7

VLSI INTEGRATED CIRCUITS

7.1 INTRODUCTION

7.1.1 Integrated Circuits, VLSI, and CMOS

Integrated circuits (ICs) or VLSI chips, as they are frequently called, control almost everything in our external environment, from IP routers that process Internet traffic, PCs, cell phones, and car engines to household appliances. They are complex electronic systems embedded in a small volume of highly processed silicon. Although the cost of designing and developing them can be very high, when spread across millions of production units, the individual IC cost can be very low. ICs have migrated consistently to smaller feature sizes over the years, allowing more circuitry to be packed on each chip. This increased capacity per unit area can be used to decrease cost and/or increase functionality. It has been observed that the number of transistors in an IC doubles every two years or so. This famous observation is called *Moore's law*, after Gordon Moore, the first person to make that observation.[†]

Very-large-scale integration (VLSI) is the process of creating ICs by combining millions of transistors into a single chip. The VLSI era began in the 1970s when silicon chips achieved substantial complexity. Depending on the actual number of transistors, ICs can be of large-scale integration (LSI), very large-scale integration (VLSI), or ultralarge-scale integration (ULSI), but we will not bother with these distinctions; instead, we refer to all IC here as VLSI chips. The current complexity of VLSI is well represented by Intel's Montecito Itanium server chip, which contains over 1 billion transistors and is manufactured in the extremely advanced 65-nm manufacturing process, where 65 nm represents the smallest physical feature used on the chip.

VLSI chips typically use CMOS (complementary metal-oxide-semiconductor) processing technology to define basic transistors and to connect them using on-chip metal lines. CMOS represents a major portion of ICs sold worldwide (over 90%). The "MOS" in CMOS represents the basic sandwich structure of a MOS transistor, consisting of a metal gate formed on top of an oxide that in turn is grown on top of semiconductor (silicon) substrate. The word *complementary* in CMOS refers to the fact that both n-channel (carrying electrons) and p-channel (carrying holes) MOS transistors are available.

CMOS logic uses a combination of p- and n-type MOS transistors to implement logic gates and other digital circuits. CMOS digital gates feature the very interesting property that they do not dissipate any power in static conditions. As a result, CMOS circuits are power efficient, as power is dissipated only during transistor switching.[‡]

[†]Gordon Moore, Cramming more components onto integrated circuits, *Electronics Magazine*, April 19, 1965.

[‡]In very advanced CMOS circuits there is some finite amount of static power dissipation due to transistor leakage currents, so this particular advantage of CMOS technology is slowly diminishing as transistors continue to shrink.

7.1.2 Classification of Integrated Circuits

Integrated circuits can be classified into analog, digital, and mixed signal (where both analog and digital functions are performed by the same circuit). *Digital integrated circuits* can contain NAND/NOR logic gates, flip-flops, latches, multiplexers, counters, and higher-complexity blocks. They effectively process information in terms of 1 and 0 signals. *Analog integrated circuits*, such as operational amplifiers, comparators, references, or power management circuits, work by processing signals that have a continuous scale from 0 to 1. They perform functions such as amplification, filtering, modulation, or mixing. *Mixed-signal circuits* combine analog and digital functionality, and typical examples of this class include analog-to-digital (ADC) and digital-to-analog (DAC) converters.

Complex ICs, frequently referred to as a *system-on-chip* (SOC), can contain numerous classes of analog, digital, and mixed-signal circuits as shown schematically in Figure 7.1. To generate internal high-speed clocking, the analog clock generator is needed, typically designed using a phase-locked loop (PLL). To communicate with other ICs using traces on the same printed circuit board (PCB) or through copper/coaxial cables, high-speed I/O interface circuitry is required. Digital signal processing is performed using logic gates, microprocessor cores, on-chip field-programmable gate arrays, or dedicated digital signal processing (DSP) blocks. The data are stored internally in static random access memories (SRAMs) and/or dynamic random access memories (DRAMs). Various support circuits, such as a low drop-out (LDO) voltage regulator, might be required as well. Finally, circuits that support chip testing—built-in self-testing (BIST) and boundary scan (JTAG)—are also required.

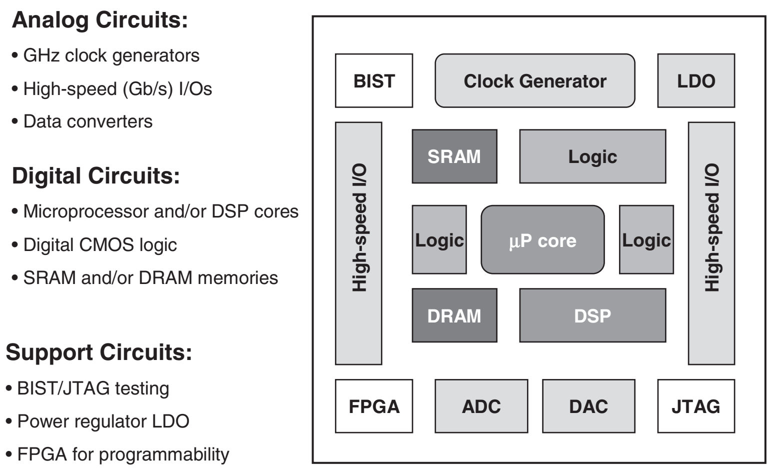

FIGURE 7.1 Schematic representation of a system-on-chip containing various analog, digital, and mixed-signal blocks.

ICs can be classified depending on the flexibility of their use. ICs designed for a specific application are called *application-specific integrated circuits* (ASICs). An example of the IC customized for a particular use could be a SONET framer that is being designed to process SONET frames in data networking equipment. We discuss the architecture of a SONET framer in more detail in Section 7.4.2.

More general ICs, which can be used in multiple applications, are called *standard products*. Examples of such ICs are ADC data converters or temperature sensor chips. Both ASICs and standard products have a fixed functionality that cannot be changed. We can say that in this case the functionality is permanently hardwired. One type of ASIC, known as a *gate array*, provides some flexibility of IC functionality at the manufacturing phase. A gate array circuit is a prefabricated silicon chip with no particular function in which transistors, standard NAND/NOR logic gates, flip-flops, and other circuits are placed at regular predefined positions and manufactured on a wafer, usually called the *master slice*. Creation of a circuit with a specified function is accomplished by adding a final surface layer metal that is interconnected to the chips on the master slice late in the manufacturing process, joining these elements to allow the function of the chip to be customized as desired.

Gate arrays therefore provide certain flexibility for the ASIC manufacturer, but still grant no flexibility to users. User programmability can be accomplished in hardware by using field-programmable gate arrays (FPGAs). FPGAs contain custom logic blocks and programmable interconnects that can be enabled (programmed) using built-in fuse technology, as shown in Figure 7.2. The

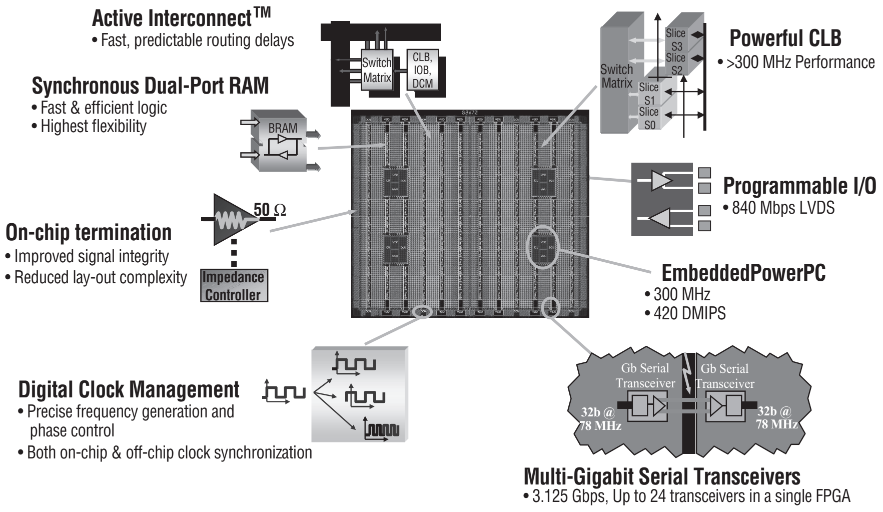

FIGURE 7.2 Characteristics of a field-programmable gate array chip. (Courtesy of Xilinx Inc.)

programmable logic components can be programmed to duplicate the functionality of basic logic gates or more complex combinational functions. Complex FPGAs contain various flip-flop configurations, memory blocks, processor cores, and programmable inputs/outputs (I/Os) as well. These characteristics allow the same FPGA chip to be used in many different applications.

FPGAs are generally slower than their ASIC counterparts, might not be able to handle complex designs, and will dissipate more power. However, they have several advantages, such as a shorter time to market, ability to program in the field to change functionality, and lower nonrecurring engineering costs. FPGAs could also be simpler, faster, and more effective to use than general-purpose microprocessors. For these reasons their use is becoming increasing popular, in particular when prototyping new systems.

7.1.3 Looking Ahead

This chapter is organized as follows: First, in Section 7.2 we discuss the application of IC in data networking. We divide ICs into layer 1, 2, and 3 devices and see how they correspond to the OSI open interconnect model and networking protocols discussed in Chapters 5 and 6. In Section 7.3 we cover chip I/O interfaces. We will show how networking ICs interface to each other, to memory chips, and to microprocessors. In Section 7.4 we discuss some architectural design examples, including time slicing, SONET framer, and network processor architectures.

The design of ICs is a complex process. Over the last 40 years it has changed so dramatically that today's techniques of modern IC design would probably seem strange even for the most visionary IC designers from the past. Because the subject matter is the focus of numerous books, and due to space limitation, we only touch on the most important topics in Section 7.5.

By the end of the chapter you should have some idea of what a VLSI chip is, how it is used in data networking, and how one goes about designing it. You will not have become a VLSI designer, though. If you would like to learn about silicon chips and methods to design them, we have provide a list of selected references at the end of the chapter.

7.2 INTEGRATED CIRCUITS FOR DATA NETWORKING

Data networking ICs are VLSI chips that enable processing and transmission of voice, video, and data, in either electrical or optical form, from one location to another across a broadband Internet network. They typically are responsible for the following functions:

- Modulation and amplification of electrical signals for transmission through a physical medium (an optical fiber, a twisted copper pair, or a coaxial cable). Similarly, in the receive direction: demodulation and equalization for reception of electrical signals. These devices are typically referred to as physical medium devices (PMDs) or physical layer devices (PHYs). Both PMD and PHY ICs are considered to be layer 1 devices, using OSI as a frame of reference.

- Formatting of data into frames or cells using predefined protocols (ATM, Ethernet, fiber channel, SONET/SDH). These devices, typically referred to as *framers* or *mappers*, are considered to be layer 2 devices.
- Processing of data packets. Processing functions include protocol conversion, packet forwarding, policing, lookup, classification, encryption, and traffic management. These devices are typically referred to as network processors, classification engines, or data link devices, and are considered to be layer 3 (occasionally, layer 4) devices.

Various classes of ICs for data networking are compared in Figure 7.3. Layer 1 devices are responsible for physical aspects of data transmission such as signal timing and frame formatting. Their functionality is defined rigorously by various standards established to create networking equipment interoperability. As a result, they typically have a fixed functionality and are implemented as ASICs. On the other hand, layer 3 devices require a good deal of flexibility, as they deal with tasks such as IP packet processing and forwarding that could be specific to a particular use of the networking equipment. As a result, they are implemented in ICs that provide means of programmability such as a general-purpose central processing unit or network

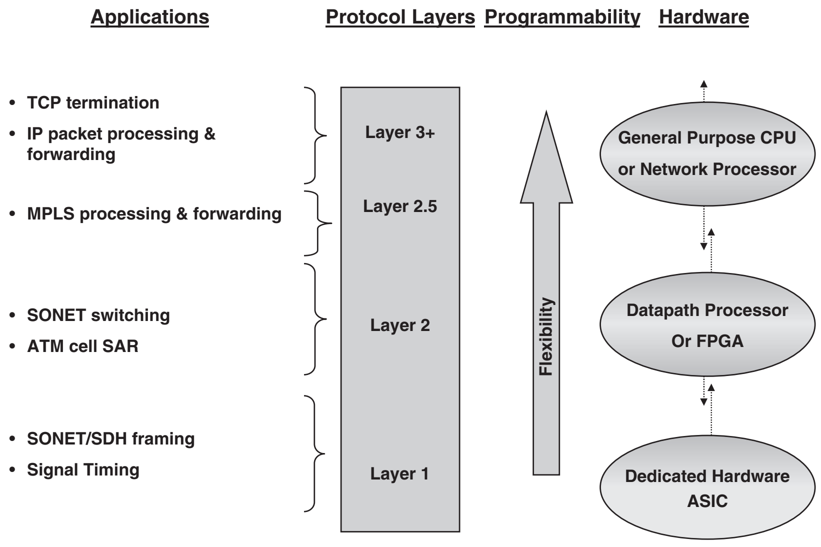

FIGURE 7.3 Comparison of data networking ICs, indicating their application space and position within the OSI model.

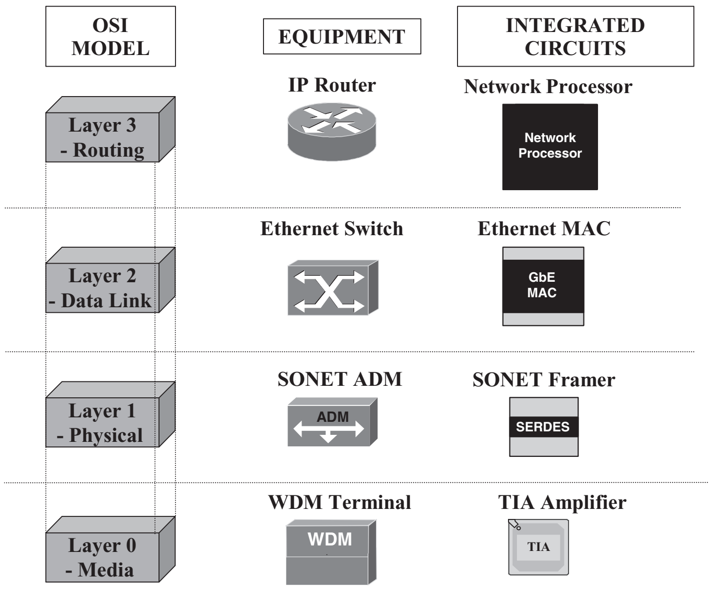

FIGURE 7.4 Conceptual relationship between ICs, data networking equipment, and the OSI reference model.

processor. Layer 2 devices fall somewhere in between and can be implemented as datapath processors (DSPs) or FPGAs. A conceptual relationship between data networking ICs, networking equipment, and the OSI reference model is shown in Figure 7.4.

7.2.1 PMD and PHY Devices (Layer 1)

Physical medium devices (PMDs) are used for I/O interfacing to physical media such as an optical fiber, a copper wire, or a coaxial cable. Typically, these ICs are data protocol independent, as they deal only with physical effects and electrical signals. PMDs include devices such as amplifiers for photodetectors or drivers for lasers. PMD devices require analog expertise to design and are discussed in more detail in Chapter 8.

Physical layer devices (PHYs) are responsible for defining how the traffic will be transported from one location to another. PHY devices include SONET serializers and deserializers (SERDES devices), Ethernet transceivers, cable modem chips, and xDSL transceiverPHY devices deal with issues such as clock generation and clock extraction. As a result, one of the important properties for these devices is signal *jitter*, which expresses uncertainty in the timing position of the signal. PHY devices have to comply with elaborate specifications for the amount of jitter being generated (intrinsic jitter), the amount of jitter the device can tolerate at its input (jitter tolerance), and the transfer

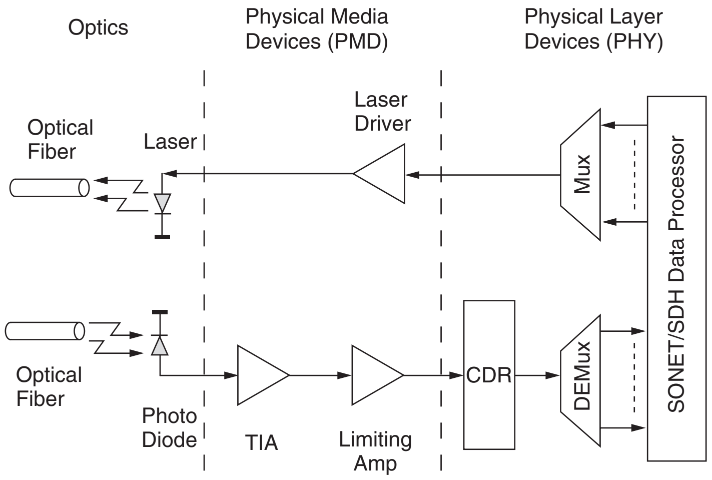

FIGURE 7.5 SONET link indicating the roles of PMD and PHY devices.

characteristics of the output jitter vs. applied input jitter (jitter transfer). Different jitter specifications exist for different transport technologies (SONET/SDH or Ethernet).

The interaction between PMD and PHY devices is illustrated using a SONET link as shown in Figure 7.5. We use that example to summarize basic functions performed by PMD devices.

Amplification An optical signal is typically received by a photodiode and converted to an electrical signal. Since the photodiode produces current I whereas most electronic devices require voltage signal V, I is converted to V by a PMD device known as a *trans-impedance amplifier* (TIA). A TIA is a rather sensitive device, as it needs to clean and amplify small signals generated by the photodiode. The small voltage signal from a TIA frequently needs to be amplified further, with the limiting amplifier performing this function. The term *limited* comes from the fact that the limited amplifier output provides the same signal amplitude regardless of the strength of the optical signal being converted. A limiting amplifier is another example of a PMD device.

Clock and Data Recovery The signal from the limiting amplifier is received by the clock and data recovery (CDR) circuit. The CDR recovers the clock from the NRZ data stream using analog phase-locked loop techniques. A detailed description of this process is given in Chapter 8. A CDR jitter tolerance parameter is typically the most challenging parameter to be met in design. After recovering the clock and "cleaning up," the signal received needs to be deserialized to a parallel stream of lower-data-rate signals. The chip that performs this function is called a *demultiplexer* or *deserializer*. Using the clock

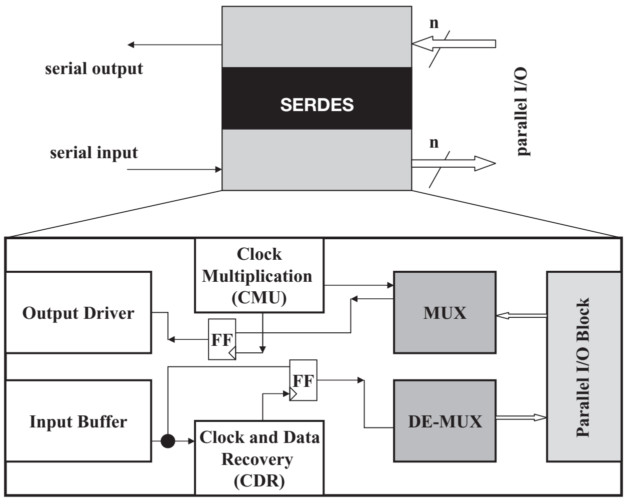

FIGURE 7.6 SERDES chip containing data multiplexing (mux and demux) and clock generation and recovery (CMU and CDR) functionality.

recovered and lower speed, further data processing can be done in the upstream framer device.

Clock Multiplexing Similar processing occurs in the reverse, egress direction. The data generated by the framer device is serialized using a multiplexer or a serializer device. The clock multiplication unit (CMU) is used to generate a high-purity reference clock for data transmission. The data are transmitted to a PMD device known as a *laser driver*, which in turn modulates the laser, effectively converting an electrical signal into an optical signal. Deserializers and serializers are frequently combined on the same chip, creating a *SERDES device*, shown in Figure 7.6. SERDES devices convert a high-speed serial data stream into a parallel combination of lower-rate signals. SERDES devices are used in numerous applications beyond data networking, as serial-to-parallel signal conversions are frequently desired in hardware design. More details of SERDES and CDR operation are given in Chapter 8. While Figure 7.5 shows all processing functions in separate IC blocks, many of the functions can be integrated on one chip. SERDES, CMU, and CRU are frequently integrated into devices called *transceivers*; the chip might sometimes also include a laser driver or limiting amplifies. At lower data transmission rates, integration of framer and transceiver functions is also possible.

7.2.2 Framers and Mappers (Layer 2)

Layer 2 devices perform various data coding, framing, and mapping functions. These tasks are layer 2 protocol specific, and as a result, the devices might have various names, such as DS3 mapper, STS-3 framer, or Ethernet MAC. Intrinsically, they all perform the same function: They process packets/frames/cells at the data link layer.

Data Coding Data coding translates user data into a format that is suitable for transmission. Various coding schemes are used in communication protocols. The most popular ones for broadband are scrambling, used in SONET, and 8b/10b, used in Ethernet. The 8b/10b scheme codes 8 bits of data into 10-bit code words. The expanded code space is used to maintain sufficient transitions for data recovery, to detect transmission errors, and to control emitted energy. In addition, special codes can be used to delineate frames. 8b/10b is a well-known coding scheme used in Ethernet protocol, although due to the high 25% overhead, similar, more efficient schemes such as 64b/65b or 64b/66b have been proposed for high-speed (10 Gb/s and above) transmissions.

The scrambling scheme transmits the exclusive-or of the data and the output of a pseudorandom source. The primary reason for scrambling is to introduce some transitions into long streams of zeros or ones. Scrambling is easy to implement and has no coding overhead. However, scrambling success is only probabilistic, and once in awhile very long sequence of zeros or ones will be produced, possibly causing bit error (although the probability of this happening is very low).

The most important feature of coding is transition density, as coding has to ensure that there are enough zero–one transitions to allow the receiver to recover the data sent by the transmitter. In that respect, 8b/10b has a higher transition density than PRBS23-type SONET traffic at the expense of additional overhead.

Framing Framing divides transmitted data into blocks, typically of fixed lengths. ATM cells, SONET frames, or Ethernet packets are results of this framing process. Framing allows the receiver to align with the transmitter and detect transmission errors. Framing devices are usually protocol specific, although multiprotocol devices have started appearing in the marketplace. An example of the multiframe device is an IP/ATM framer, shown in Figure 7.7, which can be configured to process IP packets over SONET or ATM cells.

Packet-over-SONET (POS) is an established technology for carrying IP and other data traffic over a SONET backbone. Variable-length data packets are mapped directly into the SONET synchronous payload envelope (SPE). POS provides reliable, high-capacity, point-to-point data links using SONET physical layer transmission standards. The POS/ATM framer from Figure 7.7 locates and locks onto the boundaries of each payload in the coming signal. As it checks for transmission errors it can detect the loss of a signal or frame. The framer might also perform some overhead processing functions. These functions include extracting key bits and initiating automatic protection switching (APS) when the fatal error has occurred. Another function of the framer device is section, line, and path termination. These extra bytes are added by

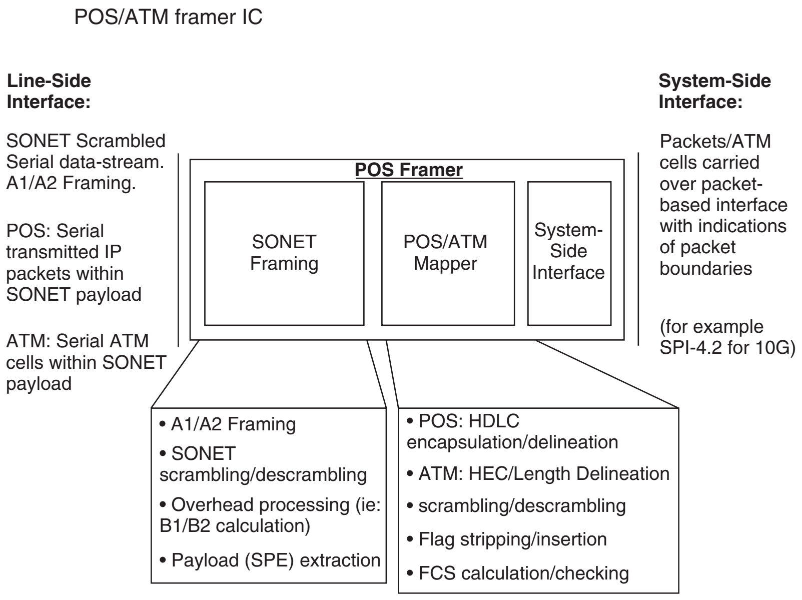
POS/ATM framer IC
Line-Side Interface:
SONET Scrambled Serial data-stream. A1/A2 Framing.
POS: Serial transmitted IP packets within SONET payload
ATM: Serial ATM cells within SONET payload
POS Framer
SONET Framing
POS/ATM Mapper
System-Side Interface
System-Side Interface:
Packets/ATM cells carried over packet-based interface with indications of packet boundaries
(for example SPI-4.2 for 10G)
• A1/A2 Framing
• SONET scrambling/descrambling
• Overhead processing (ie: B1/B2 calculation)
• Payload (SPE) extraction
• POS: HDLC encapsulation/delineation
• ATM: HEC/Length Delineation
• scrambling/descrambling
• Flag stripping/insertion
• FCS calculation/checking

FIGURE 7.7 POS/ATM framer integrated circuit.

SONET protocol and need to be inserted or extracted as required by networking functionality.

Mapping Mapping processes involve squeezing different payloads into various frame types, such as a SONET envelope. The device performing mapping, the *mapper*, contains an elastic store to accommodate varying payload lengths. It can also insert or drop traffic, as needed. A related function is pointer processing, which keeps track of the location of each payload within a SONET envelope. The pointer processor also tries to align different tributary rates into a synchronous optical channel.

Forward Error Correction A critical factor for high-speed transmission is the bit error rate (BER). Due to the high throughput used, the consequences of sending "wrong" bits are serious, as many customers might be affected. To improve BER coding techniques one should use forward error correction (FEC). ICs that employ FEC algorithms greatly reduce the bit error rates in long-haul systems. The concept behind FEC is straightforward; extra bits are added to the data frame to introduce redundancy. These extra bits are used to check whether the transmission error has occurred and to correct it. FEC techniques are not unique to broadband communication; they are used in cell phones, compact disks, memory, and satellite systems.

For high-rate-data (10 Gb/s and above) transmission, system engineering of FEC is a must. The high-speed SONET link shown in Figure 7.5 might require one extra IC that performs the FEC function and would be positioned between the high-speed transceiver and the digital framer. There are two types of FEC techniques for broadband ICs: in-band and out-of-band. *In-band FEC* uses certain locations in the SONET frame for extra FEC bits; *out-of-band FEC* appends these extra bits after the frame. As expected, out-of-band FEC is more powerful than in-band FEC. One typical Reed–Solomon code implementation for OC-192 can improve BER from 1×10^{-4} to a value better than 5×10^{-15}. This is an improvement by over 10 orders of magnitude! The price for this BER improvement is the additional bit overhead required; in this example it is 7%, so the data have to be transferred at the 10.65-Gb/s rate. That requirement creates some extra demand on analog circuitry that has to perform clock and data recovery function at 10.65 GHz instead of 9.95 GHz.

7.2.3 Packet Processing Devices (Layer 3)

As expected, layer 3 ICs perform in hardware networking functions assigned in the OSI model. Examples of this include IP routing, ATM layer policing, and fiber channel interworking with the SCSI protocol in storage area networks. Layer 3 devices can be implemented as ASICs or as network processors. FPGA can also be used if the required gate count is low. Network processors (NPs) provide flexibility at the cost of complex firmware. As we will see later, a major decision in NP architecture design is to determine the amount of internal and external memory required (both SRAM and DRAM). A network processor consists of multiple CPUs, standardized high-speed interfaces, and data buffering capabilities. It is a software-based, highly flexible solution for networking processing.

Network Processor The concept of the network processor is based on the microprocessors used in the computing industry. NPs are frequently advertised as universal devices for layer 3–7 packet processing. Although NP flexibility is a big bonus, it comes at the price of compromised performance. Complex firmware development is another disadvantage. Network processors are microprocessors designed specifically for packet processing, a core element of high-speed communication routers. Similar to the microprocessors used in a PC, a network processor is a programmable chip, but its instruction set has been optimized to support the operations that are needed in networking, especially for packet processing. The success of the microprocessor for the computing platform is based on the fact that its architecture is optimized according to the software characteristics of the application intended. For example, a desktop CPU is optimized for a broad range of window applications, while a graphic processor is optimized for two- and three-dimensional rendering, a DSP for signal processing, and a high-end RISC for scientific computations.

Packet Processing Although commercial vendors are advertising NP as a "do it all" solution, it is quite possible that there will be no unique NP platform at all, but instead, that NPs will be application specific to some degree, as microprocessors for computing are. To define NP one needs to consider the commonality of various target applications. No matter what protocol is involved, packet processing requires the following tasks: (1) header processing, lookups, and classifications; (2) scheduling, policing, and queuing; and (3) encrypting and security. Due to these requirements, memory bandwidth becomes a common bottleneck for NPs in networking applications. In particular, lookups, packet classification, and queuing are memory-intensive tasks. As a result, NPs typically contain several external SRAM/DRAM I/O interfaces. The challenge for NP architects is to define memory architecture in such a way that the overall NP performance is close to the optimized hardware solution in which both external and internal memories are optimized for the given application.

Another key architecture design issue is the organization of parallel processing. There are multiple approaches: single-instruction multiple data; single-program multiple data; multiple-program multiple data; simultaneous multithreading, superscalar, as used in Intel's Pentium; super pipelining, as used by MIPS; or very long instruction word, as used in Intel's Itanium. It remains to be seen which parallel-processing scheme is the most efficient for packet processing. This area will probably be an interesting research subject for years to come.

In closing this network processor description, we would like to mention that there are examples of network processors that are designed for dedicated applications. They include classification processors, which can be used to lighten the processing load; security processors, which can be used to lighten the processing load by handling security algorithms; or traffic/policy managers, which can handle queuing, load balancing, and quality of service. However, the differences between these devices are beyond the scope of complexity that we can afford in this chapter.

7.3 CHIP I/O INTERFACES

In the world of communication the word *interface* may have several meanings, depending on the context within which it is used. For example, the open system interconnection (OSI) reference model outlines a seven-layer abstract description of the communication protocol that includes interfaces between different layers so that they can talk to the ones above or below. In this section, the meaning of *interface* is limited to the external interface, which allows communication of one IC device with another. As in any type of communication, this can be accomplished by means of certain predefined protocol. These protocols define the data and control signals as well as their timing parameters. Many of these interfaces are the subsets of certain standards that also define information about transmission mediums, type of connectors, and similar physical level information.

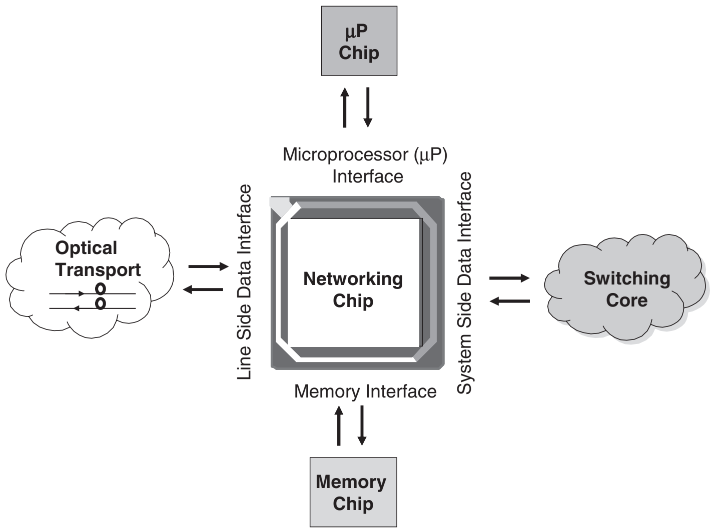

FIGURE 7.8 Networking IC with its external I/O interfaces.

Digital VLSI chips associated with data networking are responsible for processing data paths: extracting, processing, and inserting information from and to the data stream as it passes through the networking node. For that reason, networking chips typically have system- and line-side interfaces. The system side is described as the interface connecting the data path to the system side of a node within which the chip is placed. The line side is described as the side closest to the optical fiber transmit/receive facility.

A schematic representation of a networking IC with its data interfaces is shown in Figure 7.8. The chip contains two data interfaces, one on the line side and another on the system side. This is the path for the data flow. In addition, there is one interface dedicated to communication with a control microprocessor and another one to store data in the external memory. The microprocessor path is used for control purposes, and the memory path is used for storage. Needless to say, if demands for control or storage are small, they can be incorporated on-chip, resulting in no external microprocessor or memory interfaces being present.

In this section we first consider the physical nature of the I/O scheme: serial or parallel. Later we discuss some typical networking standards, followed by a description of data networking interfaces. Finally, we briefly examine both microprocessor and external memory interfaces.

7.3.1 Serial vs. Parallel I/O

Chip I/O interfaces can be divided into serial and parallel interfaces. In *serial interfaces*, each bit of data is allocated a time slot defined by a clock signal and

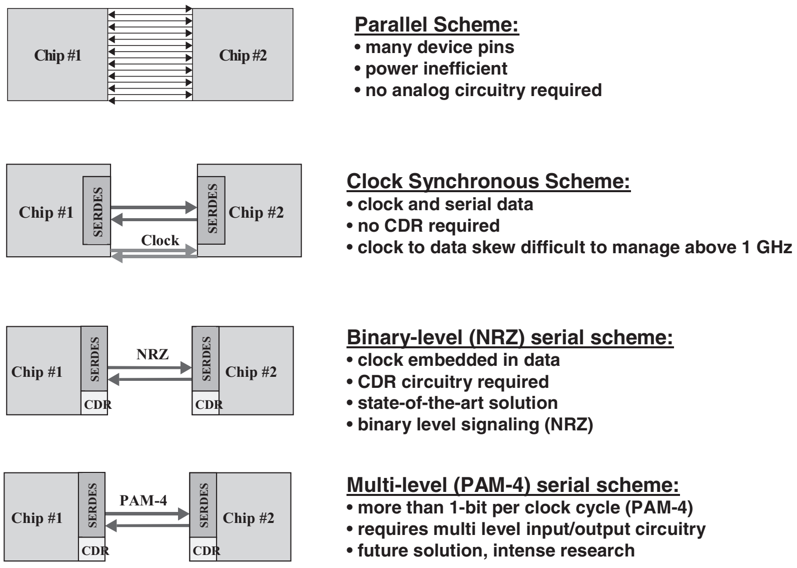
Chip #1
Chip #2
Parallel Scheme:
• many device pins
• power inefficient
• no analog circuitry required
Chip #1
SERDES
Clock
SERDES
Chip #2
Clock Synchronous Scheme:
• clock and serial data
• no CDR required
• clock to data skew difficult to manage above 1 GHz
Chip #1
SERDES
CDR
NRZ
SERDES
CDR
Chip #2
Binary-level (NRZ) serial scheme:
• clock embedded in data
• CDR circuitry required
• state-of-the-art solution
• binary level signaling (NRZ)
Chip #1
SERDES
CDR
PAM-4
SERDES
CDR
Chip #2
Multi-level (PAM-4) serial scheme:
• more than 1-bit per clock cycle (PAM-4)
• requires multi level input/output circuitry
• future solution, intense research

FIGURE 7.9 Four types of chip I/O interfaces.

transmitted one after another. Sometimes, the serial interface includes one or more control signals, the function of which is defined by a particular protocol.

Parallel interfaces require more device pins as the data are presented in bytes or words of a desired width. Parallel interfaces typically run at slower clock speeds, but the data throughput is multiplied by the number of bits transmitted or received in one time slot. In addition to data and clock signals, parallel interfaces frequently include control or indicator signals that help further processing of data.

A summary of various chip I/O interfaces is shown in Figure 7.9. The first, at the top, is a standard, parallel interface which is used frequently to connect low-speed ICs. It uses many device pins but its architecture is uncomplicated and requires no analog circuitry. Due to its large number of parallel signals, this scheme is power inefficient.

A clock synchronous interface uses the SERDES device described earlier to serialize parallel data for transmission, with clock information being sent separately. This scheme works well for medium speeds; however, above 1 Gb/s, significant issues arise due to a clock skew between the clock and the data lines that are not identical. For very high speeds (above 1 Gb/s), an NRZ (no return to zero) scheme with embedded clock is preferred. As the clock edge is embedded in the data stream, additional clock and data recovery (CDR) circuitry is required. As a result, the interface block has to contain both SERDES and CDR functionality, but this is the price that has to be paid for very fast data transmission.

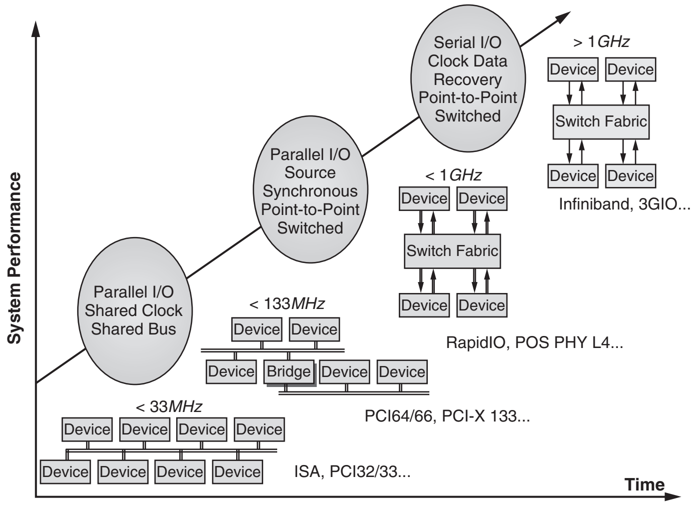

FIGURE 7.10 Time evolution of I/O signaling schemes. (Courtesy of Xilinx Inc.)

An NRZ scheme uses binary-level signaling (0 and 1 levels). More efficient schemes utilize multibit encoding to sent more than 1 bit of data in a given clock cycle. For example, PAM-4 uses four levels (1, 3/4, 1/2, 1/4, 0) to encode 2 bits of information. Although it would seem that PAM-4 would be twice as efficient as NRZ, significant difficulties exist in the implementation of high-speed analog circuitry that can resolve four voltage levels at gigahertz speed. As a result, the multilevel scheme is a subject of university research but is not used in commercial ICs.

As the required throughput of I/O communication lines increases, the serial I/O scheme is gaining more widespread adoption. Figure 7.10 illustrates this historical evolution. High-speed serial links are prevalent today in most point-to-point applications and frequently use differential signaling (see Figure 7.11). Although single-ended solution will save some device pins, typically it will not reduce dissipated power. Single-ended links are also problematic from a signal integrity and electromagnetic compatibility point of view.

From an electrical signal point of view there are numerous I/O electrical standards and proprietary schemes used for high-speed serial links. Low-voltage differential signaling, current mode logic, and pseudo-emitter-coupled logic are examples of electrical standards for I/O signals frequently used to connect networking ICs. We discuss some circuit aspects of high-speed I/Os in Chapter 8.

7.3.2 Networking I/O Standards

Data networking VLSI devices are integrated in the Internet system that spans the entire globe. As a result, they process data that in many instances

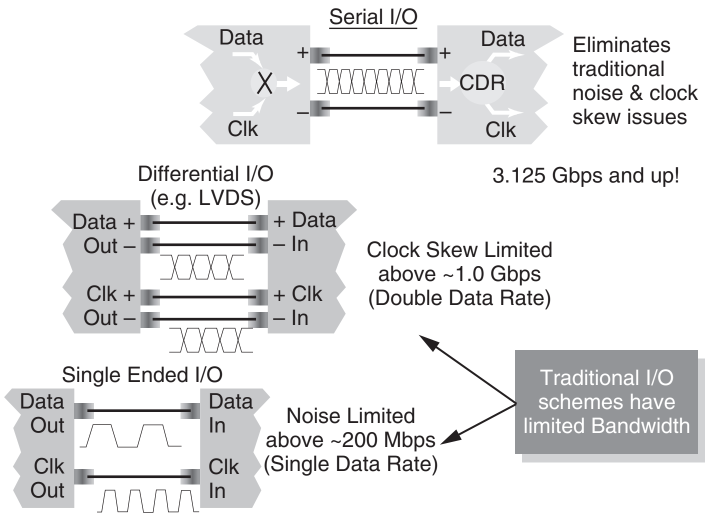
Serial I/O
Data
Clk
X
+
–
Data
CDR
Clk
Eliminates traditional noise & clock skew issues
3.125 Gbps and up!
Differential I/O (e.g. LVDS)
Data + Out –
Clk + Out –
+ Data – In
+ Clk – In
Clock Skew Limited above ~1.0 Gbps (Double Data Rate)
Single Ended I/O
Data Out
Clk Out
Data In
Clk In
Noise Limited above ~200 Mbps (Single Data Rate)
Traditional I/O schemes have limited Bandwidth

FIGURE 7.11 Serial I/O with clock recovery capability vs. source synchronous single-ended and differential I/O schemes. (Courtesy of Xilinx Inc.)

originated thousands of miles away. Therefore, it is easy to imagine that there is a need for standardization of data formats, data interfaces, and many other aspects of the communication system which have to be considered while designing an integrated circuit for communications. The networking standards make it easier for VLSI designers to derive specifications for the chip they design. This means that the specification of the chip begins from some common, internationally predetermined conditions within which the device has to operate. For example, transmitting data in certain formats dictates the frequency at which the data are transferred. The interfaces are agreed upon in different standard bodies that consist of many engineers from around the world, on many occasions from competing companies. Because of this standardization effort, various IC components can talk to each other regardless of who designed them. To illustrate the standardization concept, we discuss some of the networking I/O interfaces developed by the Optical Internetworking Forum (OIF). Although we use OIF standards here almost exclusively, keep in mind that many other standards and proprietary schemes do exist.

Optical Internetworking Forum Standards[†] The mission of the OIF is to foster the development and deployment of interoperable data switching and routing products and services that use optical networking technologies. To promote multivendor interoperability at the chip-to-chip and module-to-module level, the physical and link layer (PLL) working group within the OIF

[†]The text in this section is adapted directly from the OIF Forum Web site: http://www.oiforum.com/.

has defined electrical interfaces at two different points within a synchronous optical network/synchronous digital hierarchy (SONET/SDH) based communication system. These interfaces, the system packet interface (SPI) and the SERDES framer interface (SFI), are depicted in the reference model shown in Figure 7.12. SPI and SFI I/O physical hardware correspondence to the OSI model is also shown in Figure 7.13.

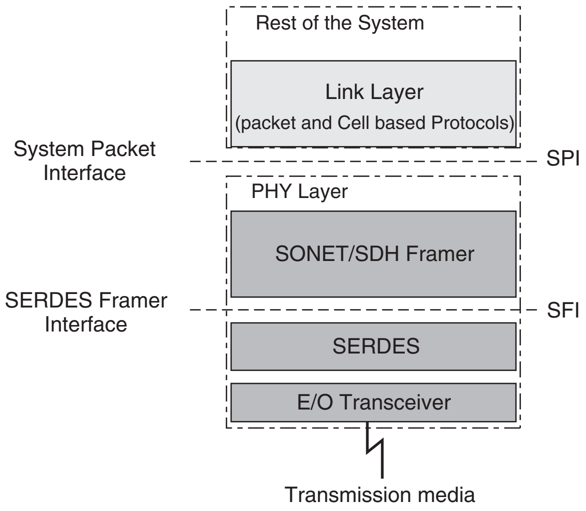

FIGURE 7.12 SPI and SFI I/O reference model adopted by OIF.

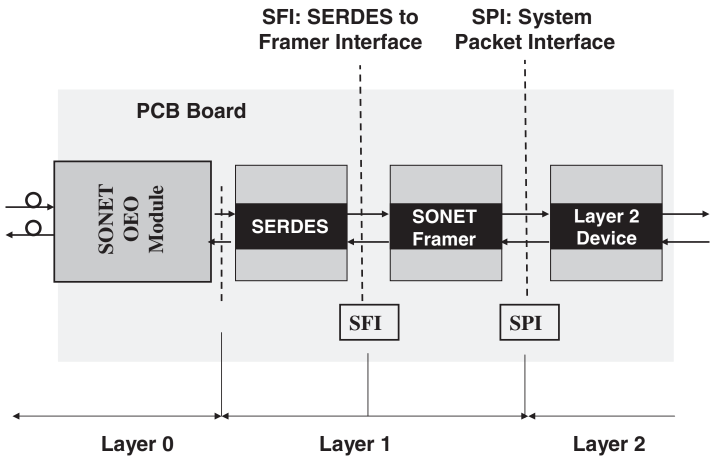

FIGURE 7.13 SPI and SFI interfaces in a printed circuit board hardware implementation.

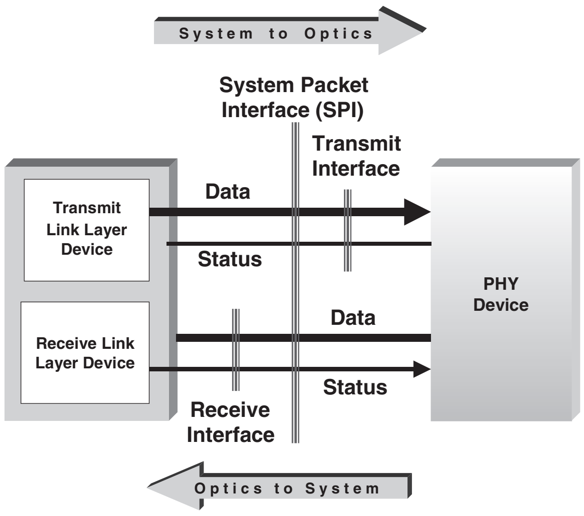

FIGURE 7.14 System packet interface between a framer and a link layer device adopted by OIF.

The SPI is found between the physical layer (PHY) device(s) and the rest of the SONET/SDH system (i.e. between the SONET/SDH framer and the link layer), as shown in Figure 7.14. This interface separates the synchronous PHY layer from the asynchronous packet-based processing performed by the higher layers. As such, the SPI supports transmitting and receiving data transfers at clock rates independent of the actual line bit rate. It is designed for efficient transfer of both variable-sized packets and fixed-sized cell data.

The SERDES framer interface (SFI) defines an electrical interface between a SONET/SDH framer and a high-speed parallel-to-serial/serial-to-parallel (SERDES) logic, as shown in Figure 7.15. This permits the SERDES and framer to be implemented in different speed technologies, allowing a cost-effective multiple-chip solution for the SONET/SDH PHY. The SFI interface can also be used as an interface for an FEC chip inserted between the SERDES and the framer as illustrated in Figure 7.16.

To keep up with evolving transmission speeds and technology enhancements, the OIF has defined several different versions of these electrical interfaces. Each version is tailored for a specific application environment and time frame. The OIF has adopted the following naming conventions to identify the various PLL interfaces and the application environments they are designed for:

- S*x*I-3 OC-48/STM-16 and below (2.488 Gb/s range)
- S*x*I-4 OC-192/STM-64 (10 Gb/s range)
- S*x*I-5 OC-768/STM-256 (40 Gb/s range)

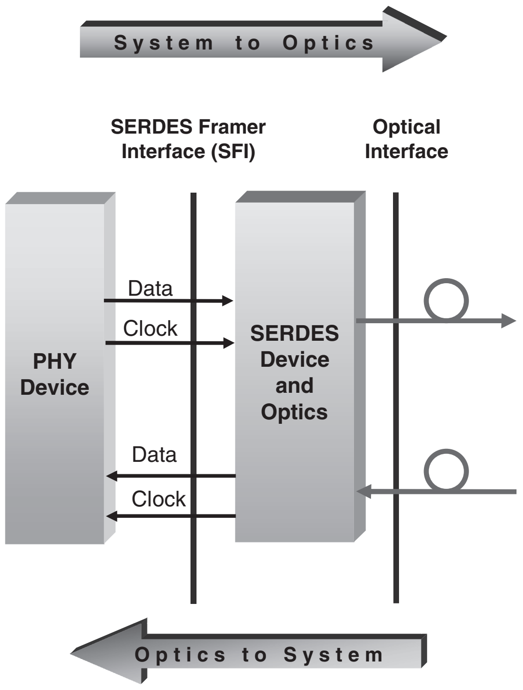

FIGURE 7.15 SERDES framer interface adopted by OIF.

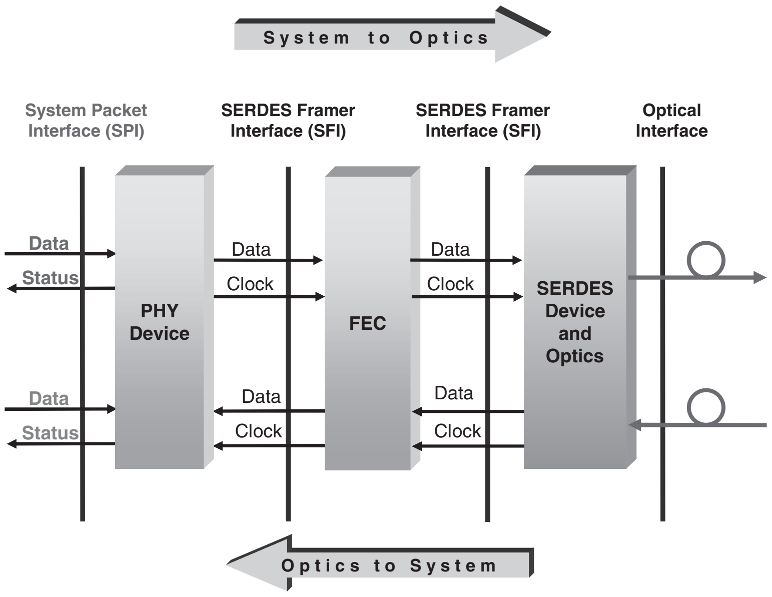

FIGURE 7.16 Forward error correction chip inserted between the SERDES and the framer.

where *x* can be either F (framer) or P (packet) in S*x*I. Both SONET and SDH terminology are adopted in the following manner: OC, optical carrier (SONET terminology); and STM, synchronous transport module (SDH terminology). To illustrate some attributes of SPI and SFI interfaces, we discuss briefly their 10-Gb/s implementations.

SPI-4 OC-192 Interface SPI-4 is an interface for packet and cell transfer between a PHY device and a link layer device that runs at a minimum of 10 Gb/s and supports the aggregate bandwidths required of ATM and packet over SONET/SDH (POS) applications. SPI-4 specifies independently its transmitting and receiving interfaces that allow more flexibility in the design of higher-layer devices. SPI-4 is well positioned as a versatile, general-purpose interface for exchanging packets anywhere within a communications system. The SPI-4 interface, illustrated in Figure 7.17, has the following attributes:

- Point-to-point connection (i.e., between single PHY and single link layer devices). Variable-length packets and fixed-sized cells. Transmit/receive data path that is 16 bits wide.
- Source-synchronous double-edge clocking with a 311 MHz minimum. 622 Mbps minimum data rate per line. Low-voltage differential signaling (LVDS) I/O (IEEE 1596.3–1996, ANSI/TIA/EIA-644–1995).
- Control word extension supported. In-band PHY port addressing. Support for 256 ports (suitable for STS-1 granularity in SONET/SDH applications (192 ports) and fast Ethernet granularity in Ethernet applications (100 ports)). Extended addressing supported for highly channelized applications. In-band start/end-of-packet indication, error-control code.

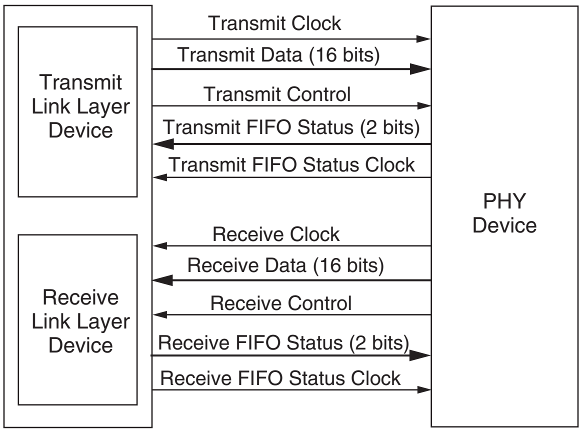
Transmit Link Layer Device
Receive Link Layer Device
PHY Device
Transmit Clock
Transmit Data (16 bits)
Transmit Control
Transmit FIFO Status (2 bits)
Transmit FIFO Status Clock
Receive Clock
Receive Data (16 bits)
Receive Control
Receive FIFO Status (2 bits)
Receive FIFO Status Clock

FIGURE 7.17 SPI-4 interface for 10 Gb/s.

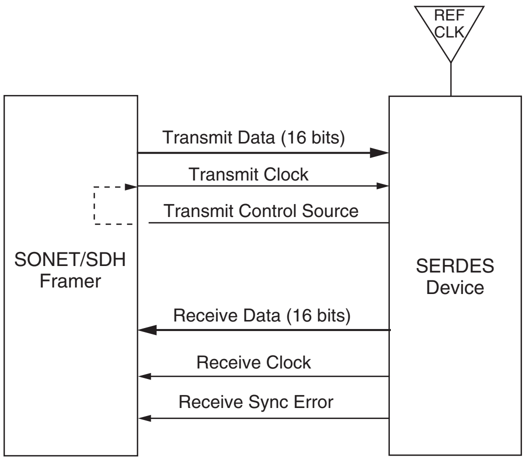

FIGURE 7.18 SFI-4 interface for 10 Gb/s.

- Transmitter/receives FIFO status interface; 2-bit parallel FIFO status bus; in-band start-of-FIFO-status signal; Source-synchronous clocking.

SFI-4 OC-192 Interface The SFI-4 interface supports transmitting and receiving data transfers at clock rates locked to the actual line bit rate of OC-192. It is optimized for the pure transfer of data. There is no protocol or framing overhead. Information passed over the interface is serialized by the SERDES and transmitted on the external link. The SFI-4 interface, illustrated in Figure 7.18, has the following attributes:

- Point-to-point connection (i.e., between single framer and single SERDES devices). Transmitter/receives data path that is 16 bits wide. Source-synchronous clocking at 622.08 MHz.
- 622 Mbps minimum data rate per line. An aggregate of 9953.28 Mbps is transferred in each direction. Timing specifications allow operation up to 10.66-Gbps LVDS I/O (IEEE Std 1596.3–1996) in order to accommodate 7% FEC overhead.
- The 622.08-MHz framer transmit clock sourced from the SERDES. Uses a 622.08-MHz LV-PECL reference clock input. A low-speed receive loss of synchronization error is signaled when the receive clock and receive data are not derived from the optical signal received.

7.3.3 Design of Data Networking I/O Interfaces

As described above, many serial interface standards are used in networking devices. In this section we sketch how one might go about designing the simple

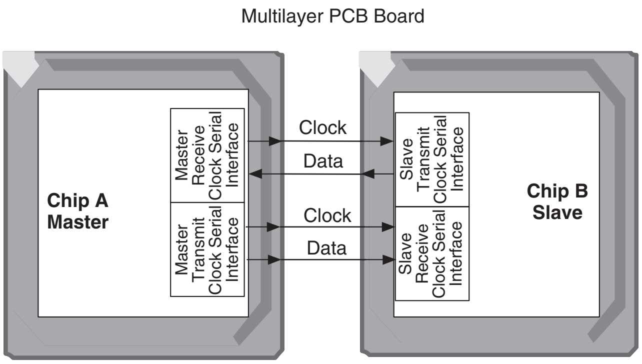
Multilayer PCB Board
Chip A Master
Master Receive Clock Serial Interface
Master Transmit Clock Serial Interface
Clock
Data
Clock
Data
Slave Transmit Clock Serial Interface
Slave Receive Clock Serial Interface
Chip B Slave

FIGURE 7.19 Master–slave source synchronous serial interface.

clocked serial interface shown in Figure 7.19. That serial interface will facilitate data communication between master and slave devices and requires a clock and data pair of signals in each direction (transmitting and receiving). As the name indicates, a master device controls the timing of the data transfer. Being the master in this case means initiating the clock used for data transfer.

Let us consider the task of designing a simple slave receive clock serial interface. Let us assume that we have already gone through the specification phase and know the block diagram surrounding the interface we are going to design. We also know the interfacing signal behavior as well as the timing of the interface signals. With these assumptions we can place the block we are designing, let us call it RxSerial, within the networking chip as illustrated in Figure 7.20. Note that all clock signals are distributed by a specialized module called a clock distribution block, which ensures the ease of the clock network design.

In the networking architecture, data are usually processed in bytes or words rather than serially. This is why the serial interface converts the serial bit stream of data into parallel sequences of bytes or words, which are presented to the rest of the device at much lower speed. Its function typically includes synchronization of the data to the internal system clock. This synchronization task is performed by means of a first-in first-out (FIFO) block, which is included in the interface circuitry.

In our example the system clock (SysClock) is decoupled from the timing of the data received; therefore, the only synchronization requirement that we have is related to data bandwidth [i.e., the frequency of the SysClock has to be higher than one-eighth that of the frequency of the RxClock]. This requirement is a result of the serial data being converted to octets of data passed to the logic interfacing RxSerial. Of course, due to other requirements of the chip, we may need to run SysClock at a higher frequency; therefore, together with SysData[7:0], we need a SysDataEn signal which indicates to the rest of the logic when the output data are valid. Figure 7.21 is an RxSerial implementation block diagram. The circuitry operates as follows. The serial data stream RxData

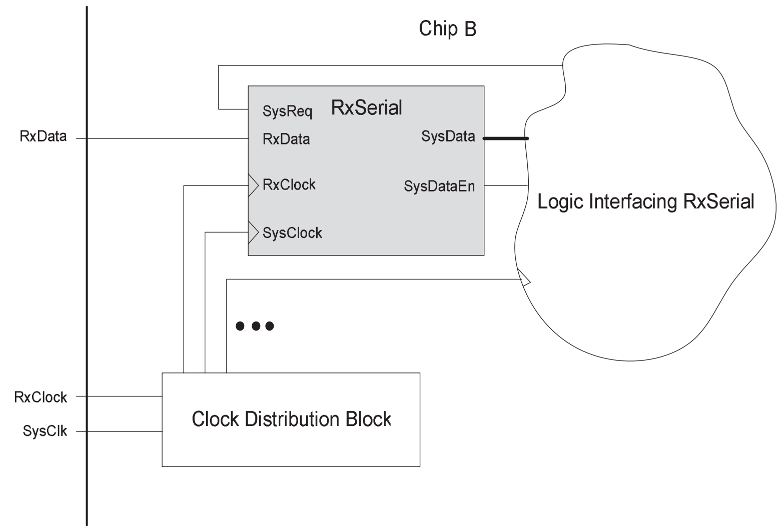

FIGURE 7.20 RxSerial design block.

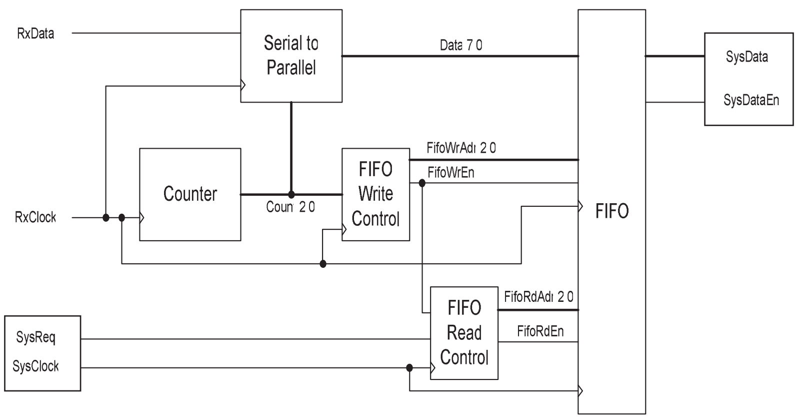

FIGURE 7.21 RxSerial implementation block diagram.

is clocked by the RxClock signal. A free-running counter counts each bit as it is collected in the serial-to-parallel converter. The most common implementation of this converter is a shift register. The counter value is used to control both the serial-to-parallel converter and FIFO write control.

When each time octet of Data[7:0] is ready, the Fifo Write control issues a FifoWrEn indicator and increments FifoRdAdr[2:0].[†] The write frequency to

[†]In our example the FIFO depth is 8. In general, FIFO depth depends on system requirements.

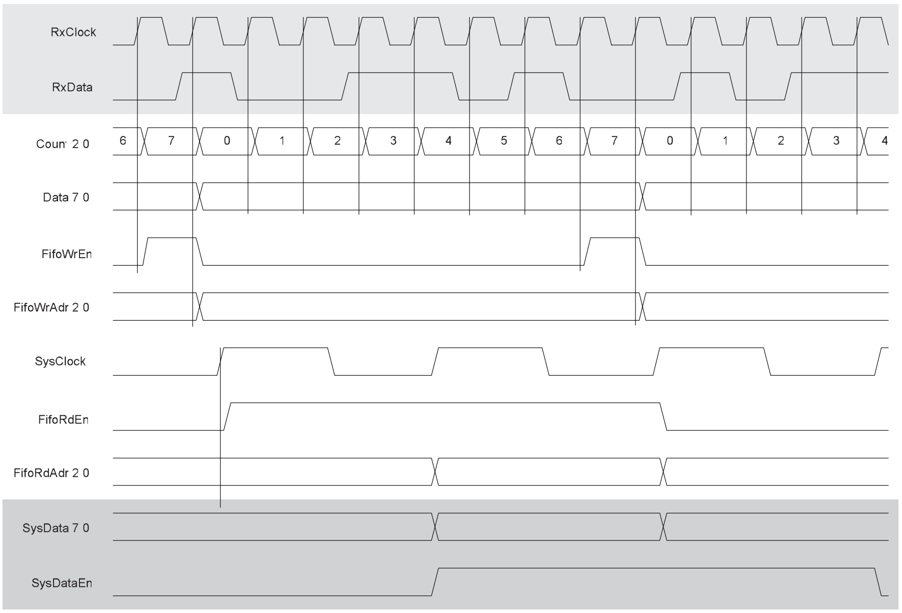

FIGURE 7.22 Timing diagram for the RxSerial operation.

the FIFO is predetermined by the RxData bit rate and has priority over FIFO read operations. FIFO is read each time there is a request from the logic interfacing RxSerial and there is something to read from it (i.e., FIFO is not empty). The FIFO not-empty condition is determined in the FIFO read control, which issues a FifoRdEn signal and generates FifoRdAdr[2:0]. The SysData read from the FIFO is passed to the rest of the core logic. The behavior of the RxSerial circuit, with individual signal behavior, is presented in the timing diagram of Figure 7.22. The shaded waveforms at the top and bottom belong to the external signals of the RxSerial block, while the remaining waveforms are internally generated signals.

7.3.4 Memory I/O Interfaces

To perform frame/packet processing the networking ICs need to store the processed data in memory elements. If these storage requirements are not too severe, the storage operation can be performed using internal on-chip memories. However, if the memory requirements are large, the external memory has to be used. As a result, many networking ICs need to interface to the external memory chips. External memories can be SRAMs or DRAMs. SRAM memories are faster but offer lower density and cost more than DRAMs. Memory interfaces are somewhat different than the data interfaces discussed previously. In this section we examine briefly the basic properties of memory I/Os.

The first important memory I/O standard, enhanced data out (EDO), was introduced in the mid-1990s. EDO provided an asynchronous operation, meaning that the output data were not aligned with a column access signal. The frequency of operation was less than 50 MHz. The I/O was specified to operate as a 5-V logic, had nonterminated signaling, and represented input capacitance of each memory pin to be CIN = 5 pF.

To improve I/O speed the next-generation synchronous DRAM interface (SDRAM) introduced synchronous operation. Data were sampled on the rising edge of the clock, and frequency of operation was in the range 66 to 133 MHz. Part of the speed improvement was due to the lower capacitive load of CIN = 2.5 pF. SDRAM I/O is still used today, although faster interfaces such as DDR I and II are available. DDR uses both the rising and falling edges of the clock to transport data. Hence, even with the same clock frequency as SDRAM, we can double the data rate. For example, a DDR200 provides the following transfer rate:

$$\begin{aligned}\text{data rate} &= (\text{bus clock frequency})(2 \text{ for dual data rate})(8 \text{ bytes per cycle})\\ &= (100\,\text{MHz})(2)(8 \text{ bytes}) = 1600\,\text{MB/s}\end{aligned}$$

Note that in the DDR200 designation, the three numbers represent the effective clock rate of the memory. The effective clock rate is given by the actual clock frequency used (100 MHz in the example above), multiplied by 2. DDR I technology has been specified to work at a supply voltage of 2.5 V while DDR II of 1.8 V. With the high-data-rate requirements, DDR uses an I/O interface called stub series terminated logic.

Further improvements in speed are obtained in DDR II by having the clock run at double the rate of the data. Also, the output data are aligned with the clock signal using a delay locked loop (DLL). A frequency of operation of 333 to 400 MHz is possible using 1.8-V terminated signaling with a low input capacitance of CIN = 1.0 pF and a programmable on-die termination (ODT) to minimize signal integrity problems. A comparison of SDRAM, DDR I, and DDR II is presented in Figure 7.23. It is interesting to note that as memory I/O schemes have evolved, the power supply has decreased (Figure 7.24) and the memory bandwidth has increased (Figure 7.25). These trends are typical for VLSI and partially reflect improvements in underlying transistor technology and, in part, innovations in circuit design techniques over the years.

In closing this memory I/O discussion, it is worth pointing out that DDR III, quadruple date rate, and Rambus I/O schemes promise further improvements in memory I/O performance. In fact, data rates as high as 10 Gb/s per pin have already been demonstrated in the lab using sophisticated DLL/PLL techniques, as shown in Figure 7.26.

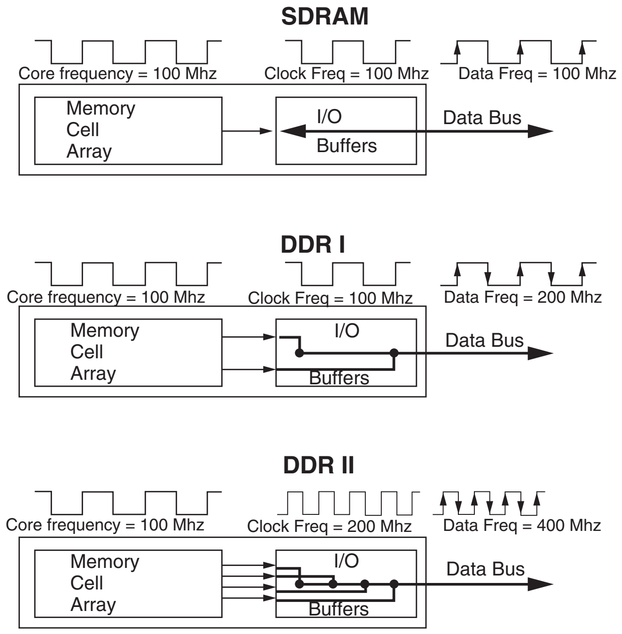

FIGURE 7.23 Comparison of SDRAM, DDR I, and DDR II.

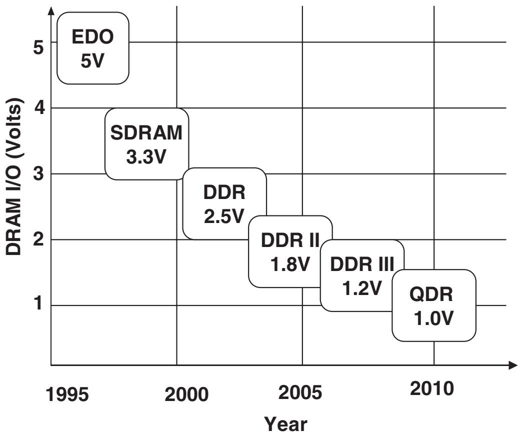

FIGURE 7.24 Evolution of the DRAM I/O power supply.

7.3.5 Microprocessor I/O Interfaces

A microprocessor interface serves as an interface to the higher levels of the OSI reference model. It is used to configure the networking device as well as to collect status information from the data received through the external

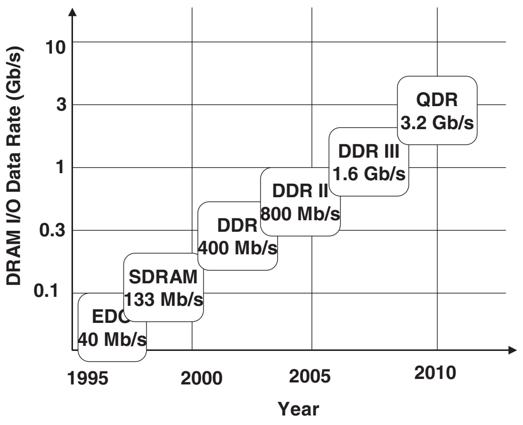

FIGURE 7.25 Evolution of the DRAM I/O data rate.

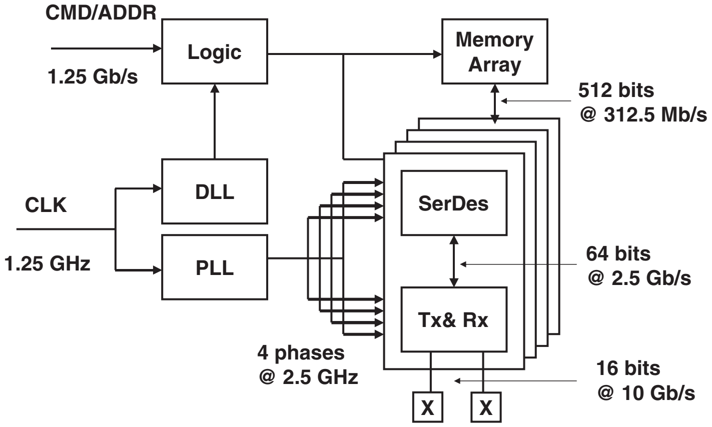

FIGURE 7.26 Internal architecture of a 10-Gb/s DRAM I/O scheme. (From Kim et al., 2005.)

interfaces. The choice of the processor used in various networking applications dictates the support of a variety of microprocessor interfaces designed in a networking chip. Some examples of the microprocessors that are used as an interface between software and networking devices are the Motorola MPC860, Motorola MC68302, and Intel processors with Ebus interface. A typical external microprocessor bus includes address, data and control signals. The control signals are responsible for indicating read or write operation, chip select, data burst, and data or address indicators when data and address are sharing the bus. The microprocessor interfaces can be asynchronous or synchronous, both solutions having advantages and disadvantages.

In the larger and more complicated networking devices, there are a very large number of configuration settings that can be controlled by a microprocessor. These settings are stored internally in the configuration registers. Similarly, the status of the data stream extracted by the networking device is stored in the internal status registers so that it can be read by the microprocessor when needed. The availability of the status data is usually communicated via interrupt control circuitry. Both configuration and status registers can be implemented as latches, flip-flops, or internal RAM bits. These registers are usually associated with the operation of individual blocks. Each register is given a specific address that is used in the same way as the memory address in RAMs. That is why registers within a chip are often referred to as memory maps. To access configuration and status registers, a microprocessor issues the underlying address within the memory map. Each chip specification describes this memory map in detail, in much the same way that it describes the external pin description.

The microprocessor interface of a networking device serves as a bridge between external microprocessor and internal microcontrollers of individual blocks. This is done for design portability, design reuse, and better design partitioning, which leads to a faster design cycle. A typical configuration of a microprocessor interface is shown in Figure 7.27. In the high-aggregation networking devices where the amount of information transferred to the higher OSI level is very high, there may not be enough bandwidth to communicate it through a simple microprocessor interface. The solution to this bottleneck is to move some of the software computations to the networking chip by including an embedded processor. The microprocessor interface is then moved to the internal part of the chip, allowing for faster communication between embedded processor and internal registers. At the same time, the external software interface is based on the instruction set understood by the embedded processor used.

7.4 EXAMPLES OF CHIP ARCHITECTURES

7.4.1 Time-Slice Architecture

In most applications the data passing through the VLSI digital networking devices is time-division multiplexed. The time multiplexing means that the data contain channels of information distributed uniformly in time. These channels of data are usually processed separately. For typical networking IC architectures this means that there are pieces of logic designed for one channel that are replicated a number of times in the chip and which perform the same function on all channels in parallel. A typical example of such

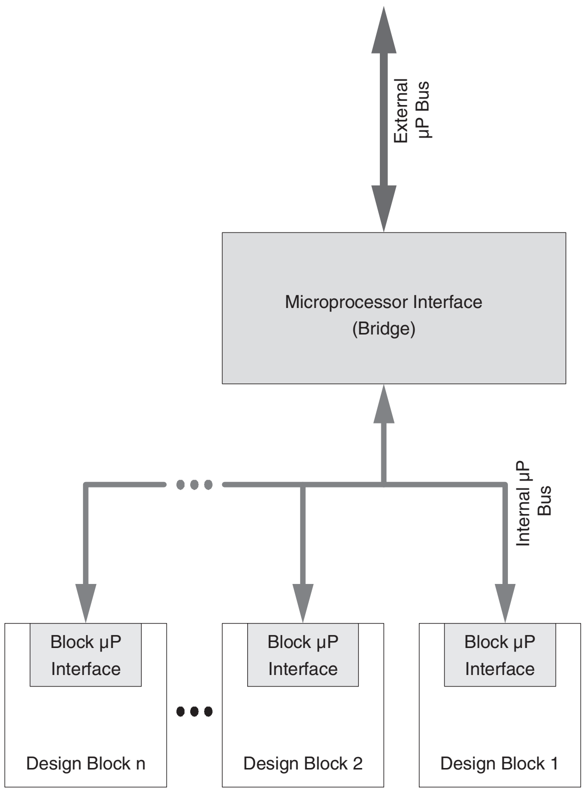

FIGURE 7.27 Block diagram of a typical microprocessor interface internal implementation.

architecture is the processing of the STS-12 SONET frame in the framer device where the time-division-multiplexed individual STS-1 subframes are processed independently in 12 parallel blocks (Figure 7.28). The parallelism of such data processing has a number of advantages. First, it is simple to implement. Second, one individual parallel stream can be processed at much lower clock speed than the original data stream. For example, an STS-12 SONET framer running at a bit rate of 622.08 Mb/s must have a clock of 77.76 MHz to process STS-12 octets of data. On the other hand, if the STS-12 data are demultiplexed into 12 STS-1 channels, the clock handling each channel is 6.48 MHz. This makes the design easier with a smaller switching power density and a more uniform current flow.

There is, however, another architecture that more naturally fits the nature of time-division-multiplexed data; the general idea behind this architecture, called *time-slice architecture*, is to reuse the combinatorial part of the logic for all the channels and to use channel addressable memory to store the states needed to calculate the next state for each channel. Consequently, the states

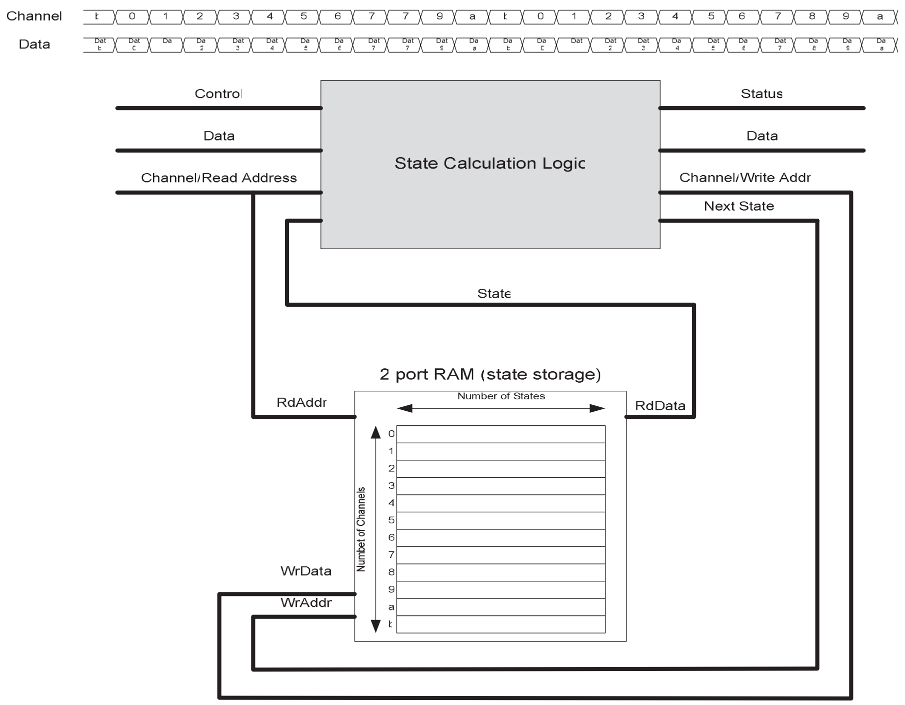

FIGURE 7.28 Generic time-slice architecture.

are stored again as if the set of flip-flops were used. This architecture works with any type of channelized data; however, one needs to evaluate its advantages versus the parallel processing architecture described above before making the final decision.

Generic time-slice architecture is presented in Figure 7.28. The time-division-multiplexed receive data present at the input of the time-sliced design can come at the predetermined order defined by a particular standard (as in the case of SONET/SDH) or can be received in any order. When the order at which data belonging to a particular channel is not known, the channel indicator input is necessary to determine the association of the data with a particular channel, as indicated in Figure 7.28. In addition to the data path being processed by the state calculation logic, the configuration control bus controls the desired behavior of the logic on a per channel basis. These control signals are inserted to the state bit words stored in the two-port RAM at specific locations and act as configuration register bits in the parallel architecture. The control signals can be provided directly by the microprocessor interface or by the control bus of some other block within the chip. Similarly, the status bits of individual channels can be presented to the outside world as sets of register status bits, or they can control the next block in the data path.

Some of the possible uses of time-sliced architecture include ATM cell delineation and processing logic, DS1/E1, DS3 framers, M13 multiplexers, and STS framers.

7.4.2 SONET Framer Architecture

In this section we discuss the architecture of a typical IC used to process SONET frames in order to show a more specific implementation of integrated circuits in data networking. A device like this, called a *SONET framer*, is shown in Figure 7.29. On one side the framer is connected through a SERDES device into an optical module using the SFI interface. This side is called the *line side*, as ultimately the signal is pushed into an optical fiber. On the other side, the framer is connected to a higher-layer device link, a link layer device, or a mapper. In some applications it may be connected directly to a switching fabric. As more frame processing is done at this side, it is called the *system side*.

A SONET framer IC has two paths, in the receiving and transmitting directions, which are largely independent of each other. The only dependencies come from various loop-back modes, where the data can be looped from a receiving side to a transmitting side, or vice versa, for various testing and diagnostic purposes. In a normal mode of operation the data received from the optical module (the lower path in Figure 7.28) is received by the SFI I/O block and after SONET processing is transmitted at the SPI interface I/O. Similarly, the data received from the system side (the upper path in Figure 7.28) is received by the SPI I/O block and after SONET processing is transmitted at the SFI I/O interface.

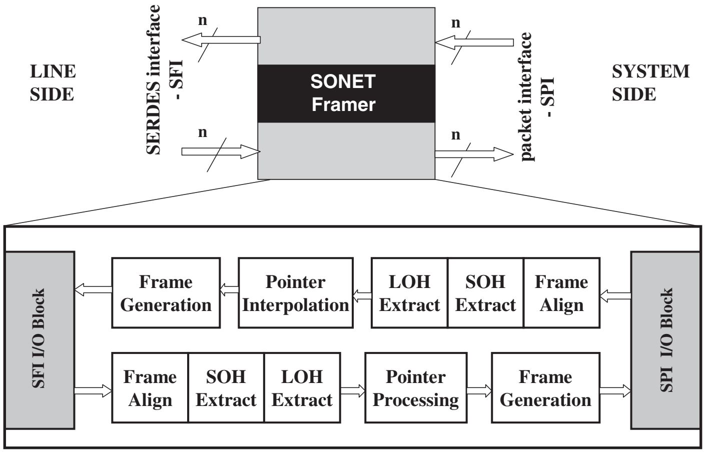

FIGURE 7.29 SONET framer chip.

SONET processing implies full analysis of all overhead bytes in the section, line, and path overhead. When the data are received, the first operations of the framer are to find A1 and A2 bytes and to find the position of the frame. Once the start of the frame is known, the section and line overheads can be extracted. Bytes H1 and H2 in the line overhead indicate the position of the path overhead. The operation of finding the position of the synchronous payload envelope is typically referred to as *pointer processing* in a downstream direction or *pointer interpolation* in an upstream direction.

Typical IC implementation supports SONET/SDH alarm detection and insertion functionality as well as bit-interleaved parity processing. Section, line, and path overhead are extracted and can be sent to the external control processor. Similarly, section, line, and path overhead can be supplied by the external processor and used to override the internal values. These features enable additional flexibility in system implementation where the external processor can implement additional, proprietary functions.

One can build various pieces of SONET networking equipment by using SONET framers and other ICs. As an example, schematic architecture of the SONET add–drop multiplexer (ADM) is shown in Figure 7.30; the individual line card was shown in Figure 7.13. Needless to say the drawings show only the key components of the system. For clarity, various additional pieces, such as power supplies, switches, reference clocks, and E^{2}PROM memories, are not shown. The ADM consists of various line cards for working and protecting lines that connect to pairs of optical fibers. In Figure 7.30 the WEST and EAST links both contain two pairs; in addition, various tributary cards represent lower-rate data streams used to insert or drop SONET traffic into this WEST–EAST connection. The heart of this system is the cross-connecting platform,

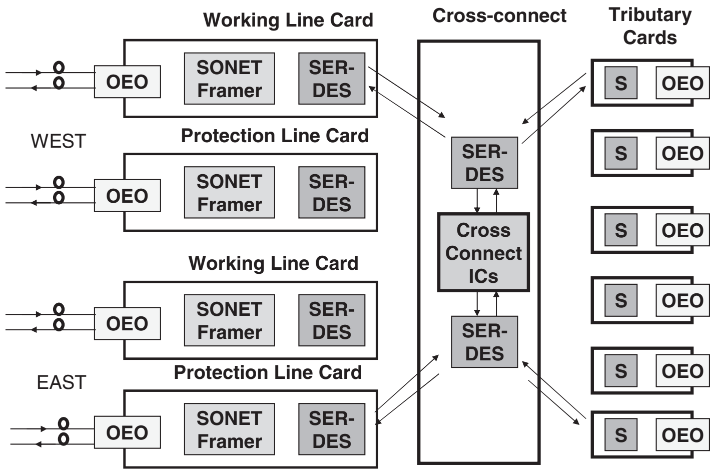

FIGURE 7.30 Schematic architecture for a SONET add/drop multiplexer.

where traffic from various line cards is cross-connected to appropriate outputs as required by information contained in the SONET frame overhead. SONET networking gear is described in more detail in Chapters 5 and 13. Here we just wanted to highlight how SONET integrated circuits might fit into a larger system.

7.4.3 Network Processor Architecture

As mentioned earlier, a network processor is a networking IC that is programmable through software. It offers higher flexibility compared to the hardwired ASIC, at the cost of slower processing speed. A network processor performs packet header operations, such as packet parsing, modification, and forwarding, between the physical layer interface and the switching fabric. The major advantage of a network processor is that it features one common hardware platform that can be utilized in different networking cards and various networking boxes. The major drawback of network processor solutions is difficulty keeping up with the line rate, as the line rate keeps increasing.

To illustrate network processor architecture, let us have a look at one of the early Intel processors, the IXP1200.† The Intel IXP1200 network processor is a highly integrated, hybrid data processor that delivers high-performance parallel processing power and flexibility to a wide variety of networking, communications, and other data-intensive products. The IXP1200 is designed specifically as a data control element for applications that require access to a fast memory subsystem, a fast interface to I/O devices such as network MAC devices, and processing power to perform efficient manipulation on bits, bytes, words, and longword data. As illustrated in Figure 7.31, IXP1200 combines the StrongARM processor with six independent 32-bit RISC data engines with hardware multithread support that, when combined, provide over 1 giga-operation per second. The microengines contain the processing power to perform tasks typically reserved for high-speed ASICs. In LAN switching applications, the six microengines are capable of packet forwarding of over 3 million Ethernet packets per second at layer 3. The StrongARM processor can then be used for more complex tasks, such as address learning, building and maintaining forwarding tables, and managing the network.

The chip contains on-chip FIFO with 16 entries of 64 bytes each. It contains a 32-bit-wide PCI interface that communicates with other PCI devices. The IXP1200 interfaces to a maximum of 256 MB of SDRAM over a 64-bit data bus. A separate 32-bit SRAM bus supports up to 8 MB of SRAM and 8 MB of read-only memory (ROM). The SRAM bus also supports memory-mapped I/O devices within a 2-MB memory space. The 64-bit data bus supports the attachment of MACs, framers, or other custom logic devices, and an additional IXP1200.

†More information about this network processor and more advanced Intel chips can be found at the following Web site: http://www.intel.com/design/network/products/npfamily/index.htm.

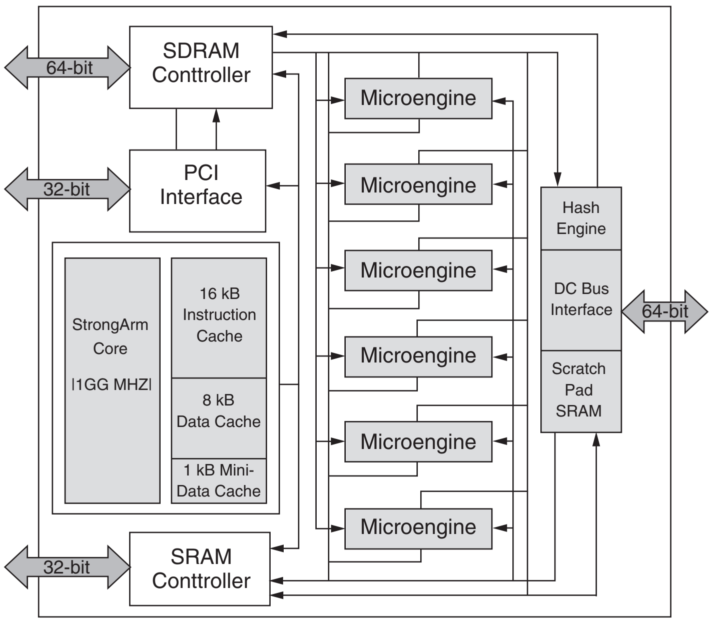

FIGURE 7.31 IXP1200 network processor. (Courtesy of Intel Inc.)

Implemented in 0.28-μm CMOS processes, the chip contains over 6 million transistors, dissipates about 5W, and is packaged in a 432-pin ball grid array. The silicon die photo is shown in Figure 7.32. As can be seen from this description, even this simple IC is quite complex, although it delivers only 1-Gb/s data-throughput capabilities. One can imagine the complexities involved in designing a network processor for OC-192 applications that would require a data throughput which is higher by two orders of magnitude.

7.5 VLSI DESIGN METHODOLOGY

The process of VLSI chip design is a lengthy systematic process that consists of many tasks performed by numerous specialized engineers. A modern IC might require 30 to 50 engineers working together for a period of one to two years to complete the design. At the end of this process a chip containing several million transistors will have been created. One can easily imagine the complexities involved and requirements to partition the design work.

The partitioning of tasks and distribution of them among design engineers seem to become more important as the level of integration marches along Moore's law. Historically, the design process can be divided into two main groups of tasks: front-end design and back-end design. The front-end design consists of design concept and architecture, design entry, and functional

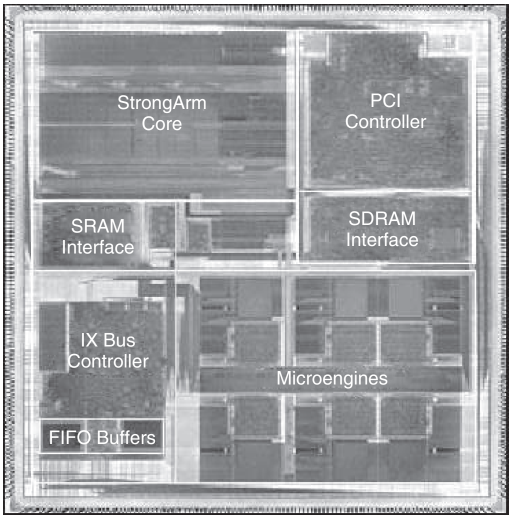

FIGURE 7.32 IXP1200 silicon die. (Courtesy of Intel Inc.)

verification. Back-end design takes on the data created by a front-end design process and creates the physical topology of an integrated circuit using rules for the particular manufacturing technology available.

VLSI design methodology is highly dependent on the technology used in the process of IC fabrication. The constant advance in fabrication process technology introduces new problems that have to be addressed by designers through the means of computer-aided design (CAD) automation. These problems are associated with power consumption and power distribution, signal integrity, accuracy of timing analysis, testability issues, design verification, and so on. The use of specialized design automation tools in solving the problems described above calls for specific sets of design rules that a designer has to be aware of while designing. Together with CAD tools, these design rules are constantly updated with the advances in IC process technology.

Design methodology depends on the fabrication path chosen. Typical examples of design methodologies classified based on fabrication path are custom IC and standard cell and gate array methodologies. Custom cell designs are used in leading-edge parts requiring special attributes, such as very high speed, low power, or small chip area. These designs are made of custom layout cells placed manually in the circuit topology. Once the custom design layout is finished, it can be used as a hard macro cell connecting to the rest of the chip's topology. This methodology is often used in implementing parts of microprocessors and usually requires knowledge of both front- and back-end design tasks.

Standard cell design is much more common than custom design. It relies on the library of standard cells that perform basic logic operations such as *and,* *or*, and *exclusive or*. These cells have a common physical topology height so that they can be arranged arbitrarily in rows of cells. After all the cells are placed, they can be connected through a network of wires (nets) using a process known as *routing*. Both placement and routing are based on a front-end design-produced *netlist*, a text representation of the design schematic using standard cells as basic components.

Finally, a third approach, gate array design methodology, is very similar to standard cells except the cells are prearranged and prefabricated on the wafer. Once the front-end netlist is generated, it is used to determine association of the prearranged cells with the gates used in the netlist (this process is still called *placement*). Subsequently, the nets are routed and the routed nets (metal layer wires) are superimposed on the prefabricated waver. This process is faster and less expensive than the standard cell design process but is not as efficient. As a result, ICs designed using gate array methodology are bigger, slower, and consume more power.

Based on the differences between fabrication paths described above, it is evident that the corresponding design methodologies must differ substantially as well. Different methodologies and their fabrication paths can have different economic impacts. For example, standard cell IC methodology yields smaller designs, which, in turn, can be packaged in smaller, usually cheaper packages. In high-volume production the low unit cost can offset high initial design-related costs.

In the remainder of this section we discuss a digital IC design process using a standard, cell-based design flow. The IC design process typically consists of the following stages:

1. *Design specification.* During this stage the function of an IC is determined. All architectural trade-offs are explored to determine what algorithms to use, how to partition hardware and software, what interfaces to use so that a device communicates most efficiently with the rest of the world, and the optimum CPU and memory architecture.
2. *Functional design and RTL coding.* Functional design is a process of translating very abstract design description, as written in the specification document, into a specific design entry format that is understood by CAD tools. Typically, Verilog or VHDL hardware description language is used for that purpose and referred to as RTL coding.
3. *High-level verification.* At this stage the RTL code is verified against the design intent by exhaustive simulations. This is accomplished by writing a behavioral simulation test bench within which the RTL code of the design is placed and verified. When verification is complete, the chip is ready for synthesis and physical implementation.
4. *Design synthesis.* Design synthesis comprises design compilation and mapping to a particular technology and subsequently optimizing the result so that it meets the timing, area, and power constraints imposed

by the design specifications. Usually, test circuitry is inserted during that stage. A gate-level netlist is produced as the result of that design stage.

5. *Physical design and verification.* At this stage the gate-level netlist is transferred into a physical representation of the chip. The process is usually iterative and involves cell placement and detailed metal routing. Once the design is placed and routed, the resulting physical layout needs to be verified for correctness against the transistor-level netlist. The process is known as *layout vs. schematic* (LVS) *verification*. In addition, the physical layout needs to be verified against the design rules required by wafer manufacturing. This process is called *design rule checking* (DRC). Chip-level LVS and DRC are final steps before sending the design in the form of GDSII file for mask making and subsequent manufacturing.

A flowchart showing the steps described above is often used to capture the design methodology adopted by a design team. The chart, frequently referred to as a *design flow*, shows the relationship between various tasks so that it is easier to coordinate the work of various members of the design team. A typical simplified design flow represented graphically is shown in Figure 7.33. Note the recursive nature of the design flow. Each of the design tasks includes an initial stage followed by the verification and, consequently, modification of the

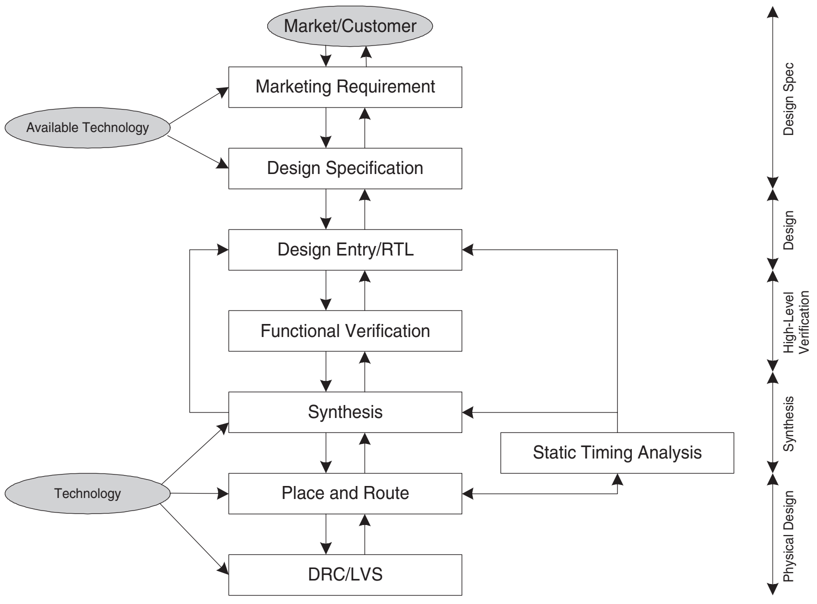

FIGURE 7.33 Typical simplified standard cell design flow.

original task. The design efficiency is highly dependent on the number of iterations performed within the design task or a set of tasks. With an assumption of the recursive nature of the design cycle, the design flows are planned to adopt concurrency of the design process. In the design flowchart, the tasks are often linked to appropriate CAD tools that are used in the design process. This is to communicate promptly to the design team the tools to be used in the design process.

7.5.1 Design Specification

A design of networking chips does not differ much from the design of many other types of ICs used in a variety of applications. The first step is always to create a design specification that determines the function to be performed by a device. The specification is based on the system requirements. The system, to a large extent, determines the partition of the tasks performed by hardware and software. Within the hardware part of the system, one can partition a set of functions to be performed into one or more ICs. The last partition depends heavily on the technological limitations as well as economic factors. Architecting the design from the specifications means that one has to partition the design into smaller blocks of logic that can be handed to individual designers. With the advent of design automation, a chip architect has to evaluate possible design architectures using CAD tools. Once the architecture of a chip has been determined, a set of specifications has to be written to cover block design issues.

7.5.2 Functional Design and RTL Coding

Functional design is the process of translating very abstract design descriptions, as written in the specification document, into a specific design entry format that is understood by CAD tools. It can be further transformed under the guidance of a designer into a circuit topology that at the end is used in an IC fabrication process. Various design entry formats are in use today, but the majority of digital VLSI circuits are designed using a register transfer language (RTL).

There are two mainstream RTL choices (with some variations of each): VHDL and Verilog. VHDL stands for very high speed integrated circuit (VHSIC) hardware description language (VHDL), and its roots lie in work done by the U.S. military in 1980. The newer RTL choice, Verilog, was originally developed by Gateway Design Automation Inc. in 1984. After years of evolution and changes in ownership, IEEE standardized both VHDL and Verilog and now they are known as IEEE Standards 1076 and 1364, respectively. For years, despite arguments by Verilog and VHDL proponents "proving" the superiority of one language to the other, both languages are equally popular in the digital VLSI design community. Examples of simple

VHDL

```
bit_counter:
PROCESS(clk, reset)
BEGIN
  IF(reset = '0') THEN
    count <= 0;
  ELSIF (clk'EVENT AND clk = '1') THEN
    IF (count = 0) THEN
      count <= dat_width-1;
    ELSE
      count <= count - 1;
    END IF;
  END IF;
END PROCESS bit_counter;
```

VERILOG

```
always @(posedge clk or posedge reset)
begin
  if (reset)
    count <= 0;
  else
    if (count == 0)
      count <= dat_width-1;
    else
      count <= count - 1;
end
```

FIGURE 7.34 Examples of binary counter RTL representations in VHDL and Verilog.

synthesizable RTL code snippets written in VHDL and Verilog representing an implementation of a binary counter are presented in Figure 7.34.[†]

Most recently, in an effort to further increase the productivity of digital designers, new language extensions were introduced to Verilog. As a result, a new language called System Verilog has emerged. System Verilog is Verilog backward compatible. In 2005 System Verilog became IEEE Standard 1800–2005. Since the IEEE standard ratification, many CAD tool vendors adopted support for System Verilog, giving a designer the ability to design and verify digital ICs using less verbose and more efficient language.

A digital circuit is a combination of logic gates, some of which are complex combinations of basic logic functions such as *and, or*, and *not* (combinational logic) as well as memory elements such as flip-flops or latches (sequential logic) which hold states of logic calculated by logic gates. Sequential logic is timed by a clock signal that alternates between states 0 and 1. The time between clock transitions is the interval dedicated for calculating states in combinational logic so that it can be stored in the sequential elements. That is why delay through the combinational logic and interconnecting nets determines the maximum frequency of the clock the circuit operates in.

The absolute value of delays through the combinational gates depends on many factors that are technology dependent. A front-end designer can, however, control the delay by introducing more sequential elements and can thereby reduce the amount of combinational logic between flip-flops or latches. The decision on how to break the combinational logic with sequential elements has to be reached early in a design cycle. At the end of the RTL coding phase the design architecture has to be evaluated against the timing requirements established in the specification document.

When an RTL code is written, a designer uses functional verification tools that include a syntax checking tool and a simulator combined with a waveform

[†]Note that the VHDL and Verilog examples do not represent complete RTL implementation of a binary counter, as they do not include declaration of ports, signals, and parameters.

viewer. These tools perform basic checks for desired functionality and enable a designer to add RTL code incrementally with a certain degree of confidence. For this to happen, a designer has to develop a basic simulation environment that consists of a stimulus generator and a basic result checker in the form of a simulation testbench. Using this environment, a designer can inspect signal waveforms within the design to make sure that the stimulus causes the desired response. RTL code that becomes part of the design functionality is written using a synthesizable subset of a given language (VHDL or Verilog), whereas testbench code used to verify design functionality is written using constructs of the language often referred to as behavioral constructs.

Apart from basic RTL design verification, a synthesis exploration phase is often required to make sure that the design can meet the electrical and physical constraints imposed by the specification. A typical RTL design process is iterative, and usually a number of iterations associated with the correction of functional behavior as well as electrical characteristics of the design are quite large.

7.5.3 Functional Verification

There are many approaches to functional verification that are used in the semiconductor industry today. Some involve hardware–software cosimulation; others involve hardware verification and platform verification approaches. We concentrate on the traditional testbench approach where the design under verification is verified within the simulation code responsible to produce input stimulus and observe the resulting response. Testbenches are traditionally written using the same hardware description language that was used to write the RTL code.

With an increasing design complexity, functional verification is gaining the reputation of being the most complex task in the digital design process. It is important that an engineer other than the original circuit designer perform the design verification. This is to ensure that the interpretation of the design specification that the RTL designer relied on is the same as the spec interpretation of the verification engineer. In complex networking chip design projects, the number of verification engineers usually exceeds the number of RTL design engineers.

The typical testbench architecture in a data networking device consists of bus functional models (BFMs) which are responsible for creating stimulus, clock generators, and reference models. These BFMs may include data generators and data analyzers with appropriate bus protocol interfaces. They also include microprocessor interface models so that the testbench can control the design under verification using the same networking protocol that will be used when the device is fabricated.

To be able to verify a design successfully with a certain amount of confidence, special simulation tools that are capable of reporting verification coverage are needed. Verification coverage may be represented in a variety of ways, including toggle coverage, line coverage, finite state machine coverage, statement coverage, and assertion coverage. By analyzing the results of the verification coverage numbers, one can conclude whether or not a particular design has reached sufficient verification coverage.

7.5.4 Design Synthesis

Design synthesis comprises design compilation and mapping to a particular technology cells specified in the technology library. The technology library describes the functions of each cell in the library so that it can be matched with compiled logical functional primitives. It also describes the electrical and physical parameters of each cell, such as load capacitance, intrinsic delays, power dissipation, and area. Based on that information, a synthesis tool has to calculate the delays through the wires connecting logical cells by taking into account a particular wire load model defined based on the area of the synthesized circuit. All of these parameters play a role in cell selection and substitution during design optimization. During design synthesis the designer has to place a set of design constraints such as clock frequency, input and output delays, input driving strength, output load, area, and power.

The two most common approaches to design synthesis are the bottom-top and top-bottom methodologies. The *bottom-top methodology* refers to synthesizing design in a particular order starting at the bottom of the design hierarchy and ending at the top level. This method was historically prescribed to large designs, due to the capacity limitations of the synthesis tools. Although synthesis capacity limitations are largely alleviated in today's synthesis technologies, many design houses maintain this methodology. *Top-bottom methodology* refers to synthesizing the entire design at once from the top level. This method is much simpler, as the design constraints are applied at the top level and propagated down the hierarchy by the synthesis tool.

As the technology geometries shrink, the synthesis tasks become more intricate and the synthesis methodologies evolve accordingly. One example of this evolution is the incorporation of the elements of physical design into synthesis flow. This is done in response to the challenge of representing wire delays in the deep-submicron technologies. Another example is that if automatic test structure insertions such as internal scan and build-in self-tests for both embedded memories and logic.

7.5.5 Physical Design and Verification

Physical-level design involves the process of taking the gate- or transistor-level netlist that represents the design schematic and producing a physical layout that will be used for IC manufacturing. *IC layout*, also known as *mask layout*, is a physical representation of the chip in terms of planar geometric shapes that correspond to the patterns of individual transistors and metal interconnect lines. Modern IC layouts are performed with the aid of IC layout CAD software in a semiautomated fashion.

The first phase of the physical layout design is floor planning. Based on the estimated sizes of all blocks and known chip architecture, the positions of individual blocks are distributed in a manner that provides the most effective signal distribution. The power distribution network and clock trees are planned at this stage. Floor planning is followed by an actual placement, where the precise positioning of blocks and cells takes place. Placement is performed iteratively with routing where the interconnections between blocks and individual cells are conducted. The final layout needs to be extracted for parasitic elements (distributed resistances and capacitance) that might affect delays between logical gates. Using extracted parasitic components, postlayout simulation and static timing analysis have to be performed to verify that the designed layout still complies with the chip specifications.

At the last stage the layout must pass a series of checks in a process known as *layout verification*. The two most common checks in the verification process are design rule checking and layout vs. schematic. When all verification is complete, the data are translated into an industry standard format called GDSII and sent to a semiconductor foundry. The process of sending these data to the foundry is called a *tapeout*, due to the fact that the data used to be shipped out on a magnetic tape. Today, the process is done electronically, but the old term remains. The foundry converts the data into another format and uses them to generate the masks used in the process of chip manufacturing.

KEY POINTS

Networking ICs:

- Networking ICs can be divided based on their functionality into layer 1, layer 2, and layer 3 devices using an ISO model as a reference.
- Layer 1 devices perform modulation and amplification of electrical signals for transmission through a physical medium (an optical fiber, a twisted copper pair, or a coaxial cable). Similarly, in the receiving direction: demodulation and equalization for reception of electrical signals. These devices are typically referred to as physical medium devices and physical layer devices and are considered to be layer 1 devices.
- Layer 2 devices format data into frames or cells using predefined protocols (ATM, Ethernet, fiber channel, SONET/SDH). These devices are typically referred to as framers or mappers.
- Layer 3 devices perform data packet processing. These processing functions include protocol conversion, packet forwarding, policing, lookup, classification, encryption, and traffic management. These devices are

typically referred to as network processors, classification engines, or data link devices. Layer 3 ICs perform in hardware networking functions assigned in the OSI model. Examples include IP routing, ATM layer policing, or fiber channel interworking with SCSI protocol in storage area networks.

- Layer 1 and 2 devices are typically implemented as ASICs. Layer 3 devices can be implemented as ASICs or as network processors, the latter providing flexibility at the cost of complex firmware. FPGAs can also be used for layer 3 processing if the required gate count is low.

I/O interfaces:

- The networking IC has data interfaces, microprocessor interfaces, and memory interfaces. The data interfaces are divided into a line side, closer to the optical fiber transmission, and a system side, close to the switching fabric core.
- A microprocessor interface is used to send control information. It includes address, data, and control signals. The control signals are responsible for indicating read or write operation, chip select, data burst, and a data or address indicator when data and address are sharing the bus.
- To enable interworking between various networking ICs, international standards are required that specify electrical conditions at the data I/O interfaces.
- The Optical Internetworking Forum (OIF) has defined two electrical interfaces within SONET-based communication systems: SFI and SPI. The OIF has published numerous implementation agreements for the SPI and SFI interfaces to which architects of networking ICs must adhere.
- The SERDES (serializer/deserializer) framer interface (SFI) defines an electrical interface between a SONET framer and the high-speed SONET SERDES logic.
- SERDES serializes (parallel-to-serial) and deserializes (serial-to-parallel) data.
- In some hardware implementations the SERDES device is integrated with the framer, effectively eliminating the SFI interface. In other implementations the SERDES device is integrated within an optical module, in which case the SFI interface becomes internal to the OEO module.
- The system packet interface (SPI) is the electrical interface between the physical and link layers in a SONET system. It separates the synchronous physical layer from the asynchronous packet-based processing performed by the higher layers of the OSI reference model. It is designed for the efficient transfer of both variable-sized packets (e.g., Ethernet) and fixed-sized cell data (e.g., ATM).

- SFI and SPI interfaces are defined for various speeds: 2.5, 10, and 40 Gb/s.

IC design process:

- The design of ICs is a complex process that comprises the following stages: design specification, functional design, high-level verification, design synthesis, and physical design with its corresponding verification.
- Design specification captures IC functionality using high-level computer language. All architectural trade-offs are being explored to determine what algorithms to use, how to partition hardware and software, and the optimum central processing unit and memory architecture. The RTL (register transfer language) description is the final result of this design stage.
- Functional design is the process of translating very abstract design description, as written in the RTL specification document, into a specific design entry format that is understood by CAD tools. Typically, Verilog or VHDL hardware description language is used for RTL coding.
- High-level verification is used to verify the VHDL/Verilog code against the design intent.
- Design synthesis comprises design compilation and mapping to a particular technology. A gate-level netlist is being produced as the result of that design stage.
- Physical design takes the gate-level netlist and transfers it into a physical representation of the chip. At the end of this process, the final layout, captured as a gdsii file, is complete and is used later for mask making.

Acknowledgments

We would like to acknowledge major contributions of Jerzy Świć, of Silicomotive Solutions, to this chapter.

REFERENCES

Intel network process Web site: http://www.intel.com/design/network/products/npfamily/index.htm.

Kim, K., et al., A 20 GB/s 256 Mb DRAM with an inductorless quadrature PLL and a cascaded pre-emphasis transmitter, presented at the IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, 2005.

Liu, S., J. Kramer, G. Indiveri, T. Delbrück, and R. Douglas, *Analog VLSI: Circuits and Principles*, MIT Press, Cambridge, MA, 2002.

OIF Forum Web site: http://www.oiforum.com/.

Piguet, C., Ed., *Low-Power Electronics Design*, CRC Press, Boca Raton, FL, 2005.

Rabaey, J. M., A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, Upper Saddle River, NJ, 2003.

Sharma, A., *Semiconductor Memories: Technology, Testing and Reliability*, IEEE Press, Piscataway, NJ, 1997.