XILINX®

Adapting ASIC Designs for Use with Spartan FPGAs

XAPP119 July 20, 1998 (Version 0.5)   Application Note by Kim Goldblatt

Summary

Spartan FPGAs are an exciting, new alternative for implementing digital designs that, previously, would have employed ASIC[1] technology. Pre-existing ASIC intellectual property[2] can be adapted for use with Spartan devices by following a straightforward procedure. Each step of the procedure is explained in detail. Guidelines show how an ASIC design, in the form of an RTL-level HDL[3] file, can be revised to take full advantage of the Spartan series' capabilities, thereby achieving efficient, high-performance implementations.

Xilinx Family

Spartan and SpartanXL

Introduction

The Xilinx Spartan Series, offers high density, rich functionality, and high performance at unprecedented low cost. For many new design projects, these FPGAs are not only a viable alternative to mask gate arrays, but *the* superior choice. The Spartan series, consisting of the 5 V Spartan and 3.3V SpartanXL families, provides numerous important capabilities that add significant value to a design:

- Spartan devices are available in a range of densities from 5,000 to 40,000 system gates. This range can accommodate the majority of today's ASIC designs.
- Spartan designs commonly run at system clock frequencies up to 80 MHz. Carefully placed and well-routed designs can operate at frequencies in excess of 100 MHz. This level of performance is more than adequate for most ASIC designs. Dedicated global clock lines keep clock signals clean and reliable from source to destination. Continual process improvements ensure that Spartan devices will continue to meet advanced system performance requirements well into the future.
- Spartan FPGAs are fully programmable, providing flexibility unknown to ASICs. As a result, Spartan designs can be modified quickly, at little or no cost, throughout a product's development cycle. Designs can even be upgraded in the field.
- Compared with ASICs, Spartan FPGAs are rich in specialized architectural features that not only simplify the task of designing, but also make meeting functional and performance requirements easy. Global reset, global three-state, JTAG, distributed RAM, and dual port operation are a few such features.

- For designers accustomed to using ASICs, perhaps the most significant of the Spartan series advantages is the exceptionally low pricing, made possible by an advanced fabrication process and a streamlined test procedure. For high volume production, the price of Spartan FPGAs is comparable to ASICs.

Given the Spartan series' considerable benefits, sold at prices that compare favorably to ASICs, many designers will prefer an FPGA solution for their next generation of products. In this context, they will want to make use of pre-existing ASIC intellectual property for their new Spartan designs. Such IP can readily be adapted to suit Spartan by following the straightforward procedure presented in this note. In this way, the design will best be able to utilize the Spartan series' capabilities, thereby achieving efficient, high performance implementations. Specific information is provided on how to resolve design issues arising from differences in HDL practice, architecture, timing, and with respect to FPGAs and ASICs. Finally, the use of Xilinx LogiCORE and LogiBLOX components as replacements for ASIC library elements is discussed.

ASIC-to-Spartan Adaptation

The procedure for adapting ASIC intellectual property for use with the Spartan series can be divided into two sections. The first section, known as the "ASIC-to-Spartan Adaptation Flow" is shown in Figure 1. The steps include: (I) obtain an ASIC design file coded in RTL-level HDL. Then, revise the HDL file to suit the Spartan series by (II) resolving architectural differences, (III) replacing ASIC

---

1. For the purposes of this application note, ASIC refers to mask gate array.
2. The term intellectual property includes preexisting design material (i.e. legacy code), cores, and vendor ASIC libraries.
3. Two kinds of Hardware Description Language (HDL) are recommended for the ASIC-to-FPGA design adaptation: VHDL and Verilog.

code with Xilinx LogiCORE and LogiBLOX functions, (IV) adding pipeline registers, (V) adjusting local timing and (VI) following the appropriate HDL coding style.

The second section, called the "Synthesis-Implementation Flow", is shown in Figure 2. Essentially, it consists of HDL synthesis followed by a fairly routine implementation (including map, place and route) using the Xilinx development software (version 1.4 or later). Continuing from the ASIC-to-Spartan Adaptation Flow, (VII) select a suitable Spartan device and (VIII) synthesize the revised HDL file. Then, perform (IX) functional simulation, (X) implementation and (XI) timing simulation. If errors are encountered during functional simulation, then revise and re-synthesize the HDL design. If errors occur during timing simulation, either revise and re-synthesize the HDL file or adjust the timing constraints and re-implement the netlist created by the last synthesis. Finally, once simulation is successful, then (XII) create a bitstream out of the design and configure the Spartan device with it. Each of the steps will now be considered in more detail.

ASIC-to-Spartan Adaptation Flow

Step I

Starting with the ASIC-to-Spartan Adaptation Flow (Figure 1), the first step consists of obtaining ASIC intellectual property expressed in Register-Transfer Language. RTL refers to the level of abstraction in HDL code at which designs are described in terms of data storage (i.e. registers) and transformation (i.e. logic). More than 90% of recent ASIC designs are expressed in RTL form using either VHDL or Verilog. ASIC designs created before 1990, prior to HDL's surge in popularity, will generally be available only in schematic or netlist form.

Either VHDL or Verilog is the logical choice for use with the adaptation procedure since both these languages are not only easy to read and modify, but are also supported by a wide range of ASIC and programmable logic development software. Most important of all, RTL-level HDL code is reasonably portable, provided that it is generic (i.e. does not contain references to device- or tool-specific features). This means that synthesis software can optimize and map logic to take good advantage of the target device's unique capabilities. As a result, the synthesis tool automatically accomplishes much of the work associated with adapting the design to the Spartan series, saving significant time and effort.

If the ASIC design is only available in schematic form, it will be necessary to create an RTL version before going ahead with the adaptation procedure. Some tools provide an option to convert schematics to a "verilog" file, consisting of gate-level code. Such a file, when synthesized for an FPGA, would result in an inefficient, low-performance implementation; therefore, it is unsuitable for the adaptation procedure.
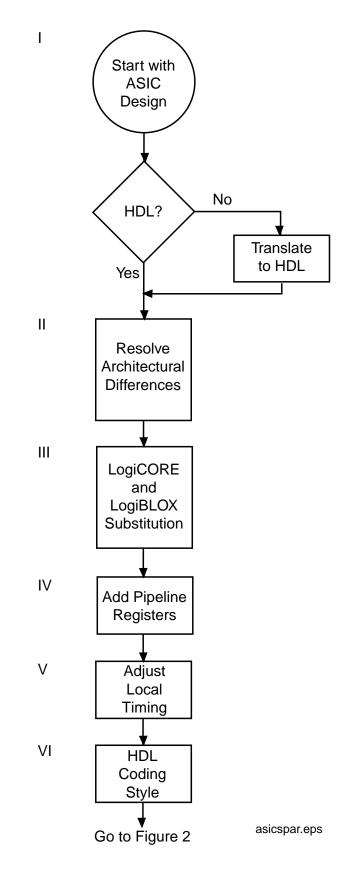

Figure 1: ASIC-to-Spartan Adaptation Flow

Nor does it make sense to use a netlist as a direct input to the adaptation procedure, since library elements or architectural features specific to ASICs are difficult to identify and remove.

Even if the netlist is generic enough that it can be synthesized for a Spartan device, placement and routing would be unable to use the device's logic resources efficiently. This is the same problem described earlier for gate-level "verilog" files. In this case, just as before, an RTL-level HDL file needs to be created.

Step II

Make any necessary corrections to the HDL code to account for architectural differences between the Spartan series and mask gate arrays. The HDL input file is likely to contain instances of library components specific to the ASIC architecture. The synthesis tool targeting a Spartan device will be unable to recognize such instances. Therefore, it is necessary to replace them with code that makes use of comparable functions supported by the Spartan series. See the section entitled "Architectural differences" on page 5.

Step III

Consider using Xilinx LogiBLOX and LogicCORE modules to replace comparable functions in the ASIC design. The Spartan series also supports IP available from a number of partner companies collectively known as AllianceCORE. LogiBLOX macros save time and effort by providing pre-designed, simple functions, such as adders and memories, that are ready for use. The LogiCORE and AllianceCORE modules provide more complex functions. They are specially optimized to make the best possible use of Spartan resources, resulting in efficient logic utilization and high performance. Using cores to replace HDL code where timing is critical helps to ensure successful operation once the design is transferred to Spartan. Cores also conserve logic, permitting larger designs to fit into a given Spartan device. Refer to "LogiCORE and LogiBLOX" on page 7 for more details.

Step IV

Introducing pipelining into the design is one of the most significant actions the designer can take not only to preserve the original ASIC design's performance, but also to eliminate the possibility of timing problems. The total delay from register to register is the sum of the clock-to-out delay of the first register plus the logic delay plus the setup time for the next register. The reciprocal of the register-to-register delay is the maximum clock frequency that can be supported between the registers. The insertion of pipeline registers between layers of logic effectively reduces the register-to-register delay, which, in turn, raises the maximum sustainable clock frequency.

Mask gate array designs can have as many as 30 layers of logic between registers. While this arrangement can be made to work for some ASIC devices, in general, it makes achieving high operating frequencies rather difficult. The fewer the layers of logic between registers, the easier it is to attain high-performance operation. Keeping the number of logic layers under three significantly reduces the possibility that race conditions due to differences in the interconnect delay between the Spartan series and ASICs can occur. As a general rule to be applied when adapting an ASIC design to the Spartan series, every HDL module (i.e. process) should have registered outputs and a clock signal in the sensitivity list.

Note that every pipeline register added to the design increases the overall data latency (i.e. the time it takes to clock data from the input to the output of the system) by one clock cycle. Fortunately, in many designs this latency is not an issue.

Step V

It may also be necessary to address timing problems that arise in cases of unusual or nonstandard design practice. One such example is the case where data synchronized to one clock is re-synchronized to another clock. A complete list is provided in "Timing Considerations" on page 6.

The present step involves a general inspection of the HDL code for poor digital design practices, especially with regard to synchronizing clocks. Where required, HDL code can be revised in accordance with the sound design methodology recommended in "Timing Considerations" on page 6. Timing problems that cannot be identified prior to synthesis will be caught at "Step XI" on page 5 and addressed at that time.

Step VI

HDL designs intended for the Spartan series should comply with the recommended coding practice for FPGAs. Given alternative ways of coding the same logic, some HDL statements work better for Spartan FPGAs, others for ASICs. For example, the "if-else" statement specifies priority encoding whereas the "case" statement specifies parallel behavior. Even though the resulting logical function for both statements is identical, for an FPGA implementation, the "case" statement can result in a faster circuit. For an ASIC implementation, this effect is unlikely to be noticed. Appropriate coding style for Xilinx FPGAs is discussed extensively in the *Xilinx Synthesis and Simulation Guide* (See "References" on page 7). Inspect the HDL design and make sure that it conforms to recommended practice. This step concludes the ASIC-to-Spartan Adaptation Flow.

Synthesis-Implementation Flow

Step VII

Continue with the Synthesis-Implementation Flow, as shown in Figure 2. This step selects a suitable Spartan device to serve as a target for synthesis. To get a general idea which Spartan device has sufficient logic resources to hold the design, count "system gates". This measure provides a reasonably effective comparison between FPGAs and mask gate arrays. The density of ASICs is commonly measured using a "logic gate count", which is the total number of two-input NAND gates (four transistors per gate) in the design. One logic gate is equivalent to two system gates. If the estimate for a given design is 25,000 system gates, then a good choice for target device would be the Spartan XCS30, which, as the part number indicates, has a maximum of 30,000 system gates.

The easiest way to obtain a gate count for a design is to use a synthesis tool that reports logic utilization. Just select a Spartan XCS40 and synthesize the design. Note that a system gate range is specified for each Spartan device. The number of gates available for a given Spartan device actually varies depending on how many of the Configurable Logic Blocks are implemented as memory. While the maximum system gate count specified for Spartan assumes 20% of all CLBs are employed as memory, a higher percentage can make even more gates available. This is because a CLB provides 28 gates for logic (including the H LUT), but 146 gates for RAM (including flip-flops). Therefore, the more memory a design employs, the greater the available number of system gates for a Spartan device.

Generally speaking, the faster the Spartan device, the less impact differences in signal path delays between the two technologies will have on the design's timing. If possible, use the fastest available speed grade (presently -4). As a result, the task of adjusting timing in the HDL code will be easier.

Step VIII

Synthesize the design for the selected device type. It may be necessary to make some adjustments to the HDL code before synthesis is successful. This is especially true if the design has never been run on the synthesis tool before. Make sure all required libraries are accessible. Correct errors in the code and re-synthesize as necessary, until synthesis is successful. In addition to verifying the fit of the design into the Spartan device, most synthesis tools will provide a post-map breakdown of logic utilization.

Choosing the appropriate software tools can greatly facilitate the ASIC-to-Spartan conversion process. Of central importance is the Xilinx development software, which comes in two versions, Alliance and Foundation, both of which perform the map, place, route, simulation, bitstream generation and device programming operations for Xilinx FPGAs and CPLDs. Alliance is designed to be integrated
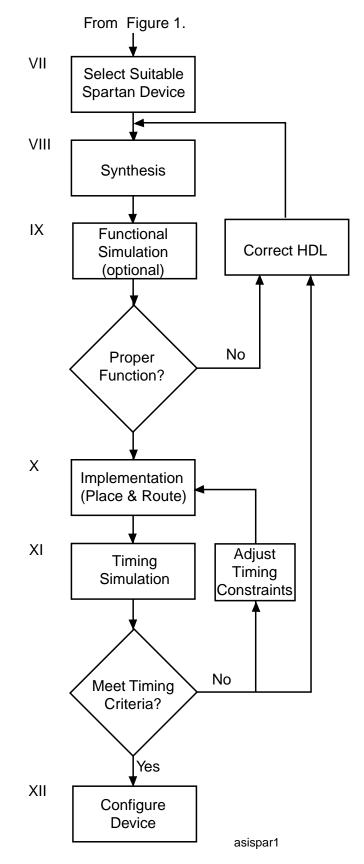

Figure 2: Synthesis-Implementation Flow

with leading EDA environments. Foundation is a complete, ready-to-use solution that includes the Foundation Express synthesis tool as well as a timing simulation tool. Either Alliance or Foundation can be used in conjunction with a variety of popular synthesis tools including: FPGA Express (Synopsys), Synplify (Synplicity), and Leonardo Spectrum (Exemplar). Simulation of HDL synthesis results is becoming an increasingly important part of the design verification process. The Xilinx development software can also be used together with simulation tools available from a number of vendors, including Verilog-XL (Cadence) and ModelsSim (Model Technology, Inc.). Consult the Xilinx sales office nearest you for more information on selecting the appropriate software for FPGA development.

Step IX

Functional simulation, whether performed inside the Xilinx development software or with a third party tool, is optional, but recommended on two counts. First, it verifies that changes to the design (i.e. library substitutions) work as expected. Second, it identifies aspects of the design that still require adaptation to the Spartan series (i.e. ASIC library elements not yet replaced by Spartan library elements). Simulate and revise the HDL code as necessary, then re- synthesize (i.e. go back to Step VII). When the design functions satisfactorily, move on to Step X.

Following synthesis, ASICs typically require customers to perform a SCAN test to detect stuck-at-0 and stuck-at-1 faults. This test is unnecessary for FPGAs which, unlike ASICs, are thoroughly tested before shipping.

Step X

Using the Xilinx development software, select the target Spartan device and implement the design. Be sure to generate a "timing-annotated netlist", which is an EDIF file to with timing information added for the purposes of timing simulation. Check the implementation reports to gauge the success of the implementation. For example, any difficulty accessing libraries will generate an error in the translation report. The map and PAR reports provide information on logic utilization as well expected performance.

Step XI

Check the design's timing. The Timing Analyzer (part of the Xilinx Development Software) performs static timing analysis by comparing implementation results against the timing constraints specified by the designer. The Xilinx software also generates a timing-annotated netlist as input to a timing simulation tool.

Many timing issues will already have been resolved by the addition of pipelines in . Nevertheless, it is worthwhile taking this opportunity to check for any race conditions that may have escaped detection prior to synthesis. Refer to " for a description of questionable design practices that lead to

race conditions and how they can be avoided. Where necessary, correct the HDL code, go back to and re-synthesize the design. Continue with the synthesize-implement-simulate loop until all timing issues have been resolved.

Another strategy for addressing timing problems is to adjust the timing required in the User Constraint File (UCF). Then, loop directly back and re-implement. (Go back to).

Step XII

Run the Bitstream Compiler, part of the Xilinx development software, and configure the target Spartan device.

Architectural differences

Significant differences in architecture between the Spartan series and mask gate arrays give rise to differences in the way features are implemented. When adapting an ASIC design to suit the Spartan series, it is necessary to remove any references to ASIC-specific features and replace them with FPGA equivalents. In some cases, this will mean substituting code in the HDL file, in other cases, it will be a matter of selecting the appropriate switch in the Xilinx development software. Refer to the *Xilinx Synthesis and Simulation Design Guide* and the Spartan data sheet for more information. A summary of the architectural differences between Spartan and mask gate arrays follows:

Functional Building Blocks

Mask gate arrays contain a sea of a relatively undifferentiated NAND gates, whereas the Spartan series employs more highly-structured functional building blocks known as Configurable Logic Blocks (CLBs) and Input/Output Blocks (IOBs). Once a target Spartan device is selected, the synthesis tool automatically maps generic RTL to suit an architecture based on CLBs and IOBs. Nevertheless, how effectively CLBs and IOBs can be utilized may depend on how the design is coded in HDL. An example of this is shown in the next section.

Configurable Logic Blocks

The CLB has two four-input LUTs (Look-Up Tables), one three-input LUT, and two flip-flops. The LUTs are used to implement either logic functions or memory, the outputs of which can be registered using the flip-flops. Anywhere from 100 to 784 CLBs are organized as an array within the Spartan device.

As an example of the way HDL coding impacts logic utilization and speed, consider state machine design. Among the methods available for designing state machine logic, the one-hot encoding, which use one flip-flop per state, is particularly well suited to the limited fan-in and abundance of flip-flops characteristic of the Spartan series architecture. Converting state machine designs employing other meth-

ods (i.e. Moore and Mealy) to one-hot encoding can result in more efficient, higher performance implementations See "Accelerate FPGA Macros with One-Hot Approach" in the *Xilinx Synthesis and Simulation Design Guide* for more details.

Input/Output Blocks

IOBs connect between the I/O pins and routing within Spartan. There are anywhere from 80 to 224 IOBs arranged around the periphery of a single Spartan device. One IOB contains two flip-flops, which can be used to register the input and output signals. There are a number of other IOB options described in the Spartan data sheet. An IOB is selected for use by adding input, output, or I/O buffers to signals where they enter or leave the Spartan device. While these buffers can be instantiated in the HDL code, it is more common to select an option in the synthesis tool to add them when synthesizing the top-level file of the design hierarchy.

Global Nets

Spartan devices have eight dedicated global nets for clock distribution. Low capacitance makes these lines effective for high frequency, high fan-out clock signals. Global nets can be selected by inserting the generic global buffers BUFG in the paths of clock signals. The BUFG symbol can be instantiated in HDL code. Many synthesis tools will automatically assign HDL signals named CLOCK or CLK to global nets. Make sure clock signals from the original ASIC design use global nets in Spartan. This will uphold performance and, at the same time, reduce the chance that race conditions can occur. Finally, the design should not employ more than eight clock signals.

Global Set/Reset and Three State Control

The Global Set/Reset feature (GSR) is a dedicated net that provides asynchronous reset (or set) to all registers in a Spartan device. The Global Three-State Control (GTS) is a dedicated net that puts all FPGA outputs into a high impedance state. The low capacitance of these global nets ensures that registers will be reset (or outputs will go into a high impedance state) simultaneously. Both GSR and GTS are instantiated in HDL code using the same STARTBUF (or STARTUP) component. When adapting a design to the Spartan series, be sure to remove any ASIC-specific code associated with global reset, set and three-state operations. Replace them with STARTBUF.

Spartan also supports the synchronous reset of registers using general purpose routing. If the fan-out of the reset signal is relatively low, a synchronous reset (or set) may provide faster, more reliable timing. However, for a reset (set) of the entire Spartan device, the GSR net is the best choice. Using GSR, a reset signal with a high fan-out is better protected from changes in interconnect delay that can occur when moving from an ASIC to Spartan.

Resistors on I/Os

Either pull-up or pull-down resistors can be connected to the I/O pins using a programmable option in the synthesis software. For example, Synopsys FPGA Express uses the set_pad_type command.

Memory

Spartan supports distributed RAM, which means that the two four-input LUTs of any CLB can be used to implement a variety of memory types, including synchronous RAM, ROM, Dual Ports, and FIFOs. This approach is quite different from the majority of mask gate arrays for which RAM is synthesized from generic NAND gates. When adapting ASIC code for Spartan, it important to replace all gate-level memory functions with either LogiBLOX or LogiCOREs. In this way, an efficient implementation of memory in Spartan is assured.

JTAG

JTAG is implemented in ASICs using intellectual property that builds the required circuits out of the logic available. For Spartan, JTAG circuits are actually built into the silicon. JTAG is selected for operation simply by applying the appropriate signals to dedicated pins on the Spartan device. When preparing an ASIC design for use with Spartan, JTAG implementation files are no longer necessary and should be removed.

Timing Considerations

The importance of sound digital design methodology cannot be over-emphasized. Synchronous designs can be transferred from ASICs to Spartan more reliably than asynchronous designs. Provided that the clock cycle can absorb any variation in interconnect delay when a design is transferred from an ASIC to a Spartan device, then no race conditions can occur. In this regard, asynchronous designs are more likely to experience timing problems.

Following is a brief summary of the timing anomalies that can occur when deficient ASIC designs are transferred to FPGAs. Using good digital design techniques can prevent such problems.

Wherever logic lies between clocked registers, verify that the access time of the source register plus the delay through the logic plus the setup time to the destination register is less than the clock period.

Skew associated with gated clock signals in Spartan may differ from that in an ASIC. As a result, race conditions may occur. Avoid inserting logic in the path of clock signals. For the same reason, do not use logic signals for the purpose of synchronization. Keep clock skew to a minimum by using the Spartan series dedicated global clock lines for timing signals. This becomes particularly important when clock fan-out is high.

Synchronizing registers on the trailing as well as the rising edge of the clock is not a good idea. This practice effectively reduces the time between synchronized events to half the clock period, which may not be sufficient to accommodate variability in the interconnect delay.

Avoid using logical signals to drive reset (and set) lines. This makes a simultaneous reset (set) of all registers difficult to achieve. Use the GSR net that Spartan provides. Similarly, the GTS net, not generic routing, should be used for global three-state.

Exercise caution when using more than one clock. The minimum skew necessary between two clocks so that data can be re-synchronized from one to the other may change when going from an ASIC to Spartan.

Avoid adding delay to signal paths by inserting gates, buffers or other logic. The associated propagation delays depend on the physical properties of the silicon and can vary significantly once a design is transferred from ASICs to FPGAs. Note that non-inverting buffers inserted within logic (not connected to I/Os) are trimmed by the Xilinx development software.

Avoid pulse generators, one-shots, glitch detectors, oscillators, and ripple-counters, since these functions also rely on the physical properties of the silicon to generate timing.

Avoid metastability by ensuring that the design meets all setup and hold time requirements. Setup and hold time requirements may be longer for FPGAs than for ASICs. A signal presented in sufficient time to meet the smaller ASIC setup or hold time may not be stable within the FPGA's setup-to- hold window.

LogiCORE and LogiBLOX

Xilinx offers two forms of IP for use with Spartan, LogiBLOX and LogiCORE, both of which can be instantiated as components in an HDL design. LogiBLOX components are provided as part of the Xilinx development software. They are parameterizable functions of relatively low complexity, such as adder/subtractors, counters, decoders, multiplexers, registers, and memories. LogiCORE components are created using the Xilinx CORE Generator, which is distributed in the form of a complementary CD-ROM. They consist of relatively complex, parameterizable functions, such as correlators, filters, transforms, adder/subtractors, multipliers, and memories.

LogiBLOX and LogiCORE components can be used to add or replace HDL code as part of the ASIC-to-Spartan adaptation process. Ready-to-use LogiBLOX components save time and effort creating original HDL code. For example, pipeline registers can be added using a simple component instantiation (see "Step IV" on page 3). They are also a convenient way to instantiate memory.

LogiCORE components have the additional advantage of being specially optimized for high performance. Their use ensures efficient logic utilization, high performance and reliable timing. When transferring an ASIC design to Spartan, LogiCORE components can be used where timing is critical. They help keep timing skew down to tolerable levels, thereby preventing race conditions from occurring.

When synthesizable HDL (or a schematic) is lacking, as is often the case for ASIC vendor libraries, replacement with a comparable LogiCORE or logiBLOX component is recommended. If a good match is not available, then it may be necessary to create an HDL component file that describes the function in RTL.

Proprietary ASIC library components may come with behavioral HDL files intended for simulation purposes. Synthesis of this code for the purpose of configuring a Spartan device is not recommended. Even if possible, such synthesis leads to inefficient logic utilization and poor timing.

References

The Spartan Series Datasheet, www.xilinx.com

Xilinx: The Programmable Logic Data Book

Xilinx Synthesis and Simulation Design Guide

Xilinx CORE Solutions Data Book

"FPGA Design Considerations for HardWire Designs", www.xilinx.com

"The Ten Commandments of Excellent Design", Peter Chambers, *Electronic Design*, April 1, 1997, pp. 33-40

"The Ten Commandments of Excellent Design: VHDL Code Examples", Peter Chambers, *Electronic Design*, April 14, 1997, pp. 123-126

Memec Design Services in Mesa, Arizona: This Xilinx-certified design center provides a variety of useful technical services, including the conversion of ASIC designs to FPGAs.