1 Abstract

With the recent introduction of the ACCESS.bus product, Digital has affirmed its commitment to open systems and thus to facilitating better solutions for interactive computing. This open desktop bus provides a simple, uniform way to link a desktop computer to as many as 14 low-speed I/O devices such as a keyboard, mouse, tablet, or three-dimensional tracker. ACCESS.bus features a 100-kilobit-per-second maximum data rate, hardware arbitration, dynamic reconfiguration, a mature capabilities grammar to support generic device drivers, and off-the-shelf, low-cost I^2C microcontroller technology. [The BUS paper begins here.]

As the cost of personal interactive computing decreases, the range of applications and the need for specialized I/O devices is growing dramatically. Traditional personal computers were designed to accept only a small number of standard devices; adding devices beyond those originally envisioned usually requires specialized hardware or software. This custom interfacing is expensive for both vendors and users and thus limits the availability of new devices.

ACCESS.bus provides a simple, uniform way to link a desktop computer to a number of low-speed I/O devices such as a keyboard, a mouse, a tablet, or a three-dimensional (3-D) tracker. Designed from the beginning as an open desktop bus, ACCESS.bus facilitates cooperative solutions using equipment from different vendors. This paper describes the ACCESS.bus design and gives some insight into how the idea was adopted at Digital.

2 Design Goal, Process, and Advantages

The design goal for the desktop bus follows from our experience within the Video, Image and Print Systems (VIPS) Input Device Group with trying to support new devices on Digital terminals and workstations. While various new devices have been successfully prototyped over the years, the need for nonstandard hardware and custom software drivers was always an expensive, time-consuming obstacle. Even after successful prototyping, these devices could not be readily adapted to our standard systems, limiting their use to custom applications. In designing the desktop bus, our goal was to make it as easy as possible to interface previously unavailable I/O devices to our systems in a way that was both practical and marketable. This section explains the benefits of using a desktop bus, describes the process we went through to convert to a new bus architecture, and summarizes the key advantages of the chosen design.

The basic desktop bus concept is illustrated in Figure 1. The bus allows multiple, low-speed I/O devices to be interconnected and thus interfaced through a single host port. Desktop bus devices such as a keyboard or a tablet, which are not hand-held, provide two connectors and allow another device to be daisychained. A hand-held device such as a mouse can be placed at the end of the daisychain, or a connector expansion box can be attached to accommodate additional devices that do not provide two connectors.

The desktop bus has the following benefits:

- Enables greater flexibility and variety of use
- Reduces the cost of connecting multiple devices
- Expedites bringing new technology to market
- Helps leverage third-party devices

The first benefit, greater flexibility, can be simply achieved by allowing additional devices and more modular solutions. We further extended this benefit by designing a way for devices to be added at run time without disrupting system operation. Configuration should be automatic; connecting standard devices should not require powering down or rebooting the system before a new device can be used. The desktop bus supports multiple like devices without switches or jumpers.

The second benefit, reduced cost, was crucial to having the bus accepted as a solution across a wide range of products from low-end video terminals to high-end workstations. We recognized that contemporary electrical techniques could eliminate the need for level translation circuits, -12 volt (V) power supplies, and perhaps some of the protective components used with RS-232 interfacing. Although many devices would now require two connectors, system cost would decrease because we would need to supply only as many connectors as the number of devices to be attached, or possibly one more.

The third benefit, expediting the time to market for new technology, allows us to better satisfy changing requirements. Key to this benefit is having the means to connect new devices without changing the system hardware or software. Based on our experience with input devices, we developed the concept of device capability reporting and generic device protocols. Standard devices like keyboards and locators, e.g., mice, tablets, and trackballs, all work in similar ways. For this class of device, we define a simple device protocol and a way to parameterize and report device unique characteristics. A single generic driver can adapt itself to work with a class of similar devices so that no custom software is required for basic operation of standard devices.

Leveraging third-party devices, the fourth benefit, is aimed at satisfying diverse customer requirements. Because the use of computers continues to proliferate, the range of applications far exceeds that which any one vendor can master. By making the

bus truly open, we encourage third parties to add value to our systems.

The benefits of a desktop bus are significant. But converting to a new architecture, especially one that is not backward compatible, is expensive in terms of the time and effort required. How does a large corporation build agreement to make such an investment decision? The desktop bus project started as a grass roots engineering effort and gradually built momentum. The process was one of dialogue to attract partners. Initially, three groups with slightly different objectives worked together to develop the bus. The visibility of separate groups jointly supporting the bus concept was essential to transform the idea into action. People are more willing to accept an idea that others around them have already adopted.

The three groups that initiated the desktop bus project were our VIPS Input Device Group in Westford, MA, mentioned previously; the Workstation Systems Engineering (WSE) Group, located in Palo Alto, CA; and the Video Advanced Development (A/D) Group in Albuquerque, NM. Our Input Device Group was looking for ways to simplify the process of prototyping specialized input devices and of getting related software support for our video terminals and workstations. WSE was developing a low-cost, personal workstation and needed a flexible way to support multiple input devices without greatly increasing the cost of the base workstation. The Albuquerque A/D Group had been experimenting with next generation I/O devices, i.e., force-feedback joystick, 3-D tracker, and real-time audio and video, and was interested in having these technologies adopted by other Digital groups. This A/D Group had used I^2C technology successfully in one of its previous video projects.

In January of 1990, engineers from each group realized they were working on similar problems and began to collaborate. The WSE Group was to build the desktop bus host interface and software drivers into their workstation; the VIPS Group was to help define the device protocols and supply desktop bus keyboards and

mice; and the Albuquerque A/D Group was to support bus development and prototype additional devices. Within four months, VIPS had defined the basic protocols and could demonstrate a working I^2C keyboard and mouse. These early prototypes helped persuade WSE to support the project and, in turn, helped reinforce the importance of the project to the VIPS Group.

We began presenting the desktop bus idea to interested groups within Digital and received many useful suggestions including

- Use the same keycodes as on the LK201 keyboard to eliminate the need to rewrite keyboard lookup tables.
- Store the country keyboard variation inside the keyboard so users will not need to enter it manually.
- Keep the devices simple, without modes.

In addition, third-party input device vendors made the following suggestions.

- Use a modular connector that is easy to plug and unplug correctly.
- Provide enough power for several additional devices.
- Allow vendors to supply their own device drivers; tuning their own device drivers is part of the value added by the vendor.

The bus idea was elegant and generally well received. Most of the reservations centered around the likely impact on existing system components, the current problems, and whether conversion to the bus was feasible. Because we recognized that other groups were facing tight development schedules, we did not pressure these groups to support our desktop bus work. We presented the desktop bus as a possible solution to interface problems, made our design information available, and worked to incorporate suggestions. But as the development work progressed, more partners supported our effort.

Once we decided to use a desktop bus, we looked at available designs, including the Apple DeskTop Bus, the Musical Instrument Digital Interface (MIDI), and serial buses offered by other semiconductor vendors, and evaluated these alternatives with respect to our design goal. Key advantages of the design chosen, i.e., the ACCESS.bus, are

- Off-the-shelf interintegrated circuit (I^2C) microcontroller technology with 100-kilobit-per-second (kb/s) maximum data rate. This technology is low-cost, yet fast enough for sophisticated input devices like a 3-D tracker.

- Built-in hardware arbitration, which simplifies the software and allows reliable communication without inventing a new protocol.

- Dynamic reconfiguration. The hardware and software allow bus devices to be "hot-plugged" and used immediately, without restarting the system. The devices are recognized automatically and assigned unique addresses. This advantage results in a plug-and-play user interface.

- A mature capabilities grammar to support generic device drivers. An extensible free-form grammar allows devices to describe their characteristics to a generic driver. Most common devices can work with standard drivers.

Bus or network interconnection has become widely accepted as a means of providing flexible open solutions. To appreciate ACCESS.bus, it is helpful to position its performance capabilities with respect to those of other network interconnect technologies, as shown in Table 1.

Table 1: Network Interconnects

Bus Type	Order of Magnitude Performance (kilobits per second)

Table 1 (Cont.): Network Interconnects

Apple DeskTop Bus, ACCESS.bus	10-100
LocalTalk	100-1,000
Ethernet	1,000-10,000
FDDI	10,000-100,000

At first glance, the 100-kb/s speed of the ACCESS.bus may seem adequate for large desktop devices like printers and modems. But these devices can transmit long data streams independent of any user activity and, if not restricted, could compromise the interactive performance of the bus. Thus, ACCESS.bus is intended for low-speed activities that people perform with their hands and is fast enough to handle multiple interactive devices like a keyboard, mouse, or 3-D tracker.

3 Hardware Description

Before discussing the ACCESS.bus design, we present a description of the Philips I²C technology upon which the design is based. Details of the specific ACCESS.bus implementation follow.

Integrate Circuit Fundamentals

ACCESS.bus extends the Philips I²C bus to operate off-board and, thus, connect desktop devices. The I²C is a two-wire serial clock and serial data open-collector bus. An open-collector design means that the clock and data lines are normally in a high-impedance floating state and are pulled up to a logical high state.

A device that wants to send a message waits for any message frame in progress to complete, then asserts a START signal to become bus master and begins to generate data and clock signals. The bus clock is synchronized among all devices by its wired AND connection. Each device, whether transmitting or receiving, stretches the low period of the clock until ready for the next bit to be transferred. When the last device is ready, the bus clock is allowed to go high, generating a rising edge on the serial clock. At this time, all active devices sense the state of the bus data line. For a receiving device, the state represents the received data bit. For a transmitting device, the state determines whether the device has successfully asserted its data on the bus. A transmitter that is sending a logical high state and detects that the data line is being held low by another sender, recognizes that it has lost arbitration and must try again later. When a "collision" or arbitration occurs, no data is lost, one message is transmitted and received, and the remaining messages must be sent again.

I²C data messages are transmitted as 8-bit bytes, with each byte being acknowledged by a ninth ACKNOWLEDGE bit from the receiver. I²C technology also defines unique START and STOP signals to delimit message frames. The first byte of any message frame is always the destination address.

ACCESS.bus Physical Implementation

Details of the physical implementation of ACCESS.bus are as follows:

- Basic electrical configuration. ACCESS.bus uses four-pin, shielded, modular-type connectors that feature positive orientation and locking tabs. Data and power for the bus are transmitted over low-capacitance, four-wire, shielded cable. The four conductors are used for ground, serial data, serial clock, and +12 V.

- Available power. The maximum available power for all devices is 12 V at 500 milliamperes (mA). ACCESS.bus devices may supply their own power from a separate source, if needed. A power-up reset circuit must still be provided to reset the device when bus power is applied.

- Cable length. The maximum cable length for the entire bus is 8 meters. The limiting factor is a maximum capacitance not to exceed 700 picofarads (pF).

- Number of devices. The maximum number of ACCESS.bus devices allowed on the bus is 14. Limiting factors are the device addressing range and the power distribution (a total of 500 mA for all devices).

- Hardware interfaces. ACCESS.bus hardware interfaces are implemented using standard I^2C microcontrollers developed by the Signetics Company or under license from Philips Corporation. (Signetics Company is a division of North American Philips Corporation.)

4 ACCESS.bus PROTOCOL

Every device on the bus is a microcontroller with an I^2C interface and behaves as either a master transmitter or a slave receiver, exclusively, as defined by the I^2C Bus Specification.

Message Format

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.

ACCESS.bus messages have the following format:

Byte Number	Bit Number [1 2 3 4 5 6 7 8]
1	[destaddr |0] Destination address
2	[srcaddr |0] Source address
3	[P| length] Protocol flag, length (the number of data bytes from 0 to 127)
4 through (length + 3)	[body] Consists of 0 to 127 data bytes
length + 4	[checksum]

Initially, devices respond to a default power-up address. During the configuration process, the computer assigns a unique address to every device on the bus. Messages are either device data stream (P=0) or control/status (P=1), as indicated by the protocol flag. The minimum length of a message is 4 bytes; the maximum length is 131 bytes (127 data bytes and 4 bytes for overhead). The message checksum is computed as the logical XOR of all previous bytes, including the message address.

Standard Messages

The ACCESS.bus protocol defines the seven standard interface messages summarized in Table 2. Parameters defined within the body of the message are listed in parentheses.

Table 2: Standard ACCESS.bus Protocol Messages

Computer-to-device Messages	Purpose
Reset ()	Force device to power-up state and default I^2C address.
Identification Request ()	Ask device for its "identification string."
Assign Address (identification string, new address)	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."
Device-to-computer Messages	
Attention (status)	Inform computer that a device has finished its power-up/reset test and needs to be configured; "status" is the test result.
Identification Reply (identification string)	Reply to Identification Request with device's unique "identification string."

Table 2 (Cont.): Standard ACCESS.bus Protocol Messages

Capabilities Reply (offset, data fragment)	Reply to Capabilities Request with "data fragment," a fragment of the device's capabilities string; the computer uses "offset" to reassemble fragments.

Identification

Since the ACCESS.bus is a bus-topology network, unique identification strings are used to distinguish devices. These strings are structured as follows:

```
protocol revision:
  1 byte   (e.g., "A")
module revision:
  7 bytes  (e.g., "X1.3   ")
vendor name:
  8 bytes  (e.g., "DEC     ")
module name:
  8 bytes  (e.g., "LK501   ")
device number:
  32-bit signed integer
```

The module revision, vendor name, and module name strings are left-justified ASCII character strings padded with spaces. The device number string is a 32-bit two's complement signed integer and may be either a random number (if negative) or a unique serial number (if positive).

5 Configuration Process

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

Power-up/Reset Phase

When reset or powered up, a device always reverts to the default address and sends an Attention message to alert the computer to its presence. At system start-up or reinitialization, the computer sends a Reset message to all I²C addresses in the ACCESS.bus device address range (14 messages) to ensure that all devices on the bus respond at the power-up default address.

Identification Phase

To begin address assignment, the computer sends an Identification 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 I²C 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. When a device receives this message and finds a complete match with the identification string, it moves its device address to the new assigned value. As soon as a device has a unique address, it is allowed to send data to the computer.

The I²C physical 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 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 host to inventory which devices are present.

Capabilities Phase

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 is an ASCII string constructed from a simple, readable grammar. The grammar allows text strings to be formed into lists, nested lists, and lists with tagged elements. The capabilities string for a locator might read as follows:

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

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 is also made available to application programs using the device.

Normal Operation

During normal operation, the computer periodically requests inactive devices to identify themselves. If a device is found to be missing, or a new device appears by sending an Attention message at the default address, the computer sends an Identification Request message to each device address previously recorded as in use (up to 14 messages) to confirm which devices are still present. The computer also sends a Reset message to each device address previously recorded as not in use. The computer then begins the address assignment process by sending an Identification message to the default address and assigning each device that responds to an unused device address.

6 Generic Device Concepts

ACCESS.bus uses the concept of generic device drivers to support familiar I/O devices using only a few drivers. Generic specifications for keyboards, locators, and text devices have been developed.

Keyboards

The keyboard device protocol attempts to define the simplest set of functions from which a Digital LK201 or a common personal computer keyboard user interface can be built. A generic keyboard consists of an array of key stations assigned numbers between 8 and 255. When any key station transitions between open and closed, the entire list of key stations currently closed or depressed is transmitted to the host. This reporting scheme is functionally complete; the host can detect every key transition, and the scheme provides information about the full state of the keyboard on each report. No special resynchronization reports are required.

In addition to reporting key stations, the generic keyboard device can support simple feedback mechanisms such as keyclicks, 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 table can also be stored in the keyboard itself as part of the capabilities string.

Locators

The locator device 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.

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., two, for a mouse or a tablet
- Dimension type: absolute, i.e., referenced to some fixed origin, like a tablet; or relative, i.e., changed 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.

Text Devices

The text device protocol is intended to provide a simple way to transmit character data to and from character devices such as a bar code reader or a small character display. 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.

7 Summary

The ACCESS.bus network interconnect offers the possibility of a standardized, low-speed, plug-and-play serial communications channel that can untangle peripheral interfacing and open the way to new applications. As the advantages of this open desktop bus design become well known, we expect wider adoption of this product. The ACCESS.bus is currently implemented on Digital's Personal DECstation 5000 workstation, with implementations underway for the next generation of RISC workstations and video terminals.

8 Acknowledgements

Many people contributed to the design and development of ACCESS.bus. I would especially like to acknowledge Tom Stockebrand and Tom Furlong for their vision and early support; Chris Cued, Mark Shepard, and Ernie Souliere for their contributions to the ACCESS.bus electrical design and protocols; and Robert Clemens for the excellent demonstration hardware and firmware development support.

9 General References

D. Lieberman, "Desktop Bus is Born Free," Electronic Engineering Times (September 2, 1991): 16.

ACCESS.bus Developer's Kit (Palo Alto, CA: Digital Equipment Corporation, Workstation Systems Engineering TRI/ADD Program, 1991).

Signetics I²C Bus Specification (Sunnyvale, CA: Signetics Company, a Division of North American Philips Corporation, February 1987).

Peter A. Sichel As a principal software engineer in the Video Terminals Architecture Group, Peter Sichel led the development of the ACCESS.bus architecture and device protocol specifications, in addition to writing the initial ACCESS.bus device firmware. He worked on the VT420 video terminal and the DECterm DECwindows terminal emulator, and helps maintain Digital standards for video terminals and keyboards. Peter joined Digital in 1981 after receiving B.S. and M.S. degrees in computer engineering from the University of Michigan.