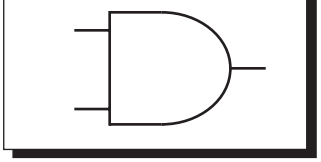


Support options for programmable logic don't differ much from board design

Steve Knapp

One must consider many options when purchasing hardware and software for programmable-logic design. Choosing the right solution takes some time to evaluate various products. Some products demo well but come up short when you use them through the complete design process. Before embarking on a new design path, one has to learn something about the tools involved. The methodology and software engineers use to complete a programmable-logic design are similar to those used for a board design. This article provides a basic overview of the design process and discusses the costs of associated hardware and software.

Programmable-logic design invariably involves four fundamental steps (Fig 1):

- design entry
- physical implementation, usually performed in software
- design verification, including simulation and in-system debugging
- device programming.

Steven K Knapp is the founder and president of OptiMagic Inc (Aptos, CA, info@optimagic.com), a firm that develops intellectual property and design software for programmable logic. Prior to founding this firm he held various applications, engineering and management positions at Xilinx and Intel's former programmable-logic division.

Design entry

A variety of tools are available to perform the first step, design entry. Some designers prefer to use their favorite schematic editors, while others want to specify a design with a hardware-description language such as Verilog, VHDL or ABEL. Still others choose to combine both methods in the same design. An on-going battle brews about which technique is best.

Traditionally, schematic-based tools provided experienced designers with more control over the physical placement and partitioning of logic on a device. However, this extra tailoring takes time. Likewise, language-based tools allowed quick design entry but often at the cost of lower performance or density, although synthesis for language-based designs has significantly improved in the last few years, especially for FPGA design. In either case, learning the architecture and the tool help you create a better design. Technology-ignorant design is possible, but at the expense of density and performance.

To help speed development with common design sections in any of these environments, many chip vendors provide intricate elements known as cores. These specialized elements, comprising such functionality as a PCI-bus interface or a DMA

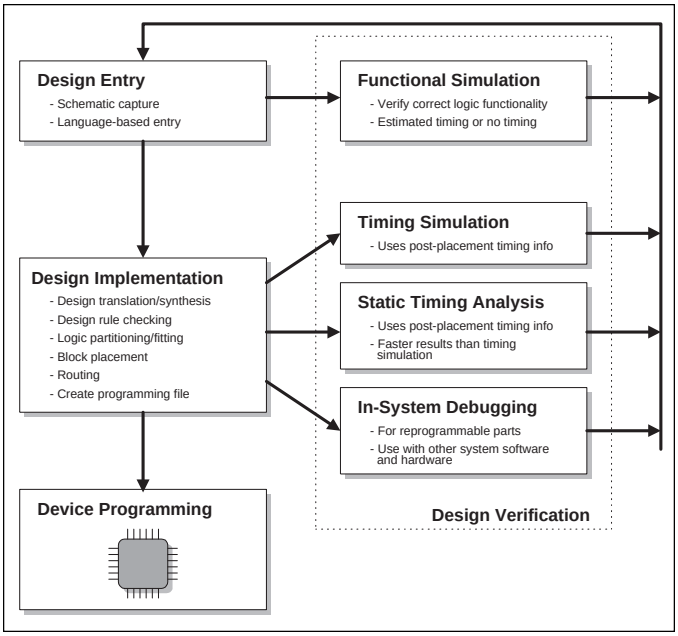


Fig 1—Design flow for programmable devices differs slightly from board design. Several simulation and analysis tasks can take place in parallel for large, multiengineer projects.

Minimum Usable Machine	Recommended Machine
• 66-MHz 486 • 16M bytes of RAM • 50M bytes of disk space • mouse • 2 serial ports • 1 parallel port • SVGA graphics (800 X 600)	• 200-MHz or faster Pentium, Pentium Pro, Pentium II • 64M to 128M bytes of RAM • 500M bytes of disk space • mouse • 2 serial ports • 1 parallel port • SVGA or better graphics • modem and Internet service

Table 1—Minimum and recommended PCs for programmable-logic design.

controller, are an increasingly important addition to the programmable-logic world. Core providers implement and verify these predefined functions in programmable-logic elements. Cores have been available for gate arrays for several years, but now that FPGA devices push beyond the 50,000-gate density level, cores should become a popular design-entry tool for programmable logic, as well.

Physical implementation

After an engineer enters and synthesizes a design (if using an HDL instead of a schematic for entry), it's ready for implementation on the target device. This first step involves converting the design into a format that the family-specific implementation tools recognize. Most implementation tools (also known as back-end tools) read standard netlist formats, and the translation process is usually automatic.

Once the back-end tools translate an incoming netlist, they perform a design-rule check and optimize it. Then the software partitions designs into the logic blocks available on the device. Partitioning is an important step for both FPGAs and CPLDs because it results in higher-routing completion and better performance for FPGAs and increased density and performance for CPLDs.

Once it partitions a design into logic blocks, the implementation software searches for the best location to place each block among all of the possibilities. The primary goal is to reduce required routing resources and maximize system performance. This operation is compute intensive for FPGAs and larger CPLDs because the implementation software monitors routing length and track congestion while placing a large number of blocks. In some systems, the software also tracks path delays in order to meet user-specified timing constraints. Overall, the process mimics printed circuit-board placement and routing. After completing the place / route process, the software creates a binary programming file that configures the device, much like a board-layout package produces Gerber files at the end of its design flow.

In large or complex applications, the software might not be able to place and route the design. Some packages try different options or run many iterations in an attempt to obtain a fully routed design. Generally, some designers try to use less than 85% of available device resources. This technique gives the software extra resources to help route a design. Also, some vendors supply floorplanning tools to aid in physical layout, which is especially important for larger FPGAs because some tools have problems recognizing design structure. A good floorplanning tool allows designers to convey this structure to the place / route software.

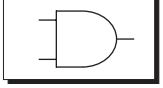
Design verification

Design verification exists at various levels and steps throughout this design process. Engineers should recognize several fundamental types of verification as applied to programmable logic. Functional simulation occurs in conjunction with design entry, but before place and route, to verify correct logic functionality, while full-timing simulation must wait until after the place / route step. Then the software backannotates logic and routing delays to the netlist for simulation. While simulation is always a good idea, programmable logic usually doesn't require the same exhaustive timing stimulation that gate arrays do.

In a gate array, full-timing simulation is important because the devices are mask-programmed and therefore not changeable. In addition, a design change typically involves thousands of extra dollars in non-recurring expenses (NRE) and weeks of time. Compare this penalty to a programmable device where changes are possible in minutes to hours at little or no cost. With in-system programmable (ISP) devices, such as SRAM-based FPGAs and ISP CPLDs, changes are possible even while the parts are mounted in the system.

One successful and popular technique for programmable logic is to functionally simulate a design to guarantee proper logic execution, verify timing using a static timing calculator and then verify complete functionality by testing the design in the system.

Some device vendors supply additional in-system debugging capabilities. For example, Xilinx ships a small pod called an XChecker cable that connects to a PC's serial port and allows downloading of a design. With



a few simple additions to a design and board, the XChecker cable can stop or single-step the system clock and read back the state of internal flip-flops. Likewise, Actel's Action Probes provide access to internal nodes within its antifuse based FPGAs. However, keep one caveat in mind—even though in-system debugging is quick and relatively easy, don't view it as a complete replacement for simulation.

Device programming

After creating and simulating a programming file, you're ready to program the device. The method depends on the target technology. Most programmable-logic technologies, including the PROMs used with SRAM-based FPGAs, require some sort of a device programmer. For a nominal fee, a local distributor could perform production programming, but you generally need a low-volume device programmer for development and prepro-duction work.

In-system programmable devices, including SRAM-based FPGAs, might not require a physical programmer, but they do need some intelligent system resource to download the file into the device. In this case, a board design must account for this requirement by providing programming support with a microprocessor, micro-controller or a JTAG test port.

How much does it cost?

No matter how device programming takes place, you must pay for tools somewhere along the line. A common question when starting out in programmable logic is "How much will it cost?" Like the punch line to the old joke, it all depends on how much you want to spend. Engineers can get started for less than $500 or can spend more than $20,000, depending on the desired features, capabilities, and interfaces.

With design software you'll find various options at $500 and below. Many vendors—including Actel, Lattice, Motorola, Philips and Xilinx—supply downloadable or free demo versions of their software for evaluation purposes. Most of these packages allow you to create complete designs for one or a few of their smaller devices (see the Resources list at the end of this column). At just below $500, Xilinx supplies complete schematic and simulation support for its lower-density FPGAs and all of its CPLD products. Altera offers a similar system, including logic synthesis, for under $1000.

Generally plan on spending $2500 to $20,000 for complete software, including logic-synthesis entry and broad support for a range of densities (all from one vendor, of course). Most programmable-logic suppliers don't want the price of the development system to be an obstacle to a large-volume design. Additionally, most vendors provide the full software on a free evaluation basis for 30 days or more. Most also offer significant multiuser or site-license discounts. However, don't forget the expense of software maintenance; most vendors charge roughly 15% of the purchase price per year for updates.

Computer hardware

Another expense in creating programmable logic is the computer that runs all this wonderful software. You might handle the job with an old legacy machine sitting around the office if it has the necessary resources. Table 1 shows the minimum usable machine and the recommended version. Actual requirements depend somewhat on the tools, device family and selected device, so it's usually best to contact the vendor about specific requirements. Generally, CPLDs require less computing power than do higher-density FPGAs, so a less-capable machine should work fine.

Engineers purchasing a new machine for the job should realize they gain no benefit from an MMX-capable processor for PLD software. A dual-processor machine is beneficial, especially when working with bigger (about 20,000 gates and above) FPGA devices. The software, even under Windows NT, doesn't execute any faster on a dual-processor machine, but the extra processor allows you to continue using the machine, while the other processor is consumed with

Space left intentionally blank.

placing and routing a large design. Be sure to buy enough memory to support the extra processor.

Larger devices, some now pushing past the 100,000-gate range, are massive memory hogs. If you plan on using these behemoths, be sure to stock up on enough RAM. Plan on a minimum of 64M bytes, with some vendors recommending 128M bytes or more. The bad news with memory requirements is that they'll only get worse.

Another consideration is the operating system. Most vendors today support either Windows 95 or NT 4.0 or plan to do so in the immediate future. Anyone still using Windows 3.1 should plan on upgrading.

A modem and Internet connection are also no longer just nice to have. The various software and device vendors typically offer on-line technical support including software upgrades and the latest technical information. Some have just started releasing web-based software tools such as the Xilinx LogiCore PCI configuration tool. Also the Internet connection allows you to stay in contact with fellow engineers through newsgroups. One of the most relevant newsgroup to designers is COMP.ARCH.FPGA.

Programming support

Beyond a computer, you'll probably need a device programmer. A wide variety of units offer various cost-flexibility tradeoffs. Most device vendors sell a dedicated or point-solution programmer, which is generally far less expensive (starting from about $500) than a more universal programmer but might require a separate purchase for each new device family. Universal programmers are more expensive (from $2500) but are recommended for engineers who plan to work with a variety of devices from multiple vendors. Antifuse-based devices usually require a more sophisticated programmer because the unit performs some device testing and antifuse integrity checking. Also, quad flat packages and other surface-mount components might need expensive sockets.

If you're using SRAM-based FPGAs and plan to download the design from system memory to the FPGA with a processor or if you plan to have a system boot from an external byte-wide EPROM, an additional programmer might not be necessary—a byte-wide EPROM programmer works fine. Likewise, if you're using ISP or downloadable devices, again a programmer might not be necessary. The download cable should suffice for most prototyping. ***PE&IN***

Resources

A few of the many resources available on the world wide web to help engineers with FPGA design include the following:

Various programmable-logic design software: *www.optimagic.com/software.html*

Low-cost or free design software: *www.optimagic.com/lowcost.html*

Xilinx web-base PCI core configuration tool: *www.xilinx.com/products/logicore/cg_intro.htm*

Various programmable-logic-related newsgroups: *www.optimagic.com/newsgroups.html*

Editorial Feedback
This article's value to me was:
High—263 Average—264 Low—265

Space left intentionally blank.