SpyGlass® Application in an FPGA to ASIC Conversion Flow

Bernard Murphy
Sanjay Churiwala
INTRODUCTION

Mapping from a field programmable gate array (FPGA) to an application specific IC (ASIC) is subject to some limitations. In this White Paper, we identify some of the most common limitations in this mapping process and show how the use of Atrenta’s GuideWare™ methodologies and SpyGlass® product family help the designer quickly identify and address these limitations.

FPGAs are a perfect platform for prototyping functionality or for handling logic which may need to be upgraded in the field. In low volumes, they may also be the most cost-effective solution for many applications. However, in high volumes, the unit cost associated with an ASIC can be significantly more attractive. Therefore, for high volume designs, it is quite common to develop a design on an FPGA platform and, when functionality has stabilized, map that same design to an ASIC implementation.

Since considerable effort has typically been put into the design and verification phase, it is clearly desirable to use a translation from the proven FPGA implementation that is as automatic as possible. By far the best way to do this, in most cases, is to start from the source RTL description for the design and then re-map that, through synthesis, to the target ASIC library. This approach “almost” entirely solves the problem of mapping between two potentially very different target technologies.

A designer who takes these strategies into account when designing for the FPGA can be confident they will run into few problems when ready to map. Conversely, if the designer knows what limitations apply, they can identify those areas in the FPGA RTL that will require re-design before mapping. Some of the areas that require special attention are:

- Testability – not normally an issue in FPGA design but critically important for ASICs
- Clock distribution – handled very differently in FPGA and ASIC architectures
- Resets – often implied in FPGA, but not in ASICs
- Use of special macros
- ASIC tool compatibility
- Packaging – cost of changing a package is exorbitant, migration is easier if packages chosen for the FPGA and ASIC are common

NAMING CONVENTIONS

FPGA design flows are largely delivered by one vendor, and therefore tend to have a high level of compatibility between the various tools used in the design process. ASIC design flows typically use tools from multiple vendors, and sometimes even in-house tools. In an ideal world, all these tools would share common limitations. In practice, they do not. As a result, most ASIC flows will need to impose naming convention limitations to avoid potential downstream problems. It is advisable to find out what your target ASIC vendor’s convention is and stick to that. If you have not yet chosen a vendor, use the safest assumption you can – names should contain only alphanumeric characters and underscores and should start with an alpha.

You should further avoid using duplicate names (e.g. the same name for a net and an instance), names which share a large number of initial characters, and names that are identical apart from case (if using Verilog). These styles are legal and will be supported by your simulator but may cause problems in the back-end ASIC implementation flow.

Finally, you should check with your ASIC vendor to make sure there are no special reserved names or reserved name prefixes. These names should not be used in your design.

How SpyGlass can help
SpyGlass provides all of the above checks as standard checks in the OpenMORE™ policy:
For example:
o Standard naming convention checks are supported by the *Name rules. The default settings for these rules should work fine for most ASIC vendors. You can change the conventions if your vendor has more or less restrictive requirements.

o Duplicate names are checked and unique names can be checked. In the latter case, the default requires all names be unique in the first 8 characters, again a safe default for most vendors. The default can be changed if needed.

o Reserved names can be checked. By default, this will check for Verilog keywords if your design is VHDL and VHDL keywords if your design is Verilog. We advise that you check with your ASIC vendor for any special keyword restrictions and program those into the rule check (a simple parameter change in SpyGlass).

SYNTHESESIZABILITY

FPGA synthesis tools can synthesize a wider subset of the Verilog and VHDL languages than is supported by standard ASIC synthesis tools. For example, FPGA synthesis tools will infer RAMs automatically from 2-dimensional array declarations, where ASIC synthesis tools will not. ASIC synthesis will generally explode these into a netlist of one-bit flip-flops, almost certainly not what you wanted!

How SpyGlass can help
SpyGlass provides a complete set of synthesizability checks in the Synthesis template from BaseSpyGlass. These can be windowed down to cover just those checks that are applicable to the target ASIC synthesis tool. Design Compiler (DC) is the most likely target synthesis tool.

HARD-CODED CELL AND MACRO INSTANCES

FPGA design tools allow the designer to hand-instantiate a number of special cell-types directly in the RTL, each of which must be replaced, possibly with some local re-design, for map to ASIC. The most common cases are:

o Special I/Os – for example, LVDS, SSTL, etc. The ASIC library will typically have equivalents to these cells, but often with different names and sometimes with different pinouts. It is best to put all I/Os in a separate pad-ring module, instantiated in the top-level of the design. This will greatly simplify conversion.

o Clock distribution macros – these macros generate divided and delayed clocks and are optimized specifically to the FPGA architecture. In general the ASIC library will not have equivalent macros. The clocking structure in these cases will need to be re-designed to accomplish similar behavior using PLLs or DLLs. It is best to encapsulate any such function in a wrapper cell so it can be replaced with logic constructed around a similar ASIC function.

o Analog cells (ADC, PLL, etc.) often require special handling for analog power and ground. In an FPGA, these requirements are handled automatically. In an ASIC, they are not. Analog cells must be identified and managed carefully in physical design.

o Primitive macros – FPGA designers are allowed to hand-instantiate primitive cells built around leaf-level logic cells. These will not have an equivalent in the ASIC library and should be avoided.

o Generated macro functions – FPGA vendors provide function generators for a wide variety of cores. For example, Xilinx offers XBLOX and LogiCores which generate arithmetic functions, DSP functions, memories and many others. The ASIC library will not typically have an equivalent set of generators. If possible, use of these generated functions should be avoided. If they must be used, again encapsulate them in wrapper instances so each can be replaced with logic constructed around a similar ASIC function.

o Memories – FPGAs and ASICs implement memories (from a designer viewpoint) in very different ways. Since memories cannot be avoided, encapsulate them in wrapper instances so each can be replace with logic constructed around a similar ASIC memory.

How SpyGlass can help
All cells of this type will appear as black-boxes (cells with no RTL description) in the design. SpyGlass provides several methods to catalog and investigate black boxes. This information can most easily be seen
through the Audit-RTL function. In addition, a few other examples where SpyGlass checks will be useful in this area are:

- Black-boxes in the clock tree (typically clock distribution or buffer macros) will be identified as clock sources in the Audit-Structure report and will be reported as informational rules (with schematic highlighting) in Atrenta Console.
- The OpenMORE policy has a rule to disallow specific named macros. The list of names can be specified through a parameter. This should be set to cover those macros you know should not be used in the design.
- The WrapInstance_ML rule can be used to check for black-box instance wrapping.

**CLOCKS**

Clocks critically affect behavior and timing of a design and are handled in very different ways in FPGA and ASIC architectures. Here are some items to watch out for:

- Your clock tree should not be buffered. Clock tree balancing will be handled quite differently in standard ASIC methodologies than FPGA design, which have limited clock routing resources. It is particularly important for this reason that you do not attempt to “fix” timing by inserting buffers into the clock tree.
- Do not infer gated clocks. These can behave correctly in an FPGA (although FPGA vendors advise against this style) but are dangerous in an ASIC because the clock can glitch when the hold condition switches. Instead, infer enabled flops which will map to structures which avoid the glitching (either a muxed input, or a more complex latched gating structure which suppresses glitches).
- Use a minimum number of external clocks. Generate all other clocks, where feasible, internally. Phase delays between external clocks which may be tolerated in the FPGA implementation may cause problems in some cases in the ASIC implementation.
- Generate all internal clocks in a single module/entity if possible. Avoid generating/gating clocks locally in the design. This helps when it come to balancing the clock tree in the ASIC.
- Avoid using ripple clock dividers to generate multiple divisions of clocks. This is a common technique in FPGA design. This design style causes problems for ASIC design tools and is therefore error-prone. It also introduces phase delay into the clock, increasing problems in balancing the clock tree. It is better to use a DLL or digital clock macro which can be replaced with a PLL in the ASIC implementation.

**How SpyGlass can help**

SpyGlass provides multiple methods to analyze clocks as per the Clocks Methodology and here are a few clock related examples:

- Buffers in the clock tree will be flagged under the OpenMORE policy.
- Similarly, SpyGlass will flag gated clocks with rules from the same policy.
- While it is difficult to define what is a “minimum number” of external clocks for any given application, SpyGlass can help by reporting all clock sources in the Audit-Structure report.
- Another rule in the OpenMORE policy will indicate if clocks are generated or gated in more than one place in the design.
o SpyGlass provides multiple methods to visualize clock trees both in reports and in a schematic view of the design. These are useful in helping you make sure you will not run into problems when the design is mapped.
o There are rules that can be used to check for ripple clock dividers as well.

RESETS

Although resets are nominally only applied on startup and occasionally thereafter, ensuring a device is correctly reset is often non-trivial and must be carefully managed in ASIC designs. Some of the issues that should be considered are:

o ASIC architectures do not include default power-on reset logic, and therefore you must explicitly include reset logic in your design.
o Your reset tree should not be buffered. ASIC methodologies aim to balance reset trees just as they aim to balance clock trees. You should make sure that your reset trees contain no buffer cells for this reason.
o Use a minimum number of external resets. Generate all other resets, where feasible, internally - in a single module/entity if possible. This helps in balancing the reset trees in the ASIC design.
o Be careful about the possibility of releasing an asynchronous reset at the same time as an active clock edge. This may never be a problem in your FPGA, but could become a problem in the ASIC, given different loading and routing delays. If using an asynchronous reset, consider synchronizing it to the master clock to avoid this problem. One safe scheme is to synchronize the reset to the negative edge of the clock, ensuring that the active reset edge will be well separated from the active – presumed positive – clock edge.

How SpyGlass can help

SpyGlass provides many methods to analyze resets and here are a few of them:

o SpyGlass can check that all flops are asynchronously reset under the STARC policy. If you want to follow a synchronous reset style then you can forbid asynchronous resets using the AvoidAsync rule under the OpenMORE policy.
o While it is difficult to define what is a “minimum number” of external resets for any given application, SpyGlass can help by reporting all reset sources in the Audit-Structure report.
o In the OpenMORE policy there is a rule that will indicate if resets are generated or gated in more than one place in the design.
o SpyGlass provides multiple methods to visualize reset trees both in reports and in a schematic view of the design. These are useful in helping you check if you will run into problems when the design is mapped.
o The clock-reset policy can be used to check for synchronized asynchronous resets.

DESIGN FOR TEST

FPGA vendors test their devices for manufacturing defects before shipping them to customers, so this kind of test is usually not an issue in FPGA design. However it is a significant issue in ASIC design. The ASIC itself must be tested for manufacturing defects and that requires automated insertion of test logic. Inserting this logic creates some further restrictions on design style:

o Do not use asynchronous loops (combinational loops).
o Do not use clocks as data signals, or data as clock signals (with the exception of clock dividers). These structures will undermine automatic test pattern generation.
Avoid latches unless you are using them purely for re-timing. In all other cases, if you must use latches, provide a testmode signal which will make them transparent for test configurations.

Do not use tristate buses internally. Tristates can cause a variety of problems in test, including possible shorts in scan mode. Use muxes internally.

All clocks and set and reset signals should be directly controllable in test mode.

**How SpyGlass can help**

All of these checks are covered under the SpyGlass-DFT methodology. In addition, SpyGlass will give an estimate of total attainable test coverage on the design, which can be useful in assessing just how much trouble you may be looking at before you start the conversion!

**TIMING-DEPENDENT LOGIC**

You should avoid timing-dependent structures at all costs. Detailed timing dependencies that may work in an FPGA are unlikely to work the same way in an ASIC, where placement, routing and interconnect delays may be significantly different in the final implementation. Examples of these kinds of structures are:

- Asynchronous loops (combinational loops).
- Pulse/glitch generators (e.g., A AND (NOT A)).
- Circuits which depend on clock duty cycle.
- Circuits which attempt to match delays through separate paths. Often this is accomplished through chains of invertors. If these are represented in the RTL, they will be optimized out, so no action is necessary. If you have instantiated FPGA invertors or buffers, these will be flagged in the black-box check mentioned earlier in this document. Avoid instantiating buffers or invertors.
- Do not generate clocks from combinational feedback. Timing is likely to be very different in the ASIC implementation.
- Avoid creating races from flop output to reset or set on the same flop. Equally avoid possible races to clock and enable on an enabled flop.
- Some FPGAs provide latch elements. It is sometimes tempting to use these for time borrowing. They should be avoided however because timing in these case may not map identically in the ASIC implementation.

In most cases there is no one-to-one substitution for these structures. You should re-think your design to avoid using timing-dependent logic.

**How SpyGlass can help**

As part of the GuideWare methodology, SpyGlass provides several checks that are relevant to this area, for example:

- Combinational loop detection.
- There is support in BaseSpyGlass to check for clocks driven from combinational feedback.
- Similarly, BaseSpyGlass also will check for reset feedback races.
OTHER DESIGN TECHNIQUES

- Register the outputs from the FPGA. Also register outputs from major blocks which are likely to fall in separate partitions in the ASIC implementation (a safe guideline is blocks of ~200,000 equivalent gates). This ensures that timing at the output of the ASIC will be predictable and maximum timing margin is reserved for the possibility of long (high delay) routes between blocks.
- Avoid using SR latches and flops. Circuit behavior is unpredictable and will generally not match ASIC behavior when set and reset are asserted at the same time. Use only set or reset, not both, on any flop or latch.
- Do not create your own state elements, i.e., do not create latches from combinational logic, or flips-flops from back-to-back latches. Most ASIC design tools will not recognize these structures correctly.
- Many FPGAs provide the ability to initialize memories during programming. You should not use this feature since it is difficult to replicate in the ASIC implementation. You should provide an explicit method to clear the memory on reset.
- Avoid using FPGA specific features, e.g., relying on a DONE signal when programming has completed.
- Do not infer memories (using 2-dimensional arrays). They will be very poorly mapped in the ASIC implementation. Always use instantiated memories.
- Synchronize asynchronous inputs using, for example, 2 flip-flops to control metastability issues. Failing to do this can lead to random synchronization failures.
- Do not connect unused pins to NC (no connect). This is an FPGA-specific feature. The way this feature will be handled in an FPGA is unpredictable and cannot be emulated in an ASIC. It is better to tie all unused pins to a defined fixed state.
- FSMs should always have a reset state and a default state. In reset, there is no guarantee the FSM will come up in the same way in the ASIC as in the FPGA. A reset ensures you can force the same starting state. Similarly, if you do not specify all states, you should have a defined default state. If you do not, glitchy behavior may differ between the two implementations.
- You should also not generate circuits that contain feed-throughs (a direct connection from an input port to an output port on a block). These are often problematic in ASIC environments.

How SpyGlass can help
SpyGlass provides several checks that are relevant to this area and some of them are:

- There are rules to check that all top-level outputs are registered. These rules are also able to check block level outputs, where blocks are identified by a pragma in the RTL code.
- The rules will also check for SR-style combinational loops. They will also flag an attempt to generate latches from logic. A particular rule in the DFT methodology can be used to check for flops with both set and reset used.
- There is support in the OpenMORE policy that can be parameterized to forbid use of signal names like DONE and NC.
- The Clocks methodology includes a rule to check for clock/reset feedback races. The Timing methodology includes a check for clock/enable races.
- The OpenMORE policy will check that all FSMs have a defined default state, if required. The reset rules mentioned earlier will check that each synchronous FSM has a reset.

SUMMARY

Mapping an FPGA design to an ASIC can be a problem-free experience if the FPGA was designed from the outset with a re-map in mind. If you did not take this precaution, you may find that so many changes are required to make the FPGA RTL ASIC-compliant that you must effectively re-design and re-verify the RTL. In this case, the FPGA implementation becomes little more than an existence proof that a working implementation can be built.

Avoiding these pitfalls requires keeping in mind a large number of constraints, a challenging task for a designer pre-occupied with meeting an immediate design goal. It is much simpler in this case to automatically verify compliance to the required set of rules using SpyGlass with a pre-defined set of checks.
These checks can be run frequently with minimal designer input to verify that problems are kept to a minimum. Moreover, when you are ready to re-map, SpyGlass can be used to quickly summarize all problem areas which will require some level of design attention. Once a design is ported to an ASIC for implementation at a given process node, the methods described in this White Paper will also allow further shrinking to more advanced nodes later.

Thus, by following the Atrenta GuideWare methodology using the SpyGlass family of tools, you can achieve portability of the design from the FPGA implementation to the ASIC implementation, and from one process node to the next. Atrenta’s GuideWare methodologies contain a comprehensive set of checks and qualified templates that allow for maximal portability of the design.

ABOUT ATRENTA

Atrenta is the leading provider of Early Design Closure® solutions to radically improve design efficiency throughout the IC design flow. Customers benefit from Atrenta tools and methodologies to capture design intent, explore implementation alternatives, validate RTL and optimize designs early, before expensive and time-consuming detailed implementation. With over 150 customers, including the world’s top 10 semiconductor companies, Atrenta provides the most comprehensive solution in the industry for Early Design Closure. For more information, visit www.atrenta.com. Atrenta, Right from the Start!