The role of the Intelligent Peripheral Interface in systems architecture

By I. DAL ALLAN
Sperry Univac
Santa Clara, California

ABSTRACT

The American National Standards Institute (ANSI) work committees on computer interfaces are actively developing proposed standards that will affect almost all vendors of computers and peripherals. This paper reviews one project, the Intelligent Peripheral Interface (IPI), in particular, and references the application area of the Small Computer Systems Interface (SCSI).

The architecture of future computer systems is going to depend in large part on the types of interfaces used, because of the ever-increasing role that peripherals play in systems performance, cost, availability, and reliability. The capability of peripherals dictates system performance—more than the central processor does, in most configurations. The cost of peripherals represents more than half of what the user pays for a system.

The interface chosen to interconnect peripherals has a major influence on the cost and performance tradeoffs that a vendor can make. This paper addresses these issues with regard to the ANSI efforts on intelligent interfaces.
The Small Computer Systems Interface (SCSI) and the Intelligent Peripheral Interface (IPI) are being defined as proposed American National Standards Institute (ANSI) standards by the ANS committees X3T9.2 and X3T9.3, respectively. Every industry representative has a voice in how these interfaces are defined, because ANSI standards are defined by industry volunteers who attend the meetings and contribute.

In the U.S., both the definition of and compliance with an ANSI standard are voluntary, and usually the result of market pressures. This is not necessarily true overseas, where standards may be set by the governments, and compliance with them is a requirement for all vendors. The standards defined by ISO are often adopted in their entirety, so it is quite feasible for a voluntary ANSI standard to become an ISO standard and wind up as a regulated requirement in foreign markets.

The availability of microprocessors and large-scale integration (LSI), both commercial and custom, has changed the way in which systems can be configured. The inexpensive micro makes it feasible to incorporate logic that formerly resided in the controller into the device, and to place in the controller logic that formerly resided in the operating system. The definition of controller varies by market environment, and is commonly known by different designations, such as storage director (SD) on mainframes and the direct memory adapter (DMA) on minis.

There are three general definitions of interfaces—channel, control, and device. Usually, the three only occur as separate and discrete interfaces on large mainframe configurations. In most mini configurations such as that illustrated in Figure 1, the functions of the channel and control interfaces are integrated onto the system bus. The device interface (which is most commonly an SMD) connects directly to a DMA controller that shares the CPU bus with other system elements.

This type of implementation has made DMA controllers, and subsequently the processor complex, very sensitive to technology changes in devices. Either every new device has to act just like the previous, or it causes some level of change in the DMA controller. The control interface, which connects to a head of string (SC—string controller), is used to mask unique device dependencies.

The minicomputer world is turning toward the string concept (which has been much used by mainframes), because it addresses the general market desire for a device-independent interface. Figure 2 illustrates both a mainframe and a mini with heads of string. The head of string may stand alone or be integrated into the first device, with others attached to it via a device interface. Alternatively, each device may have an integrated head of string so that there is a one-to-one relationship.

In covering the application areas for a peripheral interface it is first necessary to look at the characteristics of each of the three types of interface:

1. Device interface—This is usually one of close proximity, typically less than 50 feet. The transfer rate across the interface is dictated by the speed of the device (data clocking rate). The commands used across the interface are device dependent and both the timing constraints and the individual commands are uniquely device dependent.

2. Control interface—The distance is longer, and is typically in the range of 50 to 200 feet, depending on the size of the configuration. Data transfer rates are a function of where the buffering is located; that is, if buffering is done at the CPU, then the rate is determined by the speed of the device because it is transferring data at the rate of the device. If buffering is incorporated in the head of string, then speed is a function of how fast the CPU chooses to accept data because speed matching is a func-
3. Channel interface—Distances are expected to be much longer, in the range of 200 to 400 feet or more. CPUs that use channels are fast enough that performance is either controller- or interface-limited. Commands are disconnected so that controllers may be multiplexed; it is desirable that they be function generic (i.e. random/sequential, read/write) to mask all device uniqueness.

In the past, as the functionality of an interface increased so too did its manufacturing cost. It was not practical to limit the number of different implementations, because the costs associated with each tended to vary dramatically. To the degree to which each interface varies, there are unique costs incurred, and the more there are the higher the levels of support in engineering, training, and spares that are required.

Today, more cost is associated with the development and maintenance of system components than with manufacturing cost. Life-cycle cost analysis proves that it is cheaper to use common parts than it is to optimize individual items (there are huge savings in spare parts alone). Interfaces have to become a generic system component in order to achieve similar life-cycle cost savings.

The IPI is a modular interface that is layered in an architecture similar to that of the Open Systems Interconnect (OSI) model. It is not a proprietary interface; instead, it represents a combination of the best features of all the interfaces that were proposed to the committee.

Since terminology does not have the same meaning to everyone, the proposed standard has defined several terms to avoid confusion; for example, in an intelligent environment the term host is used to identify the master and the term unit is used to define the slave (which controls devices). In a device-oriented environment the term unit defines the master and the device is the slave. Both the physical interface and the logical interface definitions of the IPI contain a glossary of terms and definitions.

It is practical to use the IPI as a channel, control, or device interface, depending on the repertoire of commands used. The master-slave configuration is used to obtain simplicity and performance, and there are two major divisions: The physical interface defines the interconnection for transferring information; the logical interface discriminates between information as either the operations to be executed or the data to be transferred.

The two divisions operate independently of each other, in the sense that it is possible to replace one physical interface with another and have the logical interface remain the same (e.g. open-emitter logic using coaxial cable for long distances may be replaced with a fiber optics loop). Alternatively, it is possible to retain the same physical interface and replace or extend the logical interface command repertoire to provide a database backend processor in place of a disk controller.

The physical interface defines the mechanical, electrical, and protocol specifications necessary for transferring information across the IPI. The multiple interconnect specifications listed in Table I are provided so that cost considerations of

<table>
<thead>
<tr>
<th>DISTANCE</th>
<th>ELECTRONICS</th>
<th>MECHANICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3 METERS</td>
<td>THREE STATE</td>
</tr>
<tr>
<td>B</td>
<td>15 METERS</td>
<td>OPEN COLLECTOR</td>
</tr>
<tr>
<td>C</td>
<td>65 METERS</td>
<td>DIFFERENTIAL</td>
</tr>
<tr>
<td>D</td>
<td>5 METERS</td>
<td>OPEN Emitter</td>
</tr>
<tr>
<td>E</td>
<td>125 METERS</td>
<td>Emitter</td>
</tr>
</tbody>
</table>

Table I—Interconnect classes

in-cabinet harnesses versus external cabling can be managed by the manufacturer.

Different applications need different physical interconnections, thus the choice between logic families and cable configurations. Class A is suited to small systems and uses the same logic, connectors, and cables as defined in the rigid disk interface (BSR X3.101M). These limit cable lengths to 3 meters, are cheap, and are quite suitable for in-cabinet configurations. At the other extreme is the requirement for large mainframe complexes that place subsystems hundreds of feet away from the central-processor complex. Industry has a diverse range of requirements, so a standard must not limit the environments where it can be used.

Intermixing of the different classes of physical interconnects is likely to occur often. Classes C and D have the same electrical specifications but use different cables and connectors—for mainframe installations Class D is suitable for the long cable lengths between system controller and peripheral-subsystem strings, but within a string Class C would be used to save cost. As another application, a high-capacity disk subsystem using Class D to the CPU could incorporate a small cartridge tape in each cabinet to serve as an audit trail by logging activity against the disk. The interconnect to the tape cartridge would be internal to the disk cabinet and able to use Class A.

As seen in Figure 3, 5 signals (three out and two in) are used to control the interface. One signal change at a time is alter-
nated between master and slave to provide complete hand-
shake control of the interface. There are two unidirectional
8-bit buses with a transfer rate that exceeds 3MBs, and as a
performance option the two can be used bidirectionally at
over 6MBs. The remaining signal is used as an ORed attention
line to advise the master of interrupt conditions in the slaves.

Implementing the IPI requires but a simple state machine,
virtually free of timing dependencies. With 5 control signals
there are only 32 possible states. Of these, 12 are illegal and
6 are resets. That leaves only 14 states needed for normal
operation, and the 9 sequences require fewer than 30 transi-
tions between states.

The few signals, small number of states, and the limited
sequences reduce the size and complexity of logic necessary
to control the interface. Everything can be managed by a
micro except recognition of reset, a precaution necessary to
recover from microprocessor failure, microcode loops, bugs,
and so on.

Implementation techniques vary widely; one uses a Z8
micro and a handful of chips, one is using custom gate arrays,
and another uses the Signetics 50ns 825105 FPLS as a “state
machine in a chip.” Common to all is that they are cheap by
comparison with the interfaces they are replacing.

A microprocessor is not expected to handle the transfer
function, unless the rate involved is quite low. The slave
defines the transfer clock rate so that it can operate at the max-
imum rate speed of the device (or buffer, if any). Transfer rate
is a product of cable length, skew, and technique used. If
transfers between the master and the slave are interlocked, all
information is “handshaked” over the bus and there are de-
lays (in both directions) that are associated with cable length.

Data streaming may be incorporated in the intelligent
host-unit environment to double the transfer rate by not inter-
locking the syncs (data clocks). Information on the buses is
strobed by a sync (clock pulse), and there is no interlocked
handshake to confirm receipt of the pulse. This is similar to
the way most bit-serial device interfaces clock data into the
receiver. The effect of cable delay is that the interface con-
tains multiple clock and information pulses at the same time.

The host responds to each SYNC IN with a corresponding
response syncs are still “on the cable” and it is the responsibility
of the unit to check that there was the same number of syncs
in and out. The way in which data streaming has been imple-
mented in the IPI permits a host to operate transparently
between both interlocked and data streaming modes. If the
width of the SYNC IN pulse is greater than the cable delay,
the transfer is interlocked.

An important feature is the ability to intermix devices with
widely varying transfer rates on the same cable. Transfers
can be made in any combination of interlocked or data streaming,
8 bits or 16 bits wide. The capacity of up to eight units, each
with up to 16 devices, gives a maximum of 128 devices ad-
dressable by the host on a single daisy-chained cable.

As illustrated in Figure 4, functions on the IPI are com-
plementary to each other. At the physical interface a bus
exchange is used to frame an information transfer, which can
be data, an operation command, or an operation response.
The bus control defines the parameters (operation or data, in
or out, 8 or 16 bits wide) of the information transfer, and slave
status reports on its completion (success or failure).

The contents of an information transfer are transparent to
the physical interface, which is simply a vehicle for using the
IPI as a device, control, or channel interface. The logical
interface frames data transfers with an operation command
and its associated response. The “intelligence” of an IPI
implementation is a product of the command repertoire defined
between master and slave. The repertoire itself is a product of
the environment in which the IPI is used:

Master-controlled devices.—The master functions in a fash-
ion similar to that used with device interfaces. This type of
operation is presumed to offer little or no increase in func-
tionality over rigid disk interface (RDI) or storage module
drive (SMD) types of interfaces. Vendors that have devices
installed and operational with current software need to install
enhanced peripherals. It is much easier to introduce new
hardware into a system when there is no need for new soft-
ware. Software changes can be introduced gradually if the new
hardware incorporates new functionality.

It may be impossible to install new hardware on the old
interface; for example, disks presently available with a 3MBs
data rate (24 MHz) cannot be implemented with an SMD
interface. The IPI can be used to emulate the SMD command
set to the master so that no software need be changed. The
extended functionality offered by the IPI can be taken advan-
tage of in stages. For example, to improve serviceability the
first stage may be to provide new maintenance and diagnostics
capabilities. Operating system software changes usually take
longer to implement, but as they occur they can be made
without hardware impact.

Some uses of the IPI will be device and vendor specific if it
is necessary to extend the command repertoire beyond that
defined in the logical interface. The proposed standard does
provide for this kind of use as a vendor-unique application,
because such implementations provide the vendor with a
migration path to compliant logical interface operation.

One example of this kind of use is defined as synchronous
because it represents a time-critical operation executing be-
tween CPU and device. This situation can occur when the IPI
is used subservient to another interface such as the FIPS 60
Block Multiplexer Channel. The channel issues channel command words (CCWs) that are chained together and require responses within a very specific time period. On disks with count-key-data (CKD) records the channel must actually "turn around" during the gap times of the disk. If the CPU-SCU and SCU-disk interfaces are FIPS 60 and IPI respectively, the IPI must run synchronous to the timing constraints of the channel. Similar situations may occur whenever the IPI is dropped into any pre-existing environment.

Shared control of devices.—This is an intelligent environment where host-unit operation may have any of a wide range of characteristics—from one where the host dictates control of the devices through the unit, to a more independent mode of operation where the unit controls the devices. The command repertoire is oriented to device characteristics (disk, tape, etc.) and uses device-generic commands that support logical addressing (e.g., relative block) and a format (e.g., fixed sectors). This level of operation permits the unit to incorporate such features as buffers, error correction, error retry, and defect handling.

The command repertoire defined is extensive, because it incorporates pathways to let vendors migrate to more intelligent use of peripherals. The commands defined not only replicate those to be found in the SCSI (X3T9.2/SASI) and ISI (CDC/MP1), but represent a superset that permits application over a much more diverse and varied set of configurations and architectures. Since the objective is to permit the vendor to coexist with the installed environment there are no predefined characteristics that require buffering, error correction, error recovery, and so on in the unit—if they are present, the command repertoire can use them.

Different performance profiles can be configured—if commands are issued as individuals, only one operation per device is active at a time, and it has no timing-critical dependencies. If performance is a criterion, the unit may operate with queued commands. In this implementation the unit is capable of stacking multiple operations per device on behalf of the host, and of servicing them in a manner defined by host and/or unit algorithms. One example of this would be performance improvements on disks, obtained by the unit queuing requests and executing them in a sequence according to some seek-optimizing algorithm (unless overridden by a host-defined priority request).

Unit-controlled devices.—Operations are host-unit and require that a number of architectural attributes such as memory management reside in the units. The control of devices is distributed, since their management is within the scope of the unit; the host has limited control, if any, over the details of device manipulation. Either individual or queued operations are executed, depending on the capability of the unit.

At this level of operation few commands are needed, because each one is powerful and rich in functionality. The host is cognizant only of the attributes of the data (i.e., random, sequential, input, output; thus there are no device-dependent characteristics. The addressing structure does not recognize physical boundaries, since data is allocated in files on logical volumes, which may or may not be on a single physical device.

When execution occurs on the IPI for this type of implementation it is probably being used as a channel. The level of capability necessary in the unit represents a high level of functionality that is relatively high in cost. As such, it is assuming a considerable degree of the data manipulation burden of the host.

To support these variations in environment and application, the IPI has applied techniques implemented successfully in communications networks to support multiple environments of use. The logical interface uses message packets for operation commands and operation responses, with sequences to perform a task constructed from a series of macros. For example to read data in from a slave to the master requires a sequence of five macros:

```
SELECT
OPERATION COMMAND
READ DATA
OPERATION RESPONSE
DESELECT
```

Only simple operations are defined for device interface applications, but in the intelligent host-unit environments the complexity of the tasks that can be accomplished is limited only by the functionality incorporated in the unit. The operation commands and operation responses are message packets of variable length that are transferred across the physical interface as information. (See Figure 4.) A command packet has a generic root of 14 bytes plus, if needed, a variable-length parameter list. A response packet images back the generic root plus 4 bytes of status and, if needed, a variable-length parameter list. The commands are broken up into the following groups:

```
READ LOGICAL COMBINATION READ/WRITE
READ PHYSICAL CONTROL
WRITE LOGICAL DIAGNOSTICS
WRITE PHYSICAL OTHER
```

Some of the commands are generic to all uses (e.g. some control commands), but others are very specific (e.g. a physical read to a disk). The read and write physical commands are unique to the device interface level of operation. In most applications only a subset of the available command repertoire is needed.

It is unnecessary to define the ultimate environment for use of the IPI in advance. Since commands are assumed to be implemented in microcode, unique requirements can be added to the chosen repertoire to meet specific vendor needs and problems. Over time, as systems adapt to higher levels of function the unit will grow in complexity as it incorporates functions currently considered part of the operating system responsibility. The most fascinating part of such a scenario is that the same physical interface can support all of the envisaged environments, and the IPI can evolve to meet the vendor needs.

To assist in making this kind of growth feasible and modifiable, the IPI provides extensive housekeeping capabilities...
that permit the master to configure the slave to the required environment of use. Each slave has attributes that can be examined by the master—some, such as physical record size, will be soft on one device but hard on another (e.g., on embedded servo disks the physical record size cannot be altered). The combination of attributes offered by a slave defines the ways by which the master can customize it to the particular installation. Soft attributes can be selected and modified by the master to tailor the slave, while hard ones cannot be modified and must be accepted as given. Attributes will normally be established after initial power-on, but they can be used dynamically to reconfigure systems to adapt them to changing workloads or cover hardware outages.

Using the IPI as a device interface has been discussed already; it can represent major cost savings. Tapes are parallel, and all the new coding schemes (2/7, 3PM) for disk depend on byte constructs (data is recorded bit-serially, and the data separator does the necessary conversions). As shown in Figure 5, a device interface such as the SMD requires that it be converted back to bit serial to go to the outside world—where it is reconverted into bytes again by a head of string or a DMA controller. Given the increasing use being made of high-density coding schemes this makes no economic sense at all—the cheapest, and also the fastest, approach is to leave the data parallel from the data separator on, as can be done with the IPI.

The on-board electronics are not the only savings—there are even more significant economies in cabling costs. Bit-serial device interfaces are radial (Figure 1), and usually have two cables (the SMD has one for control and one for data). A parallel device interface requires only a single daisy-chained cable (Figure 7), and can operate at much higher data rates. When the cost of the separate cables and the difficulties of installation are addressed, the parallel IPI provides savings to both the manufacturer and the installer.

At the other end of the spectrum, the IPI is very suitable for use as a channel, because it provides the ability to operate under the FIPS 60-63 architectural constraints. A channel interface has to be able to interconnect over long distances, at a very high data rate. Cabling costs are very high, electromagnetic interference (EMI) is a serious concern, and grounding isolation is difficult to achieve. The latter concerns make fiber optics appear very attractive, but a fiber optics implementation of FIPS 60 requires three fibers and cannot be installed cost-effectively.

A single fiber loop or ring can be cost competitive with the current FIPS 60 type of coaxial-cable connections. The IPI has been designed with this consideration in mind, and a conversion of the physical interface such as that shown in Figure 6 has no impact upon the logical interface command repertoire implementation.

One of the biggest problems with EMI compliance is that even though cabinets may individually comply, when they are configured into a system the system may not comply. The cost of testing for compliance is a significant factor that all vendors of computing systems have to face as an ever-increasing burden. When fiber optics is used as an interface there are no intercabinet EMI considerations, which means that system compliance is a near-automatic follow-on after cabinet compliance has been achieved. The FCC and VDE regulations have caused considerable trauma in recent years, and unless there is a switch to fiber, future systems will represent an even worse problem.

The choice of how to implement an IPI is a function of cost and performance—on a small system with limited performance requirements, integrating both functions into the on-board processor makes the most sense. However, for higher-performance systems it is better to have a head of string on a separate board that uses a fast processor. The advantage to this structure is that the systems integrator only has to support one interface, and it is easy to offer simple performance upgrades from device-only configurations by adding heads of string boards.

One point that must not be overlooked in implementing the IPI is undersizing the microprocessor. The physical interface can be controlled by an inexpensive slow processor, but as the level of intelligence offered increases, so too does the need for faster processing. An IPI used to interface devices at a primitive command repertoire level does little processing, but as functionality is added the power to process commands at the logical interface must be provided—if it is not, the unit will take too long to respond to commands and will be limited in performance. This may be acceptable in some low-end appli-
cations, but where intelligent processing is needed, either a fast processor in every device or a fast processor at head of string is needed.

Figure 7 illustrates how a mainframe can configure the IPI as all three interfaces (channel, control, and device), a large mini can configure it as two (control and device), and a small mini can configure it as one (device). The configuration is made up as follows:

1. The mainframe CPU is connected by a channel interface to an SD controller.
2. The SD and a mini, via its DMA controller, are daisy-chained to two dual-port heads of string (one standalone and one integrated unit/device combination).
3. The standalone head of string and a small mini, via its DMA controller are daisychained to dual-port devices.

In the control or intelligent interface environment the X3T9.2 project for the SCSI can be considered an alternative to the IPI. The SCSI has an 8-bit bus and offers a peer-to-peer capability, such that units attached to it may assume the role of either initiator or target. There is an arbitration algorithm that settles possible conflicts in use on interfaces less than 30 meters in length (which is adequate for small systems).

There are implementations of the SCSI available in custom LSI, which makes the job of systems integration much easier for both the device and the controller manufacturers. Not only is the physical interface integrated but so too is the basic command repertoire. This makes for a high degree of applicability and compatibility in future SCSI installations.

The SCSI can coexist with the IPI because both use a structured, modular approach to implementation and are oriented to use in new systems applications where there is no need for compatibility with existing software and hardware. Equivalence between SCSI and IPI configurations can be obtained by using an SCSI-like logical interface command repertoire. An SCSI complex could be configured with one node dedicated to high performance disk activity that uses IPI interfaces to the disks. The operating system software in such a complex would be almost totally unaware that it was operating with two different physical interfaces except at the I/O dispatcher level.

When a vendor has to configure systems where multiple levels of interface must be supported, the IPI is unique in its portability of function. It is possible to use the same physical interface as device, control, and channel with the only difference being the command repertoires at each. Even more likely is that the same physical interface will be used with more than one level of logical interface capability. As new peripherals and software are introduced, the command repertoire can expand and grow to suit higher levels of functionality. In this way the IPI serves as a migration and growth vehicle for vendors.

There is no equivalent interface offered by any vendor that offers quite the same breadth of application. The IPI represents the essence of simplicity, and very little has been compromised for its application in any of the three interface definitions. This is a testimony to the efforts and capabilities of the ANSC X3T9.3 membership, who have given generously of their time and effort. The IPI is not, and never was, a proprietary interface—and this may in itself be the major reason for its universality, simplicity, performance, and ease of use.

REFERENCES

1. ANSC Document Number X3T9.3/176
2. ANSC Document Number X3T9.3/82-19