

Running an FPGA at ASIC speed

TAKE ADVANTAGE OF PROGRAMMABLE HARDWARE TO MAXIMIZE ASIC-PROTOTYPE SUCCESS.

The hardware-emulation team at Texas Instruments (www.ti.com) is getting word about its next project: developing a prototype of the modem of a baseband-processor ASIC for next-generation wireless handsets. The prototype is necessary to help the software team test its code using real hardware as soon as possible, because the processor itself won't be ready for another 12 to 14 months. Getting this head start on software development is key to the product schedule, because the software effort represents most of the overall engineering effort.

After hearing more details, the team leader knows he is in for a challenge. Testing the software requires an actual air interface, so the prototype needs to run at the same speed as the modem in the final ASIC. By itself, that part is not daunting; the team leader has lots of experience using FPGAs from several vendors to build prototypes that run at the same speed as the final ASICs. He estimates that the prototype will take multiple FPGAs to implement; even the smallest partition of the design that he would want to fit into a single device is larger than the largest FPGA available. Two other requirements, in combination with the performance requirement, present the team with its biggest challenges. First, the team must complete the prototype in little more than four months. Second, the RTL description of the ASIC is coming from another team, so the hardware-emulation team cannot change it, making partitioning and debugging difficult.

Thus began the emulation project for the modem of Texas Instruments' OMAPV2230 (Open Multimedia Applications Platform) digital-baseband and applications processor for advanced 3G handsets. Although the team had considerable experience developing multi-FPGA designs under deadline, the size of this design represented a new milestone for team members. This new ground, combined with extreme time pressure, forced the team to devise new strategies in addition to applying existing ones. Without knowing exactly how long it would take to complete certain portions of the project, the team focused on practices that would shorten task times and ease the overall development effort. What follows are some of the team's best practices and strategies for achieving that goal, which derive from its many projects, and examples from the OMAPV2230 modem project.

ASSESSING THE DESIGN

The project began with an assessment of the scope of the work. You'll need an estimate of your design size early on to make decisions about the size and number of FPGAs you'll need. You make this determination by counting pins, memory, and gates; in our experience, that is the typical order in

which resources run out. Don't try to get perfect estimates of how many FPGA gates your design will require. It takes too long, and the risk of failure by underestimating the resource requirements can be catastrophic. Experience is the best way to get an estimate of how many FPGA gates your design will use. You can also get estimates of resource usage by function from FPGA and IP (intellectual-property) vendors; these estimates may vary widely, but it is better to err on the side of getting a worst-case estimate rather than a precise measurement.

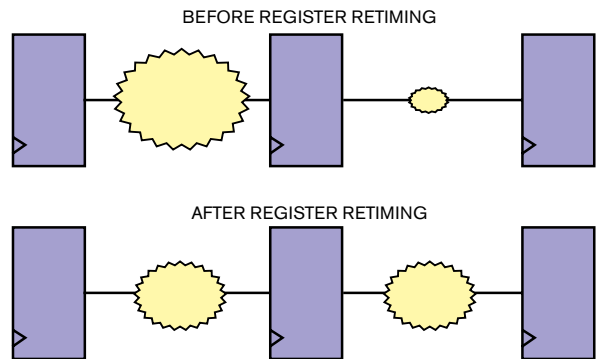


Figure 1 Pipelined, synchronous designs enable your design to benefit more from register retiming.

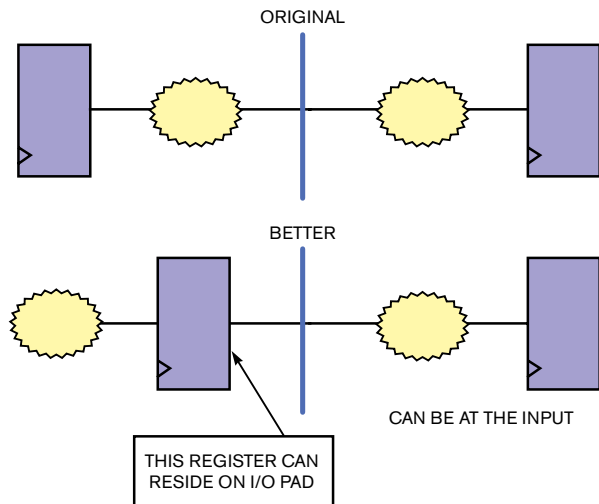


Figure 2 Registering your outputs results in more predictable delays and eases the burden of meeting timing constraints during compilation.

After you get an initial gate-count estimate, double it. By having more than enough FPGA resources on hand, you will ease and more than likely shorten your development effort. Design compilations that are not resource-constrained are generally a lot faster, and the amount of time you will probably save may help you avoid a lot of headaches. The emphasis here is to successfully prototype your design under time pressure, rather than save money on the cost of the programmable logic. Finally, you have a much better chance of extending the usefulness of your prototype platform if you have additional device resources available. (See sidebar “Postmortem” at the Web version of this article at www.edn.com/ms4233 for more information.)

BUILDING AND WRITING RTL

Try to split your design into smaller, unrelated problems for ease of tackling. Start problematic parts of your design, particularly bus interfaces, early. Design your system such that you can exercise and test individual blocks, even if they aren’t yet present in the design. Besides helping out early in the development process when blocks might be available to test while you are still finalizing others, this practice also allows you to

make progress when specific blocks of your design are under revision or otherwise unavailable.

Follow good synchronous-design practices; asynchronous designs that are possible in ASICs because of tight control over timing delays can easily run into trouble in FPGAs. Lots of pipelining, as well as registering all ports, also provides several benefits. First, it breaks combinatorial logic into more easily synthesizable portions. Pipelining also allows easier debugging, because FPGA-verification tools can easily access the inputs and outputs of registers. Finally, it allows more options for optimizing performance by register placement.

Figure 1 shows a pipeline of three registers in which a large amount of combinatorial logic is in place between the first two registers and relatively little combinatorial logic is in place between the second and the third register. By using a logic-synthesis option—register retiming—you can balance the amount of logic on either side, such that the worst-case combinatorial delays on both the input and the output sides of the registers are more equal. Register retiming is a relatively common logic-synthesis option and, in the case of this design flow, the team enabled it by simply turning on the option in the Altera (www.altera.com) Quartus II software’s physical-synthesis settings.

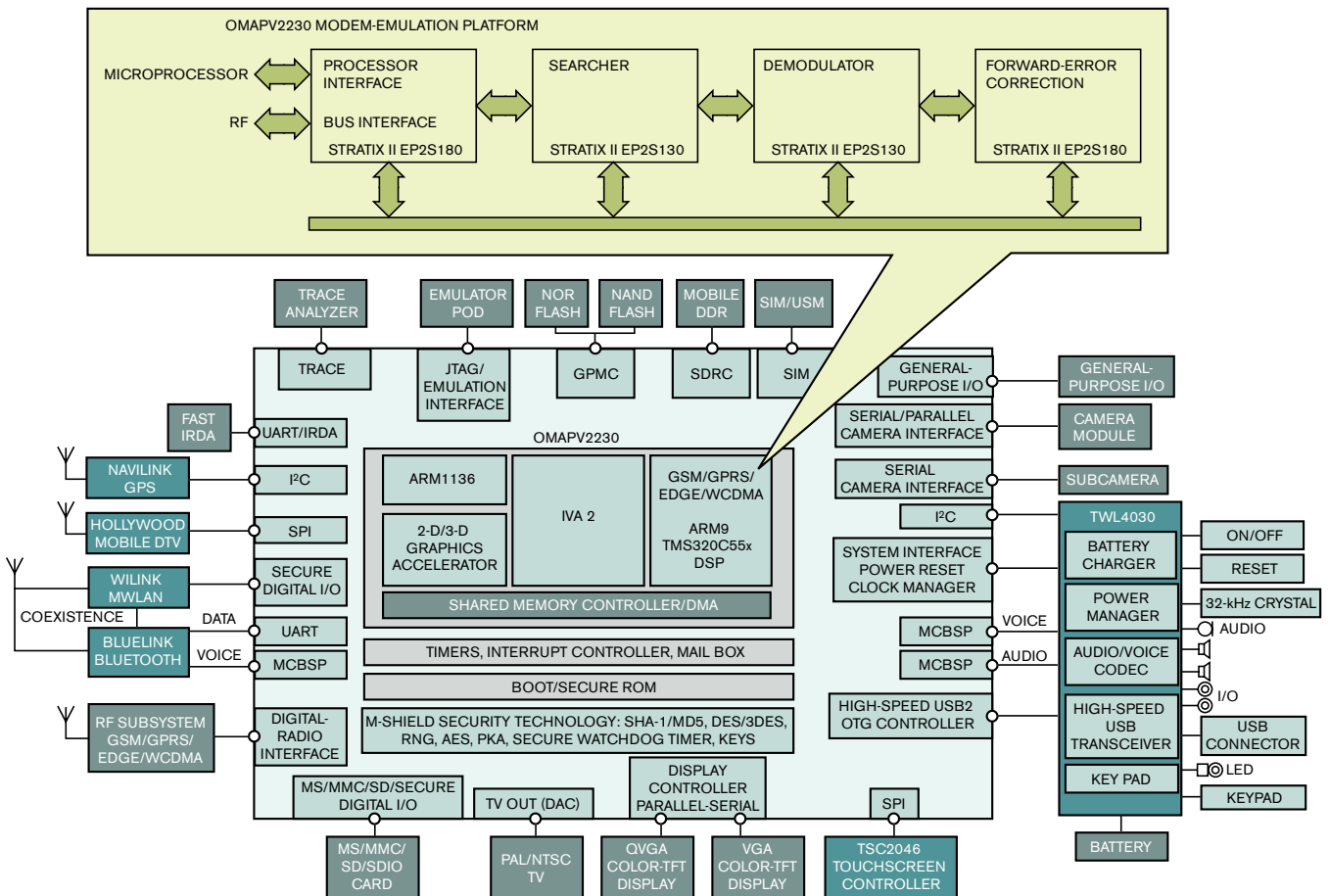


Figure 3 The OMAPV2230 modem-emulation platform partitions functions into Stratix II FPGAs.

Figure 2 shows the output of a register feeding logic to and from one FPGA and then feeding logic and a register into another FPGA. Moving the register to the edge of the device and placing it into an I/O element associated with the pin reduce the logic-synthesis burden and the associated compilation time. Doing so also results in a more predictable delay path from the output of the I/O register to the input of the register in the next device.

If running your FPGA prototype at high speed is your goal, then it helps to code for the ASIC with the FPGA architecture in mind. On the other hand, if you have RTL code that mainly targets an ASIC, then you can get the best performance in your prototype by using FPGAs with a flexible logic building block, such as the Altera Stratix II devices that this project uses. In either case, using the same RTL database for both the ASIC and the FPGA design as much as possible helps minimize differences between the two. The OMAPV2230 project uses the same database with the exception of clocking and memories, which the team instantiated in wrappers.

One of the inevitable differences between the ASIC design and the FPGA design is memory, because the ASIC design includes memory structures that you must build from generic memory blocks in the FPGA design. A good approach to this situation is to code logical-level memories in your ASIC RTL, specifying the exact size you need, rather than using physical memories. The FPGA-synthesis tools can more efficiently map logical-level memories into the FPGA, providing better performance and resource usage. It is worth the extra level of hierarchy, even though you ultimately must map the memories into a physical memory for the ASIC. Also, it helps to standardize on positive- or negative-enabled memories within the ASIC RTL and then add inverters as necessary at the wrapper level. At the Web version of this article at www.edn.com/ms4233, the VHDL code in **Listing 1** illustrates the common practice of using physical memories in an ASIC-design description, and the VHDL in **Listing 2** shows how to change the physical memory to a logical-level memory, and it shows a separate wrapper for the ASIC design.

With an RTL that specifies logical-level memories, you can manually map the ASIC memories to the FPGA memories. The FPGA tools can make it easy to build and instantiate the memory structures you need. The OMAPV2230 project uses the memory megawizard in Quartus II software to aid this instantiation, which automatically generates a VHDL wrapper for the FPGA design. Using instantiated memories also provides the additional benefit of enabling the use of the Quartus II software's in-system memory-content-editing capability, which allows the user to capture and update the content of memories independently of the system clock—a valuable asset during debugging.

When mapping memories, it helps to have a variety of memory structures to choose from in the target FPGA. In the case of this modem design, which has lots of memories of approximately 1 kbit each as well as some larger ones, the portfolio of memory-block sizes that the TriMatrix memory architecture in Stratix II devices provides produced the most efficient mapping. You can also achieve a better mapping by taking advantage of the FPGA-memory features. For example, in one of the

MORE AT EDN.COM ▶

+ Go to www.edn.com/ms4233 and click on Feedback Loop to post a comment on this article.

projects, the RTL includes bit-enabled memories. The memories in the target FPGA include hardware support for byte-enabled memories. After consulting with the ASIC team, we determined that byte-enabled memories would meet the need, and, by making this modification, we achieved a four-times-more-efficient mapping of the memory structures into the FPGA.

PARTITIONING THE FPGAs

Most of the team's prototyping projects require the entire FPGA design to run at the same speed as the ASIC, so the first consideration in partitioning designs is performance. The OMAPV2230 modem project required hardware-based verification to truly test the software with an actual RF interface and device. Using simulated data was impractical, because it was easier to use a real RF interface than try to generate all the data that would represent all of the cases we needed to test. To achieve a real RF interface, the prototype would need to run at the same speed as the modem in the final ASIC.

The next major considerations in partitioning the design are minimizing I/O connections between devices and monitoring memory usage in each device. These goals are often complementary to the performance goal, because minimizing on-chip/off-chip delays contributes to higher performance. You can do quick bottom-up compilations of your RTL blocks in the FPGA to get an idea of their logic and memory usage and I/O interactions. Automated partitioning tools are available to do this kind of work, and they are generally good at keeping statistics and running totals to do trade-offs. The team used no partitioning tools in the OMAPV2230 project for a couple of reasons. First, partitioning tools generally require that all the RTL in the project be available and synthesizable; with some projects, you may want to begin partitioning before the RTL is at this stage. Second, the team felt that the best quality of results would come from partitioning the design by hand with an understanding of the RTL, working first and foremost toward the performance goal. If the goal was not to get to speed, partitioning tools would have made the work easier.

You can also partition for ease of debugging. For example, if you are concerned about an interface, you can place it at the edge of a device with the interface feeding I/O pins, which makes it easy to monitor. In this case, the team partitioned the demodulator and forward-error correction into separate devices to get better visibility into their interaction and to enable the software team to access their interaction with a simple logic analyzer. **Figure 3** shows the modem-emulation platform, indicating the four Stratix II FPGAs and the role of the modem in the overall OMAPV2230 device.

Reserve some of your I/O pins for debugging if possible; the team's target was 20%. Many FPGA tools allow you to route internal signals out to spare I/O pins, and, in this case, the SignalProbe feature in Altera's Quartus II software served this need well, especially because of the limitation that the team not change the RTL. If you can't reserve these pins, don't worry; FPGA tools such as SignalTap allow you to monitor internal signals and bring them out through JTAG pins. SignalTap also played a significant role in extending the value of the platform.

Target FPGA usage should be 30 to 40%, which is not always

possible, especially in this case, in which the team is pushing the capacity of even the largest available device. In general, the closer you get, the easier it will be for you. This statement is not to say that you can't effectively use more or all of the FPGA resources; you certainly can. But designs that are not resource-constrained are easier for the FPGA-design software to handle and therefore faster to compile. For the amount of recompiling you'll be doing, you'll want to keep compilation times short; doing so can save lots of time in the long run, which is more valuable than saving some money on the cost of the FPGAs. Also, having lots of available resources makes it easier to get timing closure.

The OMAPV2230 project required devices with the most pins and logic resources, so the team chose Altera's Stratix II family to take advantage of the EP2S180 device. There was some risk in this decision because the EP2S180s were not available at the time the team started the project but were slated for release about three months later. A slip in that release date would have seriously jeopardized the project, but, in the meantime, the team had enough information and design support for the devices to make progress, including package pinouts to build boards and compilation support in the Quartus II software.

OVERENGINEERING THE BOARD

You can begin your board design as soon as you know which FPGAs and packages you will be using. Lay out board traces for as many of your FPGA I/O pins as possible. In this case, the team made board traces for every I/O pin and made generous use of Mictor connectors. Having the flexibility to probe every pin provides a great deal of debugging freedom.

Overestimate your power budget and put in a much larger power supply than you think you will need. You may find yourself clocking your FPGAs at higher than expected speeds or adding significant amounts of logic to your design in this or future projects and thereby drawing more power. The team found pin-compatible power supplies and initially chose ones that were larger than needed. In that way, the team had the option of using a smaller one when it came time to make many copies of the board.

For projects that require multiple FPGAs, use the same size and package device if possible. It's easier to remember one set of device characteristics, such as the amount of resources, features, clocks, and resets going to the same pins. Using the

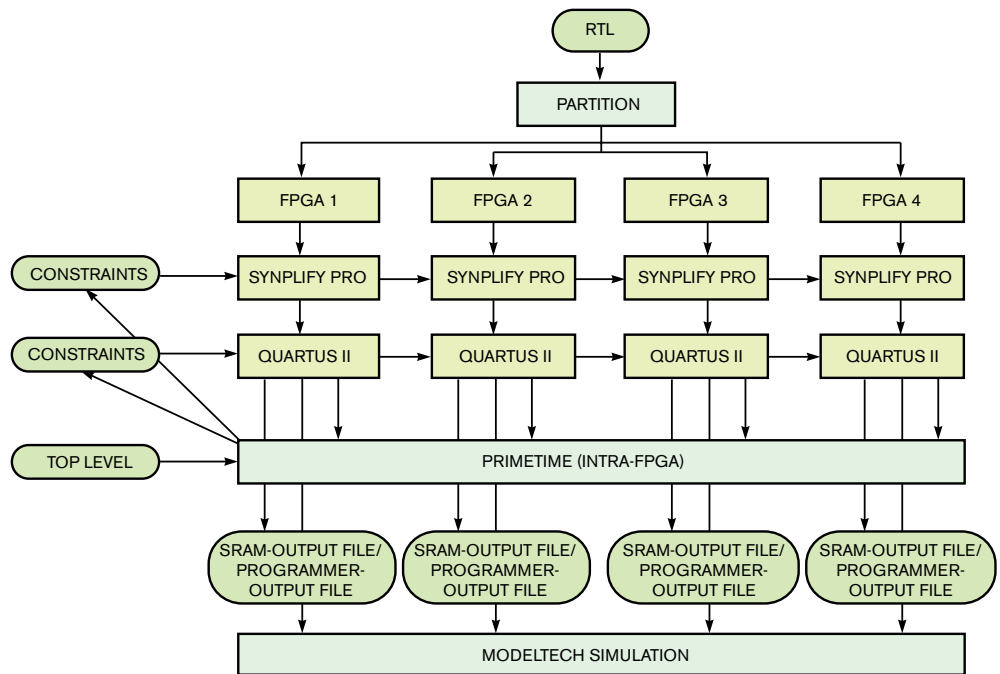


Figure 4 The TI hardware-emulation team relied on this tool flow to develop the OMAPV2230 emulation platform.

same devices also simplifies writing scripts and inventory control. If you have multiple teams working on the design, it's also a good idea to partition it such that no single FPGA overlays the domain of too many teams. This practice minimizes the impact that a single FPGA design has on the overall team and ideally enables most of the team to make progress even if one of the FPGA designs stalls.

RINSE AND REPEAT

Compiling an FPGA design for these kinds of projects often becomes an iterative process, as you try different tactics and design modifications to reach your performance target or as you focus on successfully completing one part of the design before moving on to another. It helps to develop a comprehensive script that you can run once to manage everything related to your FPGA flow, including choosing a synthesis option, applying compilation settings and performance constraints, extracting timing information, and performing timing analysis on your critical paths.

Compilation times for complex, multi-FPGA designs in the largest devices can take multiple hours and even longer as you place more performance constraints on them. During this time, you still want to make progress, so take steps to reduce compilation times and keep your team productive during these times. For example, eliminating false paths from your performance constraints helps minimize compilation times. With the latest timing-analysis tools from FPGA vendors, such as Altera's TimeQuest, it is now possible to specify multiple paths through multiple nodes and cut false paths from the analysis as easily as in ASIC-grade tools. Do not overconstrain your compilations; performance constraints increase compilation time, and exceeding your performance goal with your

prototype doesn't necessarily provide any benefit.

Take advantage of both a dedicated synthesis tool and the FPGA software's native synthesis. The team used Synplicity's (www.synplicity.com) Synplify Pro as well as the synthesis that Quartus II software provides. You can use the best results from either, or one tool may provide a better description of a warning or error message than the other. In the case of the OMAPV2230, the team used multiple simultaneous compilations to explore paths toward meeting the performance goal of running at the same speed as the ASIC. The team used a dozen synthesis-software licenses on a server farm to perform compilations. Save all of your results and analyze them so you can make the best informed decisions on which actions to pursue. The team generated more than a terabyte of data and so had to enlist the IT group to support the amount of computer time and storage space it needed. **Figure 4** shows the FPGA-development flow, including synthesis, timing analysis, and simulation tools.

For simulation, the ideal situation is a single process for both the FPGA prototype and the ASIC design. To achieve this goal, you must develop the ASIC-simulation environment with the prototype in mind to support the gate-level netlists that the FPGA-design flow generates—for example, Mentor Graphics' (www.mentor.com) Modelsim. The team found that simulating the FPGA design is most valuable in the early stages of the project, in particular when debugging interfaces and drivers. As the FPGA design matured, the team relied less on simulation in favor of using the actual hardware platform.

In the case of the OMAPV2230, the team simulated by using a testbench with the gate-level netlist that the FPGA-design flow generated and a top-level netlist to connect the multiple FPGAs. However, because of the long runtime, the team used this method for only the most elementary tests. In the end, the team relied on the hardware to provide the best verification; the team tested the design with the same interface and the same test equipment as the final ASIC. **EDN**

AUTHORS' BIOGRAPHIES

Edwin C Park is a staff engineer and member of the group technical staff at Texas Instruments, where he is responsible for advanced prototyping and architecture of 3G modems. He holds master's and bachelor's degrees in electrical engineering and a bachelor's degree in economics from Rice University (Houston). He enjoys gardening and hiking.

Martin S Won is a senior member of the technical staff at Altera Corp with more than 16 years of experience with programmable logic and programmable-logic applications. He joined Altera in August 1990 as an applications engineer and has held various positions in his career, during which he founded Altera's customer-training program and managed several other technical and marketing programs. His articles have appeared in many industry publications. Won holds a bachelor's degree in electrical engineering from the University of California—Santa Barbara, and his personal interests include reading, writing, hiking, and volleyball.
