ACCESS.bus™

Specifications — Version 2.2

Previous versions of this document are obsolete and should be discarded.
This document supersedes all previous versions.

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX: 1-408-991-3773

ACCESS.bus™

Specifications — Version 2.2

Previous versions of this document are obsolete and should be discarded.
This document supersedes all previous versions.

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX: 1-408-991-3773

February 1994

TABLE OF CONTENTS

SECTION 2 -- BASE PROTOCOL SPECIFICATION

SECTION 1

ACCESS.bus

Hardware Specification

SECTION 1

ACCESS.bus

Hardware Specification

February 1994

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone:	1-408-991-3517
FAX:	1-408-991-3773

1.0 Introduction

ACCESS.bus is a serial communication protocol between a computer host and its peripheral devices. It provides a simple, uniform and inexpensive way to connect peripheral devices such as keyboards, mice, joysticks, modems, monitors, and printers to a single computer port.

The ACCESS.bus protocol includes a physical layer based on the I^2C serial bus developed by Philips, and several software layers. The software layers include the base protocol, the device driver interface, and several specific device protocols (keyboard, locator, text, etc.).

The base protocol defines standard messages for device initialization, device identifications, address assignment, and a message envelope for device reports and control information. In the following discussion the host computer is simply called the **computer** and all the other partners on the bus are called **devices**. A bus transaction is called a **message**.

This document describes the various layers of the ACCESS.bus protocol starting with the physical layer hardware specification.

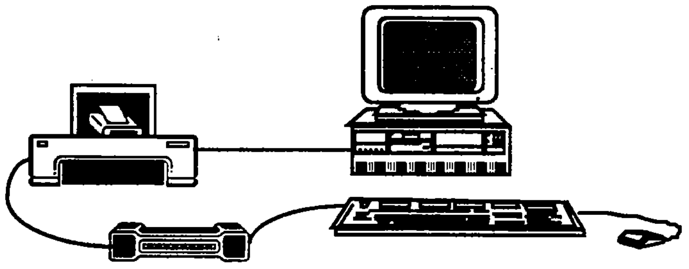

Figure 1.1: A Typical ACCESS.bus System

1.1 General Description

ACCESS.bus is based on two wires, serial data (SDA) and serial clock (SCL), which carry information between the devices connected to the bus. Following initialization, each device is recognized by a unique address and can operate as either a master transmitter or slave receiver. A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. The master device always transmits data to the slave. Any device addressed by the master is considered a slave.

The ACCESS.bus is a multi-master bus. Every device connected to the ACCESS.bus must be capable of being both a bus master and a bus slave.

The master generates the timing and terminates the transfer. Since more than one device will be connected to the ACCESS.bus, more than one master could try to initiate a data transfer at the same time. To avoid the chaos that might ensue from such an event - an arbitration procedure has been developed. This procedure relies on the wired-AND connection of all ACCESS.bus interfaces to the ACCESS.bus.

If two or more masters have to put information onto the bus, the first to produce a 'one' when the other produces a 'zero' will lose the arbitration. The clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line

Generation of clock signals on the ACCESS.bus is always the responsibility of master devices; each master generates its own clock signals when transferring data on the bus. Bus clock signals from a master can only be altered when they are stretched by a slow-slave device holding-down the clock line, or by another master when arbitration occurs.

1.2 General Characteristics

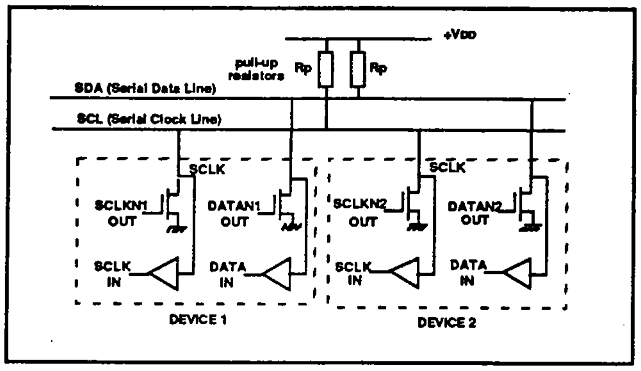

Figure 1.2: Connecting ACCESS.bus devices to the ACCESS.bus

Both SDA and SCL are bi-directional lines, connected to a positive supply voltage via a current source or pull-up resistor. When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain or open collector in order to perform the wired-AND function. Data on the ACCESS.bus can be transferred at a rate up to 100 kbit/s in the standard-mode. The number of interfaces connected to the bus is dependent on the bus capacitance limit of 1000 pF, the overall bus length of 10 meters, and the current available to power the devices.

The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see Figure 1.3).

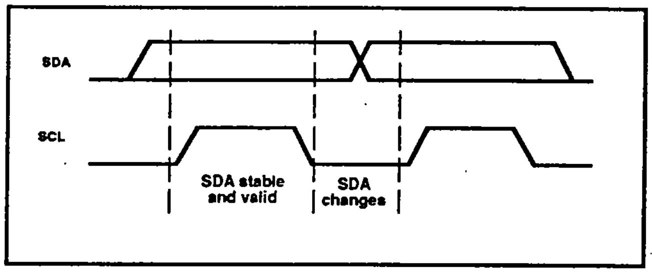

Figure 1.3: Bit transfer on the ACCESS.bus

1.2.1 START and STOP Conditions

Within the procedure of the ACCESS.bus, unique sequences are defined as START and STOP conditions (see Figure 1.4).

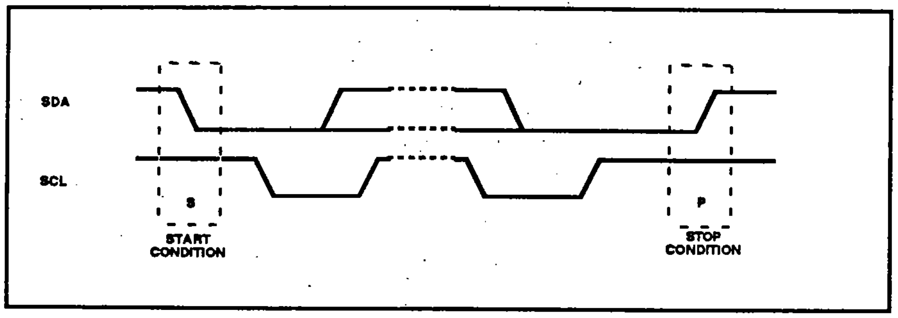

Figure 1.4: START and STOP conditions

A HIGH to LOW transition on the SDA line while SCL is HIGH indicates a START condition.

A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.

START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free again a certain time after the STOP condition.

1.3 Data Transfer

1.3.1 Byte format

Every byte put on the SDA line must be 8-bits long. The number of bytes that can be transmitted per transfer is restricted by the ACCESS.bus protocol. Each byte must be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first (Figure 1.5).

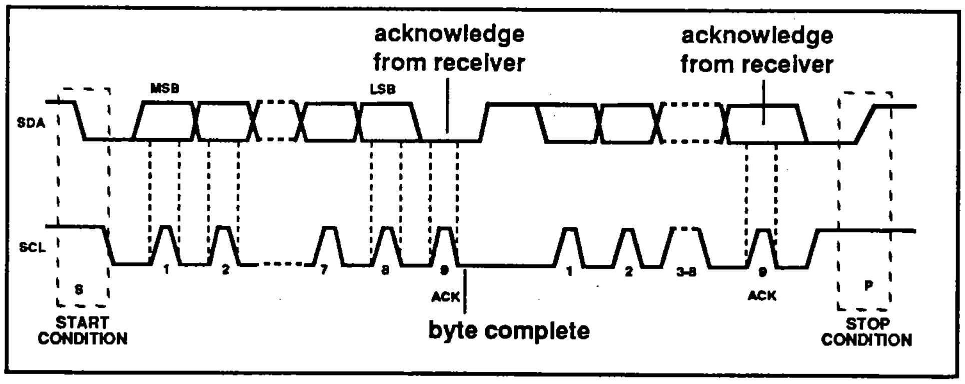

Figure 1.5: data transfer on the ACCESS.bus

If a receiver can't receive another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL LOW to force the transmitter into a wait state for a limited amount of time. Data transfer then continues when the receiver is ready for another byte of data and releases clock line SCL.

1.3.2 Acknowledge

The acknowledge related clock pulse is generated by the master. The master releases the SDA line (HIGH) during the acknowledge clock pulse.

The slave must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse (Figure 1.6) meeting set-up and hold times.

When a slave-receiver doesn't acknowledge the slave address (for example, it's unable to receive because it's performing some real-time function), the data line must be left HIGH by the slave. The master can then generate a STOP condition to abort the transfer. This is called 'negative acknowledge.'

If a slave-receiver does acknowledge the slave address but, some time later in the transfer cannot receive any more data bytes, the master must again abort the transfer. This is indicated by the slave generating the negative acknowledge on the first byte that cannot be received. The slave leaves the data line HIGH and the master generates the STOP condition.

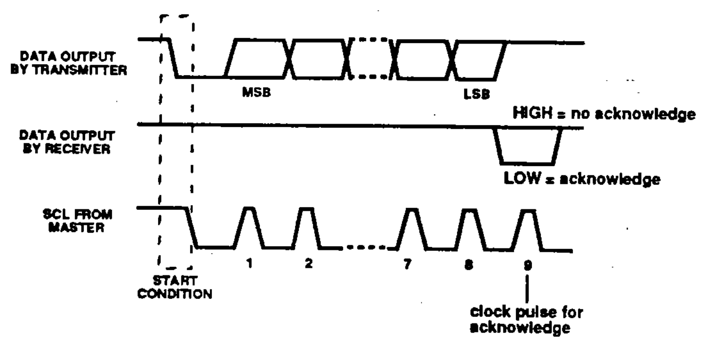

Figure 1.6 Acknowledge on the ACCESS.bus

1.4 Arbitration and Clock Generation

1.4.1 Synchronization

All masters generate their own clock on the SCL line to transfer messages on the ACCESS.bus. Data is only valid during the HIGH period of the clock. A defined clock is therefore needed for the bit-by-bit arbitration procedure to take place.

Clock synchronization is performed using the wired-AND connection of ACCESS.bus interfaces to the SCL line. This means that a HIGH to LOW transfer on the SCL line will cause the devices concerned to start counting off their minimum LOW period and, once a device clock has gone LOW, it will hold the SCL line in that state until the clock HIGH state is reached (Figure 1.7). However, the LOW to HIGH transition of this clock may not change the state of the SCL line if another clock is still within its LOW period. The SCL line will therefore be held LOW by the device with the longest LOW period. Devices with shorter LOW periods enter a HIGH wait-state during this time. ACCESS.bus defines a minimum actual bus speed for devices, to limit this type of clock stretching.

When all devices concerned have counted off their LOW period, the clock line will be released and go HIGH. There will then be no difference between the device clocks and the state of the SCL line, and all the devices will start counting their minimum HIGH periods. The first device to complete the HIGH period will again pull the SCL line LOW.

In this way, a synchronized SCL clock is generated with its LOW period determined by the device with the longest clock LOW period, and its HIGH period determined by the one with the shortest clock HIGH period.

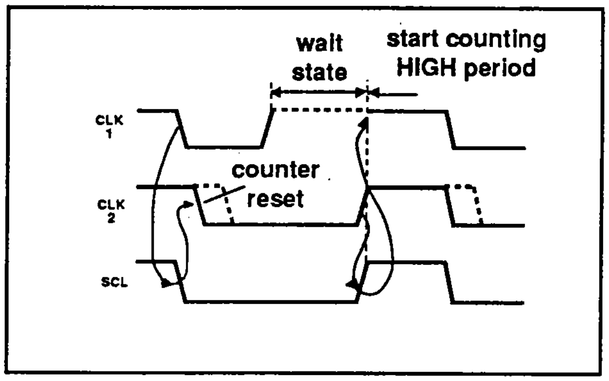

Figure 1.7: Clock synchronization during the arbitration procedure

1.4.2 Arbitration

A master may start a transfer only if the bus is free. Two or more masters may generate a START condition within the minimum hold time ($t_{HD;STA}$) which results in a valid START condition on the bus.

Arbitration takes place on the SDA line, while the SCL line is at the HIGH level, in such a way that the master which transmits a HIGH level, while another master is transmitting a LOW level will switch off its DATA output stage because the level on the bus doesn't correspond to its own level.

Arbitration can continue for many bits. Its first stage is a comparison of the slave address bits . If the masters are each trying to address the same device, arbitration continues with comparison of the sending master address. Arbitration should always be completed during the slave and master address transmission. Because address information on the ACCESS.bus is used for arbitration, no information is lost during this process.

A master which loses the arbitration can generate clock pulses until the end of the byte which it loses the arbitration.

If a master loses arbitration during the addressing stage, it's possible that the winning master is trying to address it. The losing master must therefore switch over immediately to its slave-receiver mode.

Figure 1.8 shows the arbitration procedure for two masters. Of course, more may be involved (depending on how many devices are connected to the bus).

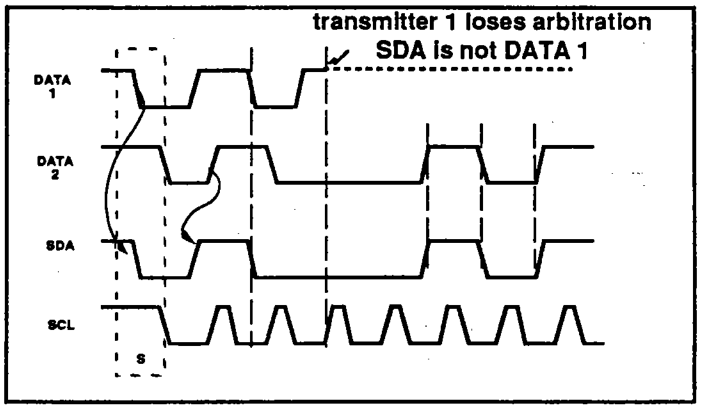

Figure 1.8: Arbitration procedure of two masters

The moment there is a difference between the internal data level of the master generating DATA 1 and the actual level on the SDA line, its data output is switched off, which means that a HIGH output level is then connected to the bus. This will not affect the data transfer initiated by the winning master.

1.4.3 Use of the clock synchronizing mechanism as a handshake

In addition to being used during the arbitration procedure, the clock synchronization mechanism can be used to enable receivers to cope with fast data transfers, on either a byte level or a bit level.

On the byte level, a device may be able to receive bytes of data at a fast rate, but needs more time to store a received byte or prepare another byte to be transmitted. Slaves can then hold the SCL line LOW after reception and acknowledgment of a byte to force the master into a wait state until the slave is ready for the next byte transfer in a type of handshake procedure.

On the bit level, a device such as a microcontroller without, or with only a limited hardware ACCESS.bus interface on-chip can slow down the bus clock by extending each clock LOW period. The speed of any master is thereby adapted to the internal operating rate of this device.

1.5 Format of 7-Bit Addresses

Data transfers follow the format shown in Figure 1.9. After the START condition (S), a slave address is sent. The address is 7 bits long followed by an eighth bit which is always a zero. The 'one' value is reserved for future expansion.

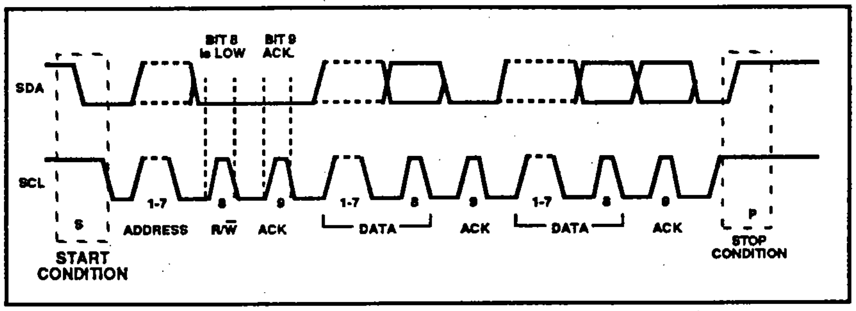

Figure 1.9: A complete data transfer

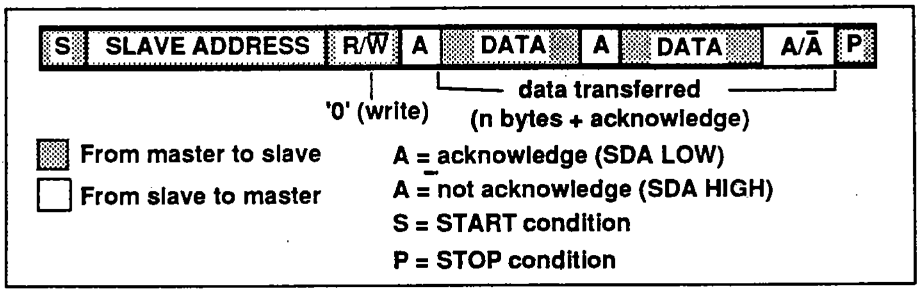

Figure 1.10: A master-transmitter addresses a slave receiver with a 7-bit address.

Notes:

1. Each byte is followed by an acknowledgment bit as indicated by the A or $\overline{A}$ blocks in the sequence.

2. ACCESS.bus compatible devices must reset their bus logic on receipt of a START or repeated START condition such that they all anticipate the sending of a slave address.

1.6 7-Bit Addressing

The addressing procedure for the ACCESS.bus is such that the first byte after the START condition normally determines which slave device will be selected by the master.

1.6.1 definition of bits in the first byte

The first seven bits of the first byte make up the slave address (Figure 1.11). The eighth bit is the LSB (least significant bit) and is always a zero in the current ACCESS.bus specification.

When an address is sent, each device in a system compares the first seven bits after the START condition with its address. If they match, the device considers itself addressed by the master as a slave-receiver.

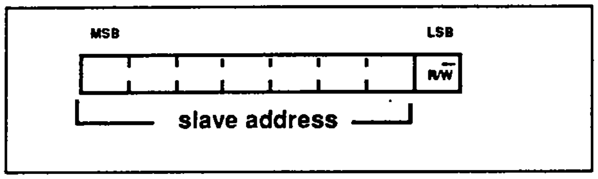

Figure 1.11: The first byte after the START procedure

1.7 Cabling and Connectors

ACCESS.bus uses a four-pin, shielded MOLEX SEMCONN or AMP SDL, modular-type connector. The MOLEX version is not keyed, the AMP version has key "D". This "D" key provides compatibility between AMP and MOLEX connectors.

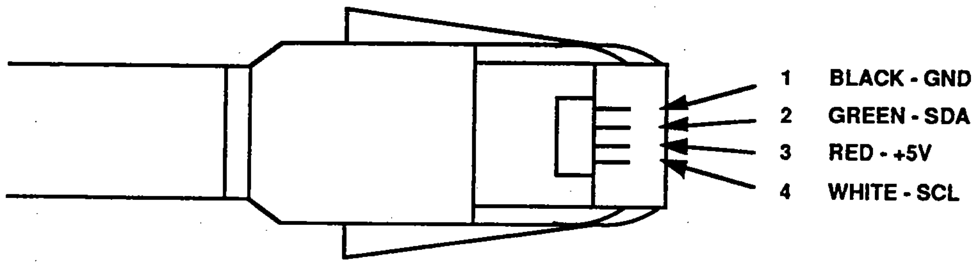

Figure 1.12: Shielded Modular Male Connector - Pins Side (Not to Scale)

The ACCESS.bus cable is a four conductor, low capacitance shielded cable (refer to Section 7 of this document for dimensioned outline drawings). The four conductors are used for ground - GND, serial data - SDA, plus five volts - +5V, and serial clock - SCL.

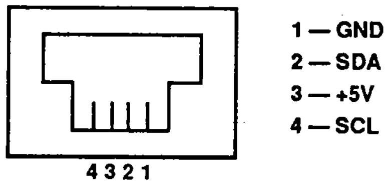

Figure 1.13: Female Connector, Front View (Not to Scale)

The host computer and the devices which do not have built-in captive cables, have one or two female connectors.

Hand-held devices such as mice or bar code readers have one male connector at the end of a captive cable.

Tee connections are allowed to connect multiple devices.

The ACCESS bus cable has the following wire sizes:

SDA and SCL wire size	AWG #28.
GND and +5V wire size	AWG #26.

The capacitance of the SCL and SDA conductors shall be less than 70 pF per meter between one conductor and other conductors connected to shield.

1.8 Power

ACCESS.bus host should supply +4.75V to 5.25V. The rise time of the +5V supply should be less than 100 milliseconds to insure power up reset for all the devices connected to the bus.

Each device connected to the bus should have a decoupling capacitor of 10μF connected to the +5V and GND lines.

ACCESS.bus devices may get the +5V from their own power supply and not use the host's +5V supply. The device's external power supply should provide a power up reset when power is applied to the bus by the host. The external +5V power supplies are connected to the ACCESS.bus by GND pin only, the +5V must be isolated from the ACCESS.bus +5V signal.

ACCESS.bus host's power supply or device's power supplies must supply a minimum of 50mA and a maximum of 1A. Current limiting or overcurrent protection is recommended. The current consumption of each device should be stated in the device documentation.

Each ACCESS.bus device must have two ratings: maximum current in miliampers ("I") and maximum capacitive load in pF ("C").

Table 1.1: Typical Ratings of ACCESS.bus devices

Device	Typical Rating
ACCESS.bus host computer	I200 C100
ACCESS.bus cable, 1 meter	I0 C65
ACCESS.bus keyboard	I110 C20
ACCESS.bus mouse	I30 C80

It is recommended that the ratings will be labeled on the device itself.

1.9 Maximum Number of Devices

The maximum number of ACCESS.bus devices is limited by three factors:

1. Device address - limited to 125 devices.
2. Power consumption - less than the host's power supply rating.
3. The capacitance on the bus - less than 1000pF.

The ACCESS.bus does not have any recommended topology or configurations. The only restriction is to satisfy the limits of power consumption from the host's power supply and not exceed the limit of 1000pF on the bus.

The maximum cable length is limited to maximum 10 meters.

For any cable length the bus must conform with the following limits:

1. The total capacitance on the bus should not to exceed 1000pF.
2. The voltage measured on pins +5V and GND of each device on the bus should not drop in any case below 4.5V.
3. Plugging-in a new device to the bus should not drop the supply voltage to the other devices below 4.5V

The cable length may exceed the limit of 10 meters by using an ACCESS.bus repeater for the SDA and SCL signals.

1.10 Cable Shield

The cable shield is connected only to the host computer connector's case. The host's connector case is connected also to the host's GND.

The cable's shield is not used by any device and not connected to any device.

The minimum resistance between the cable's shield to any signal on the bus is 100K Ohms, while the cable is connected to all the devices and disconnected from the host.

1.11 Pull-ups and Series Resistors

The host provides a pull-up resistor or a 6mA current source for the SDA and SCL open drain signals.

The pull-up resistors or the current sources provide 6mA per line to pull the line to HIGH logic level. The minimum pull-up resistor is 820 Ohms.

A 51 Ohm maximum series resistor is connected between the SDA and SCL pins on each device and the corresponding signals on the ACCESS bus. This resistor smoothes the bus signals and offers additional ESD (electrostatic discharge) protection to the device. Two clamping diodes to GND and +5V offer additional ESD protection. The diodes are needed only if they do not exist in the devices controller.

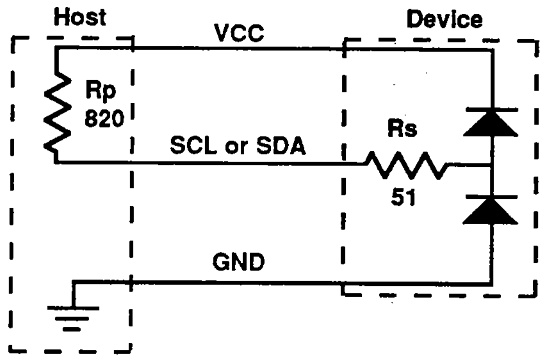

Figure 1.14: Pull-up and serial resistors for the SDA and SCL lines

1.12 Main Differences Between the I^2C and ACCESS.bus

1. The ACCESS.bus specifies a connector and a cable. I^2C-bus does not.
2. The V_{CC} and GND are supplied by the host in addition to the SDA and SCL. The I^2C-bus has only SDA and SCL lines.
3. The master device always transmits data to a slave. In I^2C-bus protocol the master can also read from the slave.
4. Every device connected to the ACCESS.bus must be capable of being a bus master and a bus slave.
5. The fast mode of the I^2C-bus was omitted from this version of the ACCESS.bus specification. The ACCESS.bus currently works only at 100Kbit/sec.
6. IOL = 6mA for the ACCESS.bus, vs. IOL = 3mA for the I^2C-bus. The pull-up resistors and the serial resistors for ACCESS.bus operation were decreased to allow IOL = 6mA.
7. The maximum capacitance per line was increased from 400pF for the I^2C-bus to 1000pF on the ACCESS.bus.
8. The ACCESS.bus maximum cable length without a repeater is 10 meters. The I^2C-bus does not define a maximum cable length.

1.13 Electrical Specifications and Timing For I/O Stages and Bus Lines

The I/O levels, and I/O current for ACCESS.bus devices are given in Table 1.2. The ACCESS.bus timing is given in Table 1.3. Figure 1.15 shows the timing definitions for the ACCESS.bus.

Table 1.2: Characteristics of SDA and SCL I/O stages for ACCESS.bus devices

Parameter	Symbol	standard-mode devices Min.	standard-mode devices Max.	Unit
Supply voltage measured at the host	V_{DD}	4.75	5.25	V
Supply voltage measured at the device	V_{DD}	4.5	-	V
LOW level input voltage: fixed input levels V_{DD} -related input levels	V_{IL}	-0.5	$0.3V_{DD}$	V
HIGH level input voltage: fixed input levels V_{DD} -related input levels	V_{IH}	$0.7V_{DD}$	*1)	V
LOW level output voltage (open drain or open collector: at 6 mA sink current	V_{OL}	0	0.6	V
Output fall time from $V_{IH\ min.}$ to $V_{IL\ max.}$ with a bus capacitance from 10 pF to 1000 pF: with up to 6 mA sink current at V_{OL}	t_{OF}	-	250 2)	ns
Input current each I/O pin with an input voltage between 0.6 V and $0.9V_{DD\ max.}$	I_i	-10	10	uA

1) maximum $V_{IH} = V_{DD\ max.} +0.5V$

2) c_b = capacitance of one bus line in pF. Note that the maximum t_F for the SDA and SCL bus lines quoted in Table 1.3 (300 ns) is longer than the specified maximum t_{OF} for the output stages (250 ns). This allows series protection resistors (R_s) to be connected between the SDA/SCL pins and the SDA/SCL bus lines.

The minimum HIGH and LOW periods of the SCL clock specified in Table 1.3 determine the maximum bit transfer rates of 100 kbit/s for standard-mode devices. Standard-mode ACCESS.bus devices must be able to follow transfers at their own maximum bit rates, either by being able to transmit or receive at that speed or by applying the clock synchronization procedure which will force the master into a wait state and stretch the LOW periods of the SCL signal. Of course, in the later case the bit transfer rate is reduced.

Table 1.3: Characteristics of SDA and SCL bus lines for ACCESS.bus devices

Parameter	Symbol	standard-mode Access.bus		Unit
		Min.	Max.	
SCL clock frequency [3]	f_{SCL}	0	100	kHz
Bus free time between a STOP and START condition	t_{BUF}	4.7	-	us
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	-	us
LOW period of the SCL clock	t_{LOW}	4.7	-	us
HIGH period of the SCL clock	t_{HIGH}	4.0	-	us
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	-	us
Data hold time: for Access.bus devices	$t_{HD;DAT}$	0[1]	-	us
Data set-up time	$t_{SU;DAT}$	250	-	ns
Rise time of both SDA and SCL signals [2]	t_R	-	1000	ns
Fall time of both SDA and SCL signals	t_F	-	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	-	us
Capacitive load for each bus line with pull up	C_b	-	1000	pF
Capacitive load for each bus line with current source	-	-	1500	pF
Cable length	-	-	10	meter
Pull-up resistor	R_p	820	-	Ohm
Serial resistor	R_s	-	51	Ohm
Cable Capacitance	C	-	70	pF/m

1) A device must internally provide a hold time of at least 300 ns for the SDA signal (referred to the $V_{IH\ min}$ of the SCL signal) in order to bridge the undefined region of the falling edge of SCL.

2) Rise time is measured from V_{SS} to 0.7 V_{DD}.

3) See Section 2.8.3 for minimum data rate definition.

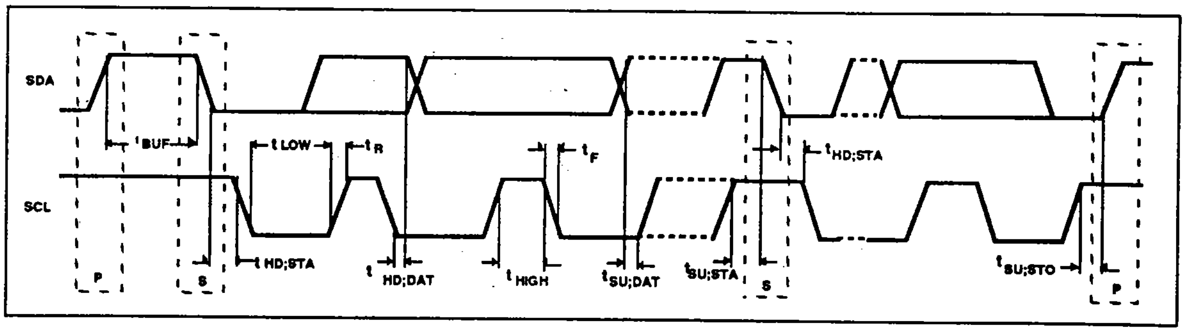

Figure 1.15: Definition of timing on the ACCESS.bus

SECTION 2

ACCESS.bus

Base Protocol Specification

SECTION 2

ACCESS.bus

Base Protocol Specification

February 1994

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX: 1-408-991-3773

2.0 ACCESS.bus Protocol

2.1 General Description

Every device on the bus must support the ACCESS.bus physical layer interface.

A message transmits information between a device and the computer or between the computer and one or more devices. There is one exception: a device may attempt to reset other devices assigned to the same address by sending a Reset message to itself.

Initially, all devices respond to a default power up address. During the configuration process the computer assigns a unique address to every device on the bus. ACCESS.bus supports multiple devices of the same type, or different types without switches or jumpers.

The bus supports dynamic reconfiguration while the system is operating. Connecting new devices shall not require powering down or rebooting the system before the new devices can be used ("hot plugging" is permitted).

2.2 Message Format

ACCESS.bus messages have the following format:

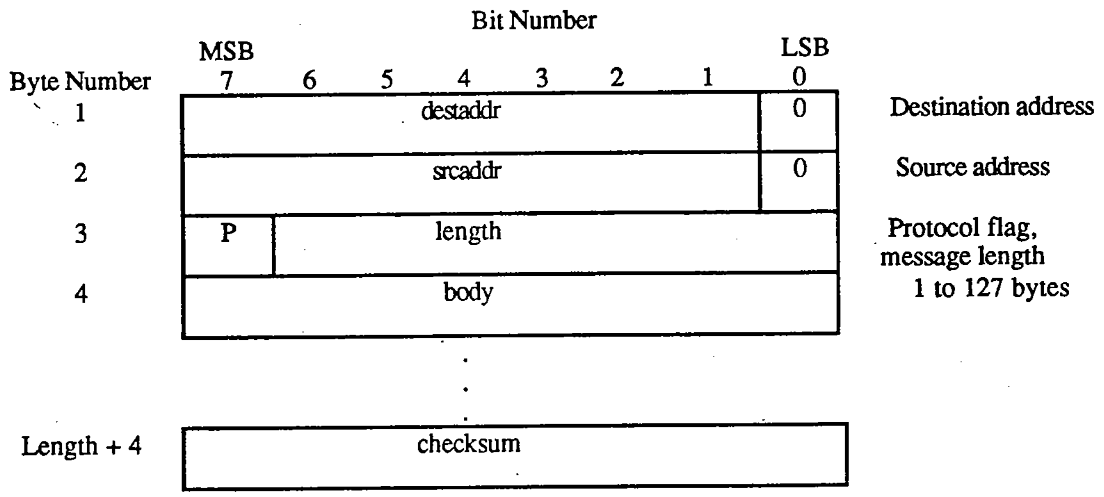

Figure 2.1: ACCESS.bus Message Format

Messages are either Device Data Stream (P=0) or control/status (P=1), as indicated by the protocol (P) flag. The minimum length of a message shall be four bytes. The maximum theoretical length of a message is 131 bytes (127 data bytes and four bytes for overhead); the maximum practical length is constrained by the transmission speed and the maximum time a device may hold the bus as master (see Timing Rules).

The message checksum shall be computed as the logical XOR of all previous bytes, including the message address. The checksum shall be computed such that the logical XOR of all previous bytes plus the ECC is equal to zero (0). A device or the computer shall only execute commands with a valid checksum.

ACCESS.bus data follows Big Endian bit order (see section 1). The most significant bits are always sent first.

The standard byte order for multibyte integers is Big Endian.

Example:

S	6E	A	50	A	81	A	F1	A	4E	A	P

Where:

S	START Signal
6E	destination address (device default)
A	Acknowledge pulse from receiver
50	source address (computer)
81	Control/Status length 1
F1	Identification Request
4E	Checksum
P	STOP Signal

Figure 2.2: ACCESS.bus Message Example

2.3 Standard ACCESS.bus Protocol Messages

The ACCESS.bus protocol defines nine required interface messages that are summarized below. Parameters defined within the body of the message are listed in parenthesis.

Table 2.1: Protocol Messages

Computer-to-device Messages	**Purpose**
Reset()	Force device to power-up state and default ACCESS.bus address.
Identification Request()	Ask device for its "identification string."
Assign Address(ID strng, new addr)	Tell device with matching "identification string" to change its address to "new address."
Capabilities Request(offset)	Ask device to send the fragment of its capabilities information that starts at "offset."
Enable Application Report	Enable or disable a device to send application reports to the host computer
Presence Check	Check if the device is present on the bus at the specific address.
Device-to-computer Messages	**Purpose**
Attention	Inform computer that a device has finished its power-up/reset test and needs to be configured.
Identification Reply(ID string)	Reply to Identification Request with device's unique "identification string."
Capabilities Reply(offset, data frag)	Reply to Capabilities Request with "data fragment," a fragment of the device's capabilities string; the computer uses "offset" to reassemble fragments.

2.4 Addressing

ACCESS.bus addresses follow the address format defined below. The LSB shall always be zero (0) indicating a master transmitter or write operation.

MSB							LSB
A7	A6	A5	A4	A3	A2	A1	0

Figure 2.3: ACCESS.bus Address Format

The following ACCESS.bus addresses are pre-assigned (even numbers only):

50h	host computer address
6Eh	power up default address for all devices
02-4Eh; 52-6Ch; 70-FEh	125 assignable ACCESS.bus device addresses

An ACCESS.bus computer implementation must support the 125 assignable device addresses noted above.

The computer address shall never be changed unless the computer wants to simulate a device. This feature may be useful for debugging.

At power up or after a reset command, devices will respond to the default address (6Eh).

The least significant bit of the source address field which is not used for addressing (indicates R/W in destination address) is reserved for future protocol extension. Conforming devices shall transmit this bit as zero and ignore received messages in which this bit is one until such extensions, if any, are defined.

2.5 Identification

ACCESS.bus is a bus-topology network that uses unique identification strings to distinguish devices. These strings are structured as follows:

protocol revision:	1 byte ("B")
module revision:	7 bytes (e.g., "V1.0 ")
vendor name:	8 bytes (e.g., "DEC ")
module name:	8 bytes (e.g., "LK501 ")
device number:	32-bit signed integer

The protocol revision shall be a single byte used to identify the protocol implementation to the computer. This specification defines protocol revision "B" (42h). Any new protocol revisions shall be approved by the ACCESS.bus Industry Group.

The module revision, vendor name, and module name strings are left justified ASCII character strings padded with spaces. The content of these fields shall be determined by the device vendor.

The device number string shall be a 32-bit two's complement signed integer and may be either a random number (if negative) or a unique serial number (if positive).

It is important to note that the host software should never attempt to recognize a device based on the hardware ID string. Host software should always rely on the capabilities string fields for device identification.

2.5.1 Random versus Serial Device Numbers

Interactive devices may use either a random number or a (fixed) serial number in their Identification Reply messages. Unique serial numbers are more expensive to implement, but allow device identification and usage to be remembered between sessions when the system has been turned off or device hot plugged.

When devices of the same type have to be physically identified, a fixed serial number may be used. Example:

Three identical ACCESS.bus laser printers are located in three different offices. Each printer reports a fixed serial number that is interpreted by the user 's software as the physical location of the printer.

The purpose of the 32-bit device number is to distinguish otherwise like devices with the same firmware. To aide ACCESS.bus management software, serial numbers are reported as positive (2's complement integers), while random numbers are always reported as negative.

If a pseudo random number is used in the Identification Reply message, it must be produced in a way that will help distinguish like devices.

Guideline: The number of clock cycles since power on at the time a command is received may be used as a pseudo random number. The natural dispersion of resonator frequencies is usually sufficient to separate otherwise identical devices.

A new pseudo random number must be generated if a reset command is received. The pseudo random number shall not change between identify reports unless an intervening power-up or reset command occurs.

2.6 Capabilities Information

Device capabilities is the set of information that describes the functional characteristics of an ACCESS.bus peripheral. The purpose of capabilities information is to allow software to recognize and use the features of bus devices without prior knowledge of their particular implementation. By having locator devices report their resolution, for example, generic software can be written to support a range of device resolutions. Capabilities information provides a level of device independence and modularity.

The structure of capabilities information is designed to be simple and compact for efficiency, but also extensible to support new devices without requiring changes to existing software or peripherals. These objectives are supported by making the structure hierarchical and representing capabilities information in a form that applications (and humans) can use directly. The capabilities information shall be an ASCII string constructed from a simple, readable grammar. The capabilities string for a locator might read as follows:

```
(
prot(locator)
type(mouse)
model(VSXXX-AA)
buttons(1(L)2(R)3(M))
dim(2) rel res(200 inch) range(-127 127)
d0(dname(X))
d1(dname(Y))
)
```

Capabilities information is normally constant for a device. However, the capabilities of some devices may change over time. Devices whose capabilities change will notify the host with an application Status Message when their capabilities have changed. The information shall describe the potential operating modes and characteristics of a device. From the point of view of an ACCESS.bus peripheral, capabilities information can simply be a string of bytes which is transferred to the computer via the Capabilities Request and Capabilities Reply commands.

Capabilities information must be placed in the capabilities string in the following order:

```
prot()
type()
```

model()
pwr()
bwm()
protocol specific portions of the Capabilities string
device specific portions of the Capabilities string

prot, type, and model must always be the first three items in the capabilities string and they must occur within the first sixty-four (64) characters. These three properties are used by the host software to identify the appropriate device driver for the particular device.

2.6.1 Capabilities String Syntax and Semantics

Capabilities information shall be an ASCII string constructed from the simple grammar as follows.

1. The terminal symbols of the grammar are STRING, TAG, WS, `(', and `)'.
2. WS is a sequence of one or more white space characters: SPACE, TAB, RETURN or LF.
3. A STRING is a sequence of one or more non-white space characters. All Capabilities data shall be represented as STRING: integers (123), floats (+3.0e8), strings (keyboard). Special characters may be included in STRINGs by escaping them as \xHH where HH represents two hexadecimal digits. SPACE, TAB, RETURN, LF, `(', `)', and `\' are special characters and must be escaped as \xHH to include them in STRINGs.
4. A TAG is a STRING which is immediately followed by a `('.
5. `(' and `)' are open and close parenthesis used for grouping.
6. The grammar allows STRINGs to be formed into lists separated by white space, lists with tagged elements, and nested lists according to the following rules (BNF).

```
Capabilities ::= ( cap string )
cap string ::= STRING
cap string ::= cap string WS cap string
cap string ::= TAG cap string )
cap string ::= TAG cap string ) cap string
```

Notes:

1. All 8-bits of characters are significant. This is to allow 8-bit multinational character sets such as ISO Latin-1 to be used.
2. Keyword comparisons are not case sensitive.
3. Capabilities are hierarchical. The meaning of a keyword within a tagged list depends on the tagged list in which it appears.

2.6.2 Standard Capabilities and Conventions

Certain keywords appearing at the top level of a capabilities string are defined to have standard meanings within the ACCESS.bus protocol as follows:

Keyword	**Meaning**
prot()	The "prot()" entry identifies the generic protocol or device type to system software. prot() is the device driver's view of a device. prot() defines the behavior of a device in terms of commands and responses in addition to those defined in the Base Protocol.
prot(keyb)	generic keyboard
prot(locator)	generic locator

prot(text)	generic text
type()	The "type()" entry is intended to identify the device type to the user in a recognizable form. type() is a user's view of a device. That is, a joystick, mouse, or whatever. It is also a second level identifier of the device used by the system software.
type(keyboard)	Keyboard
type(mouse)	Mouse
type(digitizer)	Digitizing tablet
type(tball)	Trackball
type(ptrstick)	Force activated joystick (typically embedded in keyboards)
type(touchscn)	Touchscreen
type(dial)	Dials, arrays of dials, and other single axis valuators
type(swpad)	Switch pads, such as those used for game control, where a set of switches are used to control position and functions. (as opposed to keyboard)
model()	The "model()" entry is provided to present the full model name to the user if different from the module name. The "model()" entry is optional. It is also a third level identifier of the device used by the system software.
pwr( )	Power management capability string, see Section 2.11.3
bwm()	Bandwidth management capability string, see Section 2.12.1

The following capability usage convention is recommended to maximize compatibility between hardware and software from different vendors.

1. Tags used as keywords to identify device features should be no more than eight characters in length (only the first eight characters are significant). Characters A-Z, a-z, and 0-9 are assumed to be handled transparently on all systems.

2. Device features or events that are represented by bit positions should be numbered sequentially starting with "1" for the least significant bit (bit 0). For example, locator buttons within a 16 bit keyswitch word might be identified as "buttons(1(L)2(R)3(M))". 1 is related to bit 0, 2 is related to bit 1, and 3 is related to bit 2.

Example: The Keyswitch word represents the possible 16 functional buttons of a locator.

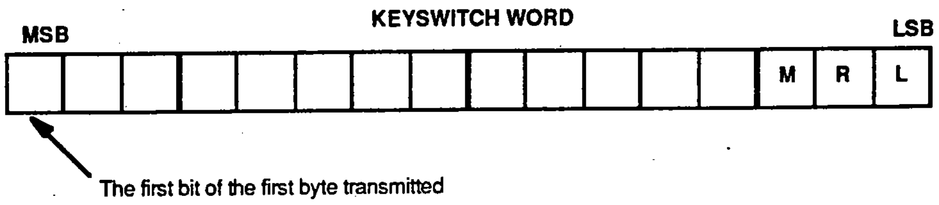

Figure 2.4: Keyswitch Word

3. Features controlled by a numeric value parameter can be described by the feature name as a tag followed by the parameter range. If the range is from zero to maximum, the zero minimum value can be omitted. Keyclick volume ranging from 0 (off) to 7 (maximum) can be described as "click(7)" for example. It is suggested that these values not be normalized. By not normalizing it is possible to preserve the full resolution of the

device, thus allowing the computer to determine the smallest meaningful volume increment.

Specific additional capabilities are defined in generic device protocol specifications. Generic specifications for keyboards, locators, and text devices have been developed to date.

2.6.3 Binary Data in Capabilities Strings

To include binary information in the Capabilities string a new keyword bin is defined. The binary data within the Capabilities string has the following format:

```
bin(count(binary data bytes))
```

where count is an integer count of the number of binary data bytes.

2.7 Configuration Process

The configuration process shall be used to detect the devices that are present on the bus, assign each device a unique address, and connect devices to the appropriate software driver. Configuration shall occur at system start-up, or at any time when the computer detects the addition or removal of a device.

2.7.1 Power-up/Reset Phase

When reset or powered-up, a device shall always revert to the default address and send an Attention message to alert the computer to its presence. At system start-up or reinitialization, the computer shall send a Reset message to all ACCESS.bus addresses in the ACCESS.bus device address range to insure that all devices on the bus respond at the power up default address.

2.7.2 Identification Phase

To begin address assignment, the computer sends an Identification Request message at the device default address. Every device at this address must then respond with an Identification Reply message. As each device sends its message, the ACCESS.bus physical layer arbitration mechanism automatically separates the messages based on the identification strings. The computer can then assign an address to each device by including the matching identification string in the Assign Address message. A device that receives this message and finds a complete match with the identification string moves its device address to the new assigned value. As soon as a device has a unique address, it shall change to 'on line state' in a 'disable mode'. in this state the device is waiting to receive an "Enable Application Report" control message from the host to start sending its application reports.

The ACCESS.bus physical layer bus protocol allows multiple devices on the bus at the same time, if those devices are transmitting exactly the same message. In the rare event that two like devices report the same random number or are mistakenly assigned to the same address, each interactive device transmits a Reset message to its assigned address immediately prior to sending its first data message after being assigned a new address. The self-addressed Reset message forces other devices at the same address back to the power-up default address, as if they had just been hot-plugged. The message guarantees that each device has a unique address, but not until the device is actually used. The pseudo random number (or serial number, if available) distinguishes devices at identification time before they are used, allowing the computer to inventory which devices are present.

2.7.3 Capabilities Phase

After assigning a unique address to a device, the computer retrieves the device's capabilities string as a series of fragments using the Capabilities Request and Capabilities Reply messages. The computer then parses the capabilities string to choose the appropriate application driver for the device. The parsed string shall also be made available to application programs using the device.

2.7.4 Normal Operation

During normal operation, the ACCESS.bus manager periodically checks the presence of all devices on the bus. If a device is found to be missing, the bus manager will notify the device driver with a Device Disconnected message and will update the device table.

2.8 Timing Rules

2.8.1 Bus Timing

Since the ACCESS.bus may be shared by multiple peripheral devices, it is important to assign the bus bandwidth in such a way that every device will be able to send (or receive) its messages on time. Assuming that the bus bandwidth is sufficient to serve all the devices, we need a mechanism that will control the time duration and the rate that each device occupies the bus. This mechanism is the Bandwidth Management that is described later in this section. Since the Bandwidth Management is optional it is important that each device will obey the following timing rules:

1. A device must allow at least fifty microseconds (50 microseconds) between releasing bus mastership at the end of a message and requesting to become bus master again. This is to give other devices a chance to access the bus without arbitration.

2. ACCESS.bus interfaces shall not hold SCL low for more than two milliseconds (2ms). A watchdog timer or other provision shall be implemented by each device to assure it releases SCL before the two millisecond (2ms) limit is reached.

2.8.2 Response Timing

Time limits for certain commands to execute are specified so that the computer can determine when all the devices present have had sufficient time to respond (time out). The following limits apply:

1. Devices shall complete the Reset command or the power on reset within 250ms. This is believed to be long enough for basic power up self test (from stable power) without causing excessive delays. All devices should attempt to minimize this time.

2. Devices shall respond to all other commands that require a response within forty milliseconds (40ms).

3. If a command can be responded to by more than one device, the time limit shall be extended to forty milliseconds (40ms) since the last device that responded.

2.8.3 Minimum Device Performance

Each device connected to the ACCESS.bus, including the host, can affect the overall bus performance. When a device is transmitting as a bus master if the device is too slow it occupies the bus for a longer time than is necessary and reduces the overall bus performance. Second, as a listener, if a device is too slow, it slows the transmitting speed of the master device which also results in lower bus performance.

To guarantee good ACCESS.bus performance all ACCESS.bus devices should comply with the following requirements.

1. The host interface acting as master transmitter shall transmit at a minimum data rate of 8 Kbyte/sec (assuming that the receiver does not stretch the clock during the message).

2. The host interface acting as a slave receiver shall not slow down the data rate to less than 8 Kbyte/sec (assuming that the transmitter is faster than 8 Kbyte/sec).

3. Devices that in nominal bandwidth mode require 50% to 100% of the bus time shall transmit at a minimum rate of 8 Kbyte/sec (assuming that the receiver does not stretch the clock during the message) and as a receiver shall not slow down the data rate to less than 8 Kbyte/sec (assuming that the transmitter is faster than 8 Kbyte/sec).

4. Devices that in nominal bandwidth mode require 25% to 50% of the bus time shall transmit at a minimum rate of 7.5 Kbyte/sec (assuming that the receiver does not stretch the clock during the message) and as a receiver shall not slow down the data rate to less than 7.5 Kbyte/sec (assuming that the transmitter is faster than 7.5 Kbyte/sec).

5. Devices that in nominal bandwidth mode require 10% to 25% of the bus time shall transmit at a minimum rate of 7 Kbyte/sec (assuming that the receiver does not stretch the clock during the message) and as a receiver shall not slow down the data rate to less than 7 Kbyte/sec (assuming that the transmitter is faster than 7 Kbyte/sec).

6. Devices that in nominal bandwidth mode require 0% to 10% of the bus time shall transmit at a minimum rate of 6 Kbyte/sec (assuming that the receiver does not stretch the clock during the message) and as a receiver shall not slow down the data rate to less than 6 Kbyte/sec (assuming that the transmitter is faster than 6 Kbyte/sec).

7. A peripheral device or host interface acting as slave receiver shall ignore any data on the bus, therefore not affect the transmission data rate, if the destination address of the message does not match the device's address. The peripheral device or host interface shall start listening to the bus again after they have detected the previous message Stop condition.

2.9 Exception Handling

The following general requirements and recommendations are defined for handling ACCESS.bus exception conditions. Additional requirements may be defined by the individual device protocol specifications.

1. Arbitration Loss - If a device detects that it has lost arbitration, the device shall cease transmitting and then try again to become bus master to resend the message. If the device was trying to send an Attention, Identification Reply or Capabilities Reply message then the device should get bus ownership as soon as possible, and try to send the message again. The device should try as many times as necessary until the message is received by the host.

   If the message is an application report, it is the device's decision whether to discard the message, or to re-send the message. In general, devices should retry as many times as necessary to win arbitration and send their message.

2. Negative Acknowledge - If a device transmits a byte which is negatively acknowledged (no receiver, or rejected for some reason), or if the bus times out, the device shall abort the transfer immediately by generating a Stop condition. If the device was trying to send an Attention, Identification Reply or Capabilities Reply message then the device should get bus ownership as soon as possible, and try to send the message again. The device should try as many times as necessary until the message is received by the host.

If the message is an application report, it is the device's decision whether to discard the message, or to re-send the message.

3. Checksum Error - If a device detects a checksum error in a received message, the message data shall be ignored.

4. Premature STOP - If a device detects a premature STOP signal before the end of a message is reached, the message data shall be ignored.

5. If the computer detects a checksum error or premature STOP condition, it is suggested the computer log the error and re-issue its most recent request to that device. A large number of interface errors may indicate a software error or faulty hardware configuration. In this case, it is suggested the computer notify the user and attempt to restart the configuration process.

6. Repeated STOP - A device should not respond to any event of a repeated STOP signal.

7. Unrecognized Commands and Parameters - If a device receives a command with valid checksum but does not recognize the command op-code, the entire command shall be ignored. If a device receives a command with valid checksum but does not recognize the value of a required parameter, the entire command shall be ignored. If a device receives a command with valid checksum that includes additional data beyond that expected, it is recommended the additional data be ignored, however, the device may execute the command.

8. A device will never hang up the bus by pulling the SCL or the SDA lines to low level for more than 2msec. Every device has to have a watchdog mechanism to release the bus after the 2 msec period.

9. A device must respond to the default address (6EH) before it is assigned an address by the host, and to its assigned address afterwards. As an exception, a device can negatively acknowledge the message when the device is busy with internal data processing. A well behaved device should minimize the number of these situations.

2.10 Detailed Command/Message Descriptions

2.10.1 Command Coding

For the purpose of illustrating message encoding, the following notation is used:

PLLLLLLL		'P' is a 1-bit protocol flag: P=0 denotes a data stream; P=1 denotes a control/status message LLLLLLL is a 7-bit length field that encodes values 0-127
ddddddd0	=	7-bit destination ACCESS.bus address plus 0 as the LSB
sssssss0	=	7-bit source ACCESS.bus address plus 0 as the LSB
xxxxxxxx	=	8-bit message op-code
cccccccc	=	8-bit checksum

Numeric values are always transmitted MSB first.

ACCESS.bus messages are either Device Data Stream (P=0), or Control /Status (P=1). Data Stream messages always refer to the application part of a device, as opposed to the interface part. The coding of Data Stream messages is dependent on the device. They are not pre-defined or restricted by the ACCESS.bus protocol.

```
Data Stream Msg          Control/Status Msg
ddddddd0                   ddddddd0
sssssss0                   sssssss0
0LLLLLLL                   1LLLLLLL
|                          xxxxxxxx  op code
body[0-127]                |
|                          body[0-126]
cccccccc                   |
                           cccccccc
```

The first data byte of Control/Status messages (P=1) shall always be an op-code. Control/Status messages may refer to either the application part or the interface part of a device. Pre-defined Control/Status op-codes (messages) are used to initialize and configure ACCESS.bus devices. A range of Control/Status op-codes are reserved for private use by devices and will not be pre-defined as part of the ACCESS.bus protocol.

The breakdown of ACCESS.bus message types is shown in the following diagram:

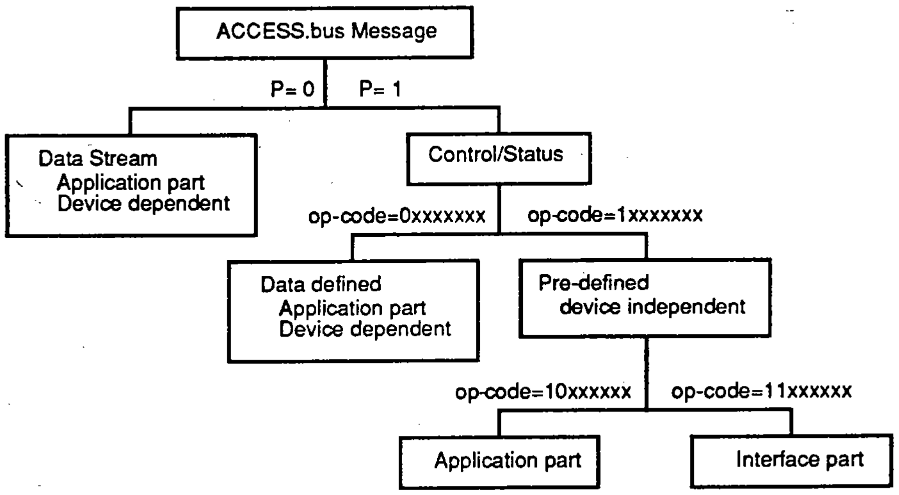

Figure 2.5: ACCESS.bus Message Types Breakdown

2.10.2 Device Data Stream Message

These messages are usually used for the bulk of the device's data. The intent is that the most common messages (keypress reports, locator movements, etc.) should be easiest to send and have the lowest overhead. Device Data Stream messages are distinguished by the Protocol flag bit being zero. The body of these messages are defined by the individual device protocol specifications.

Devices are not permitted to send device data stream messages at the power on Default Address. After a successful Assign Address command, devices shall change into an on line state in a disabled mode. After receiving an Enable Application Report command the device should send a self addressed reset message immediately prior to transmission of its first application report.

2.10.3 Device Defined Control/Status (C/S) Messages

Device control or status information shall be sent with the protocol flag set to one (P=1). Op-codes in the range 00h to 7Fh are reserved for device defined peripheral messages (application part). Refer to the individual device protocol specifications for details.

2.10.4 Pre-defined C/S Messages, Interface Part

Interface Part Control/Status messages are used to control the ACCESS.bus itself.

2.10.4.1 Reset

A Reset message shall be used to instruct the addressed device(s) to reset to the power up state.

Format:

```
ddddddd0
sssssss0
10000001        (81h, P=1, length=1)
I
11110000        (F0h, Reset op-code)
I
cccccccc
```

This command is intended to be completely equivalent to power on reset including changing of device address to the Default Address, power on self testing, and transmission of an Attention message at the Default Address.

Notes:

1. The device shall ignore all commands from the computer until it has completed power-up processing and transmitted a successful Attention message. This is the only command which may cause a device to temporarily ignore bus messages.

2. A device must wait at least eight milliseconds (8ms) after its ACCESS.bus hardware is reset before transmitting to insure its ACCESS.bus hardware has synchronized with any message frame in progress.

3. The Reset command must be completed within 250ms (excluding the time spent waiting to become bus master).

4. Because the ACCESS.bus does not have a General Call or Broadcast address, the ACCESS.bus shall be reset by issuing Reset commands to all 125 standard ACCESS.bus device addresses.

2.10.4.2 Attention

An Attention message shall be sent by a device to the computer. This message notifies the computer that a device needs attention after power up or reset.

Format:

```
ddddddd0
sssssss0
10000001        (81h, P=1, length=1)
I
11100000        (E0h, Attention op-code)
I
cccccccc
```

Notes:

1. This message must be transmitted within 250ms after a reset command or stable power is applied (excluding time spent waiting to become bus master).
2. A device must wait at least eight milliseconds (8ms) after its ACCESS.bus hardware is reset before transmitting to insure its ACCESS.bus hardware has synchronized with any message frame in progress.
3. The device must ignore all commands from the computer until it has completed power-up processing and transmitted a successful Attention message. Devices that fail must not respond to any commands.
4. Because Attention is only generated after power on or reset, it will always be transmitted at the Default Address. Receiving an Attention message shall indicate to the computer that a device is present on the bus and awaiting configuration.

2.10.4.3 Identification Request

An Identification Request message shall instruct the addressed device(s) to send a complete identification report (see Section 2.10.4.4 - Identification Reply).

Format:

```
ddddddd0
sssssss0
10000001      (81h, P=1, length=1)
|
11110001      (F1h, Identification Request op-code)
|
cccccccc
```

The Identification Request message will cause all devices at the Default Address to send a unique identification string so that the computer can assign each device to a distinct address.

2.10.4.4 Identification Reply

The Identification Reply message shall be issued by a device in reply to an Identification Request message. The reply shall consist of a string that identifies the device hardware. A four byte pseudo random number (or serial number if available) shall be included in order to distinguish like hardware devices.

Format:

```
ddddddd0
sssssss0
10011101              (P=1, length=29)
|
11100001              (E1h, Identification Reply op-code)
protocol revision     (1 byte, "B")
module revision       (7 bytes, e.g., "V1.0   ")
vendor name           (8 bytes, e.g., "DEC     ")
module name           (8 bytes, e.g., "LK501   ")
device number         (32 bit signed integer)
|
cccccccc
```

The module revision, vendor name, and module name shall be left justified ASCII strings padded with the space character (20h). See "Random versus Serial Device Numbers" for a description of the device number.

2.10.4.5 Assign Address

An Assign Address message shall be issued by the computer to instruct the addressed device(s) with matching identification strings to move to the specified device address.

Format:

```
dddddddd0
sssssss0
10011110            (P=1, length=30)
|
11110010            (F2h, Assign Address op-code)
protocol revision   (1 byte, "B")
module revision     (7 bytes, e.g., "V1.0   ")
vendor name         (8 bytes, e.g., "DEC     ")
module name         (8 bytes, e.g., "LK501   ")
device number       (32 bit signed integer)
new A.b address     (1 byte)
|
cccccccc
```

When a device receives this command and finds a complete match in the protocol revision, module revision, vendor name, module name, and device number field it moves its address to the new assigned value. If the device is successfully assigned a new address, it shall be able to respond to messages at the new address immediately, that is, on the next valid ACCESS.bus message frame.

If the identify information does not match that of the receiving device, the entire message shall be ignored.

Devices shall not transmit user data while at the Default Address. If the Assign Address is successful, the device change to an on line state waiting for Enable Application Report message to start sending its application reports. Once an Enable Application Report message is received, the device shall send a self addressed reset command on its new address immediately prior to sending its first data report.

2.10.4.6 Capabilities Request

A Capabilities Request message shall be issued by the computer to a device to instruct the addressed device to reply with a Capabilities Reply. The Capabilities Reply shall contain data starting at "offset".

Format:

```
dddddddd0
sssssss0
10000011            (P=1, length=3)
|
11110011            (F3h, Capabilities Request op-code)
offset              (16-bit unsigned integer)
|
cccccccc
```

The Capabilities Request and Capabilities Reply messages form a protocol for transferring an arbitrary byte-string from the device to the computer, via a series of fragments. "Offset" shall be the index (from 0) into this string. To simplify the device's implementation of this protocol, "offset" shall be restricted to three values:

"send first" zero, indicating the computer wants to start over at the beginning;

"send again" the offset from the most recently transmitted Capabilities Request, indicating the computer did not receive a response and wants a retransmit;

"send next" the offset from the most recently received Capabilities Reply plus the number of bytes in the message fragment. (new offset = old offset + fragment length) (fragment length = message length - 3).

With these restrictions the computer can make three requests: start over, send current, and send next. See Capabilities Reply message for further details.

2.10.4.7 Capabilities Reply

A Capabilities Reply shall be used to reply to a Capabilities Request message with a fragment of data starting at "offset".

Format:

```
ddddddd0
sssssss0
10LLLLLL             (P=1, length=3-35)
|
11100011             (E3h, Capabilities Reply op-code)
offset               (16-bit unsigned integer)
data                 (0-32 bytes)
|
cccccccc
```

The protocol is designed to be simple for the device to implement: The device is free to choose the most convenient fragment size from one message to the next.

The only state information the device should need to maintain is the current offset and length of the most recently transmitted fragment.

On receiving a Capabilities Request message, the device shall examine the "offset" field:

- If equal to zero, the device shall set the current offset to zero and send the fragment from offset zero (0).
- If equal to the current offset, the device shall re-send the fragment from the current offset.
- If equal to the "current offset" + "fragment length", the device shall update the current offset (current offset := current offset + fragment length) and then look up (or calculates) the next fragment to send and sends it.
- If the device has reached end-of-string, it shall send a fragment with the next offset but zero data bytes. This will indicate an end of string.
- Otherwise, the device shall set the "current offset" to zero and send the fragment from offset 0.

2.10.4.8 Enable Application Report

An Enable Application report command shall be used to instruct an on line device to start (Enable) or to stop (Disable) sending application reports.

Format:

```
ddddddd0
sssssss0
10000010     (82h, P=1, length=2)
|
11110101     (F5h, Enable Application Report op-code)
000000XX     (XX=00 for disable, XX=01 for enable)
cccccccc
```

Device in the disable mode shall not send application reports.

2.10.4.9 Presence Check

This command is used by the host to check if a device is connected to the bus and responding to its ACCESS.bus address. This message requires no response from the device. The host uses the device acknowledge bits in each byte (see section 1) as an indication for the device presence.

Format:

```
ddddddd0
sssssss0
10000010           (82h, P=1, length=2)
|
11110111           (F7h, Presence Check op-code)
00000000           (for future use)
cccccccc
```

2.10.4.10 Resource Request (Optional)

A Resource Request shall be sent by a device to request a resource from the computer. The Resource Request command is optional, but is used by the Power Management and Bandwidth Management commands.

Format:

```
ddddddd0
sssssss0
100xxxxx     (P=1, length=n)
|
11100101     (E5h, Resource Request op-code)
res code     (resource designator)
data         (optional data)
|
cccccccc
```

Table 2.2 lists supported resource codes.

Table 2.2: Supported Resource Codes

Resource Code (hex)	Description
01	Request ACCESS.bus address for private use. Data byte 1, if specified, shall be the desired ACCESS.bus address.
02	Relinquish ACCESS.bus address. Data byte 1 shall be the ACCESS.bus address to be relinquished.
03	Request time and date. No additional data provided.
04	Write data request. The host saves a block of data for the device. See section 2.11.4, Power Management, for additional information.
05	Read data request. The host sends the block of stored data back to the device. See section 2.11.4, Power Management, for additional information.
06	Request continued bus power. The device requests that the host continue to provide power to it so that it can complete an operation in progress. See section 2.11.4, Power Management, for additional information.
10	Request bandwidth from the host. The device requests permission to use additional bus bandwidth. See section 2.12.4, Bandwidth Management, for additional information.

2.10.4.11 Resource Grant (Optional)

The Resource Grant command shall be sent by the computer to a device to indicate that a requested resource has been granted to the requesting device.

Format:

```
ddddddd0
sssssss0
100xxxxx          (P=1, length=n)
|
11110100          (F4h, Resource Grant op-code)
res code          (resource designator)
status            (0=success, 1=unsupported,
                  2=failed- try again later, 4=failed)
data              (optional data)
|
cccccccc
```

Table 2.3 lists supported resource codes.

Table 2.3: Supported Resource Codes

Resource Code (hex)	Description
01	ACCESS.bus address granted for private use. Data byte 2 shall be the ACCESS.bus address
02	Response to relinquish ACCESS.bus address. Data byte 2 shall be the ACCESS.bus address relinquished.
03	Current time and date. Data bytes 2,3 = year Data byte 4 = month Data byte 5 = day of month Data byte 6 = hour Data byte 7 = minute Data byte 8 = second Data byte 9 = .01 second

2.10.4.12 Vendor Reserved Commands (Optional, Interface Part)

A small range of op-codes are reserved for vendors to invoke private functions for testing, alignment, or set-up at manufacturing time. These commands might be used to down load an EEPROM with a serial number for example. These commands are device dependent and should not be used during normal ACCESS.bus operation. If these commands invoke any private modes, it is recommended that such modes be exited by the Reset command (op-code F0h).

Op-codes C0 to C8 hex are reserved for vendor specific use.

2.10.5 Pre-defined C/S Messages, Application Part

2.10.5.1 Application Hardware Signal (Optional)

The Application Hardware Signal shall be sent by a device to the computer. It shall be used to instruct the computer ACCESS.bus controller to generate a high priority hardware signal.

Format:

```
ddddddd0
sssssss0
10000010        (P=1, length=2)
|
10100000        (A0h, Application Hardware Signal op-code)
sigcode         (signal code 1=reset, 2=halt, 3=attention)
|
cccccccc
```

The action taken by the computer on receiving this messages shall be dependent on the computer. The intent is that the computer's ACCESS.bus controller physically interrupt the computer.

Table 2.3: Application Hardware Signals

Number	Name	Purpose
1	Reset	Attempt soft reset of host computer. If no response within pre-set time limit, activate hardware reset of host computer
2	Halt	Primarily a debugging tool; could cause exit to a ROM debugger
3	Attention	Gain the attention of the computer

This command is optional for both the host computer and for the devices.

2.10.5.2 Application Test

An Application Test command shall instruct the addressed device to reply with the results of a self-test specific to the device.

Format:

```
ddddddd0
sssssss0
10000001    (P=1, length=1)
|
10110001    (B1h, Application Test op-code)
|
cccccccc
```

This command shall be used by the Application driver to test the Application Part of the device. When the command is issued, the device has already been configured so the basic ACCESS.bus functions are assumed to be working.

Unlike power up testing, the device must respond promptly and may not continue testing until the test succeeds; the device may not ignore other bus commands while testing; and the testing shall not alter the application state. If no application specific testing is required, it is recommended that the device report the previous test results gathered during power up testing.

If useful in a particular application, the device may return a new test report each time the Application Test command is received. The device's Application driver may then issue Application Test commands until the device signals success.

2.10.5.3 Application Test Reply

An Application Test Reply shall reply to an Application Test message with a self-test report specific to the device.

Format:

```
ddddddd0
sssssss0
1LLLLLLL    (P=1, length=2-32)
|
10100001    (A1h, Application Test Reply op-code)
status      (0=passed, non-zero=failure)
body        (0-30 bytes of additional test information)
|
cccccccc
```

The only restriction on the message is that the first byte of the body (status) be zero if the test succeeded, and non-zero otherwise.

The computer may respond to a failed test report by either ignoring the message (and the device), or printing the body of the message, perhaps in both hexadecimal and ASCII (if printable).

2.10.5.4 Application Status Message

Devices may report changes in their status, or various error conditions to the host computer by sending an Application Status Message. This message is intended to be used for reporting information to device drivers about changes in the application specific portion of the device. It includes both pre-defined and private fields. The private fields are ignored by generic device drivers but may be used to encode vendor status information for use by vendor supplied device drivers.

Application Status Message - A2h 00h (optional)

Format:

```
ddddddd0
sssssss0
10000101         (85h, P=1, length=5)
10100010         (A2h, Application Status Message)
00000000         (00h, Second byte of Application Status Message Command)
ssssssss         (Status Code)
yyyyyyyy         (Device specific data, set to 0 if unused)
yyyyyyyy
cccccccc
```

Device to Host 50 XX 85 A2 00 SS YY YY CS

Where: A2h 00h is the Application Status Message Command.
SS is the status code
- 00 - Device ready
- 01 - Device not ready
- 02 - Device capabilities have changed
- 03 - Device has lost its internal state, most likely due to a failed state restore operation after a low power suspend or power off mode.
- 04 - Device has lost applications data, most likely due to an internal data buffer overrun.
- 05 - Reserved for vendor use, suggested for status changes.
- 06 - Reserved for vendor use, suggested for errors.

YY YY are two bytes of vendor specific data. Devices that do not use these fields should return 00 00 in them.

2.11 Device Power Management (Optional)

The Device Power Management command shall be sent by the computer to a device to request that the device change its operating mode in order to control power usage by the device, or to notify a device of the computer's intentions to turn off the ACCESS.bus power supply. All of the Device Power Management (DPM) commands are advisory, and a device may continue to operate in the mode it is in as necessary to complete its tasks. As soon as possible, the device should attempt to switch to the operating mode requested by the computer. Devices may also make internal decisions with respect to which operating mode they should be in to control power usage, and they may transition to a low power level without specific direction from the host computer.

Devices which support power management should power up in the lowest power operating mode possible.

There are five operating modes supported by the DPM commands, each requiring successively less power:

1. Run. In this mode the device is operating at full power, and is either in operation or ready for immediate use.

2. Standby. In standby mode the device has reduced its power consumption as much as is possible while still being able to respond to applications data reports or operations without undue delay or restart power consumption. For example, a laser printer would shut off its toner fusing heater and imaging laser, but would probably leave its image rasterizing processor and memory on.

3. Suspend. In suspend mode the device should reduce its power consumption to the lowest possible level. Only Interface Part Control/Status messages will be sent to the device by the computer while suspend mode is in effect. The device may send applications messages to the computer, allowing the possible use of keyboard, mouse, or other peripheral events to end a system wide suspend mode, however the computer may elect to ignore applications messages received while it is in suspend mode.

4. Shutdown. Shutdown mode is equivalent to suspend mode except that the device may not initiate a transition to a state which requires higher ACCESS.bus power on its own. (A device in suspend mode could do so in response to some external event.) Shutdown mode will be used by the host for ACCESS.bus power supply load shedding. In shutdown mode the device should reduce its power consumption to the lowest possible level. Only Interface Part Control/Status messages will be sent to the device by the computer while shutdown mode is in effect. A device which has received a shutdown mode command shall remain in shutdown mode until it receives another DPM command.

5. Power off. In each of the three operating modes the device's bus interface is still active, and the device can respond to commands from the computer. In this mode the bus power supply is turned off, causing all bus powered devices to be turned off. During an orderly shutdown the computer will issue the Power Off DPM command to all DPM capable devices and then wait for 100 ms after the last ACCESS.bus packet before shutting off the bus power supply. Devices which require continued power to complete a task in process can request that the computer leave the power on by using a Resource Request for power during this interval. The computer will attempt to comply with the request, but may be unable to.

Devices entering Suspend, Shutdown or Power Off mode must, if at all possible, save their internal state information so that they can resume operation transparently after they return to Standby or Run mode. There are many ways to do this including local non volatile storage, a device power supply separate from the ACCESS.bus, an ACCESS.bus power supply separate from the host computer, and temporary storage of device state information by the host computer. This last approach is supported by Resource Requests to read and write device data, and additional DPM command, Restart, which is used to notify a device that has been powered off that saved state information is available for it.

Devices that have saved their internal state in response to DPM Suspend or Shutdown command using a Resource Write Data Request should restore their state when they receive a DPM Restate command By using a Resource Read Data Request.

Devices that save their internal state in response to a DPM Power Off command using a Resource Write Data Request should restore their state when they receive a DPM Restart command by using a Resource Read Data Request. It is possible that the device's attempt to restore its state will fail. In this case the device should retain its addressing and other bus interface related state and restore the rest of its state to the condition it was in prior to attempting to restore its state. The device should then notify its driver using the Application Status Message.

The host computer can query a device for its operating mode and power consumption by sending the DPM Query power mode command. The device will respond with a Device Power Usage Reply which includes the operating mode the device is in and optionally the devices actual or estimated power consumption.

2.11.1 Device Power Management Command

Host to Device XX 50 82 F6 {00,01,02,03,04,05,06} CS

Where:
F6 - Device Power Management Command
00 - Run Mode
01 - Standby mode
02 - Suspend mode
03 - Shutdown mode
05 - Restart
04 - Power off advisory
06 - Query power mode

Devices which only implement a subset of the Device Power Management Command should simply ignore unimplemented commands, or treat unsupported modes as equivalent to some supported mode. Thus, a keyboard might treat Run mode and Standby mode commands identically.

2.11.2 Device Power Usage Reply

Device to Host 50 XX 87 E6 00 OM BH BL LH LL CS

Where: OM is the operating mode
00 - Run mode
01 - Standby mode
02 - Suspend mode
03 - Shutdown mode
04 - ready for Power off

BHBL is a 16 bit integer representing the ACCESS.bus power usage in .01 watt units. A value of FFFFh indicates unknown power usage.

LHLL is a 16 bit integer representing the line power usage in .01 watt units. A value of FFFFh indicates unknown power usage.

2.11.3 Power Management Capabilities String

Devices which support the Device Power Management command shall include as a part of their capabilities string a tag pwr, and if possible the power capabilities string specified below. The DPM capabilities must follow immediately after the prot(), type(), and model() declaration

```
prot()
type()
model()
pwr( {run( ) stdby( ) susp( ) shut( ) ssave( ) psave( ) } )
```

where the { } brackets indicate optional items, and:

pwr( ) indicates a device that supports DPM commands but whose support for specific modes is undisclosed and whose power consumption in different modes in unknown.

run, stdby indicate that the device supports the named mod, and if susp and numeric values are included what the power usage is for that mode.

ssave(#) indicates that the device can save its state across a suspend and if it requires use of host computer resources approximately how many bytes of data. Zero as an amount indicates that the actual number of bytes required is unknown. No number indicates that the device does not require the use of host resources.

psave(#) indicates that the device can save its state across a power off mode and if it requires use of host computer resources approximately how many bytes of data. Zero as an amount indicates that the actual number of bytes required is unknown. No number indicates that the device does not require the use of host resources.

The format of the power usage specifier is:

run({B}# {L#})

where power usage from the ACCESS.bus power supply is specified by either a number with no prefix, or the prefix B followed by a number, and power usage from another source is specified by the prefix L followed by a number. In each case the number represents the approximate power consumption of the device in .01 watt units.

For example, the power management capabilities string for a laser printer which uses line power for its toner fusing heaters and rasterizer, and the ACCESS.bus power supply only for its ACCESS.bus interface might be:

pwr(run(B10 L 10000) stdby(B10 L 10000) susp(B10 L2) ssave( ) psave( ))

This example device doesn't require any storage from the host computer to save its state since it keeps a small amount of memory active using line power even if the ACCESS.bus power is turned off.

For a keyboard that uses state save to save the state of its indicator lights, the power management capabilities string might be:

pwr(run(15) susp(3) ssave( ) psave(1))

In this example the keyboard only requires storage from the host computer when it will be totally powered off. Otherwise its local Microcontroller can save the state of the indicator lights even though they are turned off during suspend mode.

Devices do not need to fully disclose their power management capabilities in the pwr string in order to receive DPM commands from the host computer, however devices should disclose their state saving ability whenever possible.

2.11.4 Power Management Resource Request command

Resource Write Data Request - resource code 04

Device to Host 50 XX 100xxxxx E5 04 data CS

where the third byte contains a five bit count of the number of bytes to write, and the data bytes are private binary data.

The host computer stores the data with a tag identifying the device which saved the data and the sequence of writing the data. Resource Read Data Requests are fulfilled in first in first out order.

Resource Read Data Request - resource code 05

Device to Host 50 XX 82 E5 05 CS

Resource Power Request - resource code 06

Device to Host 50 XX 82 E5 06 CS

This request is made by devices that have received a DPM Power Off advisory and need additional time to complete the task they are doing, or otherwise want to continue operation. There is no guarantee that the host computer will grant this request to continue providing ACCESS.bus power. If the host computer cannot grant the request it will respond with a Response Grant with the status code set to failed.

2.11.5 Power Management Resource Grant command

The response provided by the Resource Grant command is unchanged for the Resource Write Data Request, and the status byte reflects the action taken by the host. Note that a host computer which has run out of storage it can use for this purpose would return a status of either failed - try again later or failed.

The response provided by the Resource Grant command consists of the data from the oldest successful Resource Data Write Request for the requesting device. (Note that the host computer must perform the necessary mapping between bus addresses and device identification if the assignment of bus address to devices has changed since the Resource Data Write Request.)

Host to Device XX 50 100xxxxx F4 data CS

Where the third byte contains a five bit count of the number of bytes read, and the data bytes are private binary data.

In the event that there are no data records available to the device, the status will be set to failed. Data records which are not read before the second power shutdown after the data record was written are discarded by the host.

Devices using the Resource Read Data Request to restore their internal state should include information in the data they save which will allow them to verify the correctness and completeness of the data they receive. Devices which attempt to reset their state and are unsuccessful should revert to the state they were in prior to attempting to reset their state and send an Application Status Message to the computer from the assigned address.

The response provided by the Resource Grant command is unchanged for the Resource Power Request, and the status byte reflects the action taken by the host. A host computer which cannot or will not continue to provide ACCESS.bus power will return a status of failed.

2.11.6 Power Management Status Message

The Application status message is used by devices to report a failure of a state restore operation after a power down/restore sequence.

It is recommended that devices which rely on the Resource Write Data and Read Data request to store internal state information use this message to report failures which occur during state restorations. Device drivers which receive this message may be able to reset the device, restore its state and continue with operations.

2.12. Device Bandwidth Management (Optional)

2.12.1 Bandwidth Management Capabilities String

To put in place the foundation for software Bandwidth Management mechanism - BWM, each ACCESS.bus device will report to the host, as part of its capabilities string, its requirements for bus bandwidth.

Devices which support the Bandwidth Management mechanism - BWM command shall include as a part of their capabilities string a tag bwm, and if possible the bandwidth capabilities string specified below. The BWM capabilities has to be immediately after the DPM (Device Power Management) capabilities

```
prot()
type()
model()
pwr()
```

bwm({mlength(maximum nominal minimum) mwtmaximum nominal minimum)})

bwm is the keyword for bandwidth management group of parameters.

mlength - stands for message length. The maximum, nominal, and minimum attributes are the maximum, nominal, and minimum of the number of bytes in the device application message. The message length includes all the message bytes (destination and source addresses, message length, data, and checksum) in hexadecimal.

mwt - stands for "message wait time". The mwt is the minimum waiting time between the end of the current message, and the beginning of the next message. This time is calculated as the following:

mwt = decimal value of (mwtH mwtL} x 100sec [seconds]

The maximum, nominal, and minimum attributes are the maximum, nominal, and minimum values of the device wait time.

Examples:

A mouse reports

bwm({mlength(0A 0A 0A) mwt00122 0064 005A)})

A keyboard reports

bwm({mlength(0E 05 05) mwt2710 1388 07D0)})

When a device is connected to the bus, it will set itself to the nominal bandwidth mode. The Bandwidth Management mechanism - BWM will get the bandwidth attributes of all the devices, and will instruct each one of the devices in which mode to stay. This decision can also be manually controlled by the user via an ACCESS.bus control panel. The BWM will alert the user if there are too many devices connected to the bus and will advise him to disconnect or disable a device / devices (Enable Application Reports).

2.12.2 Device Bandwidth Management Command

This command is used by the host to send the device its operating bandwidth parameters.
Host to Device

Format:

XX 50 84 F8 YY mlength mwtH mwtL CS

Where: F8 - Device Bandwidth Management command
YY = 00 instruction to the device to switch to the specified bandwidth mode.
YY = 01 request to the device to report its current bandwidth mode
(Device Bandwidth Usage Reply)
mlength - Maximum message length - hexadecimal
mwtH, mwtL - Two bytes of the Maximum messages rate (messages per second) - hexadecimal

2.12.3 Device Bandwidth Usage Reply

This command is used by the device to tell the host the device operating bandwidth parameters. This command is sent to the host to acknowledge the acceptance of Device Bandwidth Management command (YY=00), or as a reply to host bandwidth inquiry (YY=01)

Device to Host
Format:

50 XX 84 E8 mlength mwtH mwtL CS

Where: E8 - Device Bandwidth Usage Reply
mlength - Maximum message length - hexadecimal
mwtH mwtL - Two bytes of the Maximum messages rate (messages per second) - hexadecimal

2.12.4 Bandwidth Resource Request command

When a device finds that it needs more bus bandwidth (the device loses too many messages), the device will send a BWM resource request message to request more bus time.

Device to host:
Format:

```
dddddddd0
sssssss0
100xxxxx          (P=1, length=n)
|
11100101          (E5h, Resource Request op-code)
00010000          (10h BWM resource designator)
mlength           (message length)
mwtH              (message rate high)
mwtL              (message rate low)
|
cccccccc
```

2.12.5 Bandwidth Resource Grant command

This command is used by the host as a response to the device bandwidth request (Bandwidth Resource Request)

Host to device
Format:

ddddddd0	
sssssss0	
100xxxxx	(P=1, length=n)
|	
11110100	(F4h, Resource Grant op-code)
00010000	(10h BWM resource designator)
status	(0=success, 1=unsupported, 2=failed- try again later, 4=failed)
mlength	(message length granted)
mwtH	(message rate granted high)
mwtL	(message rate granted low)
cccccccc	

Appendix A

Summary of Standard ACCESS.bus Interface Op-codes

Op-code (hex)	Function
F0	Reset
F1	Identification Request
F2	Assign Address
F3	Capabilities Request
F4	Resource Grant (optional)
F5	Enable Application Report
F6	Power Management (optional)
F7	Presence Check
F8	Device Bandwidth Management (optional)
E0	Attention
E1	Identification Reply
E2	(Reserved)
E3	Capabilities Reply
E4	(Reserved)
E5	Resource Request (optional)
E6	Power Usage Reply (optional)
E7	(Reserved)
E8	Device Bandwidth Usage Reply (optional)
C0	(reserved for vendor usage)
C1	(reserved for vendor usage)
C2	(reserved for vendor usage)
C3	(reserved for vendor usage)
C4	(reserved for vendor usage)
C5	(reserved for vendor usage)
C6	(reserved for vendor usage)
C7	(reserved for vendor usage)
C8	(reserved for vendor usage)
B1	Application Test
A0	Application Hardware Signal (optional)
A1	Application Test Reply
A2	Application Status Message

SECTION 3

ACCESS.bus

Device Driver Interface Specification

SECTION 3

ACCESS.bus

Device Driver Interface Specification

February 1994

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX: 1-408-991-3773

3.0 Introduction

One of the goals of the ACCESS.bus technology is to allow the creation of generic device drivers capable of handling many different devices that belong to the same functional family, while simultaneously allowing a driver to be written for a specific device which takes advantage of its unique features without complex, hardware interface specific code. In addition, it is desirable to isolate the device drivers from the details of managing the ACCESS.bus itself. In order to do this, ACCESS.bus systems provide a bus manager. The bus manager controls the ACCESS.bus host interface hardware, implements the base protocol, including device address assignment and the identification of arriving or departing bus devices; and routes messages to and from devices and their corresponding device drivers.

The purpose of this document is to define the interface between device drivers and the ACCESS.bus bus manager. At this time, the internal structure of bus managers is not specified. Different hardware architectures may require different bus manager architectures.

The following schema illustrates the functional responsibility sharing between the bus manager and the device drivers:

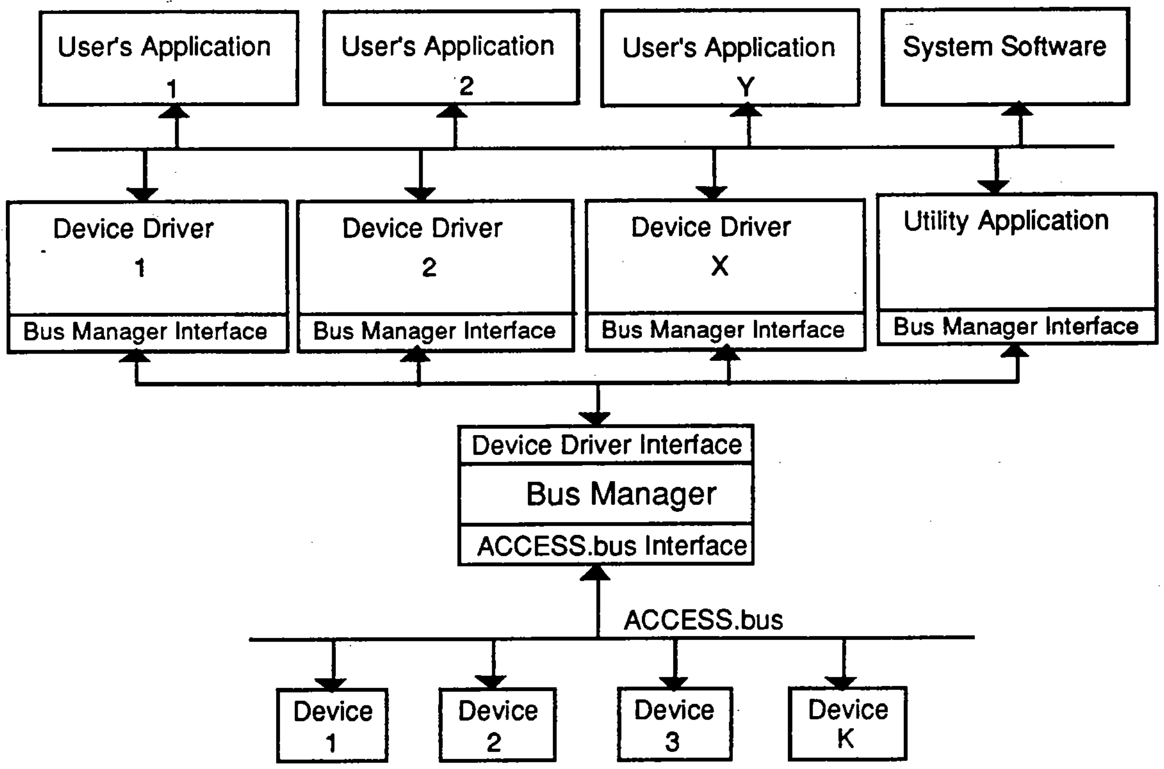

Figure 3.1: Bus Manager/Device Driver Relationship

The bus manager and the device drivers communicate through shared memory buffers. Both the bus manager and the device drivers place formatted messages in memory buffers. For Intel type PCs operating in real mode, device drivers notify the bus manager of a new message via a software interrupt. The bus manager replies to most messages from the device drivers by writing the response into the same message buffer and then returning control to the driver. The bus manager notifies the device drivers of a new message via a far call to the device driver.

The bus manager returns a status code in the AX register for each message it receives from the device driver.

In order to simplify the description of these transactions, we use the phrase "send a message" to describe all of the processes used to transfer data between the device drivers and the bus manager. (See section 3.4 for a detailed description of each of the message passing interfaces for the real mode Intel PC environment).

This approach of using messages in memory buffers is extensible to other processor execution modes, and also to other processor types and operating systems. Of course, the details of how a memory buffer reference is passed, and how control is passed when a new message is present, must change for each environment.

3.1 Basic terminology

This section defines the common terminology to be used in this specification.

3.1.1 Device driver

A functional piece of software that supplies the interface between the physical device, (through the bus manager) and an application or operating system program. The device driver receives the operational data from the device, and transfers it to the application or operating system in the appropriate format.

3.1.2 Device capabilities

A set of attributes that describe the functional characteristics of an ACCESS.bus peripheral device. These attributes are encoded into a capabilities string, which consists of a defined set of tokens and values.

3.1.3 Driver capabilities list

A hierarchical list of device prot(), type(), and model() strings which identifies the set of devices that are supported by the specific driver.

3.1.4 prot( )

The highest level definition of a device's family category. Each prot() is exclusive, and each device belongs to at most one prot ( ).

Example:

prot(locator), defines the family of all devices that are used to obtain pointing input from a user. This category includes all types of mice, digitizing tablets, trackballs, light pens and more.

The value of prot is an ASCII character string. For the bus manager, only the first 8 characters are significant.

3.1.5 type( )

The second level definition of a device's family category. Any device that does not belong to a given type category and belongs to the same prot must be in a different type category.

Example:

type(mouse), defines a subset of pointing devices. A digitizing tablet belongs to the same prot category, but is not included in the type(mouse) category.

The value of type is an ASCII character string. For the bus manager, only the first 8 characters are significant.

3.1.6 model( )

The third and lowest level definition of a device's family category. Any device that does not fall in a given model category, and which belongs to the same prot and type, must be in a different model category.

Example:

model(3D-15L), defines a subset of mice that can handle position and motion in a three-dimensional space.

The value of model is an ASCII character string. For the bus manager, only the first 8 characters are significant.

3.1.7 Device ID

A unique, internal ID number that is assigned to a device by the bus manager. The Device ID is not necessarily equal to the ACCESS.bus address assigned to the same device. The bus manager translates the ID number into the ACCESS.bus-assigned address within outgoing messages, and translates the ACCESS.bus assigned address into the device ID for incoming messages.

3.1.8 Device Table

The Device Table is an ACCESS.bus bus manager data structure that is available to device drivers. The following information is included in the device table for each device present on the bus:

- The device ID
- The device's complete Identification string
- The first eight bytes of the device's 'prot', 'type', and 'model' capabilities strings
- The device status

3.1.9 Device driver linking process

The part of the device driver that communicates with the bus manager to establish a logical link with a physical device or devices on the ACCESS.bus. Since device drivers may support multiple physical devices, and the devices may or may not be present when the driver loads, the bus manager provides a mechanism for a device driver to specify the types of devices it wants to be connected to, and also to register itself for future connection to any devices that meet the driver's connection criteria.

3.1.10 Device Table entry

One entry in the Device table (of a specific device) that specifies the device assigned address, the device identification string, the device capability values of prot, type and model, and the value of the device status.

3.1.11 Device status and status register

One byte of data that indicates the device current status. The bus manager keeps this data as part of the bus manager's Device Table.

The device status byte has the following format:

D7	D6	D5	D4	D3	D2	D1	D0	
not linked	reserved			bad	non-active	virtual	disable	0
linked				good	active	physical	enable	1

D0 indicates whether the device is enabled or disabled. The bus manager will not transfer messages to the driver from a device that is disabled.

A virtual device is a device that physically does not exist, but has an entry in the bus manager Device Table and is being simulated by the bus manager or other software. Virtual devices are useful for software testing, and to allow the simulation of ACCESS.bus devices using non-ACCESS.bus hardware.

A non-active device is a device that has not sent any messages to the host since the last time the bus manager checked its presence.

A bad device is a device that the host was able to assign an address to, but then could not get its prot(), type(), and model() capabilities values, for some reason.

D7 indicates whether the device is connected (linked) to a device driver or is still free (not linked).

3.2 The ACCESS.bus device driver

3.2.1 System configuration

The bus manager can simultaneously support any number of drivers of any category. The set of device drivers installed, and the method through which the set of drivers is defined and loaded, is dependent on the operating systems environment.

3.2.2 Driver capabilities list

Each device driver will normally maintain an internal list of the types of devices it supports. Generic drivers will support any model of a given type or types of devices. More specific drivers may support only one model or device. Normally, the generic drivers and the device specific drivers will coexist.

For example, a user may have both a mouse and a trackball. The generic ACCESS.bus locator driver might be used to support the mouse while a device-specific trackball driver with enhanced features is used to support the trackball. Both drivers must coexist and function appropriately.

In general, device drivers should provide a means to display to the user the types of devices that the driver can support as well as the devices to which the driver has been connected. In some cases, it may be desirable for a device driver to provide the user with control over which devices it will connect to, thereby allowing the user to dynamically control which peripherals are being controlled by a given driver.

Other strategies may be used in specific operating system environments to define the binding of devices to drivers.

The special prot value "Abmon" is used to indicate to the bus manager that the device driver is a debugger or other form of bus monitor that should be connected on a non exclusive basis to all devices. When an Abmon driver is active copies of each device message are sent to the driver.

3.2.3 The Linking process

The linking process is the most complex aspect of the device driver's interface with the bus manager. Device drivers define which devices they want to be linked to by specifying the prot, type and model of the devices they want. The bus manager then replies with a sequence of matching devices which the driver can choose to link to or ignore. Finally the driver can choose to receive notification of new devices that meet its defined categories from the bus manager when the new devices are connected to the bus.

The sequence of the linking process is as follows:

1. The device driver sends a Link request message to the bus manager in which it declares the prot, type, and model categories of the peripheral devices it would like to be connected to.

   The Link request message contains three 9 byte fields, one each for prot, type and model. Only the first eight bytes of each field can be used for a string, at least one null termination byte must be present for each string.

   The asterisk character ("*" , 02A hex) is used as a wild card to indicate any device of that category.

   Examples (see section 3.3.1 for message structure.):

   To specify every pointing device of type mouse:

   prot(locator), type(mouse), model(*)

```
    db          00                  ; Link request minor op code
    db          21h                 ; Link request major op code
    db          00                  ; Device ID = 0
    db          1Bh                 ; Message length = 27 bytes
    db          "locator",00,00
    db          "mouse",00,00,00,00
    db          "*",00,00,00,00,00,00,00,00
    db          29 dup(0)           ; padding to a 60 byte buffer
```

   To specify any available device:

   prot(*), type(*), model(*)

```
    db          00                  ; Link request minor op code
    db          21h                 ; Link request major op code
    db          00                  ; Device ID = 0
    db          1Bh                 ; Message length = 27 bytes
    db          "*",00,00,00,00,00,00,00,00
    db          "*",00,00,00,00,00,00,00,00
    db          "*",00,00,00,00,00,00,00,00
    db          29 dup(0)           ; padding to a 60 byte buffer
```

   {On a real mode Intel PC platform; to send the message the device driver sets the DS (segment), and DX (offset) registers to point to the message and then executes a software interrupt 60 hex. The bus manager uses this same buffer for the Link reply.}

2. The bus manager scans the Device Table for any bus devices that match the categories specified by the device driver.

3. The bus manager replies by sending a Link reply message to the device driver in the same memory buffer. If there is an available bus device that matches the specifications, the reply message will contain the device's complete ID string and its prot, type and model, and the device status.

4. The device driver responds with a Link approval/disapproval message, thereby choosing whether or not to be linked to the device. The linkage is exclusive, with the exception of special bus monitoring drivers which can receive a copy of each message sent or received by a bus device. (These bus monitoring drivers are typically used for diagnostic test or debugging purposes.). Once a device driver approves a link with a device, the device is no longer available to any other device driver.

   If the driver chooses to approve the link, it provides the bus manager with a callback address (a 4 byte far call address seg:offset form) which the bus manager will use to send device events to the driver.

5. Repeat steps 1 to 4 for additional devices until the bus manager indicates that it has reached the end of the list of devices meeting the driver's category specification

6. When the requested device does not exist in the Device Table, the bus manager sends a Link reply with minor op code = FFh (no matching device). If the driver approves the Link reply, the bus manager saves the request in a special pending requests list.

7. Once the link is in place, the driver can send messages to the device itself or to the bus manager.

   The bus manager sends all the application reports of the linked devices to the device driver.

   The driver can manage each device it is linked to separately (by their Device IDs). It is up to the driver or its application to decide how to make use of multiple devices.

8. When a new device is connected to the ACCESS.bus (hot plugged in), the bus manager sends the Device Table entry (op-code 41h) to each driver with a matching pending request. The first driver that responds with a Link approval to the message will be connected to the new device.

Note: The manager sends the Device Table Entry message with the same way as it sends a new application report message (push a pointer to the link reply message buffer onto the stack).

A typical linking protocol sequence is:

Example 1:

Driver	Bus Manager
Link request	
	Link reply (DevID = 1)
Link approval (DevID = 1)	
Link request	
	Link reply (DevID = 2)
Link approval (DevID = 2)	
Link request	
	Link reply minor op code = FFh (no matching device)
Link approval (pending request)	

(for more details see the Link request, and Link reply message descriptions)

Example 2:

The other alternative of linking process is a connection of a new device that was requested previously and was on a pending request list.

Driver	Bus Manager
Link request	
	Link reply (DevID = 1)
Link approval (DevID = 1)	
Link request	
	Link reply minor op code = FFh (no matching device)
Link approval (pending request)	

New device with matching capabilities is hot plugged in

	New device table entry to driver (op code 41h)
Link approval (new device)	

3.3 The Bus manager - Device driver interface

The following protocol is used for sending messages between the bus manager and the device driver on Intel style PCs operating in real mode.

The purpose of this protocol is to open a link from the device driver to the ACCESS.bus peripheral device through the bus manager. The following section lists the messages and describes each message format.

3.3.1 Message Fields

Each message starts with four header bytes, a minor op code, a major op code, the device ID and the message length . The messages are passed between the device driver and the bus manager in a memory buffer. Since the same buffer is used by the bus manager to reply to most of the device driver messages, the device driver must allocate a buffer large enough to hold the expected response. For simplicity, it is a good idea for device drivers to allocate a single message buffer larger than is required by any message/response pair and use the buffer for all communication with the bus manager. A buffer of 132 bytes is sufficient to hold the largest possible message.

All message buffers should be aligned on a word boundary, and messages with odd numbers of data bytes must pad their buffer to an even word boundary to allow the bus manager to read the message by words.

The minor op code is located in the first byte of the message, and the major op code is located in the second byte. The third byte of the message contains the device ID, and the fourth byte of the message contains the length of the message in bytes. The message length is the number of data bytes, excluding the four header bytes, in the message. Note that the buffer length is frequently longer than the message length!

The message header structure is as follows:

Byte Number	Field
1	Minor op-code
2	Major op-code
3	Device ID
4	Message Length

The remainder of the message consists of data bytes.

To avoid byte ordering conflicts, the device and the device driver mutually agree on the data byte order. For example, if a device is sending words in Motorola format, the driver is responsible for converting it to Intel format. (The Manager transfers the data in the same order as it is received from the bus).

3.3.1.1 Minor op code

The first byte of the message is the minor op code. Usually, this field gives more details about the major op code.

3.3.1.2 Major op code

The second byte of the message is the major op code. Usually, this field defines a specific control message or type of data to be transferred.

3.3.1.3 Device ID (DevID)

The third byte of the header defines the peripheral device associated with the message.

3.3.1.4 Message Length

The fourth byte of the header indicates the number of data/control bytes to be transmitted within this message. (Length does not include the four message header bytes).

3.3.1.5 Data Bytes

The body of the message is contained here. Note that messages must start on word boundaries and be in buffers which contain an even number of bytes to allow the bus manager to access the message by word reads. Extra storage after the end of the data bytes is ignored.

3.3.2 Driver to Bus Manager Messages

3.3.2.1 Reset op code - 20h

Minor:	00h
Major:	20h
DevID:	00h
MsgLen:	00h

Notes:
This is a special command message which resets the ACCESS.bus host interface hardware, and as a consequence, all of the devices on the ACCESS.bus. This command should never be issued by normal device drivers!

3.3.2.2 Link request op code - 21h

Minor:	00h
Major:	21h
DevID:	00h
MsgLen:	1Bh (27d)

Data:

9 bytes for the prot value (up to 8 characters + 00h)

9 bytes for the type value (up to 8 characters + 00h)

9 bytes for the model value (up to 8 characters +00h)

All the three category levels have to be defined. If the link request is not for a specific device, the category value for model and possibly type or prot should be set to the asterisk character which acts as a wild card value. The asterisk character ("*", 2A hex) has to be placed in the first byte of the category value. The other 7 bytes will be 00h .

Example 1:

prot(locator)

l	o	c	a	t	o	r		
6C	6F	63	61	74	6F	72	00	00

type(*)

*								
2A	00	00	00	00	00	00	00	00

model(*)

*								
2A	00	00	00	00	00	00	00	00

Notes:
The bus manager will reply with a Link reply message. Programmers should ensure that the buffer they provide is large enough to hold the resonse message.

3.3.2.3 Link approval/disapproval - op code - 22h

Minor:	01h	Yes - connect
	00h	No - don't connect
Major:	22h	
DevID:	Device ID of the bus manager Link reply	
MsgLen:	04h	
Data:	Callback address in 4 byte segment:offset format. This address will be called by the bus manager with messages at interrupt time.	

3.3.2.4 Get specific device table - op code - 23h

Minor	00h
Major:	23h
DevID:	device ID
MsgLen:	00h

Notes:
The bus manager will reply with the device table entry for the specified device if it exists. Programmers should ensure that the buffer they provide is large enough to hold the resonse message.

3.3.2.5 Get device ID string op code - 24h

Minor:	00h
Major:	24h
DevID:	device ID
MsgLen:	00h

Notes:
The bus manager will reply with the specific device ID string. Programmers should ensure that the buffer they provide is large enough to hold the resonse message.

3.3.2.6 Get device type op code - 25h

Minor:	00h
Major:	25h
DevID:	device ID
MsgLen:	00h

Notes:
The bus manager will reply with the value of the prot(), type(), and model() strings of the specified device if it exists. Programmers should ensure that the buffer they provide is large enough to hold the resonse message.

3.3.2.7 Get device status - op code - 26h

Minor:	00h
Major:	26h
DevID:	device ID
MsgLen:	00h

Notes:
The bus manager will reply with the value of the device status for the specified device if it exists. Programmers should ensure that the buffer they provide is large enough to hold the resonse message.

3.3.2.8 Device enable/disable - op code - 27h

Minor:	00h	disable
	01h	enable
Major:	27h	
DevID:	device ID	
MsgLen:	00h	

Notes:
Upon receiving this message the bus manager will enable or disable messages from the specific device to the device driver.

3.3.2.9 Message to device - op code - 28h

This message is used to send a message from the device driver directly to the peripheral device. The contents of this message can be any valid ACCESS.bus message, or any vendor or device specific message. Well-behaved drivers should not ordinarily send Interface Part Control/Status messages to their device because such messages may interfere with the bus manager's control of the ACCESS.bus devices.

Minor:	00h	data message
	01h	control message
Major:	28h	
DevID:	device ID	
MsgLen:	Message length	
Data:	any valid ACCESS.bus or device specific message	

Note:

The bus manager will send this message to the selected device on the ACCESS.bus in the correct ACCESS.bus format by adding the ACCESS.bus headers and the checksum byte and translating the device ID into the actual bus address.

The driver can verify that the message was transmitted successfully by checking the status code returned by the bus manager.

3.3.2.10 Driver disconnect op code - 29h

Before a driver terminates, it must send the above Driver disconnect message to the bus manager. A driver that terminates without sending this message to the bus manager will crash the system, since the bus manager will not know that the driver's call entry point has become invalid.

Minor:	00h
Major:	29h
DevID:	device ID
MsgLen:	00h

3.3.3 Bus manager to Driver Messages

3.3.3.1 Link reply op code - 40h

As a response to Link request, the bus manager sends the following information for each device in the device table that matches the link request specification. After sending a message for a device, the bus manager waits for the driver to respond with a Link approval or disapproval message for that device. For each of the matching devices, the Minor op code is set to 00h. When the bus manager has sent Link reply messages for each bus device that matches the link request filter, it sends one additional Link reply message with a Minor op code of 0FFh to indicate the end of the list of matching bus devices.

Minor:	00h	for a matching device.
	FFh	for the end of the list of devices matching the given set of capabilities
Major:	40h	
DevID:	Device ID	
MsgLen:	38h (56 decimal)	

This message sends the following device related data:

- The device ID
- The device complete ID string
- The device linking capabilities:

  'prot'

  'type'

'model'

Each one of the above capabilities strings is 8 bytes long followed by a 00h.

When the capability string value is shorter than 8 characters it is padded with nulls.

- The device status

The message format is as defined below:

Byte number	Content
1 - (1 byte)	Minor op code - 00 or FFh
2 - (1 byte)	Major op code - 40 h
3 - (1 byte)	Device ID.
4 - (1 byte)	Message Length 38h (56 decimal)
5 - (1 byte)	protocol_revision
6 - 12 - (7 bytes)	module_revision
13 - 20 - (8 bytes)	vendor_name
21 - 28 - (8 bytes)	module_name
29 - 32 - (4 bytes)	device_number
33 - 41 - (9 bytes)	Value of 'prot'
42 - 50 - (9 bytes)	Value of 'type'
51 - 59 - (9 bytes)	Value of 'model'
60 - (1 byte)	Value of the Status register

Note:
The bus manager uses the same message buffer that was used by the driver for the Link request message (specified in DS :DX registers)

3.3.3.2 Specific device table - op code - 41h

This message is a reply to Get specific device table.

Minor:	00h
Major:	41h
DevID:	Device ID
MsgLen:	38h (56 decimal)

This message sends the following device related data:

- The device ID
- The complete ID string for the device.
- The device linking portions of the device's capabilities string:

  'prot'

  'type'

  'model'
- The device status

The message format is as defined below:

Byte number	Content
1 - (1 byte)	Minor op code - 00
2 - (1 byte)	Major op code - 41 h
3 - (1 byte)	Device ID.
4 - (1 byte)	Message Length 38h (56 decimal)
5 - (1 byte)	protocol_revision
6 - 12 - (7 bytes)	module_revision
13 - 20 - (8 bytes)	vendor_name
21 - 28 - (8 bytes)	module_name
29 - 32 - (4 bytes)	device_number
33 - 41 - (9 bytes)	Value of 'prot'
42 - 50 - (9 bytes)	Value of 'type'
51 - 59 - (9 bytes)	Value of 'model'
60 - (1 byte)	Value of the Status register

Note:
The bus manager uses the same message buffer that was used by the driver for the Get specific device table message (specified in DS :DX registers)

3.3.3.3 Device ID op code - 42h

This message is sent as a reply to a Get device ID command

Minor:	00h
Major:	42h
DevID:	Device ID
MsgLen:	1Ch (28d bytes)

Data: 1 byte protocol_revision
7 bytes module_revision
8 bytes vendor_name
8 bytes module_name
4 bytes device_number

Note:
The bus manager uses the same message buffer that was used by the driver for the Get device ID message (specified in DS :DX registers)

3.3.3.4 Device type op code - 43h

This message is the reply to a Get device type command.

Minor:	00h
Major:	43h
DevID:	Device ID
MsgLen:	1Bh (27d bytes)

Data:
'prot' value (8 bytes + 00h)
'type' value (8 bytes + 00h)
'model' value (8 bytes + 00h)

Note:
The bus manager uses the same message buffer that was used by the driver for the Get device type message (specified in DS :DX registers)

3.3.3.5 Device status op code - 44h

This message is sent as the reply to a Get device status message. The status register has the following format:

	D7	D3	D2	D1	D0
0	not linked	bad	non-active	virtual	disable
1	linked	good	active	physical	enable

Minor:	value of the status register
Major:	44h
DevID:	Device ID
MsgLen:	00h

Note:
The bus manager uses the same message buffer that was used by the driver for the Get device status message (specified in DS :DX registers)

3.3.3.6 Message from device op code - 45h

This function is the primary means of communication between the device and the corresponding device driver. The bus manager passes the message to the driver as it is received from the device. Parsing the contents of the message is the device driver's responsibility.

The message is contained in a memory buffer which is only valid for the duration of the call to the device driver. The driver should copy any information it wants to access after the call to its own private storage. Since the call to the device driver will occur at interrupt time, the device driver writer should be aware of the limitations on resources available at interrupt time under the operating system he or she is using.

The bus manager's protocol headers replace the first three bytes of the ACCESS.bus message. (destination address, source address, and protocol flag + length).

The device's bus address is translated into the internal device ID.

The message body is identical to the message on the bus (except for the checksum byte which is removed by the bus manager).

Minor:	00h - data message 01h - control message
Major:	45h
DevID:	Device ID
MsgLen:	message length
Data:	The ACCESS.bus message body (without the checksum byte).

3.3.3.7 Device disconnect - op code - 46h

This message is sent to notify a device driver that a device to which it was linked was disconnected from the bus.

Minor:	00h
Major:	46h
DevID:	device ID
MsgLen:	00h

3.4 The device driver to bus manager interface mechanism (IBM PC specific)

3.4.1 General

The device driver and the bus manager communicate by sending messages in shared memory buffers. All bus manager to device driver messages that are replies to a driver message (op codes: 40h, 41h, 42h, 43h, 44h) use the same memory buffer that was used by the device driver's message. The device driver first allocates a memory buffer large enough to hold the larger of its message or the expected reply and then calls the bus manager by issuing a software interrupt to int 60 hex with the address of the memory buffer in DS:DX . The bus manager's reply will be in the message buffer when control returns to the device driver as long as the call was successful. The device driver should check the bus manager status returned in the AX register before reading the data in the message buffer.

All device application reports (op code 45h) and device disconnect messages (op code 46h) are sent to the driver by the bus manager in a message buffer allocated by the bus manager. Prior to calling the device driver event entry point, the bus manager pushes a far pointer to the message buffer onto the stack.

3.4.2 Driver to bus manager

The following describes how a device driver sends a message to the bus manager:

1. Allocate a memory buffer large enough to hold the message and the anticipated response. Prepare the message in the memory buffer.
2. Set DS:DX to point to the message buffer.
3. Call the bus manager (S/W interrupt - 60h)
4. Check register AX for status:

Status code (AX value)	Status
0x0000	OK
0x00EE	bus manager is busy - release the CPU and try again later
0x0101	unknown op code
0x0102	unknown device ID.
0x0103	resend message
0x0104	outgoing buffer full
0x0105	message length does not match

Example assembler code for the driver's call to the bus manager:

```
MessageBuffer:
        db      00h             ; minor op code
        db      21h             ; Link Request major op code
        db      00              ; Device ID = 0
        db      1Bh             ; Message length = 27 bytes
        db      "locator",00,00
        db      "mouse",00,00,00,00
        db      "*",00,00,00,00,00,00,00,00
        db      101 dup(0)      ; Reserve a 132 byte buffer

CallManager:
        mov     dx,MessageBuffer ; form pointer to message in ds:dx
        int     60h             ; Call the bus manager
        cmp     al,00           ; Test for success
        je      Success
Error:
        <Error handler>
Success:
        mov     bx,dx
        mov     ax,(bx)  ; Read the reply message
        .
        .                       ; process the reply message
        .
```

3.4.3 Bus manager to driver - replies

The following describes how the bus manager replies to a message from the device driver:

1. Prepare the message in the same memory buffer provided by the device driver.
2. Load the AX register with status.
3. Return to the device driver

3.4.4 Bus manager to driver - device and disconnect messages

The following describes how the bus manager initiates a message to the device driver:

1. Allocate a memory buffer and then prepare the message in the buffer
2. Push a far pointer (segment:offset) to the message buffer onto the stack.
3. Call the device driver event handler address with a far call.
4. When it has completed processing the message, the device driver returns control to the bus manager via a far return .

Notes:
The event handler address was defined by the device driver in the Link approval message. The device driver should return from its processing with a far return as soon as possible. The event call will most likely occur at interrupt time, so the device driver should only use system resources that are safe to use at interrupt time. The message buffer provided by the bus manager is only valid during the call to the device driver; the driver should copy

any data it wants to retain from the message buffer to its own local storage prior to returning to the bus manager.

The device driver must ensure that its event handler address is valid at all times until it issues a driver disconnect message to the bus manager.

Example C prototype for the driver's event handler entry point:

```
        void far EventHandler(void far *EventBuffer)
```

Example assembler code for the driver's event handler entry point:

```
EventHandler:
        push    bp              ; preserve registers
        mov     bp,sp
        push    es
        push    dx
        mov     es,(bp+8)
        mov     dx,(bp+6)       ; form pointer to message in es:dx
        .
        .
        .                       ; process the message
        .
        pop     dx
        pop     es
        pop     bp
        retf
```

3.5 Message summary

3.5.1 Driver to bus manager messages

op code	Command
20h	Reset
21h	Link request
22h	Link Approval/Disapproval
23h	Get specific device table
24h	Get device ID
25h	Get device type
26h	Get device status
27h	Device enable/disable
28h	Message to device
29h	Driver disconnect

3.5.2 Bus manager to driver messages

op code	Command
40h	Link reply
41h	Specific device table
42h	Device ID.
43h	Device type
44h	Device status
45h	Message from device
46h	Device disconnect

SECTION 4

ACCESS.bus

Locator Device Protocol Specification

SECTION 4

ACCESS.bus

Locator Device Protocol Specification

February 1994

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX: 1-408-991-3773

4.0 Introduction

4.0.1 Design Objectives

The locator device protocol described in this specification defines standard messages for reporting locator movement and key switch activation as needed for mice, tablets, and other basic positioning devices. The protocol is designed to accommodate a range of basic locator devices such as a mouse or tablet. More complex devices can be modeled as a combination of basic devices or can provide their own device driver, thus minimizing the burden on the protocol.

4.0.2 Overview of Generic Locator

A generic locator consists of one or more dimensions described by numeric values and, optionally, a small number of key switches. The standard driver requires the locator device to identify the type of data it will report from a small list of options and adjusts to handle this data type. These options are:

- Number of dimensions, e.g., 2-D, for a mouse or a tablet
- Dimension type: absolute, i.e., referenced to some fixed origin, like a tablet; or relative, i.e., change since last report, like a mouse
- Resolution in divisions per unit, e.g., counts per inch or counts per revolution
- Dynamic range of values that can be reported, i.e., the minimum and maximum values
- Number of key switches, from 0 to 15

The assignment of scalar-value dimensions returned from one or more devices to the user interface functions is left to the application. However, to accommodate most conventions, the scalar dimensions and the key switches can be labeled in the capabilities string.

4.1 Locator Event Reports

Locator reports are generated in response to a poll command, or at each sampling interval in which a change in position or button state has been detected. The sampling interval defaults to twelve milliseconds (12ms) (83Hz update), but is setable from the host.

Locator event reports include the current button state and the current position or movement since the last report. For simplicity, these are coded as a sequence of two byte integers. The first integer contains the state of up to sixteen (16) locator key switches. The remaining integers represent locator dimensions. Locator event reports are transmitted using the device data stream message (see ACCESS.bus Description and Protocol Specification).

Example: A 2D mouse might report:

50h	computer address
54h	device address
06h	data stream, data length 6
0001h	button "B1" is down
0017h	X movement is 17 units
FFF4h	Y movement is -12 units
xx	message checksum

4.2 Capabilities Information

The keywords defined in this section have standard meanings within the ACCESS.bus Generic Locator Device Protocol.

Keyword	Meaning
prev()	The "prev()" value are two bytes that indicate the current ACCESS.bus protocol revision, and the current device specific protocol revision. The current prev value for locator is BA- prev(BA) - B is the base protocol revision and A is the current locator protocol revision.
type()	The "type()" entry is intended to identify the device type to the user in a recognizable form. type() is a user's view of a device. That is, a joystick, mouse, or whatever. It is also a second level identifier of the device used by the system software.
type(mouse)	Mouse
type(digitizer)	Digitizing tablet
type(tball)	Trackball
type(ptrstick)	Force activated joystick (typically embedded in keyboards)
type(touchscn)	Touchscreen
type(dial)	Dials, arrays of dials, and other single axis valuators
type(swpad)	Switch pads, such as those used for game control, where a set of switches are used to control position and functions. (as opposed to keyboard)
buttons()	The "buttons()" entry lists and describes the key switches that are reported to host software. The parenthesis contain a list of tagged labels giving the bit number in the keyswitch word or name.
L	Left mouse button.
R	Right mouse button.
M	Middle mouse button.
Bl B2 B3 B4	Numbered locator buttons.
Tip	Stylus tip button.
Barrel	Stylus barrel button.
P	Sensor in proximity indicator.
dim()	The "dim()" entry gives the number of dimensions reported by a locator device.
d0() . . .dn()	The "d0()", "dl()", . . . , "dn()" entry groups' attributes that apply to a single dimension. Capability attributes within the parenthesis apply only to that dimension. Capability attributes outside a "dn()" entry apply to all dimensions reported by the device.
rel	The "rel" attribute identifies dimensions that report relative coordinates, i.e., change since last report, like a mouse.
abs	The "abs" attribute identifies dimensions that report absolute coordinates, i.e., referenced to some fixed origin, like a tablet.
res()	The "res()" entry describes the resolution of one or more locator dimensions. The parenthesis contain an integer number and unit. The number represents the number of movement increments reported for a change of one unit.
inch	Counts per inch.
cm	Counts per centimeter.

rev	Counts per revolution.
range()	The "range()" entry describes the range of values that can be reported for one or more dimensions. The parenthesis contain two integer numbers corresponding to the minimum and maximum values that can be reported.
dname()	The "dname()" entry specifies a label or name for a dimensions.
X	X dimension
Y	Y dimension
Z	Z dimension
RX	Rotation about X axis
RY	Rotation about Y axis
RZ	Rotation about Z axis
PN	Pressure Normal to the sensing surface
PT	Pressure Tangent to the sensing surface, typically the squeeze pressure on a pen barrel.

Consider the following example locator device capabilities string:

```
(
prot(locator)
type(mouse)
model(VSXXX)
buttons(l(L)2(R)3(M)) dim(2) rel res(200 inch) range(-127 127)
d0(dname(X))
dl(dname(Y))
)
```

"prot(locator)" tells host software that this device is a generic locator and follows the locator device protocol.

"type(mouse)" provides a user recognizable description of the type of locator device.

"model(VSXXX)" is a user readable identification of the device model.

"buttons(l(L)2(R)3(M))" describes the device as having 3 key switches or buttons labeled "L" (left), "R" (right), and "M" (middle). The corresponding bits in the key switch word are also identified.

"dim(2)" describes the device as a two dimensional locator.

"rel res(200 inch) range(-127 127)" are characteristics of the device that apply to all of its dimensions since they are not enclosed within a single dimension tag. In this case, each dimension reports relative movement with resolution of 200 counts per inch. The reported movements can range from -127 to +127.

The dimension tag "d0( )" indicates characteristics that apply to a single dimension only. The "dname(X)" tag names dimension "d0" as "X".

4.3 Locator Conventions

The following conventions are recommended for devices used to input 2D or 3D spatial information:

1. Position coordinates are reported in order: X positive from left-to-right; Y positive from down-to-up; Z positive out of the screen (toward the operator viewpoint).

2. Rotations are reported in order around the X, Y, and Z axis using a "right hand" coordinate system.

3. Key switches are reported as bits in a 16 bit key switch word (l=depressed or on) and should be labeled in the Capabilities String. The following default assignments are recommended (Bit 1 = LSB).

Table 4.1: Recommended Default Bit Assignment in Locator Key Switch

Bit-numbered (label)	Description
1 (L)	Left mouse button
2(R)	Right mouse button
3(M)	Middle mouse button
1 (B1) 2(B2) 3(B3) 4(B4)	Button 1-4
1 (Tip)	Stylus tip button
2(Barrel)	Stylus barrel button
15(P)	Sensor in proximity

4. Dials are reported in order from left to right and top to bottom, with increasing values corresponding to clockwise rotation. Obviously, not all devices will fit these conventions. These recommendations are intended to simplify interchanging common locator devices such as mice, tablets, trackballs, joysticks, touch screens, and dial boxes.

4.4 Timing and Exceptions

If a dimension reporting relative movement overflows within a single reporting interval, the maximum value should be reported.

4.5 Locator Messages and Commands

4.5.1 Locator Report (Device Data Stream)

The Locator Report message reports the current locator position or movement, and key switch state.

Format;

```
ddddddd0
sssssss0
000xxxxx                (P=0, length=4-34)
locator button state    (16-bit keyswitch word)
|
one to 16 scalar         (each value is a 16-bit
dimension values          signed integer)
|
cccccccc
```

4.5.2 Application Set Sampling Interval

The Set Sampling Interval command sets the locator sampling interval from 1 to 255 milliseconds (3.92 to 1000 reports/second). A parameter value of zero selects polled operation, that is no unsolicited reports. Devices may not be able to set their interval to the exact value requested by the host. Devices should set their interval to the closest value possible that is less than or equal to the requested interval. As a design guide, devices will typically only be requested to set their sampling interval in the range of 8 to 25 milliseconds (40 to 120 reports/second). Set Sampling Interval is a device defined control/status message:

Op-code: 01 Data: 1-byte number of milliseconds.

4.5.3 Application Poll

The Application Poll command instructs the locator to report its current state as a Locator Event Report. The Locator Event Report includes the current movement or position and status of any locator buttons.

Op-code: 02 Data: none.

4.5.4 Locator Self Test Report

Upon receiving an Application Test command, the locator will test its electronics and firmware and report the results as an Application Test Reply report (see ACCESS.bus Description and Protocol Specification). The locator uses the following status values in its Attention report:

0	Success
1	ROM checksum error detected
2	RAM error detected
3	Sensing or hardware error detected
4	Sensing device or cursor out of proximity
5	Other error

If a ROM checksum error is detected, the second data byte will give the computed non-zero checksum.

SECTION 5

ACCESS.bus

Keyboard Device Protocol Specification

SECTION 5

ACCESS.bus

Keyboard Device Protocol Specification

February 1994

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX: 1-408-991-3773

5.0 Introduction

5.0.1 Design Objectives

The keyboard device protocol described in this specification defines standard messages for reporting keystrokes and controlling keyboard peripherals. The keyboard device protocol attempts to define the simplest set of functions from which common industry standard keyboard interfaces can be built. The following principles were used to guide the design:

1. Provide sufficient functional completeness to support existing user interfaces.
2. Reduce complexity in human terms wherever possible. Existing keyboard drivers do not utilize many of the features provided. Some have introduced errors in handling the subtle interactions defined.
3. Minimize state information that must be modeled in both the keyboard and the host. This is to avoid synchronization problems.
4. Minimize memory required in the keyboard. Cost per bit is much lower on the host.
5. Minimize per unit cost for hardware and firmware while allowing high function alternatives.
6. Provide standard Key Code for office keyboards so that they can use standard drivers. Other keyboards can provide alternate tables with the standard drivers or replace the drivers altogether

5.0.2 Generic Keyboard Overview

A generic keyboard consists of an array of key stations assigned numbers (Key codes) between 8 and 255 (08 - FF). When any key station transitions between open and closed, the entire list of Key Codes for key stations currently closed or depressed is transmitted to the host.

In addition to reporting key stations, the generic keyboard device can support simple feedback mechanisms such as key clicks, bells, and light-emitting diodes. These mechanisms are controlled explicitly from the host so that minimal keyboard state modeling is required. The capabilities information is used to identify the keyboard mapping table and the feedback mechanisms available. The keyboard mapping and language configuration can also be stored in the keyboard itself as part of the capabilities string.

5.1 Key Event Reporting

Each key is assigned a unique 8-bit number (8-255). The first eight (8) codes are reserved for other keyboard functions. On each key transition, up or down, the keyboard will report the complete state of the key array as a list of zero to ten key stations that are currently down.

Example: user enters the modified keystroke Alt-Shift-A

```
transition      report

Alt   down      Alt
Shift down      Alt Shift
`A'   down      Alt Shift A
`A'   up        Alt Shift
Shift up        Alt
Alt   up        <empty list> (see 5.7.1)
```

This reporting scheme is functionally complete in that the host can detect every key transition and it provides the full state of the keyboard on each report. No special re synchronization reports are needed.

To simplify generating the reports, keys can be reported in any order.

5.2 Auto Repeat

Auto-repeat is the responsibility of the host. The common model is to have appropriate keys begin auto-repeating at a constant <rate> if held down for longer than some start up <delay>. The auto-repeat <rate> and <delay> are often user setable.

5.3 Key click and Bell

Key click is handled manually by the host sending the command to click after each appropriate key transition or auto repeated keystroke. In order for key click to appear instantaneous, key click should occur within 100 milliseconds of the corresponding key transition. The key click volume is specified by the host on each command so there is no state to set or remember in the keyboard.

Bell is handled manually by the host sending the command to sound the bell as needed. The bell volume is specified by the host on each command so there is no state to set or remember in the keyboard.

5.4 Capabilities Information

The keywords defined in this section have standard meanings within the ACCESS.bus Generic Keyboard Device Protocol.

Keyword	Meaning
prev()	The "prev()" value are two bytes that indicate the current ACCESS.bus protocol revision, and the current device specific protocol revision. The current prev value for keyboards is BA- prev(BA) - B is the base protocol revision and A is the current keyboards protocol revision.
keymap()	The "keymap()" entry identifies what platform/layout a keyboard is configured for. The primary usage will be to define the keys available on the current keyboard as a subset of those on the master table. This feature allows a single keyboard driver to support multiple platforms and multiple keyboard designs for each platform. A second usage would allow the system to substitute other key codes from the master list to convert some or all of those stored and sent by the keyboard, This feature provides the user flexibility through remapping and improved emulation. The keymap() parameter will also define special key combinations that control functions such as reset, pause, attention and reboot.
lang()	The "lang()" entry identifies keyboard language. The keyboard will indicate to the system driver the language/country for which it is configured. The system ACCESS.bus driver will then substitute the appropriate characters for those keys which undergo language/country specific changes. The key used for a period or comma on the numeric keypad should be determined by keymap(). Whether it generates a comma or period should be determined by the operating system, locale settings. However if the system does not have locale settings, lang() may be used. The lang() parameter will also control language specific keyboard functions such as but not limited to: Dead key tables Treat Caps Lock as Shift Lock Toggle or latch Shift Lock Use Shift-Alt instead of Ctrl-Alt Alt-gr use
feedback()	The "feedback()" entry lists and describes the feedback mechanisms available for use by host software.
click()	The "click()" entry indicates the keyboard provides a host controlled click feature. The parenthesis contain an integer representing the maximum volume setting. Zero is assumed to be the minimum.
bell()	The "bell()" entry indicates the keyboard provides a host controlled bell feature. The parenthesis contain an integer representing the maximum volume setting. Zero is assumed to be minimum.

pitch()	The "pitch()" entry indicates the keyboard provides a host controlled bell pitch control feature. The parenthesis contain an integer representing the maximum pitch range setting. Drivers with no user interface to control pitch should use the median value. Zero is the lowest pitch supported.
led()	The "led()" entry indicates the keyboard provides 1 to 16 host controlled LED indicators. The parenthesis contain a list of tagged labels giving the bit switch number in the illumination mask and corresponding .indicator label or name. The following labels are currently defined: Other labels are reserved for future assignment.

hold, com (compose), wait, num (num lock),
cap (caps lock/shift lock), scr (scroll lock)

Warning - Bits switches are numbered from 1 (Least Significant Bit) to 16 (most Significant Bit). This is different from normal bit numbering.

5.5 Timing and Exceptions (Guideline)

The keyboard microprocessor scans the key array repeatedly to detect key transitions. If more than one key transition is detected during a single scan, all key transitions will be reported together as part of the new keyboard state. In this case, the order of key transitions cannot be determined.

When the microprocessor completes a scan of the key array and has detected one or more key transitions, it will try to assume bus mastership to send a keyboard report. Since other devices may be using the bus, it could take some time before the keyboard is allowed to become bus master. The processor may restart scanning the key array during this time, and wait to be interrupted when bus mastership has been granted.

It is assumed devices attached to the ACCESS.bus will have an opportunity to report during every 20 milliseconds.

5.6 Host Commands to Keyboard

5.6.1 Application Click (Optional)

The Application Click command instructs the keyboard to generate a click sound (device defined Control/Status, P=1).

Op-code:	01
Data:	1-byte click volume

5.6.2 Application Bell (Optional)

The Application Bell command instructs the keyboard to generate a bell sound (device defined Control Status, P=l).

Op-code:	02
Data:	1-byte bell volume
Data:	1-byte bell pitch

5.6.3 Application LEDs

The Application LEDs command instructs the keyboard to illuminate one or more LED indicators (device defined Control/Status, P=1).

Format:

Op-code:	03
Data:	2-byte illumination mask: 0=off, 1=on

Note: bits are defined in LED capabilities. See 2.6.2.2

5.6.4 Application Poll

The Application Poll command instructs the keyboard to report its current state showing which keys are currently down (device defined Control/Status, P=1). See Keyboard State Report.

Op-code:	04
Data:	none

5.7 Keyboard To Host Data

5.7.1 Keyboard State Report (Device Data Stream)

The keyboard State Report transmits a list of up to ten key codes (8-255) for the keys that are currently down. Code value zero (0) means the key list is empty, no keys are down.

0101000	(50 = Host Dest addr)
ddddddd	Device Address
00000001	(P=0, Length = 1)
00000000	No keys down
cccccccc	Check Sum

5.7.2 Keyboard Output Error

The Keyboard Output Error message indicates that the keyboard has detected a key state it cannot report. This might occur because more than ten keys are being held down simultaneously or a possible phantom key has been detected. The keyboard will transmit a valid key state report as soon as the condition preventing the key state from being sent is corrected. Keyboard Output Error is a device defined Control/Status message.

Keyboard Output Error should be reported using the uniform Application Status Message (op code = A200). Although other devices use vendor dependent error reporting, all keyboards will use the following standard error reporting to allow the use of a common device driver.

The error report message would be:

0101000	(50 = Host Dest addr)
ddddddd	Device Address
10000101	P=1, Length = 5)
10100010	A2 op code
00000000	Secondary op code
00000110	(06 = Error Occurred)
000000pt	p=1 if phantom key error (bit 1) t=1 if too many keys depressed (bit 0) (LSB)
00000000	Second error byte
cccccccc	Check Sum

5.7.3 Keyboard Startup

At device power up, or upon receiving a "Reset" command, the keyboard will set its device address to the ACCESS.bus default address (6E) and test its electronics and firmware and will report its presence with an "Attention" report (see ACCESS.bus Description and Protocol Specification).

Start up guidelines:

> If a ROM or RAM error is detected, the keyboard may attempt to start normal operation anyway.
>
> If a key down error is detected, the keyboard will send the attention message. It will flag the key down condition when the driver asks for the self-test results and then resume scanning the key array until all keys are up. It will then use the Applications Status Message to report 'device ready'.

5.7.4 Keyboard Self-Test report

The keyboard will report its self-test results using the Application Test Reply message (op code A1). The first Application Test (op code B1) after startup will signal the keyboard to send its startup self test results in the following format:

00	Successful Self-Test
01	ROM checksum error detected
02	RAM error detected
03	Key down error detected

The ROM error will be followed by the computed 16 bit checksum and Key down error will be followed by the key code for the first key detected, respectively.

Multiple errors can be combined in the same test report.

5.7.5 Keyboard Capabilities Change notification

Keyboard capability changes must be signaled with the Applications Status Message Command (op code A200). In response to the new host request for capabilities the keyboard will return a new capability set that will completely replace the previous set.

This feature will usually be used to change lang() for a multilingual keyboard or keymap() for a user programmable keyboard.

5.8 Keyboard Mapping Tables

5.8.1 Keyboard Mapping

This standard will establish standard mapping for Key Codes since it is not practical for every keyboard manufacturer to provide keyboard mapping tables for every country variation of every language for every operating system. Once a Key Code is assigned it will not change in all future standards. This standard will set Key Code to character or function relationships. **Key positioning information is not part of the standard and is only provided as a guide.**

Mapping is controlled by two capability parameters, keymap() and lang(). Keymap identified the platform specific layout of the keyboard. For example keymap(EPC) identifies the keyboard as being a 101/102 Extended PC keyboard. Keymap(SUN5) identifies it as a type 5 Sun keyboard and a keymap(LK501) is a DEC keyboard layout.

The Lang() mapping is consistent across platforms but may vary by country or locale. French keyboards in France typically use the AZERTY layout regardless of platform. The lang() mapping tables will be provided with the operating system specific keyboard driver. The operating system will usually allow the user to override lang() to allow users to type in languages not supported by the keyboard hardware. This will allow users to put stickers on the key caps without reprogramming or reoptioning the keyboard. Usually the user will allow the keyboard to designate the language so that they can use either smart multi-lingual keyboards or different language specific keyboards. If no lang() is specified or if it does not match any language supported by the operating system then the default will be US English. (See Appendix 5D)

5.8.2 PC Keyboard Mapping

Because of the large number of PC keyboards, Appendix 5.C shows the standard Key Codes in a format specific to US English PC 101 keyboards.

APPENDIX 5.A
LANGUAGE MAPPING

Currently the only language defined is US English which is the default if no lang() parameter is supplied.

Table 5A.1 Language Table

Key Code	LANG()	Typical Position
0E	`~	E00
16	1!	E01
1E	2@	E02
26	3#	E03
25	4$	E04
2E	5%	E05
36	6^	E06
3D	7&	E07
3E	8*	E08
46	9(	E09
45	0)	E10
4E	-_	E11
55	=+	E12
5D	|\	E13
15	Q	D01
1D	W	D02
24	E	D03
2D	R	D04
2C	T	D05
35	Y	D06
3C	U	D07
43	I	D08
44	O	D09
4D	P	D10
54	[{	D11
5B	]}	D12
5C	\\|	D13

Key Code	LANG()	Typical Position
1C	A	C01
1B	S	C02
23	D	C03
2B	F	C04
34	G	C05
33	H	C06
3B	J	C07
42	K	C08
4B	L	C09
4C	;:	C10
52	'"	C11
53	\\|	C12
61		B00
1A	Z	B01
22	X	B02
21	C	B03
2A	V	B04
32	B	B05
31	N	B06
3A	M	B07
41	,<	B08
49	.>	B09
4A	/?	B10
62		B11
67		A02
29	Space	A05
64		A08
13		A09

Note that the shifted 6 is a caret '^' on an ASCII system and a not sign '¬' on an EBCDIC system with a standard US language layout.

APPENDIX 5.B
KEYMAPS

This is the standard key mapping for Key Codes of normal office systems keyboards. Normally only a subset of these codes are implemented. Each platform has different requirements. Cross platform mixes of keyboards and systems will require KEYMAPS() to invoke special remapping so that the keyboard will support a minimal set of required functions of the system. Table 5B.1 is to be used as a guide line. Developers can deviate from this table in two ways:

1. They can move keys to other positions on the keyboard and move the key codes to the new position. The advantage is that they can use standard keymaps.

2. Key functions can be moved to other locations using the key codes at those new locations. New keymaps must be used but this technique save having to redo the internal key mapping within that keyboard controller

The following table provides a suggested starting point to maintain commonalty between hardware and devices driver designs. The typical positions are for reference purposes only.

Table 5B.1 Standard Key Code Mapping

Typical Position	Key Code	KEYMAP(EPC) PC 101/102	IBM 101 Position	Other Functions
E14	66	Backspace	15	
D0Q	0D	Tab	16	
C99	AF			Ctrl
C00	58	Lock	30	
C13	5A	Return	43	
B99	12	Left Shift	44	
B12	59	Right Shift	57	
A99	14	Left Ctrl	58	Ctrl
A00	B1			Left Compose/Alt
A01	11	Left Alt	60	Left Diamond
A10	91	Right Alt/AltGr	62	AltGr
A11	AD			Right Compose
A12	94	Right Ctrl	64	Right Ctrl
E31	F0	Insert	75	Find/PA1
E32	EC	Home	80	Insert/Page Up
E33	FD	Page Up	85	Page Down/Remove
D31	F1	Delete	76	End/Select
D32	E9	End	81	Insert/Prev
D33	FA	Page Down	86	Delete/Next
C31	82			
C32	85			Up Arrow
C33	84			
B31	86			Left Arrow
B32	F5	Up Arrow	83	
B33	87			Right Arrow

Typical Position	Key Code	KEYMAP(EPC) PC 101/102	IBM 101 Position	Other Functions
A31	E5	Left Arrow	79	
A32	F2	Down Arrow	84	
A33	F4	Right Arrow	89	
E51	77	Num Lock	90	Esc
E52	CA	KP /	95	
E53	7C	KP *	100	
E54	7B	KP -	105	
D51	6C	KP 7	91	
D52	75	KP 8	96	
D53	7D	KP 9	101	
D54	79	KP +	106	
C51	6B	KP 4	92	
C52	73	KP 5	97	
C53	74	KP 6	102	
C54	65		107	
B51	69	KP 1	93	
B52	72	KP 2	98	
B53	7A	KP 3	103	
B54	DA	KP Enter	108	
A51	6D		94	KP 0
A52	70	KP 0	99	KP 00
A53	71	KP . or ,	104	KP 000
A54	DC		109	
L99	76	Esc	110	
L01	17	F1	112	
L02	18	F2	113	
L03	19	F3	114	
L04	0C	F4	115	
L05	1F	F5	116	
L06	0B	F6	117	
L07	83	F7	118	
L08	0A	F8	119	
L09	20	F9	120	
L10	09	F10	121	
L11	78	F11	122	
L12	27	F12	123	
L13	0F			F13
L14	10			F14
L31	92	Print Screen	124	Help
L32	7E	Scroll Lock	125	
L33	95	Pause	126	Pause
L51	28			Mute
L52	2F			Vol -
L53	30			Vol +
L54	37			Power
K01	38			F13
K02	39			F14

Typical Position	Key Code	KEYMAP(EPC) PC 101/102	IBM 101 Position	Other Functions
K03	3F			F15
K04	40			F16
K05	47			F17
K06	48			F18
K07	4F			F19
K08	50			F20
K09	51			F21
K10	56			F22
K11	57			F23
K12	5E			F24
E79	5F			Attn/SysReq/Stop
E80	60			Clear/Again
D79	63			CrSel/Props
D80	68			Undo
C79	6A			ExSel/Front
C80	6E			ErEOF
B79	6F			Oper
B80	7F			Copy
A79	80			Find
A80	81			Out
L79	88			
L80	89			Help

APPENDIX 5.C
PC 101/102 KEYBOARD DESIGN

This appendix describes an example ACCESS.bus keyboard implementation and should not be viewed as specifying any conformance requirements.

5.C.1 PC 101 Capabilities Information Example

prot(keyb)
model(PC101)
type(keyboard)
keymap(EPC)
feedback(click(15)bell(15)led(2(scr)3(cap)4(num)))

lang() defaults to US English, Click and bell volumes can range for 0 to 15.

The LED control byte for standard PC keyboards:

Bitswitch 1	Not Used	(Least Significant Bit)
Bitswitch 2	Scroll Lock	
Bitswitch 3	Caps Lock	
Bitswitch 4	Num Lock	
Bitswitch 5-16	Not Used	

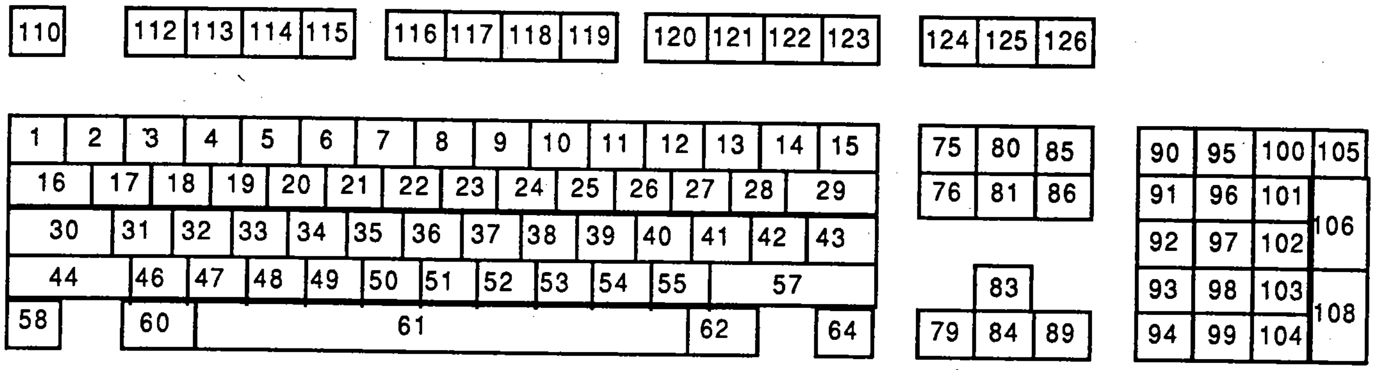

Figure 5C.1 PC 101 key keyboard Key Numbers

Table 5C.1: Key Codes as applied to an Extended PC keyboard.

Key Number	Key Code	LANG()	KEYMAP(EPC)	Position
1	0E	`~		E00
2	16	1!		E01
3	1E	2@		E02
4	26	3#		E03
5	25	4$		E04
6	2E	5%		E05
7	36	6^		E06
8	3D	7&		E07

Key Number	Key Code	LANG()	KEYMAP(EPC)	Position
9	3E	8*		E08
10	46	9(		E09
11	45	0)		E10
12	4E	-_		E11
13	55	=+		E12
14	5D	\\\|		E13
15	66		Backspace	E14
16	0D		Tab	D00
17	15	Q		D01
18	1D	W		D02
19	24	E		D03
20	2D	R		D04
21	2C	T		D05
22	35	Y		D06
23	3C	U		D07
24	43	I		D08
25	44	O		D09
26	4D	P		D10
27	54	[{		D11
28	5B	]}		D12
29	5C	\\\|		D13
30	58		Caps Lock	C00
31	1C	A		C01
32	1B	S		C02
33	23	D		C03
34	2B	F		C04
35	34	G		C05
36	33	H		C06
37	3B	J		C07
38	42	K		C08
39	4B	L		C09
40	4C	;:		C10
41	52	'"		C11
42	53	\\\|		C12
43	5A		Return	C13
44	12		Left Shift	B99
45	61	Reserved		B00
46	1A	Z		B01
47	22	X		B02
48	21	C		B03
49	2A	V		B04
50	32	B		B05
51	31	N		B06
52	3A	M		B07
53	41	,<		B08
54	49	.>		B09

Key Number	Key Code	LANG()	KEYMAP(EPC)	Position
55	4A	/?		B10
56	62	Reserved		B11
57	59		Right Shift	B12
58	14		Left Ctrl	A99
60	11		Left Alt	A01
61	29	Space		A05
62	91		Right Alt	A10
64	94		Right Ctrl	A12
65	67	Reserved		A02
66	64	Reserved		A08
67	13	Reserved		A09
75	F0		Insert	E31
76	F1		Delete	D31
77	82		Reserved	C31
78	86		Reserved	B31
79	E5		Left Arrow	A31
80	EC		Home	E32
81	E9		End	D32
82	85		Reserved	C32
83	F5		Up Arrow	B32
84	F2		Down Arrow	A32
85	FD		Page Up	E33
86	FA		Page Down	D33
87	84		Reserved	C33
88	87		Reserved	B33
89	F4		Right Arrow	A33
90	77		Num Lock	E51
91	6C		KP 7	D51
92	6B		KP 4	C51
93	69		KP 1	B51
94	6D		Reserved	A51
95	CA		KP /	E52
96	75		KP 8	D52
97	73		KP 5	C52
98	72		KP 2	B52
99	70		KP 0	A52
100	7C		KP *	E53
101	7D		KP 9	D53
102	74		KP 6	C53
103	7A		KP 3	B53
104	71		KP . or ,	A53
105	7B		KP -	E54
106	79		KP +	D54
107	65		Reservede	C54
108	DA		HP Enter	B54
109	DC		Reserved	A54

Key Number	Key Code	LANG()	KEYMAP(EPC)	Position
110	76		Esc	L99
112	17		F1	L01
113	18		F2	L02
114	19		F3	L03
115	0C		F4	L04
116	1F		F5	L05
117	0B		F6	L06
118	83		F7	L07
119	0A		F8	L08
120	20		F9	L09
121	09		F10	L10
122	78		F11	L11
123	27		F12	L12
124	92		Print Screen	L31
125	7E		Scroll Lock	L32
126	95		Pause	L33

Some key numbers do not have keys caps on a standard PC 101 keyboard, but they may have switches that are wired into the matrix even if they can not be used. The Key Codes are provided because they may be used in other configurations. These key numbers and the corresponding Key Codes are marked as "Reserved" in either the LANG() or KEYMAP() columns.

APPENDIX 5.D
EXAMPLE ISO/EDUC 9995 KEY POSITIONS

The ISO/EUC 9995 standard is a universal standard that identifies key positions using a coordinate system. Other vendor specific systems are often incomplete and require special accompanying diagrams. Columns 78 and 80 are on the left-hand side of the keyboard. These position identifiers are only used for reference purposes only. Actual key positioning is up to the keyboard implementer.

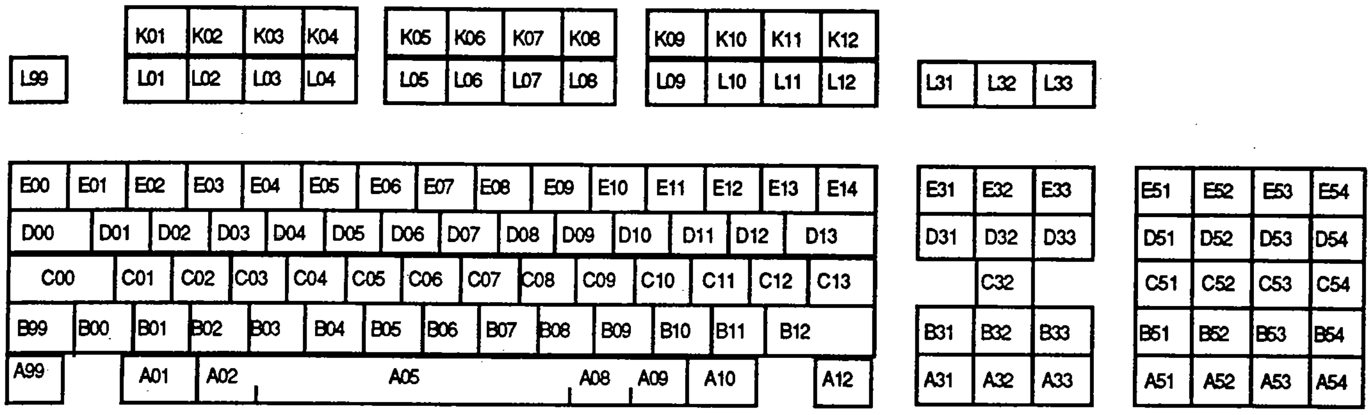
K01 K02 K03 K04 K05 K06 K07 K08 K09 K10 K11 K12
L99 L01 L02 L03 L04 L05 L06 L07 L08 L09 L10 L11 L12 L31 L32 L33
E00 E01 E02 E03 E04 E05 E06 E07 E08 E09 E10 E11 E12 E13 E14 E31 E32 E33 E51 E52 E53 E54
D00 D01 D02 D03 D04 D05 D06 D07 D08 D09 D10 D11 D12 D13 D31 D32 D33 D51 D52 D53 D54
C00 C01 C02 C03 C04 C05 C06 C07 C08 C09 C10 C11 C12 C13 C32 C51 C52 C53 C54
B99 B00 B01 B02 B03 B04 B05 B06 B07 B08 B09 B10 B11 B12 B31 B32 B33 B51 B52 B53 B54
A99 A01 A02 A05 A08 A09 A10 A12 A31 A32 A33 A51 A52 A53 A54

Figure 5.d1: Example of some ISO/EUC 9995 Key positions

SECTION 6

ACCESS.bus

Text Device Protocol Specification

SECTION 6

ACCESS.bus

Text Device Protocol Specification

February 1994

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX: 1-408-991-3773

6.0 Introduction

6.0.1 Design Objectives

The text device protocol described in this specification is intended to provide a simple way to transmit character or binary data to and from stream oriented devices such as a bar code reader, or character display. The sequential character stream model also serves as a common denominator for connecting RS-232 interfaced devices.

6.0.2 Text Device Overview

A generic text device transmits a stream of 8-bit bytes from a character set. Simple control messages are defined to support flow control and to select communication parameters that might be used to interface with a modem. The capabilities string contains information that identifies the specific character set and communication parameters used.

6.1 Text Device Protocol

Text characters to be transmitted are sent using the Device Data Stream message (see ACCESS.bus, Description and Protocol Specification).

Format:

```
ddddddd0      (destination address)
sssssss0      (source address)
0LLLLLLL      (P=0, LLLLLLL=body-length)
|
body          (0-127 bytes)
|
cccccccc      (checksum)
```

Characters are assumed to be from the coded character set specified in the Capabilities Information unless otherwise agreed upon between the application and device. This agreement might be established using ISO standardized control functions, but this is defined at a higher level than the text device protocol.

6.1.1 Device Number and Identification

A text device may be interactive like a bar code reader that transmits a stream of characters in response to user action, or non-interactive like an RS-232 converter attached to a printer. Non-interactive devices are required to provide a fixed device number since they cannot be distinguished by the order in which they are used.

A general purpose ACCESS.bus to RS-232 adapter may have no way of reporting what is attached to its RS-232 port. In this case, the host system or application must rely on the RS-232 device for device identification. Information about what is attached to an RS-232 adapter can be maintained on the host and associated by the device number if desired.

6.1.2 Message Length and Timing

Interactive text devices must not occupy the bus as bus master for more than eight milliseconds at a time, and must wait at least fifty microseconds (50ms) between message transmissions. Eight milliseconds (8ms) limits the maximum packet size to eighty characters or less at 100Kbps.

Non-interactive devices must not occupy the bus as bus master for more than five milliseconds at a time, and must wait at least twelve milliseconds (12ms) between message transmissions. Five milliseconds (5ms) limits the maximum packet size to fifty characters or less at 100Kbps.

The Text Device Protocol defines a Record Separator command to group message packets into identifiable records.

Application Record Separator ( )

Op-code:	30
Data:	none

The Record Separator is sent by a device or the host computer to indicate the next data byte transmitted will be the first byte of a new record. Can also be used as a "end of record" command.

6.1.3 Flow Control

When a computer or device is capable of sending data faster than its bus partner can receive and process it, flow control may be needed to avoid losing data. In its simplest form, flow control allows the receiver to tell the sender to wait until it is ready before sending more data. While the I^2C hardware provides low level flow control by stretching clock signals, this will not be appropriate for many applications because it blocks all traffic on the bus.

An example application requiring higher level flow control is a printer that cannot print data as fast as the host can send it. Depending on the application, the printer may need to instruct the computer to not transmit any more messages until it has room in its buffer to hold them.

Two application commands are defined for this purpose:

Application Hold ( )

Op-code:	13h
Data:	none.

Tell the sender to hold transmission until requested to resume.

Application Resume (count)

Op-code: 11h Data: 16-bit unsigned integer count of number of bytes device is ready to accept.

Tell sender it may resume transmitting data up to `count' characters. If `count' is zero or omitted, the sender may continue transmitting until requested to hold.

By using a non-zero count in the AppResume command, a receiving device need not transmit an urgent message to the sender when its buffer is full.

Notes:

1. AppHold and AppResume only affect device data stream messages. All ControVStatus commands must continue to be processed regardless of the held state.

2. When a device is reset it has no way of knowing the previous flow control state. After being reset and the configuration process is complete, a device using flow control shall transmit a single AppResume command and clear its transmit held state (if any).

6.1.4 Serial Asynchronous Communication Parameters

The Text Device Protocol defines the following commands for interfacing with common UARTs.

Application Set Format (controlmask)

OpCode:	01
Data:	16-bit controlmask

The control mask specifies UART parameters as follows. Choices listed in parenthesis are selected in order (0, 1, 2, . . .).

b0-b3	ransmit speed (default, 19200, 9600, 4800, 2400, 1200, 600, 300)
b4-b7	receive speed (rx=tx, 19200, 9600, 4800, 2400, 1200, 600, 300)
b8	word size (8-bits, 7-bits)
b9	stop bits (one stop bit, two stop bits)
bl0-bl1	parity (none, even, odd)
bl2	local echo (FDX, HDX)
bl3-bl4	flow control (none, XON/XOFF, DSR/DTR, RTS/CTS)

Application Request Format ( )

Op-code:	02
Data:	none

Application Request Format is sent from the host to a device and requests the device respond with an Application Report Format message.

Application Report Format (controlmask)

Op-code:	03
Data:	16-bit control mask, same as Application Set Format

6.1.5 Serial Asynchronous Control Signals

The Text Device Protocol defines the following commands for interfacing with common RS232 control signals.

Application Set Control (controlmask)

Op-code:	04
Data:	16-bit control mask

The control mask identifies the control signals to be set as follows (DCE side):

b0	DSR
bl	DTR
b2	RTS
b3	CTS
b4	RLSD (or CD)
b5	FE (framing error)
b6	OE (overrun error)
b7	PE (parity error)
b8	Break detected

b9	Send break
bl0-11	reserved
bl2-15	available for application use

Application Reset Control (controlmask)

Op-code:	05
Data:	16-bit control mask.

The control number identifies the control signals to be reset using the same definitions as Application Set Control.

Application Request Control ( )

Op-code:	06
Data:	none

Application Request Control is sent from the host to a device and requests the device respond with an Application Report Control message.

Application Report Control (controlmask)

Op-code:	07
Data:	16-bit control mask

Sent by a device to the host when an input control signal changes or in response to a request. The controlmask reports the status of UART control signals using the same definitions as Application Set Control.

6.1.6 Direction Control

The Text Device Protocol defines the following command for controlling the direction of bi-modal interfaces.

Application Set Direction (controlmask)

Op-code:	08
Data:	16-bit control mask

Sent by the host computer to a device to specify the direction of interface operation.

bO-bl	Direction (transmit, receive, full duplex)
b2-b3	Rank (Default, Master, Slave, Off-line) Examples of Master: RS-232 DTE Host centronics port Dialing FAX Examples of Slave: RS-232 DCE Printer centronics port Answering FAX
b4-b7	Reserved
b8-b15	Available for application use

6.2 Capabilities Information

The keywords defined in this section have standard meanings within the ACCESS.bus Generic Text Device Protocol.

Keyword	Meaning
Charset()	Tags the default character set used to encode text data. The following names are defined: ASCII, I50:8859/1
Direction()	Tags whether device can be used for data input, data output, or half-duplex, or full-duplex. Direction(input output HDX FDX)
FlowControl()	Tags whether low level flow control is used for input and/or output (AppHold and AppResume). FlowControl(input output)
UARTformat()	Lists common UART format selections that are supported and can be modified or examined by Application Format commands. UARTformat(speed, size, parity, stop, echo, flow(0 1 2 3)) The sublist of numbers, if present, indicates which selections are supported.
UARTcontrol()	Lists common modem control signals that are supported and can be modified or examined by Application Control commands. UARTcontrol(DSR DTR RTS CTS RLSD FE OE PE REAK)

SECTION 7

ACCESS.bus

Mechanical Drawings

SECTION 7

ACCESS.bus

Mechanical Drawings

February 1994

ACCESS.bus Industry Group
370 Altair Way, Suite 215
Sunnyvale, California 94086

Telephone: 1-408-991-3517
FAX:1-408-991-3773

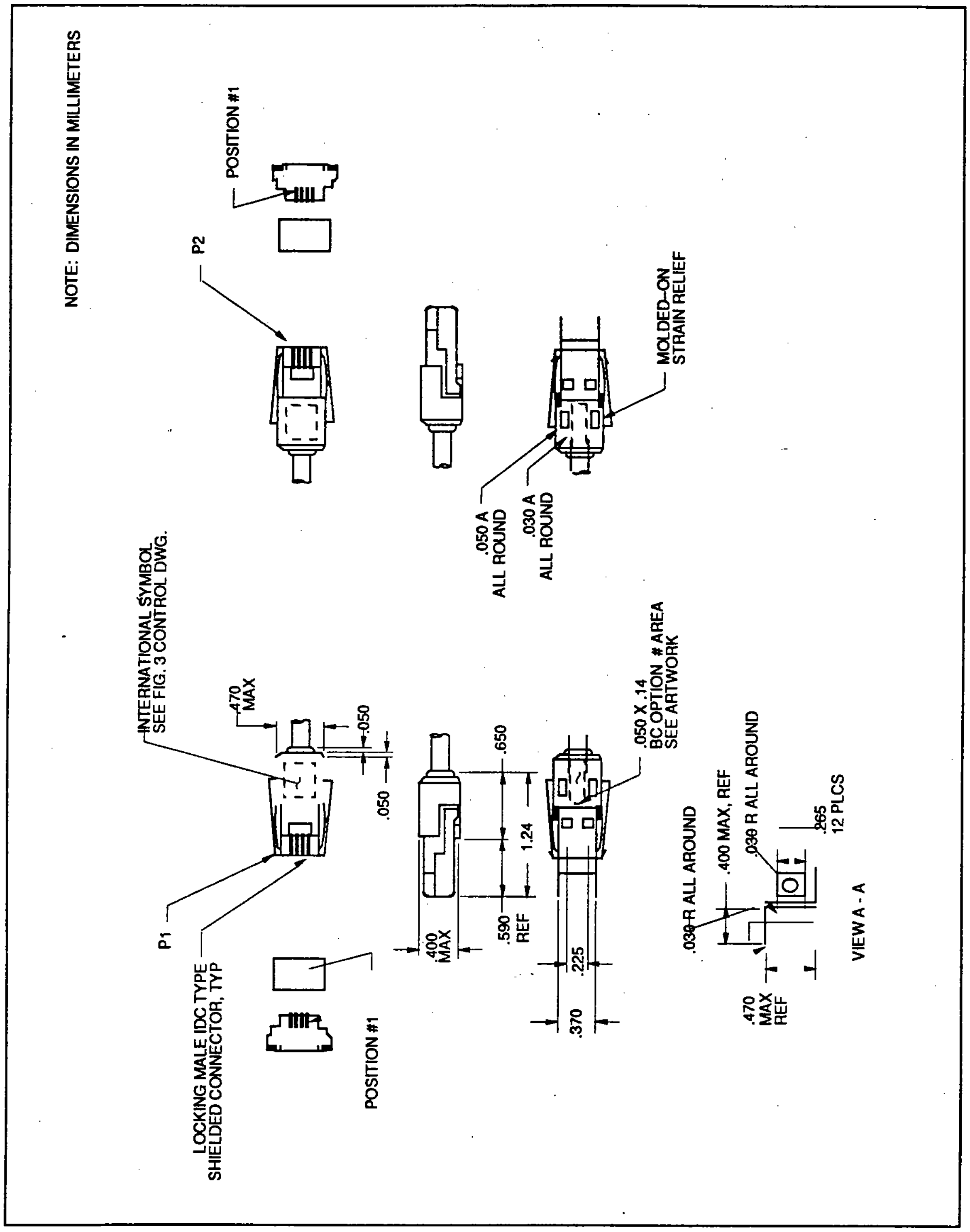

Figure 7.3: Cable Assembly - Configuration/Dimensions

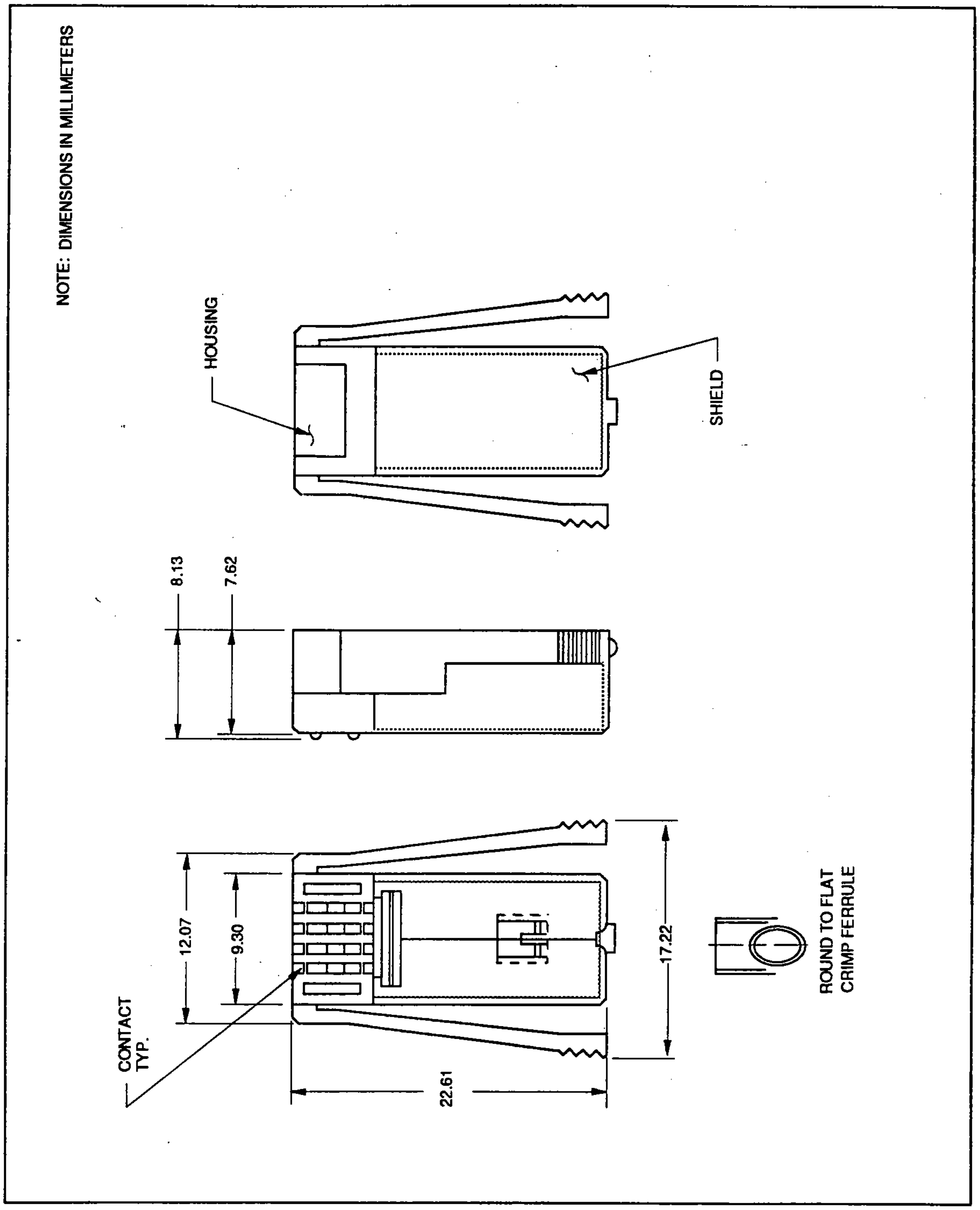

Figure 7.4: Male Connector - Configuration/Dimensions

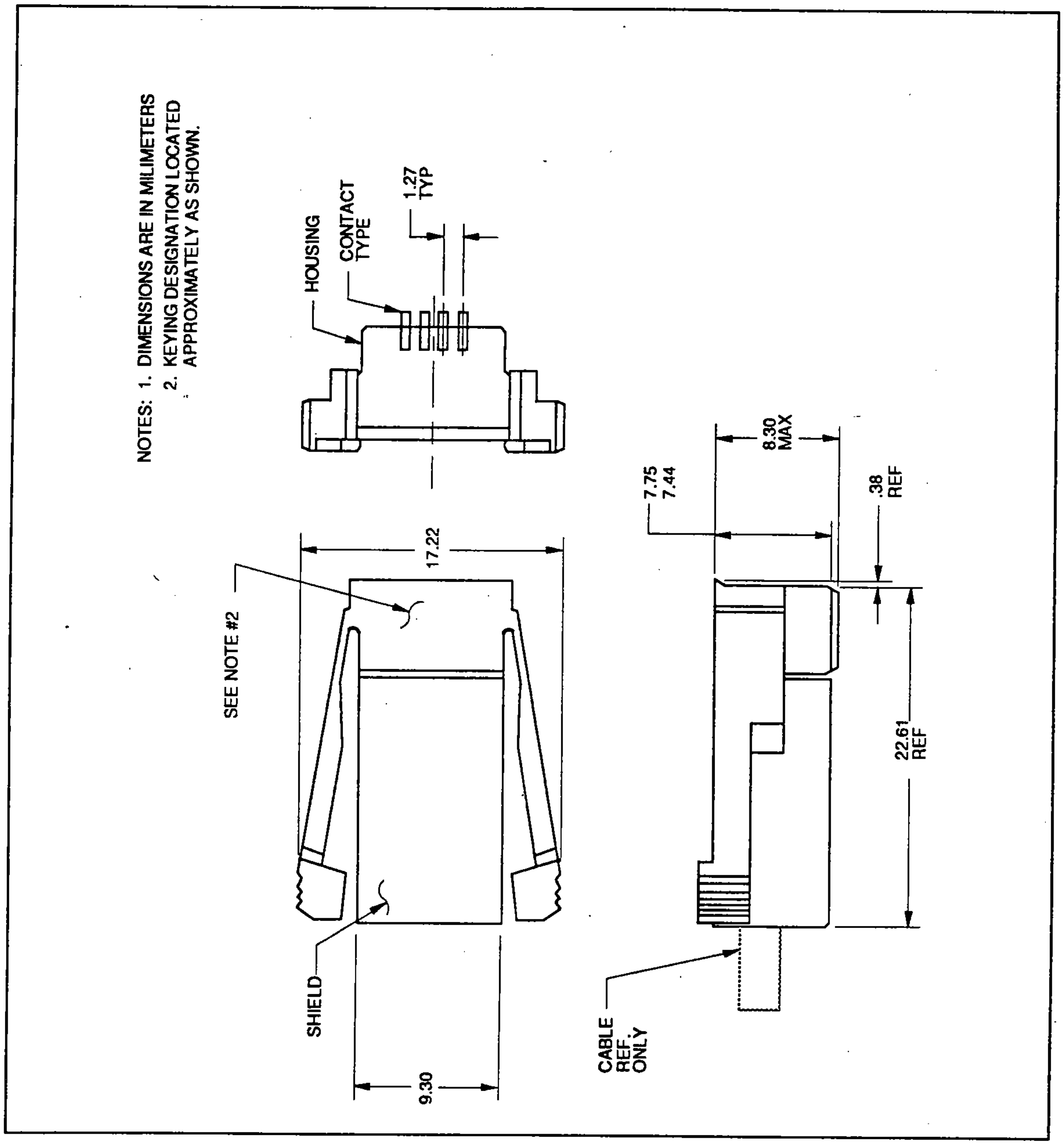

Figure 7.5: Male Connector - Configuration/Dimensions

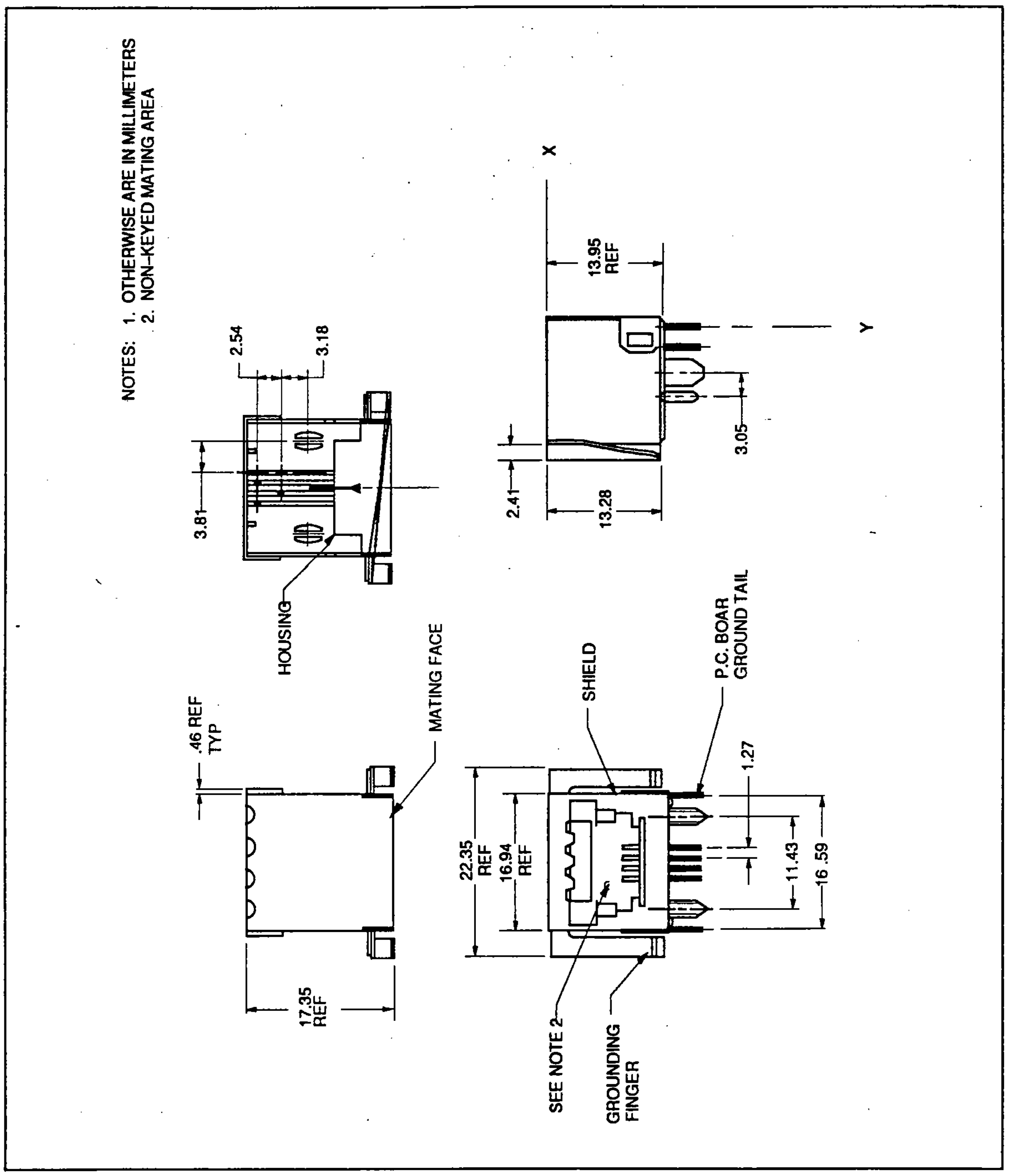

Figure 7.6: Female Connector (non-keyed shielded) Configuration/Dimensions

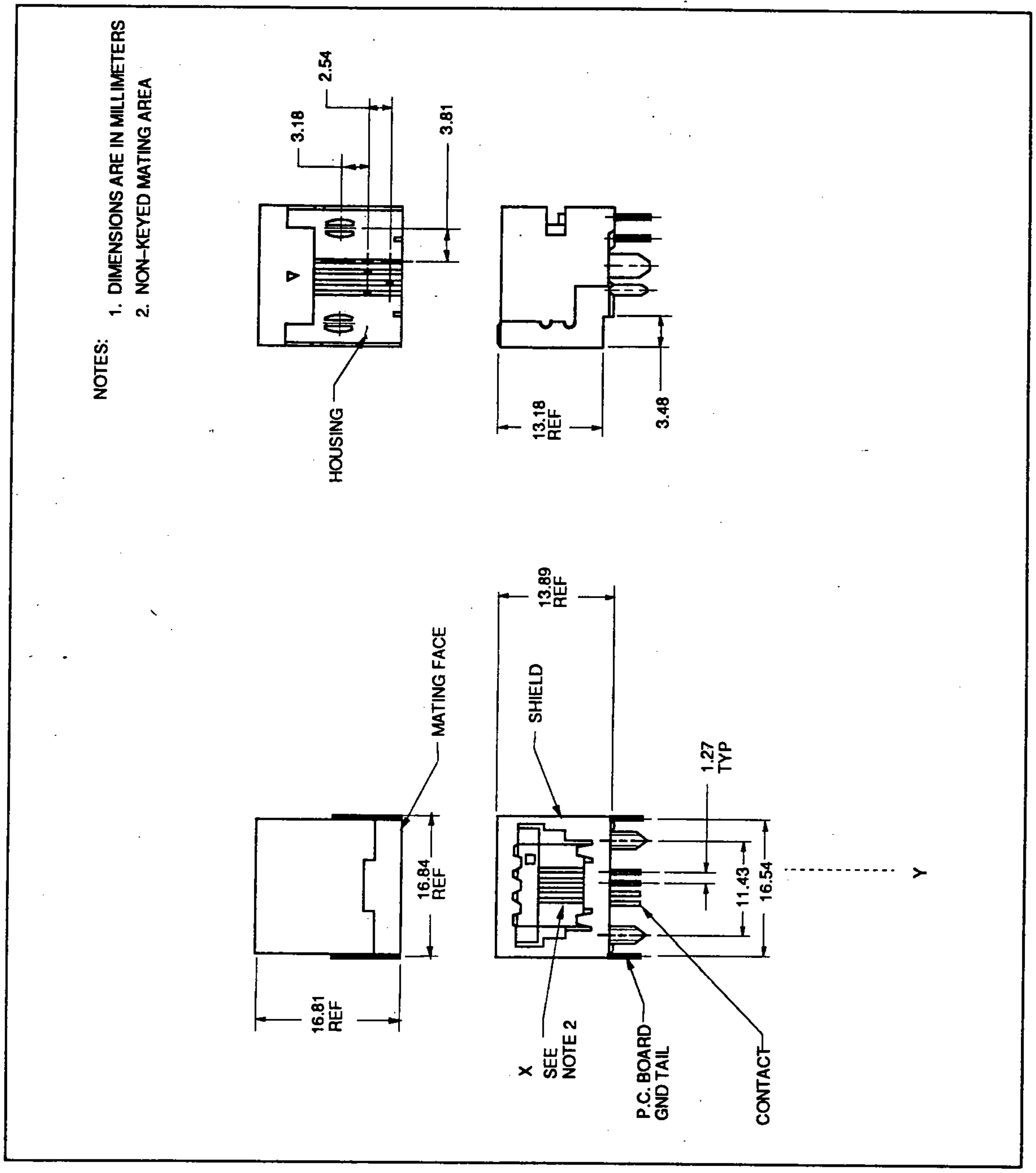

Figure 7.7: Female Connector (non-keyed partially shielded) Configuration/Dimensions