2.1 TERMINOLOGY

We begin by reviewing the basic terminology used to describe bus-based communication architectures, and systems deploying these architectures. Figure 2.1 shows a simple SoC design in which several (computational) components are interconnected using a bus-based communication architecture. Components which initiate and control read and write data transfers are referred to as *masters*. The *Processor* and *DSP* (*digital signal processor*) components in Fig. 2.1 are examples of master components that read/write data from/to other components in the system. Every master component is connected to the bus using a set of signals which are collectively referred to as a *master port*. The components that simply respond to data transfer requests from masters (and cannot initiate transfers themselves) are referred to as *slaves*, and have corresponding *slave ports*. The three memory blocks in Fig. 2.1 are examples of slaves that can handle requests for data read and write from other components (e.g., Processor, DSP), but cannot initiate such transfers themselves. The component ports are actually a part of its *interface* with the bus. An interface can be simple, consisting merely of the set of connecting wires to the bus (i.e., master or slave ports). Or it could be more complex, consisting of buffers, frequency converters, etc. in order to improve communication performance.

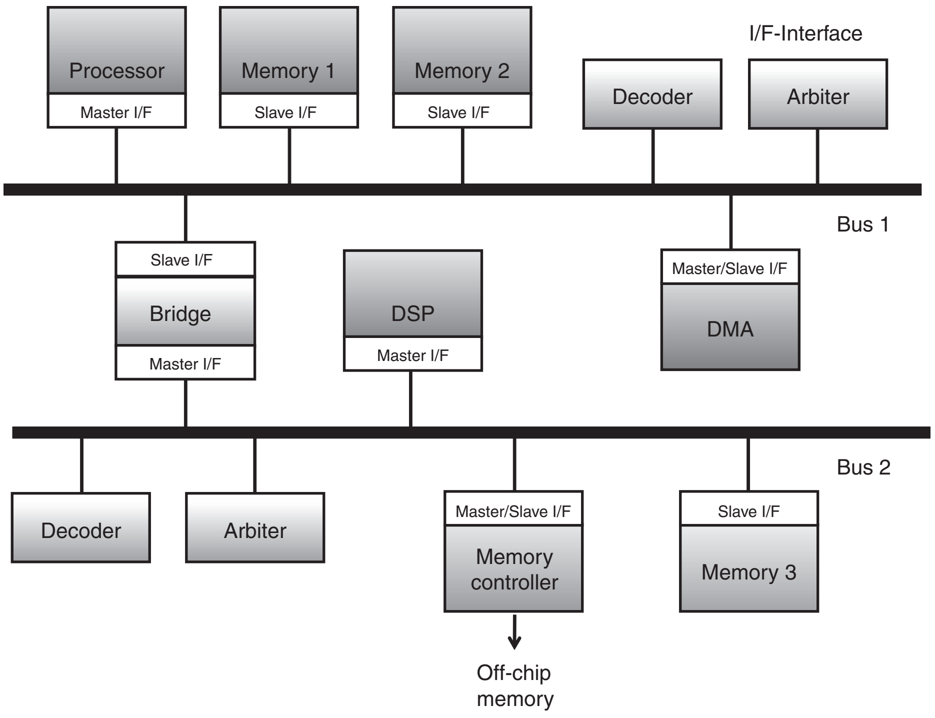

FIGURE 2.1

Example of an SoC with a bus-based communication architecture

Some components can have both master and slave ports, which means that they can act as both masters and slaves. These components are *master/slave hybrid* components. For instance, the *DMA* (*direct memory access*) component in Fig. 2.1 has a slave port that allows the *Processor* to write into (and read from) the *DMA* configuration register file, in order to initialize and configure it. Once configured, the *DMA* component uses its master port to initiate and control data transfers between memory blocks (which would otherwise have been managed by the *Processor*; as a result the *Processor* is freed up to perform other activity which typically improves system performance). Similarly, the *Memory Controller* component has a slave port which is used by the DSP component to initialize and configure its functionality. Once configured, the Memory Controller can initiate and control data transfers with external memory components connected to it, using its master port.

In addition to the wires, a bus-based communication architecture also consists of logic components such as *decoders*, *arbiters*, and *bridges*. A *decoder* is a logic component that decodes the destination address of a data transfer initiated by a master, and selects the appropriate slave to receive the data. It can either be a separate logic component, or integrated into a component interface. An *arbiter* is a logic component that determines which master to grant access to the bus, if multiple masters access the bus simultaneously. Typically, some form of a priority scheme is used, to ensure that critical data transfers in the system are not delayed. Finally, a *bridge* is a logic component that is used to connect two buses. It can have a fairly simple implementation if it connects two buses with the same protocols and clock frequencies. However, if the two buses have different protocols or clock frequencies, some form of protocol or frequency conversion is required in the bridge, which adds to its complexity. A bridge connects to a bus using a master or a slave port, just like any other component. The type of port used to connect to a bus depends on the direction of data transfers passing through it. For instance, in the example shown in Fig. 2.1, the *DMA* and *Processor* components on *Bus 1* initiate and control data transfers to *Bus 2* by sending data to the slave port of the bridge on *Bus 1*, which transfers it to its master port on *Bus 2* and sends the data to its destination. Since the *DSP* and *Memory Controller* do not initiate and control data transfers to components on *Bus 1*, a single bridge is sufficient as shown in Fig. 2.1. However, if these components needed to transfer data to *Bus 1*, another bridge with a slave port on *Bus 2* and a master port on *Bus 1* would be required.

2.2 CHARACTERISTICS OF BUS-BASED COMMUNICATION ARCHITECTURES

Bus-based communication architectures are defined by various architectural and physical characteristics that can have many different implementations. These implementation choices have trade-offs that can significantly affect the power, performance, and occupied area of the communication architecture. In this