7

An Overview of SoC Buses

M. Mitić
M. Stojčev
University of Niš

Z. Stamenković
IHP GmbH—Innovations for High Performance Microelectronics

7.1 Introduction

The electronics industry has entered the era of multimillion-gate chips, and there is no turning back. This technology promises new levels of integration on a single chip, called the system-on-a-chip (SoC) design, but also presents significant challenges to the chip designers. Processing cores on a single chip may number well into the high tens within the next decade, given the current rate of advancements [1]. Interconnection networks in such an environment are, therefore, becoming more and more important [2]. Currently, on-chip interconnection networks are mostly implemented using buses. For SoC applications, design reuse becomes easier if standard internal connection buses are used for interconnecting components of the design. Design teams developing modules intended for future reuse can design interfaces for the standard bus around their particular modules. This allows future designers to slot the reuse module into their new design simply, which is also based around the same standard bus [3].

Shrinking process technologies and increasing design sizes have led to highly complex billion-transistor integrated circuits (ICs). As a consequence, manufacturers are integrating increasing numbers of components on a chip. A heterogeneous SoC might include one or more programmable components such as general-purpose processors cores, digital signal processor cores, or application-specific intellectual property (IP) cores, as well as an analog front end, on-chip memory, I/O devices, and other application-specific circuits. In other words, a SoC is an IC that implements most or all the functions of a complete electronic system [4].

On-chip bus organized communication architecture (CA) is among the top challenges in CMOS SoC technology due to rapidly increasing operation frequencies and growing chip size. In general, the performance of the SoC design heavily depends upon the efficiency of its bus structure. The balance of computation and communication in any application or task is, of course, known as a fundamental determinant of

delivered performance. Usually, IP cores, as constituents of SoCs, are designed with many different interfaces and communication protocols. Integrating such cores in a SoC often requires insertion of suboptimal glue logic. Standards of on-chip bus structures were developed to avoid this problem. Currently, there are a few publicly available bus architectures from leading manufacturers, such as CoreConnect from IBM [5], AMBA from ARM [6], SiliconBackplane from Sonics [7], and others. These bus architectures are usually tied to processor architecture, such as the PowerPC or the ARM processor. Manufacturers provide cores optimized to work with these bus architectures, thus requiring minimal extra interface logic.

This chapter gives an overview of the more popular on-chip standardized bus architectures such as AMBA, CoreConnect, Wishbone, STBus, and others, both from an industrial and research viewpoint. The crucial features, including bus topologies, arbitration methods, bus-widths, and types of data transfers are considered.

The rest of this chapter is organized as follows: Section 7.2 presents background material on CAs, including a survey of typical topologies and communication protocols in use today. Section 7.3, as a central part of this chapter, gives an overview of several popular SoC CAs.

7.2 On-Chip Communication Architectures

7.2.1 Background

The design of on-chip CAs addresses the following three issues [8]:

Definition of CA topology: This defines the physical structure of the CA. Numerous topologies exist, ranging from single shared bus to more complex architectures such as bus hierarchies, token ring, crossbar, or custom networks.

Selection and configuration of the communication protocols: For each channel/bus in the CA, communication protocols specify the exact manner in which communication transaction occurs. These protocols include arbitration mechanisms (e.g., round robin access, priority-based selection [5,6], time division multiplexed access (TDMA) [7], which are implemented in centralized or distributed bus arbiters.

Communication mapping: This refers to the process of associating abstract system-level communications with physical communication paths in the CA topology [8].

7.2.2 Topologies

With respect to topology, on-chip communication architectures can be classified as follows:

Shared bus: The system bus is the simplest example of a shared communication architecture topology and is commonly found in many commercial SoCs [9]. Several masters and slaves can be connected to a shared bus. A block bus arbiter periodically examines accumulated requests from the multiple master interfaces and grants access to a master using arbitration mechanisms specified by the bus protocol. Increased load on a global bus lines limits the bus bandwidth. The advantages of shared-bus architecture include simple topology, extensibility, low area cost, efficient to implement, and easy to build. The disadvantages of shared-bus architecture are larger load per data bus line, longer delay for data transfer, larger energy consumption, and lower bandwidth. Fortunately, the above disadvantages, with the exception of the lower bandwidth, may be overcome by using a low-voltage swing signaling technique.

Hierarchical bus: This architecture consists of several shared busses interconnected by bridges to form a hierarchy. SoC components are placed at the appropriate level in the hierarchy according to the performance level they require. Low-performance SoC components are placed at lower performance buses, which are bridged to the higher performance buses so as not to burden the higher performance SoC components. Commercial examples of such architectures include the AMBA bus [6], CoreConnect [5], etc. Transactions across the bridge involve additional overhead, and during the transfer both buses remain inaccessible to other SoC components. Hierarchical buses offer large throughput improvements over the shared busses due to (1) decreased load per bus, (2) the potential for transactions to proceed in parallel on different buses, and multiple ward communications can be preceded across the bridge in a pipelined manner [8].

Ring: In numerous applications, ring-based applications, such as network processors, and ATM switches are widely used [5,8]. In a ring, each node component (master/slave) communicates using a ring interface, and is usually implemented by a token-pass protocol.

7.2.3 On-Chip Communication Protocols

Communication protocols deal with different types of resource management algorithms used for determining access right to shared communication channels. From this point of view, in the rest of this section, we give a brief comment related to the main feature of the existing communication protocols.

Static-priority: This protocol employs an arbitration technique and is used in shared-bus communication architectures. A centralized arbiter examines accumulated requests from each master and grants access to the requesting master that is of the highest priority. Transactions may be of non-preemptive or preemptive type. AMBA, CoreConnect, etc., use this protocol [5,6].

Time division multiple access: The arbitration mechanism is based on a timing wheel with each slot statically reserved for a unique master. Special techniques are used to alleviate the problem of wasted slots. Sonics uses this protocol [7].

Lottery: A centralized lottery manager accumulates request for ownership of shared communication resources from one ore more masters, each of which has, statically or dynamically, assigned a number of lottery tickets [10].

Token passing: This protocol is used in ring-based architectures. A special data word, called token, circulates on the ring. An interface that receives a token is allowed to initiate a transaction. When the transaction completes, the interface releases the token and sends it to the neighboring interface.

Code division multiple access (CDMA): This protocol has been proposed for sharing on-chip communication channel. In a sharing medium, it provides better resilience to noise/interference and has an ability to support simultaneously transfer of data streams. But this protocol requires implementation of complex special direct sequence spread spectrum coding schemes, and energy/battery inefficient systems such as pseudorandom code generators, modulation and demodulation circuits at the component bus interfaces, and differential signaling [11].

AQ1

7.2.4 Other Interconnect Issues

We now point to several interconnect issues that have direct impact on bus organization and its efficiency:

Programming model: This consists of a load and store operations. These operations are implemented as a sequence of primitive bus transactions. Modules issuing requests are called masters and those serving requests are called slaves [12].

Split versus non-split buses: If there is a single arbitration for a request–response pair, the bus is called non-split. In this case, the bus remains allocated to the master of the transaction until the response is delivered. Alternatively, in a split bus, the bus is released after the request to allow transactions from different masters to be initiated [13].

Transaction ordering: Usually, all transactions on a bus are ordered. However, on a split bus, a total ordering of transactions on a single master may cause performance degradation. This situation is typical when slaves respond at different speed. To solve this problem, recent extensions to bus protocols allow transactions to be performed on connections [14,15].

Atomic chains of transactions: This represents a sequence of transactions initiated by a single master that is executed on a single slave exclusively. During this activity, other masters are denied access to that slave until the end of the first transaction. This mechanism is standardly used to implement synchronization mechanisms between master modules (i.e., semaphores) [13].

Media arbitration: Bus master modules access the bus and the arbiter grants access. Arbitration is centralized as there is only one arbiter component. It is also global, since all requests, as well as the state of the bus, are visible to the arbiter. When a grant is given, the complete path from the source to the destination is exclusively reserved [12,13].

Destination name and routing: Command address and data are broadcasted on the bus. They reach every destination; only one of each activates, based on the broadcasted address, and executes the requested command [12,13].

Latency: This is caused by the following two factors: (1) the access time to the bus, which is the time until the bus is granted; and (2) the latency introduced by the bus to transfer the data [12].

Data format: This is defined by separate wire groups for the transaction type, address, write data, read data, and return acknowledgments/errors [5,6,16,17].

7.2.5 Advantages and Disadvantages of On-Chip Buses

In the bus-based design approach, IP components communicate through one or more buses usually interconnected by bus bridges. Since the bus specification can be standardized, libraries of components whose interfaces directly match this specification can be developed. Even if components follow the bus standard, very simple bus interface adapters may still be needed. For components that do not directly match the specification, wrappers have to be built. Companies offer very rich component libraries and specialized development and simulation environments for designing systems around their buses. A somewhat different approach is a core-based design. In this case, IP components are compliant to a bus-independent and standardized interface and, thus, are directly connected to each other. Although the standard may support a wide range of functionalities, each component may have an interface containing only the functions that are relevant to it. These components may also be interconnected through a bus, in which case standard wrappers can adapt the component interface to the bus.

As a conclusion we can say that on-chip-bus-design and on-chip-core-based design methodologies are integration approaches that depend on standardized components or bus interfaces. They allow the integration of homogeneous IP components that follow these standards to be directly connected to each other, without requiring the development of complex wrappers. Let us note that on-chip buses rely on shared communication resources and on arbitration mechanism that is in charge of serializing bus access requests. This widely adopted solution unfortunately suffers from power and performance scalability limitations, and restricted sharing of resources between communicating entities. For bus networks, the bus is occupied by a single communication even if multiple communications could operate simultaneously on different portions on the bus. Therefore, a lot of effort has been devoted to the development of advanced bus topologies (e.g., partial or full crossbar, bridged buses) and protocols for better support of route-ability, flexibility, reliability, and reconfigure-ability. Therefore, a systematic way of designing networks with possibly arbitrary topology is gaining the importance [2].

In the long run, a more aggressive approach is needed. For particular needs, the SoC may be built around a sophisticated and dedicated network-on-chip that may deliver very high performance for connecting a large number of components. It seems that this design paradigm shifts toward a packetized on-chip communication based on micro-networks of interconnects or networks-on-chip [18]. More details concerning NoC design are given in Refs. [13,19,20].

7.3 System-On-Chip Buses

In the sequel, an overview of the more relevant SoC communication architectures will be given. Because of the space limitation, the discussion will be focused on describing the more distinctive features of each of them.

7.3.1 AMBA Bus

AMBA (advanced microcontroller bus architecture) [6,21] is a bus standard devised by ARM with aim to support efficient on-chip communications among ARM processor cores. Nowadays, AMBA is one of the leading on-chip busing systems used in high-performance SoC design. AMBA (see Figure 7.1) is hierarchically organized into two bus segments, system- and peripheral bus, mutually connected via bridge that

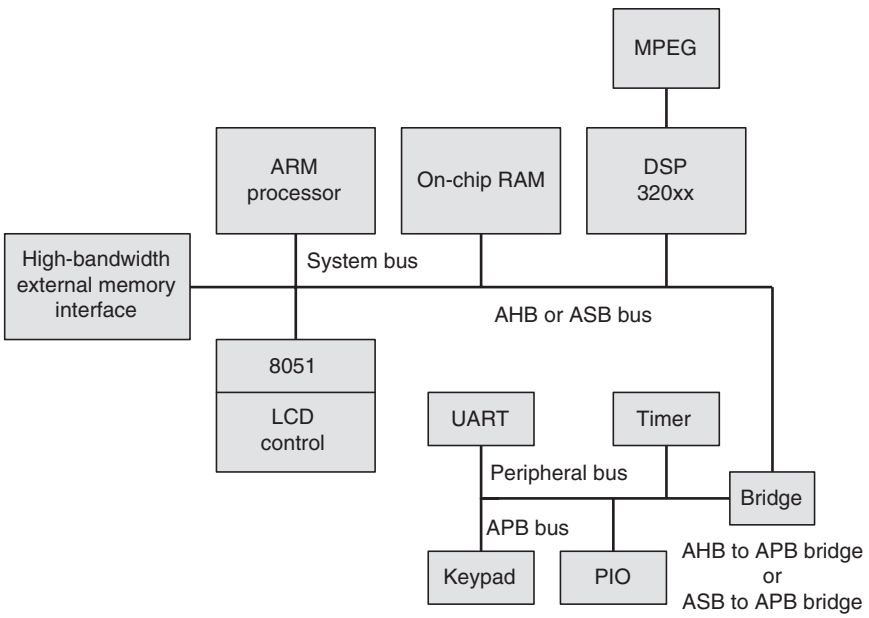

FIGURE 7.1 AMBA-based system architecture.

buffers data and operations between them. Standard bus protocols for connecting on-chip components generalized for different SoC structures, independent of the processor type, are defined by AMBA specifications. AMBA does not define method of arbitration. Instead it allows the arbiter to be designed to suit the applications needs, the best. The following are the three distinct buses specified within the AMBA bus:

1. ASB (advanced system bus): First generation of AMBA system bus used for simple cost-effective designs that support burst transfer, pipelined transfer operation, and multiple bus masters.
2. AHB (advanced high-performance bus): As a later generation of AMBA, this bus is intended for high-performance, high-clock synthesizable designs. It provides high-bandwidth communication channel between embedded processor (ARM, MIPS, AVR, DSP 320xx, 8051, etc.) and high-performance peripherals/hardware accelerators (ASICs MPEG, color LCD, etc), on-chip SRAM, on-chip external memory interface, and APB bridge. AHB supports a multiple bus masters operation, peripheral and a burst transfer, split transactions, wide data bus configurations, and non-tristate implementations. Constituents of AHB are AHB-master, -slave, -arbiter, and -decoder.
3. APB (advanced peripheral bus): This bus is used to connect general-purpose low-speed, low-power peripheral devices. The bridge is peripheral bus master, whereas all buses devices (Timer, UART, PIA, etc) are slaves. APB is static bus that provides a simple addressing with latched addresses and control signals for easy interfacing.

Recently, two new specifications for AMBA bus—multilayer AHB and AMBA AXI— are defined [6,22]. Multilayer AHB provides more flexible interconnect architecture (matrix that enables parallel access paths between multiple masters and slaves) with respect to AMBA AHB, and keeps the AHB protocol unchanged. AMBA AXI is based on the concept point-to-point connection. Good overview papers related to AMBA specifications are Refs. [6,22,23].

7.3.2 Avalon

Avalon bus (see Figure 7.2) is a bus architecture designed for connecting on-chip processors and peripherals together into a system-on-a-programmable-chip (SOPC). As an Altera's parameterized bus, Avalon is mainly used for FPGA SoC design based on Nios processor [24,25].

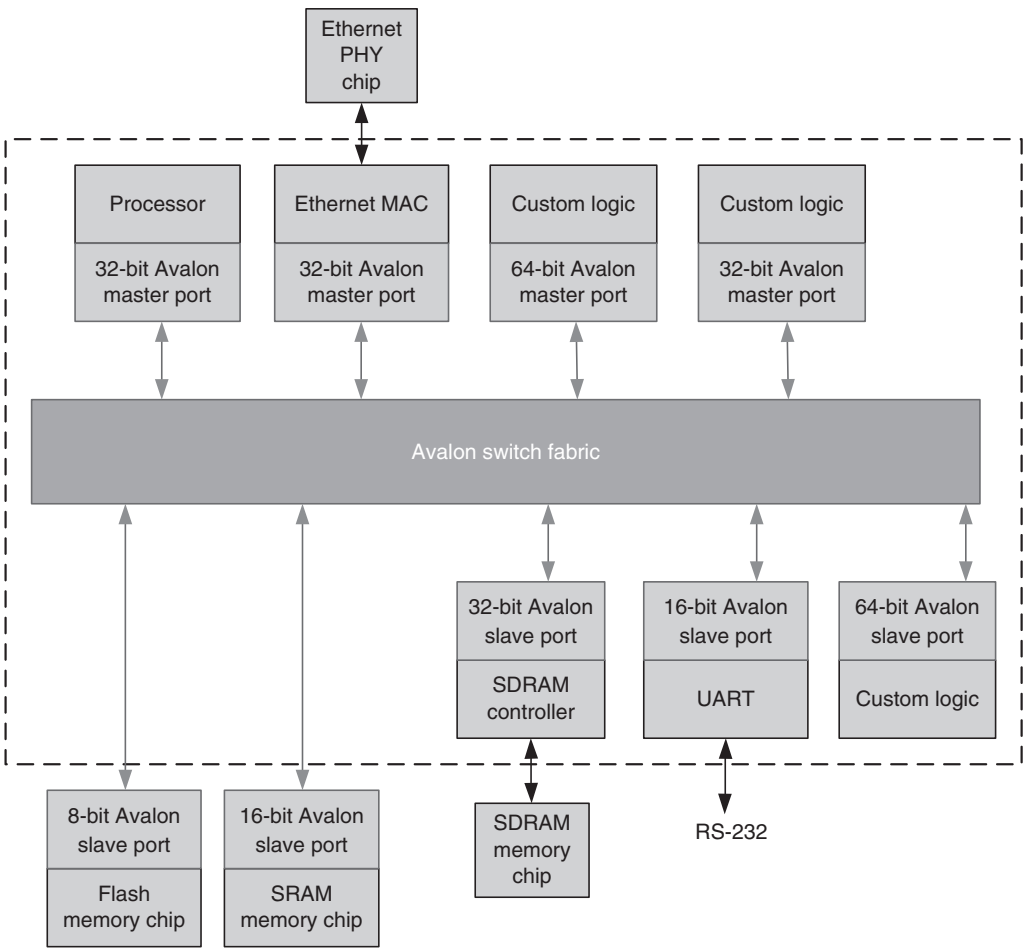

FIGURE 7.2 Avalon bus-based system.

Avalon has a set of predefined signal types with which a user can connect IP blocks. Avalon is a synchronous interface and specifies the port connections between master and slave components and specifies the timing by which these components communicate. Basic Avalon bus transactions transfer one data item 8-, 16-, 32-, 64-, or 128-bits wide. Avalon uses separate address, data, and control lines.

This bus supports multiple bus masters. Masters and slaves interact with each other based on a technique called slave-side (distributed) arbitration.

The Avalon bus model (switch fabric) provides the following services to Avalon peripherals connected to the bus: data-path multiplexing, address decoding, wait-state generation, dynamic bus sizing, interrupt priority assignment, latent transfer capabilities, and a streaming read and write capabilities [24,25].

Altera's SOPC Builder, as a system development tool, automatically generates the switch fabric logic that supports each type of transfer supported by the Avalon interface.

7.3.3 CoreConnect

CoreConnect [5] is an IBM-developed on-chip bus. By reusing processor, subsystem, and peripheral cores, supplied from different sources, enables their integration into a single VLSI design. CoreConnect is a hierarchically organized architecture. It is comprised of three buses that provide an efficient interconnection of cores, library macros, and custom logic within a SoC (see Figure 7.3).

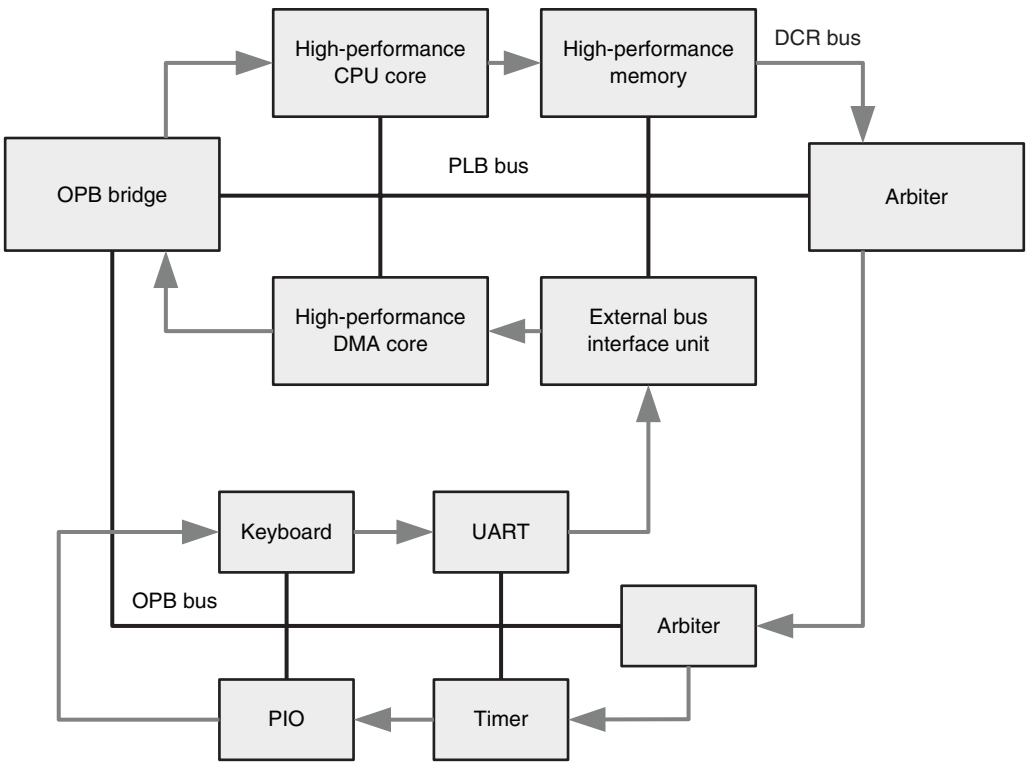

FIGURE 7.3 CoreConnect bus-based system.

Processor local bus (PLB): This is the main system bus. It is synchronous, multi-master, central arbitrated bus that allows achieving high-performance and low-latency on-chip communication. Separate address and data buses support concurrent read and write transfers. PLB macro, such as glue logic, is used to interconnect various master and slave macros. Each PLB master is attached to the PLB through separate addresses, read-data and write-data buses, and other control signals. PLB slaves are attached to PLB through shared, but decoupled, address, and read-data and write-data buses. Up to 16 masters can be supported by the arbitration unit, whereas there are no restrictions in the number of slave devices [21].

On-chip peripheral bus (OPB): This bus is optimized to connect lower speed, low throughput peripherals, such as serial and parallel port, UART, etc. Crucial features of OPB are fully synchronous operation, dynamic bus sizing, separate address and data buses, multiple OPB bus masters, single cycle transfer of data between bus masters, single cycle transfer of data between OPB bus master and OPB slaves, etc. OPB is implemented as multi-master, arbitrated buses. Instead of tristate drivers, OPB uses distributed multiplexer. PLB masters gain access to the peripherals on the OPB bus through the OPB bridge macro. The OPB bridge acts as a slave device on the PLB and a master on the OPB.

Device control register bus (DCRB): This is a single-master bus mainly used as an alternative relatively low speed data path to the system for (1) passing status and setting configuration information into the individual device-control registers between the processor core and other SoC constituents such as auxiliary processors, on-chip memory, system cores, peripheral cores, etc; and (2) design for testability purposes. DCR is synchronous bus based on a ring topology implemented as distributed multiplexer across the chip. It consists of a 10-bit address bus and a 32-bit data bus. CoreConnect implements arbitration based on a static priority, with programmable priority fairness.

7.3.4 STBus

STBus is an on-chip bus protocol developed by STMicroelectronics [16]. It represents a set of protocol, interfaces, and architectural specifications intended to implement the communication network of digital systems. The STBus interfaces and protocols are closely related to the virtual component interface (VCI) industry standard. STBus implements both the protocols definition and the bus components. The following protocols are used [16,26]:

1. Type I (peripheral protocol): It is a simple synchronous handshake protocol with limited set of available command types, suitable for register access and slow peripherals. No pipelining is applied. Type I acts as a request-grant protocol. Only limited operation code and length are supported.
2. Type II (basic protocol): This protocol is more efficient than type I as it supports split transactions and adds pipelining features. The transaction set include read/write operation with different sizes (up to 64 bytes) and also specific operations like read-modify-write and swap. Type II is equivalent to the request-grant-valid protocol. Transactions may also be grouped into chunks to ensure allocation of the slave and to ensure no interruption of the data stream. This protocol is typically suited for external memory controllers. A limitation of this protocol is that the traffic and symmetric transactions must be ordered (i.e., the number of the requesting cells equals to the number of the response ones).
3. Type III (advanced protocol): This is the most efficient protocol, as it adds support for split transactions, out-of-order executions, and asymmetric communications (i.e., the number of cells might differ between request and response). Type III is mainly used by CPUs, multichannel DMAs, and DDR controllers.

The STBus is modular and allows master and slaves of any protocol type and data size to communicate, through the use of appropriate type/size converters. A wide variety of arbitration policies is also available, such as bandwidth limitation, latency arbitration, least recently used (LRU), priority-based arbitration, etc.

The components interconnected by the STBus can be either initiators (initiates transactions on the bus by sending requests, such as CPUs or ASICs) or targets (responds to requests, such as memories, registers, or dedicated peripherals). Initiators can load or store data through the STBus backbone (see Figure 7.4). Some resources might be both initiators and peripheral/targets.

STBus-based system includes three kinds of components [26]:

Switch or node: this block arbitrates and routes the requests and responses. Different kinds of arbitration are possible, including fixed priorities, variable priorities, dynamic priorities, latency based, bandwidth based, and LRU.

Converter or bridge domain: This converts the request from the protocol to another, for example from basic protocol to advanced protocol.

Size converter: It is used between two buses of same type of different widths; it includes buffering capacity.

STBus can instantiate different bus topologies as follows [21]:

Single shared bus: This is suitable for simple low-performance implementations; this bus characterizes minimal wiring area but limited scalability.

Full crossbar: This bus topology is intended for high-performance systems; wiring area is large.

Partial crossbar: This is used in medium-performance systems, represents a good compromise with respect to the previous two proposals.

7.3.5 Wishbone

Wishbone [27] bus architecture was developed by Silicore Corporation. In August 2002, OpenCores (organization that promotes open IP cores development) put it into the public domain. This means that Wishbone is not copyrighted and can be freely copied and distributed.

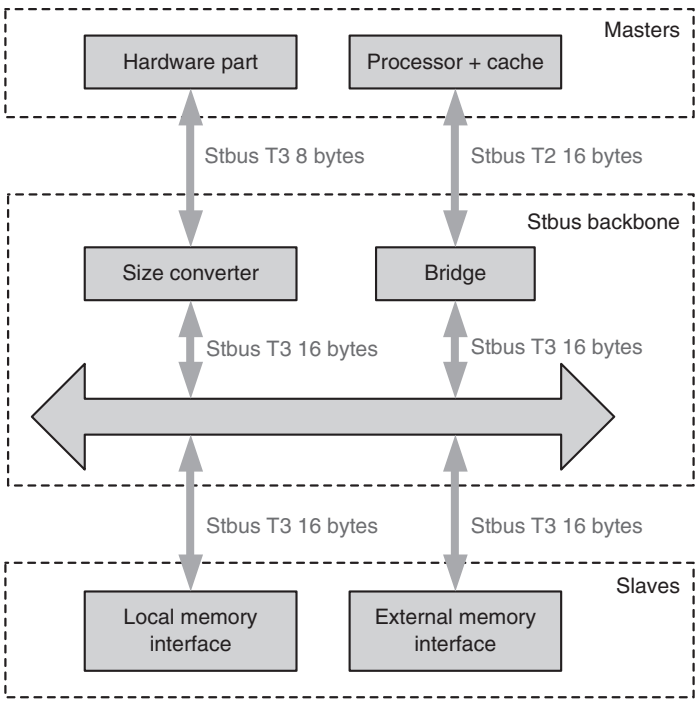

FIGURE 7.4 STBus interconnect.

The Wishbone defines two types of interfaces, called master and slave. Master interfaces are IPs capable of initiating bus cycles, whereas slave interfaces are capable of accepting bus cycles [21]. The hardware implementations support various types of interconnection topologies (see Figure 7.5) such as

1. Point-to-point connection—used for direct connection of two participants that transfer data according to some handshake protocol
2. Dataflow interconnection—used in linear systolic array architectures for implementation of DSP algorithms
3. Shared bus—typical for MPSoCs organized around single system bus
4. Crossbar switch interconnection—usually used in MPSoCs when more than one masters can simultaneously access several different slaves. The master requests a channel on the switch; once this is established, data is transferred in a point-to-point manner.

The Wishbone supports different types of bus transactions, such as read/write, implementing blocking/unblocking access. A read-modify-write transfer is also supported. Wishbone does not define hierarchical buses. In applications where two buses should exist, one slow and one fast, two separated Wishbone interfaces could be created. Designer can also choose the arbitration mechanism and implements it to fit the application needs best.

7.3.6 CoreFrame

The CoreFrame [28] architecture is low power high-performance on-chip interconnect architecture for integration of SoC blocks. From a high-level point of view, the CoreFrame architecture (see Figure 7.6) is viewed as a system of three buses (CPU bus, PalmBus, and MBus). The CPU bus is connected to PalmBus via PalmBus controller and to the MBus through a cache or bridge. The PalmBus and MBus are independent parallel buses, rather than a hierarchy of buses. Concurrent activities may be achieved on both buses maximizing available bandwidth resources. To avoid three-state buffering, CoreFrame does

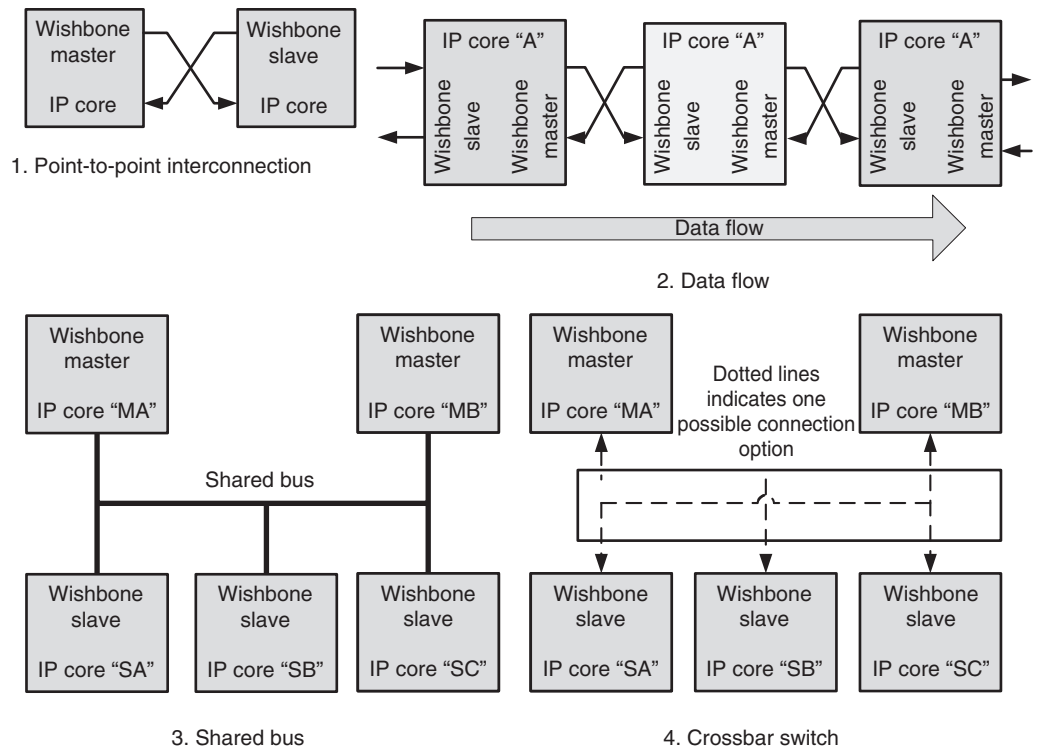

FIGURE 7.5 Possible Wishbone interconnections.

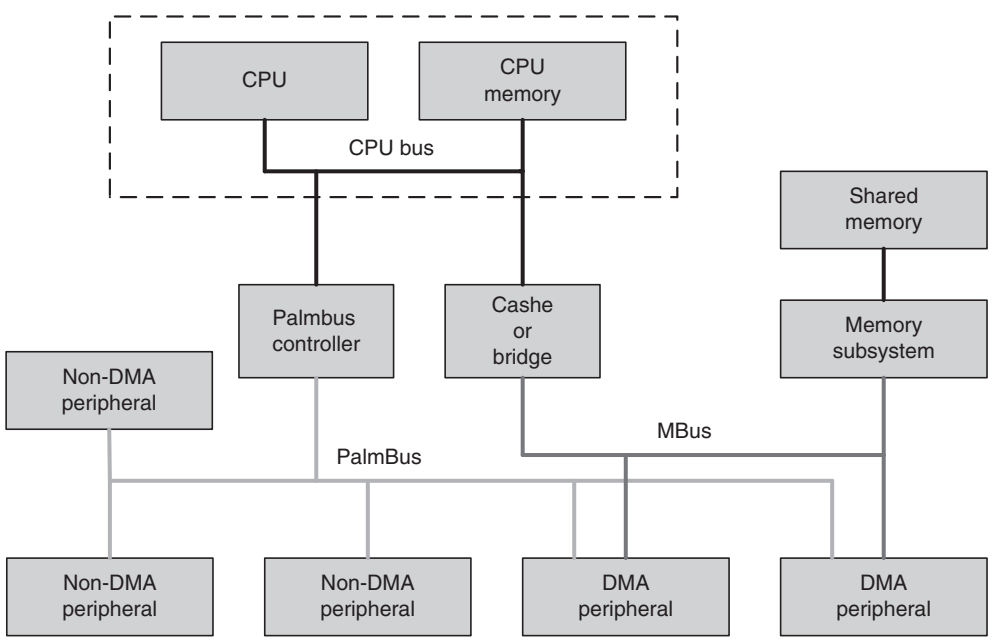

FIGURE 7.6 CoreFrame architecture-based system structure.

not use shared signal lines. Instead it uses point-to-point signals and multiplexing. Communication between subsystems is carried out through shared memory variables.

A PalmBus represents a master–slave interface with a single-master intended for communications between the CPU and peripheral blocks. It is not used to access memories. The PalmBus is designed for low-speed access from the CPU core and it provides the I/O backplane and allows the processor to configure and control peripheral blocks. Timings of the bus are synchronous with the CPU core. PalmBus is also designed with low-power consumption in mind [29].

The MBus is designed for high-speed accesses to shared memory from the CPU core and peripheral blocks. The MBus protocol is optimized for both ASIC-type implementations and data transfers to and from the data memory devices [30].

7.3.7 Manchester Asynchronous Bus for Low Energy

Manchester asynchronous bus for low energy (MARBLE) developed at the Manchester University is on-chip two channel micropipeline bus with centralized arbitration and address decoding that operates without global clock pulse. It is intended to provide interconnections of asynchronous macrocells within the VLSI ICs [31].

MARBLE is based on a split-transfer architecture allowing transfers between different initiators and targets to be interleaved without the needs for retries, thus giving low-energy operation and low latency. A MARBLE bus consists of two asynchronous multipoint channels. One of these channels carries the command from the initiator to the target returning either the accept status or defer status. The other multipoint channel carries a response from the target to the initiator (and the read data or write data in the appropriate direction). The two channels are used in a decoupled transfer scheme with loose coupling between channels to implement split transactions [32].

The interconnection provided by MARBLE is used in AMULET3H microprocessor (see Figure 7.7). It is intended to connect CPU core and DMA controller to RAM, ROM, and other peripherals [32]. In general, MARBLE demonstrates that all the features of a high-speed on-chip macrocell bus can be implemented efficiently in a fully asynchronous design style.

7.3.8 PI Bus

The PI (peripheral interconnect) bus was developed by several European semiconductor companies (Advanced RISC Machines, Philips Semiconductors, SGS-Thomson Microelectronics, Siemens, and TEMIC/MATRA MHS) within a framework of European project OMI (open microprocessor initiative framework). PI bus is an open standard published by OMI. For the purpose of SoC design, PI bus System Toolkit is developed. VHDL codes for master, slave, and control units are freely distributed. In addition, synthesis scripts for different ASIC and FPGA technologies, and examples of system solutions are available [9].

PI bus is a synchronous bus with un-multiplexed address and data signals that supports operation of multiple masters and bridges. It is an on-chip bus used in modular, highly integrated SoC designs. PI bus is designed for memory-mapped data transfers between its bus agents. Bus agents are on-chip modules equipped with PI bus interface and connected via PI bus signals. A PI bus agent acts as a PI bus master when it initiates data read/write operations, because the bus ownership has been granted to the agent. A PI bus agent who is addressed at PI bus operation acts as a PI bus slave when it performs the requested data read/write operation. Typical masters are processor modules, coprocessors, or DMA modules, while typical slaves are on-chip memory and input–output interfaces to the external world (see Figure 7.8) [9].

The main features of PI bus are the following: (1) processor independent implementation and design, (2) demultiplexed operation, (3) clock synchronous, (4) peak transfer rate of 200 MHz (50 MHz bus clock), (5) address and data bus scalable (up to 32 bits), (6) 8-, 16-, 32-bit data access, (7) broad range of transfer types from single to multiple data transfers, and (8) multi-master capability. The PI bus does not provide (1) cache coherency support, (2) broadcast, (3) dynamic bus sizing, and (4) unaligned data access [9].

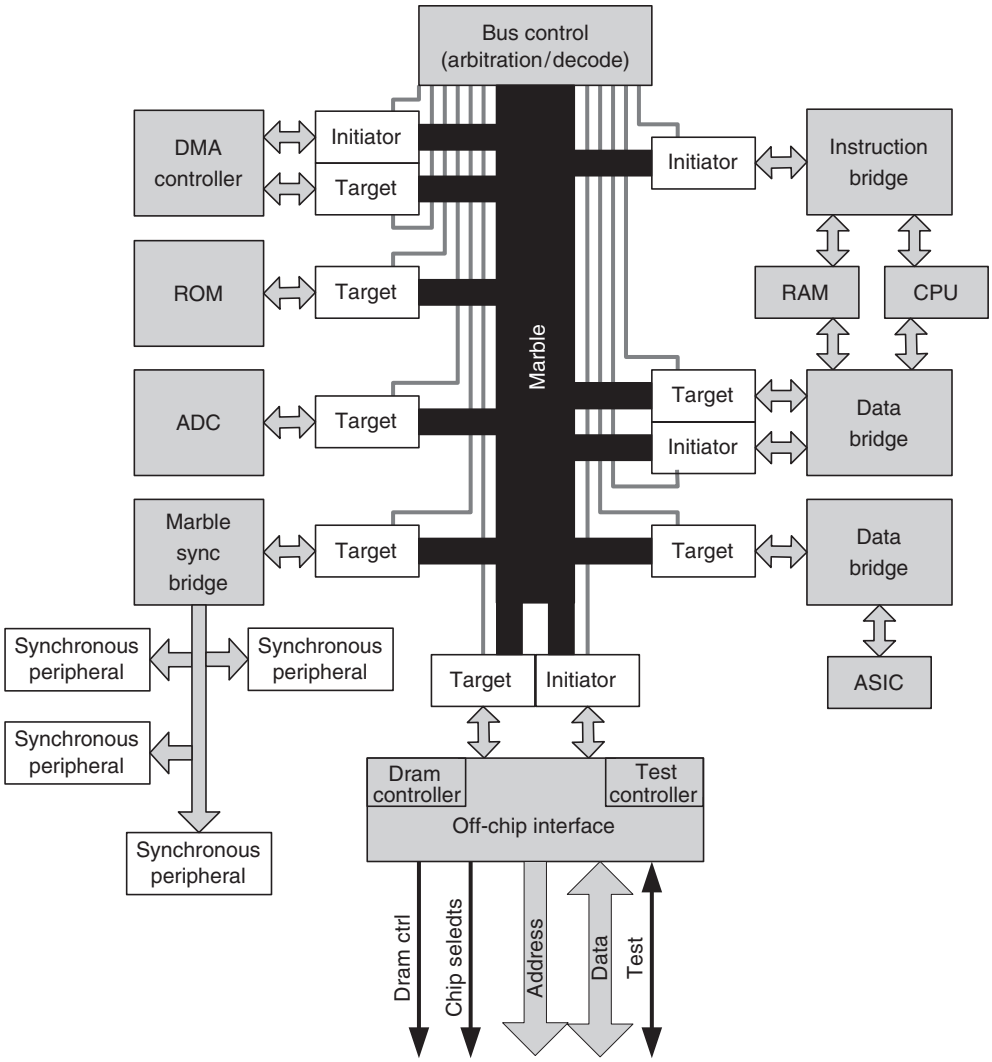

FIGURE 7.7 AMULETH3H system.

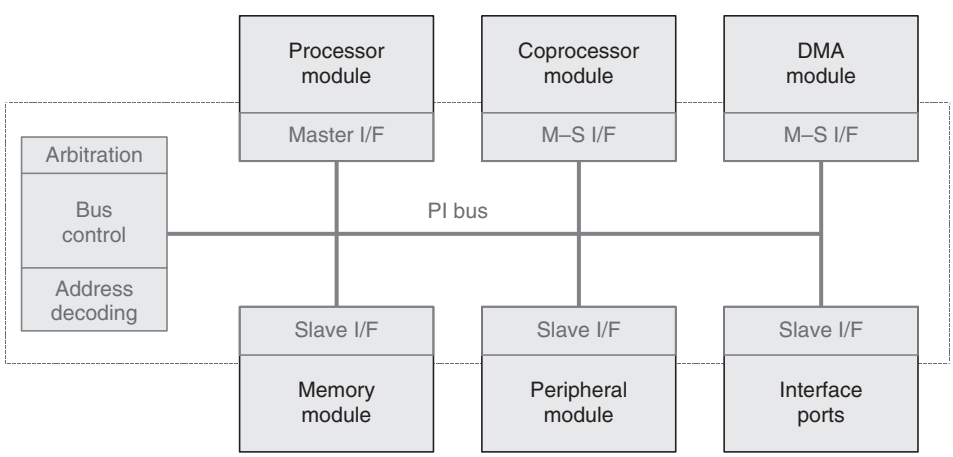

FIGURE 7.8 Modules of a PI bus connected system.

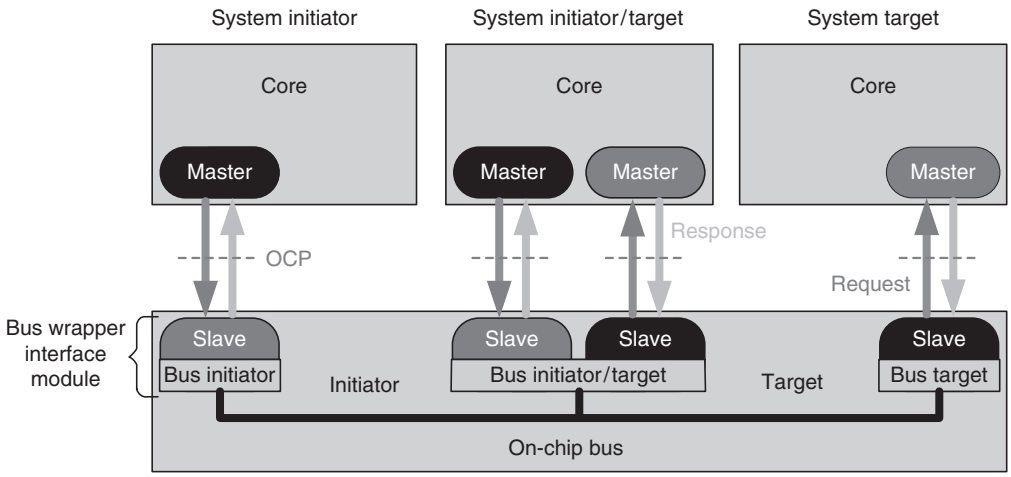

FIGURE 7.9 Wrapped bus and OCP instances.

7.3.9 Open Core Protocol

Open core protocol (OCP) [14] is an interface standard that interconnects IP cores to on-chip bus. The OCP defines a comprehensive, bus-independent, high-performance, and configurable interface between IP cores and on-chip communication subsystems. A designer selects only those signals and features from the palette of OCP configurations needed to fulfill all of IP core's unique data, control, and test signaling requirements. Existing IP cores may be inexpensively adapted. Defining a core interface using the OCP provides a complete description for system integration. The following are the main features of OCP interface: (1) master–slave interface with unidirectional signals; (2) driven and sampled by the rising edge of the OCP clock; (3) fully synchronous, no multi-cycle timing paths; (4) all signals are strictly point-to-point (except clock and reset); (5) simple request/acknowledge protocol; (6) supports data transfer on every clock cycle; (7) allows master or slave to control transfer rate; (8) configurable data word width; (9) configurable address width; (10) pipelined or blocking reads; and (11) specific description formats for core characteristics, interfaces (signals, timing, and configuration), and performance [15].

Some of the standard on-chip buses, such as AMBA and SiliconBackplane μNetwork, use OCP. Communication requirements concerning IP core can be described using this protocol format. OCP interface is user settable, so the designer can define interface attribute, such as address and data bus width. Beside basic OCP version, there are four extensions: simple extension, complex extension, sideband extension and debug and test interface extension. Basic OCP includes only data flow signals and is based on simple request and acknowledge protocol. However, the optional extensions support more functionality in control, verification, and testing. Simple extension and complex extension support burst transaction and pipelined write operations. In addition, sideband extension supports user-defined signals and asynchronous reset. Also, debug and test interface extension supports JTAG (Joint Test Action Group) and clock control. This is the reason why, when integrated in SoC, the OCP allows debugging and IP block test generating. Figure 7.9 presents SoC design based on the OCP.

7.3.10 Virtual Component Interface

The VCI [17] is an interface rather than a bus. Thus, the VCI specifies (1) a request–response protocol, (2) a protocol for the transfer of requests and responses, and (3) the contents and coding of these requests and responses. The VCI does not touch areas such as bus allocation schemes, competing for a bus, and so forth.

There are three complexity levels for the VCI: peripheral VCI (PVCI), basic VCI (BVCI), and advanced VCI (AVCI). The PVCI provides a simple, easily implementable interface for applications that do not

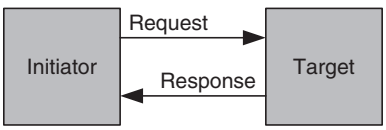

FIGURE 7.10 VCI as a point-to-point connection.

need all the features of the BVCI. The BVCI defines an interface that is suitable for most applications. It has a powerful, but not overly complex protocol. The AVCI adds more sophisticated features, such as threads, to support high-performance applications. The PVCI is a subset of the BVCI, and the AVCI is a superset of the BVCI.

BVCI and AVCI make use of a "split protocol." That is, the timing of the request and the response are fully separate. The initiator can issue as many requests as needed, without waiting for the response. The protocol does not prescribe any connection between issuing of requests and arrival of the corresponding responses. The only thing specified is that the order of responses corresponds to the order of requests. In the AVCI, requests may be tagged with identifiers, which allow such requests and request threads to be interleaved and they response to arrive in a different order. Responses bear the same tags issued with the corresponding requests, such that the relation can be restored upon the reception of a response. As an interface, the VCI can be used as a point-to-point connection between two units called the initiator and the target, where the initiator issues a request and the target responds (see Figure 7.10).

The VCI can be used as the interface to a wrapper, which means a connection to a bus. This is how the VCI allows the VC to be connected to any bus. An initiator is connected to that bus by using a bus initiator wrapper. A target is connected to that bus by using a bus target wrapper. Once the wrappers for the bus have been designed, any IPs can be connected to that bus, as depicted in Figure 7.11.

7.3.11 SiliconBackplane μNetwork

Sonics μNetwork [7] consists of a set of architectures and SoC design tools. Defined architectures are SiliconBackplane for on-chip interconnection and multichip for off-chip interconnection. SiliconBackplane implements two-level arbitration, based on TDMA and round-robin.

SiliconBackplane μNetwork is a network on a chip that connects IP blocks in a SoC. μNetwork isolates the system of IP blocks from network by requiring all blocks to use single bus interface protocol, OCP. Every IP block communicates via "wrapper," which μNetwork calls an agent, using OCP. Agents communicate with each other through μNetwork. As system's requirements change, OCP and μNetwork support modification of many system's parameter in real time. System requirements relate to, for example, selection of arbitration scheme, definition of address space, etc. An agent is generated using the tool Fast Forward Development Environment, developed by Sonics. Basic building blocks of SiliconBackplane μNetwork are given in Figure 7.12.

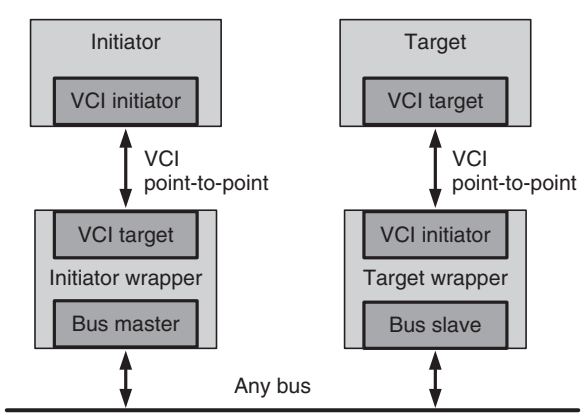

FIGURE 7.11 Two VCI connections used to realize a bus connection.

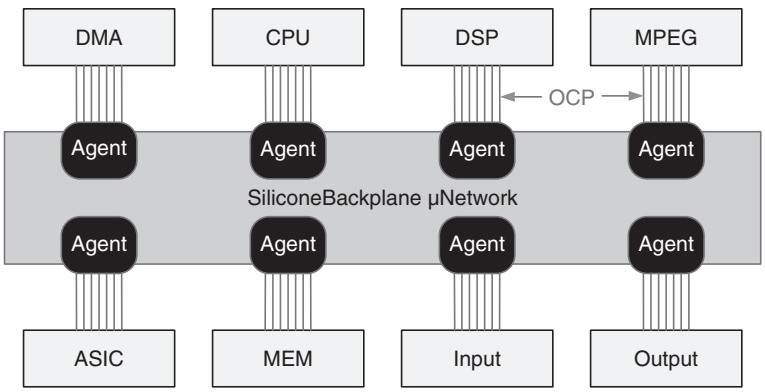

FIGURE 7.12 SiliconBackplane μNetwork constituents.

7.4 Summary

Complex VLSI IC design has been revolutionized by the widespread adoption of the SoC paradigm. The benefits of the SoC approaches are numerous, including improvements in system performance, cost, size, power dissipation, and design turnaround time. Many SoC designs consist of one or more IPs, designed for a single or narrow set of applications with a highly characterizable communication. As the level of a chip integration continues to advances at a fast pace, the desire for efficient interconnects rapidly increases. Currently on-chip interconnections networks are mostly implemented using traditional interconnects like buses. The wide variety of buses used in SoC designs presents the major problem for reusable design. A number of companies and standards committees have attempted to standardize buses and interface with mixed results.

In this chapter we have given an overview of the most popular on-chip bus-based interconnection networks such as AMBA, Avalon, CoreConnect, STBus, Wishbone, etc. The main characteristics of the considered buses with respect to topology, arbitration method, bus-width, and types of data transfers are discussed. In addition, we have pointed to some of the issues that SoC designers are facing in determining the bus architecture to use to provide flexible and high bandwidth between IP cores.

References

AQ2

1. Keating M. and Bricaud P., *Reuse Methodology Manual for System-on-a-Chip Designs,* 2/E/ Kluwer Academic Publishers, Boston, MA, 1999.
2. Ho W.H. and Pinkston T.M., A design methodology for efficient application-specific on-chip interconnects, *IEEE Trans. Parallel Distributed Syst.*, 17(2): 174–190, February 2006.
3. Horspool N. and Gorman P., *The ASIC Handbook*, Prentice Hall, PTR, Upper Saddle River, NJ, 2001.
4. Bernini L. and De Micheli G., Networks on chips: A new paradigm for component-based mpsoc design, chapter 3, pp. 49–80, in Jerraya A.A. and Wolf W., (Eds.), *Miltiprocessor Systems-on-Chips*, Elsevier, Amsterdam, 2005.
5. CoreConnect Bus Architecture, IBM Microelectronics, available at http://www.ibm.com/chips/-products/coreconnect, January 2006.
6. ARM.AMBA Specifications v2.0, 1999, available at http://www.arm.com, January 2006.
7. Sonics μNetwork Technical overview, January 2002, available at http://www.sonicsinc.com, January 2006.
8. Lahiri K., Dey S., and Raghunathan A., Design of communication architectures for high-performance and energy-efficient systems-on-chip, chapter 7, pp. 187–222, in Jerraya A.A. and Wolf W., (Eds.), *Multiprocessor Systems-on-Chips*, Elsevier, Amsterdam, 2005.

9. Draft Standard OMI 324: PI-Bus, Rev. 0.3d, Open Microprocessor Systems Initiative, Copyright 1994 by Siemens AC, Munich, available at http://www.cordis.lu/esprit/src/omi-home.htm, August 2005.
10. Dally W.J. and Towel B., *Principles and Practices of Interconnection Networks*, Elsevier, Amsterdam, 2004.
11. Shandhag N.R., Reliable and efficient system-on-chip design, *IEEE Comput.*, 37(3): 42–50, March 2004.
12. Hennessy J. and Petterson D., *Computer Architecture: A Quantitative Approach*, Elsevier, Amsterdam, 2003.
13. Radulescu A. and Goossens K., Communication services for networks on chip, in Bhattacharyya S., Deprettere E., Teich J., (Eds.), *Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation*, Marcel Dekker Inc., New York, 2004, pp. 193–213.
14. Overview of Open Core Protocol (OCP-2001-9-26), OCP International Partnership Association, Portland, OR 97221, USA, available at www.ocpip.org, January 2006.
15. Rowen C., *Engineering the Complex SoC: Facts, Flexible Design with Configurable Processors*, Prentice Hall, PTR, Upper Saddle River, NJ, 2004.
16. Strano G., Tiralongo S., and Pistrito C., *OCP/STBus Plug-In Methodology*, http://www.techoline.com/community/tech_group/com/tech_paper/37923, January 2006.
17. Virtual Component Interface Standard Version 2 (OCB 2 2.0), VCI Alliance, April 2001, available at www.vsi.org, March 2006.
18. Ayala J.L. et al., State-of-the-art SoC communication architectures, chapter 20, pp. 20.1–20.2, in Zurawski R., (Ed.), *Embedded System Handbook*, CRC Taylor & Francis Group, Boca Raton, FL, 2006.
19. Bertozzi D. and Benini L., Xpipes: A network-on-chip architecture for gigascale systems-on-chip, *IEEE Circuits Syst.*, 4(2): 18–31, Second Quarter 2004.
20. Jantasch A. and Tenhunen H., *Networks on Chip*, Kluwer Academic Publishers, Boston, MA, February 2003.
21. Ayala J., Lopez-Vellejo M., Bertozzi D., and Benini L., State-of-the-art SoC communication architectures, in Zurawski R., (Ed.), *Embedded Systems Handbook*, CRC Press, Boca Raton, FL, 2006, pp. 20.1–20.22.
22. ARM.AMBA Multi-Layer AHB Overview, 2001, available at http://www.arm.com, January 2006.
23. ARM.AMBA AXI Protocol Specifications, 2003, available at http://www.arm.com, January 2006.
24. Altera, Avalon Interface Specification, April 2005, available at www.altera.com, March 2006.
25. Altera, Avalon Bus Specification: Reference Manual, July 2003, available at www.altera.com, March 2006.
26. Pelissier G., Hersemeule R., Cambon G., Torres L., and Robert M., *Bus Analysis and Performance Evaluation on a SoC Platform at the System Level Design*, http:/www.ra.informatik.uni.stutgart.de/∼pricopin/noc03/paper_44.pdf, January 2006.
27. WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Revision: B.3, September, 2002, available at http://www.opencores.org/projects.cgi/web/wishbone/wishbone, March 2006.
28. Overview of the CoreFrame architecture, January 2002, available at http://www.palmchip.com, January 2006.
29. Palmchip, Overview of CoreFrame Architecture, White Paper, available at www.palmchip.com, February 2006.
30. Cordon B., *A Bus Architecture for System-on-Chip Designs, Palmchip Corporation*, available at www.palmchip.com, January 2006.
31. Bainbridge W., *Asynchronous system-on-chip interconnect*, Ph.D. Thesis, Department of Computer Science, University of Manchester, England, March 2000, pp. 119–140, available at www.cs.man chester.ac.uk/apt/projects/interconnect/, February 2006.
32. Bainbridge W. and Furber S., *Asynchronous macrocell interconnect using marble*, Technical Report, available at www.cs.manchester.ac.uk/apt/projects/interconnect/, February 2006.

AUTHOR QUERIES

[AQ1] Kindly specify if the changed reference citation from 10 to 11 is correct.
[AQ2] Please provide the names of authors for refs. [5,6,7,9,22,14,17,23,24,25,27,28,29] if appropriate.