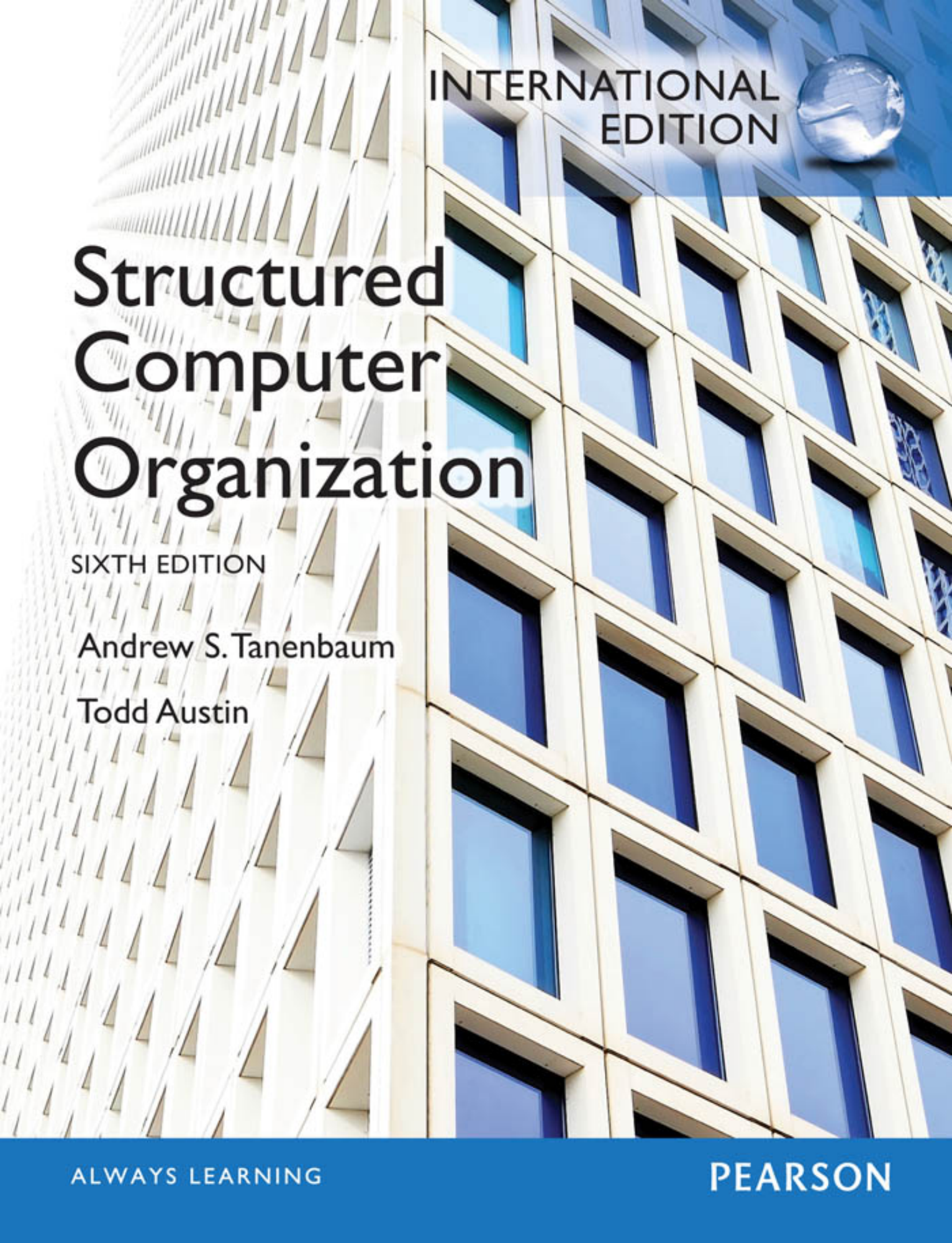

Structured Computer Organization

SIXTH EDITION

Andrew S. Tanenbaum

Todd Austin

ALWAYS LEARNING

PEARSON

STRUCTURED COMPUTER ORGANIZATION

SIXTH EDITION

and in 1995, PCI 2.1 came out. PCI 2.2 has features for mobile computers (mostly for saving battery power). The PCI bus runs at up to 66 MHz and can handle 64-bit transfers, for a total bandwidth of 528 MB/sec. With this kind of capacity, full-screen, full-motion video is doable (assuming the disk and the rest of the system are up to the job). In any event, the PCI bus will not be the bottleneck.

Even though 528 MB/sec sounds pretty fast, the PCI bus still had two problems. First, it was not good enough for a memory bus. Second, it was not compatible with all those old ISA cards out there. The solution Intel thought of was to design computers with three or more buses, as shown in Fig. 3-51. Here we see that the CPU can talk to the main memory on a special memory bus, and that an ISA bus can be connected to the PCI bus. This arrangement met all requirements, and as a consequence it was widely used in the 1990s.

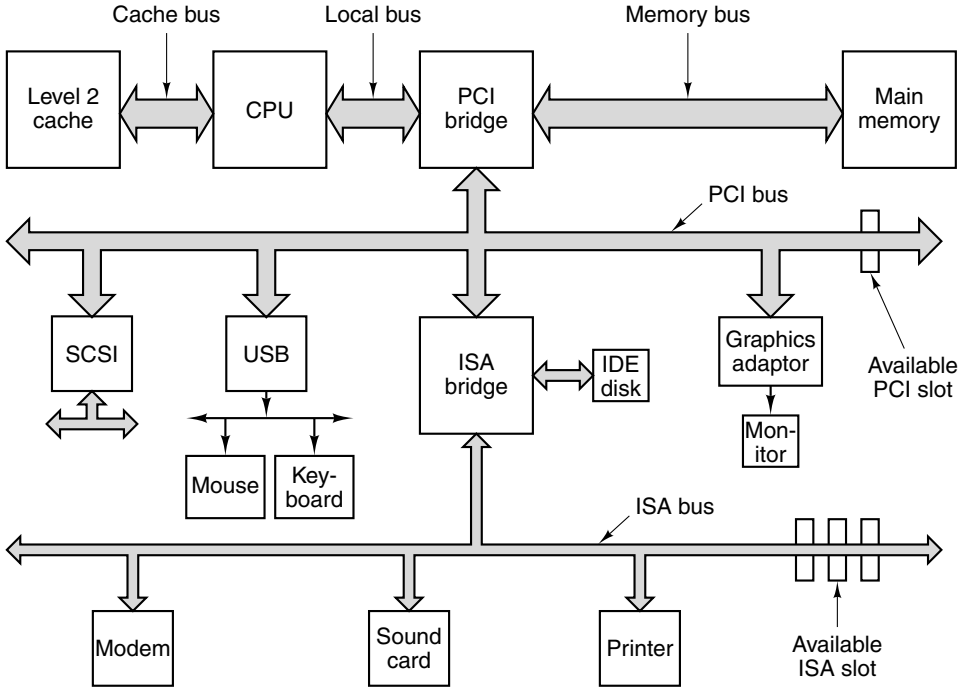

Figure 3-51. Architecture of an early Pentium system. The thicker buses have more bandwidth than the thinner ones but the figure is not to scale.

Two key components in this architecture are the two bridge chips (which Intel manufactures—hence its interest in this whole project). The PCI bridge connects the CPU, memory, and PCI bus. The ISA bridge connects the PCI bus to the ISA bus and also supports one or two IDE disks. Nearly all PC systems using this architecture would have one or more free PCI slots for adding new high-speed peripherals, and one or more ISA slots for adding low-speed peripherals.

The big advantage of the design of Fig. 3-51 is that the CPU has an extremely high bandwidth to memory using a proprietary memory bus; the PCI bus offers high bandwidth for fast peripherals such as SCSI disks, graphics adaptors, etc.; and old ISA cards can still be used. The USB box in the figure refers to the Universal Serial Bus, which will be discussed later in this chapter.

It would have been nice had there been only one kind of PCI card. Unfortunately, such is not the case. Options are provided for voltage, width, and timing. Older computers often use 5 volts and newer ones tend to use 3.3 volts, so the PCI bus supports both. The connectors are the same except for two bits of plastic that are there to prevent people from inserting a 5-volt card in a 3.3-volt PCI bus or vice versa. Fortunately, universal cards exist that support both voltages and can plug into either kind of slot. In addition to the voltage option, cards come in 32-bit and 64-bit versions. The 32-bit cards have 120 pins; the 64-bit cards have the same 120 pins plus an additional 64. A PCI bus system that supports 64-bit cards can also take 32-bit cards, but the reverse is not true. Finally, PCI buses and cards can run at either 33 or 66 MHz. The choice is made by having one pin wired either to the power supply or to ground. The connectors are identical for both speeds.

By the late 1990s, pretty much everyone agreed that the ISA bus was dead, so new designs excluded it. By then, however, monitor resolution had increased in some cases to 1600×1200 and the demand for full-screen full motion video had also increased, especially in the context of highly interactive games, so Intel added yet another bus just to drive the graphics card. This bus was called the **AGP bus** (**Accelerated Graphics Port bus**). The initial version, AGP 1.0, ran at 264 MB/sec, which was defined as 1x. While slower than the PCI bus, it was dedicated to driving the graphics card. Over the years, newer versions came out, with AGP 3.0 running at 2.1 GB/sec (8x). Today, even the high-performance AGP 3.0 bus has been usurped by even faster upstarts, in particular, the PCI Express bus, which can pump an amazing 16 GB/sec of data over high-speed serial bus links. A modern Core i7 system is illustrated in Fig. 3-52.

In a modern Core i7 based system, a number of interfaces have been integrated directly into the CPU chip. The two DDR3 memory channels, running at 1333 transactions/sec, connect to main memory and provide an aggregate bandwidth of 10 GB/sec per channel. Also integrated into the CPU is a 16-lane PCI Express channel, which optimally can be configured into a single 16-bit PCI Express bus or dual independent 8-bit PCI Express buses. The 16 lanes together provide a bandwidth of 16 GB/sec to I/O devices.

The CPU connects to the primary bridge chip, the P67, via the 20-Gb/sec (2.5 GB/sec) serial direct media interface (DMI). The P67 provides interfaces to a number of modern high-performance I/O interfaces. Eight additional PCI Express lanes are provided, plus SATA disk interfaces. The P67 also implements 14 USB 2.0 interfaces, 10G Ethernet and an audio interface.

The ICH10 chip provides legacy interface support for old devices. It is connected to the P67 via a slower DMI interface. The ICH10 implements the PCI bus,

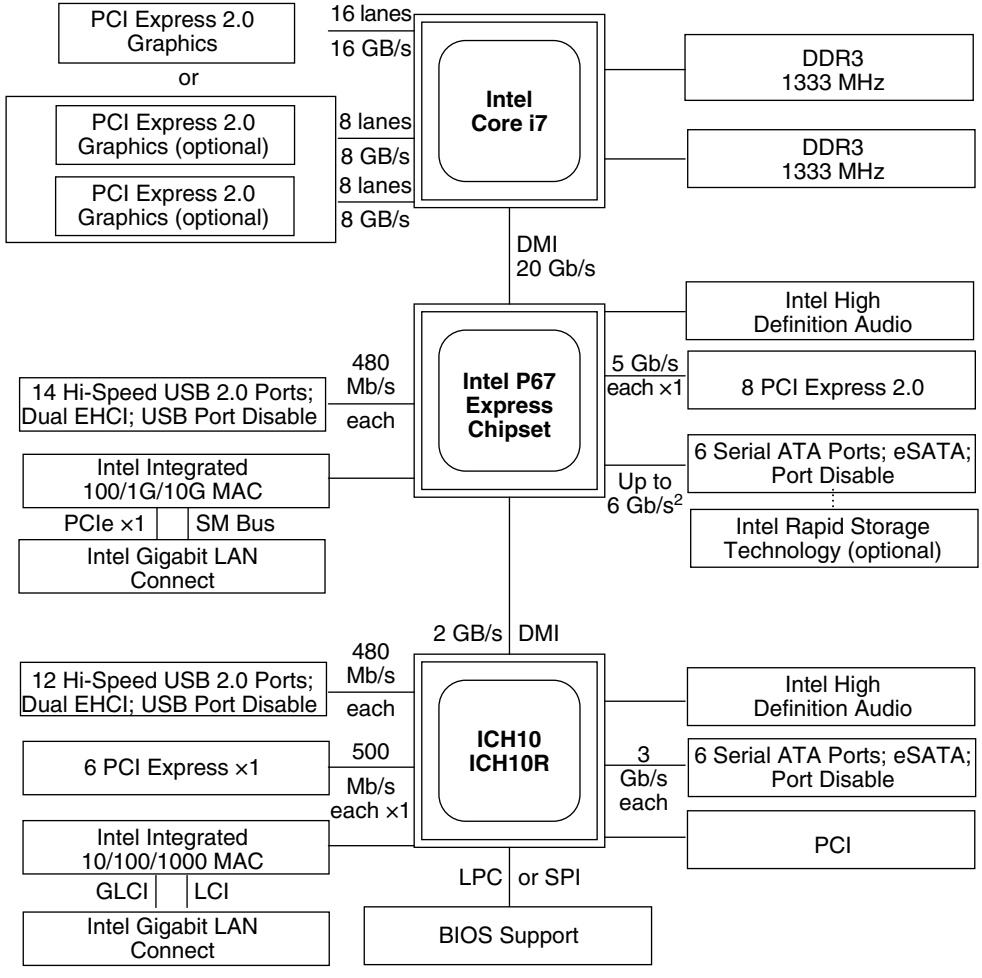

Figure 3-52. The bus structure of a modern Core i7 system.

1G Ethernet, USB ports, and old-style PCI Express and SATA interfaces. Newer systems may not incorporate the ICH10; it is required only if the system needs to support legacy interfaces.

PCI Bus Operation

Like all PC buses going back to the original IBM PC, the PCI bus is synchronous. All transactions on the PCI bus are between a master, officially called the **initiator**, and a slave, officially called the **target**. To keep the PCI pin count

down, the address and data lines are multiplexed. In this way, only 64 pins are needed on PCI cards for address plus data signals, even though PCI supports 64-bit addresses and 64-bit data.

The multiplexed address and data pins work as follows. On a read operation, during cycle 1, the master puts the address onto the bus. On cycle 2, the master removes the address and the bus is turned around so the slave can use it. On cycle 3, the slave outputs the data requested. On write operations, the bus does not have to be turned around because the master puts on both the address and the data. Nevertheless, the minimum transaction is still three cycles. If the slave is not able to respond in three cycles, it can insert wait states. Block transfers of unlimited size are also allowed, as well as several other kinds of bus cycles.

PCI Bus Arbitration

To use the PCI bus, a device must first acquire it. PCI bus arbitration uses a centralized bus arbiter, as shown in Fig. 3-53. In most designs, the bus arbiter is built into one of the bridge chips. Every PCI device has two dedicated lines running from it to the arbiter. One line, REQ#, is used to request the bus. The other line, GNT#, is used to receive bus grants. Note: REQ# is PCI-speak for $\overline{\text{REQ}}$.

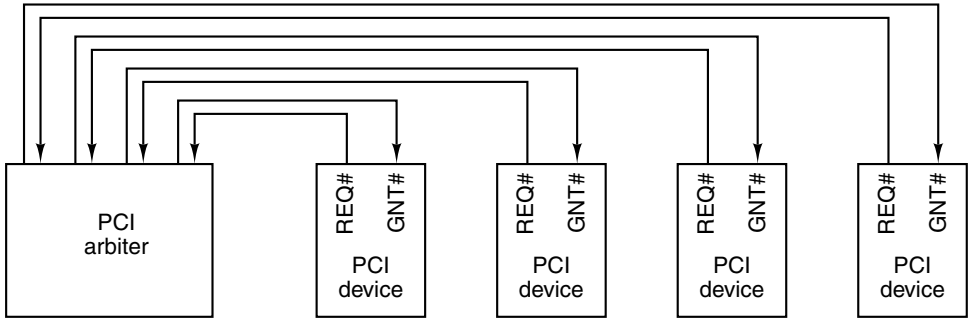

Figure 3-53. The PCI bus uses a centralized bus arbiter.

To request the bus, a PCI device (including the CPU) asserts REQ# and waits until it sees its GNT# line asserted by the arbiter. When that event happens, the device can use the bus on the next cycle. The algorithm used by the arbiter is not defined by the PCI specification. Round-robin arbitration, priority arbitration, and other schemes are all allowed. Clearly, a good arbiter will be fair, so as not to let some devices wait forever.

A bus grant is for only one transaction, although the length of this transaction is theoretically unbounded. If a device wants to run a second transaction and no other device is requesting the bus, it can go again, although often one idle cycle between transactions has to be inserted. However, under special circumstances, in

the absence of competition for the bus, a device can make back-to-back transactions without having to insert an idle cycle. If a bus master is making a very long transfer and some other device has requested the bus, the arbiter can negate the GNT# line. The current bus master is expected to monitor the GNT# line, so when it sees the negation, it must release the bus on the next cycle. This scheme allows very long transfers (which are efficient) when there is only one candidate bus master but still gives fast response to competing devices.

PCI Bus Signals

The PCI bus has a number of mandatory signals, shown in Fig. 3-54(a), and a number of optional signals, shown in Fig. 3-54(b). The remainder of the 120 or 184 pins are used for power, ground, and related miscellaneous functions and are not listed here. The *Master* (initiator) and *Slave* (target) columns tell who asserts the signal on a normal transaction. If the signal is asserted by a different device (e.g., CLK), both columns are left blank.

Let us now look briefly at each of the PCI bus signals. We will start with the mandatory (32-bit) signals, then move on to the optional (64-bit) signals. The CLK signal drives the bus. Most of the other signals are synchronous with it. A PCI bus transaction begins at the falling edge of CLK, which is in the middle of the cycle.

The 32 AD signals are for the address and data (for 32-bit transactions). Generally, during cycle 1 the address is asserted and during cycle 3 the data are asserted. The PAR signal is a parity bit for AD. The C/BE# signal is used for two different things. On cycle 1, it contains the bus command (read 1 word, block read, etc.). On cycle 2 it contains a bit map of 4 bits, telling which bytes of the 32-bit word are valid. Using C/BE# it is possible to read or write any 1, 2, or 3 bytes, as well as an entire word.

The FRAME# signal is asserted by the master to start a bus transaction. It tells the slave that the address and bus commands are now valid. On a read, usually IRDY# is asserted at the same time as FRAME#. It says the master is ready to accept incoming data. On a write, IRDY# is asserted later, when the data are on the bus.

The IDSEL signal relates to the fact that every PCI device must have a 256-byte configuration space that other devices can read (by asserting IDSEL). This configuration space contains properties of the device. The plug-and-play feature of some operating systems uses the configuration space to find out what devices are on the bus.

Now we come to signals asserted by the slave. The first of these, DEVSEL#, announces that the slave has detected its address on the AD lines and is prepared to engage in the transaction. If DEVSEL# is not asserted within a certain time limit, the master times out and assumes the device addressed is either absent or broken.

The second slave signal is TRDY#, which the slave asserts on reads to announce that the data are on the AD lines and on writes to announce that it is prepared to accept data.

Signal	Lines	Master	Slave	Description
CLK	1			Clock (33 or 66 MHz)
AD	32	×	×	Multiplexed address and data lines
PAR	1	×		Address or data parity bit
C/BE	4	×		Bus command/bit map for bytes enabled
FRAME#	1	×		Indicates that AD and C/BE are asserted
IRDY#	1	×		Read: master will accept; write: data present
IDSEL	1	×		Select configuration space instead of memory
DEVSEL#	1		×	Slave has decoded its address and is listening
TRDY#	1		×	Read: data present; write: slave will accept
STOP#	1		×	Slave wants to stop transaction immediately
PERR#	1			Data parity error detected by receiver
SERR#	1			Address parity error or system error detected
REQ#	1			Bus arbitration: request for bus ownership
GNT#	1			Bus arbitration: grant of bus ownership
RST#	1			Reset the system and all devices

(a)

Signal	Lines	Master	Slave	Description
REQ64#	1	×		Request to run a 64-bit transaction
ACK64#	1		×	Permission is granted for a 64-bit transaction
AD	32	×		Additional 32 bits of address or data
PAR64	1	×		Parity for the extra 32 address/data bits
C/BE#	4	×		Additional 4 bits for byte enables
LOCK	1	×		Lock the bus to allow multiple transactions
SBO#	1			Hit on a remote cache (for a multiprocessor)
SDONE	1			Snooping done (for a multiprocessor)
INTx	4			Request an interrupt
JTAG	5			IEEE 1149.1 JTAG test signals
M66EN	1			Wired to power or ground (66 MHz or 33 MHz)

(b)

Figure 3-54. (a) Mandatory PCI bus signals. (b) Optional PCI bus signals.

The next three signals are for error reporting. The first of these is STOP#, which the slave asserts if something disastrous happens and it wants to abort the current transaction. The next one, PERR#, is used to report a data parity error on the previous cycle. For a read, it is asserted by the master; for a write it is asserted by the slave. It is up to the receiver to take the appropriate action. Finally, SERR# is for reporting address errors and system errors.