CHAPTER 19

CONFIGURATION BITSTREAM GENERATION

Steven A. Guccione
Cmpware, Inc.

While a reconfigurable logic device shares some of the characteristics of a fixed hardware device and some of a programmable instruction set processor, the details of the underlying architecture and how it is programmed are what distinguish these machines. Both a reconfigurable logic device and an instruction set processor are programmable by "software," but the internal organization and use of this software are quite different. In an instruction set processor, the programming is a set of binary codes that are incrementally fed into the device during operation. These codes actually carry out a form of reconfiguration inside the processor. The arithmetic and logic unit(s) (ALU) is configured to perform a requested function and various control multiplexers (MUXes) that control the internal flow of data are set. In the instruction set machine, these hardware components are relatively small and fixed and the system is reconfigured on a cycle-by-cycle basis. The processor itself changes its internal logic and routing on every cycle based on the input of these binary codes.

In a processor, the binary codes—the processor's machine language—are fairly rigid and correspond to sequential "instructions." The sequence of these instructions to implement a program is often generated by some higher-level automatic tool such as a high-level language (HLL) compiler from a language such as Java, C, or C++. But they may, in reality, come from any source. What is important is that the collection of binary data fits this rigid format. The collection of binary data goes by many names, most typically an "executable" file or even more generally a "binary program."

A reconfigurable logic device, or field-programmable gate array (FPGA), is based on a very different structure than that of an instruction set machine. It is composed of a two-dimensional array of programmable logic elements joined together by some programmable interconnection network. The most significant difference between FPGA and the instruction set architecture is that the FPGA is typically intended to be programmed as a complete unit, with the various internal components acting together in parallel. While the structure of its binary programming (or configuration) data is every bit as rigid as that of an instruction set processor, the data are used spatially rather than sequentially.

In other words, the binary data used to program the reconfigurable logic device are loaded into the device's internal units before the device is placed in its operating mode, and typically, no changes are made to the data while the device is operating. There are some significant exceptions to this rule: The configuration data may in fact be changed while a device is operational, but this is somewhat akin to "self-modifying code" in instruction set architectures. This is a very powerful technique, but carries with it significant challenges.

The collection of binary data used to program the reconfigurable logic device is most commonly referred to as a "bitstream," although this is somewhat misleading because the data are no more bit oriented than that of an instruction set processor and there is generally no "streaming." While in an instruction set processor the configuration data are in fact continuously streamed into the internal units, they are typically loaded into the reconfigurable logic device only once during an initial setup phase. For historical reasons, the somewhat undescriptive "bitstream" has become the standard term.

As much as the binary instruction set interface describes and defines the architecture and functionality of the instruction set machine, the structure of the reconfigurable logic configuration data bitstream defines the architecture and functionality of the FPGA. Its format, however, currently suffers from a somewhat interesting handicap. While the format of the programming data of instruction set architectures is freely published, this is almost never the case with reconfigurable logic devices. Almost all of them that are sold by major manufacturers are based on a "closed" bitstream architecture.

The underlying structure of the data in the configuration bitstream is regarded by these companies as a trade secret for reasons that are historical and not entirely clear. In the early days of reconfigurable logic devices, the underlying architecture was also a trade secret, so publishing the configuration bitstream format would have given too many clues about it. It is presumed that this was to keep competitors from taking ideas about an architecture, or perhaps even "cloning" it and providing a hardware-compatible device. It also may have reassured nervous FPGA users that, if the bitstream format was a secret, then presumably their logic designs would be difficult to reverse-engineer.

While theft and cloning of device hardware do not appear to be a potential problem today, bitstream formats are still, perhaps out of habit alone, treated as trade secrets by the major manufacturers. This is a shame because it prohibits interesting experimentation with new tools and techniques by third parties. But this is perhaps only of interest to a very small number of people. The vast majority of users of commercial reconfigurable logic devices are happy to use the vendor-supplied tools and have little or no interest in the device's internal structure as long as the logic design functions as specified. However, for those interested in the architecture of reconfigurable logic devices, trade secrecy is an important subject.

While exact examples from popular industry devices are not possible because of this secrecy, much is publicly known about the underlying architectures, the general way a bitstream is generated, and how it operates when loaded into a device.

19.1 THE BITSTREAM

The bitstream spatially represents the configuration data of a large collection of small, relatively simple hardware components. Thus, we can identify these components and discuss the ways in which the bitstream is used to produce a working digital circuit in a reconfigurable logic device. Although there is really no limit to the types of units possible in a reconfigurable logic device, two basic structures make up the microarchitecture of most modern FPGAs. These are the lookup table (LUT) and the switch box.

The LUT is essentially a very small memory element, typically with 16 bits of bit-oriented storage. Some early FPGAs used smaller 8-bit LUTs, and other more exotic architectures used non-LUT structures. In general, however, the vast majority of commercial FPGA devices sold over the last decade use the 16-bit LUT as a primary logic building block.

The functionality of LUTs is very simple. Binary data are loaded into them to produce some Boolean function. In the case of the 16-bit LUT, there are four inputs, which can produce any arbitrary 4-input Boolean logic function. For instance, to provide the AND function of all four inputs, each bit in the memory except the bit at address `A(1,1,1,1)` is loaded with a binary `0` and the `A(1,1,1,1)` bit is loaded with a `1`. The address inputs of the LUT are used as the inputs to the logic function, with the output of the LUT providing the output of the logic function. Figure 19.1 illustrates this mapping of a 2-input LUT to a 2-input AND gate.

While the LUTs provide the logic for the circuit, the switch boxes provide the interconnection. These switch boxes are typically made up of multiplexers in various regular configurations. These multiplexers are controlled by bits of memory that select the inputs and send them to the multiplexer's outputs. Figure 19.2 shows a typical configurable interconnect element constructed using a multiplexer.

The multiplexer inputs in Figure 19.2 are controlled by two memory elements that are set during configuration. They select which input value is sent to the output. By connectiong large numbers of elements of this type, an interconnection

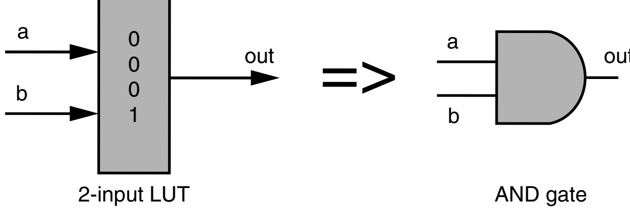

FIGURE 19.1 ■ A 2-input LUT configured as an AND gate.

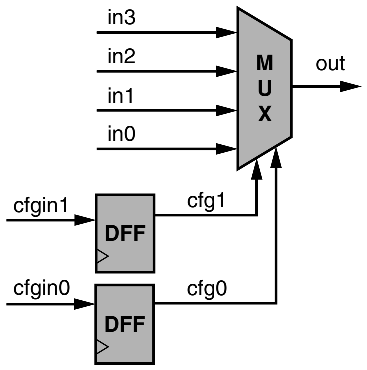

FIGURE 19.2 ■ A configurable 4-input multiplexer used in routing.

network of the kind typically used to construct modern reconfigurable logic devices can be made.

In various topologies, the ouputs of the multiplexers in the switch boxes feed the address inputs of the LUTs; the outptus of the LUTs, in turn, feed the inputs of the switch box multiplexers. This provides a basic reprogrammable architecture capable of producing arbitrary logic functions, as well as the ability to interconnect these functions in a variety of ways. How complex a circuit a given reconfigurable logic device can implement is based on both the number of LUTs and the size and complexity of the interconnection fabric.

In fact, the topology of the interconnect fabric and the implementation of the switch boxes is perhaps the defining characteristic of an FPGA architecture. Older FPGAs had a limited silicon area and few metal layers to supply wires. For this reason, the LUTs were typically "islands" of logic, with the interconnect wires running in the "channels" between them. Where these channels intersected were the switch boxes. How many wires to use and how to configure the switch boxes were the main work of the FPGA architect. Balancing the cost of more wires with the needs of typical digital circuit was important to making a cost-effective device that would be commercially successful. Covering as many potential circuit designs as possible at as high a speed as possible, but with the smallest silicon area, is still the challenge FPGA device architects must confront.

In later silicon process generations, however, more metal layers were available, which resulted in a much higher ratio of wires to logic in FPGAs. Where older generations of FPGAs often had a scarcity of interconnection resources, more modern FPGA devices seldom encounter circuits they are unable to implement because of a lack of routing resources. And these wires now tend to run on top of the logic rather than in channels, which has led to higher circuit densities, a tighter integration between the switch boxes and the logic, and faster interconnect.

The configuration bitstream data for the routing are essentially the multiplexer inputs in these switch boxes. The memory for these MUX inputs tends to be individual memory elements such as flip-flops scattered around the device as needed, establishing the basic bitstream for the FPGA: the LUT data plus the bits to control the routing multiplexers.

While the multiplexer and switch boxes are the basic elements of modern FPGA devices, many other components are possible. One of the more popular is a configurable input/output block, or IOB. An IOB is typically connected to the end of one of the wires in the routing system on one side and to a physical device pin on the other. It is then configured to define the type of pin used by this device: either input or output. More complex IOBs can configure pin voltages and even parameters such as capacitance, and some even provide higher-level support for various serial communication protocols. Much like switch boxes, the configuration bitstream data for the IOBs are some collection of bits used to set flip-flops within them to select these features.

In addition to IOBs, other, more special-purpose units have turned up in later generations of FPGA devices. Two prominent examples are block memory and multiplier units. Block memory (BlockRAM) is simply relatively large RAM units that are usually on the order of 1K bits but can be implemented in any number of ways. The actual data bits may be part of the bitstream, which initializes the BlockRAM upon power-up. To reduce the size of the bitstream, however, this data may be absent and internal circuitry may be required to reset and initialize the BlockRAM.

In addition to the internal data, the BlockRAM is typically interfaced to the switch boxes in various ways. Its location and interfacing to the interconnection network is a major architectural decision in modern reconfigurable logic device design.

Because the multiplication function has become more popular in FPGA designs and because FPGAs are so inefficient at implementing such circuits, the addition of hardwired multiplier units into modern FPGA devices has been increasing. These units typically have no internal state or configuration, but are interfaced to the interconnection network in a manner similar to the BlockRAM interface. As with the BlockRAM, where to locate these resources and how many to include are major architectural decisions that can have a large impact on the size and efficiency of modern FPGAs.

Many other features also find control bits in the FPGA bitstream. Some of these are global control related to configuration and reconfiguration; others are ID codes and error-checking information such as cyclic redundancy check codes. How these features are implented is very architecture dependent and can vary widely from device family to device family. One common feature is basic control for bit-level storage elements, often in the form of flip-flops on the LUT output. Various control bits often set circuit parameters such as the flip-flop type (D, JK, T) or the clock edge trigger type (rising or falling edge). The ability to chage the flip-flop into a transparent D-type latch is also a popular option. Each of these bits also contributes to the configuration data, with one set of flip-flop configuration settings per LUT being typical.

Finally, while the items just discussed are the major standard units used to construct modern FPGA devices and define the configuration bitstream, there

TABLE 19.1 ■ Configuration bitstream sizes

Year	Device	Bits
1986	XC2018	18 Kbits
1988	XC3090	64 Kbits
1990	XC4013	248 Kbits
1994	XC4025	422 Kbits
1996	XC4028	668 Kbits
1998	XCV1000	6.1 Mbits
2000	XCV3200	16 Mbits
2003	XC2V8000	29 Mbits

is no limit to the types of circuits and configurations possible. For example, an interest in analog FPGAs has resulted in unique architectures to perform analog signal processing. Also, some coarser-grained reconfigurable logic devices have moved up in granularity from LUTs to ALUs, and these devices have somewhat different bitstream structures. Other architectures have gone in the other direction toward extremely fine-grained architectures. One notable device, the Xilinx XC6200, has a logic cell that is essentially a 2-input multiplexer. The balance of routing and logic in these devices has made them less attractive than coarser-grained devices, but they have not been reevaluated in the context of the denser routing available with newer multilayer metal processes and so may yet have some promise.

As FPGA devices themselves have grown, so has the size of the configuration bitstreams. In fact, bitstream size can be a reasonable gauge of the size and complexity of the underlying device, which can be useful because it is a single number that is readily available. Table 19.1 gives some representative sizes of various bitstreams from members of the Xilinx family of FPGAs and the approximate dates they were introduced.

19.2 DOWNLOADING MECHANISMS

The FPGA configuration bitstream is typically saved externally in a nonvolatile memory such as an EPROM. The data are usually loaded into the device shortly after the initial power-up sequence, most often bit-serially. (This loading mechanism may be the reason that many engineers perceive the configuration data as a "stream of bits.") The reason for serial loading is primarily one of cost and convenience. Since there is usually no particular hurry in loading the FPGA configuration data on power-up, using a single physical device pin for this data is the simplest, cheapest approach. Once the data are fully loaded, this pin may even be put into service as a standard I/O pin, thus preventing the configuration downloading mechanism from consuming valuable I/O resources on the device.

A serial configuration download is the norm, but some FPGA devices have a parallel download mode that typically permits the use of eight I/O pins to download configuration data in parallel. This may be helpful for designs that use an 8-bit memory device and for applications where reprogramming is common and speed is important—often the case when an FPGA is controlled by a host processor in a coprocessor arrangement. As with the serial approach, the pins may be returned to regular I/O duty once downloading is complete.

One place where such high-bandwidth configuration is useful is in the device test in the factory. Testing FPGA devices after manufacture can be a very expensive task, mostly because of time spent attached to the test equipment. Thus, decreasing the configuration download time by a factor of eight may result in the FPGA manufacturer requiring substantially fewer pieces of test equipment, which can result in a significant cost savings during manufacture. Anecdotal evidence suggests that high-speed download is driven mostly by increased test efficiency and not by any customer requirements related to runtime reconfiguration.

One type of device that is based on nonvolatile memory bears mention here. Rather than using RAM and flip-flops as the internal logic and control, commercially available devices from companies such as Actel use nonvolatile Flash-style internal configuration memory. These devices are programmed once and do not require reloading of configuration data on power-up, which can be important in systems that must be powered-up quickly. Such devices also tend to be more resistant to soft errors that can occur in volatile RAM devices. This makes them especially popular in harsh environments such as space and military applications.

19.3 SOFTWARE TO GENERATE CONFIGURATION DATA

The software used to generate configuration bitstream data for FPGA devices is perhaps some of the most complex available. It usually consists of many layers of functionality and can run on the largest workstations for hours or even days to produce the output for a single design. While the details of this software are beyond the scope of this chapter, some of the way the software generates this bitstream will be briefly discussed in this section.

The top-level input to the FPGA design software is most often a hardware description language (HDL) or a graphical circuit design created with a schematic capture package. This representation is usually then translated into a low-level description more closely related to the implementation technology. A common choice for this intermediate format is EDIF (Electronic Design Interchange Format). This translation is fairly generic and such tools are widely available from a variety of software vendors.

The EDIF description is still not suitable for directly programming the reconfigurable logic device. In the typical FPGA, the underlying circuit must be “mapped” onto the array of LUTs and switch boxes. While the actual implementation may vary, the two basic processes for getting such abstract circuit descriptions into a physical representation of FPGA configuration data are placement/routing and mapping. Figure 19.3 shows the basic flow of this process.

Mapping refers to taking general logic descriptions and converting them into the bits used to fill in a LUT. This is sometimes referred to as “packing,” because

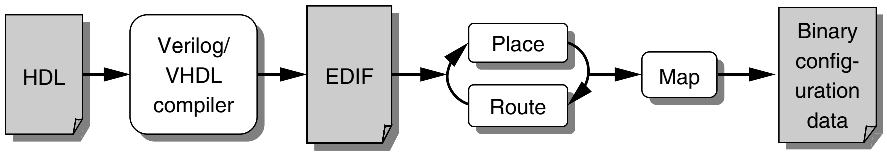

FIGURE 19.3 ■ The tool flow for producing the configuration bitstream.

several small logic gates are often "packed" into a single LUT. There is also a notion of placement that decides which LUT should receive the data, but this may also be considered a part of the mapping process.

Once the values for the LUTs have been decided, software can begin to decide how to interconnect the LUTs in a process called "routing." There are many algorithms of varying sophistication to perform routing, and factors such as circuit timing may be taken into account in the process. The result of the routing procedure is eventually used to supply the configuration data for the switch boxes.

Of course, this description is highly simplified, and mapping and routing can take place in various interleaved phases and can be optimized in a wide variety of ways. Still, this is the essential process used to produce the configuration bitstream. Finally, data for configuring the IOBs are typically input in some form that is aware of the particular package being used for the FPGA device. Once all of this data have been defined and collected, they can be written out to a single file containing the configuration bitstream.

As mentioned, FPGA configuration bitstream formats have almost always been proprietary. For this reason, the only tools available to perform bitstream generation tasks have been those supplied by the device manufacturer. The one notable exception is the Xilinx XC6200, which had an "open" bitstream. One of the XC6200's software tools was an application program interface (API) that permitted users to create configuration data or to even directly alter the configuration of an XC6200 in operation mode. Some of this technology was transferred to more mainstream Xilinx FPGAs and is available from Xilinx as a toolkit called JBits.

JBits is a Java API into the configuration bitstream for the XC4000 and Virtex device families. With JBits, the actual values on LUTs and switch box settings, as well as all other microarchitectural components, could be directly programmed. While the control data could be used to produce a traditional bitstream file, they could also be accessed directly and changed dynamically. The JBits API not only permitted dynamic reconfiguration of the FPGA but also permitted third-party tools to be built for these devices for the first time. JBits was very popular with researchers and users with exotic design requirements, but it never achieved popular use as a mainstream tool, although many of its related toolkit components, including the debug tool and partial reconfiguration support, have found their way into more mainstream software.

19.4 SUMMARY

While the generation of bitstream data to configure an FPGA device is a very common activity, there has been very little information available on the details of either the configuration bitstream or the underlying FPGA architecture. Thus, the FPGA can best be viewed as a collection of microarchitecture components, chiefly LUTs and switch boxes. These components are configured by writing data to the LUT values and to control memories associated with the switch boxes. Setting these bits to various values results in custom digital circuits.

A variety of tools and techniques are used to program reconfigurable logic devices, but all must eventually produce the relatively small configuration "bitstream" data the devices require. This data is in as rigid a format as any binary execution data for a microprocessor, but this format is typically proprietary and unpublished. While direct examination of actual commercial bitstream data is largely impossible, the general structure and the microarchitecture components configured by this data can be examined, at least in the abstract.

References

[1] Xilinx, Inc. *Virtex Data Sheet*, Xilinx, Inc., 1998.

[2] S. A. Guccione, D. Levi, P. Sundararajan. JBits: A Java-based interface for reconfigurable computing. *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, Laurel, MD, September 1999.

[3] E. Lechner, S. A. Guccione. The Java environment for reconfigurable computing. *Proceedings of the Seventh International Workshop on Field-Programmable Logic and Applications*, September 1997.

[4] Xilinx, Inc. *XAPP151: Virtex Series Configuration Architecture User Guide (version 1.7)*, (*http://direct.xilinx.com/bvdocs/appnotes/xapp151.pdf*), October 20, 2004.

[5] P. Alfke. *FPGA Configuration Guidelines (version 1.1)* (*http://direct.xilinx.com/bvdocs/appnotes/xapp090.pdf*), November 24, 1997.

[6] Xilinx, Inc. *XC6200 Field-Programmable Gate Arrays*, Xilinx, Inc., 1997.

[7] V. Betz, J. Rose. VPR: A new packing, placement, and routing tool for FPGA research. *Proceedings of the Seventh International Workshop on Field-Programmable Logic and Applications*, September 1997.

[8] Xilinx, Inc. *JBits 2.8 SDK for Virtex*, Xilinx Inc., 1999.