Section 37. Appendix

HIGHLIGHTS

This section of the manual contains the following topics:

I²C is a trademark of Philips Corporation.

APPENDIX A: I²C™ OVERVIEW

This appendix provides an overview of the Inter-Integrated Circuit (I^2C) bus, with Subsection A.2 “Addressing I^2C Devices” discussing the operation of the SSP modules in I^2Ch mode.

The I^2C bus is a two-wire serial interface. The original specification, or standard mode, is for data transfers of up to 100 Kbps. An enhanced specification, or fast mode (400 Kbps), is supported. Standard and Fast mode devices will operate when attached to the same bus, if the bus operates at the speed of the slower device.

The I^2C interface employs a comprehensive protocol to ensure reliable transmission and reception of data. When transmitting data, one device is the “master”, which initiates transfer on the bus and generates the clock signals to permit that transfer, while the other device(s) acts as the “slave.” All portions of the slave protocol are implemented in the SSP module’s hardware, except general call support, while portions of the master protocol need to be addressed in the PIC16CXX software. The MSSP module supports the full implementation of the I^2C master protocol, the general call address and data transfers up to 1 Mbps. The 1 Mbps data transfers are supported by some of Microchip’s Serial EEPROMs. Table A-1 defines some of the I^2C bus terminology.

In the I^2C interface protocol, each device has an address. When a master wishes to initiate a data transfer, it first transmits the address of the device that it wishes to “talk” to. All devices “listen” to see if this is their address. Within this address, a bit specifies if the master wishes to read-from/write-to the slave device. The master and slave are always in opposite modes (transmitter/receiver) of operation during a data transfer. That is, they can be thought of as operating in either of these two relations:

- Master-transmitter and Slave-receiver
- Slave-transmitter and Master-receiver

In both cases, the master generates the clock signal.

The output stages of the clock (SCL) and data (SDA) lines must have an open-drain or open-collector to perform the wired-AND function of the bus. External pull-up resistors ensure a high level when no device is pulling the line down. The number of devices that may be attached to the I^2C bus is limited only by the maximum bus loading specification of 400 pF and addressing capability.

A.1 Initiating and Terminating Data Transfer

During times of no data transfer (idle time), both the SCL and the data line SDA are pulled high through the external pull-up resistors. The Start and Stop conditions determine the start and stop of data transmission. The Start condition is defined as a high-to-low transition of the SDA when the SCL is high. The Stop condition is defined as a low-to-high transition of the SDA when the SCL is high. Figure A-1 shows the Start and Stop conditions. The master generates these conditions for starting and terminating data transfer. Due to the definition of the Start and Stop conditions, when data is being transmitted, the SDA line can only change state when the SCL line is low.

Figure A-1: Start and Stop Conditions

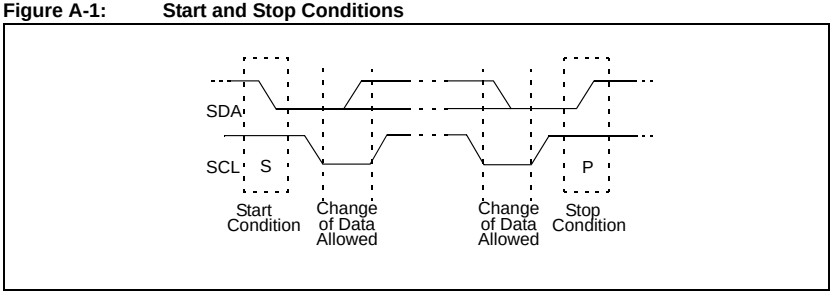


Table A-1: I^2C™ Bus Terminology

Term	Description
Transmitter	The device that sends the data to the bus.
Receiver	The device that receives the data from the bus.
Master	The device which initiates the transfer, generates the clock and terminates the transfer.
Slave	The device addressed by a master.
Multi-master	More than one master device in a system. These masters can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure that ensures that only one of the master devices will control the bus. This ensures that the transfer data does not get corrupted.
Synchronization	Procedure where the clock signals of two or more devices are synchronized.

A.2 Addressing I²C Devices

There are two address formats. The simplest is the 7-bit address format with a R/$\overline{W}$ bit (refer to Figure A-2). The more complex is the 10-bit address with a R/$\overline{W}$ bit (refer to Figure A-3). For 10-bit address format, two bytes must be transmitted. The first five bits specify this to be a 10-bit address format. The 1st transmitted byte has 5 bits which specify a 10-bit address, the two MSbs of the address, and the R/$\overline{W}$ bit. The second byte is the remaining 8 bits of the address.

Figure A-2: 7-bit Address Format

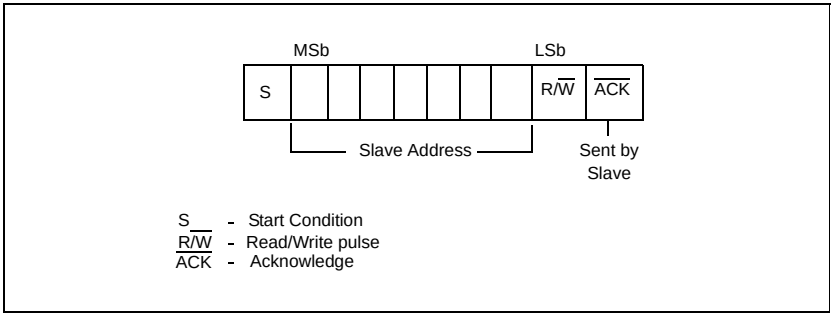
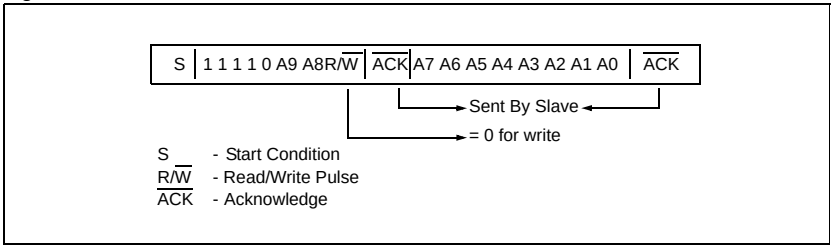


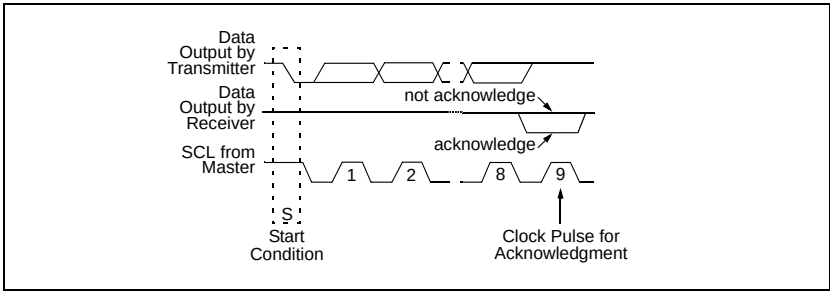
Figure A-3: I²C™ 10-bit Address Format



A.3 Transfer Acknowledge

All data must be transmitted per byte, with no limit to the number of bytes transmitted per data transfer. After each byte, the slave-receiver generates an Acknowledge bit ($\overline{ACK}$) (refer to Figure A-4). When a slave-receiver doesn't acknowledge the slave address or received data, the master must abort the transfer. The slave must leave SDA high so that the master can generate the Stop condition (refer to Figure A-1).

Figure A-4: Slave-Receiver Acknowledge



If the master is receiving the data (master-receiver), it generates an Acknowledge signal for each received byte of data, except for the last byte. To signal the end of data to the slave-transmitter, the master does not generate an acknowledge (not acknowledge). The slave then releases the SDA line so the master can generate the Stop condition. The master can also generate the Stop condition during the Acknowledge pulse for valid termination of data transfer.

If the slave needs to delay the transmission of the next byte, holding the SCL line low forces the master into a wait state. Data transfer continues when the slave releases the SCL line. This allows the slave to move the received data or fetch the data it needs to transfer before allowing the clock to start. This wait state technique can also be implemented at the bit level (refer to Figure A-5).

Figure A-5: Data Transfer Wait State

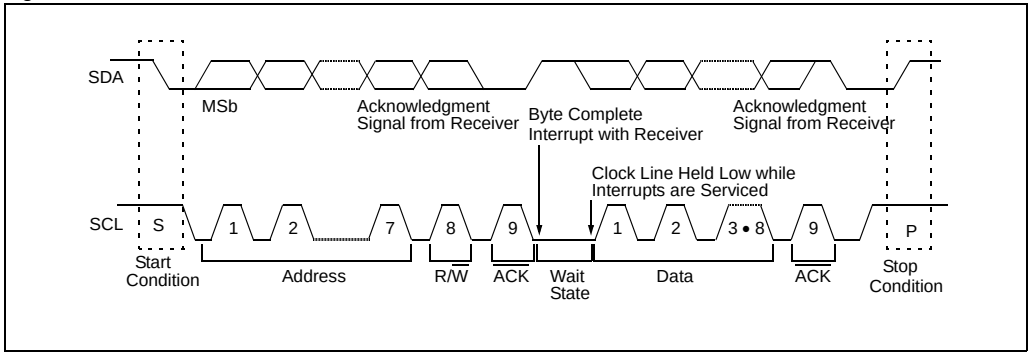


Figure A-6 and Figure A-7 illustrate master-transmitter and master-receiver data transfer sequences.

Figure A-6: Master-Transmitter Sequence

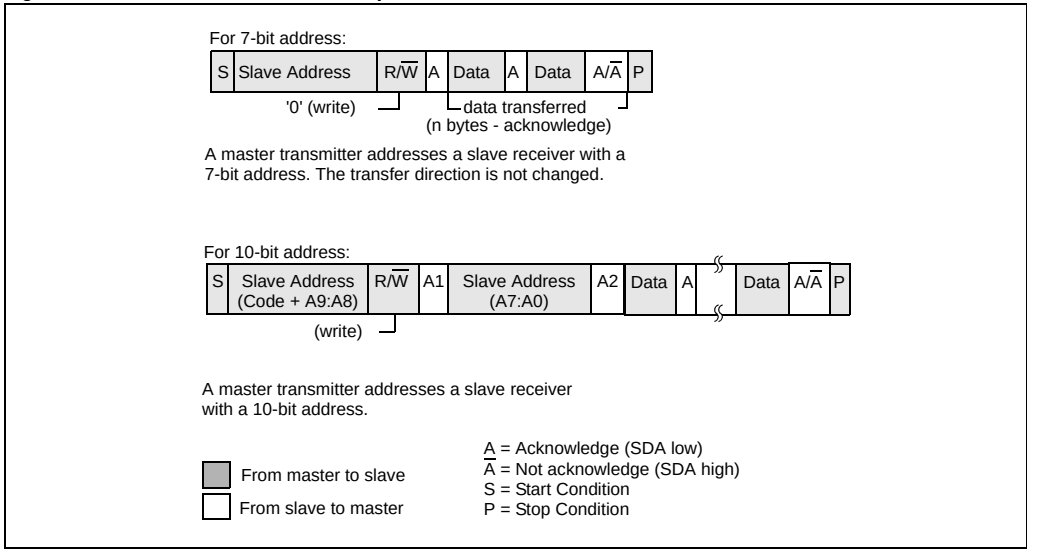
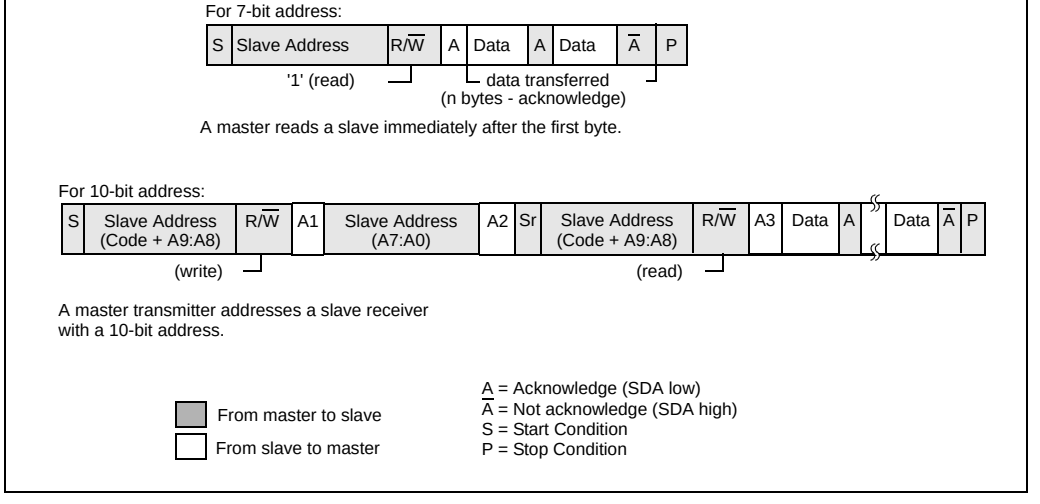
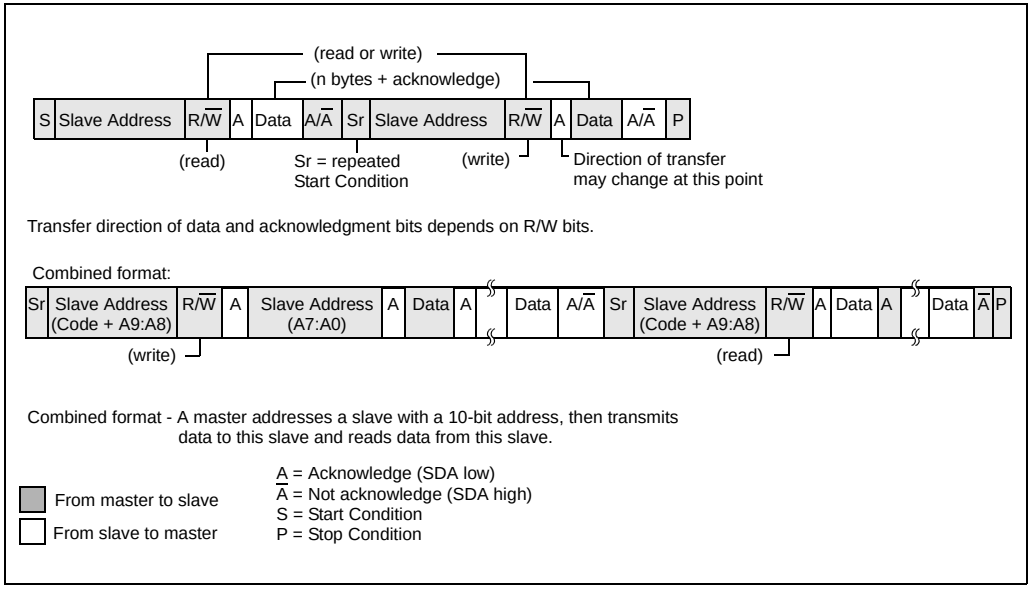


Figure A-7: Master-Receiver Sequence



When a master does not wish to relinquish the bus (which occurs by generating a Stop condition), a repeated Start condition (Sr) must be generated. This condition is identical to the Start condition (SDA goes high-to-low, while SCL is high), but occurs after a data transfer Acknowledge pulse (not the bus-free state). This allows a master to send “commands” to the slave and then receive the requested information or to address a different slave device, as Figure A-8 illustrates.

Figure A-8: Combined Format



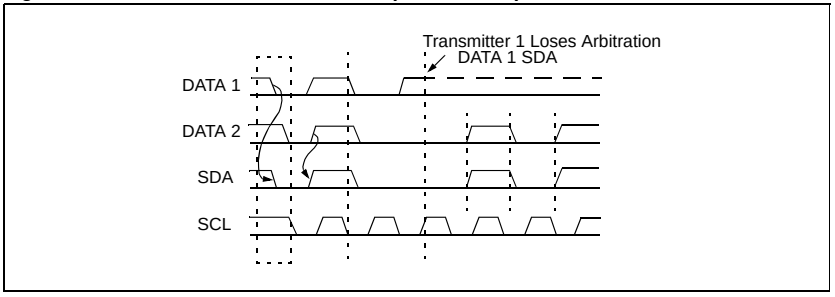
A.4 Multi-master

The I^2C protocol allows a system to have more than one master. This is called a multi-master system. When two or more masters try to transfer data at the same time, arbitration and synchronization occur.

A.4.1 Arbitration

Arbitration takes place on the SDA line, while the SCL line is high. The master which transmits a high when the other master transmits a low, loses arbitration (refer to Figure A-9) and turns off its data output stage. A master which lost arbitration can generate clock pulses until the end of the data byte where it lost arbitration. When the master devices are addressing the same device, arbitration continues into the data.

Figure A-9: Multi-Master Arbitration (Two Masters)



Masters that also incorporate the slave function, and have lost arbitration must immediately switch over to Slave-receiver mode. This is because the winning master-transmitter may be addressing it.

Arbitration is not allowed between:

- A repeated Start condition
- A Stop condition and a data bit
- A repeated Start condition and a Stop condition

Care needs to be taken to ensure that these conditions do not occur.

A.4.2 Clock Synchronization

Clock synchronization occurs after the devices have started arbitration. This is performed using a wired-AND connection to the SCL line. A high-to-low transition on the SCL line causes the concerned devices to start counting off their low period. Once a device clock has gone low, it holds the SCL line low until its SCL high state is reached. The low-to-high transition of this clock may not change the state of the SCL line if another device clock is still within its low period. The SCL line is held low by the device with the longest low period. Devices with shorter low periods enter a high wait-state until the SCL line comes high. When the SCL line comes high, all devices start counting off their high periods. The first device to complete its high period will pull the SCL line low. The SCL line high time is determined by the device with the shortest high period (refer to Figure A-10).

Figure A-10: Clock Synchronization

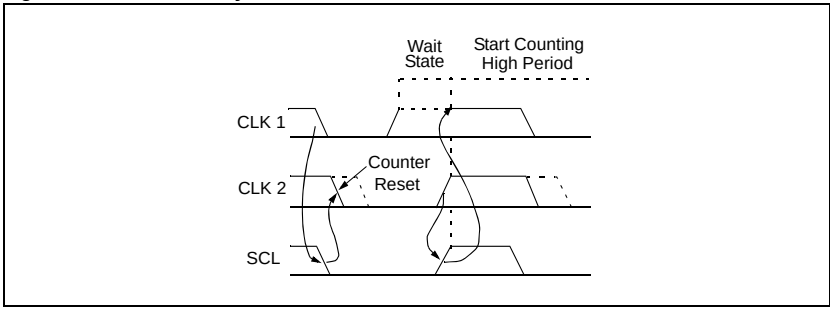


Table A-2 and Table A-3 show the specifications of a compliant I^2C bus. The column titled, Parameter No., is provided to ease the user's correlation to the corresponding parameter in the device data sheet. Figure A-11 and Figure A-12 show these times on the appropriate waveforms.

Figure A-11: I^2C™ Bus Start/Stop Bits Timing Specification

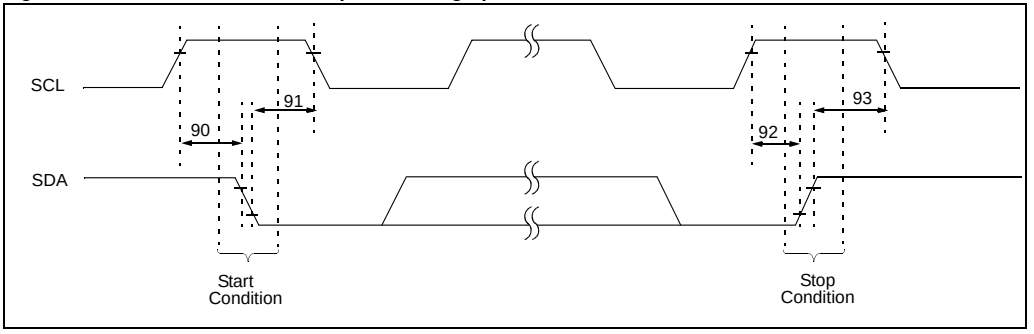


Table A-2: I^2C™ Bus Start/Stop Bits Timing Specification

Parameter No.	Sym	Characteristic		Min	Typ	Max	Units	Conditions
90	TSU:STA	Start condition	100 kHz mode	4700	—	—	ns	Only relevant for repeated Start condition
		Setup time	400 kHz mode	600	—	—		
91	THD:STA	Start condition	100 kHz mode	4000	—	—	ns	After this period the first clock pulse is generated
		Hold time	400 kHz mode	600	—	—		
92	TSU:STO	Stop condition	100 kHz mode	4700	—	—	ns	
		Setup time	400 kHz mode	600	—	—		
93	THD:STO	Stop condition	100 kHz mode	4000	—	—	ns	
		Hold time	400 kHz mode	600	—	—		

Figure A-12: I^2C™ Bus Data Timing Specification

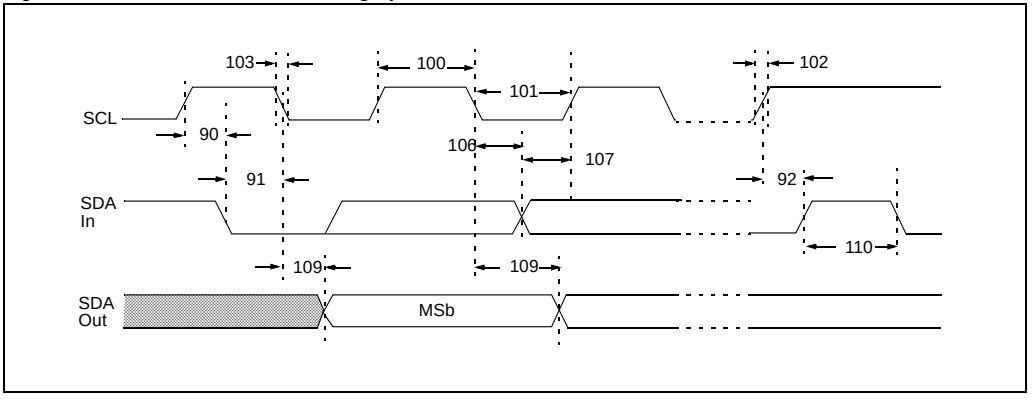


Table A-3: I²C™ Bus Data Timing Specification

Parameter No.	Sym	Characteristic		Min	Max	Units	Conditions
100	THIGH	Clock high time	100 kHz mode	4.0	—	µs	
			400 kHz mode	0.6	—	µs	
101	TLOW	Clock low time	100 kHz mode	4.7	—	µs	
			400 kHz mode	1.3	—	µs	
102	TR	SDA and SCL rise time	100 kHz mode	—	1000	ns	
			400 kHz mode	20 + 0.1Cb	300	ns	Cb is specified to be from 10 to 400 pF
103	TF	SDA and SCL fall time	100 kHz mode	—	300	ns	
			400 kHz mode	20 + 0.1Cb	300	ns	Cb is specified to be from 10 to 400 pF
90	TSU:STA	Start condition setup time	100 kHz mode	4.7	—	µs	Only relevant for repeated Start condition
			400 kHz mode	0.6	—	µs	
91	THD:STA	Start condition hold time	100 kHz mode	4.0	—	µs	After this period the first clock pulse is generated
			400 kHz mode	0.6	—	µs	
106	THD:DAT	Data input hold time	100 kHz mode	0	—	ns	
			400 kHz mode	0	0.9	µs	
107	TSU:DAT	Data input setup time	100 kHz mode	250	—	ns	**Note 2**
			400 kHz mode	100	—	ns	
92	TSU:STO	Stop condition setup time	100 kHz mode	4.7	—	µs	
			400 kHz mode	0.6	—	µs	
109	TAA	Output valid from clock	100 kHz mode	—	3500	ns	**Note 1**
			400 kHz mode	—	1000	ns	
110	TBUF	Bus free time	100 kHz mode	4.7	—	µs	Time the bus must be free before a new transmission can start
			400 kHz mode	1.3	—	µs	
D102	Cb	Bus capacitive loading		—	400	pF	

Note 1: As a transmitter, the device must provide this internal minimum delay time to bridge the undefined region (min. 300 ns) of the falling edge of SCL to avoid unintended generation of Start or Stop conditions.

2: A Fast mode I²C-bus device can be used in a Standard mode I²C-bus system, but the requirement $T_{SU;DAT} \geq 250$ ns must then be met. This automatically is the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line,
T_R max.+$T_{SU;DAT}$ = 1000 + 250 = 1250 ns (according to the Standard mode I²C bus specification) before the SCL line is released.

APPENDIX B: CAN OVERVIEW

This appendix provides an overview of the Controller Area Network (CAN) bus. The CAN Section of this reference manual discusses the implementation of the CAN protocol for that hardware module.

B.1 CAN Bus Background

The CAN is a serial communications protocol which efficiently supports distributed real-time control with a very high level of security.

Its domain of application ranges from high speed networks to low cost multiplex wiring. In automotive electronics, engine control units, sensors, anti-skid-systems, etc., are connected using CAN with bit rates up to 1 Mbit/sec. The silicon cost is also low enough to be cost effective at replacing wiring harnesses in the automobile. The robustness of the bus in noisy environments and the ability to detect and recover from fault conditions makes the bus suitable for industrial control applications such as DeviceNet, SDS and other field bus protocols.

CAN is an asynchronous serial bus system with one logical bus line. It has an open, linear bus structure with equal bus nodes. A CAN bus consists of two or more nodes. The number of nodes on the bus may be changed dynamically without disturbing the communication of other nodes. This allows easy connection and disconnection of bus nodes (e.g., for addition of system function, error recovery or bus monitoring).

The bus logic corresponds to a “wired-AND” mechanism, “recessive” bits (mostly, but not necessarily equivalent to the logic level “1”) are overwritten by “dominant” bits (mostly logic level “0”). As long as no bus node is sending a dominant bit, the bus line is in the recessive state, but a dominant bit from any bus node generates the dominant bus state. Therefore, for the CAN bus line, a medium must be chosen that is able to transmit the two possible bit states (dominant and recessive). One of the most common and cheapest ways is to use a twisted wire pair. The bus lines are then called “CANH” and “CANL”, and may be connected directly to the nodes or via a connector. There's no standard defined by CAN regarding the connector to use. The twisted wire pair is terminated by terminating resistors at each end of the bus line. The maximum bus speed is 1 Mbit, which can be achieved with a bus length of up to 40 meters. For bus lengths longer than 40 meters, the bus speed must be reduced (a 1000 m bus can be realized with a 40 Kbit bus speed). For a bus length above 1000 meters, special drivers should be used. At least 20 nodes may be connected without additional equipment. Due to the differential nature of transmission, CAN is insensitive to EMI because both bus lines are affected in the same way, which leaves the differential signal unaffected. The bus lines can also be shielded to reduce the electromagnetic emission of the bus itself, especially at high baud rates.

The binary data is coded corresponding to the NRZ code (Non-Return-to-Zero; low level = dominant state; high level = recessive state). To ensure exact synchronization of all bus nodes, bit stuffing is used. This means that during the transmission of a message a maximum of five consecutive bits may have the same polarity. Whenever five consecutive bits of the same polarity have been transmitted, the transmitter inserts one additional bit of the opposite polarity into the bit stream before transmitting further bits. The receiver also checks the number of bits with the same polarity and removes the stuff bits from the bit stream (-destuffing).

In the CAN protocol it is not bus nodes that are addressed, but the address information contained in the messages that are transmitted. This is done via an identifier (part of each message) which identifies the message content (e.g., engine speed, oil temperature etc.). The identifier additionally indicates the priority of the message. The lower the binary value of the identifier, the higher the priority of the message.

For bus arbitration, CSMA/CD with NDA is used (Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration). If bus node A wants to transmit a message across the network, it first checks that the bus is in the Idle state (“Carrier Sense“) (i.e., no node is currently transmitting). If this is the case (and no other node wishes to start a transmission at the same moment), node A becomes the bus master and sends its message. All other nodes switch to Receive mode during the first transmitted bit (Start-Of-Frame bit). After correct reception of the message (which is acknowledged by each node), each bus node checks the message identifier and stores the message, if required. Otherwise, the message is discarded.

If two or more bus nodes start their transmission at the same time (“Multiple Access”), collision of the messages is avoided by bitwise arbitration (“Collision Detection/Non-Destructive Arbitration” together with the “Wired-AND” mechanism, “dominant” bits override “recessive” bits). Each node sends the bits of its message identifier (MSB first) and monitors the bus level. A node that sends a recessive identifier bit but reads back a dominant one loses bus arbitration and switches to Receive mode. This condition occurs when the message identifier of a competing node has a lower binary value (dominant state = logic 0) and therefore, the competing node is sending a message with a higher priority. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message. All other nodes automatically try to repeat their transmission once the bus returns to the Idle state. It is not permitted for different nodes to send messages with the same identifier as arbitration could fail leading to collisions and errors.

The original CAN specifications (versions 1.0, 1.2 and 2.0A) defined the message identifier as having a length of 11 bits giving a possible 2048 message identifiers. The specification has since been updated (to version 2.0B) to remove this possible limitation. CAN specification, version 2.0B, allows message identifier lengths of 11 and/or 29 bits to be used (an identifier length of 29 bits allows over 536 Million message identifiers). Version 2.0B CAN is also referred to as “Extended CAN”, and versions 1.0, 1.2 and 2.0A are referred to as “Standard CAN”.

B.2 Different CAN Implementations

B.2.1 Standard CAN, Extended CAN

Those data frames and remote frames, which only contain the 11-bit identifier, are called standard frames according to CAN specification V2.0A. With these frames, 2048 different messages can be identified (identifiers 0-2047). However, the 16 messages with the lowest priority (2032-2047) are reserved. Extended frames, according to CAN specification V2.0B, have a 29-bit identifier. As already mentioned, this 29-bit identifier is made up of the 11-bit identifier (“Base ID”) and the 18-bit Extended identifier (“ID Extension”).

CAN modules specified by CAN V2.0A are only able to transmit and receive standard frames according to the Standard CAN protocol. Messages using the 29-bit identifier cause errors. If a device is specified by CAN V2.0B, there is one more distinction. Modules named “Part B Passive” can only transmit and receive standard frames, but tolerate extended frames without generating error frames. “Part B Active” devices are able to transmit and receive both standard and extended frames.

B.3 Basic CAN, Full CAN

There is one more CAN characteristic concerning the interface between the CAN module and the host CPU, dividing CAN chips into “Basic CAN” and “Full CAN” devices. This has nothing to do with the used protocol though (Standard or Extended CAN), which makes it possible to use both Basic and Full CAN devices in the same network.

In the Basic CAN devices, only basic functions of the protocol are implemented in hardware, (e.g., the generation and the check of the bit steam). The decision, if a received message has to be stored or not (acceptance filtering), and the whole message management, has to be done by software (i.e., by the host CPU). Mostly the CAN chip only provides one transmit buffer and one or two receive buffers. So the host CPU load is quite high using Basic CAN modules, therefore these devices should only be used at low baud rates and low bus loads with only a few different messages. The advantages of Basic CAN are the small chip size leading to low costs of these devices.

Full CAN devices do the whole bus protocol in hardware, including the acceptance filtering and the message management. They contain several so called message objects which handle the identifier, the data, the direction (receive or transmit) and the information Standard CAN/Extended CAN. During the initialization of the device, the host CPU defines which messages are to be sent and which are received. The host CPU is informed by interrupt if the identifier of a received message matches with one of the programmed (receive-) message objects. In this way, the CPU load is reduced. Using Full CAN devices, high baud rates and high bus loads with many messages can be handled. These chips are more expensive than the Basic CAN devices, though.

Many Full CAN chips provide a “Basic-CAN Feature”. One of their messages objects can be programmed in a way that every message is stored there that does not match with one of the other message objects. This can be very helpful in a number of applications.

B.4 ISO Model

The lSO/OSl Reference Model defines the layers of protocol of a communication system, as shown in Figure B-1. At the highest end, the applications need to communicate between each other. At the lowest end, some physical medium is used to provide electrical signaling.

The higher levels of the protocol are run by software. Typically, only the application layer is implemented. Within the CAN bus specification, there is no definition of the type of message or the contents or meaning of the messages transferred. These definitions are made in systems such as Volcano, the Volvo automotive CAN specification; J1939, the U.S. heavy truck multiplex wiring spec; and Allen-Bradley DeviceNet and Honeywell SDS, examples of industrial protocols.

The CAN bus module definition encompasses two levels of the overall protocol.

- The Data Link Layer
  - The Logical Link Control (LLC) sub-layer
  - The Medium Access Control (MAC) sub-layer
- - The Physical Layer
  - The Physical Signaling (PLS) sub-layer

The LLC sub layer is concerned with Message Filtering, Overload Notification and Error Recovery Management. The scope of the LLC sub-layer is:

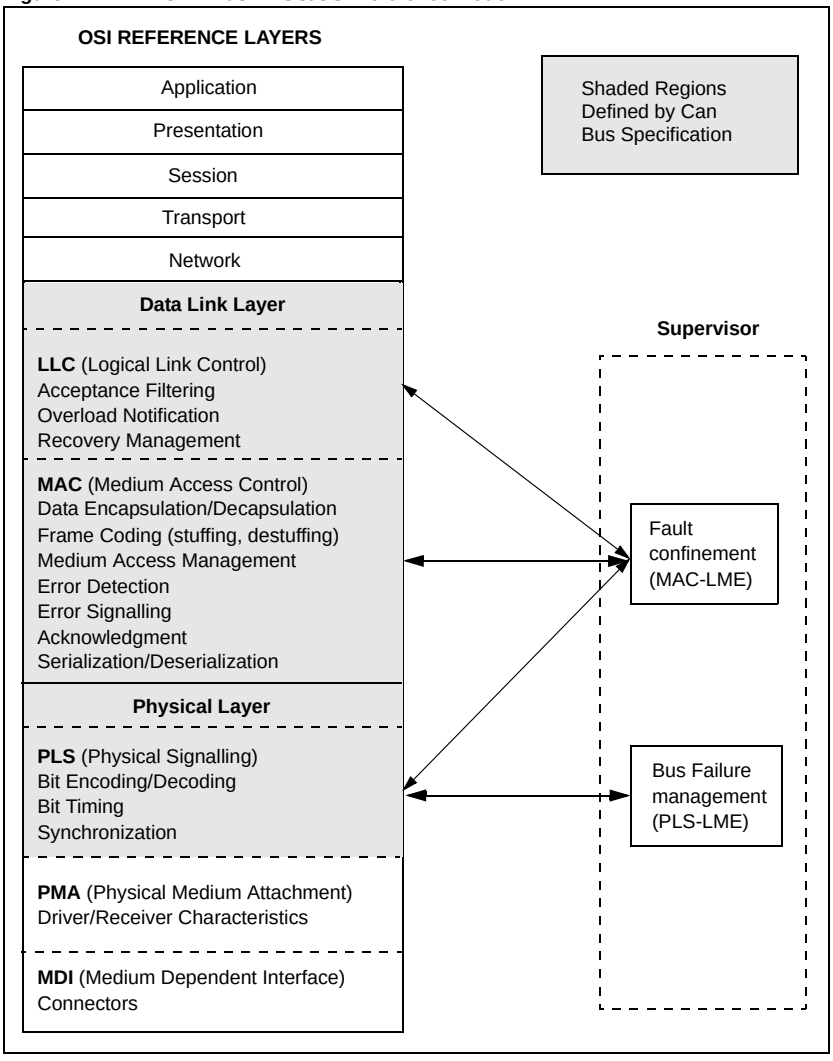
- To provide services for data transfer and remote data request
- To decide which messages received by the LLC sub layer are actually to be accepted
- To provide means for error recovery management and overload notifications

The MAC sub-layer represents the kernel of the CAN protocol. The MAC sub-layer defines the transfer protocol (i.e., controlling the Framing, Performing Arbitration, Error Checking, Error Signalling and Fault Confinement). It presents messages received from the LLC sub-layer and accepts messages to be transmitted to the LLC sub-layer. Within the MAC sub-layer, it is decided whether the bus is free for starting a new transmission or whether a reception is just starting. The MAC sub-layer is supervised by a management entity called Fault Confinement, which is a self-checking mechanism for distinguishing short disturbances from permanent failures. Also, some general features of the bit timing are regarded as part of the MAC sub-layer.

The physical layer defines the actual transfer of the bits between the different nodes with respect to all electrical properties. The PLS sub-layer defines how signals are actually transmitted, and therefore deals with the description of Bit Timing, Bit Encoding and Synchronization.

The lower levels of the protocol are implemented in driver/receiver chips and the actual interface, such as twisted pair wiring or optical fiber etc. Within one network, the physical layer has to be the same for all nodes. The Driver/Receiver Characteristics of the Physical Layer are not defined so as to allow transmission medium and signal level implementations to be optimized for their application. The most common example is defined in the ISO11898 Road Vehicles Multiplex Wiring specification.

Figure B-1: CAN Bus in ISO/OSI Reference Model



B.5 CAN Bus Features

CAN has the following properties:

- Prioritization of messages
- Latency times ensured
- Configuration flexibility
- Multi-cast reception with time synchronization
- System wide data consistency
- Multi-master
- Error detection and signaling
- Automatic retransmission of corrupted messages
- Distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes

1. Messages – information on the bus is sent in fixed format messages of different but limited length. When the bus is free any connected unit may start to transmit a new message.
2. Information Routing – In CAN systems, a CAN node does not make use of any information about the system configuration (e.g., station addresses).
3. System Flexibility – Nodes can be added to the CAN network without requiring any change in the software or hardware of any node and application layer.
4. Message Routing – The content of a message is named by an Identifier. The Identifier does not indicate the destination of the message, but describes the meaning of the data, so that all nodes in the network are able to decide by Message Filtering whether the data is to be acted upon by them or not.
5. Multicast – As a consequence of the concept of Message Filtering, any number of nodes can receive and simultaneously act upon the same message.
6. Data Consistency – Within a CAN network, it is ensured that a message is simultaneously accepted either by all nodes or by no node. Thus, data consistency of a system is achieved by the concepts of multicast and by error handling.
7. Bit Rate – The speed of CAN may be different in different systems. However, in a given system, the bit-rate is uniform and fixed.
8. Prioritization – The Identifier defines a static message priority during bus access.
9. Remote Data Request – By sending a remote frame, a node requiring data may request another node to send the corresponding data frame. The date frame and the corresponding remote frame are named by the same Identifier.
10. Multimaster – When the bus is free, any unit may start to transmit a message. The unit with the message of higher priority to be transmitted gains bus access.
11. Arbitration – Whenever the bus is free, any unit may start to transmit a message. If two or more units start transmitting messages at the same time, the bus access conflict is resolved by bitwise arbitration using the Identifier. The mechanism of arbitration ensures that neither information nor time is lost. If a data frame and a remote frame with the same Identifier are initiated at the same time, the data frame prevails over the remote frame. During arbitration, every transmitter compares the level of the bit transmitted with the level that is monitored on the bus. If these levels are equal, the unit may continue to send. When a ‘recessive’ level is sent and a ‘dominant’ level is monitored, the unit has lost arbitration and must withdraw without sending one more bit.
12. Safety – In order to achieve the highest safety of data transfer, powerful measures for error detection, signaling and self-checking are implemented in every CAN node.
13. Error Detection – For detecting errors the following measures have been taken:
    - Monitoring (transmitters compare the bit levels to be transmitted with the bit levels detected on the bus)
    - Cyclic Redundancy Check
    - Bit Stuffing
    - Message Frame Check

The error detection mechanisms have the following properties:

- All global errors are detected
- All local errors at transmitters are detected
- Up to five randomly distributed errors in a message are detected
- Burst errors of length less than 15 in a message are detected
- Errors of any odd number in a message are detected

14. Error Signalling and Recovery Time – Corrupted messages are flagged by any node detecting an error. Such messages are aborted and are transmitted automatically. The recovery time from detecting an error until the start of the next message is at most 31 bit times, if there is no further error.
15. Fault Confinement – CAN nodes are able to distinguish short disturbances from permanent failures. Defective nodes are switched off.
16. Connections – The CAN serial communication link is a bus to which a number of units may be connected. This number has no theoretical limit. Practically the total number of units are limited by delay times, and/or electrical loads on the bus line.
17. Single Channel – The bus consists of a single channel that carries bits. From this data resynchronization, information can be derived. The way in which this channel is implemented is not fixed in this specification (i.e., single wire (plus ground), two differential wires, optical fires, etc.).
18. Bus values – The bus can have one of two complementary logical values; ‘dominant’ or ‘recessive’. During simultaneous transmission of ‘dominant’ and ‘recessive’ bits, the resulting bus value is ‘dominant’. For example, in case of a wired-AND implementation of the bus, the ‘dominant’ level would is represented by a logical ‘0’ and the ‘recessive’ level by a logical ‘1’. Physical states (e.g., electrical voltage, light) that represent the logical levels are not given in the specification.
19. Acknowledgment – All receivers check the consistency of the message being received and will acknowledge a consistent message and flag an inconsistent message.
20. Sleep Mode; Wake-up – To reduce the system's power consumption, a CAN device may be set into Sleep mode without any internal activity and with disconnected bus drivers. The Sleep mode is finished with a wake-up by any bus activity or by internal conditions of the system. On wake-up, the internal activity is restarted, although the MAC sub-layer will be waiting for the system's oscillator to stabilize and it will then wait until it has synchronized itself to the bus activity (by checking for eleven consecutive ‘recessive’ bits), before the bus drivers are set to “on-bus” again.

B.6 Frame Types

B.6.1 Standard Data Frame

A data frame is generated by a node when the node wishes to transmit data. The Standard CAN Data Frame is shown in Figure B-2. In common with all other frames, the frame begins with a Start-Of-Frame bit (SOF – dominant state) for hard synchronization of all nodes.

The SOF is followed by the Arbitration field consisting of 12 bits, the 11-bit Identifier (reflecting the contents and priority of the message) and the RTR bit (Remote Transmission Request bit). The RTR bit is used to distinguish a data frame (RTR – dominant) from a remote frame.

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 frame. The following bit is reserved, RB0, 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.

The data being sent follows in the Data field, which is of the length defined by the DLC above (1-8 bytes).

The Cyclic Redundancy field (CRC) follows and detects possible transmission errors. The CRC field consists of a 15-bit CRC sequence, completed by the recessive CRC Delimiter bit.

The final 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). From this, it can be seen that CAN belongs to the “in-bit-response” group of protocols. The recessive Acknowledge Delimiter completes the Acknowledge slot and may not be overwritten by a dominant bit.

B.7 Extended Data Frame

In the Extended CAN Data frame, shown in Figure B-3, the Start-of-Frame bit (SOF) is followed by the Arbitration field consisting of 38 bits. The first 11 bits are the 11 most significant bits of the 29-bit identifier (“Base-lD”). These 11 bits are followed by the Substitute Remote Request bit, SRR, which is transmitted as recessive. The SRR is followed by the lDE bit, which is recessive to denote that the frame is an extended CAN frame. It should be noted from this, that if arbitration remains unresolved after transmission of the first 11 bits of the identifier, and one of the nodes involved in arbitration is sending a standard CAN frame (11-bit identifier), then the standard CAN frame will win arbitration due to the assertion of a dominant lDE bit. Also, the SRR bit in an extended CAN frame must be recessive to allow the assertion of a dominant RTR bit by a node that is sending a standard CAN remote frame. The SRR and lDE bits are followed by the remaining 18 bits of the identifier (“lD-Extension”) and the Remote Transmission Request bit.

To enable standard and extended frames to be sent across a shared network, it is necessary to split the 29-bit extended message identifier into 11-bit (most significant) and 18-bit (least significant) sections. This split ensures that the Identifier Extension bit (IDE) can remain at the same bit position in both standard and extended frames.

The next field is the Control field, consisting of 6 bits. The first 2 bits of this field are reserved and are at dominant state. The remaining 4 bits of the Control field are the DLC and specify the number of data bytes.

The remaining portion of the frame (Data field, CRC field, Acknowledge field, End-Of-Frame and lntermission) is constructed in the same way as for a standard data frame.

B.8 Remote Frame

Normally data transmission is performed on an autonomous basis with the data source node (e.g., a sensor sending out a data frame). It is possible, however, for a destination node to request the data from the source. For this purpose, the destination node sends a “remote frame” with an identifier that matches the identifier of the required data frame. The appropriate data source node then sends a data frame as a response to this remote request.

There are 2 differences between a remote frame and a data frame, shown in Figure B-4. First, the RTR bit is at the recessive state and second, there is no data field. In the very unlikely event of a data frame and a remote frame with the same identifier being transmitted at the same time, the data frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the remote frame receives the desired data immediately.

B.9 Error Frame

An error frame is generated by any node that detects a bus error. Figure B-5 shows how an error frame consists of 2 fields, an error flag field followed by an error delimiter field. The error delimiter that consists of 8 recessive bits and allows the bus nodes to restart bus communications cleanly after an error. There are two forms of error flag fields. The form of the error flag field depends an the error status of the node that detects the error.

If an error-active node detects a bus error then the node interrupts transmission of the current message by generating an active error flag. The active error flag is composed of six consecutive dominant bits. This bit sequence actively violates the bit stuffing rule. All other stations recognize the resulting bit stuffing error and generates error frames themselves, called error echo flags. The error flag field therefore consists of between six and twelve consecutive dominant bits (generated by one or more nodes). The error delimiter field completes the error frame. After completion of the error frame, bus activity returns to normal and the interrupted node attempts to resend the aborted message.

If an error passive node detects a bus error, then the node transmits an error passive flag followed again by the error delimiter field. The error passive flag consists of six consecutive recessive bits, and therefore the error frame for an error passive node consists of 14 recessive bits. From this, it follows that, unless the bus error is detected by the bus master node that is actually transmitting, the transmission of an error frame by an error passive node will not affect any other node on the network. If the bus master node generates an error passive flag, then this may cause other nodes to generate error frames due to the resulting bit stuffing violation. After transmission of an error frame, an error passive node must wait for 6 consecutive recessive bits on the bus before attempting to rejoin bus communications.

B.10 Interframe Space

Interframe space separates a proceeding frame (of whatever type) from a following data or remote frame. Interframe space is composed of at least 3 recessive bits, called the intermission. This is provided to allow nodes time for internal processing before the start of the next message frame. After the intermission, the bus line remains in the recessive state (Bus idle) until the next transmission starts.

Figure B-2: Standard Data Frame

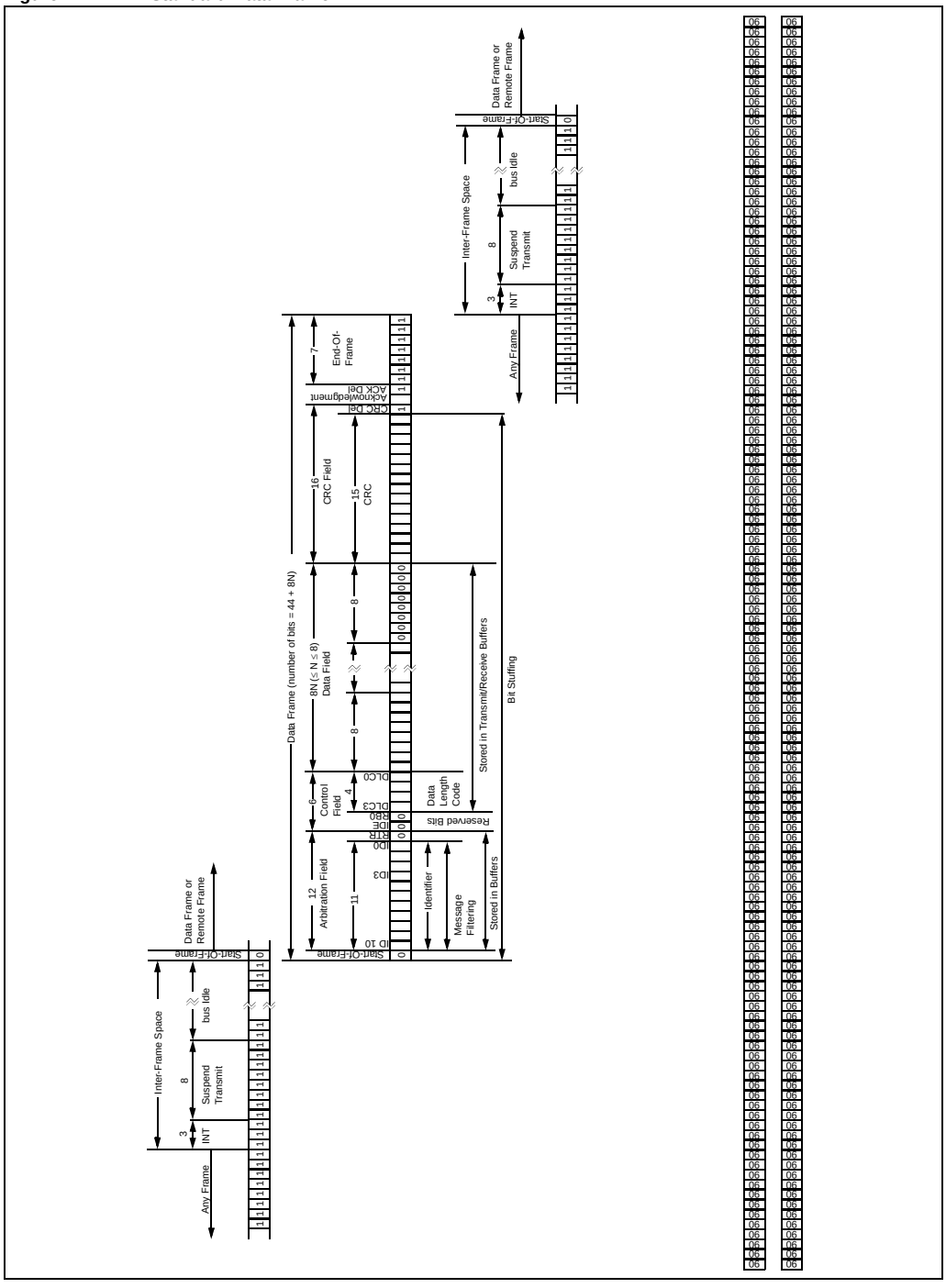


Figure B-3: Extended Data Format

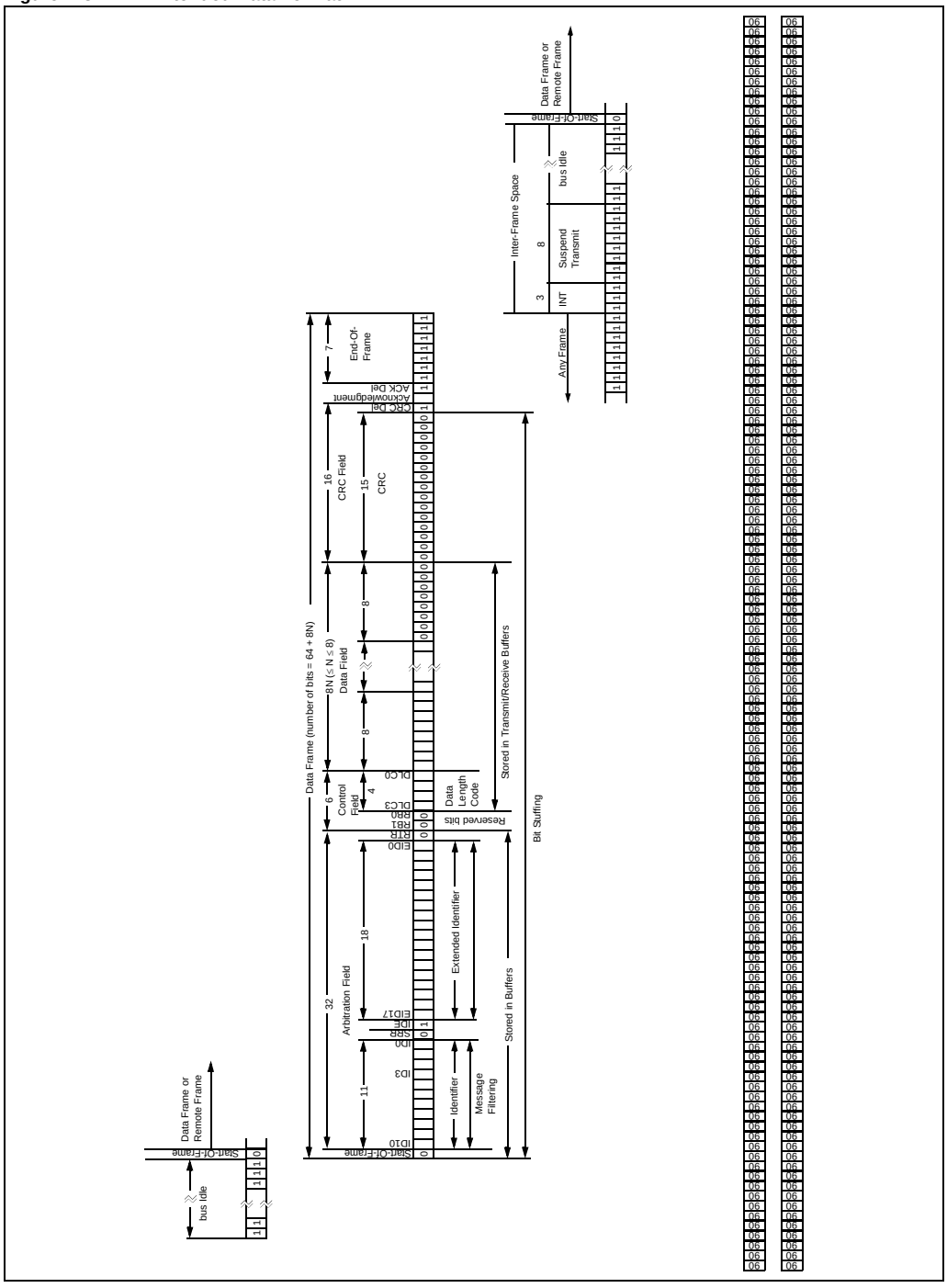


Figure B-4: Remote Data Frame

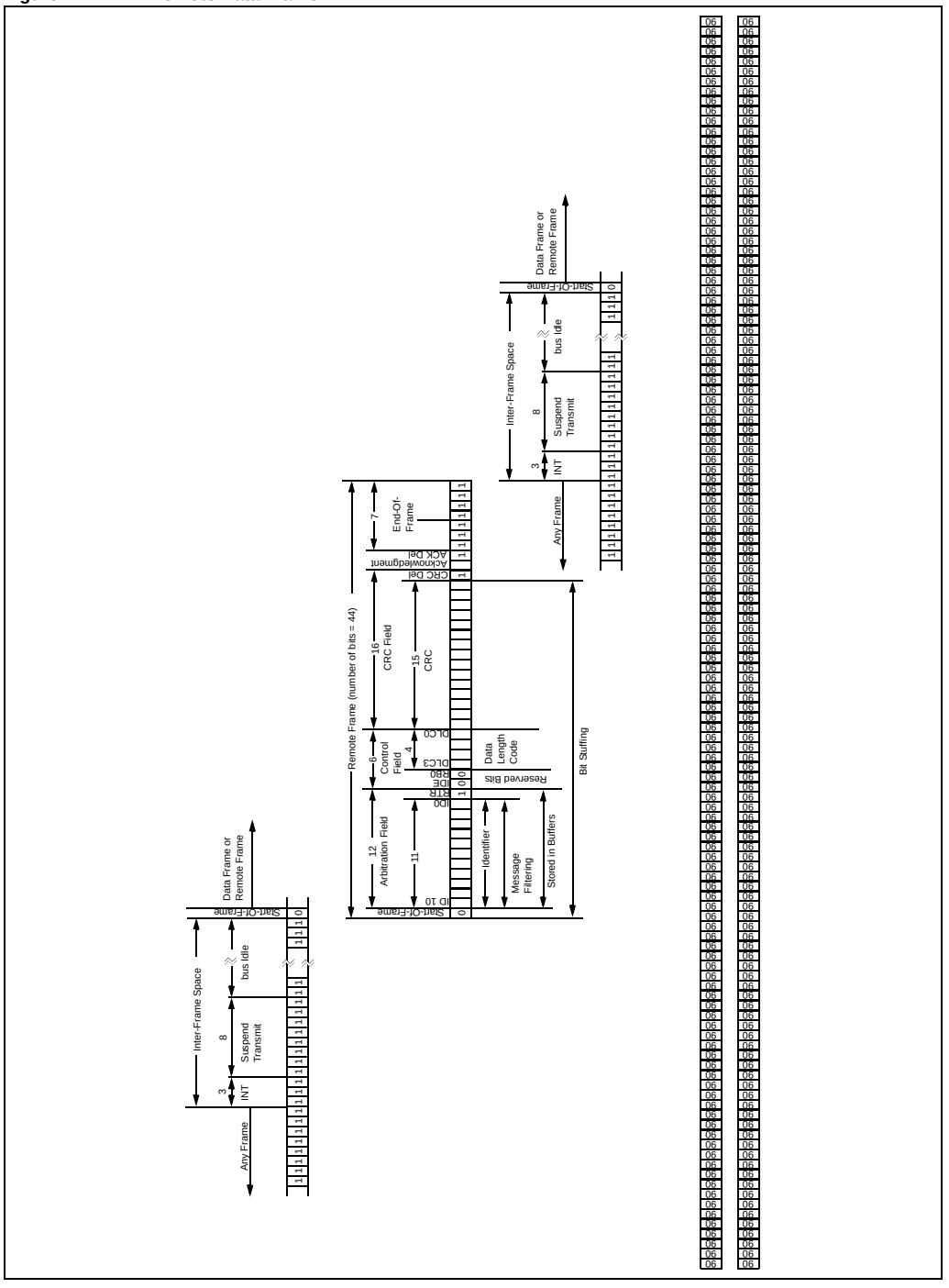
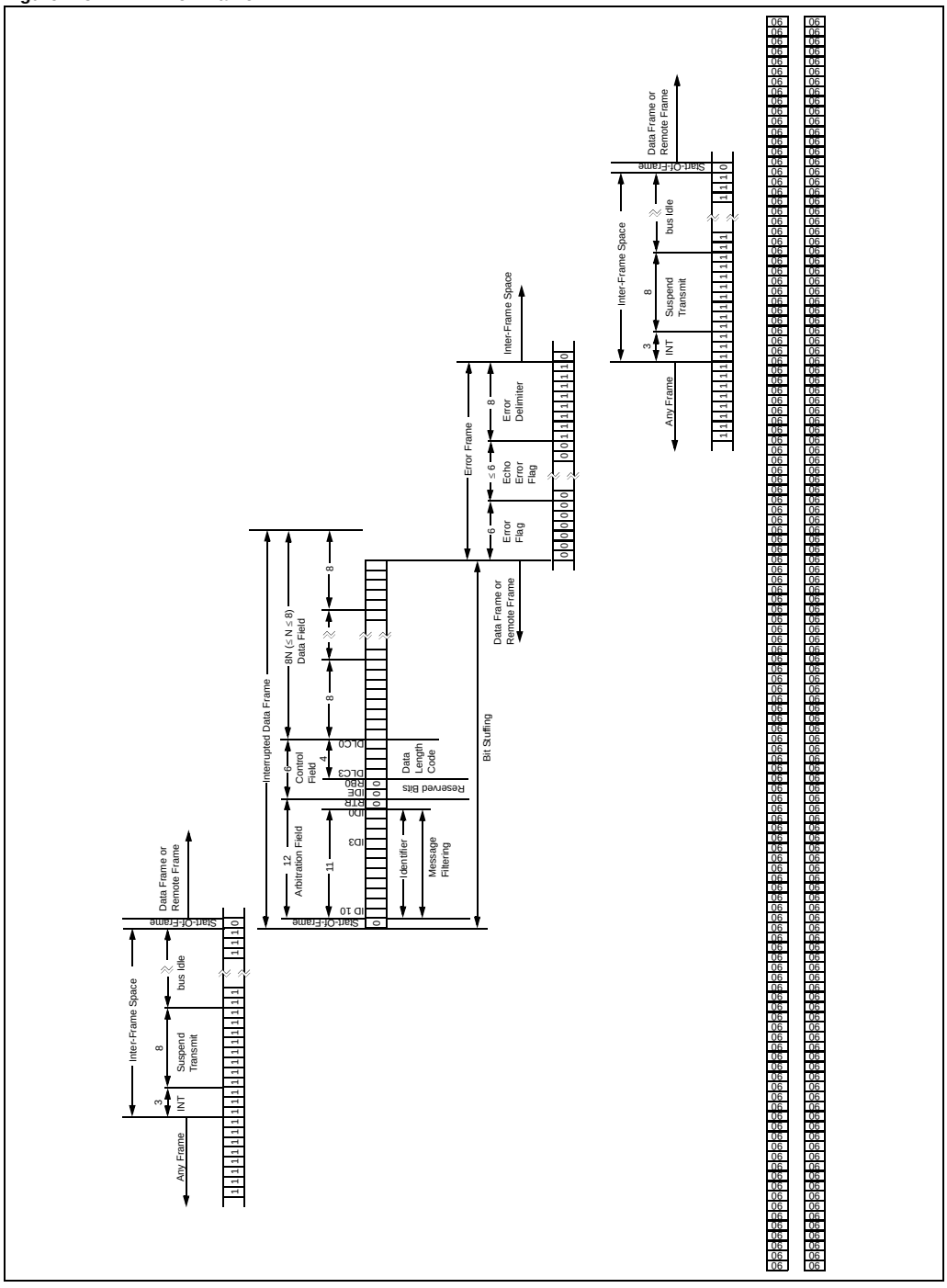


Figure B-5: Error Frame



B.11 Referenced Documents

Title	Document
Road Vehicles; Interchange of Digital Information, Controller Area Network Bosch CAN Specification Version 2.0	ISO11898

APPENDIX C: CODEC PROTOCOL OVERVIEW

This appendix summarizes audio coder/decoder (codec) protocols for Inter-IC Sound (I^2S) and AC-Link Compliant mode interfaces. Many codecs intended for use in audio applications support sampling rates between 8 kHz and 48 kHz and typically use one of the interface protocols previously mentioned. The Data Converter Interface (DCI) module automatically handles the interface timing associated with these codecs. No overhead from the CPU is required until the requested amount of data has been transmitted, and/or received by the DCI. Up to four data words may be transferred between CPU interrupts.

C.1 I^2S Protocol Description

Inter-IC Sound (I^2S) is a simple, three-wire bus interface used for the transfer of digital audio data between the following devices:

- DSP processors
- A/D and D/A converters
- Digital input/output interfaces

This Appendix information is intended to supplement the I^2S Protocol Specification®, which is published by Philips, Inc.

The I^2S bus is a time division multiplexed and transfers two channels of data. These two data channels are typically the left and right channels of a digital audio stream.

The I^2S bus has the following connection pins:

- SCK: The I^2S serial clock line
- SDx: The I^2S serial data line (input or output)
- WS: The I^2S word select line

Figure C-2A is a timing diagram for a data transfer. Serial data is transmitted on the I^2S bus in two's complement format with the MSb transmitted first. The MSb must be transferred first because the protocol allows different transmitter and receiver data word lengths. If a receiver is sent more bits than it can accept for a data word, the LSbits are ignored. If a receiver is sent fewer bits than its native word length, it must set the remaining LSbits to zero internally.

The WS line indicates the data channel transmitted. The following standard is used:

- WS = `0`: Channel 1 or left audio channel
- WS = `1`: Channel 2 or right audio channel

The WS line is sampled by the receiver on the rising edge of SCK and the MSb of the next data word is transmitted one SCK period after WS changes. The one period delay after WS changes provides the receiver time to store the previously transmitted word and prepare for the next word. Serial data sent by the transmitter is placed on the bus on the falling edge of SCK and is latched by the receiver on the rising edge of SCK.

Any device may act as the system master in an I^2S system. The system master generates the SCK and WS signals. Typically, the transmitter is the system master, but the receiver or a third device may perform this function. Figure C-1 shows possible I^2S bus configurations. Although it is not indicated in Figure C-1, the two connected devices may have both a data transmit and a data receive connection.

Figure C-1: I^2S Bus Connections

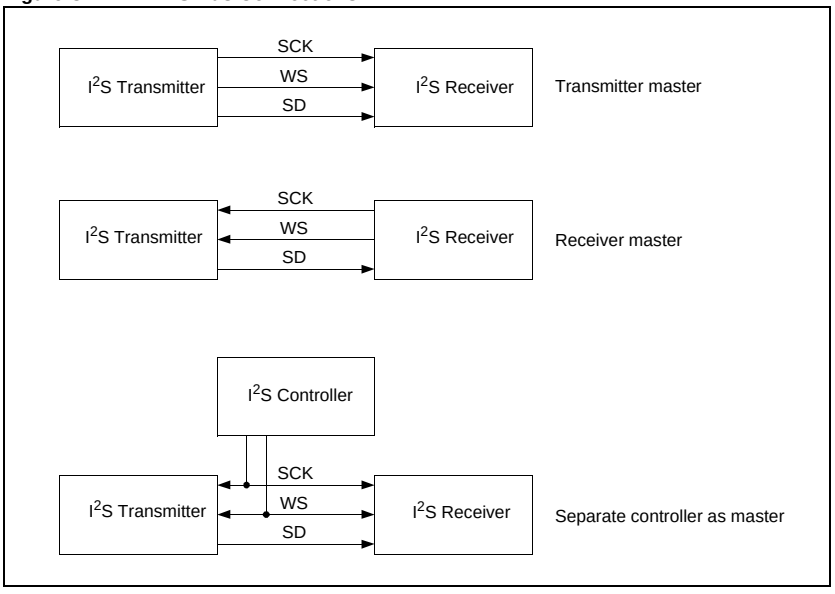
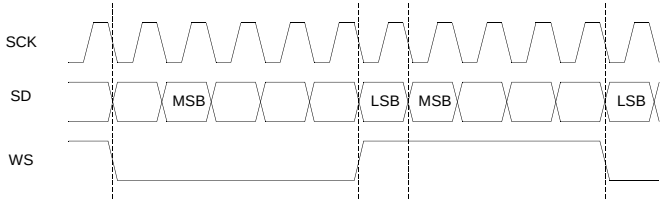


Figure C-2: I^2S Interface Timing Diagram



Note: A 5 bit transfer is shown here for illustration purposes. The I^2S protocol does not specify word length – this is system dependent.

C.2 AC '97 Protocol

The Audio Codec '97 (AC '97) specification defines a standard architecture and digital interface protocol for audio codecs used in PC platforms. The digital interface protocol for an AC '97 compliant codec is called AC-Link and is the focus of this discussion. The specific requirements and features of the AC '97 controller device are not described here.

This Appendix information is intended to supplement the AC '97 Component Specification document, published by Intel Corporation.

C.3 AC-Link Signal Descriptions

All AC-Link signals are derived from the AC '97 master clock source. The recommended clock source is a 24.576 MHz crystal connected to the AC '97 codec to minimize clock jitter. The 24.576 MHz clock may also be provided by the AC '97 controller or by an external source.

All AC-Link signal names are referenced to the AC '97 controller, not the AC '97 codec. The controller is the device that generates the SYNC signal to initiate data transfers. Each signal is described in subsequent sections.

C.3.1 Bit Clock (BIT_CLK)

A 12.288 MHz BIT_CLK signal is provided by the master AC '97 codec in a system. The BIT_CLK signal is an input to the AC '97 controller and up to three slave AC '97 codec devices in the system. All data on the AC-Link transitions on the rising edge of BIT_CLK and is sampled by the receiving device on the falling edge of BIT_CLK.

C.3.2 Serial Data Output (SDO)

SDO is a time division multiplexed data stream sent to the AC '97 codec.

C.3.3 Serial Data Input (SDI)

SDI is the time division multiplexed data stream from the AC '97 codec.

C.3.4 SYNC

SYNC is a 48 kHz fixed rate sample synchronization signal that is supplied from the AC '97 controller to the AC '97 codec. The SYNC signal is derived by dividing the BIT_CLK signal by 256. The SYNC signal is high for 16 BIT_CLK periods and is low for 240 BIT_CLK periods. The SYNC signal only changes on the rising edge of BIT_CLK and its period defines the boundaries of one audio data frame.

C.3.5 Reset

The $\overline{\text{RESET}}$ signal is an input to each AC '97 codec in the system and resets the codec hardware.

C.4 AC-Link Protocol

C.4.1 AC-Link Serial Interface Protocol

The AC-Link serial data stream uses a time division multiplexed (TDM) scheme with a 256-bit data frame. Each data frame is subdivided into 13 time slots, numbered Slot #0 – Slot #12. Slot #0 is a special time slot that contains 16 bits. The remaining 12 slots are 20-bits wide.

Figure C-4 is an example of an AC-Link frame. The frame begins with a rising edge of the SYNC signal which is coincident with the rising edge of BIT_CLK. The AC '97 codec samples the assertion of SYNC on the falling edge of BIT_CLK that immediately follows. This falling edge marks the time when both the codec and controller are aware of the start of a new frame. On the next rising edge of BIT_CLK, the codec asserts the MSb of SDATA_IN and the codec asserts the first edge of SDATA_OUT. This sequence ensures that data transitions and subsequent sample points for both incoming and outgoing data streams are time aligned.

Slot #0, Slot #1 and Slot #2 have special use for status and control in the AC-Link protocol. The remaining time slots are assigned to certain types of digital audio data. The data assignment for Slot #3 – Slot #12 is dependent on the AC '97 codec that is selected, so the slot usage is summarized briefly here. For more details on slot usage, refer to the AC '97 Component Specification.

C.4.2 Slot #0, TAG Frame

Slot #0 is commonly called the ‘tag frame’. The tag frame has a bit location for each data time slot in the AC-Link protocol. These bits specify which time slots in a frame are valid for use by the controller. A “1” in a given bit position of Slot #0 indicates that the corresponding time slot within the current audio frame has been assigned to a data stream, and contains valid data. If a slot is “tagged” invalid, it is the responsibility of the source of the data, (AC ’97 codec for the input stream, AC ’97 controller for the output stream), to stuff all bit positions with 0s during the slot’s active time.

There are also special bits in the tag frame. The MSb of the tag frame for SDATA_OUT is a ‘Frame Valid’ Status bit. The Frame Valid bit serves as a global indicator to the codec that at least one time slot in the frame has valid data. If the entire frame is tagged invalid, the codec can ignore all subsequent slots in the frame. This feature implements sample rates other than 48 kHz.

The two LSbs of the SDATA_OUT tag frame indicate the codec address. Up to four AC ’97 codecs may be connected in a system. If only one codec is used in a system, these bits remain 0’s.

The MSb of the SDATA_IN is used as a ‘Codec Ready’ Status bit. If this bit location is a ‘0’, then the codec is powered down and/or not ready for normal operation. If the ‘Codec Ready’ bit is set, it is the responsibility of the controller to query the status registers in the codec to see which subsections are operable.

C.4.3 Slot #1 (Command Address) and Slot #2 (Command Data)

Slot #1 and Slot #2 also have special uses in the AC-Link protocol. These time slots are used for address and data values when reading or writing the AC ’97 codec control registers. These time slots must be tagged as valid in Slot #0 in order to read and write the control registers. The AC ’97 Component Specification allows for sixty-four (64) 16-bit control registers in the codec. Seven address bits are provided in the AC-Link protocol, but only even-numbered addresses are used. The odd numbered address values are reserved.

Slot #1 and Slot #2 for the SDATA_OUT line are called the Command Address and Command Data, respectively. The Command Address slot on the SDATA_OUT line is used to specify the codec register address and to specify whether the register access will be a read or a write. The Command Data slot on SDATA_OUT contains the 16-bit value that written to one of the codec control registers. If a read of the codec registers is being performed, the Command Data bits are set to ‘0’s.

Slot #1 and Slot #2 for the SDATA_IN line are called the Status Address and Status Data slots, respectively. The Status Address time slot echoes the register address previously sent to the codec. If this value is ‘0’, an invalid address was previously sent to the codec.

The Status Address time slot also has ten Slot Request bits. The Slot Request bits can be manipulated by the codec for applications with variable sample rates.

The Status Data time slot returns 16-bit data read from the codec control/status registers.

C.4.4 Slot #3 (PCM Left Channel)

Slot #3 in the SDATA_OUT signal is used for the composite digital audio left playback stream. For sound card applications, this is typically the combined .WAV audio and MIDI synthesizer output.

Slot #3 in the SDATA_IN signal is the left channel record data taken from the AC ’97 codec input mixer.

C.4.5 Slot #4 (PCM Right Channel)

Slot #4 in the SDATA_OUT signal is used for the composite digital audio right playback stream. For sound card applications, this is typically the combined .WAV audio and MIDI synthesizer output.

Slot #4 in the SDATA_IN signal is the right channel record data taken from the AC ’97 codec input mixer.

C.4.6 Slot #5 (Modem Line 1)

Slot #5 in the SDATA_OUT signal is used for modem DAC data. The default resolution for modem compatible AC '97 codecs is 16 bits. As in all time slots, the unused bits in the slot are set to '0'.

Slot #5 in the SDATA_IN signal is used for the modem ADC data.

C.4.7 Slot #6

Slot #6 in the SDATA_OUT signal is used for PCM Center Channel DAC data in 4 or 6-channel sound configurations.

Slot #6 in the SDATA_IN signal is used for dedicated microphone record data. The data in this slot allows echo cancellation algorithms to be used for speakerphone applications.

C.4.8 Slot #7

Slot #7 in the SDATA_OUT signal is used for PCM Left Channel DAC data in 4 or 6-channel sound configurations.

Slot #7 in the SDATA_IN signal is reserved for future use in the AC '97 Component Specification.

C.4.9 Slot #8

Slot #8 in the SDATA_OUT signal is used for PCM Right Channel DAC data in 4 or 6-channel sound configurations.

Slot #8 in the SDATA_IN signal is reserved for future use in the AC '97 Component Specification.

C.4.10 Slot #9

Slot #9 in the SDATA_OUT signal is used for PCM LFE DAC data in 6-channel sound configurations.

Slot #9 in the SDATA_IN signal is reserved for future use in the AC '97 Component Specification.

C.4.11 Slot #10 (Modem Line 2)

Slot #10 is used for the modem line 2 ADC and DAC data in modem compatible devices.

C.4.12 Slot #11 (Modem Handset)

Slot #11 is used for the modem handset ADC and DAC data in modem compatible devices.

C.4.13 Slot #12 (GPIO Control/Status)

The bits in Slot #12 are used for reading and writing GPIO pins in the AC '97 codec. The GPIO pins are provided for modem control functions on modem compatible devices.

Figure C-3: AC-Link Signal Connections

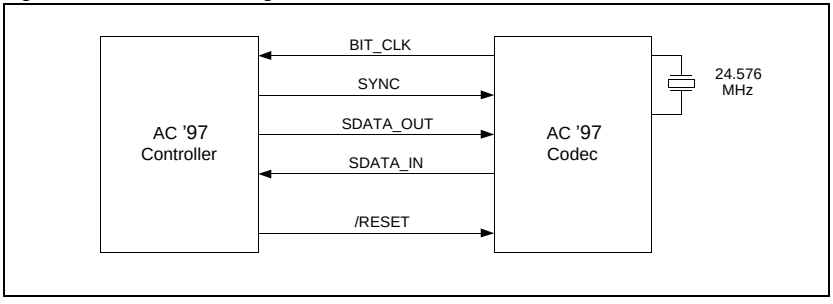


Figure C-4: AC-Link Data Frame

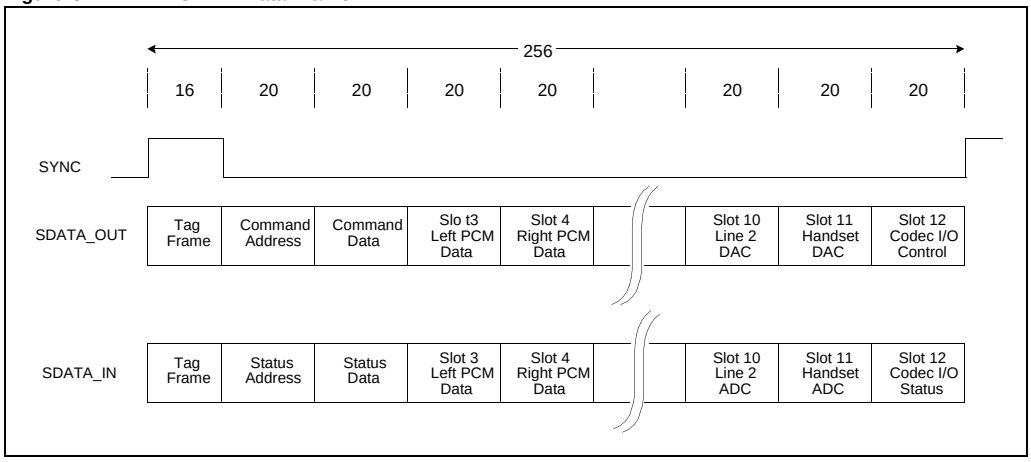


Figure C-5: SDATA_IN Bit Locations for SLOT#0, SLOT#1, SLOT#2

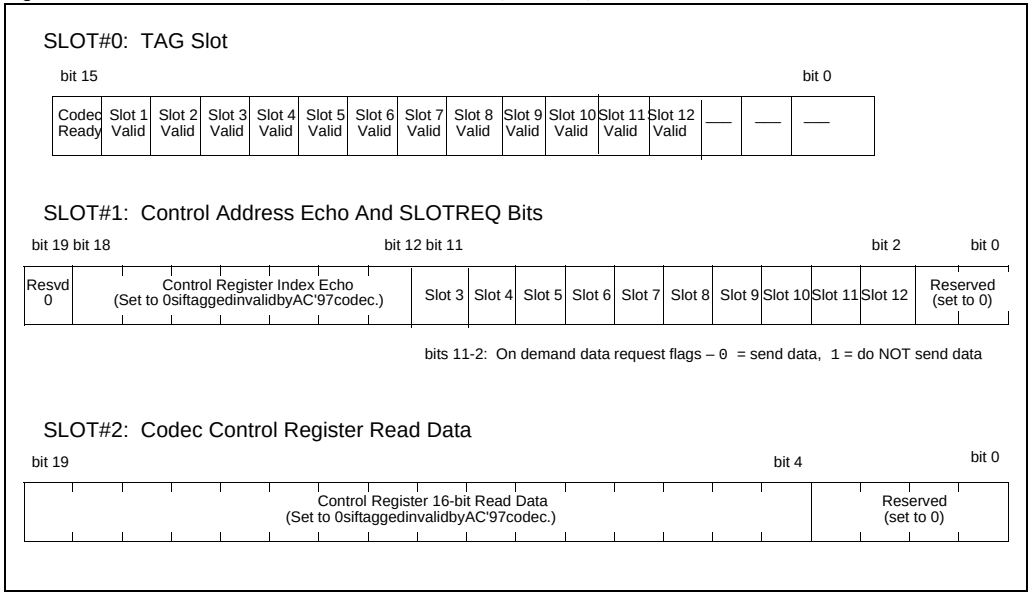
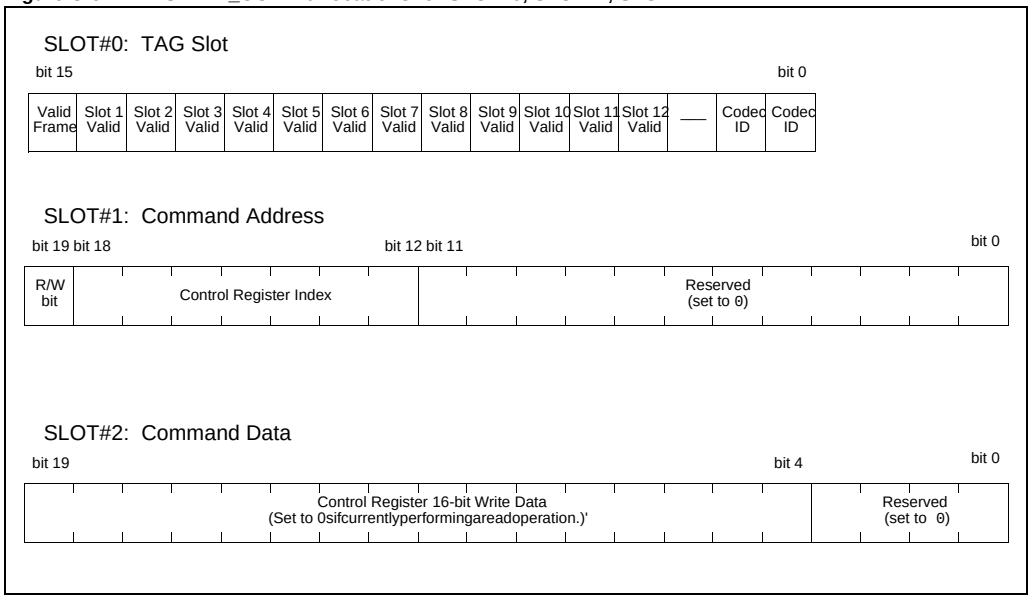


Figure C-6: SDATA_OUT Bit Locations for SLOT#0, SLOT#1, SLOT#2



NOTES: