send data in alternate I and Q words using 8B/10B FEC encoding. When MIMO is used, multiple I/Q words must be sent interleaved. The protocol overhead is typically 20% of the serial data. It is designed to provide a convenient approach for integrating components and meeting the data-intensive needs of 4G LTE air interfaces that require high channel bandwidth. It is a low-complexity solution for complex implementations that typically require multi-mode, multi-band operation. It natively handles MIMO configurations, receive diversity and carrier aggregation. In addition to LTE, it supports HSPA+, 3.5G and 2.5G air interfaces. MIPI DigRF is a high-level interface that operates on the MIPI M-PHY physical layer, enabling a single link between the baseband and RFIC[6].



DigRF Architecture.

**II.** **LII** stands for MIPI Low Latency interface, it is a point-to-point interconnection that allows two devices on separate chips to communicate as if a device attached to the remote chip is resident on the local chip. Devices on this bidirectional connection use memory mapped transactions for communications via native protocols such as Open Core Protocol and Advanced Microcontroller Bus Architecture (AMBA®), using the MIPI M-PHY® physical layer. As its primary use case, LLI targets concurrent low latency and high bandwidth chip-to-chip traffic over a low pin count interface, using any of three functionally oriented traffic classes: Service, Best Effort and Low Latency. Each provides an efficient mechanism for optimal data routing between the variety of devices supported by LLI[7].



LLI Architecture.

* **LLI Befnfits:**

•       Efficient System Partitioning

•       Very low latency (using 12 symbol transmissions

•       Supports 3 traffic classes for simultaneous low latency and high bandwidth communication:

•       Service (SVC)

•       Low Latency (LL)

•       Best Effort (BE)

•       Minimal or no software housekeeping

•       In-band signalling minimizes extra inter-chip General Purpose Input Outputs (GPIOs) and pin count

•       Asymmetric M-PHY lanes implementation support reducing M-PHY link pins count

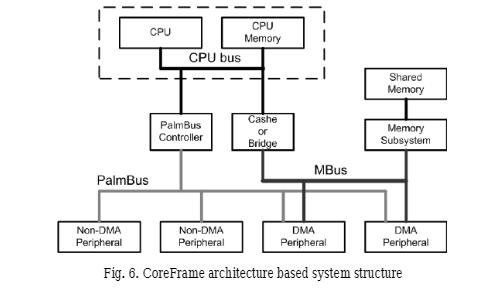
•       HS- Gear 3 Support for MIPI M-PHY Physical Layer.

**C. Core Frame Protocol**this protocol is developed by PalmBeach corporation. The Core-Frame architecture is low power high-performance on-chip interconnect architecture for integration of SOC blocks.It consists of  three independent parallel buses (CPU bus, Palm-Bus and M-Bus)  rather than a hierarchy of buses which allows concurrent activities on both buses maximizing available bandwidth.

**I. Palm-Bus** represents a master-slave interface with a single-master intended for Communications between the CPU and peripheral blocks. It is not used to access memories. it is also designed with low-power consumption in mind.

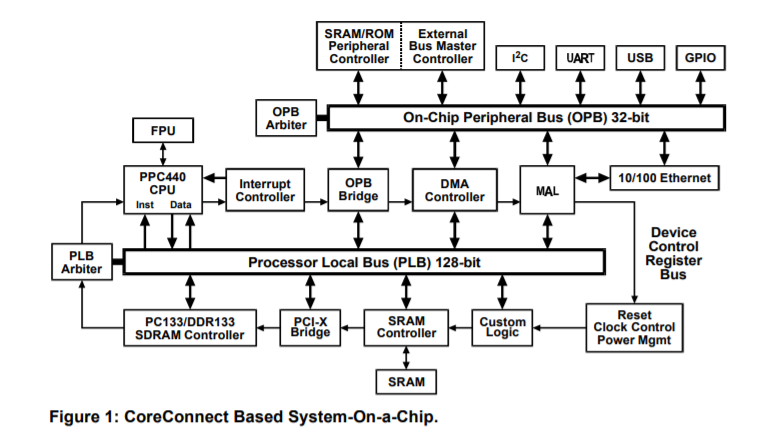
**II. M-Bus**is designed for high-speed accesses to shared memory from the CPU core and peripheral blocks.

The most distinctive feature of the core-frame is the separation of the I/O and and memory transfers on different buses. The Palm-bus provides for the I/O back plane and allows the processor to configure and control while the M-bus provides a DMA connection from peripherals to main memory, allowing direct data transfer without processor intervention.



Core-Frame Architecture

**D. Core-Connect Protocol**developed by IBM Blue Logic™,  provides three buses for interconnecting cores, library macros, and custom logic .(Processor Local Bus (PLB) ,On-Chip Peripheral Bus (OPB) and Device Control Register (DCR)).



Core-Connect Architecture

**I. Processor Local Bus (PLB) :**

**-**The PLB addresses the high performance, low latency and design flexibility issues needed in a highly integrated SOC through decoupled address, read data, and write data buses allowing concurrent read and write transfers thus maximising bus utilization also address pipelining that reduces bus latency by overlapping a new write request with an ongoing write transfer. PLB bus supports up to 16 master and any number of slaves.

- The PLB macro consists of a bus arbitration control unit and the control logic required to manage the address and data flow through the PLB. The separate address and data buses from the masters allow simultaneous transfer requests. The PLB macro arbitrates among these requests and directs the address, data and control signals from the granted master to the slave bus

   - PLB bus transaction is grouped under an address cycle and a data cycle:

**1-Address cycle:** Master drives address and transfer qualifier signals (Request)  Arbiter grants the bus ownership, presents the signals to the slave (Transfer) Slave latches the address and transfer qualifiers (Address-Acknowledge)

**2-Data cycle:** for each data beat,  Master drives the write data bus for a write transfer or sample the read data bus for a read transfer (Transfer) , Data acknowledge signals are required after the beat.

**II. On-Chip Peripheral Bus(OPB):**OPB is a secondary bus architected to reduce capacitive loading on the PLB, suitable for attachment of low-bandwidth devics .  PLB masters gain access to the peripherals on the OPB bus through the OPB bridge. The bridge acts as a slave device on the PLB and a master on the OPB.  The OPB supports multiple masters and slaves by implementing the address and data buses as a distributed multiplexer.

**III. Device Control Register Bus :**Lower performance status and configuration registers are typically read and written through the Device Control Register (DCR) Bus. The DCR provides a maximum throughput of one read or write transfer every two cycles.  Fully synchronous bus typically implemented as a distributed multiplexer.

# IV.      SYSTEM ON CHIP BUSES; AUTOMOTIVE DOMAIN

In this section, We propose the main features of  SoC buses provided in the industry. Given that the number of electronic components is increasing exponentially, point-to-point communication is not possible. The main reasons is the large number of wires needed to connect all the components, non-availability of space and in case of a failure the fault detection will be extremely difficult. As a result, serial buses as *LIN, CAN , FLEXRAY* for controlling and *MOST* for infotainment are discussed below.

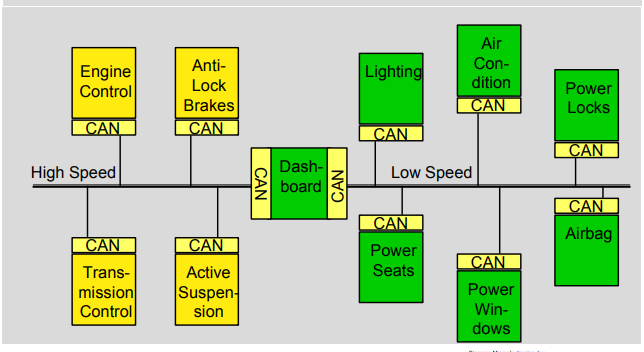
**A. Controller Area Network (CAN)***Controller Area Network* is an advanced serial bus system that efficiently supports distributed control systems. It was initially developed for the use in motor vehicles by Robert Bosch GmbH, Germany, in the late 1980s.

Bosch published several versions of the CAN specification and the latest is CAN 2.0 published in 1991. This specification has two parts; part A is for the standard format with an 11-bit identifier, and part B is for the extended format with a 29-bit identifier. A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A and a CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.

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.

 For that purpose, It originally supports medium speed data rates up to 125 kbps and high speed data rates up to 1Mbps.

A main feature to mention is that *CAN*communicates asynchronously with an event driven protocol interface along with error handling using CRC where all frame types (data, remote, error and overload frame) are transmitted in broadcast. The *CAN* protocol handles bus accesses according to the concept called “Carrier Sense Multiple Access with Arbitration on Message Priority”. Any node has the right to request transmission rights at any time. The necessary bus arbitration method to avoid transmission conflicts is the same: Frame with the highest assigned identifier get bus access without delay.  [10][11]



General CAN Architecture

CAN is a multi-master bus with an open, linear structure with one logic bus line and equal nodes. Instead of point to point wiring, one serial bus is used to connect all the control systems. The number of nodes is not limited by the protocol.

  In the CAN protocol, the bus nodes do not have a specific address. Instead, the address information is contained in the identifiers of the transmitted messages, indicating the message content and the priority of the message. The CAN protocol uses Non-Return-to-Zero or NRZ bit coding and for synchronization; bit stuffing is used.

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 are two bus states, called “dominant (0)” and “recessive (1)”. [10]

**I.  Collision Handling and arbitration**

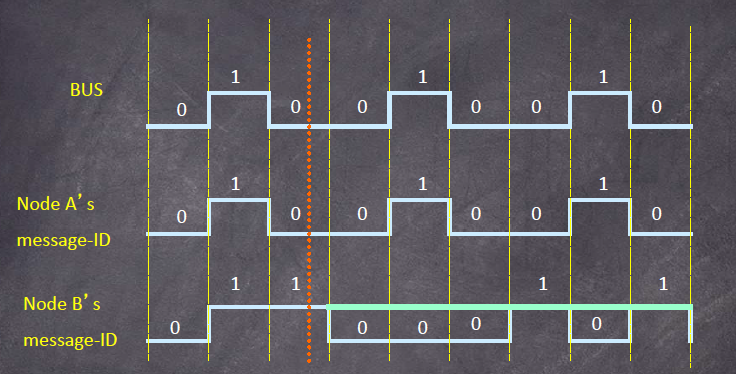
  The CAN protocol handles bus accesses according to the concept called “Carrier Sense Multiple Access with Arbitration on Message Priority”. This arbitration concept avoids collisions of messages whose transmission was started by more than one node simultaneously and makes sure the most important message is sent first without time loss.

Any node has the right to request transmission rights at any time. The necessary bus arbitration method to avoid transmission conflicts is the same: Frame with the highest assigned identifier get buss access without delay. All frame types (data, remote, error and overload frame) are transmitted in broadcast. The data frame structure comprising several field is the same.

1)      As a node transmits each bit, it verifies that it sees the same bit value on the bus that it transmitted.

2)      A “0” on the bus wins over a “1” on the bus.

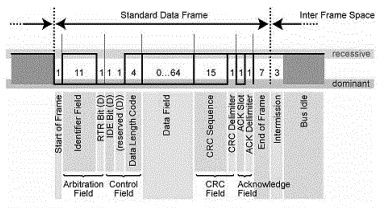
3)      Losing node stops transmitting, winner continues. [10] [11]



Arbitration Process

**II. Data frame**

 A “Data Frame” is generated by a CAN node when the node wishes to transmit data. The frame begins with a dominant Start of Frame bit for hard synchronization of all nodes.



Data Frame

* The start of frame bit marks the beginning of the data frame, it consists of a single dominant bit.

* The Arbitration Field consisting of 12 bits: The 11-bit Identifier, which reflects the contents and priority of the message, and the Remote Transmission Request bit. The Remote transmission request bit is used to distinguish a Data Frame (RTR = dominant) from a Remote Frame (RTR = recessive).

* The next field is the Control Field, consisting of 6 bits. The first bit of this field is called the IDE bit (Identifier Extension) and is at dominant state to specify that the frame is a Standard data Frame not an interframe space. The following bit is reserved and defined as a dominant bit. The remaining 4 bits of the Control Field are the Data Length Code (DLC) and specify the number of bytes of data contained in the message (0 - 8 bytes).

* Data field: The data being sent follows in the Data Field which is of the length defined by the DLC above (0, 8, 16, …., 56 or 64 bits).

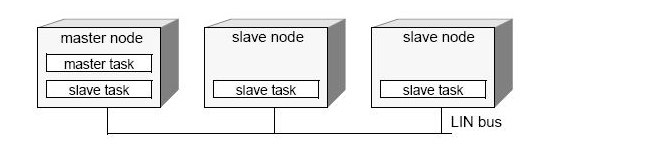
* The Cyclic Redundancy Field (CRC field) follows and is used to detect possible transmission errors. The CRC Field consists of a 15 bit CRC sequence, completed by the recessive CRC Delimiter bit.

* The next field is the Acknowledge Field. During the ACK Slot bit the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit (regardless of whether the node is configured to accept that specific message or not).

* Finally, the seven recessive bits that determine the end of frame. [10] [11]

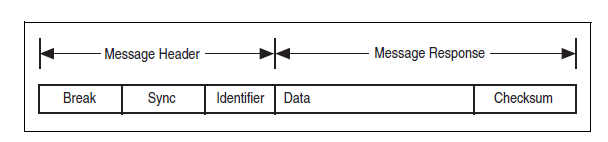
**B. Local Interconnect Network (LIN):***LIN* is an asynchronous serial broadcast bus where all messages are initiated by the one single master with at most one slave replying to a given message identifier. The  *Local Interconnect Network* (LIN) bus was developed to create a standard for low-cost, low-end multiplexed communication in automotive networks. LIN provides cost-efficient communication in applications where the bandwidth and versatility of CAN are not required.

*LIN* is a single wire communication bus with data rates up to 20 Kbps which is suitable for modules where response time is not a critical issue as mirror controls and window controls. The *LIN* bus is a polled bus with a single master device and one or more slave devices. The master device contains both a master task and a slave task. Each slave device contains only a slave task.



Master - Slave nodes

Communication over the *LIN* bus is controlled entirely by the master task in the master device. The basic unit of transfer on the *LIN*bus is the frame, which is divided into a  header and a response. The header is always transmitted by the master node and consists of three distinct fields: the break, synchronization (sync), and identifier (ID). The response, which is transmitted by a slave task and can reside in either the  master node or a slave node, consists of a data payload and a checksum.[12][13]



LIN Message Frame

**1. Break**

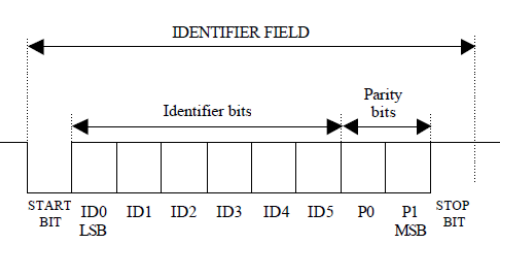
 Every LIN frame begins with the break, which comprises 13 dominant bits (nominal) followed by a break delimiter of one bit (nominal) recessive. This serves as a start-of-frame notice to all nodes on the bus.

**2. Sync**

The sync field is the second field transmitted by the master task in the header. The sync field allows slave devices that perform automatic baud rate detection to measure the period of the baud rate and adjust their internal baud rates to synchronize with the bus.

**3. ID**

The ID field is the final field transmitted by the master task in the header. This field provides identification for each message on the network and ultimately determines which nodes in the network receive or respond to each transmission. All slave tasks continually listen for ID fields, verify their parities, and determine if they are publishers or subscribers for this particular identifier. The LIN bus provides a total of 64 IDs. IDs 0 to 59 are used for signal-carrying (data) frames, 60 and 61 are used to carry diagnostic data, 62 is reserved for user-defined extensions, and 63 is reserved for future protocol enhancements. The ID is transmitted over the bus as one protected ID byte, with the lower six bits containing the raw ID and the upper two bits containing the parity.



ID Field

**4. Data Bytes**

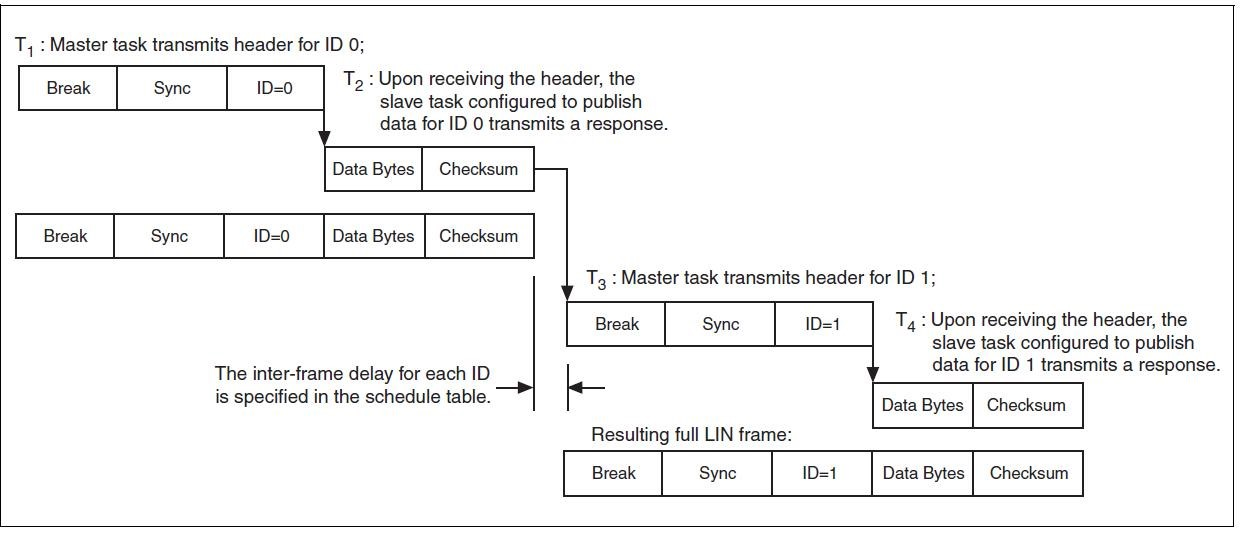
The data bytes field is transmitted by the slave task in the response. This field contains from one to eight bytes of payload data bytes.

**5. Checksum**

The checksum field is transmitted by the slave task in the response. The LIN bus defines the use of one of two checksum algorithms to calculate the value in the eight-bit checksum field. Classic checksum is calculated by summing the data bytes alone, and enhanced checksum is calculated by summing the data bytes and the protected ID.

The LIN 2.0 specification defines the checksum calculation process as the summing of all values and subtraction of 255 every time the sum is greater than or equal to 256 .

“Figure 28” illustrates how a master task header and a slave task response combine to create a LIN full frame. [12]



Creation of LIN Frames

 Two bus states — Sleep-mode and active — are used within the LIN protocol. While data is on the bus, all LIN-nodes are requested to be in active state. After a specified timeout, the nodes enter Sleep mode and will be released back to active state by a WAKEUP frame. This frame may be sent by any node requesting activity on the bus, either the LIN Master following its internal schedule, or one of the attached LIN Slaves being activated by its internal software application. After all nodes are awakened, the Master continues to schedule the next Identifier. [12][13]

**6. Schedule table**

  One of key properties of LIN protocol is **Schedule Tables**. It’s applied for managing the timing of frame slots and traffic control on the bus. The application of Schedule Tables in LIN protocol guarantees that LIN bus won’t be overloaded.

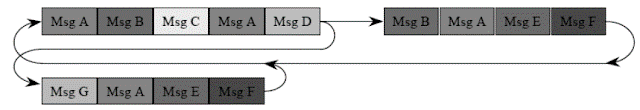
For a specifically implemented LIN system, it’s a constant. It’s used to calculate the time length of a frame slot according to the formula **Tframe\_slot = Tbase \* n**, in which **n** is different for each frame slot. It depends on the length of a frame slot.

A schedule table can incorporate more than one frame slot as shown in “Figure 29”. If only one schedule table is present in a system. When it reaches the end of the schedule table, the schedule will go back to the start of the schedule table and continue performing along the sequence in the schedule table.



Master uses a schedule table

When a schedule table has been performed, any one of the rest of schedule tables may be performed. It depends on system policies.



Master uses different schedule tables

**C. FlexRay** networking standard for motor vehicles serves as the next step beyond CAN and LIN, enabling the reliable management of many more safety and comfort features. FlexRay suits X-by-Wire applications, for example:

•  Steering-by-Wire—Typically using electronic control unit .

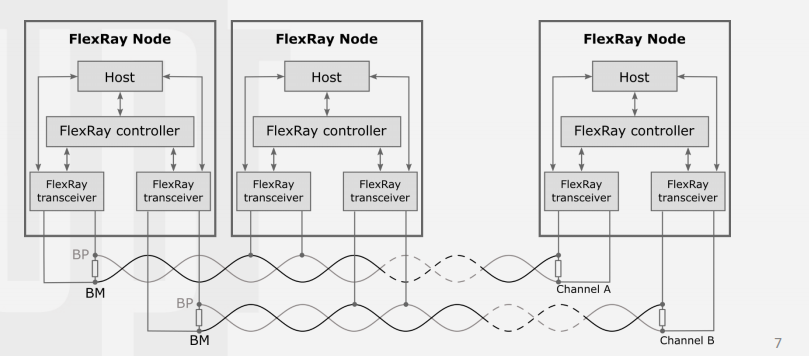
•  Anti-lock brake system (ABS)—Including vehicle stability control (VSC) and vehicle stability assist (VSA) .

•  Power train—Controlling an electronic throttle that replaces the current mechanical system.

  FlexRay focuses on a set of core needs for today’s automotive industry, including higher data rates than previous standards, flexible data communications, versatile topology options, and fault-tolerant operation. FlexRay thus delivers the speed and reliability required for in-car control systems. The CAN network has reached its performance limits with a maximum speed of 1 Mbps. **With a maximum data rate of 10 Mbps available on two channels**, giving a gross data rate of up to 20Mbit/sec, FlexRay potentially offers **20 times higher** net bandwidth than CAN when used in the same application.

FlexRay also offers many reliability features not available in CAN as flexible configurations, with support for topologies such as bus, star, and hybrid types . Designers can configure distributed systems by combining two or more of these topologies.

To meet diverse communication requirements, FlexRay also provides both **static and dynamic communication segments** within each communication cycle. The static communication segment provides bounded latency and the dynamic segment helps meet varying bandwidth requirements that can emerge at system run time. The fixed-length static segment of a FlexRay frame transfers messages with a fixed-time-trigger method determined by the longest FlexRay message or Largest transmission delay (up to 2,5 µs) . In addition to operating as a single-channel system like CAN and LIN, FlexRay can operate as a dual-channel system. The dual-channel option makes data available via a redundant network—a vital capability for a high-reliability system. [14][18]



FlexRay Nodes Architecture

Moreover, FlexRay allows both **synchronous** (real-time) and **asynchronous** data transfer to meet the demand for various systems in vehicles.

**FlexRay uses two methods for granting bus access to nodes**

**• TDMA (Time Division Multiple Access)**

The TDMA method uses a communication schedule which is split into time slots. Each FlexRay node has one or more slots assigned in which it is granted access to the bus .The communication schedule is repeated periodically by all nodes . [14]

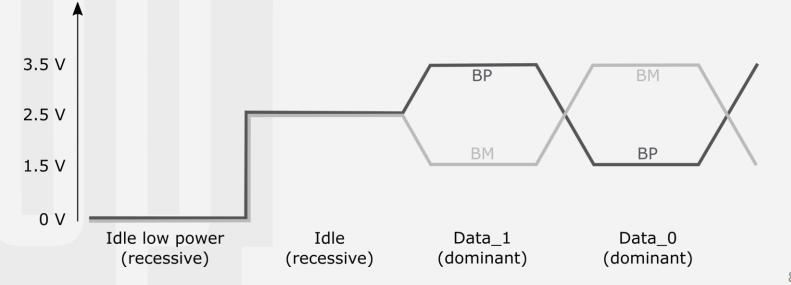
**• FTDMA (Flexible Time Division Multiple Access)**

The FTDMA method is used to implement a dynamic segment which reserves a specific slot in the communication cycle that can be used by any node to transmit messages in a non-deterministic manner.

FlexRay uses differential signal transmission and NRZ encoding with Two communication lines: Bus Plus (BP) and Bus Minus (BM). The physical layer defines four bus levels :

 • Dominant – differential voltage not equal to 0 V (Data\_1, Data\_0).

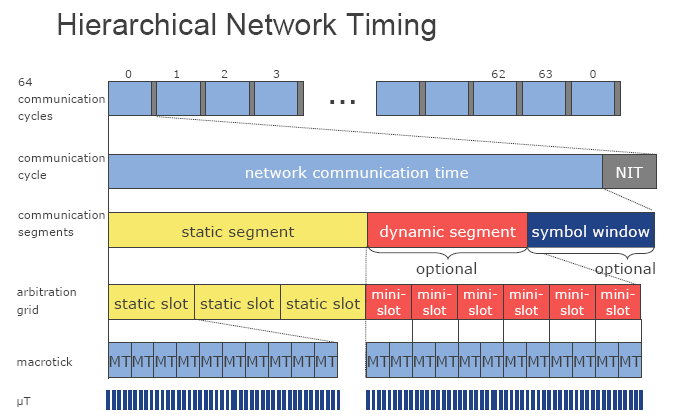
 • Recessive – differential voltage of 0V (Idle & low power).



Recessive and Dominant voltages

**I. Flexray Frame**

   Most embedded networks have a small number of high-speed messages and a large number of lower-speed, less-critical networks.  To accommodate a wide variety of data without slowing down the FlexRay cycle with an excessive number of static slots, the dynamic segment allows occasionally transmitted data.  The segment is a fixed length, so there is a limit of the fixed amount of data that can be placed in the dynamic segment per cycle. To prioritize the data, **minislots** are pre-assigned to each frame of data that is eligible for transmission in the dynamic segment.  A minislot is typically a **macrotick** (a microsecond) long. Higher priority data receives a minislot closer to the beginning of the dynamic frame. [14][18]



FlexRay Communication Cycle (I)

Each slot of a static or dynamic segment contains a FlexRay Frame. The frame is divided into three segments: Header, Payload, and Trailer. The Header is 5 bytes (40 bits) long and includes the following fields:

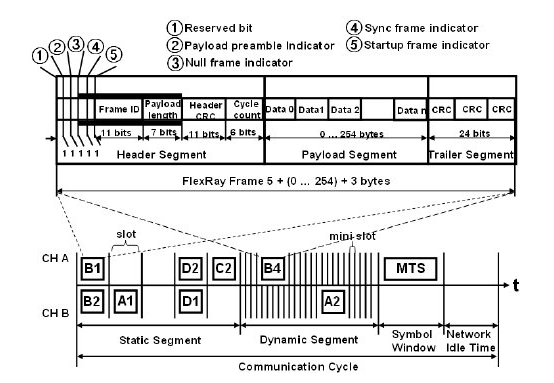
1.   Status Bits - 5 bits

2.   Frame ID - 11 bits

3.   Payload Length - 7 bits

4.   Header CRC - 11 bits

5.   Cycle Count - 6 bits



FlexRay Communication Cycle (II)

The Frame ID defines the slot in which the frame should be transmitted and is used for prioritizing event-triggered frames. The Payload Length contains the number of words which are transferred in the frame. The Header CRC is used to detect errors during the transfer. The Cycle Count contains the value of a counter that advances incrementally each time a Communication Cycle starts. The payload contains the actual data transferred by the frame. The length of the FlexRay payload or data frame is up to 127 words (254 bytes), which is over 30 times greater compared to CAN. [14][18]

* **Status Bits**

1. **Reserved bit (1 bit)**

The reserved bit is reserved for future protocol use. It shall not be used by the application.

• A transmitting node shall set the reserved bit to logical ‘0’.

• A receiving node shall ignore the reserved bit.

2. **Payload preamble indicator (1 bit)**

 The payload preamble indicator indicates whether or not an optional vector is contained within the payload segment of the frame transmitted:

• If the frame is transmitted in the dynamic segment the payload preamble indicator   indicates the presence of a message ID at the beginning of the payload.

3. **Null frame indicator (1 bit)**

The null frame indicator indicates whether or not the frame is a null frame, i.e. a frame that contains no useable data in the payload segment of the frame.

4. **Sync frame indicator (1 bit)**

The sync frame indicator indicates whether or not the frame is a sync frame, i.e. a frame that is utilized for system wide synchronization of communication (used in static segment only).

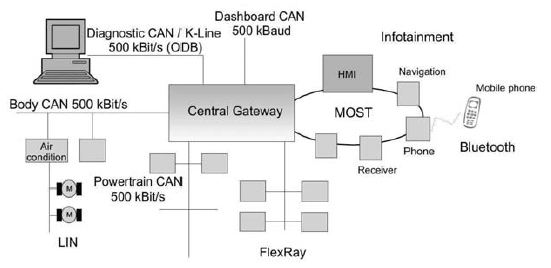
4. **Startup frame indicator (1 bit)**

The startup frame indicator indicates whether or not a frame is a startup frame. Startup frames serve a special role in the startup mechanism. Only ColdStart nodes are allowed to transmit startup frames.

• If the startup frame indicator is set to zero then the frame is not a startup frame.

• If the startup frame indicator is set to one then the frame is a startup frame.

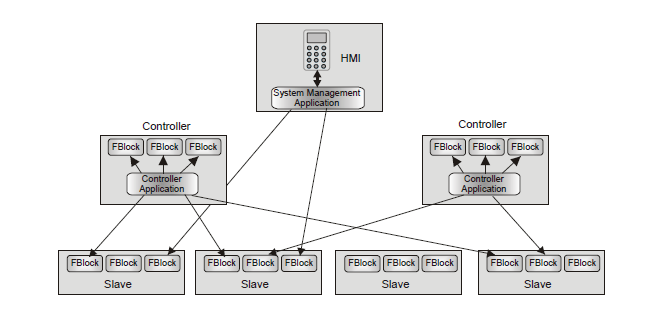
D.   **MOST,** *Media Oriented Systems Transport*, a communication system with a new, flexible architecture, used by many different manufacturers, was developed to meet these demands. It can transmit audio signals synchronously, as in telephones, and is the most widely used multimedia system in cars today. [15]



General Architecture for MOST Interfacing a Gateway

The MOST specification differentiates between three views of the interaction with function blocks :

* HMI
* Controller
* Slave



Interaction with slave blocks

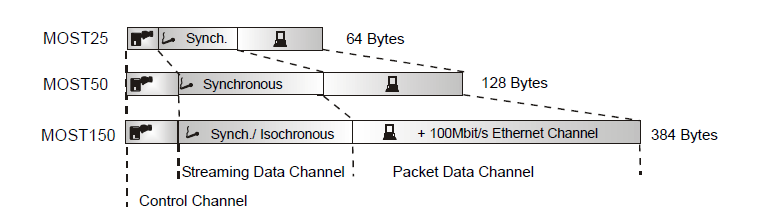
A *slave* is a MOST device, which is controlled by a Controller. It provides its functionalities by means of properties and methods of its function blocks. A slave has no system knowledge at all, i.e., information about other devices is not stored statically and consequently the slave does not control other slaves either. They can easily be added to or removed from a MOST system, without the software being modified or other slaves being influenced. This means that a CD changer or amplifier, can easily be used in different vehicle platforms, if they are implemented as slaves.

   A *Controller* is an application for administration of a functional part of a MOST system, i.e., it controls the function blocks of one or more slaves . A tuner, for example, can control its corresponding amplifier. For this purpose, the Controller requires partial system knowledge, which means that it must know the function blocks to be controlled. The Controller uses an application protocol to control a function block. It is transmitted via the Control Channel .The device address of the addressed function block does not need to be known, as it is ascertained by the Network Service.

  The *Human Machine Interface (HMI)* is the interface to the user of the MOST system and thus presents the system function on a high abstraction level. It coordinates the various Controllers. [16]

### I. Most Frame

  The synchronous transmission of multimedia data, is directly reflected in the frame structure. A frame contains one channel for the synchronous  transmission of streaming data, one channel for the asynchronous transmission of packet data, and one channel for the transmission of control data. In the Streaming Data Channel, static connections between a streaming source and one or several streaming sinks can be built up with the sampling rates 44.1 kHz (MOST25 and MOST50) and 48 kHz (MOST150). The control of the connection set-up and disconnection, and the exchange of the control messages for the function blocks are effected via the Control Channel. The data for the commands transmitted via the Control Channel are distributed over several subsequent frames. Packet data is transmitted in the Packet Data Channel without influencing the synchronous data transmission at all.  A MOST25 frame contains 64 bytes in total. MOST50 can transport 128 byte at the same time due to the double bandwidth, and MOST150 carries 384 bytes.

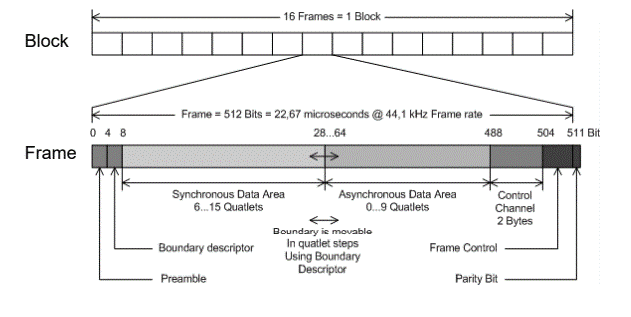


MOST Frames

Since the MOST System transmits the audio data synchronously, additional data buffering is not needed. This reduces the complexity of the device and thereby saves costs. In addition, The most typical data type is synchronous, which represents multimedia data and occupies the largest configurable portion of the frame. Asynchronous data is used to support multi-media information such as GPS systems, information about accessed files, and data of capsulated protocols within the MOST system. [16]

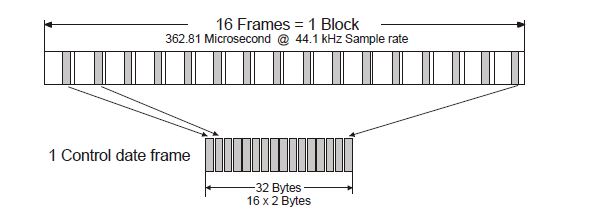
**1. Most 25 Frame**

 A MOST25 frame consists of 512 bits or 64 bytes with sampling rate equals to 44.1kHz. Sixty bytes are used for the transmission of stream and packet data. Two bytes transport one part of the control message that is made up of a total of 32 bytes for the administration of network and nodes such as the transmission of the channel resource allocation table. [15]



MOST 25 Frame

In order to prevent the Control Channel from claiming too much bandwidth per  frame, it is distributed over 16 frames which are combined into one block (see figure 39). Each frame transports 2 bytes of the channel. The preamble of the first frame of a block has a specific bit pattern to identify the block. [15]



One Control Frame

* **Preamble**

   The preamble of the MOST25 frame is used for synchronizing the TimingSlaves to the bit stream and for the initial identification of the frame. The slaves use a PLL switch to synchronize to the network. The TimingMaster generates the preamble based on its oscillator frequency. After the bit stream has been carried over all MOST nodes, it arrives at the master with a phase shift caused by the signal propagation delay in each MOST node. The master thus synchronizes itself to the oncoming frame by means of its PLL, recovers all the bits, regenerates the frame and thus compensates the phase shift. [15]

* **Boundary Descriptor**

   The Streaming Data Channel and the Packet Data Channel share a total of 60 bytes, which are available in a frame. The bandwidths of the two channels can be adapted to their corresponding requirements via the Boundary Descriptor. The boundary between the two areas can be shifted in steps of 4 bytes (a quadlet). The Streaming Channel can thus have a width between 24 and 60 bytes (6 to 15 quadlets) and the Packet Data Channel a width between 0 and 36 bytes (0 to 9

quadlets). The value of the Boundary Descriptor ranges from 6 to 15 quadlets. It is administered by the MOST Network Interface Controller of the TimingMaster. If the value of the Boundary Descriptor is changed by the TimingMaster, all synchronous connections have to be re-established as audio connection in the SIM Access Profile (SAP) . It is used exclusively for exchanging the SIM data between the cell phone and the car. By means of the SIM Access Profile, the telephone integrated in the car can read out the SIM data from the cell phone and use them as long as it is registered via Bluetooth.

* **Frame Control**

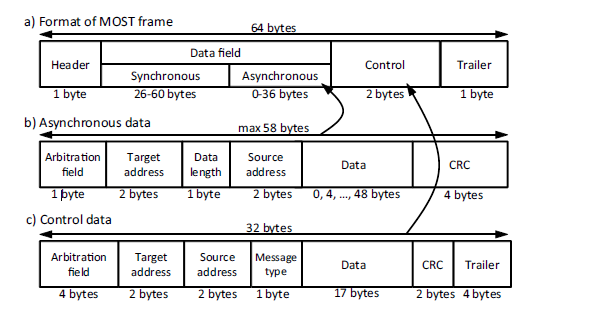
The last byte of the frame is used for controlling of the frame.

* **Parity Bit**

The parity bit enables the detection of bit errors in the frame.

* **Packet data protocol for MOST25 and MOST50**

  The protocol header consists of the arbitration field (1 byte), in which the token is stored, the target address (2 bytes), the data field length (1 byte) and the source address (2 bytes). The protocol is secured by a CRC sum (4 bytes), which is automatically generated by the MOST Network Interface Controller. There is no automatic retransmission if there is a CRC sum error. This has to be handled by the higher layers. [15]



Packet Data for MOST25 and MOST50

# IIV.     Acknowledgements

First, we would like to thank everyone for all their help and advices that encouraged us to fulfil such achievements. And special thanks to:

**Dr. Khaled Mohamed**, our supervisor, for helping and guiding us to achieve project milestones.

**Dr. Mohamed Dessouky,** for giving us the opportunity to work in such a great project.

# References

[1] Bennini L., DeMicheli G., Networks on Chips: A New SoC Paradigm, IEEE Computer, Vol. 35, No. 1, January 2002, pp. 70- 78.

[2] Richard Herveille, WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP

Cores, rev. version: B4, 2010. By Open Cores Organization, p.7, 2010. [www.opencores.org.](http://www.opencores.org)

[3] Mohandeep Sharma and Dilip Kumar, WISHBONE Bus ARCHITECTURE – A SURVEY AND COMPARISON,Department of VLSI Design, Center for Development of Advanced Computing, Mohali,india,  at [https://www.researchgate.net/publication/224926942](https://www.researchgate.net/publication/224926942_Wishbone_Bus_Architecture_-_A_Survey_and_Comparison).

[4]Altera Avalon, Avalon bus specification: Reference manual. Altera Corporation, July, 2003.

Available online at <http://www.altera.c>om.

[5] MIPI alliance specifications, [https://www.mipi.org.](https://www.mipi.org/specifications)

[6] Luis Laranjeira, Synopsys ,MIPI DigRF 3G and MIPI DigRF v4 Solutions in Action, Member-to-Member Presentations March 9, 2011 by mipi alliance [https://www.mipi.org.](https://www.mipi.org/sites/default/files/Synopsys-DigRF_3G_and_DigRF_v4_solutions_in_action_Final_v2.pdf)

[7] Low Latency  Interface (LLI) v2.1, Specification Overview, MIPI alliance,  7-Nov-2014,[https://www.mipi.org.](https://www.mipi.org/sites/default/files/LLI_Spec_Brief_2014_FINAL.pdf)

[8]  STBus communication system concepts and definitions -  User manual provided by STMicroelectronics,  STMicroelectronics,  October 2012,

<https://www.st.com/content/ccc/resource/technical/document/user_manual/39/81/fa/c8/2e/4d/41/f5/CD00176920.pdf/files/CD00176920.pdf/jcr:content/translations/en.CD00176920.pdf>

[9]  David A. Deming, The Essential Guide to Serial ATA and SATA Express, 9 October 2014.

[10] Siemens Corp. , CANPres. Version 2.0, Siemens Microelectronics Inc.

[11] Bosch Controller Area Network (CAN) Version 2.0, Protocol Standard , REV3.

[12] “Introduction to the Local Interconnect Network (LIN) Bus” from National Instruments, Aug 24, 2016

[13] “LIN Protocol and Physical Layer Requirements” from Texas Instruments, February 2018.

[14] FlexRay Communications System Protocol Specification Version 3.0.1

[15] A. GRZEMBA, “The AUTOMOTIVE MULTIMEDIA NETWORK from MOST25 to MOST150”, book is based on the MOST Specification Version 3.0., E2, 2010.

 [16] Core Connect Bus specifications, SystemOnChip, by IBM,  International Business Machines Corporation, 1999  <http://www.scarpaz.com/2100-papers/SystemOnChip/ibm_core_connect_whitepaper.pdf?fbclid=IwAR11Mxmb_bZhV9cfnEnecnm1r3H5Vc8LcnDTHLPLM005KpM5uVY_O1ikg6Q>

[17]  Rovin and Sagar, PCI Bus Specifications at  <http://electrofriends.com/articles/computer-science/protocol/introduction-to-pci-protocol/5/?fbclid=IwAR10WOyJCClkPWIm11Ai6JlPiCD-sCm09ipXKa9RBtc2T1_U59XW0O-2ilA>

[18]  Next Generation Car Network –FlexRay ,Fujitsu Microelectronics (Shanghai) Co., Jun 2006

[19] P. Patil , V. Sangamkar, “A Review of System-On-Chip Bus Protocols, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering,” Vol. 4 , Issue 1, Jan. 2015, pp. 2

[20] ARM.AMBA Specifications v2.0, 1999.

[21] AMBA System Technical Reference Manual [online ]

Available: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0138d/ch02s02s03.html>

### 