By Troy Scott

Legacy FPGA Designs Can Be Migrated to Achieve Better Performance

Common migration obstacles can be overcome through careful planning, examination of both source code and constraints, IP substations, and more.

Much attention is paid to advanced field-programmable-gate-array (FPGA) design methods. As a result, it's easy to overlook the engineering projects that involve the maintenance, upgrade, or migration of existing, legacy FPGA designs to a modern FPGA device. One of the key reasons to retarget a legacy silicon solution to a modern FPGA is to achieve better quality of results (QoR)--namely, increased throughput speed. The current mainstream process technology--130 nm--will provide high-performance solutions with internal frequencies of up to 200 MHz. Within a 90-nm device, 500-MHz internal frequencies are possible.

The availability of new, advanced, nonvolatile technology also is causing designers to consider FPGA devices that offer instant-on capabilities via on-chip Flash and SRAM. Such devices also provide both security and reprogrammability. Of course, it can be expensive to maintain and replace legacy devices with the same-generation process technology as the original. Low-cost FPGA devices, which are designed specifically to address higher-volume product applications, are often a superior alternative.

The efficiency of modern FPGA solutions also is very attractive to system engineers. Some of the attributes that have made it easier to migrate and retarget to FPGAs include smaller die size, embedded application-specific-integrated-circuit (ASIC) blocks, and system security. For example, shrinking die size helps to significantly decrease overall system size. Second-generation 130-nm devices can be as much as 83% smaller than the previous 180-nm device family.

Embedded ASIC blocks, such as DSP, PLL/DLL, and the I/O interfaces of an FPGA, provide high-performance solutions. At the same time, they leave plenty of generic programmable fabric for custom functionality. The DSP blocks within FPGAs enable the parallel processing of common operations like multiplies, accumulates, or multiply/accumulate algorithms. These operations would otherwise require multiple serial instructions executed by a general-purpose DSP device. Specialized hardware functions, such as PLLs/DLLs, DDR I/O interfaces, embedded memory, and multi-standard I/O buffers, ease the design-in of critical system features.

In addition, security is emerging as a key system consideration to prevent hardware IP from being copied and thwart reverse-engineering attempts. Modern FPGAs provide on-chip, nonvolatile key storage to support the decryption of encrypted bit streams.

Finally, end-of-life conditions may force a designer to consider replacement hardware. Modern FPGAs are often ideal replacements for end-of-life FPGAs, CPLDs, and ASSPs like PCI controllers and physical-layer interfaces. A single-chip FPGA solution on the printed-circuit board is attractive, as it eliminates the need for an additional configuration device. Another issue is the European Union's Restriction of Hazardous Substances (RoHS) directive, which is forcing system engineers to adopt modern lead-free devices.

CASE STUDY

Modern FPGA-device features make migration very attractive. After all, a rich variety of functionality is integrated on one device. Modern FPGA architectures are a hybrid of programmable elements, as shown in the simplified block diagram of a LatticeECP2 FPGA (see Figure 1).

The bulk of an FPGA fabric is comprised of programmable function units (PFUs). These units consist of lookup tables (LUTs) and registers. PFUs can be programmed to represent any multi-input function and--optionally--synchronous behavior. They also can act as RAM or ROM mode memories. From a design-migration perspective, the functionality that occupied this portion of the legacy device usually originates as logic synthesized from RTL. This logic is the most straightforward to migrate.

Surrounding the central FPGA programming fabric is a programmable I/O ring. This ring commonly supports a variety of popular signal-interfacing standards including SSTL, PCI, and LVDS. From a migration point of view, re-implementing these buffers may require some amount of constraint substitution and I/O planning in order to account for the target device's buffer ring.

Modern FPGA Architecture (LatticeECP2)

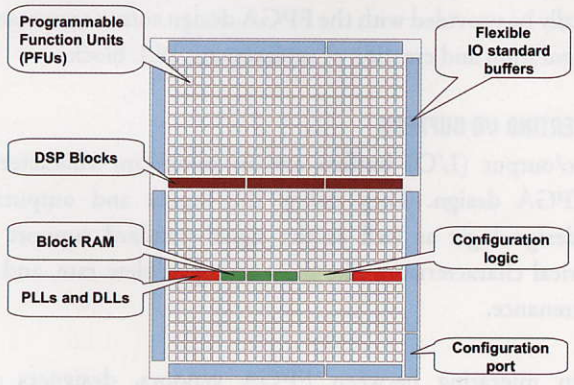

Figure 1: This graphic depicts a modern FPGA architecture.

A common characteristic of most modern FPGA architectures is the rows of embedded, high-performance specialty functions for memory or DSP applications. The embedded-RAM rows are comprised of pre-engineered high-performance blocks that can be configured in a variety of modes from ROMs to FIFOs. Typically, migrating embedded blocks will entail substituting macro-level components. Those components are produced by a module or core generator utility, which is provided in the FPGA design software.

To provide a high-performance clocking solution for the FPGA device, many devices include PLLs or DLLs. They can create clock dividers or multipliers and even tune I/O timing. Usually, the migration of these embedded blocks will require careful substitution using similar library elements and options of the target FPGA device.

It's likely that FPGA vendors will continue to pack a lot of specialized silicon features in and around the generic programmable LUT and register logic, which is traditionally considered the centerpiece of an FPGA. The conversion guidelines that follow detail the major considerations related to design migration into a modern FPGA device. Three classes of conversion guidelines are examined:

1. Mapping design constraints
2. Converting device-specific modules or IP cores including vendor library elements
3. HDL coding style for FPGAs

First, consider the issues involved in mapping design constraints. After RTL coding style, the biggest influence on the size and performance of an FPGA or ASIC is probably user-defined design constraints. The examples that follow may need to be accounted for when moving between FPGA vendors or from an ASIC to an FPGA.

Project-wide constraints may define timing exceptions, which cause timing-driven place-and-route and static-timing analysis (STA) to block or exclude certain signals. Some common cases include asynchronous paths, reset paths, read-during-write paths of RAMs, and interclock domain paths.

In addition, design component or signal-specific design constraints are common. They should be examined to determine if they would be beneficial in the new device implementation. For example, synthesis-optimization directives might be written into RTL source to direct the encoding of state machines or hierarchy directives. They could then flatten or retain branches of the design. Often, there will be equivalent directives if one is using a new synthesis tool for an FPGA project. For example, the Synopsys DC command "set_flatten" can be applied to a block to control the output hierarchy. In Synplicity Synplify, the attribute "syn_hier" is the equivalent control.

Timing constraints are commonly captured as part of a constraint or script file that directs place and route and static timing analysis. Clock frequency objectives, clock relationships, and I/O timing like setup and hold are defined as well. These types of constraints are so commonplace that all modern vendor systems will provide a syntax to declare these attributes. Recently, the Synopsys Design Constraints (SDC) format has emerged in the FPGA industry as a de-facto standard to describe "design intent" for synthesis, clocking, timing, power, test, and environmental and operating conditions.

Out of necessity, a legacy design may have timing and location constraints relative to names derived by logic synthesis or physical elements, such as LUTs. Those constraints simply won't exist in the new target design. The reason for them should be understood and accounted for by the new design constraints.

Signal placement is an important consideration for the sake of both the PCB layout and the signal flow around the FPGA. As a result, I/O planning is an important task in the new device design. The designer should anticipate the organization and signal standards supported by the target device. Modern FPGAs commonly provide both single-ended and differential interface standards. But they may require that certain signals be placed into a particular set of package banks. Fortunately, automatic placement options within FPGA design software allow design rules to be placed and checked quickly--often before the internal logic details are fully defined.

If possible, false paths written at the register transfer level should certainly be brought forward into the new design. Unfortunately, most false paths are identified much later in the design flow. They turn up as part of the routine of examining post-route critical paths to determine if they are truly bottlenecks. Timing exceptions are written in terms of gate-level physical elements that may not exist when the design is re-implemented. For best results, designers must recognize the logical paths in the RTL source and define them for synthesis in the new design. The automatic extraction of false paths from RTL is a powerful analysis capability that's available from some EDA tools.

CONVERTING MEMORIES

Memory functions can be implemented into FPGAs in a distributed style across multiple PFUs. They also may target higher-performance embedded memory blocks. As a result, designers should consider the following when migrating memory blocks:

- What is the capacity and organization of the original function? Can it be represented in the target device?
- What modes and features of the particular memory function are required? Will additional LUT logic be required for decoding or to produce status flags?
- If the original memory block was a component instance, what port connections were used? Is the function of each port equivalent to the target device? Subtle differences between memory blocks, such as the polarity of control signals, could lead to bugs in the design.

Fortunately, all major FPGA vendors provide some type of module or core generator GUI to ease the configuration and creation of new memory blocks.

CONVERTING PLLS

It has become commonplace to have PLL and DLL functionality integrated into FPGAs. This functionality also provides key features for clock generation. When migrating PLL blocks from one vendor to another, the designer should consider the following:

- What are the specifications for the PLL? Common attributes will include operating frequency, phase control, and the duty cycle of the output.
- How many PLLs are available in the target device?
- What vendor-specific blocks are available? For example, the PLL functionality in many Xilinx FPGAs is provided by a DCM/DLL instead of a true analog PLL.

As with memory blocks, a module or core generator GUI will typically be provided with the FPGA design software to ease the configuration and creation of replacement PLL blocks.

CONVERTING I/O BUFFERS

Input/output (I/O) buffers are an important characteristic of FPGA design. They define the inputs and outputs of the design logic as well as the signal-standard support and electrical characteristics like drive strength, slew rate, and bus maintenance.

When migrating between FPGA vendors, designers may encounter vendor-specific buffer component instances embedded in the source code. Depending on the target, they'll need to replace or remove them from the code before logic synthesis. In legacy Xilinx designs, for example, it's not unusual to find buffer types to identify inputs, outputs, bidirectional, and global clocks [author's claim]. In general, Altera and Lattice don't require buffer components in source code. It's common to rely on logic synthesis to infer them. Any desired global signals and buffer configurations can be identified with signal attributes within source code or a separate preference file.

Bank and I/O planning is an important consideration when migrating to an FPGA. Modern FPGAs group I/Os into banks that support certain signal standards--for example, those that provide nine sysIO buffer banks. Each has its own supply voltage and two voltage references, allowing each bank to be completely independent. Top and bottom banks provide buffer pairs of single-ended outputs only. In contrast, buffer pairs on the left and right banks of the device support both differential and single-ended outputs. Those outputs are important for LVDS interfaces. To mimic the original pad/signal placement of the legacy package, designers need to anticipate signal-standard support by location.

CONVERTING DSP BLOCKS

DSP functionality, such as multiply, accumulate, MACs, and any related pipelining, could use general LUT-based logic. Depending on the architectural features, they also could use a combination of LUTs and embedded DSP functions within an FPGA. Fortunately, most of the legacy FPGAs that include limited DSP capabilities based on simple multiplier/adder modules will benefit from the industry trend toward DSP blocks with more functionality.

To migrate an embedded DSP block, the designer must be aware that feature support can vary from vendor to vendor. He or she must understand what functions are available and whether additional LUT logic will be required. For example, the Xilinx

Spartan-3 and Virtex devices provide an embedded multiplier block for multiplication functions only. In contrast, other FPGA devices support multiply accumulate, multiply add/subtract, and multiply add/subtract sum as well as pipeline options.

HDL CODING-STYLE CONSIDERATIONS

The migration of HDL designs--especially those written for an ASIC device--requires some attention to ensure that the RTL is as FPGA-friendly as possible. First of all, one must avoid asynchronous designs. Synchronous designs can be transferred far more reliably into an FPGA, given the high-fanout, low-skew clocking structures that are common in most devices. Keep clock skew to a minimum by using the FPGA's dedicated global clock lines for timing signals. This approach becomes particularly important when clock fan-out is high.

Secondly, avoid using logical signals to drive set/reset inputs. Doing so will make a simultaneous set/reset of all registers difficult. Instead, use the global set/reset routing provided by most FPGAs. Similarly, the global tri-state net--not generic routing--should be used for global three-state.

Exercise caution when transferring data between clock domains. When going from an ASIC to an FPGA, the minimum skew that's needed between two clocks (so that data can be re-synchronized from one to the other) may change. Also, avoid adding delay to signal paths by inserting gates, buffers, or other logic. The associated propagation delays depend on the silicon's physical properties. They can vary significantly once a design is transferred from ASICs to FPGAs.

The designer also must consider the benefits of the Case versus If-Then-Else style of sequential logic. If-Then-Else constructs can infer a priority tree and make a design more complex than necessary. For best I/O timing characteristics, such as clock-to-output, designs should target FPGA I/O registers instead of general-purpose PFU registers. Gated clocks are generally bad in FPGA designs. Typically, they're used as a power-saving measure in an ASIC design. If possible, use secondary clock structures to partition synchronous logic.

In addition, it's best to use the state-machine design and encoding styles that are recommended by FPGA-synthesis-tool style guides. They will produce the most reliable and best-performing state machines. Finally, examine the clock organization of the target FPGA device, as it will likely be more restrictive than that of an ASIC. Compared to an ASIC, high-performance designs may require additional pipelining in an FPGA.

DESIGN EXAMPLE

As a working example, consider a recent conversion performed by Lattice applications personnel. Their goal was to compare the performance benefit of a modern FPGA device over one from an earlier generation. The LatticeECP2 FPGA device features an embedded DSP block with rich functionality. The schematic illustrates an 80 TAP, 14-bit FIR design (see Figure 2). Outlines indicate multiply and adder functions that will be targeted to the embedded DSP block.

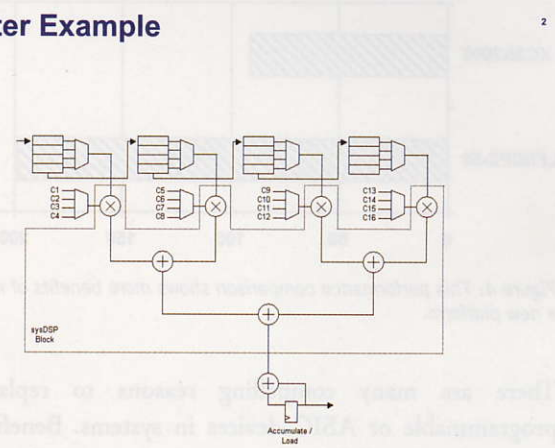

Figure 2: Here is the schematic for an 80-TAP, 14-bit FIR design.

The design was originally implemented into the Spartan-3 XC3S2000 device, which features 5120 CLBs. For the migration, a device of comparable capacity was chosen--the ECP2-50. It provides about 6000 PFUs. When performing a migration, the designer can usually scan the resource summaries near the beginning of each vendor's datasheet to find a match. Many times, a vendor also will publish recommendations per a competitive vendor.

To illustrate the area benefit of migrating to the new device, the design summary reported by the Spartan's design mapper report was compared to that of the LatticeECP2. Just over 1660 slices are required by the XC3S2000 to accommodate the synthesized logic. In contrast, 1496 slices are required by the ECP2-50--about 11% fewer than the XC3S1000. In this case, many adders have been absorbed by the sysDSP blocks and freed general LUT logic within slices.

To illustrate the performance benefit of migrating to the new ASIC platform, the fMAX results reported by the Spartan's static-timing-analysis report were compared to the results for the LatticeECP2. Given this particular FIR-filter design, a 117% performance gain was demonstrated over the legacy device (see Figures 3 and 4).

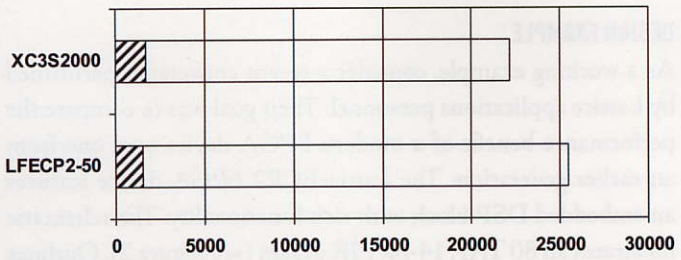

Figure 3: The benefits of migrating to a new platform can be seen in this SLICE utilization comparison.

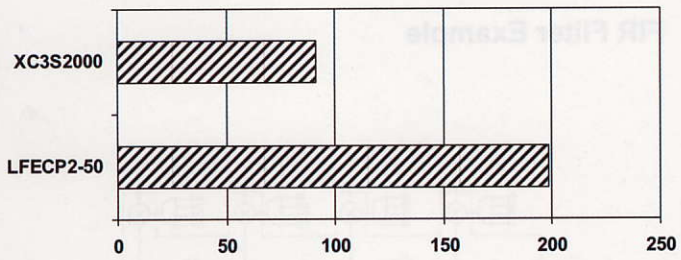

Figure 4: This performance comparison shows more benefits of migrating to a new platform.

There are many compelling reasons to replace legacy programmable or ASIC devices in systems. Benefits include better silicon availability, lower cost, better performance, and more efficient device utilization. Given the function-specific hardware in many modern FPGAs, such as embedded DSP, memories, and PLL/DLLs, the performance of a legacy system can be significantly upgraded. Finally, system security is emerging more and more as a necessary requirement. The ability to prevent snooping of bit streams is therefore an attractive feature.

To avoid obstacles when tackling a migration project, both source code and constraints should be examined. In some cases, they should be translated to the new environment. Vendor-specific blocks like embedded RAM and DSP or PLLs and DLLs will require substitution. Synthesis and place-and-route constraints will need to be translated. In most cases, there will be equivalents across FPGA vendors and even between ASIC and FPGA synthesis tools. Coding style might be worth revisiting in a legacy design--especially if the design originally targeted an ASIC. ◆

Troy Scott has been helping to design, document, QA, and promote EDA products for about 14 years. He is a Product Marketing Engineer at Lattice Semiconductor Corp. Scott welcomes feedback and can be reached at troy.scott@latticesemi.com.

What is the Right Kind of MPSoC?

The last few years has seen a considerable increase in the use of processors in embedded systems design. It seems that every interesting SoC has at least one embedded processor on it, and more and more of them are embedding multiple control and data-plane processors to handle a growing number of functions that once were implemented in hardware. On the desktop and in large servers, we have seen the processor trains that seeked to offer increased performance by increasing clock frequency be derailed in the last couple of years, having hit brick walls of heat, power dissipation and technology complexity, to be replaced by multicore architectures. It's clear that multicore and multiprocessor architectures are becoming ubiquitous.

This of course begs the question – what kind of MPSoC architecture is best? Like any good question, there are multiple answers – and the right answer depends on the application. Many deeply embedded applications in consumer products can best utilize multiprocessor technology with an asymmetric multiprocessing (AMP) approach. In this approach, tasks or their key functions are known in advance, processors can be configured and extended to match. For more general purpose computing at the high end desktop and server, where new applications cannot be predicted in advance and tasks may need to migrate from processor to processor, symmetric multiprocessing (SMP) sharing a large coherent memory may be a better approach.

No matter what the application and architecture, DAC 2006 in San Francisco can help designers and managers make the right choices. Running from July 24 to 28, DAC is offering several interesting sessions dealing with MPSoC and processor-centric design and verification, including ones on processor and communication centric SoC design, MPSoC design methodologies and applications, advanced topics in processor and system verification, and a special session on MPSoC design tools. In addition, the DAC exhibit floor will show offerings from several IP companies, and a host of new and old, large and small EDA companies who have tools that can help in MPSoC design, integration and verification. A visit to DAC will be worthwhile for everyone involved in MPSoC.

Grant Martin, Chief Scientist, Tensilica Inc.