CHAPTER

8 Custom Bus-Based On-Chip Communication Architecture Design

In addition to the bus-based on-chip communication architecture standards described in Chapter 3, there has been a lot of research on developing custom bus architectures over the last several years. Such custom architectures attempt to address the shortcomings of standard on-chip communication architectures by utilizing new topologies and protocols to obtain improvements for common design goals, such as performance and power. These novel topologies and protocols are often customized to suit a particular application, and typically include optimizations to meet application-specific design goals. Figure 8.1 shows where the custom communication architecture selection process fits into a typical electronic system level (ESL) design flow. Once a custom on-chip communication architecture has been selected, the next step is usually to perform an exploration phase, to determine the protocol and topology parameters that can best meet the design goals. In this chapter, we present some of the more significant custom bus-based on-chip communication architectures that have been proposed over the past few years. In Section 8.1, we describe split bus architectures that are useful for reducing bus power consumption. Section 8.2 presents serial bus architectures that aim to reduce wire congestion and the area footprint of the on-chip communication architecture. Section 8.3 describes code division multiple access (CDMA) based bus architectures. Section 8.4 elaborates on asynchronous bus architectures that avoid a global clock signal for synchronization, in order to reduce power consumption. Finally, Section 8.5 presents reconfigurable bus architectures that allow dynamically changing the bus topology and/or parameters (such as arbitration schemes) in order to better adapt to changing traffic conditions during application execution and improve performance.

8.1 SPLIT BUS ARCHITECTURES

Single shared bus-based communication architectures have the advantage of a simple topology and low area cost. The disadvantages of shared-bus architectures

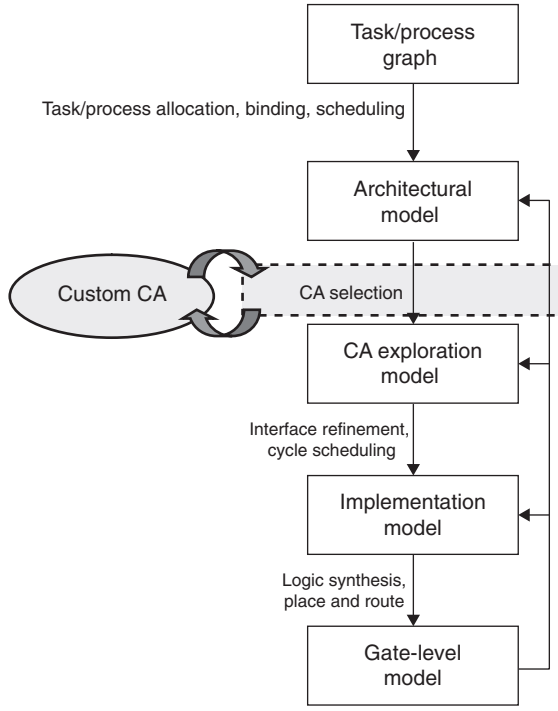

FIGURE 8.1

Custom CA selection in a typical ESL design flow

include larger wire and load capacitance, which results in a correspondingly larger power consumption as well as longer delay for data transfer (which leads to lower bandwidth). Split or segmented bus architectures attempt to overcome this drawback by splitting a shared bus into multiple segments. Split buses allow selective shutdown of unused bus segments, potentially saving energy. Furthermore, segmentation increases the parallelism by permitting parallel data transfers on different segments, which improves performance.

For instance, the segmented bus architecture [1] proposes the insertion of buffers to isolate the components on a bus. Figure 8.2 shows an example of the segmented bus architecture. An advantage of such an architecture is that only part of the bus is active at any given time. As an example, a transfer between the DSP and *X-mem* components in Fig. 8.2 causes only their corresponding segments and the buffers between them to be active, while the rest of the segments and buffers remain inactive. This reduces the effective load and wire capacitance of the active bus. Since the power consumption of the bus is proportional to the load capacitance and length of the active bus [2], the segmented bus results in power savings. Figure 8.3 compares the total bus power for the segmented bus [1] with a shared bus, for a set of benchmarks. It can be seen that a segmented bus dissipates much less power than a corresponding shared bus, since only part of the segmented bus is active at any given time, compared to a single shared bus, for which the

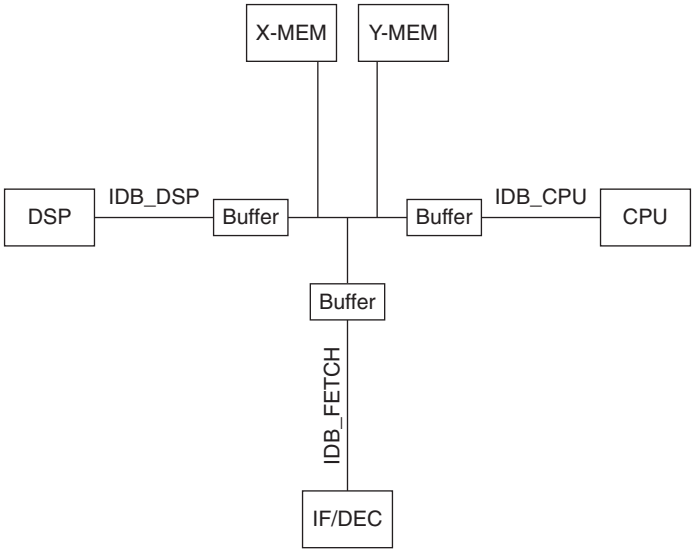

FIGURE 8.2

A segmented bus architecture [1]

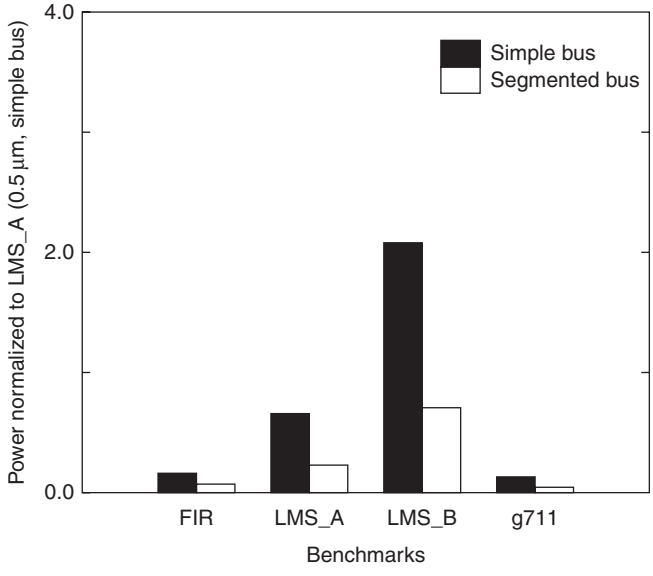

FIGURE 8.3

Total bus power comparison between segmented bus and shared bus architectures, for 0.35 micron technology [1]

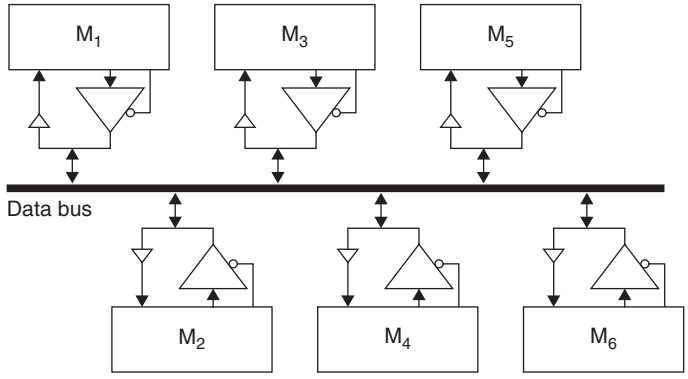

FIGURE 8.4

Monolithic single shared bus architecture [9]

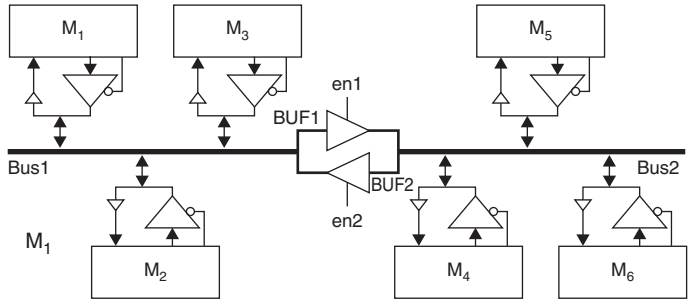

FIGURE 8.5

A split bus architecture [9]

entire bus must be active to transfer data between any two components. Such segmented/split architectures can be considered to be light-weight variants of traditional hierarchical shared bus architectures [3–8].

The split bus communication architecture proposed in [9, 10] is another variant of the split/segmented bus architecture. In a single shared bus, such as that shown in Fig. 8.4, the propagation delay between module *M1* and *M6* is large. In order to improve the timing and energy consumption of the long bus, it can be partitioned into two segments, as shown in Fig. 8.5. A dual-port driver at the boundary of *bus1* and *bus2* is responsible for relaying data between the two buses, with the data flow direction being determined by two control signals *en1* and *en2*. When *en1* is high, data can be transmitted from *bus1* to *bus2*, and when *en2* is high, data can be transmitted from *bus2* to *bus1*. When both *en1* and *en2* are low, the buses are isolated from each other.

Assuming that the internal energy of dual-port drivers is negligible and their intrinsic delay is smaller than the rest of the bus, and the energy dissipated to generate and connect control signals of the bus drivers is negligible, the split bus architecture results in energy savings compared to a simple shared bus architecture.

Table 8.1 Energy consumption of various bus architectures [9]

Architecture	Energy
$BUS = \{M_1, M_2, M_3, M_4\}$	1
$BUS1 = \{M_1, M_2\}$ $BUS2 = \{M_3, M_4\}$	0.75
$BUS1 = \{M_1, M_3\}$ $BUS2 = \{M_2, M_4\}$	0.875
$BUS1 = \{M_1, M_4\}$ $BUS2 = \{M_2, M_3\}$	0.875

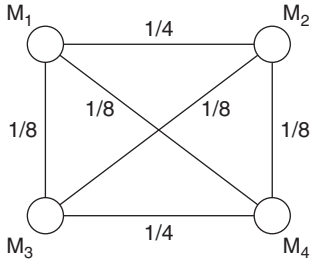

FIGURE 8.6

Data transfer probabilities between modules [9]

The energy savings will vary, depending on how the components are clustered on the different segments. Table 8.1 shows the energy consumption of different component allocations on the two segments, for a system with four modules—M_1, M_2, M_3, and M_4. The configuration in which modules M_1 and M_2 are on one segment, and modules M_3 and M_4 are on another segment results in the largest energy savings. Such a lowest energy consumption configuration can be derived from a data transfer probability graph, as shown in Fig. 8.6. The components having the highest probabilities of data transfer should be kept on the same segment, so that only that segment of the bus architecture is active during the transfer, which saves energy. This observation can be corroborated from the energy observations in Table 8.1 and the probability graph in Fig. 8.6.

Figure 8.7 shows the results of an experiment to determine energy savings of the split bus technique when compared to a shared bus architecture, where the data transfer probability between any two modules is assumed to be one of four probability distributions (*normal, exponential, delta function,* and *uniform*). The *x*-axis represents different system testbenches with varying number of modules. It can be seen that energy savings are obtained with all four data transfer probability functions when a split bus is used. Further reduction of bus energy can be obtained by using bus encoding and low-voltage swing signaling techniques [11, 12].

Another variant of the split bus was proposed by Lu and Koh [13]. The high performance bus architecture called SAMBA (single arbitration, multiple bus accesses) allows multiple masters to access the bus with only a single bus arbitration grant.

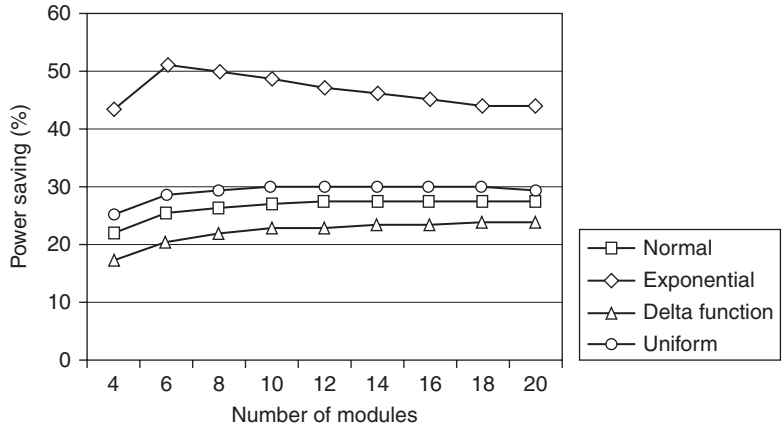

FIGURE 8.7

Power savings for different data distribution patterns [9]
© 2002 IEEE

The split nature of the bus enables idle bus segments to be used for other pending bus transactions (in addition to the transaction permitted by the arbitration), without introducing additional arbitration complexity. In traditional bus architectures, only one master can access the bus at any given time, after it is granted access to the bus by the arbiter. This leads to wastage of bus resources, which are completely consumed by a single transaction on the bus. Split [3] or out-of-order (OO) [4] transactions can overcome this bandwidth limitation, but at the cost of increased arbitration complexity, leading to an increase in arbitration delay. The SAMBA bus architecture attempts to overcome these drawbacks, and improve bus bandwidth and latency response.

Figure 8.8 shows the structure of the SAMBA bus architecture. It consists of two separate buses, each of which is used for data transfer in a forward or backward direction. The SAMBA bus architecture requires that the addresses of the modules be in an increasing or decreasing order from one end of the bus to the other. The bus transferring data from the lower address modules to the higher address modules is called the *forward* bus, while the bus transferring data from the higher address modules to the lower address modules is called the *backward bus*. A module is attached to the bus through an interface unit that can communicate with other interface units through both the forward and backward buses. Each bus operation consists of two phases: the request phase and the response phase. The communication request is sent by the initiator to the destination module via one of the buses in the request phase, and the response is sent by the destination on the other bus in the response phase. Before a transaction can begin on a bus, the interface unit of the initiator must decide whether to use the forward or backward bus, and then request the arbiter for access to that bus. The arbiter broadcasts the winner of the arbitration to all interface units, and it is possible that one module gains access to the forward bus, while another gets access to the backward bus. Multiplexers are used to combine all the signals, as shown in Fig. 8.8.

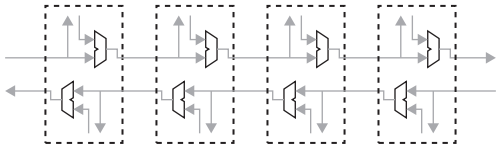

FIGURE 8.8

Structure of SAMBA bus architecture [13]

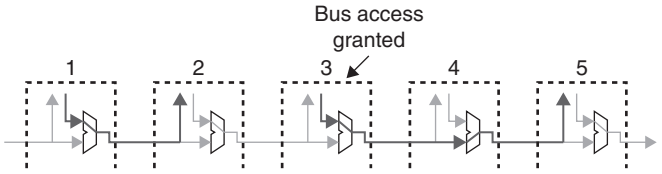

FIGURE 8.9

Multiple bus accesses with single arbitration [13]

Unlike a traditional bus architecture, a module connected to the SAMBA bus can access the bus even if it is not an arbitration winner. As long as there are no common bus segments in the paths of bus transactions, they can be performed simultaneously, allowing for more parallelism in data communication. If a module has a pending transaction for the forward bus, it can initiate the transaction if any of the following three conditions are met:

(*i*) The module wins the arbitration for the forward bus.

(*ii*) The bus transaction destination of the module is not after the arbitration winner (i.e., the address of the destination is lower than or equal to that of the arbitration winner), and no modules before this module are performing transactions with modules after it on the bus.

(*iii*) The module is after the arbitration winner and no module before it performs a bus transaction with a module after it.

Figure 8.9 depicts of a scenario where simultaneous multiple accesses to a bus occur after a single arbitration. Module 3 is the arbitration winner, and therefore the bus transaction from modules 3 to 5 is performed. Since the destination of the pending transaction at module 1 is module 2, which is before the arbitration winner, it satisfies condition (ii) above, and can therefore perform the transaction on the bus simultaneously. The ability to perform multiple transactions in a single cycle results in higher bandwidth and lower latency for the SAMBA bus. Communication latency is further reduced because transactions can be performed after automatic compatible transaction detection at the interface units without waiting for bus access from the arbiter. Figures 8.10 and 8.11 show the effective

bandwidth and average latency reduction when different SAMBA bus configurations are compared with a traditional shared bus architecture. The *x*-axis shows the number of modules in the different testbenches on which the experiments were performed. The inter-communication interval, defined as the number of bus cycles after which a new transaction is generated once the previous bus access has been granted, is randomly generated following a *Poisson* distribution. Three different SAMBA configurations are considered in the experiment, which are based on the communication distance distribution. The communication distance of a bus transaction is defined as the number of interface units between its initiator and destination. The three SAMBA configurations have different communication distance distributions: *uniform, Poisson, and exponential*. In the SAMBA-uniform configuration, a module has equal probability of generating a transaction for all other modules. In the other two distributions, an average communication distance of (*num_of_modules/4*) is also used to direct traffic generation, where *num_of_modules* is the number of modules attached to the bus. A two level TDMA/RR (time division multiple access/round-robin) arbitration scheme is used for all four

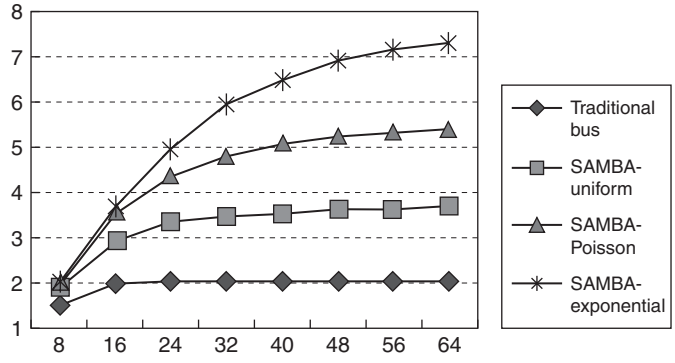

FIGURE 8.10

Effective bandwidth for buses with different number of modules [13]

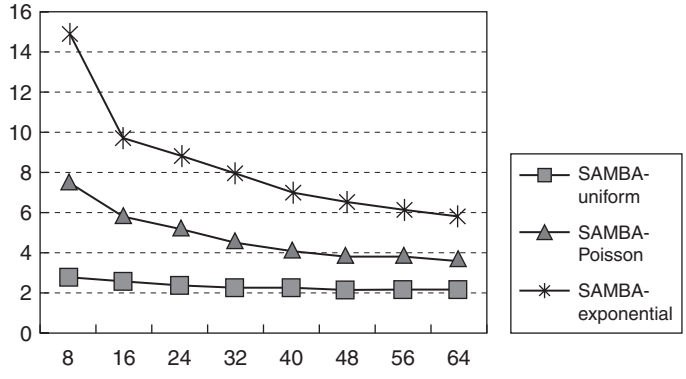

FIGURE 8.11

Average latency reduction for buses with different number of modules [13]

bus configurations that are compared, and an arbitration latency of one cycle is assumed. For fairness of comparison, it is assumed that the traditional bus architecture has two buses that can be used independently. Therefore, when the traffic on those buses is high enough, the effective bandwidth is two, as shown in Fig. 8.10. From Figs. 8.10 and 8.11 it can be seen that when the SAMBA bus is used, the effective bandwidth is improved by as much as 3.5 times, while the latency is reduced by up to 15 times.

8.2 SERIAL BUS ARCHITECTURES

Typical standard bus-based communication architectures [3–8, 14] have parallel-line buses with multiple signal lines for the data and address buses. In the deep submicron (DSM) era, coupling capacitance between adjacent signal lines leads to significant signal propagation delay and power consumption [15, 16]. Several techniques have been proposed to reduce this coupling capacitance, including increasing line-to-line spacing and non-uniform wire placement [17–20], bus ordering [21], bus swizzling [22], repeater staggering [23], and skewing signal transition timing of adjacent lines [24] (these techniques are explored in more detail in Chapters 6 and 11). One extremely effective way of reducing coupling capacitance is to reduce the number of physical signal lines, by coupling the data of m signal lines onto a single line, called a *serial link*. Such an approach was proposed by Ghoneima et al. [25] and Hatta et al. [26]. A parallel n line bus is converted into n/m serial links, as shown in Fig. 8.12. The reduction in the number of bus lines results in (i) a larger interconnect pitch, which reduces the coupling capacitance and (ii) a wider interconnect, which reduces the effective resistance,

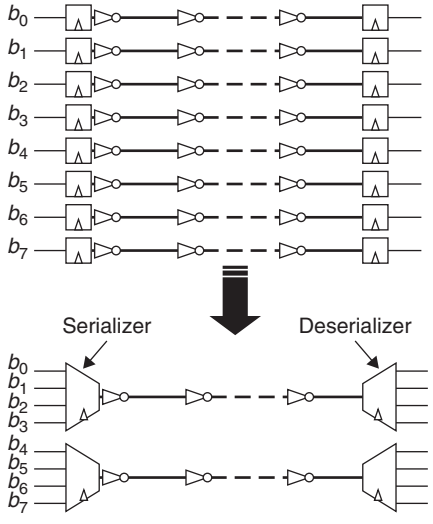

FIGURE 8.12

8-to-2 serial bus that converts an 8-bit parallel bus into 2 serial links ($n = 8$, $m = 4$) [25]

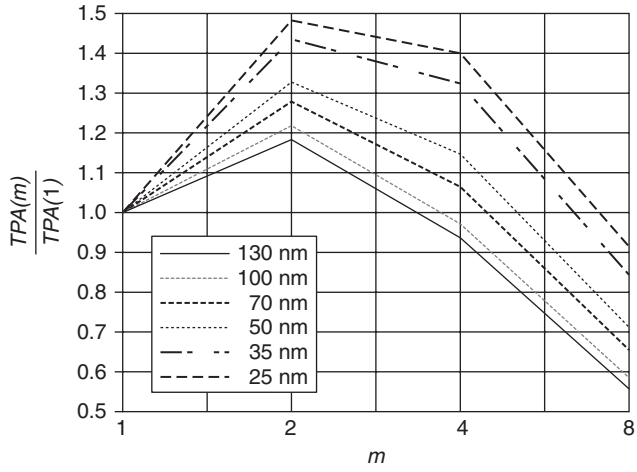

FIGURE 8.13

Relative throughput per unit area of a 64 to (64/m) serial link bus vs. degree of multiplexing *m* [25]

leading to a decrease in interconnect delay and energy consumption. A significant improvement in energy consumption can be obtained by carefully selecting the number of serial links.

Figure 8.13 shows the throughput per unit area of a 64 to (64/m) serial link bus, *TPA(m)*, normalized to the throughput per unit area of a 64-bit bus, TPA(1), and plotted as a function of the degree of multiplexing m. The technology scaling parameters were obtained from [27]. Multiplexing bus lines to create serial links results in an increase in line spacing and width (if the bus area remains constant), which reduces the line resistivity and capacitance, leading to an improvement in line bit rate. However, multiplexing also reduces the number of bus lines transferring data, which imposes a throughput penalty. Thus an optimal degree of multiplexing must exist, and is shown to have a value of 2 from Fig. 8.13. The bus energy reduction obtained for this degree of multiplexing ($m = 2$) for a 64- to 32-bit serial link is shown in Fig. 8.14. The energy reduction numbers take into account the overhead of both the driving and multiplexing circuitry. An energy reduction of up to 31.42% is obtained for the 50-nm CMOS (complementary metal-oxide semiconductor) technology node and this energy reduction is expected to be more pronounced with further technology scaling, as shown in the figure. The drawback of such a scheme is its increased latency response and a performance overhead for serializing and de-serializing communication data.

8.3 CDMA-BASED BUS ARCHITECTURES

Typically, in standard bus-based communication architectures [3–8, 14], the physical interconnect resources are shared in the time domain. A bus inherently uses a

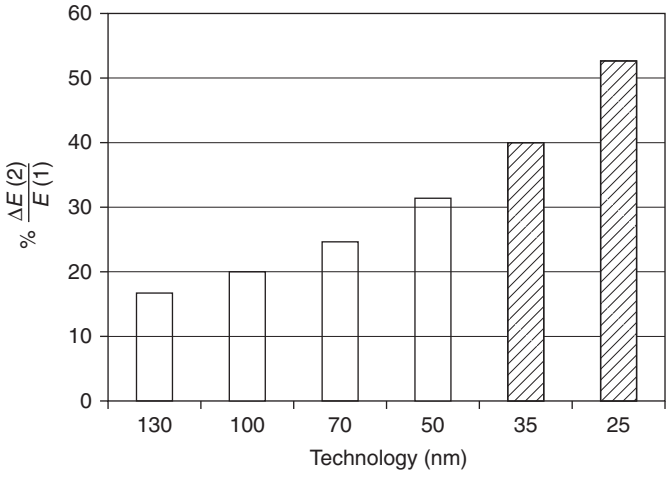

FIGURE 8.14

Bus energy reduction of a 64–32 serial link ($m = 2$) compared to a conventional 64-bit parallel line bus ($m = 1$) [25]

variant of TDMA to reuse expensive on-chip wires for multiple transactions that occur at different points of time. In contrast, CDMA is a spread spectrum technique that allows simultaneous use of the on-chip wires by multiple data transmission flows [28, 29]. It is based on the principle of codeword orthogonality, which avoids cross-correlation of codewords and allows perfect separation of data bits modulated with different codewords. In such a CDMA scheme, a transmitter on the bus modulates each of its transmitted bits with a spreading code before transmission on the bus, as shown in Fig. 8.15(a). On the bus, multiple transmissions can exist simultaneously as a multi-level signal. At the receiver's end, the signal is correlated with the same spreading code that was used by the transmitter. Since spreading codes are orthogonal, the original data that was transmitted can be retrieved. Figure 8.15(b) shows the *Walsh–Hadamard spreading* code [29] that is widely used in the communication domain. In CDMA, these spreading codes convert (or spread) each source data bit into k "chips" (which are user-specific fixed patterns), so that the spreading data rate (called *chip rate*) is k times the source bit rate. In a channel with two concurrent CDMA links, a 2-bit Walsh code is used and a data bit will be expanded into two "chips." Two clock cycles are needed to receive 1 bit of data if the receiver can receive one "chip" per clock cycle [30].

A mixed-mode bus architecture called CT-Bus was proposed by Lai et al. [30], which integrates CDMA and TDMA techniques in a hierarchical structure. The CT-Bus supports a fixed number of CDMA subchannels that are separated by different spreading codes. Two or more of these subchannels can be grouped into a subchannel group. Figure 8.16 shows the CT-Bus architecture having six subchannels, grouped into three different subchannel groups. *DF1* to *DF4* are data flows that will utilize the bus, and need to be assigned to different subchannel groups. Because of the channel isolation feature of the CDMA scheme, data flows in

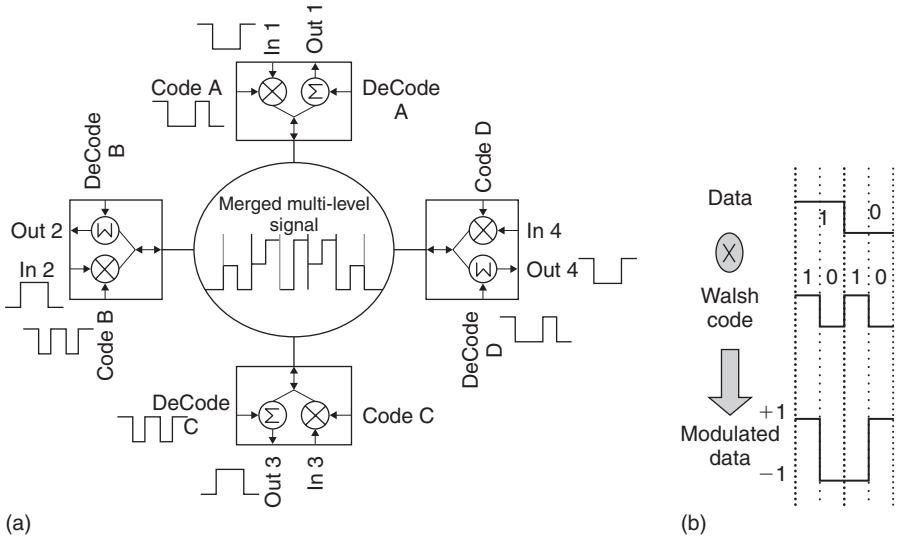

FIGURE 8.15

(a) CDMA-based interconnect (b) modulation of data and spreading code [30]

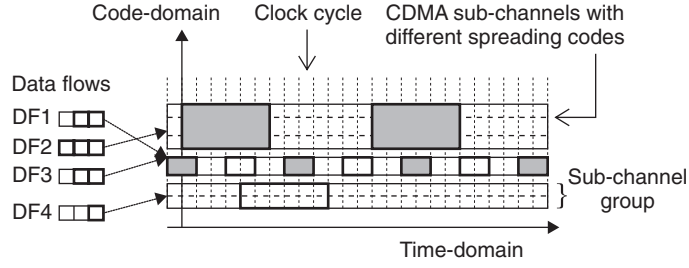

FIGURE 8.16

Architecture of CT-Bus (with three CDMA subchannel groups) [30]

different subchannel groups are isolated and do not impact each other. Within each CDMA subchannel group, data flows are assigned different TDMA timeslots. The subchannel groups can optionally use other arbitration schemes such as round-robin (*RR*) or fixed-priority (*FP*), which were described earlier in Chapter 2. The number of CDMA subchannels and subchannel groups are also configurable.

Figure 8.17 shows the latencies of two traffic flows, *DF3* and *DF4*, on a TDMA bus and on the CT-Bus, from a multimedia mobile phone case study that was simulated for a period of 1 second (*x*-axis). The *DF3* and *DF4* data flow latencies in the TDMA bus are impacted by the bursty traffic from other data flows. In contrast, due to the channel isolation feature of the CT-Bus, the latencies for the *DF3* and *DF4* data flows are very predictable and well controlled, after appropriate assignment to separate subchannel groups.

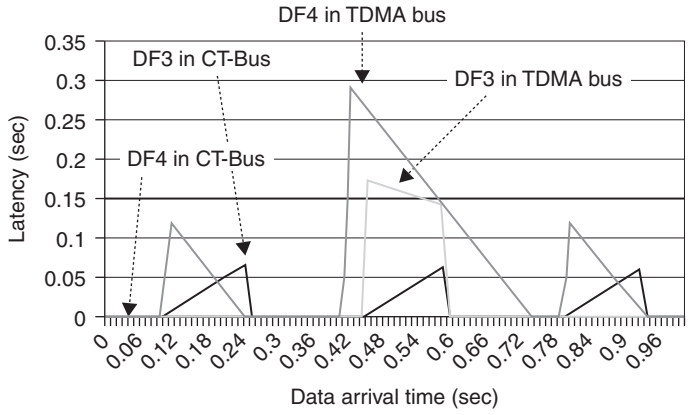

FIGURE 8.17

Latencies of DF3 and DF4 data traffic flows from multimedia mobile phone system case study [30]

An interesting discussion on CDMA and other advanced interconnects, such as frequency division multiple access (FDMA), was presented by Chang [31]. FDMA interconnects allow sharing of bus lines by assigning different data flows to different frequency channels. Data flows assigned to different frequency channels can communicate concurrently with virtually no interference, provided that undesired frequency channels are filtered out at the destination. The FDMA and CDMA can also be combined into a hybrid multi-carrier CDMA scheme, where concurrent data flows are possible by assigning appropriate codes and frequencies to each data flow pair. Such a multi-carrier CDMA scheme can have the highest aggregate data rate when compared to TDMA, CDMA, or FDMA interconnects, due to the increased bandwidth made available through the use of more than one frequency channel. However, the overhead of the modulation and demodulation transceivers at the interfaces of the communicating modules needs to be further analyzed, before CDMA-based interconnects can be considered commercially viable as on-chip communication architectures.

8.4 ASYNCHRONOUS BUS ARCHITECTURES

Asynchronous buses are implemented using clockless circuits, that is, they do not make use of a global clock signal for synchronization. Instead, synchronization occurs using additional handshake signals between transfer phases. Since the global clock generator and distributor contributes to a significant chunk of the on-chip bus power consumption, the clockless asynchronous buses have lower power consumption compared to traditional synchronous buses. Asynchronous buses also have another important advantage—resilience to clock skew even as the number of IPs (components) connected to the bus increases [32]. Performing long range communication across the chip using a widely distributed clock is difficult especially at

high frequencies where the effect of clock skew usually leads to slower and wider interconnects (Chapter 11 discusses on-chip clock networks in more detail).

An asynchronous bus architecture called MARBLE (Manchester AsynchRonous Bus for Low Energy) was proposed by Bainbridge and Furber [33, 34] for reducing on-chip bus architecture power consumption. It consists of address and data channels, and exploits pipelining of the arbitration, address, and data phases, just like in traditional synchronous bus architectures. However, due to its asynchronous nature, there is a lack of synchronization between these phases, which introduces problems in the control of bus handover between initiators. For instance, during an address phase, the bus is occupied until the slave accepts the address and completes the handshake. A similar scenario is repeated for the data channel in the subsequent data phase. The extent of handshaking in an asynchronous bus is typically more involved compared to that for a synchronous bus and is an undesirable overhead. This overhead can be overcome to an extent by the introduction of latches at the component ports, as shown in Fig. 8.18. This allows, for instance, the address packet n to be held in the latch at the target, freeing up the address channel for the initiator, which can now send packet $n + 1$. These latches thus decouple the bus from the components, and free them up for subsequent transfers.

Two separate arbiters are used in MARBLE: one for the address channel and the other for the data channel. The bus utilizes centralized decoding and arbitration. It also supports spatial locality optimizations (implemented via dedicated signals)

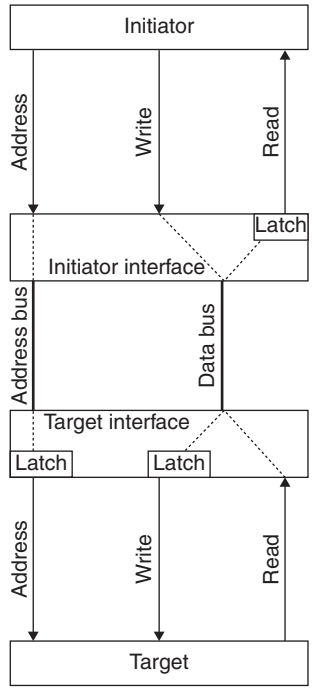

FIGURE 8.18

Decoupling the bus from components using latches in MARBLE [33]

that indicate cache line fetches and DMA transfers. Finally, MARBLE supports burst mode and in-order split transactions. All these features, coupled with its inherent asynchronous behavior result in a bus architecture that has low latency and zero quiescent state power consumption. Two drawbacks of using an asynchronous bus like MARBLE are the additional hardware logic overhead at the module interfaces, and the additional bus lines needed for handshaking.

A high performance asynchronous bus, capable of multiple issue (i.e., allowing each master to issue another transaction before the response of a previous transaction arrives), and in-order as well as OO transaction completion was presented by Jung et al. [35]. A customized layered architecture is used for the proposed asynchronous bus in order to alleviate increasing system complexity and enable optimizations. This layered architecture is shown in Fig. 8.19. The *physical, data link*, and *transport* layers are mapped to the interfaces of the IPs connected to the bus. The physical layer handles data encoding, filtering, and driving functions. The data link layer is concerned with flow and access controls. The transport layer manages burst and split transactions, multiple issue, and in-order/out-of-order transaction completion. An asynchronous handshake protocol with two-phase signaling and data insensitive (DI) encoding is used for robust and high speed data transfers on the bus [35], while four-phase signaling and bundled data transfers [35] are used at the IP interfaces for high performance and low complexity. Two asynchronous buses are proposed: (i) MI-OCB, a multiple issue on-chip bus supporting in-order transaction completion and (ii) MO-OCB, a multiple issue on-chip bus supporting OO transaction completion. Experimental results for the performance and power dissipation of these buses, when compared to a single issue on-chip asynchronous bus (SI-OCB) are shown in Figs. 8.20 and 8.21, respectively. The three asynchronous buses were implemented with a 0.25 μm CMOS process

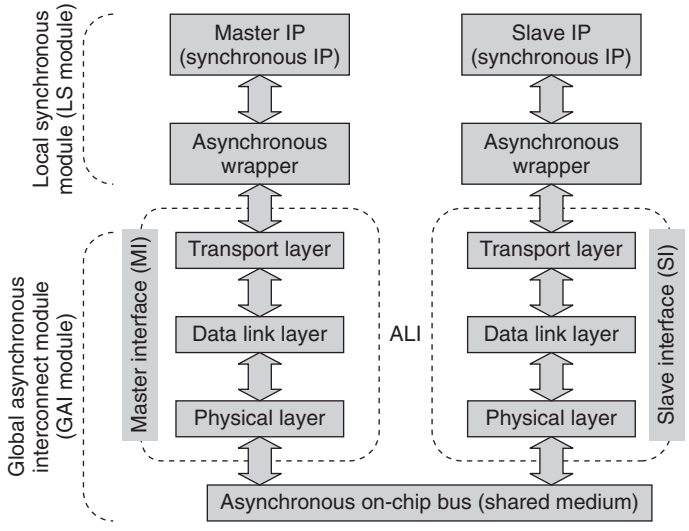

FIGURE 8.19

Layers of asynchronous bus architecture [35]

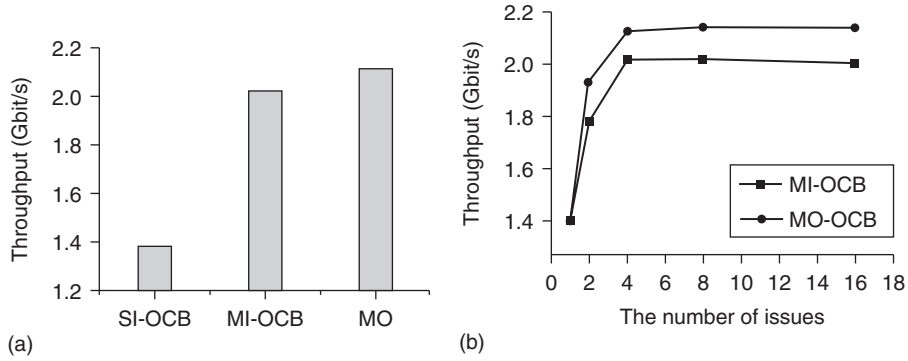

FIGURE 8.20

Simulation results for performance: (a) throughputs (b) throughputs of MI-OCB and MO-OCB as a function of the number of issues [35]

and simulation results were obtained with Synopsys NanoSim [67] at the transistor level, using the pre-layout netlist. The testbench consists of 12 IPs (4 masters and 8 slaves), with each master communicating with the slaves with the same probability. The assumptions are that the delay of all the asynchronous modules is 0, and that the ratio of the non-bus transfer time to the total transfer time per synchronous IP is also 0. The testbench workload comprises of a total of 4800 transactions. From Fig. 8.20(a), it can be seen that the throughputs of MI-OCB and MO-OCB are 31.3% and 34.3% more than for SI-OCB. The throughput of MI-OCB is lower than that of MO-OCB because no reorder buffers and related hardware controllers are needed in MO-OCB (it is the responsibility of the master IP to handle rearrangement of responses in MO-OCB). The effect of number of issues on throughputs of MI-OCB and MO-OCB is shown in Fig. 8.20(b). The value of 1 for the number of issues represents the throughput of SI-OCB. It can be seen that the throughput is saturated when the number of issues is greater than 4.

Figure 8.21(a) shows the energy consumption of the three asynchronous bus architectures. MI-OCB and MO-OCB consume 6.76% and 3.98% more energy, respectively, than SI-OCB. The energy consumption per data transaction for the three asynchronous buses as a function of the number of issues is shown in Fig. 8.21(b). The value of 1 for the number of issues represents SI-OCB. With an increase in number of issues, energy consumption increases proportionally, since the hardware complexity of the reorder buffer increases linearly.

An asynchronous crossbar bus architecture called NEXUS was proposed by Lines [36]. NEXUS is based on the Quasi-Delay-Insensitive (QDI) timing model [37] requiring that the circuit functions correctly regardless of the delays of all gates or most wires. This conservative model forbids all forms of timing races, glitches, delay assumptions and clocks, and can work robustly over delay variations caused by power supply drop, in-die variations, crosstalk, and local heating. In such a QDI system, a separate wire cannot be used to indicate the validity of the data wire because one cannot make an assumption about the relative delay of the wires. Instead, the data and validity are mixed onto two wires. Together with

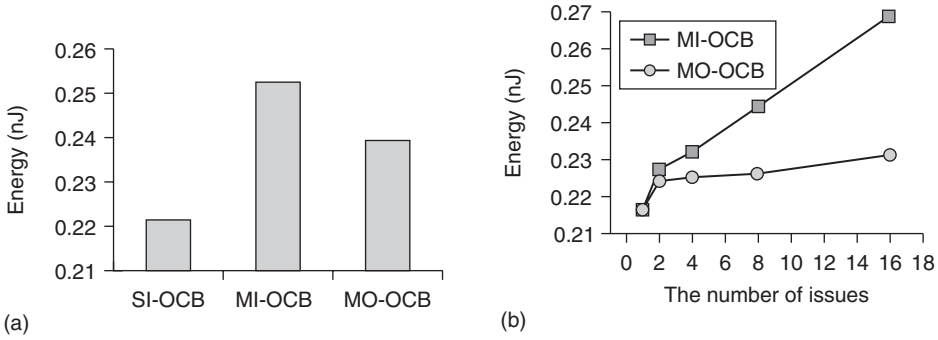

FIGURE 8.21

Simulation results for energy consumption: (a) energy consumption per data transaction (b) energy consumption per data transaction of MI-OCB and MO-OCB as a function of the number of issues [35]
© 2005 ACM Press

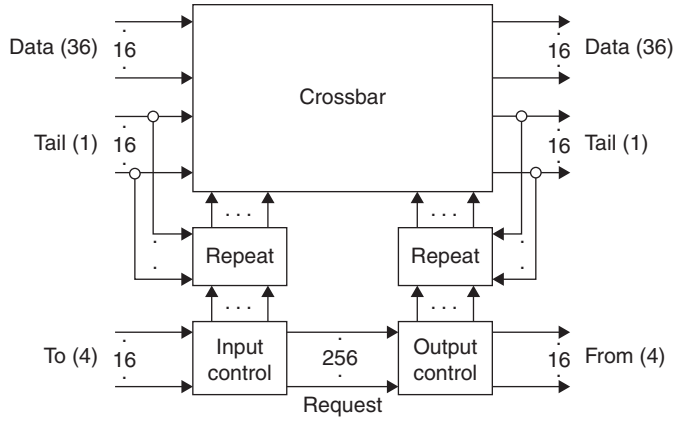

FIGURE 8.22

NEXUS crossbar decomposition [36]
© 2003 IEEE

a backward going acknowledge wire for flow control, these wires form an asynchronous channel. When both the data wires are 0, the channel is *neutral* and no data is present. When an initiator must send a bit, either the first or the second data wire is raised to send a logical 0 or 1. Once the receiver has received and stored the data, the receiver raises the acknowledge signal. Eventually, the sender puts the data wires back to *neutral*, after which the receiver lowers the acknowledge signal. This is called the four-phase dual rail handshake, and it is used in the asynchronous buses in NEXUS.

The NEXUS crossbar employs clock domain converters to bridge the asynchronous interconnect with the synchronous modules in the system. Figure 8.22 shows a decomposition of the NEXUS crossbar, which can support up to 16 modules. Data is transferred in bursts that cannot be fragmented, interleaved or dropped (i.e., *atomic* bursts). Each burst contains a variable number of words (NEXUS uses a 36-bit data path) terminated by a tail bit, and a 4-bit TO/FROM signaling to route the data to

the appropriate destination. The largest part of the crossbar is its data path, which MUXes all input data channels to the output data channels. It is controlled by a *split* channel for each input which specifies which output to send the burst to. A merge control channel is also required at the output to indicate which input to receive the burst from. The *split* control comes from the input control block, while the *merge* control comes from the output control block, as shown in the figure. In between the input/output control and the data path are *repeat* circuits that replicate the same split/merge control until a tail bit of 1 passes through the link.

The 16-port, 36-bit NEXUS crossbar with arbitration and flow control, pipelined repeaters to communicate over long wires, and clock domain converters to connect to synchronous modules was fabricated for a 130-nm CMOS process and shown to reach frequencies of 1.35 GHz with a 780Gb/s cross-section bandwidth. The area footprint of a typical NEXUS system with all 16 ports used, and an average of two pipelined repeaters per link was reported to be 4.15 mm^2, which is a small and relatively reasonable fraction of the total chip area.

8.5 DYNAMICALLY RECONFIGURABLE BUS ARCHITECTURES

Typically, once the topology and values for communication parameters such as arbitration schemes, bus clock frequencies, etc. are decided (after an exploration phase) for a bus-based communication architecture, they remain fixed for the entire lifetime of the System on-chip (SoC) design. Dynamically reconfigurable bus architectures have the ability to modify certain parameters and even the bus architecture topology dynamically during system execution. Such an ability for reconfiguration allows the communication architecture to better adapt to changing traffic patterns and needs of the system during execution, and can result in better optimization of design goals, such as power and performance. We first present research that has looked at dynamic bus parameter reconfiguration, followed by research efforts on dynamic bus topology reconfiguration.

8.5.1 Dynamic Bus Architecture Parameter Reconfiguration

On-chip communication architecture standards such as AMBA [3, 14], IBM CoreConnect [4] and Sonics Smart Interconnect [6] provide limited support for dynamically reconfiguring their parameters, to adapt to changing application requirements at runtime [38]. For instance, Sonics Smart Interconnect allows software programmable arbitration and bandwidth allocation by dynamically varying TDMA slot allocation among components. Variable-length burst sizes and software programmable arbitration priorities are supported in AMBA and CoreConnect. An additional degree of configurability is provided in CoreConnect, by allowing each bus master to indicate a desired priority to the arbiter for each bus request.

8.5.1.1 *Communication Architecture Tuners*

Lahiri et al. proposed *communication architecture tuners* (CAT) [39, 40] to adapt on-chip communication architecture parameters (mainly arbitration priority) to the varying communication needs of the system and the varying nature of data being communicated. Figure 8.23 shows an example of a CAT-based communication

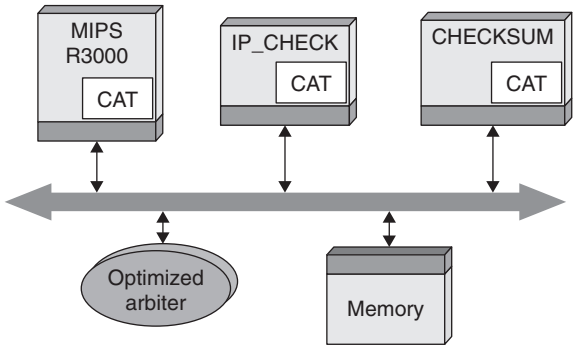

FIGURE 8.23

CAT-based communication architecture for a TCP system example [39, 40]

architecture for a TCP system from a network interface card. CATs are added to every master in the system and the arbiter and component interfaces are enhanced to handle CAT operation. Every time a communication request is generated by a component, its corresponding CAT is notified. The CAT also monitors the details of the data being communicated and the state of the component. For the system shown in Fig. 8.23, the CAT observes the packet size and deadline from the header of the packet being processed by the component, and groups communication requests based on the size and deadline of the packet being processed. The CAT then determines an appropriate arbitration priority value for the group of communication requests. Such a CAT-based communication architecture was able to dynamically adjust priorities and meet all the packet deadlines for the TCP example shown in Fig. 8.23. Whereas the traditional static priority assignment failed to meet the deadlines.

A more detailed view of the CAT module is shown in Fig. 8.24. CAT consists of a partition detector and a parameter generator circuit that generates arbitration priority values during system execution. A communication partition is a subset of the transactions generated by a component during execution. The partition detector circuit monitors and analyzes information generated by the component, such as transaction initiation requests, indications of importance of the data being processed, and tracer tokens. A component is enhanced to generate tracer tokens purely for the purpose of the CAT, to indicate specific operations to the CAT that the component is executing. The partition detector uses this information to identify the start and end of a sequence of consecutive communication transactions belonging to a partition. The parameter generator circuits generate values for communication arbitration priority based on the partition ID generated by the partition detector circuit (and other application specific data properties specified by the system designer). The generated value for the priority is sent to the arbiters and controllers in the communication architecture, to change its behavior.

Figure 8.25 shows the overall methodology for designing CAT-based communication architectures. In the first step, a performance analysis of the system execution traces for the partitioned/mapped system is performed, as described in [41], in order to obtain information and statistics for use in later steps. The trace-driven performance analysis technique used in [41] is considered comparable in accuracy to complete

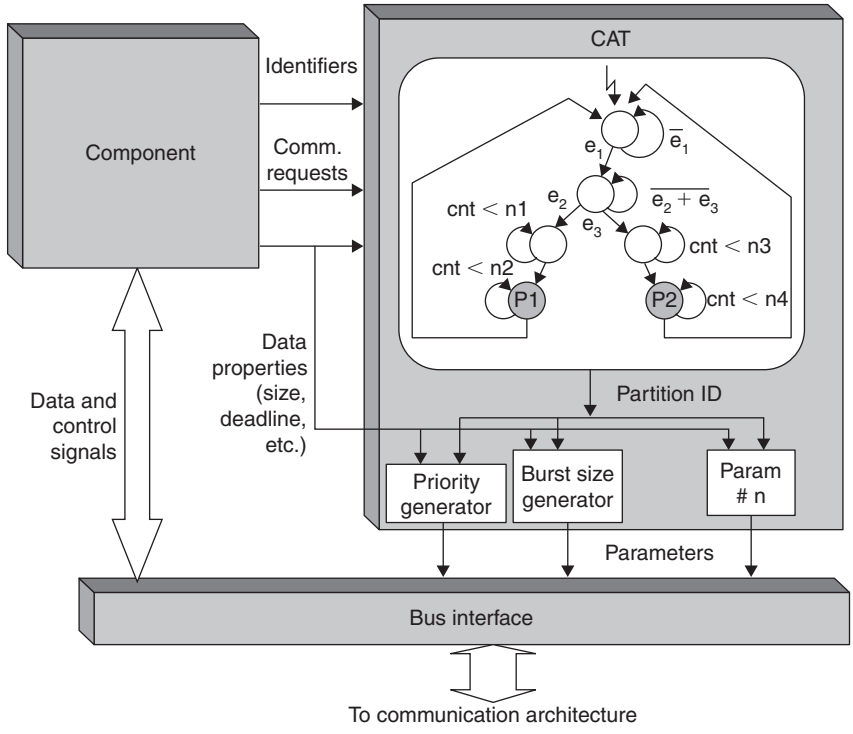

FIGURE 8.24

Detailed view of the CAT [39, 40]
© 2004 IEEE

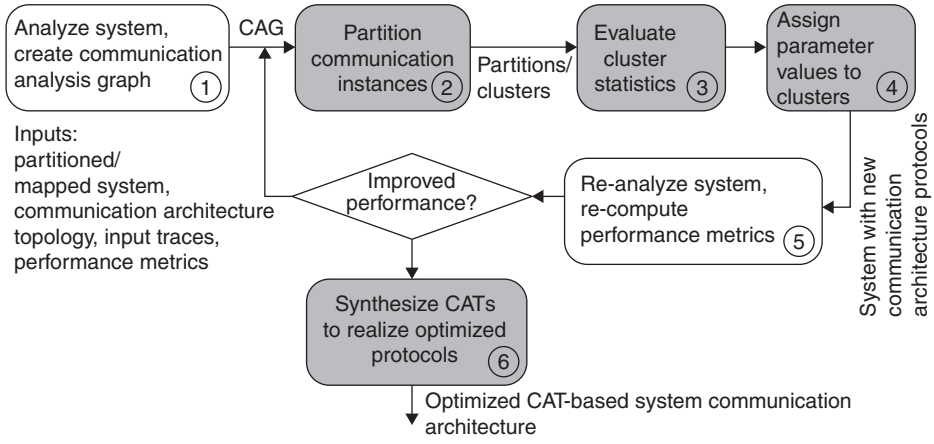

FIGURE 8.25

Methodology for designing CAT-based communication architectures [39, 40]
© 2004 IEEE

Table 8.2 Performance comparison of a CAT-based architecture with a conventional static priority-based architecture [39, 40]

Example system	Performance metric	Input trace	Static protocol-based architecture	CAT-based architecture	Improvement
TCP/IP	Missed deadlines	20 packets	10	0	–
SYS	Missed deadlines	573 transactions	413	17	24.3
ATM	Missed deadlines	169 packets	40	16	2.5
BRDG	Average cycles	10,000 cycles	304.72	254.1	1.2

system simulation, while being more efficient to employ in an iterative manner. The output of this analysis is a communication analysis graph (CAG), which is a compact representation of the system's execution for the given input traces. The vertices of this graph represent clusters of computation and abstract communications performed by the components during execution. The edges of the graph represent dependencies between the various computations and communications. The CAG can be quickly analyzed to obtain various performance statistics. In Step 2, the communication vertices in the CAG are grouped into a number of partitions. Each partition consists of events having similar communication requirements. In Step 3, various cluster statistics are evaluated, based on which arbitration priority values are assigned to the partitions in Step 4. Step 5 re-evaluates system performance for the new priority value assignments. If there is an improvement in performance, Steps 1–5 are repeated, till no further performance improvement can be achieved. Finally, Step 6 generates the CAT hardware to realize the optimized protocol generated in the previous steps.

Experimental studies were performed to compare an enhanced CAT-based communication architecture with conventional communication architectures, with static arbitration priority assignment. Four system testbenches were considered for the comparison study: (i) *TCP/IP*, a four component, single shared bus system shown in Fig. 8.23, (ii) SYS, a system with four components accessing a shared memory on a single shared bus, (iii) *ATM*, a packet forwarding unit of an ATM switch that consists of five components accessing a dual-port memory, over a single shared bus, and (iv) *BRDG*, a hierarchical shared bus-based system with two buses connected via a bridge, and six components, including two shared memories. Table 8.2 shows the performance benefits of using a CAT-based approach over a static arbitration priority-based conventional bus architecture. The performance objective for the *TCP/IP, SYS,* and *ATM* systems is to minimize the number of missed deadlines. For the *BRDG* system, each transaction is assigned a weight, and the overall performance of the system is measured using a weighted mean of the execution times of all the

Table 8.3 Effect of varying input traces (while maintaining comparable workloads) on the performance of CAT-based architecture [39, 40]

	Deadlines Met (%)	
Input trace	**Static protocol-based architecture**	**CAT-based architecture**
T-6-0	13.06	94.62
T-6-1	12.86	93.47
T-6-2	12.06	93.47
T-6-3	11.9	94.1
T-6-4	10.64	95.48
T-6-5	11.62	94.08
T-6-6	11.24	96.89
T-6-7	13.3	95.07
T-6-8	12.17	94.47
T-6-9	14.76	94.55

bus transactions. The objective for this system is to minimize the weighted average processing time. Column 4 reports the performance of the static priority-based conventional bus architecture, while column 5 reports the results of the CAT-based communication architecture. The CAT-based architecture uses information such as weights on communication requests and deadlines to provide a more flexible and higher performance communication infrastructure.

Since this methodology is dependent on performance evaluation on execution traces, it is important to determine what influence the choice of input traces has on the performance of the CAT-based communication architecture. For this purpose, the performance of the CAT-based architecture and the conventional static priority bus architecture are compared for the *SYS* system testbench, for 10 different input traces (that present comparable workloads to the communication architecture), generated using random distributions for the timing, performance requirements, and size of the communication requests. Table 8.3 compares the fraction of deadlines met by the CAT-based architecture and the conventional static priority bus architecture. It can be seen that the performance gain for the CAT-based architecture over the conventional architecture is consistent across the traces, meeting 94.66% of the deadlines on an average. It is also clear that the performance advantage provided by CATs are relatively immune to the exact sequence and timing of input stimuli experienced by the system, because CATs are not tuned to the exact arrival times of communication requests, or packet sizes, and can therefore effectively track changes in deadlines and control-flow, which can impact performance.

Figure 8.26 shows an experiment comparing the performance of the CAT-based architecture with a static priority assignment based conventional bus architecture, for the *SYS* example, for 12 different input traces that present widely varying workloads to the communication architecture. Again, it can be seen from the figure

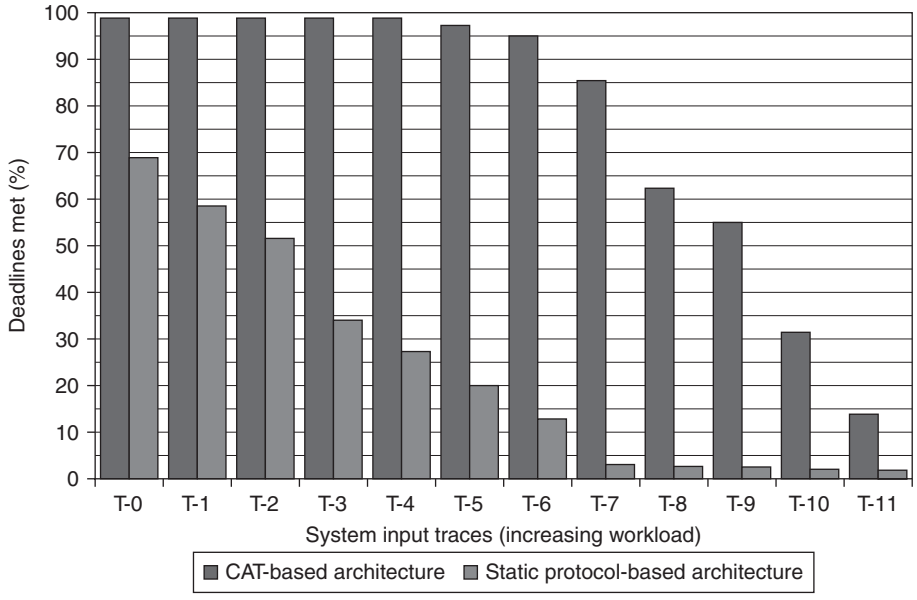

FIGURE 8.26

Performance comparison for CAT-based architecture for varying input traces (having widely different workloads) [39, 40]

that the CAT-based architecture provides better performance than the conventional bus architecture. The extent of the performance improvement varies, depending on the workload imposed by the trace. The benefit of an adaptive communication architecture like the CAT-based architecture is more pronounced for moderate to high workloads (e.g., *T-3* to *T-9*). For low workloads, both the conventional and CAT-based architectures are capable of meeting most of the deadlines, whereas for very high workloads, neither architecture can meet the deadlines. Consequently, for very low or very high workloads, the gains for the CAT-based architecture are comparatively smaller.

8.5.1.2 *LOTTERYBUS*

Lahiri et al. also proposed LOTTERYBUS [42, 43] to overcome the shortcomings of existing arbitration schemes, which can be inadequate under certain circumstances. For instance, the static priority scheme can lead to starvation of low priority masters under heavy traffic loads (i.e., the masters with low priority are rarely granted access to the bus, because of frequent high priority master transfers). On the other hand, the TDMA scheme provides a fairer distribution of bus bandwidth that can overcome starvation scenarios, but can lead to high transfer latencies due to the lack of flexibility in the static TDMA slot reservation. The LOTTERYBUS communication architecture attempts to provide effective bandwidth guarantees, while ensuring low latencies for bursty traffic with real-time latency constraints. LOTTERYBUS introduces a randomized arbitration algorithm implemented in a centralized *lottery manager* for each shared bus in an SoC. The lottery

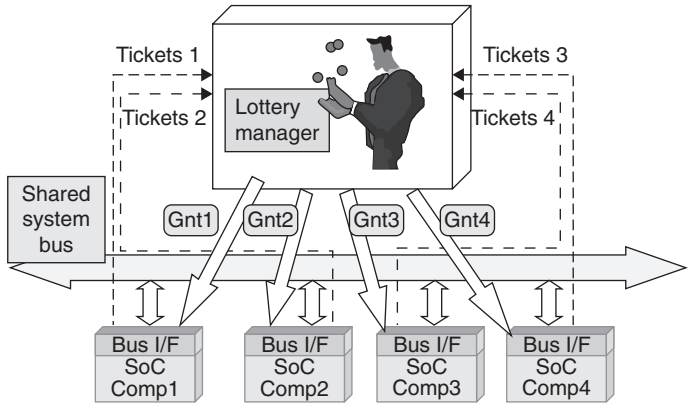

FIGURE 8.27

LOTTERYBUS communication architecture [42, 43]
© 2006 IEEE

manager receives requests from one or more masters on the bus, each of which is (statically or dynamically) assigned a number of *lottery tickets*, as shown in Fig. 8.27. The manager probabilistically chooses one of the masters as the winner and grants it access to the bus. While multiple word (burst) transfers are allowed, a maximum transfer size ensures that none of the masters monopolizes the bus for extended periods at a time.

The principle of the LOTTERYBUS operation can be explained as follows. Let $C_1, C_2, \ldots, C_n$ be the set of masters on the bus. Let the number of tickets held by a master be $t_1, t_2, \ldots, t_n$, and at any cycle. In addition, let the set of pending requests at any cycle be represented by $r_1, r_2, \ldots, r_n$, where $r_i = 1$ if master C_i has a pending request (and $r_i = 0$ otherwise). Then the probability that master C_i gets access to the bus is given by:

$$P(C_i) = \frac{r_i \cdot t_i}{\sum_{j=1}^{n} r_j \cdot t_j}$$

To decide on the arbitration winner, the lottery manager uses the notion of a lottery [68], and first examines the total number of tickets possessed by contending masters, given by $\sum_{j=1}^{n} r_j \cdot t_j$. It then generates a random number (or picks a winning lottery) from the range $[0, \sum_{j=1}^{n} r_j \cdot t_j)$[1] to determine which master to grant the bus to. If the number falls in the range $[0, r_1 \cdot t_1)$, the bus is granted to master C_1. If it falls in the range $[r_1 \cdot t_1, r_1 \cdot t_1 + r_2 \cdot t_2)$, it is granted to component C_2, and so on. For example, in Fig. 8.28, masters C_1, C_2, C_3, and C_4 are assigned 1, 2, 3, and 4 lottery tickets, respectively. In the bus cycle shown, only C_1, C_3, and C_4 have pending requests, and hence the number of current tickets is $\sum_{j=1}^{n} r_j \cdot t_j = 1 + 3 + 4 = 8$. The random number generator generates a number in the range $[0, 8)$ that happens to be 5, which lies between $r_1 \cdot t_1 + r_2 \cdot t_2 + r_3 \cdot t_3 = 4$,

[1] This set $[a, b)$ includes all the integers between a and b, inclusive of a but not b.

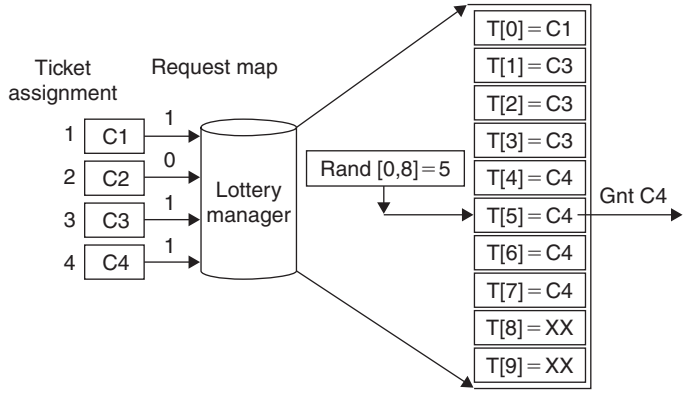

FIGURE 8.28

Example of lottery to determine arbitration winner in LOTTERYBUS [42, 43]
© 2006 IEEE

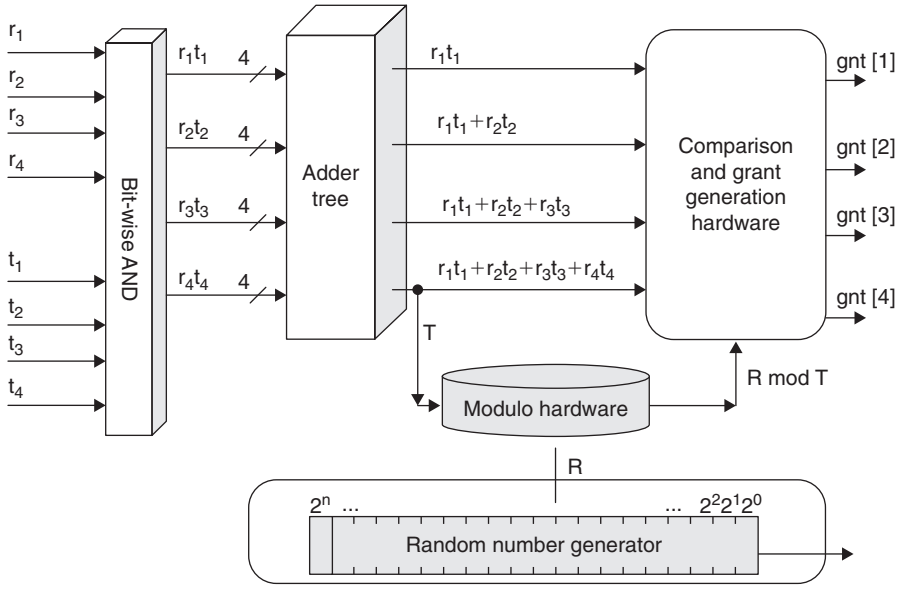

FIGURE 8.29

Lottery manager for dynamic LOTTERYBUS architecture [42, 43]
© 2006 IEEE

and $r_1 \cdot t_1 + r_2 \cdot t_2 + r_3 \cdot t_3 + r_4 \cdot t_4 = 8$. Therefore, the bus is granted to master C_4. LOTTERYBUS addresses the problem of a low priority master not being able to access the bus for extended periods of time, since the probability p that a component with t tickets is able to access the bus within n lottery drawings is given by the $1 - (t/T)^n$, which converges rapidly to 1, ensuring that no master is starved.

As mentioned earlier, tickets in the LOTTERYBUS architecture can be assigned to masters either statically or dynamically. The dynamic ticket assignment case is of particular interest, since it allows better adaptation to changing traffic and

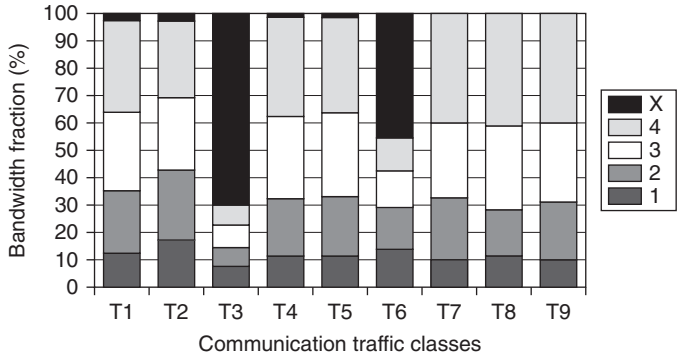

FIGURE 8.30

Bandwidth allocation of LOTTERYBUS for different communication traffic classes [42, 43]
© 2006 IEEE

performance requirements. Figure 8.29 shows the lottery manager for the LOTTERYBUS architecture with dynamic ticket assignment. The inputs to the lottery manager are the master request lines (r_1,r_2,r_3,r_4) and the number of tickets currently possessed by each master. At each lottery, the partial sum $\sum_{j=1}^{i} r_j \cdot t_j$ must be calculated for each master C_i (unlike in the static case, where the partial sum values are fixed and can be stored in a lookup table). For C_4, this yields the total range, or the sum of the number of tickets held by all masters. The final result, $T = r_1 \cdot t_1 + r_2 \cdot t_2 + r_3 \cdot t_3 + r_4 \cdot t_4$ defines the range in which the random number to be generated must lie. Modulo hardware arithmetic is used to generate the random number in the range $[0, T)$. The random number is then compared in parallel against all four partial sums using comparators, and a grant signal is generated for the appropriate winning master, using the output range analysis described earlier.

The performance of LOTTERYBUS was studied through several experiments. A simple four master, four slave shared bus system testbench was used [44], with the masters connected to parameterized traffic generators. All the system components were specified in Esterel and C, from which PTOLEMY [45] simulation models were generated using POLIS [46]. PTOLEMY was used for schematic capture and HW/SW co-simulation.

The first experiment examined the ability of LOTTERYBUS to proportionally allocate bandwidth under different classes of communication traffic. Figure 8.30 shows the results of this experiment, with the *x*-axis depicting nine different communication traffic classes and the *y*-axis depicting the fraction of the total bus bandwidth allocated to masters. It can be seen that for traffic classes with high bus utilization, the bandwidth allocated closely follows the assignment of lottery tickets. Tickets were assigned in the ratio 1:2:3:4 and for the traffic classes *T4, T5, T7, T8,* and *T9*, the bandwidth allocated is in the ratio of 1.15:2.09:2.96:3.83. However, for cases when the bus is partially un-utilized (e.g., *T3, T6*), the bandwidth allocation does not follow ticket assignment and is roughly the same for all components. This is because due to the sparse nature of communication in these classes, immediate grants are issued to most requests. These results show that LOTTERYBUS is capable of providing efficient control over bus bandwidth allocation for a variety of traffic classes and a varying level of bus utilization.

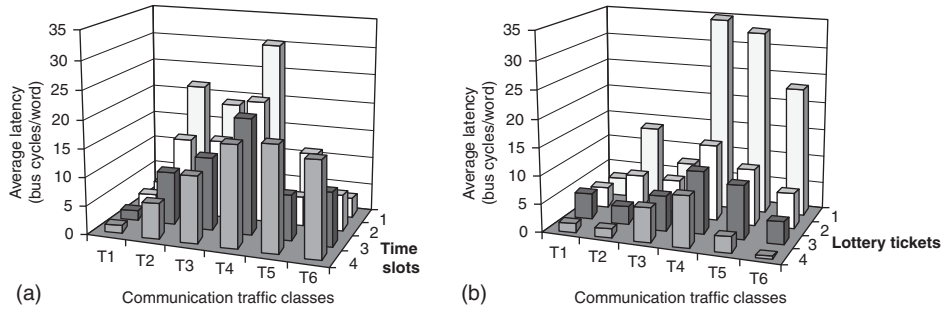

FIGURE 8.31

Communication latencies over different communication traffic classes for (a) TDMA and (b) LOTTERYBUS [42, 43]
© 2006 IEEE

Figure 8.31 compares the latency of the TDMA and LOTTERYBUS architectures over six different communication traffic classes. The *x*-axis depicts different traffic classes, while the *y*-axis depicts timeslots (Fig. 8.31(a)) and lottery tickets (Fig. 8.31(b)) assigned to the masters. The *z*-axis depicts the average communication latency per word. It can be seen from the figures that LOTTERYBUS exhibits better latency behavior than the TDMA architecture for a wide range of traffic classes. Most importantly, the communication latency for high priority masters varies significantly for the TDMA architecture (1.65 to 20.5 cycles per word), because the latency of communication in TDMA is highly sensitive to the timing wheel position (i.e., which master's slot currently has access to the bus) when the request arrives. The LOTTERYBUS architecture does not exhibit this phenomenon and ensures low latencies for high priority masters.

The experimental results thus show that LOTTEYRBUS is able to simultaneously provide low latencies for high priority traffic, while at the same time providing proportional bandwidth guarantees. The LOTTERYBUS architecture was implemented on top of the AMBA AHB architecture [3] and synthesized using the Synopsys Design Compiler [47] for a 0.15 μm CMOS cell library from NEC [48]. A communication architecture area increase of 16% for the static LOTTERYBUS and 24% for the dynamic LOTTERYBUS architecture was observed over the static priority-based communication architecture area. The critical path of the static priority-based architecture was observed to be 1.68 ns, enabling, at least theoretically, bus speeds up to 595 MHz. This critical path delay was unchanged for the static LOTTERYBUS architecture. The critical path delay for the arbiter in the dynamic LOTTERYBUS architecture was measured to be 1.92 ns, which is a 14% increase in the critical path of the overall communication architecture. With current technology scaling trends, such a logic delay will play a decreasing role compared to the global wire delay in determining overall communication architecture clock frequency [15], making the deployment of dynamic LOTTERYBUS architecture more feasible in future designs. The authors did not provide any information about the power dissipation overhead of the additional circuits, which is also a critical factor in considering such schemes for deployment.

8.5.1.3 *Other Dynamic Parameter Adaptation Schemes*

A derivative statistic-based dynamic lottery arbitration scheme was proposed by Zhang [49], which additionally makes use of the arbitration history record to determine the priority of the masters for the next arbitration grant. This implies that the priority of master *A* is higher than that of master *B* if master *A* was granted bus access more than master *B* during the last *L* times arbitration was performed. Each master has *M* registers to store their history record for the number of times it was granted bus access during the last *L* times arbitration was performed. The value of *L* is also stored in a register. In contrast to the lottery manager in LOTTERYBUS, the lottery manager issues tickets based on the values of the history record registers and the initial ticket registers. Results of experiments showed that the proposed statistic-based lottery scheme provided superior performance compared to the lottery-based scheme in LOTTERYBUS. However, no experiments were performed by the authors to determine the additional area overhead or timing impact of the history-based ticket generation. It was also not shown whether the statistic-based lottery scheme provides low latencies for high priority traffic, while at the same time providing proportional bandwidth guarantees, like LOTTERYBUS.

A *dynamic fraction control bus architecture* was proposed by Wang and Bayoumi [50] to provide similar benefits as LOTTERYBUS, but with lower system cost and design complexity. Additionally, since arbitration in LOTTERYBUS is based on probability, it becomes hard to implement accurate control over the bus bandwidth allocation for applications. In the proposed fraction control bus, bandwidth fractions are assigned to master components based on their communication requirements. The greater the fraction value, the greater the priority. Figure 8.32 shows the architecture of the fraction bus arbiter and decoder. The decoder is responsible for granting bus access to masters or produce chip select signals for comparators, based on master requests. The fraction calculator calculates the real-time bandwidth fraction for each master. The assigned fraction values for the masters are stored in a Lookup Table (LUT). Comparators perform fraction comparison and grant access to the master that satisfies arbitration conditions. The proposed fraction control bus can

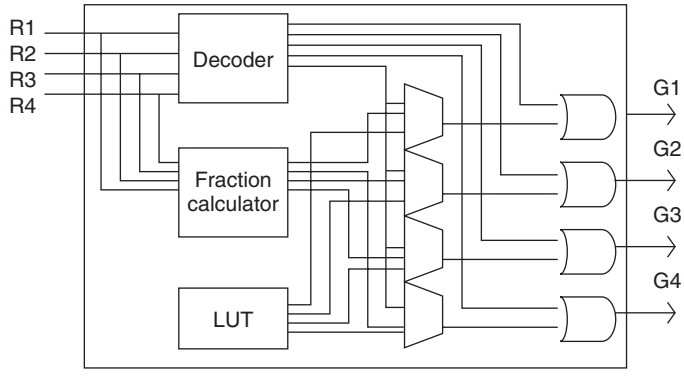

FIGURE 8.32

Architecture of the fraction bus arbiter and decoder [50]

be implemented statically or dynamically. In the *static fraction control bus* (SFCB), the fractions assigned to the masters are fixed. In the *dynamic fraction control bus* (DFCB), the fractions are initially set to a fixed value, and are thereafter continuously tuned to adapt to communication circumstances dynamically. The assigned bandwidth fraction for a master can be increased if it has pending requests more than a threshold value. In such a case, the master can *borrow* the bandwidth fractions from other masters temporarily, till the number of pending requests drops below the threshold value. Both the threshold value and the amount of bandwidth that can be borrowed are configurable by the designer.

Several experiments were performed to determine the effectiveness of the fraction control bus over LOTTERYBUS and conventional static priority bus architectures. Table 8.4 compares the gate count and achievable bus speed for the implementations of the static priority bus architecture, the LOTTERYBUS architecture and the static (SFCB) and dynamic (DFCB) fraction control bus architectures. The implementations were mapped onto the Xilinx Vertex2Pro FPGA. It can be seen from the table that the proposed fraction control bus architectures have lower area and delay compared to LOTTERYBUS. To evaluate and compare the performance of the fraction control bus, a four master subsystem for the cell forwarding unit of an ATM switch [51] was considered. Four versions of the system were implemented, with: (i) the static priority bus, with priorities 4, 3, 2, 1; (ii) LOTTERYBUS, with lottery numbers 1:1:4:6; and (iii), (iv) SFCB and DFCB architectures, with fractions 15%:15%:60%:10%. Table 8.5 shows the results of this experiment. For the static priority bus, the communication latency of the master with the highest priority is the lowest, but the bandwidth fraction for the masters with low priorities is extremely low, due to starvation. The fraction control buses (SFCB and DFCB) have lower system cost while

Table 8.4 Design complexity and achievable bus speed comparison [50]

	Gate counts	Delay (ns)	Max speed (MHz)
Priority	86	2.707	369.41
LotteryBus	152	3.276	305.72
SFCB	104	2.987	334.89
DFCB	134	3.113	321.23

Table 8.5 Performance comparison for ATM switch example [50]

	Port4 latency (cycles)	Port4 BW (%)	Port3 BW (%)	Port2 BW (%)	Port1 BW (%)
Priority	1.39	9.56	60.6	29.83	0.01
Lottery	1.4	9.32	63.6	15.15	11.93
SFCB	1.42	10.32	60.4	14.64	14.64
DFCB	1.46	9.74	58.25	16.34	15.67

maintaining comparable communication latencies, with LOTTERYBUS. SFCB and DFCB can also be seen to have a more accurate control over the allocation of bandwidth fractions, than other buses.

A time-division-based bus architecture which dynamically allocates TDMA timeslots (*dTDMA*), was proposed by Richardson et al. [52]. In dTDMA, the bus arbiter dynamically grows or shrinks the number of timeslots to match the number of active transmitters, as shown in Fig. 8.33. When a master needs to transmit data on the bus, it asserts its active signal to the arbiter, to request a timeslot. The arbiter uses a number of techniques to decide on a timeslot assignment for each master and produces a new configuration for each active transmitter and receiver before the beginning of the next clock cycle. On the next clock edge, the timeslot configuration data is loaded by the transmitters and receivers, and normal operation is continued. When a master finishes transmitting, it de-asserts its active signal, following which the arbiter de-allocates its timeslot in the same manner as it allocated it (Fig. 8.33). Such a dynamic timeslot assignment produces the most efficient timeslot allocation without any slot wastage. The only overhead is the one cycle initial communication delay when a timeslot is allocated. Various methods can be used to assign timeslot, including (but not limited to) methods based on the status of the transmit buffers or the length of the wait time.

The address mapped dTDMA bus architecture has several advantages over standard bus architectures such as AMBA [3]. Because of its memory-oriented design, AMBA imposes certain restrictions on the nature of addressing behavior. In addition to a 1kB address boundary on sequential transfers, a new transaction must be initiated if the next address is not an increment of the previous (i.e., a non-sequential access). These restrictions result in repeated arbitration overhead. For instance, a long transmission crossing the 1kB boundary must re-arbitrate, at the risk of losing bus ownership. Such re-arbitration can waste several cycles and add a significant overhead. To overcome these drawbacks during data streaming, the dTDMA bus architecture is transaction-less and address mapped, with each component on the bus being assigned a unique identifier. The dTDMA bus only requires re-arbitration when the destination of the data stream transfer changes. In contrast, for AMBA, transfer requests that are not sequential or that cross the 1kB boundary, require arbitration even if the destination of the transfer remains the same.

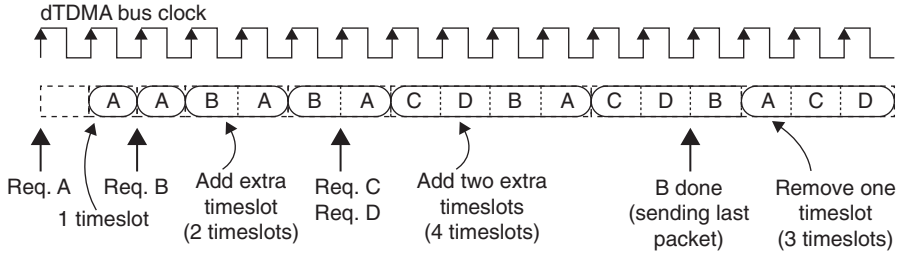

FIGURE 8.33

Dynamic timeslot allocation example [52]

A simple comparison between AMBA and dTDMA is shown in Fig. 8.34. In this example, master *A* requests access to the bus at clock *T1*, followed by a request from master *B* at clock *T2*. The AMBA arbiter does not grant access to master *A* before clock *T3*, whereas the dTDMA arbiter issues the new timeslot configuration before the end of clock *T1*. Data transmission for AMBA commences at *T5*, since *T4* is dedicated to the address phase of the transaction. In dTDMA, data transmission commences earlier, at *T2*. Master *B* has to wait five cycles from request to data transfer in AMBA, but only one cycle in dTDMA. In the example, the dTDMA bus architecture completes the transmission of two words from each master three cycles before the AMBA bus does.

An attractive quality of dTDMA is its predictable latencies, since a component is guaranteed to wait no longer than the number of active transmitters. In contrast, in the AMBA bus, a master may need to wait for an indeterminate amount of time before being granted access to the bus. The transaction-less, address-mapped dTDMA bus also requires fewer arbitrations compared to AMBA, and needs fewer cycles for the arbitration process, which improves overall performance.

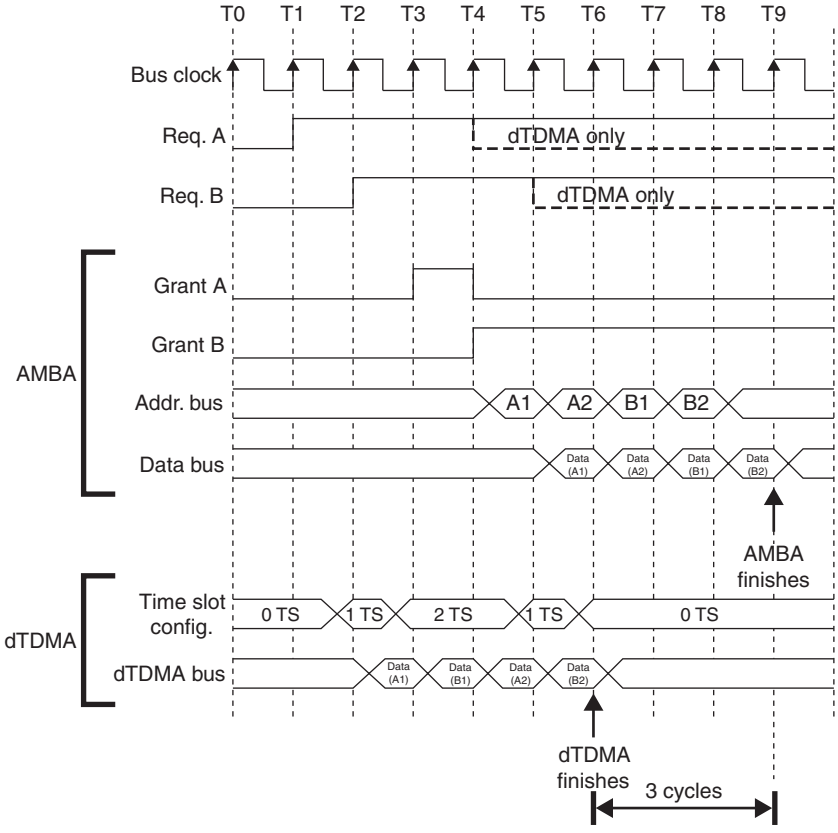

FIGURE 8.34

A simple two-component transfer on dTDMA and AMBA [52]

8.5.2 Dynamic Bus Architecture Topology Reconfiguration

In addition to dynamically configuring bus architecture protocol parameters such as arbitration schemes and burst sizes, it is also possible to change the topology of the bus architecture dynamically. Sekar et al. [51] proposed the *FLEXBUS* architecture, which is a high performance on-chip bus architecture with a dynamically configurable topology that can be implemented on top of an existing standard communication architecture such as AMBA AHB [3]. FLEXBUS is capable of detecting runtime variations in communication traffic, and adapting the topology of the communication architecture in two ways: (i) dynamic bridge bypass, which enables bus topology customizations via runtime fusing and splitting of bus segments and (ii) dynamic component re-mapping, which allows runtime switching of components from one bus segment to another. The key challenges of such an approach are maintaining compatibility with existing bus standards, minimizing the timing impact, minimizing the logic and wiring complexity, and providing low reconfiguration overhead.

The hardware required to support dynamic bridge bypass is shown in Fig. 8.35, for a system consisting of two AMBA AHB bus segments: *AHB1*, containing two masters and one slave; and *AHB2*, containing one master and one slave. This system can be operated in a single shared bus mode or a multiple bus mode by disabling or enabling the bridge bypass with the *config_select* signal. In the multiple bus mode, the signals shown by dotted lines are inactive, and the two bus segments operate concurrently, with the two arbiters resolving conflicts on each of the segment and transactions between the two segments passing through the bridge. In the single shared bus mode, *config_select* == *1*, which results in the bridge being bypassed and the signals shown with dotted lines being activated. In this mode, multiplexers are used to bypass the bridge logic and directly route data between the components in a single cycle.

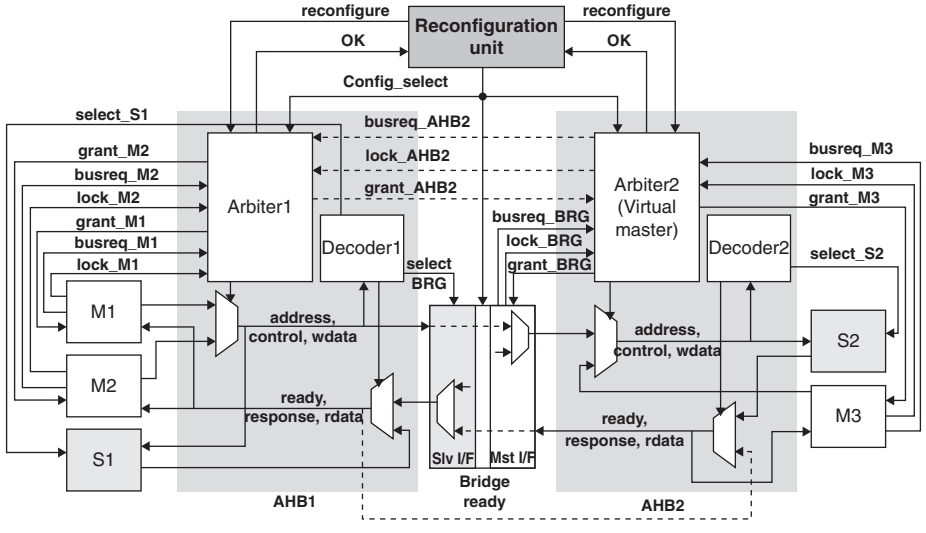

FIGURE 8.35

Dynamic bridge bypass hardware in FLEXBUS [51]

Arbitration in a single bus mode only grants one master access to the bus, whereas in multiple bus mode more than one master can have transactions executing in parallel. The arbitration mechanism of the multiple bus mode is adapted to meet the requirements of the single bus mode in FLEXBUS. This is done by using a distributed arbitration mechanism in the single bus mode, in which one of the arbiters acts as a virtual master that is regulated by the other arbiter. For example, in Fig. 8.35, if *Arbiter2* receives a transfer request from a master on *AHB2*, it immediately sends a request to *Arbiter1* using *busreq_AHB2* and *lock_AHB2* signals. *Arbiter1* arbitrates from the requests received on *AHB1*, as well as the requests from the virtual master. In parallel, to reduce arbitration latency, *Arbiter2* arbitrates among its received requests. However, it grants *AHB2* to its selected master only after receiving the *grant_AHB2* signal from *Arbiter1*, thus ensuring that only one master gets access to the bus in the single bus mode.

The reconfiguration unit (Fig. 8.35) selects the bus configuration at runtime, and ensures correct operation of the system when switching between the two configurations.The worst case overhead of bus reconfiguration for the two bus segment AMBA system is 17 clock cycles, assuming a single cycle slave response and that the bus is not locked.The runtime reconfiguration policy used in FLEXBUS can be described as follows: at runtime, the system observes the number of transactions on each local bus segment, as well as transactions between bus segments, for a time period *T*. Assuming that the average number of cycles required for a local transaction, and a cross-bridge transaction are known, the reconfiguration unit calculates the time required to process traffic for the single bus mode and the multiple bus mode. If the time taken to process the traffic for the single bus mode is less than the time for the multiple bus mode, the reconfiguration unit selects the single bus mode. Otherwise, the multiple bus mode is selected. The configuration time period *T* must be carefully set by the designer. A smaller value for *T* can result in a system more responsive to variations in traffic conditions, but if the traffic characteristics change rapidly, it can result in frequent switching between the configurations, and performance degradation due to the large reconfiguration overhead.

The hardware required to implement dynamic component re-mapping is shown in Fig. 8.36, which illustrates a two segment AMBA AHB architecture in which master *M2* and slave *S2* can be dynamically mapped to either *AHB1* or *AHB2*.The signals *config_select_M2* and *config_select_S2* are used to select the mapping of *M2* and *S2*, respectively.The signals of the re-mappable master or slave are connected to both buses, but the switch logic in *SWITCH_M* and *SWITCH_S* activates the signals for only one of the buses at a time, depending on the configuration chosen.The arbiters do not require any change when re-mapping components since master requests are only sent to the arbiter on the bus to which a master is connected. However, the decoders on the two bus segments need to be reconfigured to generate the correct signal for the re-mappable slave.The *Remap unit* (Fig. 8.36) is responsible for generating the signals to select the master and slave mapping configurations. Monitoring strategies, similar to the ones used for the dynamic bridge bypass can be used to determine when to remap the re-mappable masters and slaves.

Experimental studies were performed to evaluate the usefulness of the FLEXBUS approach. The AMBA AHB RTL description from the Synopsys

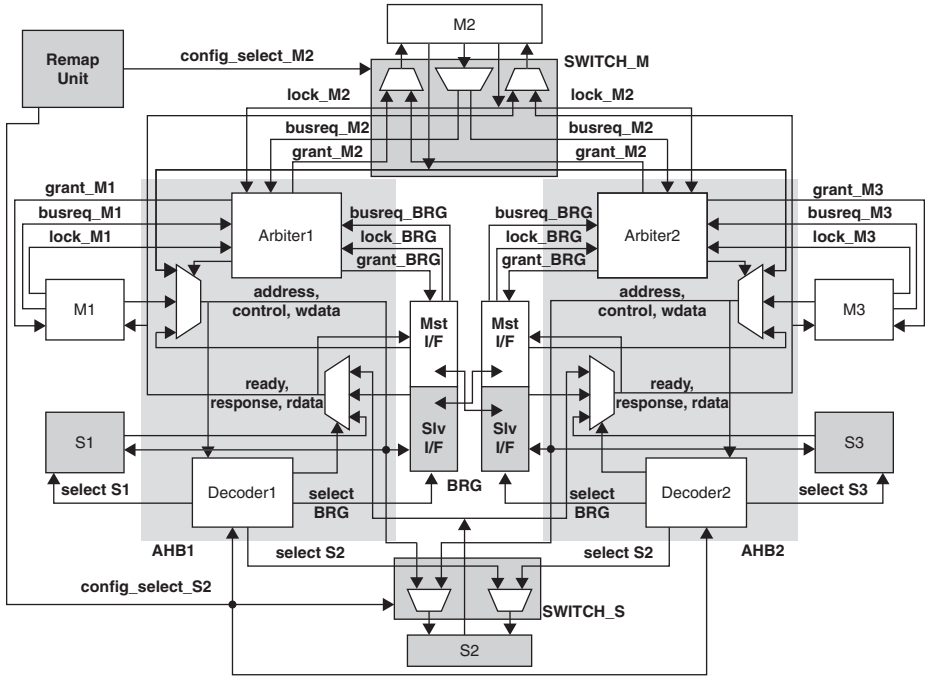

FIGURE 8.36

Dynamic component re-mapping hardware in FLEXBUS [51]
© 2005 IEEE

Designware [53] library was enhanced with the additional hardware descriptions needed to implement FLEXBUS. A system with eight masters (traffic generators) and eight slaves was used as a testbench. The first experiment explored the area and timing characteristics of the FLEXBUS implementation for a dynamic bridge bypass, and compared the results with those for a single shared bus architecture, and a multiple shared bus architecture. Table 8.6 shows the results from the experiment, for an implementation in the 0.13μm CMOS process technology [54]. FLEXBUS incurs a small delay penalty compared to statically configured architectures, due to the additional wiring and logic delay. For FLEXBUS, the critical path delay in the multiple bus mode is smaller than in the single bus mode since many of the long paths present in the single bus mode are not used in the multiple bus mode. The static multiple bus architecture has a smaller delay than the single shared bus due to less bus loading and shorter wire lengths.

Next, an experiment was performed to analyze the performance of FLEXBUS under a synthetic traffic profile [51]. Figure 8.37 plots the cumulative latency for the different architectures. It can be seen that FLEXBUS successfully adapts to frequent changes in traffic characteristics, to achieve performance improvements over the static single shared bus (21.3%) and multiple shared bus (17.5%) architectures. Finally, the performance of FLEXBUS and conventional architectures was compared for an IEEE 802.11 MAC (Message Authentication Code) processor-based SoC subsystem. All the buses were operated at 200 MHz. Table 8.7 shows the average time

Table 8.6 FLEXBUS hardware implementation results [51]

Bus architecture	Area (sq. mm)	Delay (ns)	Frequency (MHz)
Single shared bus	82.12	4.59	218
Multiple bus	84.27	3.79	264
FLEXBUS (single bus mode)	82.66	4.72	212
FLEXBUS (multiple bus mode)		3.93	254

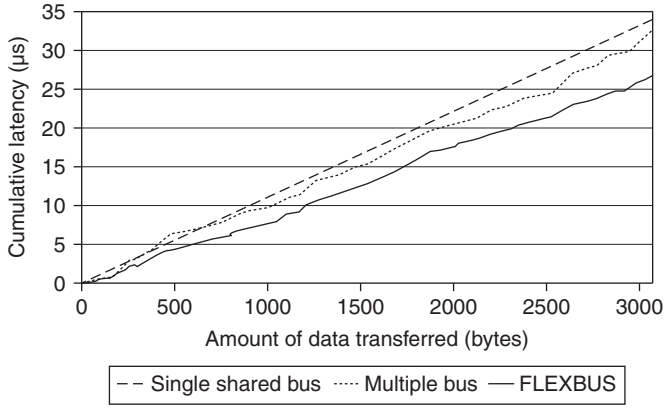

FIGURE 8.37

Cumulative frequency for different bus architecture, under synthetic traffic profiles [51]

Table 8.7 Performance of 802.11 MAC processor-based SoC subsystem for different communication architectures [51]

Bus architecture	Computation time (ns)	Data transfer time (ns)	Total time (ns)
Single shared bus	42,480	–	42,480
Multiple bus	26,905	12,800	39,705
FLEXBUS (bridge by-pass)	27,025	5,290	32,315
FLEXBUS (component re-mapping)	27,010	5,270	32,280
Ideally reconfigurable bus	26,905	5,120	32,025

taken to process a single frame of size 1kB, for different bus architectures. It can be seen that the times required by both variants of the FLEXBUS architecture are smaller compared to conventional architectures. FLEXBUS and its reconfiguration policies are also seen to perform close to the ideal case, which assumes no reconfiguration overhead, and an ideal reconfiguration policy having full knowledge of future bus traffic.

8.6 SUMMARY

In this chapter, we presented custom bus-based communication architectures that attempt to overcome limitations of commercially available standard bus-based communication architectures, and improve system design goals such as power consumption and performance. We first looked at *split/segmented bus architectures* that split a long interconnect into segments in order to reduce the wire delay, capacitive load and consequently power consumption, as well as increase parallelism during data communication. *Serial bus architectures* reduce the number of wires connecting components, compared to conventional parallel bus architectures. This reduces coupling capacitance between wires, which reduces signal propagation delay and also reduces power consumption because of the reduced capacitance. *CDMA-based bus architectures* allow multiple transmitters to send data on a shared medium simultaneously, which can reduce traffic conflicts and consequently communication latency. Asynchronous bus architectures, unlike conventional commercial bus architectures that are primarily synchronous, do not use a global clock signal for synchronization. Since the global clock consumes a significant amount of power, asynchronous bus architectures have lower power consumption. Finally, *dynamically reconfigurable bus architectures* allow the topology and/or protocol parameters such as arbitration schemes to dynamically change and adapt to changing traffic profiles and data characteristics, in order to improve performance.

FURTHER READING

Very early work in the area of custom bus architecture design resulted in a synchronous, high performance, split and burst transaction capable, pipelined, shared bus called the HiPi+ bus [55], which extended an even earlier work that proposed the high performance HiPi bus [56]. The concepts proposed in the work were a precursor to the enhancements that followed in several commercial on-chip bus architectures. Several works have proposed using custom circuit techniques [57–63] to improve the performance of hierarchical and crossbar bus-based conventional communication architectures such as AMBA AHB/AXI. A performance analysis of commonly used arbitration schemes was presented in [64], and custom variations on existing arbitration schemes, such as the direct mapped slot allocation TDMA [65], have been proposed to improve the performance of bus architectures. A custom wrapper-based bus (NECoBUS) [66] was proposed to reduce the latency of

wrappers in wrapper-based bus architectures. NECoBUS employs several latency reduction techniques such as retry encapsulation, write-buffer switching, early bus requests, and converter-based multiple bit-width connections to remove the latency penalty induced in the conventional wrapper-based bus design.

REFERENCES

[1] Y. Zhang, W. Ye and M. J. Irwin, "An alternative architecture for on-chip global interconnect: Segmented bus power modeling," in *Proceedings of Thirty-Second Asilomar Conference on Signals, Systems & Computers*, 1998, pp. 1062–1065.

[2] Y. Zhang, R.Y. Chen, W.YE and M. J. Irwin, "System Level Interconnect Modeling," *Proceedings of the International ASIC Conference*, September 1998, pp. 289–293.

[3] ARM AMBA Specification and Multi layer AHB Specification (rev2.0), http://www.arm.com, 2001.

[4] IBM CoreConnect Specification, http://www.ibm.com/chips/techlib/techlib.nsf/product families/CoreConnect_Bus_Architecture.

[5] "STBus Communication System: Concepts and Definitions," *Reference Guide*, STMicro Electronics, May 2003.

[6] Sonics SMART Interconnect, http://www.sonicsinc.com.

[7] WISHBONE specification, http://www.opencores.org/wishbone.

[8] Altera AVALON Interface Specification, April 2006, http://www.altera.com/.

[9] R. Cheng-Ta Hsieh and M. Pedram, "Architectural energy optimization by bus splitting,"in *Proceedings of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 21, No. 4, April 2002, pp. 408–414.

[10] R. Lu and C.-K. Koh, "A high performance bus communication architecture through bus splitting," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2004, pp. 751–755.

[11] J. Rabaey and M. Pedram, *Low Power Design Methodologies*, Kluwer Academic Publishers, Norwell, MA, 1996.

[12] E. Macii, M. Pedram and F. Somenzi, "High-level power modeling, estimation and optimization," *IEEE Transactions on Computer Aided Design*, Vol. 17, Nov. 1998, pp. 1061–1079.

[13] R. Lu and C.-K. Koh, "SAMBA-bus: A high performance bus architecture for system-on-chips," in *Proceedings of International Conference on Computer Aided Design, (ICCAD)*, 2003, pp. 8–12.

[14] ARM AMBA 3.0 AXI Specification www.arm.com/armtech/AXI.

[15] R. Ho, K. W. Mai and M. A. Horowitz, "The Future of Wires," in *Proceedings of the IEEE*, Vol. 89, April 2001.

[16] Semiconductor Industry Association, International Technology Roadmap for Semi-conductors, 2003.

[17] P. Saxena and C. Liu, "A postprocessing algorithm for crosstalk-driven wire perturbation," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, April 2000, pp. 691–702.

[18] L. Macchiarulo, E. Macii and M. Poncino, "Wire placement for crosstalk energy minimization in address buses," in *Proceedings of Design, Automation and Test in Europe (DATE)*, March 2002, pp. 158–162.

[19] J. Cong, L. He, C.-K. Koh and Z. Pan, "Interconnect Sizing and Spacing with Consideration of Coupling Capacitance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 20, No. 9, September 2001, pp. 1164–1169.

[20] E. Macii, M. Poncino and S. Salerno, "Combining wire swapping and spacing for low-power deep-submicron buses," in *Proceedings of the IEEE Great Lakes Symposium on VLSI (GLS-VLSI)*, April 2003, pp. 198–202.

[21] P. Gupta and A. Kahng, "Wire swizzling to reduce delay uncertainty due to capacitive coupling," in *Proceedings of the International Conference on VLSI Design (VLSID)*, January 2004.

[22] Y. Shin and T. Sakurai, "Coupling-driven bus design for low-power application-specific systems," in *Proceedings of Annual ACM/IEEE Design Automation Conference (DAC)*, 2001, pp. 750–753.

[23] A. B. Kahng et al., "Interconnect Tuning Strategies for High Performance ICs," in *Proceedings of Design, Automation and Test in Europe (DATE)*, 1998, pp. 471–478.

[24] K. Hirose and H. Yasuura, "A bus delay reduction technique considering crosstalk," in *Proceedings of Design, Automation and Test in Europe (DATE)*, 2000, pp. 441–445.

[25] M. Ghoneima, Y. Ismail, M. Khellah, J. Tschanz and V. De, "Serial-link bus: a low-power on-chip bus architecture," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2005, pp. 541–546.

[26] N. Hatta, N. Demus Barli, C. Iwama, L. Dinh Hung, D. Tashiro, S. Sakai and H. Tanaka, "Bus serialization for reducing power consumption," *IPSJ Transactions on Advanced Computing Systems*, Vol. 47, No. 3, 2006, pp. 49–57.

[27] W. Steinhoegl et al., "Scaling laws for the resistivity increase of sub-100 nm interconnects," *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, September 2003, pp. 27–30.

[28] M. Frank Chang et al., "RF/wireless interconnect for inter- and intra-chip communications," in *Proceedings of the IEEE*, Vol. 89, No. 4, 2001, pp. 456–466.

[29] J. S. Lee and L. E. Miller, *CDMA Systems Engineering Handbook*, Artech House Publish, 1998. ISBN: 0-89006-990-5.

[30] B.-C. C. Lai, P. Schaumont and I. Verbauwhede, "CT-bus: a heterogeneous CDMA/TDMA bus for future SOC," in *Proceedings of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, 2004, pp. 1868–1872.

[31] M. F. Chang, "CDMA/FDMA-interconnects for future ULSI communications," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2005, pp. 975–978.

[32] E.-G. Jung, B.-S. Choi and D.-I. Lee, "High performance asynchronous bus for SoC," in *Proceedings IEEE International Symposium on Circuits and Systems (ISCAS)*, 2003, pp. 505–508.

[33] W. J. Bainbridge and S. B. Furber, "Asynchronous macrocell interconnect using MARBLE," in *Proceedings of Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1998, pp. 122–132.

[34] W. J. Bainbridge, "Asynchronous system-on-chip interconnect," Ph.D. Thesis, University of Manchester, March 2000.

[35] E. Jung, J. Lee, S. Kwak, K. Jhang, J. Lee and D. Har, "High performance asynchronous on-chip bus with multiple issue and out-of-order/in-order completion," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI (GLS-VSLI)*, 2005, pp. 152–155.

[36] A. Lines, "Nexus: An asynchronous crossbar interconnect for synchronous system-on-chip designs," in *Proceedings of 11th Symposium on High Performance Interconnects*, 2003, pp. 2–9.

[37] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," Sixth MIT Conference on Advanced Research in VLSI, MIT Press, 1990.

[38] K. Sekar, K. Lahiri and S. Dey, "Configurable platforms with dynamic platform management: An efficient alternative to application-specific system-on-chips," in *Proceedings of 17th International Conference on VLSI Design (VLSID)*, 2004, pp. 307–315.

[39] K. Lahiri, A. Raghunathan, G. Lakshminarayana and S. Dey, "Communication architecture tuners: a methodology for the design of high-performance communication architectures for systems-on-chips," in *Proceedings of the Conference on Design Automation (DAC)*, 2000, pp. 513–518.

[40] K. Lahiri, A. Raghunathan, G. Lakshminarayana and S. Dey, "Design of high-performance system-on-chips using communication architecture tuners," *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 23, No. 5, May 2004, pp. 620–636.

[41] K. Lahiri, A. Raghunathan and S. Dey, "Fast performance analysis of bus-based system-on-chip communication architectures," in *Proceedings of International Conference Computer-Aided Design (ICCAD)*, November 1999, pp. 566–572.

[42] K. Lahiri, A. Raghunathan and G. Lakshminarayana, "LOTTERYBUS: A new high-performance communication architecture for system-on-chip designs," in *Proceedings of the Conference on Design Automation (DAC)*, 2001, pp. 15–20.

[43] K. Lahiri, A. Raghunathan and G. Lakshminarayana, "The LOTTERYBUS on-chip communication architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 14, No. 6, June 2006, pp. 596–608.

[44] K. Lahiri, A. Raghunathan and S. Dey, "Evaluation of the traffic performance characteristics of system-on-chip communication architectures," in *Proceedings of the International Conference VLSI Design (VLSID)*, January 2001, pp. 29–35.

[45] J. Buck, S. Ha, E. A. Lee and D. D. Masserchmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal on Computer Simulation, Special Issue on Simulation Software Management*, Vol. 4, April 1994, pp. 155–182.

[46] F. Balarin, M. Chiodo, H. Hsieh, A. Jureska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki and B. Tabbara, *Hardware–Software Co-Design of Embedded Systems: The POLIS Approach*, Kluwer Academic Publishers, Norwell, MA, 1997.

[47] Synopsys Inc., RTL Synthesis, Available http://www.synopsys.com/products/logic/.

[48] NEC Electronics, Cell Based IC CB-12 L/M/H Type Features/Basic Specifications. Available http://www.necel.com/cbic/en/cb12/cb12.html.

[49] Y. Zhang, "Architecture and performance comparison of a statistic-based lottery arbiter for shared bus on chip," in *Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC)*, 2005, pp. 1313–1316.

[50] N. Wang, M. A. Bayoumi, "Dynamic fraction control bus: new SOC on-chip communication architecture design," in *Proceedings of SOC Conference*, 2005, pp. 199–202.

[51] K. Sekar, K. Lahiri, A. Raghunathan and S. Dey, "FLEXBUS: A high-performance system-on-chip communication architecture with a dynamically configurable topology," in *Proceedings of Design Automation Conference (DAC)*, 2005, pp. 571–574.

[52] T. D. Richardson, C. Nicopoulos, D. Park, V. Narayanan, Y. Xie, C. Das and V. Degalahal, "A hybrid SoC interconnect with dynamic TDMA-based transaction-less buses and on-chip networks," *19th International Conference on VLSI Design (VLSID)*, 2006, pp. 8–15.

[53] "Synopsys DesignWare Intellectual Property," http://www.synopsys.com/products/designware/designware.html.

[54] "CB-12" http://www.necel.com/cbic/en/cb12/cb12.html.

[55] W.-J. Hahn, A. Ki, K.-W. Rim and S.-W. Kim, "A multiprocessor server with a new highly pipelined bus," in *Proceedings of IPPS*, 1996, pp. 512–517.

[56] A. Ki, W. Sim, B. Park and Y. Yoon, "Highly pipelined bus: Hipi-bus," *JTC-CSCC'91*, July 1991, pp. 528–533.

[57] J. P. Bissou, M. Dubois, Y. Savaria and G. Bois, "High-speed system bus for a SoC network processing platform," in *Proceedings of the 15th International Conference on Microelectronics (ICM)*, 2003, pp. 194–197.

[58] A. Landry, Y. Savaria and M. Nekili, "A Beyond-1 GHz AMBA High-Speed Bus for SoC DSP Platforms," *IEEE International Conference on Microelectronics (ICM)*, 2004, pp. 46–49.

[59] A. Landry, Y. Savaria and M. Nekili, "A novel 2 GHz multi-layer AMBA. High-speed bus interconnect matrix for SoC platforms," *IEEE International Symposium on Circuits and Systems, (ISCAS)*, 2005, pp. 3343–3346.

[60] A. Landry, Y. Savaria and M. Nekili, "Circuit techniques for a 2 GHz AMBA AHB bus," *The 3rd International IEEE-NEWCAS Conference*, 2005, pp. 311–314.

[61] M. Dubois, Y. Savaria and G. Bois, "A generic AHB bus for implementing high-speed locally synchronous islands," in *Proceedings of IEEE Southeast Conference*, 2005, pp. 11–16.

[62] S.-Y. Hwang and K.-S. Jhang, "An improved implementation method of AHB BusMatrix," in *Proceedings of IEEE International SOC Conference*, 2005, pp. 211–214.

[63] P. Wijetunga, "High-performance crossbar design for system-on-chip," *Proceedings of the 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*, 2003, pp. 138–143.

[64] F. Poletti, D. Bertozzi, B. Luca and A. Bogliolo, "Performance analysis of arbitration policies for SoC communication architectures," *Transactions on Design Automation for Embedded Systems*, Vol. 8, 2003, pp. 189–210.

[65] A. Olugbon, T. Arslan and I. Lindsay, "A formal approach to virtualisation and provisioning in AMBA AHB-based reconfigurable systems-on-chip," in *Proceedings of International Symposium on System-on-Chip*, 2005, pp. 175–178.

[66] K. Anjo, A. Okamura, T. Kajiwara, N. Mizushima, M. Omori and Y. Kuroda, "NECoBus: a high-end SOC bus with a portable and low-latency wrapper-based interface mechanism," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2002, pp. 315–318.

[67] Synopsys NanoSim, http://www.synopsys.com/products/mixedsignal/nanosim/nanosim.html.

[68] A. C. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proceedings of Symposium on Operating Systems Design and Implementation*, 1994, pp. 1–12.

CHAPTER

On-Chip Communication Architecture Refinement and Interface Synthesis

9

In a typical SoC design flow, several models of the system are created that capture different levels of detail, for different purposes. Figure 9.1 shows how communication architecture refinement and interface synthesis involve transformations between models with different levels of detail, in a typical design flow. *Functional* (or *task/process graph*) *level models* focus on capturing the functionality of the system, without any notion of hardware or software components that will ultimately implement the functionality. Such models are typically used as "golden reference" models to allow later stages of the design flow to check and validate the intended functionality of the system, as needed. *Architectural level models* on

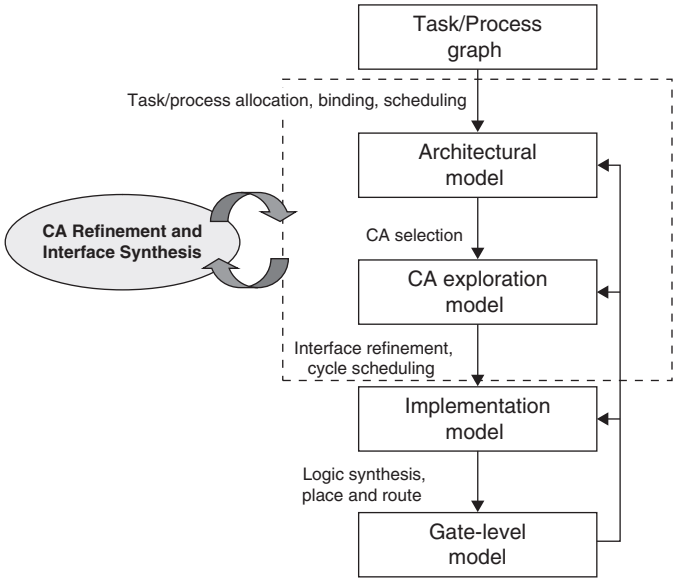

FIGURE 9.1

Communication architecture (CA) refinement and interface synthesis in a typical ESL design flow