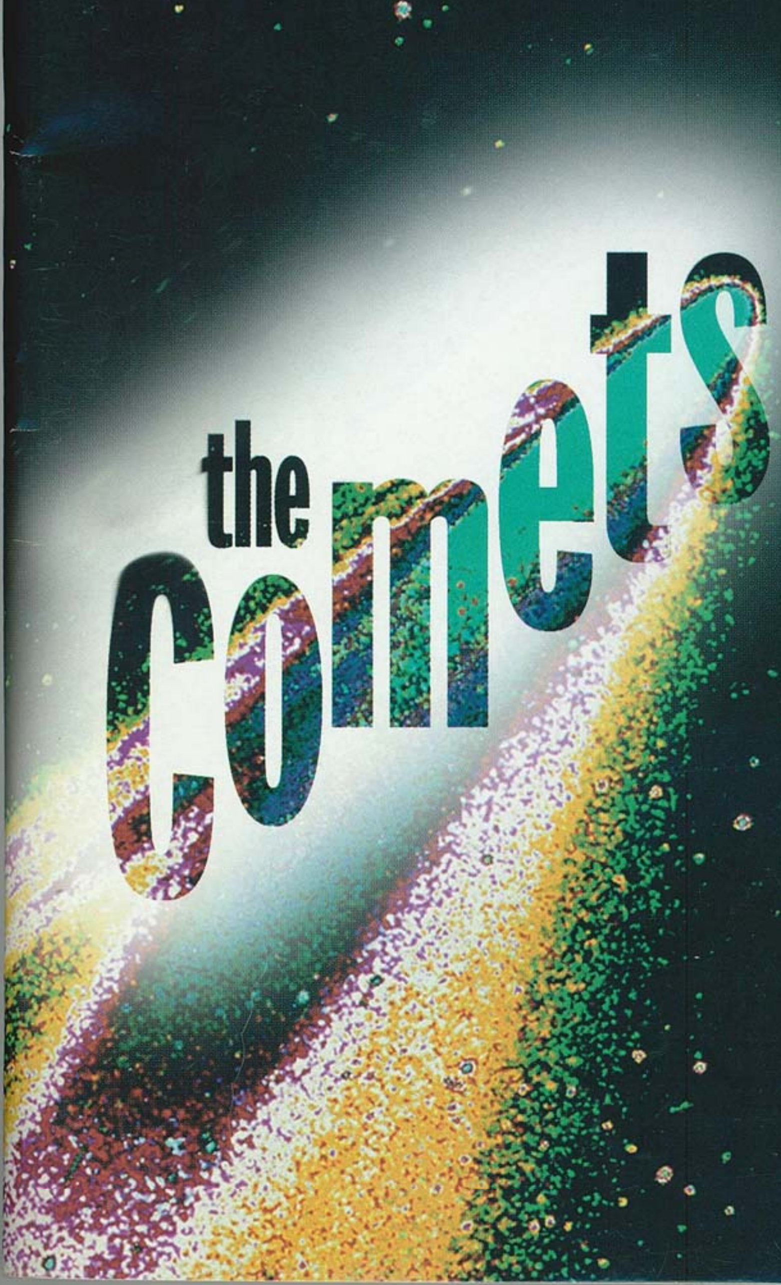
September 2000

Communication Systems Design

For OEM Design Engineers and Managers



THE COMETS
Up-and-Coming Comm Companies Step into the Spotlight.

VOICE GATEWAY DESIGN
Linking Packet and Telephony Network Architectures.

FIBER-OPTIC TEST
Digging into Optical Measurement Issues.

PLUS:
Special Section on CompactPCI Design.

www.csdmag.com

Contents

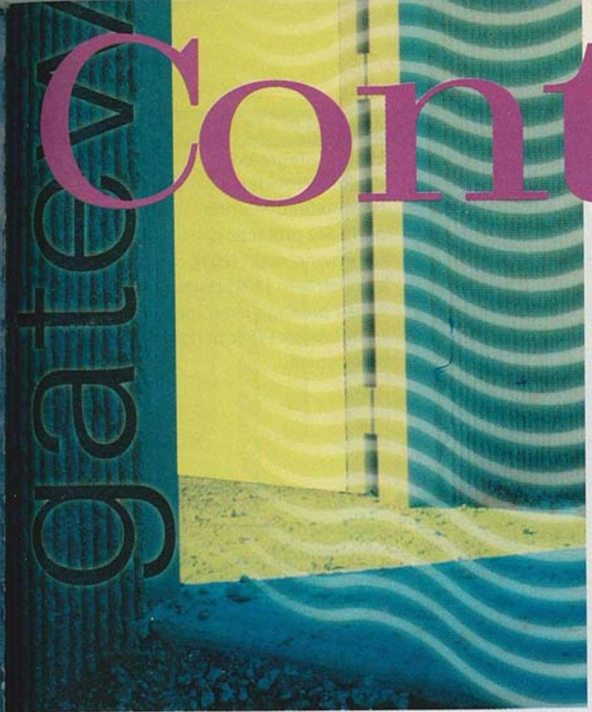


ILLUSTRATION BY JANE HAMBLETON

Volume 6, Number 9

Communication SYSTEMS design

Postal and Rates: *Communication Systems Design* (ISSN 1086-4644, USPS 013-751) is published monthly by CMP, 600 Harrison Street, San Francisco, CA 94107-1391, telephone (415) 905-2200, fax (415) 905-2499. Domestic subscription rate for one year, $50.00, two years $80.00. Canada rates for one year are $57.00, $94.00 for two years, payable in U.S. funds. For all other countries, the rate for one year is $75.00 for surface mail and $95.00 for airmail, prepayment in U.S. funds required. For subscription inquiries, please write to us in care of *Communication Systems Design*, P.O. Box 3405, Omeda Communications, Inc., Northbrook, IL 60065-9470, or phone (847) 297-5215. Single copy issues are available in the U.S. for $7.00 plus $2.00 postage; international $10.00 with postage included. Major credit cards accepted. All

Navigating Through FPGA Designs

The FPGA design process is not as seamless as one might like. However, powerful synthesis, simulation, and programming tools are here to help.

By Kurt Aronow

Dealing with various FPGA design challenges is a little like climbing one of the famous routes on Hallet's Peak in Rocky Mountain National Park. A short time ago, a large block fell off the mountain (without any climbers on it) removing a significant part of one of the most popular routes. The route, which was never easy to follow, now presents new tests.

Similarly, an engineer trying to "climb" an already intricate FPGA design route today will inevitably face a new set of challenges. For instance, the software tools to design, compile, and simulate FPGAs have become much more powerful while the design requirements have become significantly more complex.

However, any difficult climbing or designing is not nearly as intimidating when you have some of the "beta" on the route. This article explores some considerations engineers may confront when developing different FPGA routes. To do this, we will explore tools, clocking, programming language, synthesis, and simulation issues.

FPGAs are a type of programmable logic device (PLD) containing thousands of flip-flops and logic gates. By replacing discrete logic that would otherwise occupy hundreds of times more board space, FPGAs have increasingly become a common feature in telecommunication hardware designs.

Early languages

Early languages such as DataIO's ABEL and Altera's AHDL were once the tools of choice for FPGA hardware design. Both of these are well-suited for designs that are close to the hardware. However, writing code that is close to the primitives in the hardware is not necessarily the clearest way to write code. Also, neither language supports test benches for testing the code.

Two other languages, very-high-definition language (VHDL) and Verilog, have emerged to address these issues. VHDL came before Verilog and is a more highly typed and verbose language. Writing in Verilog is similar to writing in C, while VHDL more closely resembles Pascal. One feature available in VHDL and not Verilog is the ability to set up three-dimensional (3D) memory spaces. However, clear, concise code can be written with either language, and both are widely used.

Just as in writing software for embedded applications, a coding standard is important when more than one person will ever have to use the source code. The big danger is that when the person who wrote the original code leaves or moves on to another project, no one will understand how it works if the code ever has to change. Even the original designer is likely to forget it in several months.

One can easily write individual lines of understandable HDL code that collectively become extremely difficult to follow. A good coding standard will help alleviate this by providing guidelines for hierarchical structures and component instantiations. For instance, many books use various types of flip-flops as examples to model component instantiations (mostly because these are already understood by the readers).

However, in practice, it's generally poor coding style to instantiate logic by mapping each register to various kinds of flip-flops. This can lead to longer, more obfuscating logic that does not take advantage of the ability to write in VHDL and Verilog at a higher level (see **Code Listing 1**).

Writing specifications

Writing a design specification for the required logic usually appears to add too much time to a design cycle. However, it often turns out that writing a design specification actually facilitates getting a part functioning sooner. Writing the specification forces the design team to actually think through items that are clear at a high level but not so clear at a detailed level. For instance, getting the software engineers on a project to agree on what is going to be in an FPGA's memory map can be a real eye-opener. As the design progresses, it's also useful to write a description (with timing diagrams) of how the design actually works.

After writing (and agreeing on) a detailed design specification, one can estimate the size of the PLD based on the number of flip-flops likely to be needed. Sometimes, a spreadsheet is useful for this exercise. Generally, one can rest assured that the amount of logic required for a design will increase as the work progresses.

With the information in hand about the size of the design, one of the first choices is deciding whether to use a complex PLD (CPLD) or an FPGA.

CPLDs retain their program when the power is off because they are PROM or flash based. Most modern CPLDs can be purchased with in-system-programmability (ISP) capability. In the last few years, the emergence of ISP in many PLDs has enabled more parallel effort between circuit-board and logic designs. ISP enables PLDs to be reprogrammed by simply connecting a cable between a PC and a ten-pin joint test action group (JTAG) header on the circuit board with the PLD. This allows programmable test points that can be especially useful with ball-grid-array (BGA) packages.

FPGA basics

FPGAs are generally either fuse linked or static-RAM (SRAM)-based. Fuse-linked FPGAs retain their program when the power is turned off. However, they are one-time-programmable (OTP) devices and can be difficult to replace if their packages have a considerable number of pins. Also, fuse-linked FPGAs can take many minutes to actually program.

SRAM-based FPGAs lose their program when the power is removed. Serial EPROMs with ISP capability are now available for programming many SRAM-based FPGAs. Although a competitor could copy the data going from the EPROM to the FPGA, reverse engineering the de-

Code Listing 1: Clear and Obfuscating VHDL Examples of aDivide-by-14 Circuit

```
— Concise VHDL example of a high-speed synchronous divide by 14
— The output should have a 50% duty cycle.
process(nRes,Clk40Mhz) begin
 if (nRes = '0') then Cntr <= x"0"; CntD <= '0'; Div14 <= '0';
 elsif rising_edge(Clk40MHz) then
       if (CntD = '0') then Cntr <= Cntr + x"1"; else Cntr <= x"0"; end if;
       if Cntr = x"C" then CntD <= '1'; else CntD <= '0'; end if;
       if (Cntr >= x"6") then Div14 <= '1'; else Div14 <= '0';
 end if;
end process;
---------------------------------------------------------------------------
— Less concise example of the same divider
— Assume that D_Flipflop and a Counter_4_Bit are predefined components.
— Note that the code takes the same number of lines except that now, the
— reader has to find the Counter_4_Bit and D_Flipflop components to follow
— the code completely. These components must also be maintained.
Counter14: Counter_4_Bit port map(nRst => nRes, Clk => Clk40MHz,
 Rollover => CntD, CountOut <= Cntr);
Rollover_FF: D_Flipflop port map(nRst => nRes, Clk => Clk40MHz,
 Din => Cnt_At_C, Qout <= CntD);
Div_FF: D_Flipflop port map(nRst => nRes, Clk => Clk40MHz,
 Din => Cnt_GE6, Qout <= Div14);
Cnt_At_C <= '1' when (Cntr = x"C") else '0';
Cnt_GE6 <= '1' when (Cntr >= x"6") else '0';
```

sign from that data would be incredibly difficult. CPLDs (with the security bit set) and fuse-linked FPGAs are much more difficult to copy. The major advantage of FPGAs compared to CPLDs (other than lower current draw) is higher logic density at a lower cost.

If a design requires more than approximately 200 flip-flops, an FPGA often becomes a better choice than a CPLD, assuming that very low propagation delays or being programmed when power comes up are not important. For quick development cycles, ISP capability for a PLD (even in the field) is useful. For this reason, one should have a special kind of application to decide to use a fuse-linked FPGA. An example would be for a satellite application where a fuse-linked part may offer substantially better immunity to cosmic radiation.

Generally, one must use the place-and-route tools provided by the vendor of the PLD or FPGA. The place-and-route software accepts either a netlist from the synthesis software or the source code from an HDL that it synthesizes directly. The outputs of the place-and-route software include a programming file for the PLD and behavioral model for simulation.

Some of the new, larger FPGAs or PLDs offer some nice extra features. For instance, some of Altera's largest PLDs offer the ability to probe some signals during operation through the JTAG port, providing something like a virtual logic analyzer. Altera and Xilinx are offering digital delay loops to help synchronize clocks to external signals.

Importance of global clocks

The importance of global clocks is a worthwhile discussion topic during FPGA design. Global clocks should be considered for the multiple clock sources needed in many designs. Several reasons exist:

- *Synchronous logic.* Global clocks allow logic such as synchronous counters to function without glitches. (Many of the next states of the flip-flops of these synchronous counters depend on most of the current states.) With Altera's Max+PlusII, for example, synchronous logic can sometimes be worked around by declaring a small counter to be a clique. Using cliques can help place the logic cells for the counter close enough together that they receive the clock virtually simultaneously.
- *Simpler routing.* Global clock lines go to all logic cells on the FPGA (or macrocells on a CPLD), so clocks do not tie up routing resources that can be used for other signals.
- *Short clock-to-output delays.* The shortest clock-to-output delays are generally obtained with global clocks.
- *No hold time.* With global clocks, usually, no data hold times are required. Clocking a flip-flop with a nonglobal clock using input data from outside the FPGA probably means that the external input data must stay valid for several nanoseconds after the clock goes away. This was an issue in a design with a Motorola 860 microprocessor interfaced with an FPGA. The particular chip select line coming from the 860 was de-asserted at approximately the same time as the address and data. Bringing the chip select from the 860 microprocessor to a global clock input of the FPGA solved the problem.

Routing clocks

The discussion above should convince most designers, no matter which part they are using for implementation, to use global clocks for flip-flops where practical. Generally, the best performance using global clocks can be obtained by bringing in the global clock on a dedicated global clock pin. However, sometimes it's desirable to put an internally generated clock onto a global net. Actually implementing this with the synthesis and place-and-route software tools can be tricky.

In one FPGA design, the software could not route the internally generated clock. The reason turned out to be a bug in the software that would not route the internally generated clock onto a global net before it routed the other nets. Once these other nets were routed, the software lacked the routing resources to put the internally generated clock onto a global net. If this situation exists, and if one can stand the input and output pad delays, it's usually safer to feed the internally generated clock out of the chip and then back into a global clock pin.

Engineers should also watch out for global clocks that can only

Hi-Flow™ the Most Complete Phase Change Material

The Original and Most Stable "Phase Change" Thermal Interface in a Pad Form

When you choose a thermal interface look at the whole picture. One product may solve one problem but cause other problems like flow-out or grease migration.

We've put Hi-Flow into a pad form for easier handling and installation. **Flexible Hi-Flow** changes from a solid and flows (phase change temperature) assuring total wet-out of the interface.

Hi-Flow is Easy to Handle and Easy to Assemble

- Ideal for CPUs
- No Tabs, No Migration, High Stability
- Better Thermal Performance, .05°C-in²/W thermal resistance, performs as well as grease Call for samples and more information: **(888) 942-8552**.

THE BERGQUIST COMPANY

18930 West 78th Street Minneapolis, MN 55317 • (952) 835-2322 Fax (952) 835-0430 • www.bergquistcompany.com

Thermal Products • Membrane Switches • Electronic Components • Touch Screens be used for clocking flip-flops. A global clock may not be used to gate other signals. Therefore, if a design requires a signal to be a global clock and a source for combinational logic, the signal should be brought into two pins on the FPGA where one of the pins is a global clock pin. (Note that some clocks may clock flip-flops on either their rising or falling edges. Similarly, a global reset may only feed the asynchronous, negative logic reset input of flip-flops with these parts.)

Suppose a designer wants to bring two clocks into an FPGA and then select one of them internally to be a global clock for many flip-flops. It is useful to remember that this clock selection will bear a gate or logic-cell delay of several nanoseconds. For high-speed designs, this delay may be too much. An alternative is to use a high-speed field effect transistor (FET) subnanosecond switch external to the PLD. While the switch may require several nanoseconds to select the other clock, the delay will be minimal once the clock is selected.

An engineer might also want to use PLD outputs to feed high-speed backplane drivers. A potential problem is that the clock-to-output time of the PLD may be too long. This problem can sometimes be solved by using drivers with

Code Listing 2: VHDL Example of an Altera LCELL and GLOBAL

```
— This example shows how to get lcells and globals to work with Exemplar
— Leonardo Spectrum and Altera Max+PlusII. Additional comments show which
— lines need to be changed to work with Synplicity. Note that for Exemplar,
— the exemplar.lmf file had to be edited in Maxplus2 to show in1 and y as the
— ports of an lcell and a global. (This may not be necessary with the newest
— versions of Maxplus2.) Also note that in the Maxplus2 report file, the
— lcells and globals are referred to by their input signal names, not their
— output signal names. In the Maxplus2 individual logic options, they are
— referred to by their line labels.
— ****************************************************************************
library ieee; use ieee.std_logic_1164.all;
library EXAMPLAR; use EXAMPLAR.exemplar_1164.ALL;  Library for Exemplar only
use ieee.std_logic_unsigned.all;  Declare this library for Synplicity only
library altera; use altera.maxplus2.all;  Declare this for Synplicity only.
entity lcell_tst is port(
 Clk_A, Clk_B in std_logic;  input clocks
 Clk_Sel: in std_logic;  select a cloCk
 nRes: in std_logic;  global reset
 D_In: in std_logic_vector(3 downto 0);  data input
 D_Out: buffer std_logic_vector(3 downto 0);
 Clk_Out: buffer std_logic);  Selected clock output for other circuitry
end lcell_tst;
-------------------------------------------------------------------------------
architecture arch_lcell_tst of lcell_tst is
 signal Clk, GClk, Ck: std_logic;  for LCELL & GLOBAL instantiations
 attribute noopt: boolean;  Only use this with Exemplar.
 attribute noopt of lcell: component is true;  Only use this with Exemplar.
 attribute noopt of global: component is true;  Only use this with Exemplar.
 component global port(in1: in std_logic; y: out std_logic);
 end component;  Only use this component definition with Exemplar.
 component lcell port(in1: in std_logic; y: out std_logic);
 end component;  Only use this component definition with Exemplar.
-------------------------------------------------------------------------------
 begin
 Clk <= (Clk_A and Clk_Sel) or (Clk_B and (not Clk_Sel));
 l_Ck: lcell port map(Clk, Ck);
 l_Clk_Out: lcell port map(Clk, Clk_Out);
 l_GClk: global port map(Ck, GClk);
 process(nRes,GClk) begin
 if (nRes = '0') then D_Out <= x"0";
 elsif rising_edge(GClk) then D_Out <= D_In + x"1"; end if;
 end process;
end arch_lcell_tst;
```

built-in flip-flops. The clock-to-output time of the backplane drivers may be considerably faster than the PLDs.

Instantiating global signals and other special PLD-brand-specific features within the synthesis software requires different implementations for each PLD brand. It's useful to make up short test cases for the desired features to try to get the synthesis software to communicate the right information correctly to the place-and-route software. This helps to eliminate extraneous information from the inevitable troubleshooting to make things work. A VHDL example of this exercise, showing how to work with 1 cells and globals in Exemplar's Leonardo Spectrum and Altera's Max+PlusII, is shown in **Code Listing 2**.

Synthesis and simulation

For larger FPGA designs, the best VHDL or Verilog synthesis generally comes with a specialized synthesis tool rather than the PLD manufacturer's own software. For CPLDs and some small FPGAs, the PLD manufacturer's own software does an equivalent job of language synthesis. For these cases, it is better to skip the extra synthesis tool to avoid any problems in interfacing between the tools.

For simple simulations, one can make use of the graphical simulators often built into the PLD vendors' place-and-route software. However, the complexity of these simulations is often limited. Also, one must check the results of the simulation graphically.

For more complex designs, it is more effective to simulate the design with Verilog or VHDL test benches. Often, these test benches are more complicated than the original source code. One can also create models of peripheral parts to the FPGA with these test benches in order to verify that things will work together correctly. For instance, it is common to model SRAMs that are connected to PLDs with the test bench. An engineer can design the test benches to automatically check important data conditions and to report any errors to a command window.

A designer can choose to simulate the design before it is synthesized (functional model) and after it has been run through the place-and-route software (behavioral model). Functional models are easier to create since they are simply the original source code, and they usually run much faster through test benches compared to behavioral models. However, behavioral models can reveal timing problems that are not at all obvious with functional models.

There are a number of synthesis and simulation tools available from vendors such as Exemplar, Synplify, Model Technology, Aldec, and Orcad. To operate these tools, engineers typically must set up a PLD vendor's library in the simulator. For instance, for simulating Altera PLDs in ModelSim, the designer sets up a library with the following path (assuming the default installation of Altera's Max+PlusII): `c:/maxplus2/vhdl87/vital/ v3_0/alt_vtl`.

During an evaluation, it was determined that ModelSim was a good simulator although its own scripting language is obtuse. It was useful to create script files just to execute series of ModelSim commands that were also available via its GUI.

When a command was invoked from the GUI, the corresponding text for the command would appear on the command window. The text could then be copied to a script file.

Evaluations also showed that ModelSim does not work well with some editors that seize too much control of a script file while it is open in the editor and ModelSim is using it. Ultraedit was found to be one editor that did not cause this problem.

Be careful

Even the most careful simulations sometimes will not match real-world conditions, however. One should try to troubleshoot designs with a logic analyzer if at all possible.

With care, one can usually find an aesthetic, efficient route through an FPGA design within a reasonable schedule. It is useful to remember that one is on a route, and skipping steps or taking shortcuts can easily lead one away from a transparent, functioning design.

Kurt Aronow is a senior electrical engineer with Aztek Engineering, Inc. in Boulder, Colorado. He is a registered PE who received a BSEE at the University of Texas and a MEEE at the University of Colorado. He can be reached at kurt.aronow@aztek-eng.com.

So many comm choices...
So little time...
So good to have help.

T1/E1•FRACTION T1•T3/E3
FRAME RELAY N 95•ATM
X.25•LA LC•SDLC
CUSTO ICA•SNA
HSSI•WI WS NT•OS/2
SCO UNIX SDN•PCI•ISA•QNX
SUN OS•DOS•OC-3•PPP•LINUX

Let Quadron help you simplify your life and solve your communication problems with our solutions. We've been helping people like you worldwide for over a decade.

We offer a large array of communication cards that range from simple plug-and-play to custom solutions that precisely match your needs. We'll get you what works the first time. Check out our web site and call us real soon.

Quadron®
www.quadron.com

telephone **805-966-6424** • *fax* **805-966-7630** • *email* **info@quadron.com**

©2000 Quadron Corporation