6. SMALL COMPUTER SYSTEM INTERFACE

This laboratory work presents several types of electrical interfaces of the *Small Computer System Interface* (SCSI), the SCSI standards that have been developed, the SCSI bus, the structure of SCSI commands, the SCSI device configuration, and one of the SCSI programming interfaces.

6.1. Overview of Small Computer System Interface

Small Computer System Interface (SCSI) originates from the *Shugart Associates System Interface* (SASI), which has been developed by the hard disk drive manufacturer *Shugart Associates* and has been designed for connecting disk drives to a computer. This interface used logical addresses instead of physical addresses, and commands of 6 bytes each. The *X3T9* working committee of the American standards institute *ANSI* used the SASI specifications as basis for developing a standard of a parallel interface, standard that has later been named SCSI-1 and has been published in 1986. The same *X3T9* committee developed the SCSI-2 standard, which has been published in its final form in 1994. In 1993, another working committee, *T10*, started the activity for developing the SCSI-3 version of the standard. The documents of this version have been published separately, during several years, starting from 1996. These documents continue to be updated today with new versions.

SCSI is not a disk interface, that is, a certain type of controller, but rather a system interface composed of a bus to which several devices can be attached. One of these devices, the host adapter, operates as a bridge between the SCSI bus and the system bus. The SCSI bus does not communicate directly with the peripheral devices, such as disk drives, but rather with the controller that is included into these drives.

A single SCSI bus can accept up to 8 or 16 *physical units*, called SCSI *units*, out of which one is the SCSI adapter. The physical units can be magnetic disk drives, optical drives, scanners, printers. Most systems can support up to four SCSI adapters.

One of the reasons that delayed acceptance of the SCSI interface in the personal computer market was the lack of a standard for the host adapter, interface drivers, and BIOS. Due to the lack of an interface standard, several problems occurred, such as the impossibility to use the disk drives outside the SCSI bus, the impossibility to boot the operating system from these drives, and to use some operating systems. These problems were solved by developing the SCSI standards.

SCSI is an interface designed especially for workstations and high-performance servers. In the first version of the SCSI interface, the bus clock frequency was 5 MHz, and the maximum transfer rate was 5 MB/s. In later versions of this interface, the bus clock frequency is 80 MHz or 160 MHz, and the maximum transfer rates are of 320 MB/s and 640 MB/s, respectively.

Besides these parallel versions of the SCSI interface, a serial version of the interface has also been developed. This serial SCSI interface, called *Serial Attached* SCSI, gradually replaced the parallel SCSI interface. Both the parallel and the serial SCSI technologies are promoted by the SCSI *Trade Association* (www.scsita.org).

6.2. Types of SCSI Electrical Interfaces

There are two main types of SCSI electrical interfaces: normal and differential. In case of a *normal* SCSI interface (also called *Single-Ended* – SE), for each signal transmitted on the bus there is a single wire. The receiver circuits at the other end of the cable detect the electrical voltages on the bus lines. The cost of such an interface is low, but the signals transmitted are affected by noise and electromagnetic interference.

With *differential* SCSI interfaces, for each signal there is a pair of wires. One of the wires carries the same type of signal that is carried by normal SCSI interfaces. The second wire carries a signal that is obtained by the logical inversion of the original signal. The receiver circuits do not have to detect the absolute value of the voltage, but only the difference between the signals received on the two wires. Differential interfaces have several advantages: increased immunity to noise, the possibility of transfers with higher speeds, and greater cable length. With normal SCSI interfaces, the cable length can be up to 6 m for low frequencies of the SCSI bus, or up to 3 m for higher frequencies. With differential SCSI interfaces, the cable length can be up to 25 m or 12 m, depending on the bus frequency.

In the early differential SCSI interfaces, called *High Voltage Differential* (HVD), relatively high voltages were used for signal transmission. For this reason, low-power and small-sized interface chips could not be developed. For implementing the interface, circuits using several chips were required, which increased the cost of the interface. Another problem of this differential interface was that devices with differential interface could not be attached to the same bus to which devices with normal interface were attached, because the higher voltages of the differential interface could destroy the receiver circuits of the devices with normal interface. Because of the high cost and the incompatibility with devices having normal interface, the HVD differential interface was very rarely used, and its specifications were removed from the latest version of the SCSI standard.

Instead of the HVD electrical interface, a new differential interface has been developed, called LVD (*Low Voltage Differential*), which uses low voltages. This interface can be implemented with low-power and inexpensive circuits. Another advantage of this interface is that it allows connecting LVD devices to a normal SE SCSI bus, without the risk of damaging the devices' interface circuits. Optionally, devices with LVD interface can be designed so that they can be connected to both LVD and SE buses. These multimode devices detect if they are connected to an SE bus and will operate in the SE mode that is compatible with this bus. When there is even a single SE device connected to a SCSI bus, all LVD devices connected to the same bus will run in SE mode, at a reduced frequency of the bus (up to 20 MHz).

Because the SCSI connectors are the same for various types of the electrical interface, and connecting a device with HVD interface to a bus with SE or LVD devices may cause damages to these devices, it is necessary to differentiate between various types of interfaces. Devices with normal interface can be distinguished from those with HVD or LVD differential interfaces by the symbols labeled on them. Different symbols have been adopted in the industry for the normal and differential SCSI interfaces (Figure 6.1).

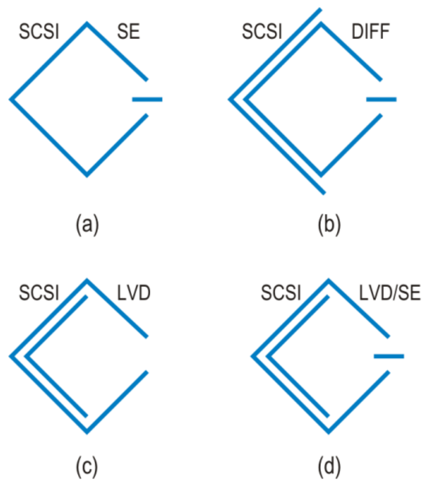

Figure 6.1. Symbols for the SCSI electrical interfaces: (a) SE normal interface; (b) HVD differential interface; (c) LVD differential interface; (d) LVD/SE multimode interface.

6.3. SCSI Standards

The SCSI interface standards have been developed by several working groups within the *ANSI* institute, which functioned or function as accredited committees for standardization. The SCSI standards define the physical and electrical parameters of an I/O bus, which is used for daisy-chaining the peripheral devices.

6.3.1. SCSI-1

The first standard of the SCSI interface, SCSI-1 (*ANSI X3.131-1986*), has been approved by the *ANSI* institute in 1986. Even before the approval of this standard, the hard disk manufacturers were worried that many of the commands and features specified by the standard were optional. For that reason, no guarantee existed that a particular peripheral will support all these commands. To solve this problem, the industry requested the *X3T9* committee that developed the SCSI-1 specifications to extend the mandatory commands and features of these specifications. Because the specifications were nearly finished, the *ANSI* institute has approved the standard in the existing form and established a new working group to develop a set of 18 basic SCSI commands. This set has been called *CCS* (*Common Command Set*) and will become the minimum set of commands supported by all peripherals. This command set became the basis of the SCSI-2 standard.

6.3.2. SCSI-2

The SCSI-2 standard is an improved version of the SCSI-1 standard, with new features and options added. Normally, SCSI-1 and SCSI-2 devices are compatible, but devices conforming to the SCSI-1 standard do not support the additional features introduced in the SCSI-2 standard.

The *X3T9* working committee finalized the SCSI-2 standard in 1990, but the document has been withdrawn at the end of the same year for some changes, to be made before the final publishing by the *ANSI* institute. The final version has been approved only at the beginning of 1994, although this document (*ANSI X3.131-1994*) contains very few changes compared to the initial version from 1990. Basically, all the specifications of the SCSI-1 standard can also be found in the SCSI-2 standard.

In addition to supporting the set of 18 basic SCSI commands, the SCSI-2 standard also contains new specifications, such as new commands for CD-ROM drives (including the possibility to use CD audio discs), tape drives, and other peripherals. Likewise, a faster version of the interface has been defined, called *Fast* SCSI-2, and a 16-bit version, called *Wide* SCSI-2. Another improvement defined by the SCSI-2 standard is the possibility to write the commands into a command queue, which allows a peripheral to accept several commands and to execute them in an order that is considered the most efficient. This possibility is important for multitasking operating systems.

Some of the changes specified by the SCSI-2 standard are minor. For instance, in the SCSI-1 standard the parity on the SCSI bus was optional, while in the SCSI-2 standard it is mandatory to use a parity bit. Another requirement is that initiator devices, such as host adapters, have to provide power to the terminator devices of the interface. However, most devices already fulfilled this requirement.

The SCSI-2 standard defines the following optional features:

- Fast SCSI;
- Wide SCSI;
- Command queuing;
- New commands;
- Improved terminators.

Fast SCSI refers to the capability to perform synchronous transfers at higher speeds. With this version, transfer rates of 10 MB/s can be achieved on the standard 8-bit SCSI bus.

When combined with a 16-bit *Wide* SCSI interface, this configuration allows transfer rates of 20 MB/s.

Wide SCSI enables data transfers on 16-bit or 32-bit buses. For these variants other cables are required than for the 8-bit variant. The standard 50-conductor cable used for 8-bit transfers is called the *A* cable. For the 16-bit *Wide* SCSI variant a 68-conductor *P* cable is required. For the 32-bit *Wide* SCSI variant, which was never actually implemented, two cables are required: the 68-conductor *P* cable and the 68-conductor *Q* cable.

According to the SCSI-1 standard, an initiator device, such as a host adapter, can send a single command per device. According to the SCSI-2 standard, the host adapter can send up to 256 commands to a single device, which will store the commands in a command queue and will process them before sending a response on the SCSI bus. The device can even change the order of commands to enable the most efficient execution possible.

The commands from the *Common Command Set* (*CCS*), which were already used in the industry, have been included officially into the SCSI-2 standard. The common command set was defined especially for disk drives and did not include specific commands for other types of devices. Many of the old commands have been changed and new commands have been added. For instance, new command sets have been added for CD-ROM drives, other optical drives, scanners, communication devices etc.

For correct operation of the single-ended SCSI bus, termination resistors with tight tolerances are needed. The 132-Ω passive terminators, defined in the SCSI-1 standard, are not adequate for the higher transfer speeds allowed by the SCSI-2 standard. These passive terminators can cause signal reflections, and errors can occur when transfer rates increase or when more devices are attached to the bus. According to the SCSI-2 standard, active components should be used as terminators, which ensure an impedance of 110 Ω and improve system integrity.

6.3.3. SCSI-3

SCSI-3 represents a collection of standards that were developed and published separately. These standards have been divided into categories that include: standards for primary commands, standards for commands that are specific to various device classes, standards for the communication protocols, and standards for the physical interconnects. In addition, a SCSI Architectural Model (SAM) exists for the physical and electrical interfaces. The SCSI-3 standards have been developed and are updated by the T10 technical committee within the *InterNational Committee on Information Technology Standards* (*INCITS*) that is accredited by *ANSI*. The working documents of the T10 committee are available at www.t10.org.

The main improvements introduced by the SCSI-3 standard include the following:

- Ultra2 (Fast-40) SCSI;
- Ultra3 (Fast-80DT) SCSI;
- Ultra4 (Fast-160DT) SCSI;
- Ultra5 (Fast-320DT) SCSI;
- New Low Voltage Differential signaling;
- Elimination of High Voltage Differential signaling.

Figure 6.2 presents the main components of the SCSI-3 collection of standards. Most of the individual standards have several versions, which are indicated in the figure.

The main components of the SCSI-3 family of standards are described next.

SCSI *Architecture Model* (*SAM*) defines the model of SCSI systems, the functional partitioning of the SCSI-3 set of standards, and the requirements applicable to all SCSI-3 implementations.

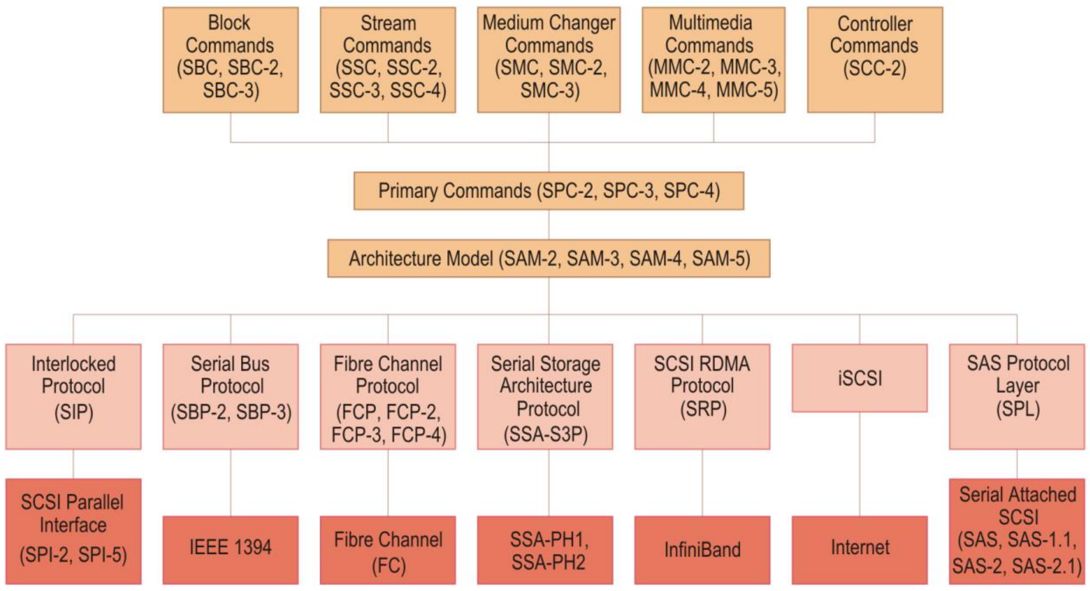
Block Commands (SBC, SBC-2, SBC-3)
Stream Commands (SSC, SSC-2, SSC-3, SSC-4)
Medium Changer Commands (SMC, SMC-2, SMC-3)
Multimedia Commands (MMC-2, MMC-3, MMC-4, MMC-5)
Controller Commands (SCC-2)
Primary Commands (SPC-2, SPC-3, SPC-4)
Architecture Model (SAM-2, SAM-3, SAM-4, SAM-5)
Interlocked Protocol (SIP)
Serial Bus Protocol (SBP-2, SBP-3)
Fibre Channel Protocol (FCP, FCP-2, FCP-3, FCP-4)
Serial Storage Architecture Protocol (SSA-S3P)
SCSI RDMA Protocol (SRP)
iSCSI
SAS Protocol Layer (SPL)
SCSI Parallel Interface (SPI-2, SPI-5)
IEEE 1394
Fibre Channel (FC)
SSA-PH1, SSA-PH2
InfiniBand
Internet
Serial Attached SCSI (SAS, SAS-1.1, SAS-2, SAS-2.1)

Figure 6.2. Main components of the SCSI-3 collection of standards.

Commands represent specifications that define the classes of devices and a device model for each class. These specifications define the commands that must be implemented by all devices or those that are specific to various classes of devices, and describe the rules that must be followed by an initiator device when it sends commands to another device. The main standards related to the commands are the following:

- *Primary Commands* (*SPC*): basic commands for all SCSI devices;
- *Block Commands* (*SBC*): commands for direct-access devices such as magnetic disk drives;
- *Stream Commands* (*SSC*): commands for sequential-access devices such as magnetic tape drives;
- *Medium Changer Commands* (*SMC*): commands for medium changer devices such as jukeboxes for audio discs;
- *Multimedia Commands* (*MMC*): commands for optical disc drives such as CD-ROM, CD-R/E (*Recordable/Erasable*), DVD;
- *Controller Commands* (*SCC*): commands for I/O controllers, e.g., for RAID (*Redundant Array of Independent Disks*) disk drive sets.

Protocols represent specifications that define the communication rules between various SCSI devices.

Interconnects contain specifications that define various physical interfaces. The SCSI-3 collection of standards defines several types of interfaces. For a long time, the most used was the SCSI *Parallel Interface* (SPI), until it begun to be replaced with the *Serial Attached* SCSI (SAS) serial interface. Another serial interface is *IEEE 1394*, especially used for video applications. *Fibre Channel* is a high-performance serial interface, which allows communication over optical fiber. The *Serial Storage Architecture* (SSA) serial interface is designed to connect disk drives or RAID disk arrays to servers. *InfiniBand* is another high-performance serial interface, intended for the connection of processors with high-speed peripherals such as storage devices. *iSCSI* (*Internet* SCSI) allows server computers to access remote disk volumes using the existing network infrastructure and the TCP/IP protocol.

We consider the parallel SCSI interface. Several versions of this interface have been developed, as the electrical protocol has been improved. The various types of the SCSI parallel interface are presented in Table 6.1.

Table 6.1. Types of parallel SCSI interfaces specified in the SCSI-3 standards.

Standard	Technology	Alternate Name	Clock Frequency (MHz)	Width	Maximum Transfer Rate (MB/s)
SPI	Fast-20	Ultra	20	8	20
SPI	Fast-20/Wide	Ultra/Wide	20	16	40
SPI-2	Fast-40	Ultra2	40	8	40
SPI-2	Fast-40/Wide	Ultra2/Wide	40	16	80
SPI-3	Fast-80DT	Ultra3 (Ultra160)	80	16	160
SPI-4	Fast-160DT	Ultra4 (Ultra320)	80	16	320
SPI-5	Fast-320DT	Ultra5 (Ultra640)	160	16	640

In Table 6.1, DT (*Double Transition*) indicates two transfers performed in each clock cycle, one at each edge of the clock signal. For the SPI-3, SPI-4, and SPI-5 interfaces, the SCSI bus width is 16 bits. Starting with the SPI-2 interface, only the LVD differential signaling is used.

6.4. The SCSI Bus

6.4.1. Communication on the SCSI Bus

Communication on the SCSI bus takes place between a device that initiates the transfer and a destination device. At any time, communication is performed only between two devices, one *initiator* that selects and controls the *target* device that performs the requested operation. Usually, a SCSI device has a fixed role of initiator or target, but some devices can fulfill both roles.

An initiator device can address up to eight logical units attached to a target device. For each data block logical addresses rather than physical addresses are used. For devices with direct addressing, each logical unit can be interrogated to determine the number of data blocks it contains. A logical unit may coincide with a peripheral device or it may be part of that device.

The SCSI standards define the signal levels on the bus, their logical function, the communication protocol, and the command sequences. All devices must allow to use the asynchronous protocol defined by the standards for data transfers. In addition, an optional protocol is defined for synchronous transfers. Similarly, a protocol is specified for sending messages in order to control the interface.

The SCSI bus uses a distributed arbitration system to support multiple initiators and the concurrent execution of I/O operations. A priority system allows granting the bus to the SCSI device with the highest priority out of those that request the bus. The time required to perform the arbitration is independent of the number of devices that request the bus and is lower than 10 μs.

The initiator may request the SCSI bus and may select a certain target device. The target may request the transfer of data, command, or status information on the data bus, and in some cases it may request the bus and it may reselect the initiator in order to continue an operation.

6.4.2. Asynchronous and Synchronous SCSI Protocol

The SCSI interface allows to use either the asynchronous or synchronous protocol for data transfers. By using the synchronous protocol faster data transfers can be achieved. Implementation of the asynchronous protocol is mandatory for all devices, while implementation of the synchronous protocol is optional.

In the *asynchronous* SCSI protocol, a device first issues a request (by asserting the *REQ* signal) and then waits for an acknowledgement from the initiator. The initiator places a byte or word on the data bus and asserts the *ACK* signal. The device reads the byte or word, de-asserts the *REQ* signal, and waits for the initiator to de-assert the *ACK* signal. These operations are then repeated for the next byte or word.

This *REQ/ACK* handshake protocol requires that the signals propagate through the SCSI cable twice for each data transfer. This requires a certain time; typically, the propagation delay is about 5.25 ns for each meter. The propagation delay is the primary speed limitation of asynchronous SCSI transfers when long cables are used.

In the *synchronous* SCSI protocol, an acknowledgement is also required for each request sent, but the acknowledgement can be delayed. Consequently, a device can send data packets one after another without the propagation delays required in the asynchronous protocol. The speed is determined by the bus cycle time without regard to propagation delay, and is therefore independent of cable length.

The first clock frequency specified for synchronous transfers was 5 MHz. This frequency has been increased gradually in the newer versions of the interface, with each version doubling the frequency of its predecessor version. Starting with the SPI-3 interface, the DT (*Double Transition*) technology is used, so that data transfers take place on both the rising and falling edge of each clock cycle.

6.4.3. SCSI Bus Signals

The 8-bit SCSI bus that uses the *A* cable contains 18 signals, out of which 9 data signals and 9 control signals. For the 16-bit and 32-bit versions there are bus extensions. The bus signals are described next.

- *BSY* (*Busy*): A wired-OR signal that indicates the busy state of the bus.
- *SEL* (*Select*): A wired-OR signal used by an initiator device to select a target device or by a target device to reselect an initiator device. The identifier of the selected device will appear on the data lines.
- *C/D* (*Control/Data*): Used by the target device to specify whether control or data information is sent on the data bus. The asserted state of this signal specifies that control information is sent.
- *I/O* (*Input/Output*): The target device controls with this signal the direction of data transfer. The direction is considered from the initiator device's viewpoint. The asserted value indicates an input operation for the initiator. This signal is also used to distinguish between the *Selection* and *Reselection* phases.
- *MSG* (*Message*): The target indicates with this signal that a message is sent on the bus (in the *Message* phase).
- *REQ* (*Request*): Generated by the target device to specify a transfer request using the asynchronous protocol.
- *ACK* (*Acknowledge*): Generated by the initiator device to acknowledge an asynchronous transfer request made by a target device by asserting the *REQ* signal.
- *ATN* (*Attention*): Used by an initiator device to indicate an attention condition for the target device.
- *RST* (*Reset*): A wired-OR signal that initialized the SCSI bus and resets all the devices attached to the bus.
- *DB* (7..0, P) (*Data Bus*). Represent the bidirectional data signals and the parity bit signal, which form a data bus. *DB* (7) is the most significant bit and it has the highest priority during the arbitration phase. *DB* (P) is the odd-parity bit. The parity is defined during the arbitration phase.
- *DB* (31..8, P1, P2, P3) (*Data Bus*). Represent the extension of the data bus. *DB* (P1, P2, P3) are odd-parity bits for *DB* (15..8), *DB* (23..16), and *DB* (31..24), respectively.
- *TERMPWR* (*Terminator Power*). Represents the power signal for the bus terminators.

6.4.4. SCSI Bus Phases

During operation, the SCSI bus passes through several distinct states, called *phases*. A phase describes the direction of transfer and the contents of information transferred. The following phases can be distinguished:

- Bus Free;
- Arbitration;
- Selection;
- Reselection;
- Command;
- Data;
- Status;
- Message.

After reset or power up, the bus enters the *Bus Free* phase. This is followed by the *Arbitration* phase, in which one of the devices gains control of the bus. If the arbitration fails, the bus returns to the *Bus Free* phase. If the arbitration succeeds, the bus enters the *Selection* or *Reselection* phase, in which a target and an initiator device that will execute a command are selected. After determining the two devices, one or more information transfer phases (*Command*, *Data*, *Status*, *Message*) follow. The last information transfer phase is normally the *Message In* phase, in which a DISCONNECT or COMMAND COMPLETE message is sent to the initiator, followed by the *Bus Free* phase.

Some examples of messages are described next.

- COMMAND COMPLETE: Sent by the target device to the initiator to indicate that a command has been completed and a valid status has been sent to the initiator.

- DISCONNECT: Sent by the target device to inform an initiator that the existing connection is going to be interrupted and a later reconnect will be required to complete the current operation.

- INITIATOR DETECTED ERROR: Sent by the initiator to inform a target device that an error (e. g., parity error) has occurred that does not preclude the target from retrying the operation.

- ABORT: Sent by the initiator to the target device to abandon the current operation.

- SYNCHRONOUS DATA TRANSFER: Sent by the initiator to a target device to select the synchronous transfer protocol.

6.4.5. SCSI Command Example

A READ command will be used as example to explain the various bus phases and signals. This command transfers data from a target device to the initiator.

The command begins in the *Bus Free* phase. Next, an *Arbitration* phase follows, in which one or more devices compete to take control of the bus. Each of the devices asserts the *BSY* signal and one of the data lines. Each device has a unique ID from 0 to 7 (or from 0 to 15), and each device asserts one of the data lines corresponding to its own ID. Each ID is assigned a priority, with 7 (or 15) having the highest priority and 0 the lowest priority. If more than one device asserts its ID during the *Arbitration* phase, then the device with the highest priority takes control of the bus.

The device that has won the arbitration becomes the initiator. This device enters the *Selection* phase by asserting the *SEL* signal. During this phase, the initiator asserts both data lines corresponding to its own ID and the target device's ID. After a delay, the initiator de-asserts the *BSY* signal. When the target device detects that the *SEL* signal is asserted, *BSY* and

I/O are de-asserted, and recognizes its ID, it will assert the *BSY* signal. When the initiator detects that the *BSY* signal is asserted, it releases the data bus and de-asserts the *SEL* signal.

Next, the target device indicates that it has entered the *Command* phase by asserting the *C/D* signal. This signal will remain asserted during this phase. The target device then asserts the *REQ* signal to request the first byte of the command from the initiator. The initiator places the first byte of the command on the data bus and asserts the *ACK* signal. The target device reads the byte and it de-asserts the *REQ* signal; the initiator then de-asserts the *ACK* signal. The first byte of the command contains the operation code, which indicates the number of bytes remained to be transferred. These additional bytes are transferred with the same *REQ*/*ACK* protocol.

After the target device has received and interpreted the command, it places the bus in the *Data In* phase by de-asserting the *C/D* signal (indicating data information) and it asserts the *I/O* signal (indicating the direction from the target to the initiator). The target device places the first byte or word of the requested data on the data bus and asserts the *REQ* signal. The initiator asserts the *ACK* signal after it has read the data. Additional data bytes or words are transferred with the same *REQ*/*ACK* protocol.

After transferring all the requested data, the target device places the bus in the *Status* phase and transfers a status byte to the initiator, indicating that it has successfully completed the transfer. The *C/D* signal is again asserted and the *I/O* signal remains asserted. The *REQ*/*ACK* protocol is used to transfer the status byte.

Finally, the target device places the bus in the *Message In* phase by asserting the *MSG* signal and transferring the COMMAND COMPLETE message byte. Once this message is received by the initiator, the target device releases all bus signals to place the bus in the *Bus Free* phase.

6.5. SCSI Commands

The SCSI standards specify a high-level command set that must be supported by SCSI devices. Mandatory and optional commands are defined, out of which some are common for all device types, while others are specific for various types of devices.

6.5.1. Structure of a Command Descriptor Block

A command is specified as a *Command Descriptor Block* (CDB), which is sent to the target device. For some commands, the command descriptor block is followed by a list of parameters that are sent during the *Data Out* phase. The command descriptor block begins with an operation code in the first byte and ends with a control byte.

There are typical structures of the descriptor block for 6-byte, 10-byte, and 16-byte commands. Figure 6.3 presents the typical structure of a command descriptor block for 10-byte commands.

Bit Byte	7	6	5	4	3	2	1	0
0	Operation Code							
1	Logical Unit Number			Reserved				
2	MSB	Logical Block Address						
3								
4								
5								LSB
6	Reserved							
7	MSB	Transfer Length, Parameter List Length, Allocation Length						
8								LSB
9	Control							

Figure 6.3. Structure of a descriptor block for 10-byte SCSI commands.

The *operation code* contains two fields: the *group code* (bits 7..5) and *command code* (bits 4..0). The three bits of the group code allow 8 groups of codes. The five bits of the command code allow 32 command codes in each group. Therefore, there are a total number of 256 possible operation codes. The group code defines separate groups for 6-byte, 10-byte, or 16-byte commands, as well as for vendor-specific codes.

The *logical unit number* is defined in the IDENTIFY message. The target will ignore the logical unit number specified in the command descriptor block if an IDENTIFY message has been received. It is recommended to set to zero the logical unit number in the descriptor block. This field was included into the command descriptor block for compatibility with some SCSI-1 devices.

The *logical block address* within a logical unit or a volume partition starts with block zero and must be contiguous up to the last logical block of the logical unit or partition. A descriptor block for 6-byte commands contains a logical block address of 21 bits. Descriptor blocks for 10-byte and 16-byte commands contain logical block addresses of 32 bits.

The *transfer length* specifies the length of data that must be transferred, usually, in number of blocks. For some commands, the transfer length specifies the number of bytes that are to be transferred. For commands that use a byte for the transfer length, a transfer length between 1 and 255 indicates the number of blocks that must be transferred by a single command. A value of 0 indicates 256 blocks. For commands that use several bytes for the transfer length, a length of 0 indicates that no data are to be transferred.

The *parameter list length* is used to specify the number of bytes that are transferred during the *Data Out* phase, bytes that represent the parameters sent to the target device.

The *allocation length* specifies the maximum number of bytes allocated by the initiator device for the data sent from the target device. The target will finish the *Data In* phase when a number of bytes indicated by the allocation length have been transferred. This length is used to limit the number of bytes returned to the initiator device.

The *control field* has the following structure:

7	6	5	4	3	2	1	0
Vendor-specific		Reserved				Flag	Link

The *Link* bit is used to continue the I/O process after the current command completes successfully. If the *Link* bit is set to 1, the target enters the *Command* phase after the current command completes. Implementation of this bit is optional. The *Flag* bit specifies the message that must be returned by the target device to the initiator if the *Link* bit is set to 1 and the command completes without errors. Usually, this bit is used to generate an interrupt of the initiator device between linked commands. Implementation of this bit is optional.

If the *Link* bit is set to 0 and the *Flag* bit is set to 1, the target device will return the CHECK CONDITION status. This status indicates that an unpredicted event occurred during the operation. The initiator device must send an additional command (*Request Sense*) to determine the unpredicted event. If the *Link* bit is set to 1, the *Flag* bit is set to 0, and the command completes successfully, the target sends the LINKED COMMAND COMPLETE message. If the *Link* bit is set to 1, the *Flag* bit is set to 1, and the command completes successfully, the target sends the LINKED COMMAND COMPLETE (WITH FLAG) message. These messages indicate the end of a linked command.

6.5.2. Command Examples

The SCSI-3 standard defines commands that can be used for all types of devices and commands that are specific for various types of devices. The main device types for which commands are defined are the following:

- Direct-access devices (magnetic disks);
- Sequential-access devices (magnetic tapes);
- Printers;
- Processors (intelligent devices);
- WORM (*Write Once, Read Multiple*) discs;
- CD-ROM discs (including audio discs);

- DVD discs;
- Scanners;
- Optical memories (several optical discs, e. g., CD-R);
- Communication devices (network nodes).

Table 6.2 presents example SCSI commands for direct-access devices.

Table 6.2. SCSI commands for direct-access devices.

Command Name	Code	Command Name	Code
ATA PASS-THROUGH (12)	0xA1	READ MEDIA SERIAL NUMBER	0xAB/0x01
ATA PASS-THROUGH (16)	0x85	RECEIVE DIAGNOSTIC RESULTS	0x1C
FORMAT UNIT	0x04	REQUEST SENSE	0x03
INQUIRY	0x12	SEND DIAGNOSTIC	0x1D
READ (10)	0x28	START STOP UNIT	0x1B
READ (12)	0xA8	TEST UNIT READY	0x00
READ (16)	0x88	VERIFY (10)	0x2F
READ BUFFER (10)	0x3C	VERIFY (16)	0x8F
READ BUFFER (16)	0x9B	WRITE (10)	0x2A
READ CAPACITY (10)	0x25	WRITE (12)	0xAA
READ CAPACITY (16)	0x9E/0x10	WRITE (16)	0x8A
READ DEFECT DATA (10)	0x37	WRITE AND VERIFY (10)	0x2E
READ DEFECT DATA (12)	0xB7	WRITE BUFFER	0x3B

6.5.3. INQUIRY Command

The INQUIRY command requests information about the logical unit and SCSI target device. The descriptor block of this command is presented in Figure 6.4.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (0x12)							
1	Reserved							EVPD
2	Page Code							
3	MSB	Allocation Length						
4								LSB
5	Control							

Figure 6.4. Structure of the descriptor block for the INQUIRY command.

When the EVPD (*Enable Vital Product Data*) bit is set to 1, the device controller will return the product data specified by the Page Code field (page codes are not described in this document). If the requested product data page is not implemented, the command is terminated with the CHECK CONDITION target status code, a “sense key” set to ILLEGAL REQUEST (0x05), and an additional sense code set to INVALID FIELD IN CDB (0x24). With the ASPI programming interface, the target status code can be found in the SRB_TargStat byte of the SCSI Request Block (SRB). The sense key can be found in the SenseArea[2] byte of the SRB, and the additional sense code can be found in the SenseArea[12] byte of the SRB. When the EVPD bit is set to 0, the device controller will return the standard INQUIRY data. In this case, the Page Code field should be set to 0; otherwise, the command is terminated with the CHECK CONDITION target status code.

The Allocation Length field specifies the maximum number of bytes allocated for the data sent from the target device (the length of the buffer allocated by the user). Byte 3 of the descriptor block should contain the most significant byte (MSB) of the buffer length, and byte 4 should contain the least significant byte (LSB) of the buffer length. The Control field should be set to 0.

The standard INQUIRY data contains at least 36 bytes. Additional bytes may contain vendor-specific data. The format of the first 36 bytes of the standard INQUIRY data is presented in Figure 6.5. Part of the fields of this data structure are described next. Other fields (shown in color, but without field names) are not described in this document.

Bit / Byte	7	6	5	4	3	2	1	0
0	Peripheral Qualifier			Peripheral Device Type				
1	RMB				Reserved			
2	Version							
3	Reserved				Response Data Format (0x02)			
4	Additional Length							
5		Res.				Reserved		
6	Res.				Reserved			
7	Reserved				Reserved		CmdQue	
8	MSB	T10 Vendor Identification						
. . .								
15								LSB
16	MSB	Product Identification						
. . .								
31								LSB
32	MSB	Product Revision						
. . .								
35								LSB

Figure 6.5. First 36 bytes of the data returned by the INQUIRY command.

The Peripheral Qualifier field specifies whether the addressed logical unit is accessible (the bits of the field are 000) or not accessible by the controller contained in the SCSI target device. The Peripheral Device Type field identifies the type of the device (e.g., direct-access block device, CD/DVD device, RAID controller device, etc.). The RMB (*Removable Medium*) bit indicates whether the medium is not removable (when set to 0) or it is removable (when set to 1).

The Version field indicates the implemented version of the *SCSI Primary Commands* (SPC) standard. This field contains 0x04 for the SPC-2 version, 0x05 for the SPC-3 version, 0x06 for the SPC-4 version, and 0x07 for the SPC-6 version. The Additional Length field indicates the length in bytes of the remaining standard INQUIRY data. The contents of this field are not dependent on the allocation length specified when issuing the command. When the CmdQue bit is set to 1, it indicates that the logical unit supports command management through a command queue.

The T10 Vendor Identification field contains eight ASCII characters that identify the manufacturer of the logical unit. Vendor identification strings are assigned by the T10 committee. The Product Identification field contains 16 ASCII characters that identify the product. Product identification strings are defined by the manufacturer. The Product Revision field contains four ASCII characters that identify the product revision. Product revision strings are defined by the manufacturer.

6.5.4. READ CAPACITY (10) Command

The READ CAPACITY (10) command requests the device controller to transfer eight bytes of data to a buffer allocated by the user. The data transferred describe the capacity and logical block length of a direct-access block device. The descriptor block of this command is presented in Figure 6.6. The Reserved and Control fields should be set to 0.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (0x25)							
1	Reserved							
. . .								
8								
9	Control							

Figure 6.6. Structure of the descriptor block for the READ CAPACITY (10) command.

The format of the data returned by the READ CAPACITY (10) command is shown in Figure 6.7. The Returned Logical Block Address field of four bytes contains the address of the last logical block of the device. This field contains 0xFFFFFFFF if the address of the last logical block is greater than the maximum value that can be specified in the field. In this case, the application should issue a READ CAPACITY (16) command to the device controller to transfer eight bytes of capacity data. The Logical Block Length field of four bytes contains the number of bytes of user data in a logical block.

Bit Byte	7	6	5	4	3	2	1	0
0	MSB							
. . .				Returned Logical Block Address				
3								LSB
4	MSB							
. . .				Logical Block Length				
7								LSB

Figure 6.7. Structure of data returned by the READ CAPACITY (10) command.

6.5.5. READ CAPACITY (16) Command

The READ CAPACITY (16) command requests the device controller to transfer data describing the capacity and medium format of a direct-access block device to a buffer allocated by the user. This command uses a CDB format called SERVICE ACTION IN (16), shown in Figure 6.8.

Bit Byte	7	6	5	4	3	2	1	0
0				Operation Code (0x9E)				
1		Reserved				Service Action (0x10)		
2								
. . .				Reserved				
9								
10	MSB							
. . .				Allocation Length				
13								LSB
14				Reserved				
15				Control				

Figure 6.8. Structure of the descriptor block for the READ CAPACITY (16) command.

The Operation Code and Service Action fields should be set to the values shown in Figure 6.8. The Reserved and Control fields should be set to 0. The Allocation Length field should be set to the maximum number of bytes allocated for the data sent from the device (the length of the buffer allocated by the user).

Bit Byte	7	6	5	4	3	2	1	0
0	MSB							
. . .				Returned Logical Block Address				
7								LSB
8	MSB							
. . .				Logical Block Length				
11								LSB

Figure 6.9. First 12 bytes of the data returned by the READ CAPACITY (16) command.

The READ CAPACITY (16) command returns 32 bytes of data. The format of the first 12 bytes of the data returned is shown in Figure 6.9. The Returned Logical Block Address field of eight bytes contains the address of the last logical block of the device. The Logical Block Length field of four bytes contains the number of bytes of user data in a logical block.

6.5.6. READ CD RECORDED CAPACITY Command

The READ CD RECORDED CAPACITY command allows to request information regarding the recorded capacity of a CD/DVD present in a logical unit. The format of the descriptor block and the structure of the data returned by this command are the same as those of the READ CAPACITY (10) command (Section 6.5.4). The address in the Returned Logical Block Address field is that of the last sector in the last complete recording session. The logical block length is reported as 2048.

6.5.7. READ MEDIA SERIAL NUMBER Command

The READ MEDIA SERIAL NUMBER command requests the device controller to return the current media serial number. This command uses the SERVICE ACTION IN (12) CDB format, shown in Figure 6.10.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (0xAB)							
1	Reserved				Service Action (0x01)			
2	Reserved							
. . .								
5								
6	MSB	Allocation Length						
. . .								
9								LSB
10	Reserved							
11	Control							

Figure 6.10. Structure of the descriptor block for the READ MEDIA SERIAL NUMBER command.

The Operation Code and Service Action fields should be set to the values shown in Figure 6.10. The Reserved and Control fields should be set to 0. The Allocation Length field should be set to the maximum number of bytes allocated for the serial number (the length of the buffer allocated by the user).

The format of the data returned by the READ MEDIA SERIAL NUMBER command is shown in Figure 6.11. The Media Serial Number Length field contains the number of bytes in the Media Serial Number field. The number of bytes is a multiple of four. The contents of the Media Serial Number Length field are not dependent on the allocation length specified when issuing the command. The Media Serial Number field contains the vendor-specific serial number of the media currently installed. If the media serial number is not available (e.g., the media currently installed has no valid serial number), then the Media Serial Number Length field is set to 0.

Bit / Byte	7	6	5	4	3	2	1	0
0	MSB	Media Serial Number Length (4n-4)						
. . .								
3								LSB
4	MSB	Media Serial Number						
. . .								
4n-1								LSB

Figure 6.11. Structure of data returned by the READ MEDIA SERIAL NUMBER command.

If no media is currently present in the device, the command is terminated with the CHECK CONDITION target status code, the “sense key” set to NOT READY (0x02), and the additional sense code set to MEDIUM NOT PRESENT (0x3A). With the ASPI programming interface, the location of these codes in the SRB has been presented in Section 6.5.3.

6.6. SCSI Device Configuration

To configure a SCSI device, two operations must be performed: setting the SCSI identifier and installing the terminators.

6.6.1. Setting the SCSI Identifier

Up to 8 or 16 SCSI devices can be attached to a SCSI bus and each of them must have a unique SCSI identifier (ID) in order to avoid conflicts. One of these identifiers, usually 7 or 15, which has the highest priority, is assigned to the host adapter. There are adapters that allow booting the operating system only from a disk drive with a specific ID. For example, older Adaptec host adapters required the boot disk drive to have the ID 0. Newer adapters allow booting the operating system from any drive, regardless its ID.

Usually, setting the ID requires positioning of three or four jumpers on the drive. The configuration of jumpers results from the binary representation of the ID values from 0 to 7 or from 0 to 15. In other cases, the ID can be set with a rotary switch.

Current SCSI systems are *Plug-and-Play* and the ID assignment is performed automatically by the operating system and the SCSI adapter. A protocol called SCAM (SCSI *Configured Automagically*) is used, which interrogates the SCSI devices and assigns unique identifiers for each of them, so that no conflicts will exist.

6.6.2. Installing the Terminators

The SCSI bus, like other buses, needs terminators at both ends of the bus. The incorrect termination of the bus lines represents one of the problems that may occur when using SCSI devices. If the host adapter is at one end of the bus, it must have its terminator enabled. If the adapter is in the middle of the bus and both internal and external bus links are present, the terminator in the adapter must be disabled and the devices at the two ends of the bus must have their terminators installed.

There are several types of terminators for the SCSI bus:

- Passive terminators;
- Active terminators;
- FPT (*Forced Perfect Termination*).

Passive terminators, composed of resistors, allow fluctuations of the bus signals. The signal levels depend on the voltage drop on these resistors. Usually, passive terminators are adequate for short distances, of up to 1 m, but for longer distances active terminators are needed.

Active terminators use, instead of voltage dividers composed of resistors, one or more voltage regulators that ensure a constant voltage. The voltage regulators ensure termination of signals on the SCSI bus at a correct voltage level. The SCSI-2 and SCSI-3 standards recommend using active terminators at both ends of the bus.

FPT *terminators* represent a type of active terminators that use stable voltage levels obtained with diodes. They eliminate fluctuations of signal levels, especially at high transfer speeds or great cable lengths.

Usually, external SCSI devices have an input and an output SCSI connector, so that several devices can be connected in a chain. When the device is at one end of the SCSI bus, a terminator must be installed in the output connector.

Some devices have embedded terminators that may be enabled or disabled through a switch or by their removal. Other devices do not have embedded terminators and for these external terminator modules must be used. Terminator modules are available in various configurations of the connectors, which also include pass-through terminators. These are necessary for devices installed at the end of the bus and which only have a single SCSI connector. Pass-through terminators are also commonly used in internal installations in which the devices do not have embedded terminating resistors. Many disk drives use such terminators for internal installations to save space on the logic board.

6.6.3. Other Configurations

There are additional options that can be configured for the SCSI drives. Following are the most common additional settings:

- Start on command;
- SCSI parity;
- Terminator power;
- Synchronous transfer negotiation.

6.6.3.1. Start on Command

If there are multiple disk drives connected to a system, they can be configured so that not all the drives start simultaneously, immediately when the system is powered on. In case of simultaneous start, the power consumption would increase significantly, the power supply can be overloaded, which can cause the system to hang or have other startup problems.

Usually, SCSI drives allow delaying the start of the spindle motor. When the SCSI adapters initialize the bus, most of them send out a command called *Start Unit* to each of the disk drives in succession. By positioning a jumper on the disk drive, the spindle motor will only start when the drive receives the *Start Unit* command form the host adapter. Because the adapter sends this command to all the disk drives, in succession, from the highest priority drive to the lowest priority drive, the higher priority drives can be configured to start first, and then lower priority drives to start sequentially. There are some adapters that do not send the *Start Unit* command. In this case, disk drives can be configured so that they delay the start with a certain number of seconds, rather than wait for the *Start Unit* command.

There is no need to enable the delayed start function for drives with separate power supply. This function is needed for internal drives that are powered from the same power supply that runs the system.

6.6.3.2. SCSI Parity

Using a parity bit allows limited error detection. Most of the host adapters support parity checking, so this option should be enabled on every device. This function is optional, because there are older adapters that do not use the SCSI parity bit, so for these adapters the parity checking must be disabled.

6.6.3.3. Terminator Power

Active terminators positioned at each end of the SCSI bus must be powered from at least one device attached to the bus. Usually, the terminator power is supplied by the host adapter, but there are some exceptions.

It is not a problem that the terminator power is supplied by multiple devices, because each source is protected with diodes. For simplicity, often all devices are configured to supply terminator power. If none of the devices supplies terminator power, the bus will not be terminated correctly and will not function properly.

6.6.3.4. Synchronous Transfer Negotiation

By default, the SCSI bus uses the asynchronous transfer protocol. It is possible to select the synchronous transfer protocol, which is faster, through a procedure called synchronous transfer negotiation. Before information is transferred across the bus, the initiator device and the target device negotiate how the transfer will take place. If both devices support the synchronous protocol, the transfer will be performed with this protocol.

Some older devices do not respond to a request for synchronous transfer and can even be disabled when they receive such requests. For this reason, host adapters and devices that support synchronous transfer negotiation often have a jumper that can be used to disable this negotiation, so they can work with older devices.

6.7. SCSI Adapters

The most important manufacturers of SCSI adapters are Adaptec and Future Domain. SCSI adapters can be easily installed, because all of their functions can be configured by software. The configuration information is written to a memory on the logic board of the adapter. The most important features of these adapters are the following:

- The ROM memory of the adapters contains software tools that allow to fully configure the adapters;
- The interrupt levels, ROM addresses, DMA and I/O port addresses, SCSI bus parity, and physical addresses of devices can be configured by the software;
- Terminators can be enabled and disabled by the software;
- No additional drivers are needed for more than two disk drives;
- Start on command of the drives is supported;
- The operating system can be loaded from any SCSI device.

6.8. SCSI Drivers

Each of the SCSI devices attached to the bus, except for disk drives, needs an external SCSI driver. Usually, disk drives have the drivers included in the BIOS program of the SCSI adapter. The external drivers are designed for a specific type of device and are specific to a certain SCSI adapter.

Two types of standard drivers for interfacing with the host adapter became popular. If a standard driver is available for the host adapter, manufacturers can create more easily new drivers for peripheral devices, which can communicate with the universal host adapter driver. This eliminates dependence on a particular type of host adapter. Universal drivers perform the link between the host adapter and operating system.

One of these drivers is the *Advanced SCSI Programming Interface* (ASPI), initiated by the Adaptec company. Later on, other SCSI device vendors have used the ASPI driver for their products. Many operating systems include an ASPI driver for several SCSI host adapters.

The second driver is called CAM (*Common Access Method*) and has been created by Future Domain and NCR companies. The CAM driver is a protocol approved by ANSI and enables to control several SCSI host adapters. Future Domain also provides a CAM-ASPI converter for its host adapters.

Current *Windows* operating systems use the *SCSI Pass Through Interface* (SPTI) for accessing SCSI devices. This programming interface has been developed by Microsoft Corporation and is accessible to software using the *DeviceIoControl()* function, which allows to send a control code directly to a device driver.

6.9. Serial Attached SCSI

Serial Attached SCSI (SAS) represents the serial version of the SCSI interface. It uses a point-to-point serial protocol and the standard SCSI command set. The SAS serial interface offers compatibility with second-generation SATA (*Serial* ATA – *Serial Advanced Technology Attachment*) disk drives, which may be connected to SAS backplanes.

The SAS standards have been developed by the *T10 Technical Committee* of *INCITS* (*InterNational Committee for Information Technology Standards*). The first version of the SAS interface standard has been published in 2003, and an improved version (SAS-1.1) has been published in 2005. Both versions specify a serial interface with a maximum speed of 3 Gbits/s. The SAS-2 standard, which has been published by the *INCITS* committee in 2009, defines the second generation of the SAS interface. This version of the standard introduces a serial link with a maximum speed of 6 Gbits/s, a physical layer that is compatible with the

SATA interface, and protocols for transferring SCSI commands to SAS devices and ATA commands to SATA devices. The SAS-2.1 version of the standard, which has been approved in 2010, defines a number of enhancements to the SAS-2 standard, including additional connectors and power management features. The SAS-3 version of the standard, which has been published in 2013, defines the third generation of the SAS interface, with a maximum speed of 12 Gbits/s. The SAS-4 version of the standard, which has been approved in 2018 by the T10 Technical Committee, increases the maximum speed of the interface to 24 Gbits/s. The SAS-5 version of the standard is under development and is expected to reach a speed of 45 Gbits/s.

Figure 6.12 illustrates the logos of the SAS-2, SAS-3, and SAS-4 interfaces.

Figure 6.12. SAS-2, SAS-3, and SAS-4 logos.

The SAS interface is a point-to-point architecture, with each device connecting directly to a SCSI port rather than connecting to a shared bus. Since the bandwidth is not shared by several devices, as in the case of the parallel SCSI interface, transfers with higher speeds are possible. By using a point-to-point connection, data reliability and the ability to locate failures are improved compared to a shared-bus architecture.

The SAS protocol specifies a full-duplex communication between the SAS controller and a disk drive. Therefore, read and write operations can be performed at the same time, which increases performance. Comparatively, SATA drives use half-duplex communication, so that when data are being sent to the drive for writing, any data that needs to be read from the drive must wait for the previous communication to complete.

A large number of SAS or SATA disk drives can be connected to a SAS controller port by using SAS *expanders*. An expander allows a single initiator to communicate with a number of SAS/SATA target devices. A SAS expander is similar to a switch in a network, which allows multiple systems to be connected using a single switch port. The cost of a system containing an expander is much lower compared to the cost of a system containing a large port-count SAS controller or multiple smaller port-count controllers. By using expanders, up to 16,384 physical links are possible.

A SAS domain consists of a set of SAS devices that communicate with one another by means of cables and backplanes, with or without expanders. Each SAS port and expander device in a SAS domain is assigned a globally unique identifier by the device manufacturer. This 64-bit identifier that represents the SAS address is called *World Wide Name* (WWN) and it uniquely identifies the device in the SAS domain just as a SCSI ID identifies a device attached to a parallel SCSI bus. Out of the 64 bits, 24 bits represent the vendor company identifier and 40 bits represent the vendor-specific identifier. Unlike for the parallel SCSI interface, there is no need to set manually the addresses in a serial SCSI system, since all the configuration is performed automatically. The SAS interface does not require installing terminators like the parallel SCSI interface.

The SAS interface uses differential signaling and data scrambling to reduce electromagnetic interference. It is possible to combine up to four ports with the same address into a wide port, which enable to increase the data rate.

SAS disk drives are dual-ported, which means that they can be directly connected to and controlled by two SAS controllers at the same time. This capability allows a redundant system to be built. When one of the SAS controllers fails, the other is still able to access the SAS disk drives and the data stored on those drives.

A SAS system can use either SAS or SATA disk drives. A SATA drive can be connected to a SAS system via an expander or a chip that implements the SAS protocol. This compatibility is possible because the SAS and SATA drive connectors are similar. These connectors are shown in Figure 6.13. Both connectors have the same number of pins and the pins have the same size and shape. However, the SATA drive connector has a notch, which is missing in the SAS drive connector. This notch prevents a SAS drive from being plugged into a SATA system. A SAS system uses a connector that allows a disk drive with a notch (SATA) or a disk drive without the notch (SAS) to be installed.

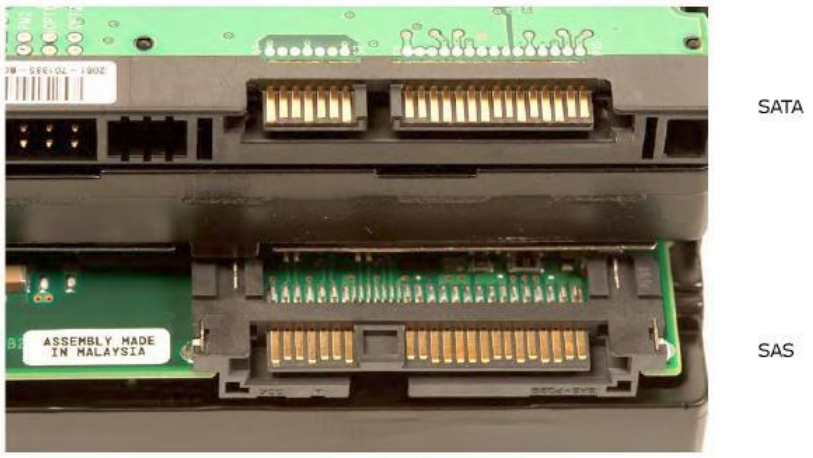

Figure 6.13. SATA and SAS disk drive connectors.

SAS disk drives have higher performance compared to SATA disk drives. The spindle speed of SAS drives is between 10,000 and 15,000 revolutions per minute (RPM), while the spindle speed of SATA drives is between 5,400 and 7,200 RPM. The higher spindle speed reduces the access time. The full-duplex communication supported by SAS drives also contributes to the higher performance of these drives. SAS drives are dual-ported, and this allows them to communicate with two host adapters or controllers simultaneously, which improves data availability. SAS drives are also much more reliable than SATA drives and are designed for much more intensive use.

6.10. Applications

6.10.1. Answer the following questions:

a. What are the improvements introduced by the SCSI-3 standards for the parallel SCSI interface?
b. What is the difference between the asynchronous and synchronous SCSI protocols?
c. What are the operations required for configuring a SCSI system?
d. What are the advantages of the serial SCSI interface compared to the parallel SCSI interface?

6.10.2. Create a *Windows* application for checking whether the ASPI manager is initialized correctly and for displaying the number of SCSI adapters in the system. As model for the *Windows* application, use the AppScroll-e application available on the laboratory web page in the AppScroll-e.zip archive. Perform the following operations to create the application project:

1. In the *Visual Studio 2022* programming environment, create a new empty *Windows Desktop* project with the *Windows Desktop Wizard*. Check the *Place solution and project in the same directory* option to avoid creating another folder for the solution.
2. Change the active solution platform to x86.
3. Change the *Character Set* project property by opening the *Properties* dialog window. In this window, expand the *Configuration Properties* option, expand the *Advanced* option, select the *Character Set* line in the right tab, and choose the *Not Set* option.
4. Copy to the project folder the files contained in the AppScroll-e.zip archive and add all the files to the project.

5. Copy to the project folder the files from the WNASPI32.zip archive, available on the laboratory web page. Add to the project the wnaspi32.h and scsidefs.h header files.
6. Open the AppScroll-e.cpp source file, delete the `#include "Hw.h"` directive, and add `#include` directives to include the wnaspi32.h and scsidefs.h header files.
7. In the `AppScroll()` function, delete the sequences for initializing the HW library with the `HwOpen()` function and for closing the HW library with the `HwClose()` function.
8. Select *Build* → *Build Solution* and make sure that the application builds without errors.

Use the specifications of the ASPI programming interface from the ASPI32.pdf document (available on the laboratory web page) and the wnaspi32.dll library to write a function with no input parameters for performing the following operations:

- Load the wnaspi32.dll library with the `LoadLibrary()` function;
- Determine the address of the `GetASPI32SupportInfo()` and `SendASPI32-Command()` functions with the `GetProcAddress()` function.
- Call the `GetASPI32SupportInfo()` function and display whether the ASPI manager is initialized correctly.
- If the ASPI manager is initialized correctly, display the number of SCSI adapters in the system.

After writing the function, include the call to this function in the `AppScroll()` function and verify the operation of the function.

6.10.3. Extend Application 6.10.2 with a function for performing the following operations for every SCSI adapter, starting with adapter number 0:

- Call the `SendASPI32Command()` function with the command code `SC_HA_INQUIRY` and display the following information from the `SRB_HAInquiry` structure: the string describing the ASPI manager (the `HA_ManagerId` member of the structure); the SCSI identifier of host adapter (the `HA_SCSI_ID` member of the structure); the string describing the host adapter (the `HA_Identifier` member of the structure).
- Determine the maximum number of SCSI devices (targets); this number is contained in the `HA_Unique[3]` byte of the `SRB_HAInquiry` structure.
- For every SCSI device and for every logical unit (from 0 to 7) of a device, call the `SendASPI32Command()` function with the command code `SC_GET_DEV_TYPE`, and display the SCSI device number, the logical unit number, and the device type (the `SRB_DeviceType` member of the `SRB_GDEVBlock` structure).

After writing the function, include the call to this function in the `AppScroll()` function, and verify the operation of the function.

Note

- For obtaining information about some SCSI devices with the wnaspi32.dll library, administrator privileges are required.

6.10.4. Extend Application 6.10.3 with a function that sends the READ CAPACITY (10) command to a direct-access device (disk drive) or optical drive. The parameters of this function are the SCSI adapter number, SCSI device number, and logical unit number. In this function, create an event with manual reset and without a name using the `CreateEvent()` function, and reset the event using the `ResetEvent()` function. Initialize the SRB structure for the `SendASPI32Command()` function specifying `SC_EXEC_SCSI_CMD` for the command code, input direction and event notification for the `SRB_Flags` field, and `SENSE_LEN` for the `SRB_SenseLen` field. Initialize the CDB structure for the READ CAPACITY (10) command, and then call the `SendASPI32Command()` function. If the function completes with the `SS_PENDING` code, call the `WaitForSingleObject()` function with the handle returned by the `CreateEvent()` function to wait until the event is in the signaled state. Then, examine the `SRB_Status` field to check the command completion. If the command completed with the `SS_ERR` code, display an error message containing the target status byte, the sense key, and the additional sense code (presented in Section 6.5.3), close the event handle, and return with an error code. If the command completed with the `SS_COMP` code, compute the capacity using the data returned by the function, display the capacity in bytes and MB, close the event handle, and return with 0.

Include the call to this function in the function written for Application 6.10.3, when an optical (CD-ROM) drive is detected. Insert a CD or DVD into the optical drive and verify the operation of the function.

6.10.5. Extend Application 6.10.4 with a function that sends the INQUIRY command to a disk drive or optical drive. The parameters of this function are the SCSI adapter number, SCSI device number, and logical unit number. In the CDB structure of the command, the EVPD bit and the Page Code field should both be 0. If the command completes successfully, display whether the medium is removable or not removable, the implemented version of the SPC standard, the vendor identification string, product identification string, and product revision string. Include the call to this function in the function written for Application 6.10.3, when a disk drive or optical drive is detected, and verify its operation by running the .exe file as administrator.

Bibliography

[1] Adaptec, Inc., “ASPI for Win32 Technical Reference”, 2001.

[2] Hoffman, C., Sawyer, S., “6Gb/s SAS – An Evolutionary Step for SAS Technology”, White Paper, SCSI Trade Association, September 2, 2008.

[3] Mueller, S., *Upgrading and Repairing PCs*, 22nd Edition, Que Publishing, 2015.

[4] Rosch, W. L., *Hardware Bible*, Sixth Edition, Que Publishing, 2003.

[5] Stallings, W., *Computer Organization and Architecture. Designing for Performance*, Ninth Edition, Pearson Education, 2013.

[6] InterNational Committee for Information Technology Standards, “Information Technology – SCSI Architecture Model – 6 (SAM-6)”, Project T10/BSR INCITS 546, Revision 07, February 17, 2021.

[7] InterNational Committee for Information Technology Standards, “Information Technology – SCSI Block Commands – 4 (SBC-4)”, Project T10/BSR INCITS 506, Revision 22, September 15, 2020.

[8] InterNational Committee for Information Technology Standards, “Information Technology – MultiMedia Command Set – 6 (MMC-6)”, Project T10/BSR INCITS 468, Revision 02g, December 11, 2009.

[9] InterNational Committee for Information Technology Standards, “Information Technology – SCSI Primary Commands – 6 (SPC-6)”, Project T10/BSR INCITS 566, Revision 5, March 8, 2021.