



BUGS CAN ORIGINATE AT EVERY STAGE IN THE FPGA DESIGN FLOW; DEBUGGING SUCCESS DEPENDS ON USING THE RIGHT TOOLS AND METHODS.

# Debugging FPGA designs may be **HARDER THAN YOU EXPECT**

BY CHRIS SCHALICK • GATEROCKET

In the not-so-distant past, the question used to be, “Can you do that task in an FPGA?” With the advent of modern FPGA devices, however, the question has become, “Why wouldn’t you use an FPGA?” Modern FPGAs’ complexity rivals that of ASICs. The chips contain hundreds of thousands of flip-flops, multimegabits of RAM, thousands of DSP slices, multiple soft or hard processor cores, multiple SERDES (serializer/deserializer) channels, and more. Integration of FPGA designs no longer takes place with one hardware engineer working in relative isolation and designing from the ground up. Instead, modern projects require team-based design, substantial quantities of third-

party IP (intellectual-property) cores, and complex hardware/software interactions. The ability to debug such an FPGA design determines time-to-market—and time-to-profit—success. However, debugging one of today’s multimil-

lion-gate FPGA designs is not a trivial task.

Consider a generic, high-level view of the conventional FPGA design flow (Figure 1). As an engineer, you could draw this figure in various ways—adding

and rearranging the various blocks and arguing about minutiae. Newcomers who come to the design with faith in various aspects of the FPGA design flow soon discover how unfounded this faith can be. For example, it’s common in FPGA IP to use two representations—one for high-level simulation and the other for the actual implementation. Many design-and-verification engineers assume that these two representations are guaranteed functionally identical. But sometimes they are not. It is also reasonable to assume that synthesis and place-and-route tools are robust and will not introduce errors in the design. But they sometimes do. When you move to FPGAs from ASICs, you may quickly discover that bugs can appear at any stage of the FPGA design flow.



Most techies occasionally use design-related forums and sites. On these sites, real-world engineers post some cases spanning the FPGA design flow. The following examples illustrate the types of issues FPGA designers can encounter. The user, vendor, and tool names in this article are not the actual names of the participants. These examples involve a number of design issues: the quality of RTL (register-transfer-level) code, the quality of the IP, the quality of results from the synthesis engine, and the quality of results from the place-and-route engines.

**CASE 1: RTL QUALITY**

It isn't surprising to hear that you're going to get better or worse results depending on how you write your RTL code. Consider an example from open cores.org (Figure 2). This engineer's well-intentioned desire was to produce a better gate-level implementation netlist. Unfortunately, this code behaves differently in synthesis from the way it behaves in simulation, simply because the simulator ignores the synthesis pragmas.

Here, the pragma tells the synthesis tool that it can make arbitrary decisions about unspecified choices. This statement is contrary to what happens in the simulator. The end result of this difference is that inadvertently writing to an unspecified address actually overwrites a real register. One approach is to use linting tools, but most FPGA designers

**AT A GLANCE**

- Issues that arise during FPGA debugging include the quality of RTL (register-transfer-level) code, the quality of the IP (intellectual property), the quality of results from the synthesis engine, and the quality of results from the place-and-route engines.
- It's common in FPGA IP to use two representations—one for high-level simulation and the other for the actual implementation.
- IP models you use for simulation may differ in significant ways from the corresponding models the place-and-route software uses.

don't use them. Linting tools are common in ASIC environments, but design engineers who gained their experience in a traditional FPGA shop tend to have no ASIC tools lying around. Managers don't want to spend money buying such tools, designers don't want to spend time learning to use them, and so registers are overwritten.

**CASE 2: IP QUALITY**

In the case of ASIC designs, third-party IP vendors typically deliver their cores in the form of RTL, which may be encrypted, obfuscated, or unencrypted. Therefore, the same RTL representations of the IP blocks that you use during initial software simulation are the representations that you subsequently

synthesize, place, and route along with the rest of the design. This commonality provides a high level of confidence that the RTL and gate-level representations are functionally equivalent.

The situation is different with FPGAs. IP vendors often supply two models—one at a high level of abstraction and one at the gate level. So the IP models you use for simulation may differ in significant ways from the corresponding models the place-and-route software uses. Often, the high-level simulation model contains behavioral constructs to make the software simulations run faster. Synthesis tools cannot handle these constructs, however. Moreover, subtle differences often exist between the behavioral- and gate-level representations that manifest themselves only when you deploy the FPGA design in its target system.

Consider an example from an FPGA vendor's user-forum site (Figure 3). Running the simulation on the gate-level netlist from the IP vendor shows different results from those of the behavioral model from the same vendor. In this case, the gate-level representation was correct and the behavioral model wasn't. In many cases, it's the other way around.

**CASE 3: SYNTHESIS PROBLEMS**

People believe that synthesis tools are more robust than they are. Even though some synthesis tools have been around for years, users are still logging bugs

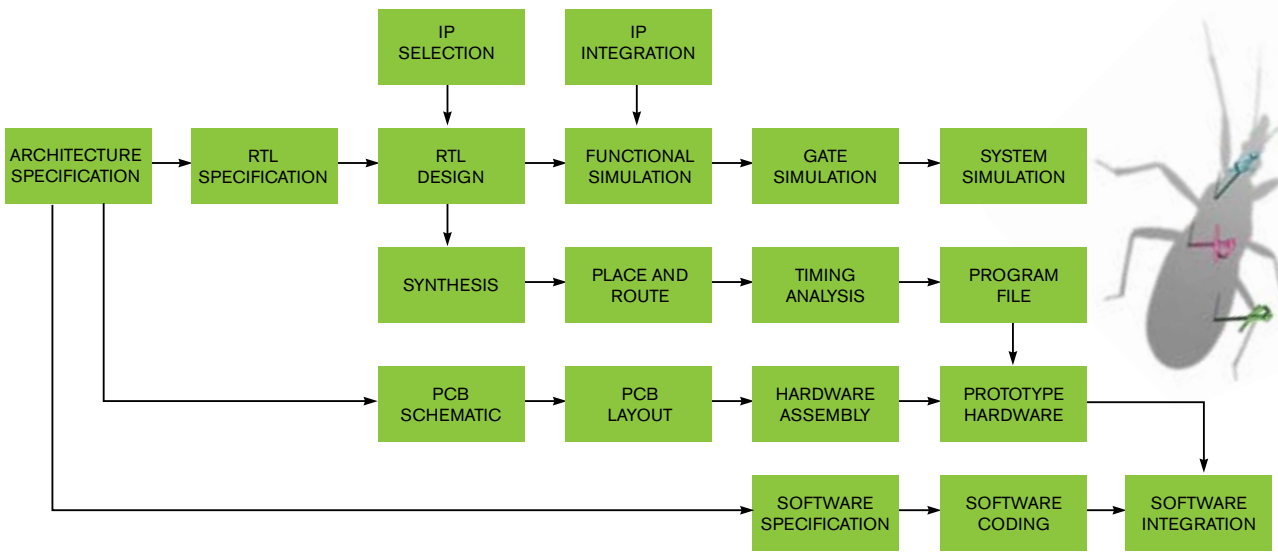


Figure 1 A modern FPGA design flow has many discrete steps.

against them. Consider a case from [fpga-faq.com](#) (**Figure 4**). This example involves a bizarre set of circumstances. First, the synthesis tool makes an odd choice. Then, critical warnings come up that are tremendously informative, but a deluge of messages buries them, causing the designer to overlook them.

In this case, the tool acknowledges that it's making an odd choice and is telling the user that it's worried. In many cases, the tool thinks that it's doing the right thing and doesn't say a

```
I2C Core RTL Simulation / FPGA mismatch...

Subject: I2C Core RTL Simulation / FPGA mismatch...
msg#00024, hardware.opencores.cores

I'm using an FPGA to verify an ASIC which is using the
I2C core. I'm getting a mismatch between the RTL simu-
lation & the actual FPGA gates when writing to the I2C
CR (command register). I setup the 2 prer bytes with
known values. Then when I write the Cr register, the
value written to the cr overwrites the prer(7:0). This
is the case only in the FPGA. In rtl simulation, it
works as expected.

Upon further examination of the rtl code, I think I
have found an issue. Please correct me if I'm wrong.

Here's the code snippet with the case statement...

// generate registers
always @(posedge wb_clk_i or negedge rst_i)
  if (!rst_i)
    begin
      prer <= #1 16'hffff;
      ctr  <= #1 8'h0;
      txr  <= #1 8'h0;
    end
  else if (wb_rst_i)
    begin
      prer <= #1 16'hffff;
      ctr  <= #1 8'h0;
      txr  <= #1 8'h0;
    end
  else
    if (wb_wacc)
      case (wb_adr_i) // synopsys full_case paral-
lel_case
        3'b000 : prer [ 7:0] <= #1 wb_dat_i;
        3'b001 : prer [15:8] <= #1 wb_dat_i;
        3'b010 : ctr          <= #1 wb_dat_i;
        3'b011 : txr          <= #1 wb_dat_i;
      endcase

Does the // synopsys full_case parallel_case not tell
the synthesis Tool that ALL the cases are listed? All
other cases are considered "don't care" by the synthe-
sis tool, right?

The "cr" register is mapped at address 3'b100. This
case is not
Listed in the case statement, so this bit is optimized
away and is not considered important.

That's the problem... now when address 3'b100 is writ-
ten, the case statement sees it as 3'bx00 and the
prer(7:0) is written as well as cr.

Does the case statement not need a default line which
keeps the
Previous values for the 4 registers? Am I misunder-
standing this concept?
```

**Figure 2** In this posting from [opencores.org](#), an engineer wanted to produce a better gate-level implementation netlist, but the code behaves differently in synthesis from the way it behaves in simulation.

word. The problem stems from aggressive optimizations on the tool's part. Today's designs are large and their corresponding synthesis can take a long time, so synthesis-engine developers take short cuts. Whenever the synthesis-engine developer cuts a corner, however, he must account for an enormous set of possible conditions. Assumptions can turn into errors.

#### CASE 4: PLACEMENT-AND-ROUTING PROBLEMS

As a final example, consider a power-up initialization problem from [fpga-faq.com](#) (**Figure 5**). In this case, the place-and-route tool doesn't know what to do; it decides that a register must have some state, so it makes an arbitrary—and unfortunate—choice that causes the silicon to do something odd and unexpected. The engineer is coming up with solutions but is unsure whether they will work because visibility into the root cause is so low.

These examples are teasers; you occasionally hear more complex stories. In one such story, a design team lost weeks go-

```
January 9th, 2008, 12:00 AM
username
Vendor Pupil
Posts: 9 Rep Power: 558

VHO bug for FFT v2.2.1 (streaming)
I have simulated a 2048 point FFT v2.2.1 (streaming)
in both ModelSim and vendor tool. In vendor tool cor-
rect streaming behaviour is observed, with assertion
of master_source_sop immediately following assertion
master_source_eop, and master_source_ena remaining
high at all times.

When simulating the VHO file in the simulator incor-
rect behaviour is observed, with a gap between master_
source_eop and master_source_sop assertions, during
which time master_source_ena is taken low.

Attached are the plots of what I have described. Does
anyone know of a solution to this problem? (Besides
trying the latest ver FFT core which is not an option)

#2 January 9th, 2008, 12:08 AM
Location: Bochum Germany
Posts: 1,011

Hello username,
if the error is with the FFT compiler generated simu-
lation model, who about Modelsim gate level simulation
of the Vendor compiled FFT design? I guess, the simu-
lation model may be faulty,

Regards,contributor
.....
#4 January 9th, 2008, 05:54 PM
Posts: 9
Rep Power: 558

Re: VHO bug for FFT v2.2.1 (streaming)
Thanks for the response contributor. I did try regen-
erating the VHO using the post synthesis netlist gen-
eration as you suggested.

This has fixed my problem but of course results in
considerably slower simulations.

username
```

**Figure 3** In this posting, a user finds that running the simulation on the gate-level netlist from the IP vendor yields different results from those of the behavioral model from the same vendor.

---

## EACH FUNCTIONAL BLOCK IN THE DESIGN SHOULD BE ABLE TO RESIDE IN THE SOFTWARE DOMAIN, THE HARDWARE REALM, OR BOTH.

---



ing back and forth with IP vendors trying to work out a problem. In one case, a place-and-route tool optimized some core functions out of the vendor's own IP. Nevertheless, you can do astounding things with a modern high-end FPGA and associated design tools. You can—in one day and on your desktop—synthesize a design that's equivalent in complexity to a Pentium CPU, run place-and-route tools, and generate the corresponding FPGA-configuration file. Just five years ago in the ASIC world, you needed a design center for one to three months to come up with a workable place-and-route strategy.

Despite these advantages, the downside is that errors can creep into the design at every stage in the process, and these errors often don't manifest themselves until the design is in a real board in the laboratory, by which time they can be difficult and time-consuming to detect, isolate, identify, and resolve. Software simulation provides a high level of visibility into the design, but it runs relatively slowly even at the RTL. By comparison, running the design in real FPGA hardware, such as a development or prototype board, provides hardware speeds, so any problems quickly manifest themselves by crashing and burning. Due to lack of visibility into the chip, it can be difficult to determine what's gone wrong. Is there an error with the RTL source code that causes it to behave differently with the simulator from the way it behaves with the synthesis engine? Are there functional differences between one or more of your behavioral third-party IP blocks and their gate-level equivalents? Is it perhaps a case in which your RTL and IP are functionally correct, but the synthesis engine, place-and-route engine, or both have introduced errors? Or do you have a mixture of all of these situations?

Hello Sir,

We have investigated this and discovered that the message "Critical Warning: Ignored Power-Up Level option on the following nodes ---" is a side effect of a synthesis bug. This bug occurs if there is a State Machine Synthesis with the following conditions:

1. The State Machine is specified in VHDL.
2. The State Machine inferred by Integrated Synthesis (map) does NOT have a reset signal in it.
3. State Machine Processing is set to either Auto or One-Hot
4. Software version x.y is being used.

If your state machine has a reset in it you can ignore this message. Your design will provide expected results on the board. If your state machine does not have a reset you will need to add a reset to it, else it will not work on the board.

This bug exists in software version x.y only. It has been fixed in version x.y service pack 1 which will be released ...

**Figure 4** In this posting from [fpga-faq.com](http://fpga-faq.com), the problem stems from aggressive optimizations on the tool's part.

---

Answering all these questions requires a different approach. Software simulations offer visibility into the design but are slow. Verifying the design in hardware is fast but provides limited visibility into the internals of the system. So the answer is to create an environment in which software and hardware representations of the design can coexist. In other words, each functional block in the design should be able to reside in the software domain, the hardware realm, or both.

For example, consider a design comprising, say, 100 functional blocks. Some of these blocks could be your own internally developed, proven IP from previous projects; some could be IP from third-party vendors; and some would be your new “secret sauce” to differentiate your design from all others. Now, suppose that you could immediately move the gate-level representations of any known-good blocks, such as your internally developed IP and trusted third-party IP cores, for example, into the same type of FPGA you are targeting for your real-world design. Also assume that you could now verify these blocks in conjunction with the rest of the design running in your software simulator of choice. Right from the start, you have dramatically speeded your verification runs.

Now, as you verify each of the new blocks at the RTL or behavioral level in the context of the full-chip design, you could move the synthesized/gate-level equivalent of each verified block into the physical FPGA. As soon as any problems manifest themselves, you could repeat the verification run with the RTL version of the suspicious block, resident in the simulation world, running in parallel with the gate-level version in the physical FPGA. A software application could—on the fly—compare the signals from the peripher-

ies of these blocks, along with any designated signals inside the blocks. This combination of conventional simulation with physical hardware and an appropriate debugging environment would make it possible to detect, isolate, identify, and resolve bugs, no matter where they originated in the design flow.

Leading FPGA vendors are continuing to innovate and produce larger, more complex chips that are quickly landing in applications that would have been ASIC-only just a few years ago. This fact means that lots of designers are on a steep FPGA-design learning curve, probably with limited resources and time. For FPGA vendors and tool suppliers, this situation presents a great opportunity to simplify and streamline development, thereby converting occasional users into lifelong customers. **EDN**

## AUTHOR'S BIOGRAPHY



*Chris Schalick is the founder and chief technology officer of GateRocket, which specializes in FPGA-design and -debugging technology. He has 20 years of experience in modular-system design and behavioral-system modeling for consumer and industrial equipment. Schalick has held senior engineering positions for raster-imaging, data-networking, and semiconductor-test-equipment companies and was instrumental in delivering products to market at Teradyne, Tenor Networks, Packet Engines, and Cabletron Systems. He holds a bachelor's degree in electrical engineering from the Massachusetts Institute of Technology (Cambridge, MA).*

```
I want a FF in my XYZ project to be '0' at power-up;
be later set by an external input; and never subse-
quently reset. I placed an SRFF with S to the exter-
nal input; clock to the clock; and R to ground. I
assumed it would power-up at '0' but I got analysis
and synthesis errors:

"Info: Power-up level of register inst is not speci-
fied -- using
Unspecified power-up level" and "output pins are stuck
at VCC or GND" - so I created an assignment:

set_instance_assignment -name POWER_UP_LEVEL LOW -to
inst

I was surprised it needed this, but it seems to work.
That sorted, I placed an inverter between the Q output
of the SRFF and an external output pin - I want the FF
to drive an external active-low signal.

This doesn't work in the simulator or the real sili-
con: the output is permanently asserted. But, if I
connect another output directly to q - so I have both
q and /q output pins - then it works!

I'm guessing that place / route software can't invert
the signal between the ... FF and output pin, so it
needs to use another ... cell just for the inverter, but
something goes wrong with the optimization.

Am I warm? Is this a known bug? Is there some option I
need to set, or must I dedicate an output pin for the
unwanted true q - just to make the /q output work??

I'm using software version x.y Web Edition.
```

**Figure 5** A posting from fpga-faq.com discusses a place-and-route tool that makes an arbitrary choice that causes the silicon do something odd and unexpected.