Module Generators for Xilinx Field Programmable Gate Arrays

Steven H. Kelem
Sr. Software Engineer

Steven K. Knapp
New Product Development Manager

Jorge Seidel
Sr. Software Engineer

Xilinx, Inc.
2100 Logic Drive
San Jose, California 95124

ABSTRACT

A new, more powerful set of module generators produce optimized implementations of particular logic functions for Xilinx XC4000 Field Programmable Gate Arrays (FPGAs). Module generators allow a designer to spend more time actually designing and less time worrying about device-specific implementation details. Module generators are particularly useful when designing with field programmable gate arrays because of their unique architectures and their ability to implement complex functions. The features and capabilities of a module generator for Xilinx XC4000 FPGAs is described in this paper.

INTRODUCTION

FPGAs are quickly pushing usable densities to 10,000 gates and more. Existing gate-level tools become ineffective for many applications at this density. New types of design tools are required to help designers manage ever-growing design complexity and to abstract functions to a higher level. Why should counters be described as a group and ANDs, ORs, and flip-flops when designers think of a counter as a complete logic function with certain attributes?

Module generators offer many benefits to FPGA or ASIC designers. They abstract the design process. Designers build applications more naturally out of functional descriptions instead of gate descriptions. Also, module generators handle multi-bit operations and full data paths which are common in large designs. Module generators help the designer become more productive and creative. By automating the actual implementation (the major portion of a design), module generators:

- Help reduce design cycles since major portions of the design are automated.
- Allow the designer to easily explore more alternate approaches to building the application.
- Aid the designer by providing expert knowledge of how to build a particular function for the target technology. This not only benefits the veteran FPGA designer but is especially valuable for the novice user.

The last point is particularly important. Ideally, a designer using FPGAs should understand the underlying architecture. This allows the user to exploit the capabilities of the device fully. However, with a vast multitude of device architectures available on the market, it is difficult to be an expert in all of them.

New architectures also require new design techniques. Techniques borrowed from 7400-series TTL do no fully exploit FPGA or ASIC architectures. For example, cascading multiple 74-163 four-bit binary counters is not the *only* way to build a large modulo clock divider (contrary to popular belief among many digital designers). An alternate approach with a linear-feedback shift register (psuedo-random counter), is much more effective in most FPGA and ASIC devices. The appropriate implementation technique is automatically selected in module generators.

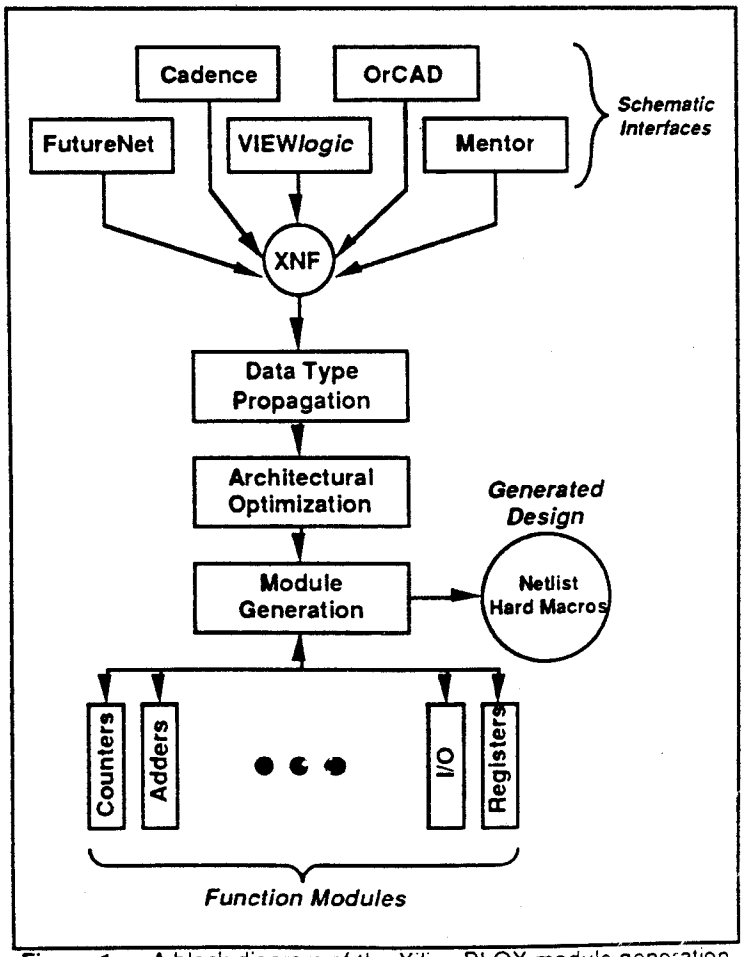

Figure 1. A block diagram of the Xilinx BLOX module generation system showing the significant elements.

THE XILINX BLOX MODULE GENERATION SYSTEM

Xilinx recognized the need for a tool that provides users optimal logic solutions without burdening the designer with all the device-specific information. The Xilinx BLOX module generation system shown in Figure 1, was created as a tool to aid designers using Xilinx XC4000 Family FPGAs.

The Xilinx BLOX system provides the following capabilities for the user:

- Designs are described at the block diagram level with parameterized modules. The modules can provide thousands of logic implementations for various popular logic functions. Gate-level descriptions are not required. However, a designer can inter-mix function modules and gate-level primitives in the same design if desired.
- Designer productivity is greatly increased since designs are done from the functional level, not the gate level.
- A designer need only specify the width and type of a bus once, anywhere along the data path. The widths and types of data carried on a data path are automatically propagated throughout the design and through levels of hierarchy. The size of an entire design can be modified by changing just a few fields on the schematic.
- Interfaces are provided to common schematic capture environments like Mentor Graphics®, VIEWlogic®, Cadence®, Data I/O®-FutureNet®, and OrCAD®. Therefore, there is no need for the designer to learn a new tool or new design process.
- The module generator custom tailors the logic implementation to the specific needs of each module. The implementation of a comparator, for example, will depend on the size of the data feeding the comparator and whether the equality, greater-than, or less-than outputs (or a combination) are used.
- Device-specific features are automatically used when applicable. For example, clocking or high fan-out signals are automatically assigned to special high-drive buffers.
- The system uses Xilinx-specific optimization techniques to boost the performance and density of logic functions. It incorporates expert knowledge of various implementation alternatives.

Design Flow. Using the Xilinx BLOX system, a designer enters the design using his or her favorite schematic editor and draws a simple block diagram for a majority of the design. Functions without a support module can be implemented using gate-level primitives or other macro library functions. Schematic entry is a natural means to describe such high-level functionality.

The design is converted into a standard, hierarchical Xilinx Netlist File (XNF) file and processed using the Xilinx BLOX software.

Xilinx BLOX performs the following operations on the design:

- The widths and types for all data paths are propagated throughout the design.
- The design is architecturally optimized to take advantage of special features available on XC4000 FPGAs. These optimizations include:
  - ◊ Assigning clock and high fan-out signals to the eight high-drive, low-skew buffers available on the chip.
  - ◊ Assigning a master reset signal to the global set/reset function, if applicable.
  - ◊ Remapping any arithmetic functions (adders, binary counters, accumulators) so that they can use the special fast-carry logic on the XC4000 device. The fast-carry logic is especially efficient for wide arithmetic functions.
  - ◊ Moving registers and flip-flops from the core array of logic blocks out to the I/O blocks where applicable. This helps provide maximum density for a design.
- The logic modules are expanded into their full implementation based on how they are used in the design and on the parameters attached to individual modules. Most modules are expanded into a netlist description of the function. Arithmetic functions, however, are expanded into hard macros which contain logic block partitioning and relative placement and routing information. This guarantees improved performance.
- After the expanded modules are merged together, the entire design is written as an expanded, generated netlist.

Available Logic Modules. Currently, the Xilinx BLOX system provides 30 different parameterized module generators including schematic symbols for several popular schematic editors. These 30 modules provide a designer with literally thousands of possible logic functions. The list of modules is shown in Table 1.

Table 1. List of Xilinx BLOX Modules.

Adder/Substracters	Accumulators
Data Registers	Shift Registers
SRAMs	PROMs
Comparators	Inputs
Outputs	Bus Interfaces
Bidirectional I/Os	Counters
Multiplexers	Bus Inversions
Force a value onto a bus	Bus-Wide Boolean Functions
Three-State Buffers	

The bus-wide Boolean functions offer different ways to perform an AND, OR, or XOR operation on the bits of a bus in the following manners:

- AND/OR/XOR all of the bits of the bus together and provide a single-bit output. For example, XOR all the elements of a bus to build an even parity generator.
- AND/OR/XOR bits from one bus with corresponding bits from a second bus and provide a bus-wide output.
- AND/OR/XOR each bit of a bus with a single-bit input and provide a bus-wide output. The single-bit input is usually a control signal like an enable.

Parameterizing a Module. The designer may attach various parameters for a specific module. This capability allows a designer to build a huge number of different implementations for a particular function. For example, the counter module (COUNTER) shown in Figure 2 can have the parameters listed in Table 2 to control the set and reset functions plus the counter's counting sequence.

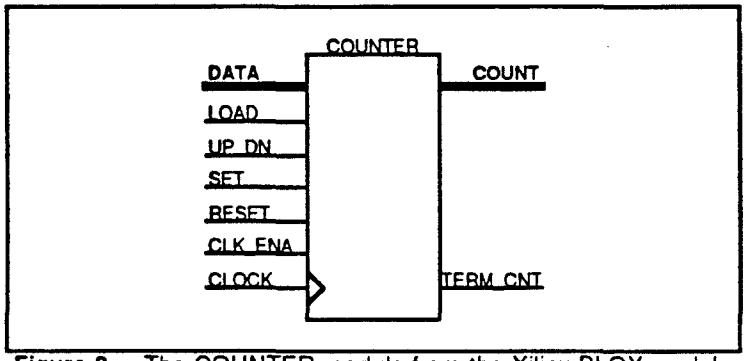

Figure 2. The COUNTER module from the Xilinx BLOX module library.

Table 2. Parameters Available on COUNTER Module.

Parameter	Value
SET	None if not connected Asynchronous Synchronous
RESET	None if not connected Asynchronous Synchronous
SEQUENCE	BINARY (default) JOHNSON (circular counters) LFSR (linear feedback shift register)

Considering just the counter widths between 1 and 32 bits, this allows 575 unique implementations! This does not include the other variations provided by preloading, clock enabling, and up/down control.

Designer Productivity. The module generators also boost the designer's productivity. The amount of time required to enter a design is significantly reduced since the engineer enters the design at the block diagram level. For example, entering a 32-bit bidirectional data bus is quite tedious if drawn at the gate level. This simple example requires 129 symbols, 160 wire connections, and 130 labels. Using the BUS_IO module generator, the same design requires only one symbol, two wire connections, two labels, and one parameter describing the bounds of the bus. Likewise, the schematic is significantly easier to view since the entire bus interface fits in a small portion of an 'A' size drawing. The same cannot be said of the gate-level drawing.

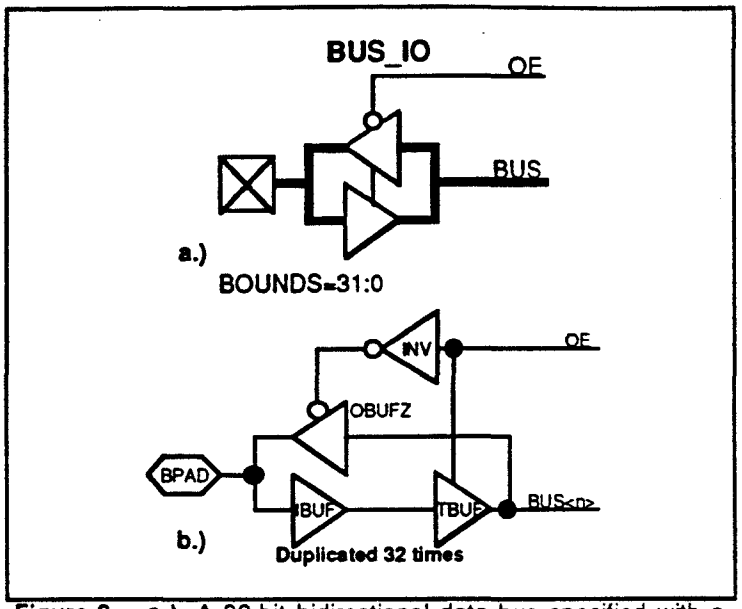

Figure 3. ***a.)*** A 32-bit bidirectional data bus specified with a single module symbol is much easier than drawing 32 gate-level representations as shown in ***b.)***.

The data propagation feature is extremely useful, especially if the size of the data path changes. Instead of having to add or remove logic on the schematic, the designer need only change the width of the data bus via a parameter on any of the I/O symbols. The Xilinx BLOX software automatically propagates that information throughout the design.

Module generators help implement the actual design. The designer need not worry how best to implement a given function. He or she merely lets the Xilinx BLOX software optimize and expand the design based on how the module is used. For example, the XC4000 FPGA device has specific logic designed to increase the performance and density of arithmetic functions like adders and counters. To use this special logic, a designer previously used pre-defined hard macro implementations of these functions with fixed data path widths. With Xilinx BLOX, however, the designer can create custom adders and counters of various sizes automatically. The Xilinx BLOX software will create the hard macros and also incorporate additional logic if there is extra space in the hard macro.

Furthermore, the architectural optimizations squeeze the maximum density and performance from a design. In many cases, a designer may be unaware of special features or design techniques for the device. The Xilinx BLOX software automatically transforms the logic to use these features. For example, Xilinx XC4000 devices have flip-flops in the input/output blocks. Instead of wasting the flip-flops in the internal core, the Xilinx BLOX software will move data registers and flip-flops into the I/O blocks when applicable.

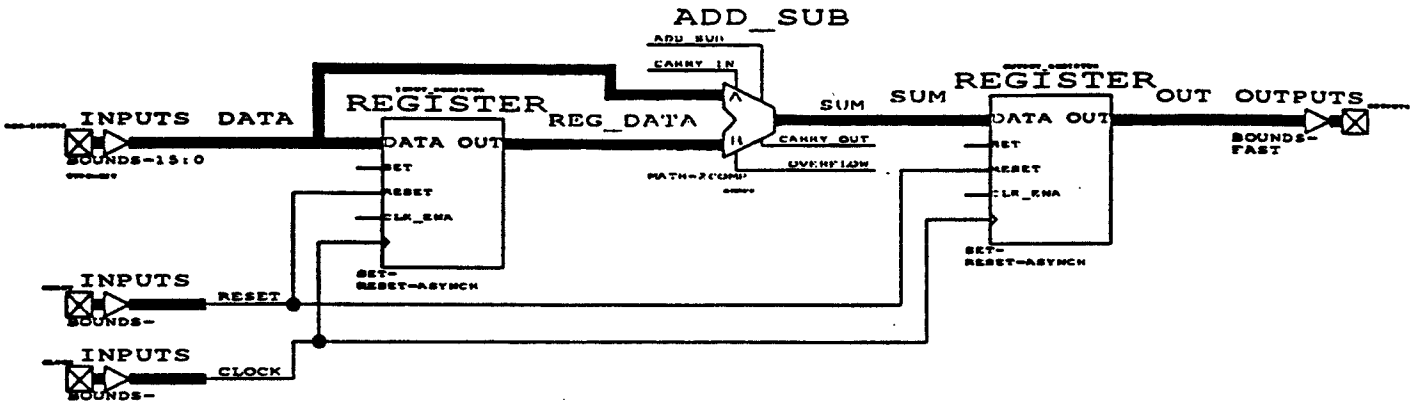

Figure 4. A complete design for a 16-bit registered adder drawn in VIEW*logic* with Xilinx BLOX modules. The width and types for all data paths is automatically propagated throughout the design. The logic functions are generated and optimized for Xilinx XC4000 Field Programmable Gate Arrays.

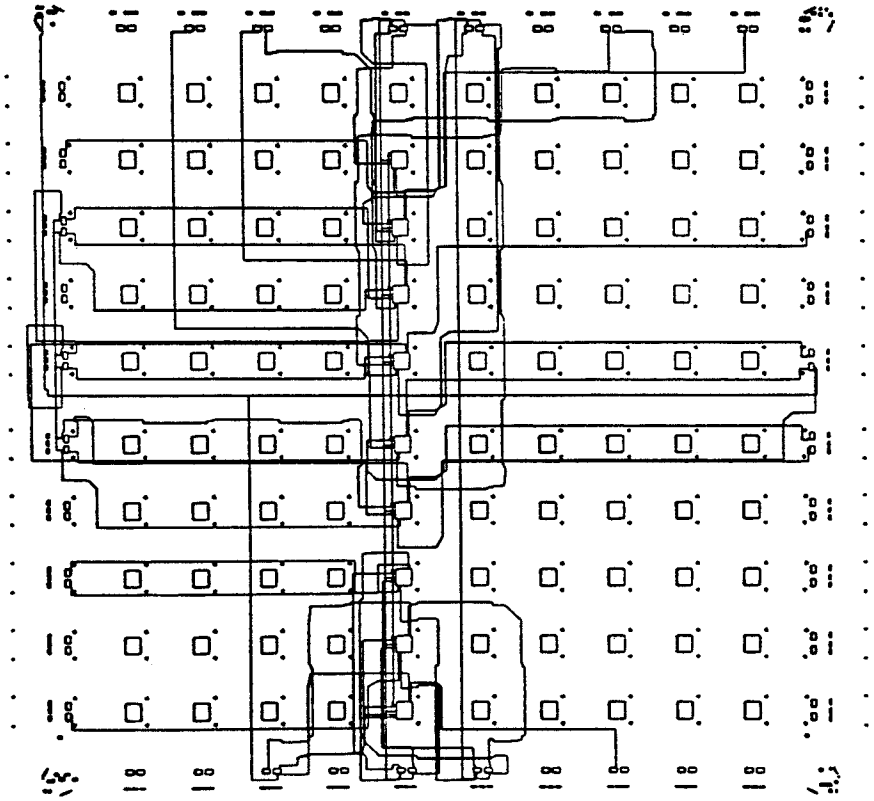

Figure 5. The same 16-bit adder design implemented in a Xilinx XC4003 device. The entire design occupies only a fraction of the available device resources. After the schematic diagram was entered, the design was automatically processed through the Xilinx software in under five minutes on a Sun® SPARCstation®.

A DESIGN EXAMPLE

The best way to describe the functionality of the Xilinx BLOX software is through an example. Figure 4 shows a block diagram for a simple design incorporating a 16-bit adder, two 16-bit data registers, plus various input and output pins. As it turns out, Figure 4 is more than just a block diagram—it also is the ***complete*** design drawn with the VIEW*logic* schematic editor (the design appears similar in other editors). Note that the size of the data path is 16 bits wide. The bus bounds are set on the INPUTS symbol feeding the first register.

The design is processed with Xilinx BLOX after being translated from the VIEW*logic* format into the Xilinx Netlist Format.

Xilinx BLOX first expands the data types on the data path. Because the data type is only defined on the INPUTS symbol in this example, the Xilinx BLOX software must determine the widths of all the other modules. Since the REGISTER module is attached to the INPUTS module via the DATA bus, Xilinx BLOX assumes that the REG_DATA output bus is the same width and type as the DATA input bus. This process continues until the width and type of all networks within the design are resolved.

The next step is the architectural optimization of the design. Xilinx BLOX improves the design by mapping logic functions into the special features of the Xilinx XC4000, where applicable. In this example, the software notices that both registers share a common asynchronous reset connection. Since the XC4000 devices have a special dedicated global set/reset function, the software removes the RESET signal from the two REGISTERS and automatically attaches it to the global set/reset function.

Likewise, the Xilinx BLOX software notices that the CLOCK signal feeds 32 flip-flops (or two 16-bit REGISTERs). It therefore assigns the CLOCK signal to one of the eight high-drive, low-skew buffers within the chip. This creates a fast, high fan-out clock distribution network with a delay of only 5.7 nanoseconds to all 32 flip-flops.

Next, the software notices that the adder function (ADD_SUB) should be implemented using the special fast-carry logic on XC4000 devices. It will automatically create a hard macro for the adder when it is expanded. Furthermore, the software notices that the second REGISTER connected to OUT can be incorporated into the same hard macro.

Finally, the software notices that the first REGISTER driving REG_DATA can be folded into flip-flops located in the I/O blocks. This boosts density by reducing the number of logic blocks.

The next major step is module expansion where the Xilinx BLOX software builds the specific implementations for each module. For this example, it will build a netlist description for the INPUTS, OUTPUTS, and the REGISTER driving the REG_DATA bus. However, the software will build a hard macro for the adder combined with the REGISTER driving the OUT bus.

After using Xilinx BLOX, the design is placed and routed using other Xilinx software. The entire design requires 34 I/O blocks and only nine logic blocks on an XC4000 device. The resulting design is shown in Figure 5.

If the designer now decided to build the same basic function but with a 32-bit data path, there would only be one change required on the schematic. The BOUNDS parameter on the INPUTS module would have to be changed from '15:0' to '31:0' and then the design reprocessed with Xilinx BLOX.

SUMMARY

Module generators, like Xilinx BLOX, significantly boost a designer's productivity. They also free designers from device-specific implementation details and allows them to focus more on actually designing the application. Furthermore, module generators allow designers to explore new or modified approaches more easily.

With the advent of 10,000+-gate FPGAs, module generators offer a powerful and effective means to make the design effort easier and more effective while handling ever-increasing design complexity.