5-2002

ASIC Technology Migrations: A Design Guide for First Pass Success

Marc Edward Royer
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Marc Edward Royer entitled "ASIC Technology Migrations: A Design Guide for First Pass Success." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Don Bouldin, Major Professor

We have read this thesis and recommend its acceptance:

Danny Newport, Gregory Peterson

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Marc Edward Royer entitled “ASIC Technology Migrations: A design guide for first pass success“. I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Don Bouldin

Major Professor

We have read this thesis and recommend its acceptance:

Danny Newport

Gregory Peterson

Accepted for the council:

Dr. Anne Mayhew

Vice Provost and
Dean of Graduate Studies

(Original Signatures are on file in the Graduate Student Services Office)

ASIC Technology Migrations:
A design guide for first pass success

A Thesis
Presented for the
Master of Science
Degree

The University of Tennessee, Knoxville

Marc Edward Royer
May 2002

Dedicated to my family.

Acknowledgements

I would like to thank Dr. Bouldin, Dr. Newport, and Dr. Peterson for agreeing to be on my thesis committee and guiding the development of this thesis.

I would like to thank Erin Miller for her proof-reading effort.

I would also like to thank the original founders of ASIC International, Inc. Vig Sherrill, Mark Goode, Nancy Hahne, and Dan Lincoln for offering me the opportunity to learn and grow as an ASIC designer.

Last, I would like to thank my wife, Melissa, for her love and support throughout my Masters program.

Abstract

This thesis presents a study of Application Specific Integrated Circuit (ASIC) technology migrations. An overview of the design flow methodology used for completing a ASIC design from concept to silicon is presented. The design flow is then augmented with special considerations specifically for ASIC technology migrations. An ASIC technology migration design example, using the special considerations, is preseted. Finally, a summary is presented with considerations regarding future work.

Table of Contents

List of Figures

Chapter 1

Overview

1.1 Introduction

Application Specific Integrated Circuits (ASIC), are integrated circuits designed to perform a specific function. An example of an ASIC would be a communications chip in an internet router designed to route millions of internet data packets per second. An example of an integrated circuit which is not an ASIC, would be an Intel Pentium processor, which is a general purpose microprocessor. When ASICs are designed, they are targeted to specific technologies, described as feature size, which is the size of a single transistor inside the ASIC. ASICs can be classified into two categories: programmable and non-programmable. The programmable ASIC category contains one type of ASIC, a Field Programmable Gate Array (FPGA). There are two types of non-programmable ASICs: masked gate array and standard cell.

This thesis contains a discussion of design methodologies and design examples for ASIC design conversions, or ASIC technology migrations, from one type of ASIC to another. The reasons for migrating from one technology to another as well as the many subtle challenges involved are discussed.

1.2 ASIC Migration Definition

Semiconductor manufacturing technology changes approximately every 18 months. A designer can expect that approximately every 18 months, the feature size of a transistor

will shrink by one-quarter. The current smallest feature size available is 0.13 um. As the technology size decreases, semiconductor manufactures (otherwise known as FAB's) are forced to close down production lines of much older technologies due to lessening demand of chips manufactured in older, larger technologies. Although many applications require the latest and greatest technology, there are many older system designs, still being manufactured today, having long term future demand, which could survive indefinitely using parts manufactured in several year-old technology. Once the FAB's decide to cease production of an older technology the companies, which use products from that particular FAB line, are faced with a problem. It is only a matter of time until their current supply of ASICs run out.

Fortunately, there is a solution to this problem. The solution is an end of life (EOL) ASIC technology migration or conversion. The purpose of an ASIC technology migration is to take a design in an older technology and exactly replicate it, preferably as a drop in replacement part, in a newer, smaller technology.

Fabricating a design using the latest process technology can be very expensive. The Non-Recurring Engineering (NRE) fee to create a set of masks for an ASIC can cost over $1M alone. The NRE fee includes all of the services required to generate a set of masks to be delivered to the FAB. The NRE fee coupled with the fact that the number of first-pass successful ASIC designs is very small makes the barrier of entry to an ASIC, specifically when prototyping a product, very large.

Companies faced with quickly prototying a product will often use an FPGA as their ASIC solution. Because the FPGA is programmable, it allows the designer to reprogram the part quickly, and as many times as needed, if the design attempt fails. If after successful prototyping, the product is released to market, companies look to reduce the cost of the electronic components inside the product as a way to increase profit margin. One way to increase the profit margin of the product is to convert the FPGA to an equivalent gate mask ASIC, a cost reduction ASIC conversion. Since the design has been prototyped the chances for first pass success are greatly increased. Additionally, although the NRE fee is large, if the ASIC is purchased in high enough volumes, the NRE fee in addition to the cost per chip will be lower than the cost of the FPGA in the long term.

An ASIC migration can be very simple or very complex. The complexity of the migration depends highly on the quality and completeness of the original design database as well as the quality of the original design. A high quality, complete database will contain elements such as the captured design in netlist format, a complete set of test vectors, a design specification, and a technology data book. A database missing one or more of the above items drastically increases the complexity of the migration – as well as reduces the chances for design success. Additionally, completely synchronous designs are rather simple to convert, where as asynchronous designs can pose unique challenges.

1.3 Scope of Thesis

This thesis presents ASIC migration design flows applied specifically to ASIC conversions. It details considerations one needs to take into account in order to ensure design success. Chapter 2 contains a discussion of a traditional ASIC design

methodology. Chapter 3 presents detailed design methodologies for EOL and cost reduction conversions. Chapter 4 contains a design example with the application of the design methodologies presented in Chapter 3. Chapter 5 contains conclusions as well as considerations of ideas for further improvement. The contribution of this thesis is to provide a guide to first pass success for ASIC technology mirations.

Chapter 2

Traditional ASIC Design Flow

2.1 System Architecture and Planning

An important first step in developing an ASIC is to develop a complete specification. The process of defining the specification may involve performing detailed analysis of the algorithms to be implemented and performing software simulations to verify the concepts. For example, a bit accurate software model of an image compression/decompression ASIC may be necessary to demonstrate that the algorithms will work well in hardware with an acceptable throughput.

After the specification for the ASIC is defined, a schedule and budget can be estimated. A well-defined specification enables realistic goals for milestones to be established. It also eliminates the temptation to add new features to the design once the development process has started (“feature creep”), since the impact on the schedule can be clearly understood. “Feature creep” is often responsible for unexpected additional development costs and missed delivery dates. A complete specification enables effective planning to be done resulting in better management of resources including people, computers, and tools. It is more cost effective to take the time to develop a sound specification than to begin the ASIC development effort prematurely.

2.2 RTL Coding and Validation

The implementation of a digital ASIC can begin soon after the architecture has been defined. Based on the overall system architecture, a Register Transfer Level (RTL) model is developed with a Hardware Description Language (HDL), such as Verilog or VHDL. Both software languages provide the necessary constructs to model the parallel nature of hardware systems. RTL code emulates the process of transferring data between registers through combinational logic. Memory and other macro cells included on the chip must also be modeled – behavioral models are provided by the macro cell vendor. The result is a behavioral model of the entire ASIC. Figure 2.1 shows a simple Verilog RTL example: a 2-bit positive edge triggered, positive synchronous reset counter.

The behavioral model must be validated to ensure it performs correctly. Model validation is performed by the application of test vectors to the model. Test vectors refer to a sequence of inputs applied to the model, for which an expected response is known. By comparing the expected and actual response of the model, the validity of the RTL model can be determined and corrected as needed. There are many different simulators, the

```
module counter(clk,reset,out);

input clk,reset;
output [1:0] out;

reg [1:0] out;

always @(posedge clk)
  if (reset)
    out <= 2'b00;
  else out <= out + 2'b01;

endmodule
```

Figure 2.1: 2-bit RTL Verilog counter.

Electronic Design Automation (EDA) software tools used to validate RTL models, available. These include Synopsys VCS and VSS, Mentor Graphics ModelSim, Avant! Polaris, and others.

2.3 Synthesis

After the behavioral model has been validated, logic synthesis software is used to map the RTL code to logic gates in the targeted foundry library. The synthesizer looks for specific constructs in the RTL code, and infers registers or combinational logic functions. The resulting "netlist" is composed of registers, combinational logic, interconnects, and macro cell instantiations. The synthesized netlist should be simulated using the same test vectors used to validate the RTL. In addition, formal verification software can be used to compare the RTL implementation to the synthesized result. Any simulation or formal verification errors need to be resolved since this may indicate a problem with the library of hardware components used by the synthesizer, a synthesis error, or improperly written RTL. Additionally, it is important the synthesized design exceeds the final timing requirements, as once the design is physically implemented additional timing delays will be introduced due to the interconnect. It is good design practice to synthesize with additional timing margin, 20% as a rule of thumb, as well as wire load models to ensure that the design will be able to meet timing after place and route. There are several synthesis EDA tools available. These include Synopsys Design Compiler as well as Cadence's BuildGates. Figure 2.2 shows a schematic of the synthesized 2-bit counter used in the previous example.

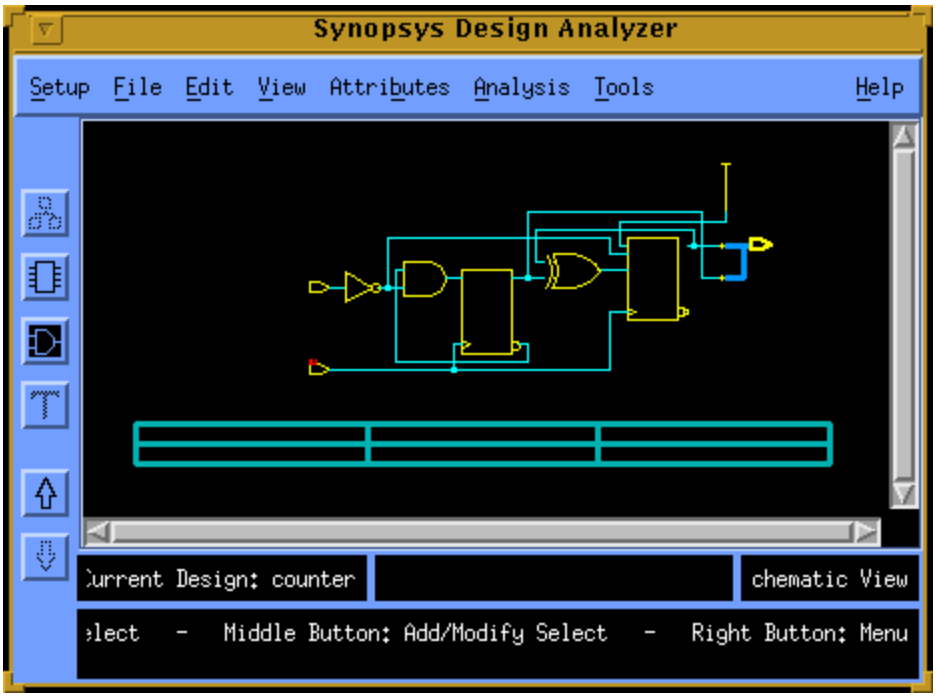

Figure 2.2: 2-bit counter schematic.

2.4 Testability

An important part of the ASIC design process is Design For Test (DFT). After the ASIC is fabricated at the foundry, it needs to be tested to determine if there are internal faults inflicted during the manufacturing process. If any of these tests fail, ASICs, and at times entire wafers, are discarded by the foundry. If the faults are not detected at the foundry, the customer can receive defective parts. Test logic is inserted into the design to enhance controllability and observability of internal nodes. Automatic Test Pattern Generation (ATPG) software is then used, in conjunction with the inserted test logic, to determine the number of faults, or fault coverage, the generated patterns can detect. High fault coverage results in a reduced chance of receiving defective parts.

A design containing macro cells should have additional test logic, such as RAM Built In Self-Test (RAMBIST) for testing of RAMs, inserted to verify the manufacturability of the macro cells. If the design allows sufficient controllability and observability to effectively verify the proper operation of the macro cells, additional test logic may not need to be inserted to test them. Additionally, boundary scan logic (JTAG) should be inserted to accommodate board level testing. Although test logic insertion has some associated overhead, such as an increase in the overall gate count of the design as well as the use of additional I/O resources, the benefit of reducing the risk of receiving bad silicon outweighs the overhead. After the test logic is inserted, it is good design practice to do a formal verification of the pre-test inserted netlist and post-test inserted netlist, with the test logic disabled, to ensure that the insertion of the test logic did not effect the functionality of the design.

2.5 Static Timing Analysis

Static timing analysis (STA) is used to ensure that the design meets the required timing constraints on primary input paths, primary output paths, and register to register paths. In a register to register transfer, although there can be many possible paths, the STA tool finds the longest path, applying worst case parasitics, to ensure that the setup time on the destination register is met. To ensure that the hold requirement on the destination register is met, the STA tool finds the shortest path using best case parasitics. Setup and hold calculations are also performed on the primary input paths. Output required time calculations are performed on primary output paths.

STA can be used to check timing at various points in the design phase. It is most often performed after synthesis (before place and route) using estimated parasitic data, and after place and route using actual parasitic data. However, it is only critical that STA be performed after place and route where actual parasitic data is available.

2.6 Floorplanning

Once the design has been synthesized and all of the test logic has been inserted, the physical design process can begin. The first step in the physical design process is floor planning. Floor planning involves partitioning the design into groupings of cells and placement of the groups as well as macrocells contained in the design on the silicon die. Depending upon the instance count of the design, the design may have to be partitioned such that each of the partitions are placed and routed individually as if each partition is a single, stand alone, design. If the instance count is small enough, the entire design can be placed and routed without partitioning. The estimated instance count of a design is an important item to consider when allocating resources in the design planning stage. Larger chips, which require partitioning, require additional place and route software licenses as well as additional place and route engineers to allow the work to be parallelized.

Other floorplanning considerations are the creation of I/O macros and the power bussing structure. I/O macros contain, at a minimum, the I/O pad and a flip flop. This helps to ensure minimal skew across system interface busses. Additionally, the I/O macros could contain other logic such as the logic to support JTAG. As feature size shrinks and die sizes increase, an important physical design consideration is IR drop – voltage drop

across the die due to insufficient power. Therefore, the power bussing structure must be well planned so that all areas of the die are provided with sufficient power.

2.7 Layout

Once the design is floor planned, place and route can begin. The library components in the netlist are first placed then the interconnect is routed. Traditionally, once the place and route is complete, parasitic data from the place and route tool is back-annotated into the STA tool. If there are any timing violations, an engineering change order (ECO) will need to be completed. To complete an ECO, the netlist and parasitics from the place and route tool are read into a synthesis tool. Design rule violations, such as setup time violations, capacitance violations, and transition time violations among others, are fixed. The updated netlist from the synthesis tool is then read back into the place and route tool (the ECO). Once the updates to the physical database from the synthesis tool are implemented, STA is performed again – if there are still problems the entire loop is repeated.

In deep sub-micron technologies, the amount of delay through interconnect is approaching the amount of delay through the logic gates. Timing closure has become a significant issue in the industry due to this. In other words, what was once a simple process has become extremely complex. Over the past several years, layout tools have been enhanced to provide ”timing driven layout” to help with timing closure in deep sub-micron designs. Using timing driven layout, it is possible to eliminate having to go outside of the place and route tool to perform an ECO loop. Timing driven layout tools have the software engines built into them to perform both STA as well as design rule fixes. The

tools have the ability to fix design rule violations using buffer insertion, gate sizing, and even logic restructuring in real time.

2.8 Functional and Factory Testing

Tests to verify the functionality of the ASIC can be written which can then be run on silicon using the automated test equipment (ATE), or tester, at the foundry. Due to the limitations of the speed and complexity of the ATE, often the ASIC cannot be completely functionally tested, or even tested at speed on the ATE. A subset of functional tests, which satisfy the ATE requirements, are chosen to be run on the ATE. However, if the tests used to validate the RTL model pass, the STA using parasitics passes, and the formal verification of the RTL model to the physical design database netlist compares, one can be fairly confident of silicon success without even running functional tests on the ATE.

In addition to the scan test and RAMBIST, another factory test, which is often required, is the IDDQ test. The IDDQ test measures the static current draw of the chip – it should be minimal, on the order of microamps for a CMOS ASIC. A large current draw is an indication of a short circuit inside the ASIC – a faulty part.

2.9 Data Preparation and Physical Verification

Typically, the place and route libraries only contain the top layers of the cells – poly and the metal layers. Once the place and route database is complete, the place and route database is merged with the base layers of the cells. After the merge, a physical design rule check (DRC), a layout versus schematic check (LVS), and an electrical rule check (ERC) are run. DRC is run to ensure that no physical design rules, such as metal layer

spacing, are violated. LVS is run to ensure that the connectivity of the physical database matches that of the netlist, checking for shorted and open nets. ERC is run to ensure that no electrical rules, such as charge collecting antenna violations, are violated. Once the database has passed physical verification, it is ready for fabrication.

2.10 Delivery to the Customer and Customer Support

Once the wafer has been fabricated, each individual site on the wafer is tested. Any faulty sites on the wafer are discarded. Sites on the wafer, which pass wafer test, are then packaged and tested again on the ATE equipment. Upon passing all of the ATE tests, parts are shipped to the customer. The moment of truth arrives when the customer places the delivered part in their system, and turns the power on. If everything works fine, the celebration can begin. However, if there is a problem, depending on the severity of the problem, an entire re-spin encompassing performing some or all of the above steps must be repeated. A solid ASIC specification, appropriate project planning, and careful attention to detail at every step in the design process is the formula for first pass success.

Chapter 3

ASIC Migration Design Considerations

3.1 System Architecture and Planning

Since an ASIC migration deals with an already existing design, there is usually minimal if not any system architecture design. However, there are definitely system architecture issues as well as ASIC architecture issues which must be taken into consideration to ensure success. The first step is to examine all of the files associated with the original design. A complete design database consists of a captured design, test fixtures, timing constraints, and target technology requirements. In most cases, due to varying circumstances, there is usually something missing from the original database. Although this does not mean that the design will not be successful, it does make the conversion process more difficult as well as increases the risk of failure.

Figure 3.1 shows there are three categories of ASIC conversions: 1. Gate array or standard cell to gate array or standard cell. 2. FPGA to gate array or standard cell. 3. Gate array or standard cell to FPGA. The first two categories are the most commonly encountered. The first type is usually due to an EOL process situation – the ASIC is fabricated in a process which is being phased out by the FAB. The second type can be encountered in two situations: cost reduction or performance increase. For the third type, the only reason why a gate array or standard cell part would be converted to an FPGA is if the original test fixtures or test vectors were missing, the original designer was unable to be located for consultation, and there is still demand for the part but the part is being

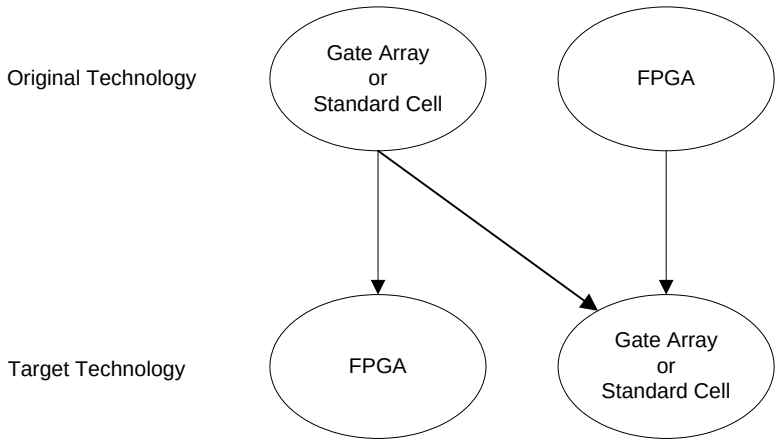

Figure 3.1: Conversion types.

discontinued by the FAB. As a result of these situations, the risk would be too great to target a masked part. It makes much more sense to target a reconfigurable solution where the design can be implemented, tested in the target system, and modified as needed.

In planning for the migration, it is important to take into careful consideration the information presented in the following sections.

3.2 RTL Coding, Validation, and Netlist Translation

The two most important items required for the migration design to begin are the design captured in some form of netlist – either gatelevel or RTL, and Verilog or VHDL RTL test fixtures or some format of test vectors.

3.2.1 Verilog or VHDL RTL Netlist

The ideal and simplest starting point is a Verilog or VHDL RTL netlist. In this case, the designer should first validate the RTL code by performing a simulation using either the

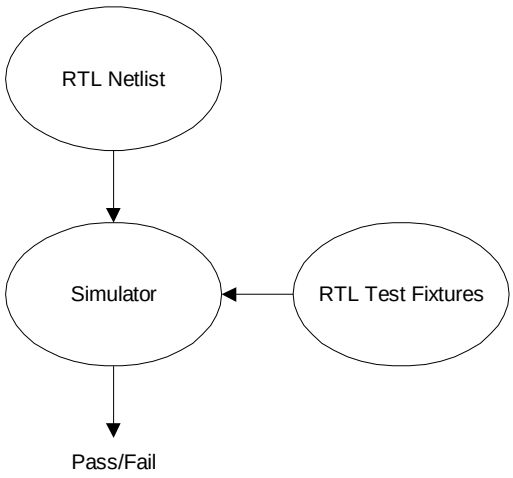

Figure 3.2: RTL flow.

provided RTL test fixtures or test vectors to ensure that the provided RTL netlist is 100% complete (“golden”). A simple diagram of this flow is shown in figure 3.2. If the simulation passes, the design process can move to synthesis. Simulation failures can be attributed to something as simple as an incorrect mapping of a logic cell in the map file or incorrect formatting/translation of the test vectors. However, the failure could be something as difficult to fix as incorrect timing through an asynchronous logic path. Regardless, if the simulation fails, the cause must be determined and resolved before going any further.

3.2.2 Verilog, VHDL, or EDIF Gatelevel Netlist

Another common starting point is a Verilog, VHDL, or EDIF gatelevel netlist. This flow is shown in Figure 3.3. A Verilog gatelevel netlist in the target technology can be generated by one of two methods, both involve the use of a synthesis tool. If the synthesis and simulation libraries are available for the original technology, the original design can simply be read into the synthesis tool, linked with the original technology libraries and re-targeted to the target libraries resulting in a gatelevel netlist written out of the tool. The

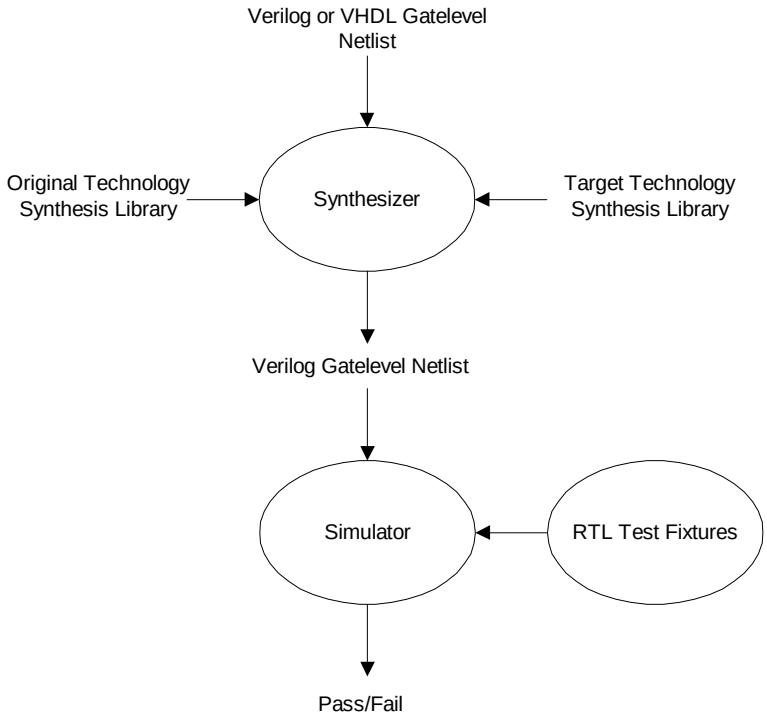

Figure 3.3: Verilog or VHDL gatelevel flow using original technology synthesis libraries.

original and resulting netlists should be simulated to verify functionality. Any mismatches should be resolved before proceeding.

In the absence of the original technology synthesis libraries, RTL wrappers can be created such that the instantiated cells in the netlist behave according to the datasheets in the databook. This allows the simulator to resolve all module calls. This flow is shown in Figurfe 3.4.

3.2.3 Other Gatelevel Netlist

A gatelevel netlist can be represented in many formats. The three most commonly seen formats are Verilog, VHDL, and EDIF. The translation can be done with a synthesis tool or netlist translation software – if the synthesis tool or netlist translation software supports

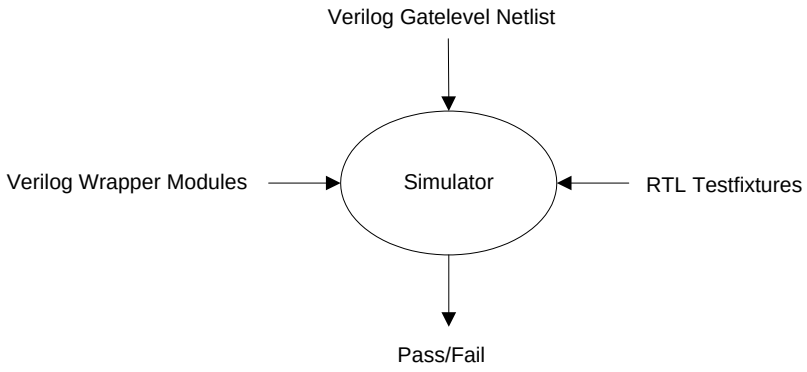

Figure 3.4: Verilog or VHDL gatelevel flow using original wrapper modules

the input netlist format – or if all else fails, manually with a text editor. In any case, once the netlist is in gatelevel format a functional simulation should be run to verify functionality. Any simulation mismatches need to be resolved before proceeding further.

3.2.4 Other Considerations

Gatelevel translation of standard logic cells, combinational gates, and sequential elements such as latches and flip-flops is rather straight forward. The challenge is in the translation of macrocells such as random access memories (RAMs), read only memories (ROMs), phase locked loops (PLLs), digital to analog converters (DACs), analog to digital converters (ADCs), microprocessors, and other intellectual property (IP) blocks. One needs to be sure that the functionality of required blocks is avaliable in the target technology. The easiest solution is a direct replacement with the exact same implementation in the new technology. For example, if a 32x32 2-port synchronous RAM is used in the original technology, and if one is available in the target technology, it is used. If one is not availabe in the target technology, the functionality of the RAM could easily be implemented in RTL and synthesized with registers to the target technology.

This is an acceptable soultion only if the implementation can meet or exceed the timing and power requirements of the original RAM. The same thing can be done with a ROM. All other macrocells and IP blocks are not as straightforward to implement. It is important that a direct replacement is available, as the development effort required or cost to obtain what is needed could be considerable.

Another item which needs to be addressed is I/O selection. Proper I/O selection is extremely important. Even if all of the core logic functions properly, if the signals cannot get on and off of the chip as needed, the chip is rendered useless. If there are special I/O requirements such as PCI, PECL, ECL, and LVDS one needs to be sure that these I/O are available in the target technology library. If a particular special I/O is not available in the target technology, one can be developed – however, it will incur an increase in cost and development schedule. For standard I/O, such as CMOS and TTL, it is important to match input and output driver types from the original design to the target design. Additionally, it is as important to match output drive strengths. An output driver which is too weak will not be able to properly drive the signal and can cause system timing problems. An output driver which is too strong can introduce unwanted problems into the system such as ground bounce, unneeded extra current draw meaning an increase in power draw, and increased heat.

3.3 Synthesis Logic Optimization

Once the netlist has passed initial functional simulation, synthesis logic optimization should be performed on the gatelevel translated netlist. Synthesis logic optimization is performed to do optimization steps such as logic reduction and restructuring and

maximum transition and capacitance design rule violation fixing. Because the cells in each process are designed differently, a logic path which is optimial in area and timing in one technology may not be optimal in the target technology. Synthesis optimization addresses this.

3.4 Testability

Testability is an important, and often overlooked, consideration for a migration. If the manufactured part cannot be sufficiently tested for manufacturing defects at the FAB, the customer cannot be guaranteed to receive parts which are tested to be free of manufacturing defects – the customer is paying for parts which will be tested at their site, in their system, fail, and be immediately discarded. This situation is a blatant waste of money, which is easily preventable, in advance, through testability methods.

Depending on the system constraints and internal logic structure of the design, several avenues of testability are available to the designer. The most ideal situation, which is rarely encountered, is a design which is completely synchronous, is designed for test, and has extra I/O available for scan insertion. In this case, internal scan can be easily inserted and ATPG run to obtain manufacturing test patterns which provide sufficient fault coverage.

Another design situation is one where there are extra I/O available for scan insertion, but the design is not completely synchronous and is not designed for test. In this case, test logic can be inserted to make the design testable such that internal scan can be inserted and ATPG performed on the design to obtain sufficient fault coverage. Situations where

test logic may need to be inserted include: gated clocks, clocks used as data, data used as clocks, and any other signals which have priority on sequential elements (such as asynchronous sets and resets) but are not controllable from a primary input pin.

Another design situation is one where there are no extra I/O available for scan insertion. In this case two approaches can be used. The functional tests used to verify the design can be fault graded, using fault simulation software, to determine the fault coverage provided by the functional tests. In most cases the functional tests alone will not provide sufficient fault coverage. To increase the coverage, additional tests can be manually written and then fault graded. However, this may not be the best approach due to the fact that it may take many, many vectors and a considerable amount of man hours to increase the coverage manually. Another approach would be to use a full-sequential ATPG tool. A fulll-sequential ATPG tool can analyze the existing functional test vectors and generate additional vectors automatically to increase coverage.

Although the full-sequential ATPG choice sounds like a silver bullet solution to the problem, it may not prove to provide sufficient results. The algorithms used by the software to generate the full-sequential patterns are geared towards sequential datapaths, like those found in a microprocessor, which are extremely timing critical paths and are therefore not good candidates for scan insertion. Because of the nature of the logic often found in migrations (gated clocks, clocks used as data, asynchronous loops, etc.) full-sequential ATPG may not be able to provide as high a fault coverage as one would have hoped. In this case, pursuing full-sequential ATPG must be abandoned and another testability methodology, like one of those discussed above, must be used.

3.5 Static Timing Analysis

Static timing analysis is a requirement for all new designs today no matter the target technology. Very often with migrations, the only designs which are ever analyzed are conversions from FPGAs and designs for which timing specifications have been provided and are relatively synchronous. All other designs can be difficult, even impossible, to analyze for two reasons: missing timing constraints and/or the design methodology used to implement the original design. Missing timing constraints is self explanatory.

Older design methodologies, particularly those used throughout the 1980's and early 1990's, differ greatly from those used today. This is mainly due to the effects associated with decreasing transistor feature sizes and the method used for design entry. In feature sizes above 1um, the delay through logic gates was considerably more than the delay through the interconnect. As the feature size continues to decrease, the delay through the interconnect becomes much more significant, in some cases, so much so that the delay through the interconnect can approach or even exceed the delay through a logic gate.

In older technologies, typically the design was captured using a schematic capture tool. Because of the nature of logic design, a particular design can be implemented several different ways. All logically equivalent implementations perform the same logic function, but will have different power consumptions and timing through the logic. As a result of this, ease of design entry and timing requirements typically took priority over what is considered today to be good design methodology practices. As a result, many older designs contain gated clocks, clocks used as data, internally generated clocks, pulse or

sliver generators, and other logic implementations which are not friendly to today's EDA STA to enable them to easily analyze the design.

A good example is a counter. Figure 3.5 shows a waveform and schematic for a synchronous counter. Figure 3.6 shows a waveform and schematic for a ripple counter. It is obvious that the schematics of these two designs are different. Specifically, in the synchronous counter, both registers are fed from the same clock. In the ripple counter, one register is fed a clock and the clock to the second register is generated directly from the output of the first register. The synchronous counter is comprised of six instances. The ripple counter is comprised of three instances. In terms of ease of design entry using a schematic editor, it is obvious that the ripple counter is more easily entered compared to the synchronous counter. Additionally, from a power consumption point of view, because the ripple counter consists of fewer gates, and has a lower switching activity factor (due to the clocking structure), the ripple counter will consume slightly less power than the synchronous counter.

The waveforms of the two counters are almost identical. However, there are subtle differences. At 50.01ns, when the synchronous counter switches from 01 to 10, both outputs switch at the same time. After layout, the only time difference between the most significant bit (MSB) and least significant bit (LSB) switching will be due to clock skew, which is usually less than 500ps. On the other hand, at 50.01ns, when the ripple counter should switch from 01 to 10, the actual output sequence is: 01, 00, 10. This is due to the clocking structure.

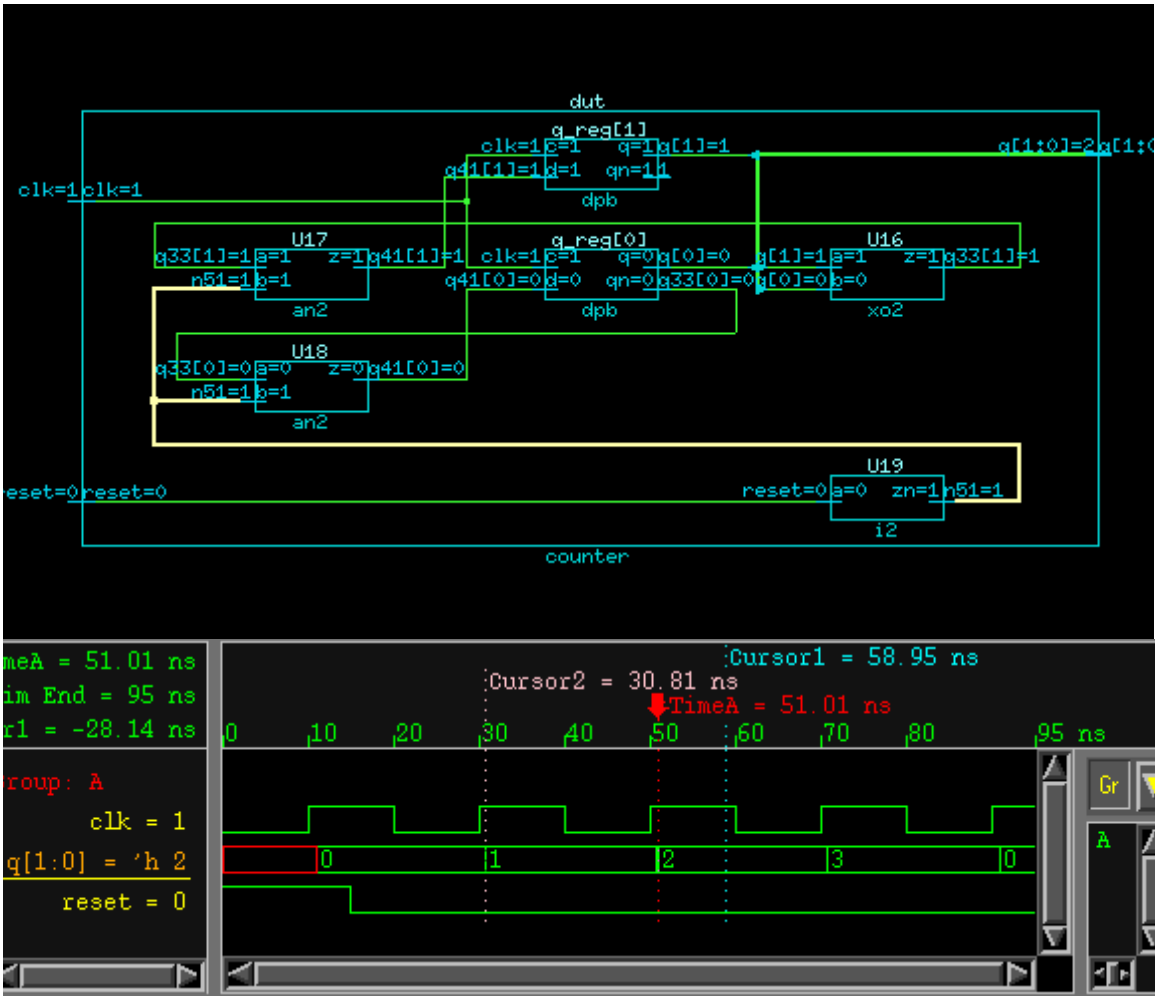

Figure 3.5: Synchrouos counter schematic and waveform.

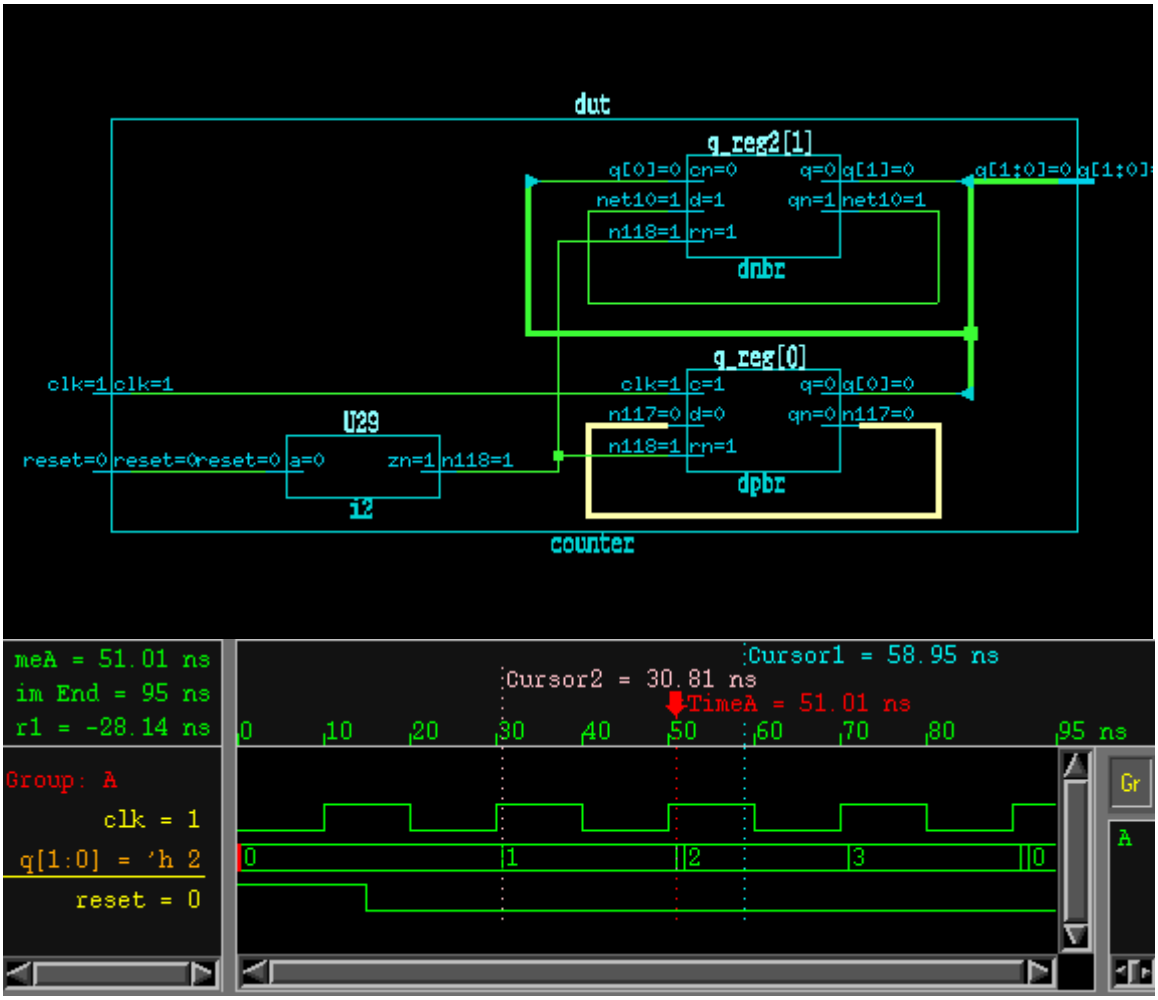

Figure 3.6: Ripple counter schematic and waveform.

After layout, the time difference between the MSB and LSB switching will be due to the clock to output propagation of the MSB flip flop in addition to the wire delay from the output of the LSB flip flop connected to the clock pin of the MSB flip flop. This can take greater than 1ns, which is much higher than 500ps for the synchronous counter. For a 2-bit counter this may seem insignificant. However, each bit added to the width of the counter increases the output stable time of the counter, which can be considerable for large counters. A solution to this problem is to reimplement the ripple counter to be a synchronous counter.

From a STA standpoint, a synchronous counter is much easier to analyze. It has is only one clock which needs to be defined. The ripple counter has as many clocks to be defined as there are bits, making it very tedious to setup the STA software. As a last point, because of their architecture, ripple counters are prone to hold time problems.

In earlier technologies, a ripple counter implementation would be acceptable. Today, due to the increase in path delay due to the interconnect (which increases the possibility of hold time problems), and the steps required to set up the STA, a ripple counter implementation is no longer acceptable.

In the event that STA can not be performed, post-layout backannotated simulation is often used as a substitute. Although this method will not provide nearly as comprehensive an analysis as STA, a design with no timing constraints or a design implemented in a way that cannot be easily analyzed with a STA tool leaves no other choice in method for timing

verification. Using this method, the analysis is only as complete as the percentage of logic toggled by the simulations run on the design.

3.6 Physical Design: Floorplanning and Layout

The first item to consider in physical design is I/O location and bonding. Since the usual motivating factor behind a migration is cost reduction, it is important that the original I/O pinout is preserved. Not preserving the pinout would result in a modification to the printed circuit board the ASIC is mounted on, thereby incurring cost and reducing the effectiveness of the cost reduction. Although it would seem that preserving the pinout is a simple task, there are several matters which need to be analyzed to be sure that the pinout can be preserved.

For larger pin designs, typically FPGAs, in array type packages, such as pin grid array (PGA) or ball grid array (BGA) the locations of power and ground pins are usually predetermined by the package vendor. In the case of a cost reduction migration, pin for pin compatibility is of utmost importance. If the target package selected does not support pin for pin compatibility, a custom substrate can be designed. However, the downside is an increase in NRE cost as well as the additional time which must be added to the schedule to accommodate the design and fabrication of the custom substrate. Although pinout considerations are presented late in the design flow (because it is part of the physical design process) pinout must be addressed early in the design flow to ensure on-time delivery of prototypes.

Two other issues which need to be addressed in this stage of the design phase are clocking and power strategy. Clocking strategy is very important. The goal of the clocking strategy is to try to achieve a reasonable amount of clock insertion delay while maintaining low clock skew. The clocking strategy used in the old technology may be quite different from that which is required in the new technology. Changes to the netlist may have to be made to accommodate this. For example, the old technology may have used several clock driving buffers in parallel, often called a clock spine, to drive the clock net to all of the flip flops, whereas the new technology may require a tree of clock buffers, often called a clock tree. Figure 3.7 shows the architectural difference between a clock spine structure and a clock tree structure. The required clock structure implementation for the target technology is usually determined by the FAB.

A proper power strategy is important to ensure that sufficient power is available to all cells at all times, across the entire area of the chip. Special consideration needs to be taken into account for powering of macrocells and any other special cells. Suggestions for the

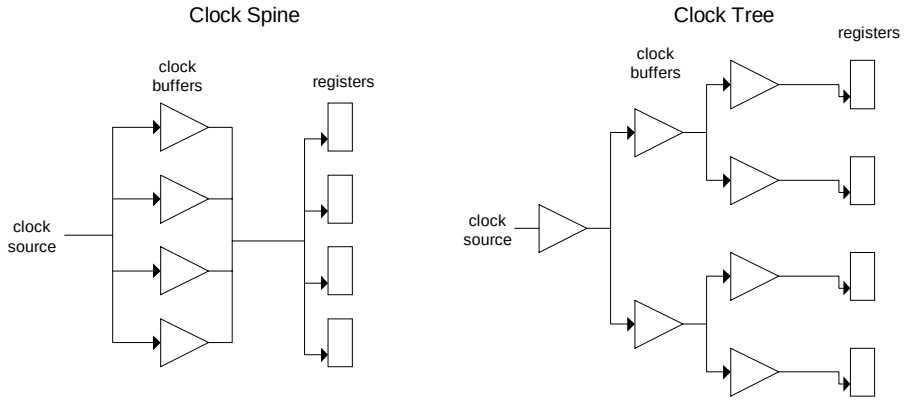

Figure 3.7: Clock spine and clock tree models.

minimum width, minimum spacing, and metal layers for the power mesh to power the logic cells are usually provided by the FAB.

3.7 Post-Layout Analysis and Data Preparation

Upon completion of layout, parasitic data is provided back to the designer. Simulations should be back-annotated with the parasitic data and run to ensure that the functionality is still met with the timing increase in path delays due to interconnect. Additionally, if all required information is available for STA, it should be run, back-annotated with the parasitic data to verify that the design still meets the required timing constraints. If either the simulations or STA fail, the appropriate modifications need to be made to the design to fix the problem. The loop through layout and post-layout verification must be gone through again until all timing problems, post-layout, are resolved.

Since the layout tool is only responsible for inserting interconnect consisting of metal and vias on the top layers of the chip, the data contained in the layout tool is only representative of the silicon layers from poly up. Therefore, once the design is verified, post-layout, the physical data must be streamed out of the layout tool, in graphical data stream II format (GDSII) and then merged with the full GDSII files of all of the library cells (which contain the layers below poly and the metal pins). Final DRC and LVS is run on the full chip GDSII file which, if clean, will be sent to the FAB for manufacturing. Any problems which appear in the LVS and DRC runs must be resolved before releasing the design to FAB. If any problems are found, once they are fixed, parasitic data should again be outputout of the layout tool, and back-annotated STA and simulations should be run to verify that the fixes did not adversely effect the timing.

Chapter 4

A Migration Design Example

The design example chosen was a 1.2um gate array to 0.8um gate array conversion. The 1.2um gate array to 0.8um gate array conversion consisted of converting a 1.2um ORBIT Semiconductor process gate array to a 0.8um Chartered process gate array. The 1.2um design was captured in a gatelevel SILOS netlist. The migration target netlist format was a 0.8um gatelevel Verilog netlist. The original test vectors were provided. However, no timing information was provided or was able to be obtained.

4.1 Netlist Translation

The netlist translation from 1.2um to 0.8um takes several steps. A block diagram of the methodology is shown in Figure 4.1. The first step is to translate the netlist from 1.2um

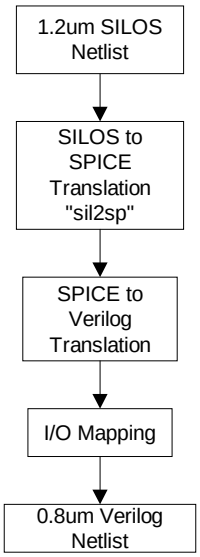

Figure 4.1: 1.2um SILOS to 0.8um Verilog netlist translation methodology block diagram.

SILOS format to 1.2um SPICE format. This can be done manually with a text editor, or can be automated using either a scripting language or C. Appendix A.1 contains the 1.2um SILOS netlist. Appendix A.2 contains the 1.2um SPICE netlist. The second step is to convert the SPICE netlist to a Verilog netlist. This can be done manually with a text editor or commercially available netlist translation software, such as Avant! Nettran. Once the netlist is in Verilog gatelevel format, the final step of netlist translation is to properly translate the input and output drivers.

In ORBIT 1.2um technology the input and output drivers consist of two library cells, the pre-driver cell and the driver cell. For example, the INBUFX3 input buffer consists of two parts: an input protection circuit, P_IP, and the TTL input driver circuit, P_ITX3. The input protection circuit is connected to the pad which is then connected to the input of the TTL input driver circuit. The output of the TTL input driver circuit is then connected to the core logic. The two cells map into a single input buffer in 0.8um, a ITX3 TTL input buffer. The input protection circuity and the TTL input driver is contained within the ITX3 TTL input buffer library cell.

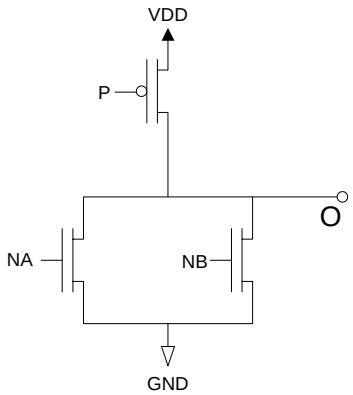

Figure 4.2: Schematic of P_OBS8 output buffer.

The mapping of the output buffers is similar to that of the input buffers, but not quite as straight forward. The OUTBUF8X cell consists of two cells, an inverter (P_I1) driving an output buffer (P_OBS8). The schematic of the P_OBS8 cell is shown in Figure 4.2.

From the schematic in Figure 4.2, the truth table, shown in Table 4.1, for the P_OBS8 output buffer is easily obtained. Just as with the input driver cells, the pre-driver and driver cells for the output drivers are combined into a single output driver cell. Therefore, the proper translation of the OUTBUF8X 1.2um cell, is a ob8 cell in 0.8um. With all inputs and outputs properly mapped, the initial translation to a 0.8um gatelevel Verilog netlist is complete and is shown in Appendix A.3.

4.2 Test Fixture Generation and Initial Simulation

Upon completion of the initial Verilog netlist translation, the netlist must be simulated to verify functionality to prove that the netlist was properly translated. A Verilog test fixture, shown in Appendix A.6, was written using the 1.2um test vectors to prove the functionality of the design. A block diagram of the test fixture is shown in Figure 4.3. The test fixture

Table 4.1: P_OBS8 output buffer truth table.

NA	NB	P	O
1	1	1	0
0	0	1	Z
0	0	0	1
1	X	0	not allowed
X	1	0	not allowed

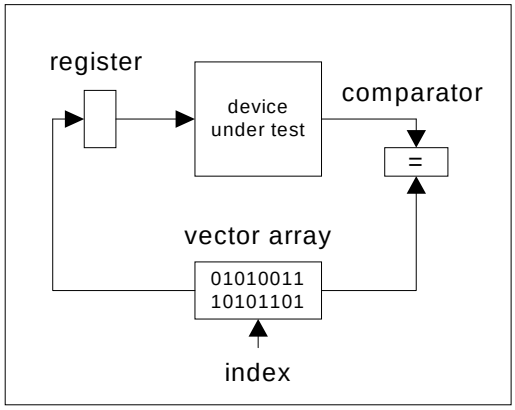

Figure 4.3: Block diagram of test fixture.

is comprised of five sections: a register and wire declaration section, an initialization section, the for loop, the chip instantiation, and the tasks.

The register and wire declaration section is used to declare register values, which are applied to the inputs of the structure under test, and wire values, which are connected to the outputs of the structure under test. Each test cycle consists of input stimulus being assigned to the registers and output response being sampled on the wires and then compared to the expected output response.

The initialization section is used for initializing the test, performing such functions as: reading in the back-annotation parasitic data when appropriate, setting up the filename and structures to record waveform data for debug purposes, and loading the test vectors from a file into an array. Appendix A.5 shows the original 1.2um test vector file. Each vector column in the file corresponds to a pin on the design. Valid values applied to input pins are logic 0 and logic 1. Valid expected response values are: logic 0 denoted as ‘L’, logic 1 denoted as ‘H’, and don’t care denoted as ‘X’. Appendix A.7 shows the input

stimulus and output expected response, read by the test fixture, on a cycle by cycle basis – one cycle per line. The input stimulus and expected response file was created directly from the original 1.2um test vector file using a text editor. The header and left column were removed, all ‘H’ output values were converted to ‘1’, and all ‘L’ output values were converted to ‘0’.

The actual testing of the chip, assigning inputs and comparing the actual response with the expected response, is done in the for loop. Each iteration through the for loop corresponds to a single tester cycle. The entire test consists of 244 iterations through the for loop, which is exactly the number of tester cycles required to test the chip as well as the number of test vectors provided. The simulation time for each iteration of the for loop is 10,000 ns.

The chip instantiation section instantiates the design into the test fixture, making the actual logical connections between the registers and wires declared in the test fixture and the inputs and outputs of the device under test.

Finally, the task section consists of two tasks called by the for loop: assign_inputs and strobe. The assign_inputs task uses the increment value of the for loop to index into the array to assign input values from the array to the registers. All input signals, except the clocks, REG_CLK and INP_CLK, are non-return to zero (NRZ) signals, meaning that they only change at the beginning of the cycle. REG_CLK and INP_CLK are 15ns delayed non-return to zero signals (DNRZ), meaning that they only change 15ns into the cycle. A graphical representation of NRZ and DNRZ signals is shown in Figure 4.4. The strobe

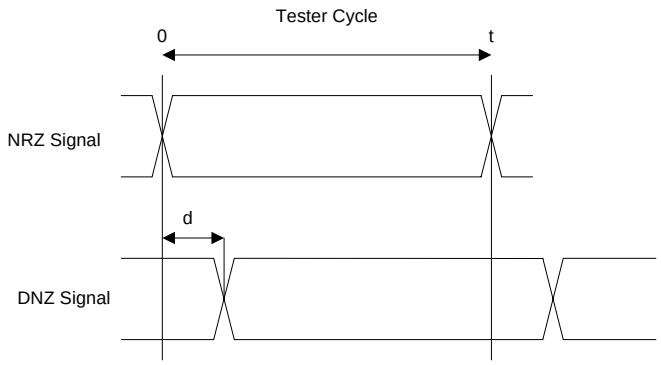

Figure 4.4: NRZ and DNRZ signal representation.

task is executed near the very end of each test cycle. It compares the simulated response of the device outputs with the expected response values. The expected response values are located by the index number, provided by the for loop, into the array. If a simulation mismatch occurs, an error message is displayed on the screen with the following information: the mismatched signal name, the failing test vector number, the expected value and the simulated value. Any signal mismatches can be debugged using the waveform data.

4.3 Synthesis Logic Optimization

After the initial simulation is complete, the netlist is then processed with synthesis software to perform logic reduction, optimization, flattening, and ungrouping. Due to the manual logic mapping process from 1.2um to 0.8um, unused extra gates can be introduced into the design. For example, the P_OR3 cell has an inverting and non-inverting output. The inverting output is made by directly inverting the non-inverting output, adding an additional gate. However, both outputs may not be used in the circuit. If the inverting output is not used it will be removed by the optimization process.

The synthesis optimization script is shown in Appendix A.8. The final optimized netlist is shown in Appendix A.9. Comparing the pre-optimized and optimized netlists side by side, the most obvious difference between them is that the optimized netlist contains a single level of hierarchy. From a size standpoint, the number of blocks contained in the pre-optimized and optimized netlists differ considerably. The pre-optimized netlist contained 197 blocks compared to 152 blocks in the optimized netlist, a 22% reduction in area.

After the first attempt at logic optimization was complete, it was discovered that the power on reset cell was modeled incorrectly in the synthesis library. This discovery was made when the functional simulation of the optimized netlist failed. The problem was easy to find, as the chip primary outputs were miscomparing, expecting a logic zero or one value, but were simulating a logic unknown. The problem was traced back from the primary outputs to the flip flops, which were driving a logic unknown because the flip flops were missing the asynchronous reset pin and they had not yet been clocked. The netlist prior to optimization contained active low asynchronous reset flip flops. The asynchronous reset pin on all of the flip flops was directly connected to the output of the power on reset cell. The power on reset cell provides an active low signal for approximately 10ms after power is applied to the device, thereby eliminating the need for the target system to reset the ASIC. Because the synthesis model of the power on reset cell was written such that the power on reset cell was always driving an inactive value, or logic high value, the synthesis tool replaced the asychronous reset flip flops with regular flip flops. The synthesis model is a logical model and does not have any provision for time relationship. Because the output of the POR cell is time dependent, it is impossible to write a model which exactly mimics the behavior of the cell. The model was re-written such that the cell

always drives a logic unknown. Although this does not exactly mimic the time dependent behavior of the POR cell, it prevents the POR cell and forward logic cone from being optimized out of the netlist.

With the synthesis library updated, the synthesis optimization was re-run. The netlist was then re-verified with a functional simulation and found to be correct.

4.4 Design For Test: Fault Simulation

The scope of the work on the design did not call for DFT in the form of scan insertion and ATPG. The functional vectors, run on the tester, were used to test for manufacturing faults. SimuCad HyperFault software was used to fault grade the functional vectors using the optimized netlist. The functional test vectors provide 85% fault coverage. The fault simulation log is shown in Appendix A.9.

For the majority of designs, 85% fault coverage is considered unacceptable. The usual target number is on the order of 95% or greater. In cases where fault coverage falls below an acceptable level, typically the customer has to sign a waiver which releases the manufacturer of financial responsibility for shipping parts which are found not to work in the target system due to manufacturing defects.

In this case, 85% fault coverage was acceptable because the scope of the work did not allow for additional measures to be taken to increase fault coverage. However, if the scope of the work did call for additional measures to be taken to increase fault coverage scan could have been inserted and ATPG run to generate additional fault coverage or

additional test vectors could have been manually developed. Scan insertion and ATPG would have been easily facilitated, physically, through the use of several of the many pins which are not connected on the package. Attempting to increase the fault coverage by manually developing additional vectors would be much more cumbersome and time consuming than the coverage which could be easily achieved through scan insertion and ATPG.

4.5 Floorplanning Preparation: Pinout and Base Array Fit

Because the design contains only combinational and sequential logic cells and no macrocells, the only floorplanning effort required was to determine the location of the I/O pads on the die and to be sure that the design would fit in the assigned base array. Since the intention is that the 0.8um part is to be a drop-in replacement of the 1.2um part, the pin locations on the 0.8um part cannot differ from the 1.2um part. The pinfile, which shows the I/O signal pad and pin assignments, is shown in Appendix A.10. The design easily fits in the assigned base array: the design contains 152 blocks, the base array can accommodate up to 2100 blocks.

4.6 Pre-Layout Analysis

Before the design can be released to layout, the design must be simulated with estimated parasitic back-annotation information. The estimated parasitic data is obtained by processing the netlist with a delay calculator. The estimated parasitic data is calculated using wireload models in conjunction with the area of the design. In addition to parasitic data as an output file, the delay calculator provides a logfile which details if nets are

underdriven. If a net is underdriven, either the drive strength of the cell driving the net must be increased, or a buffer must be inserted between the driving cell and the net.

After running the delay calculator on the design, the logfile was checked. No underdriven nets were reported. Additionally, the design was simulated with the estimated parasitics. The simulation passed and no timing violations were reported by the simulator.

If the design methodology is complete, STA would be performed at this point in the design flow to ensure that the design is meeting timing before layout. Input setup and hold times, output required times, and register to register setup and hold times would be checked. In this case, the timing information was not provided. Therefore, a STA run could not be completed. In lieu of STA, the simulator was used to verify the timing of the design. The simulator can only check setup, hold, and width violations on registers. It cannot check output required times. Unlike a STA tool, the simulator will only check timing on paths which are exercised by the vectors. Therefore, it is possible that a setup or hold problem may exist if the absolute longest or shortest paths are not exercised by the simulator. This method of verifying the timing of the design provides nowhere near as comprehensive an analysis as using STA. However, since the original timing information was not available, this method was the second best choice.

4.7 Layout

Upon successful completion of the pre-layout analysis, the design was released to layout. The design was laid out by a Flextronics Semiconductor layout engineer using the Avant! Apollo-GA software tool. Because of the very small size of the design, the layout was

completed quickly with no routing congestion problems. The actual parasitic data was then output for processing by the delay calculator.

4.8 Post-Layout Analysis

The parasitic data provided from layout was input into the delay calculator. No post-route underdriven nets were reported. A final back-annotated simulation was run using the back-annotation files output from the delay calculator. No timing errors were reported and the simulation passed.

4.9 Release to FAB

Once the post-layout analysis was complete, the design was released to FAB. The parts were fabricated and prototypes shipped back for testing on the ATE.

4.10 Prototype Test

The fabricated prototypes were tested using the original 1.2um test vector file on the ATE and passed. The prototype parts were then shipped to the customer for in system test.

Chapter 5

Conclusions and Future Work

This thesis presented an overall ASIC design flow methodology, ASIC design flow methodology considerations specifically for ASIC technology migrations, and an ASIC technology migration design example. There are many situations which can cause difficulty in the migration process. Detailing every example possible is out of the scope of this thesis. However, enough information has been presented to help a person unfamiliar with ASIC technology migrations avoid common pitfalls in the design flow which can result in a negative impact on schedule, delivery of a defective part, or both. The contribution of this thesis is to provide a guide to first pass success for ASIC technology migrations.

The information presented in this thesis has been gathered through the process of performing many ASIC technology migrations, all of which were successful. It should be pointed out that every design which was encountered presented unique problems. Additionally, it is expected that in the future problems not seen in prior migrations may be encountered.

Looking forward, the ASIC design industry may be starting to follow a new trend. The mask costs for the latest fabrication technology, 0.13um, cost as much as $1M. The cost is significant compared to what the mask costs were for what the cutting edge process was several years ago. Therefore, companies may be hesitant to invest the large amount of capital required without initial proof of the concept in silicon. This can be done using a

very large FPGA and may or may not run at the target speed for the application. Once the concept is proven, the design will then be targeted to a masked part, in the form of a migration, thereby reducing the risk. The difference between this migration process and those done to date is several fold.

The process of translating a gatelevel netlist in one technology to a gatelevel netlist in another technology is now being termed as a migration. Because the latest FPGAs are so large, over two million gates, and typically include various propriatarey intellectual property blocks it is impractical to use a gatelevel netlist as a starting point for the translation. Therefore, the starting point is the RTL code, in either Verilog or VHDL format, and the act of the translation process is now termed a ‘re-targeting’ rather than a migration.

Another difference between a migration and a re-target is in the initial development effort. The initial development effort put into the FPGA is done with the intention that the part will eventually be fabricated into a masked part. Therefore, DFT, issues relating to testing the final masked part on the tester, and other design issues applicable for a masked part but not for an FPGA are considered and accounted for during the FPGA development effort. Because of this, when it is time to ‘re-target’ the ASIC, the process will require minimal rework.

References

1. ASIC International, Inc. “Design Process” poster.

2. Marc Royer and Mark Goode, “FPGA to ASIC Migration”, ISD Magazine, November 2001.

3. Ravi Thummarukady, “Design Methodologies for DSM ASIC Designs”, ISD Magazine, March 1999.

4. Michael John Sebastian Smith, Application Specific Integrated Circuits, Addison Wesley, 1997.

Appendices

Appendix A: Design Example Code

A.1: 1.2um SILOS Netlist

```
.GLOBAL VDD VSS GLOBAL_IN7

.MACRO   INBUFX1  O PAD
(M1      P_IP       PAD
(M2      P_IT     O PAD
.EOM

.MACRO   INBUFUX1  O PAD
(M1      P_IPR       PAD
(M2      P_IT      O PAD
.EOM

.MACRO   INBUFDX1  O PAD
(M1      P_IPD       PAD
(M2      P_IT      O PAD
.EOM

.MACRO   INBUFX3   O PAD
(M1      P_IP        PAD
(M2      P_ITX3    O PAD
.EOM

.MACRO   INBUFX3_D  O  PAD
(M1      P_IP          PAD
(M2      P_IT       O1 PAD
(M3      P_ID2      O2 O1
(M4      P_ID2      O3 O2
(M5      P_ID3      O4 O3
(M6      P_I3       O  O4
.EOM

.MACRO   OUTBUFX8  PAD IN
(M1      P_OBS8    PAD O O O
(M2      P_I3      O IN
.EOM

.MACRO  AND2   O A B
(M1     P_AA2  O X A B
.EOM

.MACRO  AND3   O A B C
(M1     P_AA3  O X A B C
.EOM

.MACRO  AND4   O A B C D
(M1     P_AA4  O X A B C D
.EOM

.MACRO  AND5   O A B C D E
(M1     P_AF5  O A B C D E
.EOM

.MACRO  AND8   O A B C D E F G H
(M1     P_AF8  O A B C D E F G H
.EOM

.MACRO  NAND3  O A B C
(M1     P_NA3  O A B C
.EOM


.MACRO  OR3    O A B C
(M1     P_OR3  O X A B C
.EOM

.MACRO  OR2    O A B
(M1     P_OR2  O X A B
.EOM

.MACRO  NOR2   O A B
(M1     P_NO2  O A B
.EOM
```

```
.MACRO  NOR3   O A B C
(M1     P_NO3  O A B C
.EOM

.MACRO  DFF      Q D CLK CLR SET GLOBAL_IN7
(M1     P_DPRS   Q QN CLK D R SET
(M2     AND2     R CLR GLOBAL_IN7
.EOM


.MACRO  CHIP         OUT1 OUT2 OUT3 OUT4
+                    IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN12
+                    IN9 IN10 IN11 IN13 IN14 IN15


(OUT1          OUTBUFX8        OUT1    OUT1C
(OUT2          OUTBUFX8        OUT2    OUT2C
(OUT3          OUTBUFX8        OUT3    OUT3C
(OUT4          OUTBUFX8        OUT4    OUT4C

(IN1       INBUFX3        IN_IN1    IN1
(IN2          INBUFDX1       IN_IN2       IN2
(IN3          INBUFUX1       IN_IN3       IN3
(IN4          INBUFX1        IN_IN4       IN4
(IN5           INBUFX1        IN_IN5        IN5
(IN6           INBUFUX1       IN_IN6        IN6
(IN7         INBUFX1        IN_IN7      IN7
(IN8            INBUFX1        IN_IN8         IN8
(IN12       INBUFX3_D      IN_IN12    IN12
(IN9            INBUFX1        IN_IN9         IN9
(IN10            INBUFX1        IN_IN10         IN10
(IN11            INBUFX1        IN_IN11         IN11
(IN13          INBUFX1        IN_IN13        IN13
(IN14          INBUFX1        IN_IN14        IN14
(IN15          INBUFX1        IN_IN15        IN15

(INV_IN7     P_I1           N_IN7    IN_IN7
(INV_IN2      P_I1           N_IN2     IN_IN2
(INV_IN3      P_I1           N_IN3     IN_IN3
(INV_IN4      P_I1           N_IN4     IN_IN4
(INV_IN5       P_I1           N_IN5      IN_IN5
(INV_IN6       P_I1           N_IN6      IN_IN6
(INV_IN8        P_I1           N_IN8       IN_IN8
(INV_IN9        P_I1           N_IN9       IN_IN9
(INV_IN10        P_I1           N_IN10       IN_IN10
(INV_IN11        P_I1           N_IN11       IN_IN11
(INV_IN13       P_I1           N_IN13      IN_IN13
(INV_IN14       P_I1           N_IN14      IN_IN14
(INV_IN15       P_I1           N_IN15      IN_IN15

$        DFF      Q        D        CLK      CLR        SET GLOBAL_IN7

(C01     DFF    IN5_S1   IN_IN5   IN_IN12 VDD        VDD GLOBAL_IN7
(C02     DFF    IN5_S2F  IN5_S1   IN_IN1 VDD        VDD GLOBAL_IN7
(C03     DFF    LIN5F    IN5_EDD  IN_IN1 N_IN5_ACK  VDD GLOBAL_IN7
(C030    P_I1   N_LIN5F  LIN5F

(C04     DFF    IN4_S1  IN_IN4  IN_IN12 VDD        VDD GLOBAL_IN7
(C05     DFF    IN4_S2F IN4_S1  IN_IN1 VDD        VDD GLOBAL_IN7
(C06     DFF    LIN4F   IN4_EDD IN_IN1 N_IN4_ACK VDD GLOBAL_IN7
(C060    P_I1   N_LIN4F LIN4F

(C07     DFF    IN2_S1  IN_IN2  IN_IN12 VDD        VDD GLOBAL_IN7
(C08     DFF    IN2_S2F IN2_S1  IN_IN1 VDD        VDD GLOBAL_IN7
(C09     DFF    LIN2F   IN2_EDD IN_IN1 N_IN2_ACK VDD GLOBAL_IN7
(C090    P_I1   N_LIN2F LIN2F

(C10     NAND3  OUT4C IN_IN13 IN_IN14 IN_IN15


(C110    P_I1   N_IN5_S1   IN5_S1
(C111    AND2   IN5_       N_IN5_S1  IN5_S2F
(C112    OR2    IN5_EDD    IN5_      LIN5F

(C120    P_I1   N_IN4_S1  IN4_S1
(C121    AND2   IN4_      N_IN4_S1 IN4_S2F
(C122    OR2    IN4_EDD   IN4_     LIN4F
```

```
(C130    P_I1   N_IN2_S2F IN2_S2F
(C131    AND2   IN2_      IN2_S1   N_IN2_S2F
(C132    OR2    IN2_EDD   IN2_     LIN2F


(C140    AND8   IN5ACK N_IN11 IN_IN13 IN_IN14 IN_IN15 IN_IN10 N_IN9 N_IN8 VDD
(C141    NOR2   N_IN5_ACK  N_IN7 IN5ACK

(C150    AND8   IN4ACK N_IN11 IN_IN13 IN_IN14 IN_IN15 N_IN10 IN_IN9 IN_IN8 VDD
(C151    NOR2   N_IN4_ACK N_IN7 IN4ACK

(C160    AND8   IN2ACK N_IN11 IN_IN13 IN_IN14 IN_IN15 IN_IN10 N_IN9 IN_IN8 VDD
(C161    NOR2   N_IN2_ACK N_IN7 IN2ACK

(C170    AND5   OUT1C1 N_LIN2F N_LIN5F N_LIN4F IN_IN3 N_IN6
(C171    AND3   OUT1C2 N_LIN2F N_LIN5F LIN4F
(C172    NOR3   OUT1C  LIN2F   OUT1C1  OUT1C2

(C180    AND4   OUT2C1 N_LIN2F N_LIN5F N_LIN4F N_IN3
(C181    AND3   OUT2C2 N_LIN2F N_LIN5F LIN4F
(C182    NOR2   OUT2C  OUT2C1   OUT2C2

(C190    AND2   OUT3C1 N_LIN2F LIN5F
(C191    NOR2   OUT3C  LIN2F   OUT3C1

(PWR_IN7  P_POR GLOBAL_IN7

.EOM

(TOP    CHIP         OUT1 OUT2 OUT3 OUT4
+                    IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN12
+                    IN9 IN10 IN11 IN13 IN14 IN15

.TABLE      OUT1 OUT2 OUT3 OUT4
+                    IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN12
+                    IN9 IN10 IN11 IN13 IN14 IN15
```

A.2: 1.2um SPICE Netlist

```
.GLOBAL VDD VSS GLOBAL_IN7


.SUBCKT   INBUFX1  O PAD
XM1       PAD
+ P_IP
XM2     O PAD
+ P_IT
.ENDS
.SUBCKT   INBUFUX1  O PAD
XM1       PAD
+ P_IPR
XM2      O PAD
+ P_IT
.ENDS
.SUBCKT   INBUFDX1  O PAD
XM1       PAD
+ P_IPD
XM2      O PAD
+ P_IT
.ENDS
.SUBCKT   INBUFX3   O PAD
XM1        PAD
+ P_IP
XM2    O PAD
+ P_ITX3
.ENDS
.SUBCKT   INBUFX3_D  O  PAD
XM1          PAD
+ P_IP
XM2       O1 PAD
+ P_IT
XM3      O2 O1
+ P_ID2
XM4      O3 O2
+ P_ID2
XM5      O4 O3
+ P_ID3
XM6       O  O4
+ P_I3
.ENDS
.SUBCKT   OUTBUFX8  PAD IN
XM1    PAD O O O
+ P_OBS8
XM2      O IN
+ P_I3
.ENDS
.SUBCKT  AND2   O A B
XM1  O X A B
+ P_AA2
.ENDS
.SUBCKT  AND3   O A B C
XM1  O X A B C
+ P_AA3
.ENDS
.SUBCKT  AND4   O A B C D
XM1  O X A B C D
+ P_AA4
.ENDS
.SUBCKT  AND5   O A B C D E
XM1  O A B C D E
+ P_AF5
.ENDS
.SUBCKT  AND8   O A B C D E F G H
XM1  O A B C D E F G H
+ P_AF8
.ENDS
.SUBCKT  NAND3  O A B C
XM1  O A B C
+ P_NA3
.ENDS

.SUBCKT  OR3    O A B C
```

```
XM1  O X A B C
+ P_OR3
.ENDS
.SUBCKT  OR2    O A B
XM1  O X A B
+ P_OR2
.ENDS
.SUBCKT  NOR2   O A B
XM1  O A B
+ P_NO2
.ENDS
.SUBCKT  NOR3   O A B C
XM1  O A B C
+ P_NO3
.ENDS
.SUBCKT  DFF      Q D CLK CLR SET GLOBAL_IN7
XM1   Q QN CLK D R SET
+ P_DPRS
XM2     R CLR GLOBAL_IN7
+ AND2
.ENDS

.SUBCKT  CHIP   OUT1 OUT2 OUT3 OUT4
+                    IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN12
+                    IN9 IN10 IN11 IN13 IN14 IN15


XOUT1        OUT1    OUT1C
+ OUTBUFX8
XOUT2        OUT2    OUT2C
+ OUTBUFX8
XOUT3        OUT3    OUT3C
+ OUTBUFX8
XOUT4        OUT4    OUT4C
+ OUTBUFX8

XIN1        IN_IN1    IN1
+ INBUFX3
XIN2       IN_IN2       IN2
+ INBUFDX1
XIN3       IN_IN3       IN3
+ INBUFUX1
XIN4        IN_IN4       IN4
+ INBUFX1
XIN5        IN_IN5        IN5
+ INBUFX1
XIN6       IN_IN6        IN6
+ INBUFUX1
XIN7        IN_IN7      IN7
+ INBUFX1
XIN8        IN_IN8         IN8
+ INBUFX1
XIN12      IN_IN12    IN12
+ INBUFX3_D
XIN9        IN_IN9         IN9
+ INBUFX1
XIN10        IN_IN10         IN10
+ INBUFX1
XIN11        IN_IN11         IN11
+ INBUFX1
XIN13        IN_IN13        IN13
+ INBUFX1
XIN14        IN_IN14        IN14
+ INBUFX1
XIN15        IN_IN15        IN15
+ INBUFX1

XINV_IN7           N_IN7    IN_IN7
+ P_I1
XINV_IN2           N_IN2     IN_IN2
+ P_I1
XINV_IN3           N_IN3     IN_IN3
+ P_I1
XINV_IN4           N_IN4     IN_IN4
+ P_I1
XINV_IN5           N_IN5      IN_IN5
+ P_I1
```

```
XINV_IN6           N_IN6      IN_IN6
+ P_I1
XINV_IN8           N_IN8       IN_IN8
+ P_I1
XINV_IN9           N_IN9       IN_IN9
+ P_I1
XINV_IN10           N_IN10       IN_IN10
+ P_I1
XINV_IN11           N_IN11       IN_IN11
+ P_I1
XINV_IN13           N_IN13      IN_IN13
+ P_I1
XINV_IN14           N_IN14      IN_IN14
+ P_I1
XINV_IN15           N_IN15      IN_IN15
+ P_I1

*         DFF      Q        D         CLK      CLR        SET GLOBAL_IN7

XC01    IN5_S1   IN_IN5   IN_IN12 VDD        VDD GLOBAL_IN7
+ DFF
XC02    IN5_S2F  IN5_S1   IN_IN1 VDD        VDD GLOBAL_IN7
+ DFF
XC03    LIN5F    IN5_EDD  IN_IN1 N_IN5_ACK  VDD GLOBAL_IN7
+ DFF
XC030   N_LIN5F  LIN5F
+ P_I1

XC04    IN4_S1  IN_IN4  IN_IN12 VDD        VDD GLOBAL_IN7
+ DFF
XC05    IN4_S2F IN4_S1  IN_IN1 VDD        VDD GLOBAL_IN7
+ DFF
XC06    LIN4F   IN4_EDD IN_IN1 N_IN4_ACK VDD GLOBAL_IN7
+ DFF
XC060   N_LIN4F LIN4F
+ P_I1

XC07    IN2_S1  IN_IN2  IN_IN12 VDD        VDD GLOBAL_IN7
+ DFF
XC08    IN2_S2F IN2_S1  IN_IN1 VDD        VDD GLOBAL_IN7
+ DFF
XC09    LIN2F   IN2_EDD IN_IN1 N_IN2_ACK VDD GLOBAL_IN7
+ DFF
XC090   N_LIN2F LIN2F
+ P_I1

XC10  OUT4C IN_IN13 IN_IN14 IN_IN15
+ NAND3


XC110   N_IN5_S1   IN5_S1
+ P_I1
XC111   IN5_       N_IN5_S1  IN5_S2F
+ AND2
XC112    IN5_EDD    IN5_      LIN5F
+ OR2

XC120   N_IN4_S1  IN4_S1
+ P_I1
XC121   IN4_      N_IN4_S1 IN4_S2F
+ AND2
XC122    IN4_EDD   IN4_     LIN4F
+ OR2

XC130   N_IN2_S2F IN2_S2F
+ P_I1
XC131   IN2_      IN2_S1   N_IN2_S2F
+ AND2
XC132    IN2_EDD   IN2_     LIN2F
+ OR2


XC140   IN5ACK N_IN11 IN_IN13 IN_IN14 IN_IN15 IN_IN10 N_IN9 N_IN8 VDD
+ AND8
XC141   N_IN5_ACK  N_IN7 IN5ACK
+ NOR2
```

```
XC150   IN4ACK N_IN11 IN_IN13 IN_IN14 IN_IN15 N_IN10 IN_IN9 IN_IN8 VDD
+ AND8
XC151   N_IN4_ACK N_IN7 IN4ACK
+ NOR2

XC160   IN2ACK N_IN11 IN_IN13 IN_IN14 IN_IN15 IN_IN10 N_IN9 IN_IN8 VDD
+ AND8
XC161   N_IN2_ACK N_IN7 IN2ACK
+ NOR2

XC170   OUT1C1 N_LIN2F N_LIN5F N_LIN4F IN_IN3 N_IN6
+ AND5
XC171   OUT1C2 N_LIN2F N_LIN5F LIN4F
+ AND3
XC172   OUT1C  LIN2F   OUT1C1  OUT1C2
+ NOR3

XC180   OUT2C1 N_LIN2F N_LIN5F N_LIN4F N_IN3
+ AND4
XC181   OUT2C2 N_LIN2F N_LIN5F LIN4F
+ AND3
XC182   OUT2C  OUT2C1   OUT2C2
+ NOR2

XC190   OUT3C1 N_LIN2F LIN5F
+ AND2
XC191   OUT3C  LIN2F   OUT3C1
+ NOR2

XPWR_IN7 GLOBAL_IN7
+ P_POR

.ENDS
XTOP   OUT1 OUT2 OUT3 OUT4
+                   IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN12
+                   IN9 IN10 IN11 IN13 IN14 IN15
+ CHIP
```

A.3: 1.2um Initial Verilog Netlist With Mapped I/O

```
// Verilog output file 'test.v'
module NOR3 (
      O ,
      A ,
      B ,
      C );
      inout O ;
      inout A ;
      inout B ;
      inout C ;
     P_NO3  XM1      (
               .O(O),
               .A1(A),
               .A2(B),
               .A3(C));
endmodule
module NOR2 (
      O ,
      A ,
      B );
      inout O ;
      inout A ;
      inout B ;
     P_NO2  XM1      (
               .O(O),
               .A1(A),
               .A2(B));
endmodule
module NAND3 (
      O ,
      A ,
      B ,
      C );
      inout O ;
      inout A ;
      inout B ;
      inout C ;
     P_NA3  XM1      (
               .O(O),
               .A1(A),
               .A2(B),
               .A3(C));
endmodule
/*module INBUFX3 (
      O ,
      PAD );
      inout O ;
      inout PAD ;
     P_ITX3  XM2      (
               .O(O),
               .A(PAD));
     P_IP  XM1      (
               .O(PAD));
endmodule
module INBUFX1 (
      O ,
      PAD );
      inout O ;
      inout PAD ;
     P_IT  XM2      (
               .O(O),
               .A(PAD));
     P_IP  XM1      (
               .O(PAD));
endmodule
module INBUFUX1 (
      O ,
      PAD );
      inout O ;
      inout PAD ;
     P_IT  XM2      (
               .O(O),
               .A(PAD));
     P_IPR  XM1      (
```

```
              .O(PAD));
endmodule
module INBUFDX1 (
      O ,
      PAD );
      inout O ;
      inout PAD ;
     P_IT  XM2      (
               .O(O),
               .A(PAD));
     P_IPD  XM1      (
               .O(PAD));
endmodule
module OUTBUFX8 (
      PAD ,
      IN );
      inout PAD ;
      inout IN ;
     P_I3  XM2      (
               .O(O),
               .A(IN));
     P_OBS8  XM1      (
               .O(PAD),
               .NA(O),
               .NB(O),
               .P(O));
endmodule
module INBUFX3_D (
      O ,
      PAD );
      inout O ;
      inout PAD ;
     P_I3  XM6      (
               .O(O),
               .A(O4));
     P_ID3  XM5      (
               .O(O4),
               .A(O3));
     P_ID2  XM4      (
               .O(O3),
               .A(O2));
     P_ID2  XM3      (
               .O(O2),
               .A(O1));
     P_IT  XM2      (
               .O(O1),
               .A(PAD));
     P_IP  XM1      (
               .O(PAD));
endmodule
*/
module OR3 (
      O ,
      A ,
      B ,
      C );
      inout O ;
      inout A ;
      inout B ;
      inout C ;
     P_OR3  XM1      (
               .O(O),
               .X(X),
               .A(A),
               .B(B),
               .C(C));
endmodule
module OR2 (
      O ,
      A ,
      B );
      inout O ;
      inout A ;
      inout B ;
     P_OR2  XM1      (
               .O(O),
               .X(X),
```

```
               .A(A),
               .B(B));
endmodule
module AND5 (
      O ,
      A ,
      B ,
      C ,
      D ,
      E );
      inout O ;
      inout A ;
      inout B ;
      inout C ;
      inout D ;
      inout E ;
     P_AF5  XM1      (
               .O(O),
               .A(A),
               .B(B),
               .C(C),
               .D(D),
               .E(E));
endmodule
module AND8 (
      O ,
      A ,
      B ,
      C ,
      D ,
      E ,
      F ,
      G ,
      H );
      inout O ;
      inout A ;
      inout B ;
      inout C ;
      inout D ;
      inout E ;
      inout F ;
      inout G ;
      inout H ;
     P_AF8  XM1      (
               .O(O),
               .A(A),
               .B(B),
               .C(C),
               .D(D),
               .E(E),
               .F(F),
               .G(G),
               .H(H));
endmodule
module AND4 (
      O ,
      A ,
      B ,
      C ,
      D );
      inout O ;
      inout A ;
      inout B ;
      inout C ;
      inout D ;
     P_AA4  XM1      (
               .O(O),
               .X(X),
               .A(A),
               .B(B),
               .C(C),
               .D(D));
endmodule
module AND3 (
      O ,
      A ,
      B ,
```

```
      C );
      inout O ;
      inout A ;
      inout B ;
      inout C ;
     P_AA3  XM1      (
               .O(O),
               .X(X),
               .A(A),
               .B(B),
               .C(C));
endmodule
module AND2 (
      O ,
      A ,
      B );
      inout O ;
      inout A ;
      inout B ;
     P_AA2  XM1      (
               .O(O),
               .X(X),
               .A(A),
               .B(B));
endmodule
module DFF (
      Q ,
      D ,
      CLK ,
      CLR ,
      SET ,
      GLOBAL_IN7 );
      inout Q ;
      inout D ;
      inout CLK ;
      inout CLR ;
      inout SET ;
      inout GLOBAL_IN7 ;
     AND2  XM2      (
               .O(R),
               .A(CLR),
               .B(GLOBAL_IN7));
     P_DPRS  XM1      (
               .Q(Q),
               .XQ(QN),
               .C(CLK),
               .D(D),
               .XR(R),
               .XS(SET));
endmodule
module CHIP (
      OUT1 ,
      OUT2 ,
      OUT3 ,
      OUT4 ,
      IN1 ,
      IN2 ,
      IN3 ,
      IN4 ,
      IN5 ,
      IN6 ,
      IN7 ,
      IN8 ,
      IN12 ,
      IN9 ,
      IN10 ,
      IN11 ,
      IN13 ,
      IN14 ,
      IN15 );
      output OUT1 ;
      output OUT2 ;
      output OUT3 ;
      output OUT4 ;
      input IN1 ;
      input IN2 ;
      input IN3 ;
```

```
 input IN4 ;
 input IN5 ;
 input IN6 ;
 input IN7 ;
 input IN8 ;
 input IN12 ;
 input IN9 ;
 input IN10 ;
 input IN11 ;
 input IN13 ;
 input IN14 ;
 input IN15 ;
P_POR  XPWR_IN7      (
          .O(GLOBAL_IN7));
NOR2  XC191      (
          .O(OUT3C),
          .A(LIN2F),
          .B(OUT3C1));
AND2  XC190      (
          .O(OUT3C1),
          .A(N_LIN2F),
          .B(LIN5F));
NOR2  XC182      (
          .O(OUT2C),
          .A(OUT2C1),
          .B(OUT2C2));
AND3  XC181      (
          .O(OUT2C2),
          .A(N_LIN2F),
          .B(N_LIN5F),
          .C(LIN4F));
AND4  XC180      (
          .O(OUT2C1),
          .A(N_LIN2F),
          .B(N_LIN5F),
          .C(N_LIN4F),
          .D(N_IN3));
NOR3  XC172      (
          .O(OUT1C),
          .A(LIN2F),
          .B(OUT1C1),
          .C(OUT1C2));
AND3  XC171      (
          .O(OUT1C2),
          .A(N_LIN2F),
          .B(N_LIN5F),
          .C(LIN4F));
AND5  XC170      (
          .O(OUT1C1),
          .A(N_LIN2F),
          .B(N_LIN5F),
          .C(N_LIN4F),
          .D(IN_IN3),
          .E(N_IN6));
NOR2  XC161      (
          .O(N_IN2_ACK),
          .A(N_IN7),
          .B(IN2ACK));
AND8  XC160      (
          .O(IN2ACK),
          .A(N_IN11),
          .B(IN_IN13),
          .C(IN_IN14),
          .D(IN_IN15),
          .E(IN_IN10),
          .F(N_IN9),
          .G(IN_IN8),
          .H(VDD));
NOR2  XC151      (
          .O(N_IN4_ACK),
          .A(N_IN7),
          .B(IN4ACK));
AND8  XC150      (
          .O(IN4ACK),
          .A(N_IN11),
          .B(IN_IN13),
          .C(IN_IN14),
```

```
          .D(IN_IN15),
          .E(N_IN10),
          .F(IN_IN9),
          .G(IN_IN8),
          .H(VDD));
NOR2  XC141      (
          .O(N_IN5_ACK),
          .A(N_IN7),
          .B(IN5ACK));
AND8  XC140      (
          .O(IN5ACK),
          .A(N_IN11),
          .B(IN_IN13),
          .C(IN_IN14),
          .D(IN_IN15),
          .E(IN_IN10),
          .F(N_IN9),
          .G(N_IN8),
          .H(VDD));
OR2  XC132      (
          .O(IN2_EDD),
          .A(IN2_),
          .B(LIN2F));
AND2  XC131      (
          .O(IN2_),
          .A(IN2_S1),
          .B(N_IN2_S2F));
P_I1  XC130      (
          .O(N_IN2_S2F),
          .I(IN2_S2F));
OR2  XC122      (
          .O(IN4_EDD),
          .A(IN4_),
          .B(LIN4F));
AND2  XC121      (
          .O(IN4_),
          .A(N_IN4_S1),
          .B(IN4_S2F));
P_I1  XC120      (
          .O(N_IN4_S1),
          .I(IN4_S1));
OR2  XC112      (
          .O(IN5_EDD),
          .A(IN5_),
          .B(LIN5F));
AND2  XC111      (
          .O(IN5_),
          .A(N_IN5_S1),
          .B(IN5_S2F));
P_I1  XC110      (
          .O(N_IN5_S1),
          .I(IN5_S1));
NAND3  XC10      (
          .O(OUT4C),
          .A(IN_IN13),
          .B(IN_IN14),
          .C(IN_IN15));
P_I1  XC090      (
          .O(N_LIN2F),
          .I(LIN2F));
DFF  XC09      (
          .Q(LIN2F),
          .D(IN2_EDD),
          .CLK(IN_IN1),
          .CLR(N_IN2_ACK),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
DFF  XC08      (
          .Q(IN2_S2F),
          .D(IN2_S1),
          .CLK(IN_IN1),
          .CLR(VDD),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
DFF  XC07      (
          .Q(IN2_S1),
          .D(IN_IN2),
```

```
          .CLK(IN_IN12),
          .CLR(VDD),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
P_I1  XC060      (
          .O(N_LIN4F),
          .I(LIN4F));
DFF   XC06       (
          .Q(LIN4F),
          .D(IN4_EDD),
          .CLK(IN_IN1),
          .CLR(N_IN4_ACK),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
DFF   XC05       (
          .Q(IN4_S2F),
          .D(IN4_S1),
          .CLK(IN_IN1),
          .CLR(VDD),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
DFF   XC04       (
          .Q(IN4_S1),
          .D(IN_IN4),
          .CLK(IN_IN12),
          .CLR(VDD),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
P_I1  XC030      (
          .O(N_LIN5F),
          .I(LIN5F));
DFF   XC03       (
          .Q(LIN5F),
          .D(IN5_EDD),
          .CLK(IN_IN1),
          .CLR(N_IN5_ACK),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
DFF   XC02       (
          .Q(IN5_S2F),
          .D(IN5_S1),
          .CLK(IN_IN1),
          .CLR(VDD),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
DFF   XC01       (
          .Q(IN5_S1),
          .D(IN_IN5),
          .CLK(IN_IN12),
          .CLR(VDD),
          .SET(VDD),
          .GLOBAL_IN7(GLOBAL_IN7));
P_I1  XINV_IN15      (
          .O(N_IN15),
          .I(IN_IN15));
P_I1  XINV_IN14      (
          .O(N_IN14),
          .I(IN_IN14));
P_I1  XINV_IN13      (
          .O(N_IN13),
          .I(IN_IN13));
P_I1  XINV_IN11      (
          .O(N_IN11),
          .I(IN_IN11));
P_I1  XINV_IN10      (
          .O(N_IN10),
          .I(IN_IN10));
P_I1  XINV_IN9       (
          .O(N_IN9),
          .I(IN_IN9));
P_I1  XINV_IN8       (
          .O(N_IN8),
          .I(IN_IN8));
P_I1  XINV_IN6       (
          .O(N_IN6),
          .I(IN_IN6));
P_I1  XINV_IN5       (
```

```
              .O(N_IN5),
              .I(IN_IN5));
     P_I1  XINV_IN4      (
              .O(N_IN4),
              .I(IN_IN4));
     P_I1  XINV_IN3      (
              .O(N_IN3),
              .I(IN_IN3));
     P_I1  XINV_IN2      (
              .O(N_IN2),
              .I(IN_IN2));
     P_I1  XINV_IN7      (
              .O(N_IN7),
              .I(IN_IN7));
    it IN13_pad (.pad(IN13),.z(IN_IN13));
    it IN14_pad (.pad(IN14),.z(IN_IN14));
    it IN15_pad (.pad(IN15),.z(IN_IN15));
/*
     INBUFX1  XIN15      (
              .O(IN_IN15),
              .PAD(IN15));
     INBUFX1  XIN14      (
              .O(IN_IN14),
              .PAD(IN14));
     INBUFX1  XIN13      (
              .O(IN_IN13),
              .PAD(IN13));
*/
    it IN11_pad (.pad(IN11),.z(IN_IN11));
/*
     INBUFX1  XIN11      (
              .O(IN_IN11),
              .PAD(IN11));
*/
    it IN10_pad (.pad(IN10),.z(IN_IN10));
/*
     INBUFX1  XIN10      (
              .O(IN_IN10),
              .PAD(IN10));
*/
    it IN9_pad (.pad(IN9),.z(IN_IN9));
/*
     INBUFX1  XIN9      (
              .O(IN_IN9),
              .PAD(IN9));
*/
    itx3 IN12_pad (.pad(IN12),.z(IN_IN12));
/*
     INBUFX3_D  XIN12      (
              .O(IN_IN12),
              .PAD(IN12));
*/
    it IN8_pad (.pad(IN8),.z(IN_IN8));
/*
     INBUFX1  XIN8      (
              .O(IN_IN8),
              .PAD(IN8));
*/
    it IN7_pad (.pad(IN7),.z(IN_IN7));
/*
     INBUFX1  XIN7      (
              .O(IN_IN7),
              .PAD(IN7));
*/
    it IN6_pad (.pad(IN6),.z(IN_IN6));
/*
     INBUFUX1  XIN6      (
              .O(IN_IN6),
              .PAD(IN6));
*/
    it IN5_pad (.pad(IN5),.z(IN_IN5));
/*
     INBUFX1  XIN5      (
              .O(IN_IN5),
              .PAD(IN5));
*/
    it IN4_pad (.pad(IN4),.z(IN_IN4));
```

```
/*
     INBUFX1  XIN4      (
               .O(IN_IN4),
               .PAD(IN4));
*/

    it IN3_pad (.pad(IN3),.z(IN_IN3));
/*
     INBUFUX1  XIN3      (
               .O(IN_IN3),
               .PAD(IN3));
*/
    it IN2_pad (.pad(IN2),.z(IN_IN2));
/*
     INBUFDX1  XIN2      (
               .O(IN_IN2),
               .PAD(IN2));
*/
    itx3 IN1_pad (.pad(IN1),.z(IN_IN1));
/*
     INBUFX3  XIN1      (
               .O(IN_IN1),
               .PAD(IN1));
*/
     ob8 OUT1_pad (.pad(OUT1),.a(OUT1C));
     ob8 OUT2_pad (.pad(OUT2),.a(OUT2C));
     ob8 OUT3_pad (.pad(OUT3),.a(OUT3C));
     ob8 OUT4_pad (.pad(OUT4),.a(OUT4C));

/*     OUTBUFX8  XOUT4      (
               .PAD(OUT4),
               .IN(OUT4C));
     OUTBUFX8  XOUT3      (
               .PAD(OUT3),
               .IN(OUT3C));
     OUTBUFX8  XOUT2      (
               .PAD(OUT2),
               .IN(OUT2C));

     OUTBUFX8  XOUT1      (
               .PAD(OUT1),
               .IN(OUT1C));
*/
endmodule
/*module NT_TOP_61373lvs (
);
     CHIP  XTOP      (
               .OUT1(OUT1),
               .OUT2(OUT2),
               .OUT3(OUT3),
               .OUT4(OUT4),
               .IN1(IN1),
               .IN2(IN2),
               .IN3(IN3),
               .IN4(IN4),
               .IN5(IN5),
               .IN6(IN6),
               .IN7(IN7),
               .IN8(IN8),
               .IN12(IN12),
               .IN9(IN9),
               .IN10(IN10),
               .IN11(IN11),
               .IN13(IN13),
               .IN14(IN14),
               .IN15(IN15));
endmodule
*/
```

A.4: 1.2um to 0.8um Verilog Mapping File

```
module P_POR(O);
output O;

por i_1(.zn(O));

endmodule

module P_NO3(O, A1, A2, A3);
input A1, A2, A3;
output O;

no3 M01 (.zn(O), .c(A3), .b(A2), .a(A1));
endmodule

module P_NA3(O, A1, A2, A3);
input A1, A2, A3;
output O;

na3 M01 (.zn(O), .a(A1), .b(A2), .c(A3));
endmodule


module P_NO2(O, A1, A2);
input A1, A2;
output O;

no2 M01 (.zn(O), .b(A2), .a(A1));
endmodule

module P_OR3(O, X, A, B, C);
input A, B, C;
  output O, X;

or3 MO1 (.z(O), .a(A), .b(B), .c(C));
i1 M02 (.zn(X), .a(O));
endmodule

module P_OR2(O, X, A, B);
input A, B;
output O, X;

or2 M01 (.z(O), .a(A), .b(B));
i1 M02 (.zn(X), .a(O));
endmodule

module P_AF5(O,A,B,C,D,E);

input A,B,C,D,E;
output O;

an5 i_1 (.z(O),.a(A),.b(B),.c(C),.d(D),.e(E));
endmodule

module P_AF8(O,A,B,C,D,E,F,G,H);

input A,B,C,D,E,F,G,H;
output O;

an8 i_1 (.z(O),.a(A),.b(B),.c(C),.d(D),.e(E),.f(F),.g(G),.h(H));
endmodule

module P_AA4(O, X, A, B, C, D);
input A, B, C, D;
output O, X;

an4 M01 (.z(O), .a(A), .b(B), .c(C), .d(D));
i1 M02 (.zn(X), .a(O));
endmodule

module P_AA3(O, X, A, B, C);
input A, B, C;
output O, X;

an3 MO1 (.z(O), .a(A), .b(B), .c(C));
```

```
i1 M02 (.zn(X), .a(O));
endmodule

module P_AA2(O, X, A, B);
input A, B;
output O, X;

an2 M01 (.z(O), .a(A), .b(B));
i1  M02 (.zn(X), .a(O));
endmodule

module P_DPRS(Q, XQ, C, D, XR, XS);
input C, D, XR, XS;
output Q, XQ;

dpbrs M01(.q(Q), .qn(XQ), .c(C), .d(D), .rn(XR), .sn(XS));
endmodule

module P_I1(O, I);
input I;
output O;

i1 M01 (.zn(O), .a(I));
endmodule
```

A.5: 1.2um Original Test Vector File

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$  S I L O S  I I I    ver 97.1          Tue Dec 30 14:00:21 1997
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


            viviiiiivvviivviiiviiiioooov
            dnsnnnnnsssnnssnnnsnnnnuuuud
            d1s23456sss78ss191s1111ttttd
                          2 0 13451234



       TIME
        4999  10001111000000000001000HHHH1
        9999  10001111000000000001000HHHH1
       14999  11001111000000010001000HHHH1
       19999  10001111000000000001000HHHH1
       24999  11001111000000010001000HHHH1
       29999  10001111000000000001000HHHH1
       34999  11001111000000010001000HHHH1
       39999  10001111000000000001000HHHH1
       44999  11001111000000010001000HHHH1
       49999  10001111000000000001000HHHH1
       54999  11001111000000010001000HHHH1
       59999  10001111000000000001000HHHH1
       64999  11001111000000010001000HHHH1
       69999  10001111000000000001000HHHH1
       74999  11001111000000010001000HHHH1
       79999  10001111000000000001000HHHH1
       84999  11001111000000010001000HHHH1
       89999  10001111000100000001000HHHH1
       94999  11001111000100010001000HHHH1
       99999  10001111000100000001000HHHH1
      104999  11001111000100010001000HHHH1
      109999  10001111000100000001000HHHH1
      114999  11001111000100010001000HHHH1
      119999  10001111000100000001001HHHH1
      124999  11011111000100010001010HHHH1
      129999  10011111000100000000011HHHH1
      134999  11011111000100010000100LHLH1
      139999  10011111000100000000101LHLH1
      144999  11011111000100010000110LHLH1
      149999  10011111000100000000000LHLH1
      154999  11011111000100010001000LHLH1
      159999  10011111000100000001000LHLH1
      164999  11011111000100010001000LHLH1
      169999  10011111000100000001000LHLH1
      174999  11011111000100010001111LHLL1
      179999  10011111000100000001111LHLL1
      184999  11011111000110010101111LHLL1
      189999  10011111000110000100111HHHL1
      194999  11011111000110010100111HHHL1
      199999  10011111000110000100111HHHL1
      204999  11011111000110010100111HHHL1
      209999  10011111000110000101111HHHL1
      214999  11001111000110010101111HHHL1
      219999  10001111000100000001000HHHH1
      224999  11001111000100010001000HHHH1
      229999  10001111000100000001000HHHH1
      234999  11001011000100010001000HHHH1
      239999  10001011000100000000000HHHH1
      244999  11001011000100010000000LLHH1
      249999  10001011000100000000000LLHH1
      254999  11001011000100010000000LLHH1
      259999  10001011000100000000000LLHH1
      264999  11001011000100010001000LLHH1
      269999  10001011000100000001000LLHH1
      274999  11001011000100010001000LLHH1
      279999  10001011000100000001000LLHH1
      284999  11001011000100010001111LLHL1
      289999  10001011000100000001111LLHL1
      294999  11001011000110011001111LLHL1
      299999  10001011000110001000111HHHL1
```

```
304999  11001011000110011000111HHHL1
309999  10001011000110001000111HHHL1
314999  11001011000110011000111HHHL1
319999  10001011000110001001111HHHL1
324999  11001111000110011001111HHHL1
329999  10001111000100000001000HHHH1
334999  11001111000100010001000HHHH1
339999  10001111000100000001000HHHH1
344999  11001101000100010001000HHHH1
349999  10001101000100000000000HHHH1
354999  11001101000100010000000HHLH1
359999  10001101000100000000000HHLH1
364999  11001101000100010000000HHLH1
369999  10001101000100000000000HHLH1
374999  11001101000100010001000HHLH1
379999  10001101000100000001000HHLH1
384999  11001101000100010001000HHLH1
389999  10001101000100000001000HHLH1
394999  11001101000100010001111HHLL1
399999  10001101000100000001111HHLL1
404999  11001101000100010101111HHLL1
409999  10001101000100000100111HHHL1
414999  11001101000100010100111HHHL1
419999  10001101000100000100111HHHL1
424999  11001101000100010100111HHHL1
429999  10001101000100000101111HHHL1
434999  11001111000100010101111HHHL1
439999  10001111000100000001000HHHH1
444999  11001111000100010001000HHHH1
449999  10001111000100000001000HHHH1
454999  11001111000100010001000HHHH1
459999  10000111000100000000000HLHH1
464999  11000111000100010000000HLHH1
469999  10000111000100000000000HLHH1
474999  11000111000100010000000HLHH1
479999  10000111000100000000000HLHH1
484999  11000111000100010001000HLHH1
489999  10000111000100000001000HLHH1
494999  11000111000100010001000HLHH1
499999  10000111000100000001000HLHH1
504999  11000111000100010001111HLHL1
509999  10000111000100000001111HLHL1
514999  11000111000100011001111HLHL1
519999  10000111000100001000111HLHL1
524999  11000111000100011000111HLHL1
529999  10000111000100001000111HLHL1
534999  11000111000100011000111HLHL1
539999  10000111000100001001111HLHL1
544999  11001111000100011001111HHHL1
549999  10001111000100000001000HHHH1
554999  11001111000100010001000HHHH1
559999  10001111000100000001000HHHH1
564999  11001111000100010001000HHHH1
569999  10001110000100000000000LHHH1
574999  11001110000100010000000LHHH1
579999  10001110000100000000000LHHH1
584999  11001110000100010000000LHHH1
589999  10001110000100000000000LHHH1
594999  11001110000100010001000LHHH1
599999  10001110000100000001000LHHH1
604999  11001110000100010001000LHHH1
609999  10001110000100000001000LHHH1
614999  11001110000100010001111LHHL1
619999  10001110000100000001111LHHL1
624999  11001110000110010001111LHHL1
629999  10001110000110000000111LHHL1
634999  11001110000110010000111LHHL1
639999  10001110000110000000111LHHL1
644999  11001110000110010000111LHHL1
649999  10001110000110000001111LHHL1
654999  11001111000110010001111HHHL1
659999  10001111000100000001000HHHH1
664999  11001111000100010001000HHHH1
669999  10001111000100000001000HHHH1
674999  11010000000100010001000HLHH1
679999  10010000000100000000000HLHH1
684999  11010000000100010000000LHLH1
```

```
 689999  10010000000100000000000LHLH1
 694999  11010000000100010000000LHLH1
 699999  10010000000100000000000LHLH1
 704999  11010000000100010001000LHLH1
 709999  10010000000100000001000LHLH1
 714999  11010000000100010001000LHLH1
 719999  10010000000100000001000LHLH1
 724999  11010000000100010001111LHLL1
 729999  10010000000100000001111LHLL1
 734999  11010000000110010101111LHLL1
 739999  10010000000110000100111HHLL1
 744999  11010000000110010100111HHLL1
 749999  10010000000110000100111HHLL1
 754999  11010000000110010100111HHLL1
 759999  10010000000110000101111HHLL1
 764999  11000000000110010101111HHLL1
 769999  10000000000100000001000HHLH1
 774999  11000000000100010001000HHLH1
 779999  10000000000100000001000HHLH1
 784999  11000000000100010001000HHLH1
 789999  10000000000100000000000HHLH1
 794999  11000000000100010000000HHLH1
 799999  10000000000100000000000HHLH1
 804999  11000000000100010000000HHLH1
 809999  10000000000100000000000HHLH1
 814999  11000000000100010001000HHLH1
 819999  10000000000100000001000HHLH1
 824999  11000000000100010001000HHLH1
 829999  10000000000100000001000HHLH1
 834999  11000000000100010001111HHLL1
 839999  10000000000100000001111HHLL1
 844999  11000000000100010101111HHLL1
 849999  10000000000100000100111LLHL1
 854999  11000000000100010100111LLHL1
 859999  10000000000100000100111LLHL1
 864999  11000000000100010100111LLHL1
 869999  10000000000100000101111LLHL1
 874999  11000010000100010101111LLHL1
 879999  10000010000100000001000LLHH1
 884999  11000010000100010001000LLHH1
 889999  10000010000100000001000LLHH1
 894999  11000010000100010001000LLHH1
 899999  10000010000100000000000LLHH1
 904999  11000010000100010000000LLHH1
 909999  10000010000100000000000LLHH1
 914999  11000010000100010000000LLHH1
 919999  10000010000100000000000LLHH1
 924999  11000010000100010001000LLHH1
 929999  10000010000100000001000LLHH1
 934999  11000010000100010001000LLHH1
 939999  10000010000100000001000LLHH1
 944999  11000010000100010001111LLHL1
 949999  10000010000100000001111LLHL1
 954999  11000010000110011001111LLHL1
 959999  10000010000110001000111HLHL1
 964999  11000010000110011000111HLHL1
 969999  10000010000110001000111HLHL1
 974999  11000010000110011000111HLHL1
 979999  10000010000110001001111HLHL1
 984999  11000110000110011001111HLHL1
 989999  10000110000100000001000HLHH1
 994999  11000110000100010001000HLHH1
 999999  10000110000100000001000HLHH1
1004999  11000110000100010001000HLHH1
1009999  10000110000100000000000HLHH1
1014999  11000110000100010000000HLHH1
1019999  10000110000100000000000HLHH1
1024999  11000110000100010000000HLHH1
1029999  10000110000100000000000HLHH1
1034999  11000110000100010001000HLHH1
1039999  10000110000100000001000HLHH1
1044999  11000110000100010001000HLHH1
1049999  10000110000100000001000HLHH1
1054999  11000110000100010001111HLHL1
1059999  10000110000100000001111HLHL1
1064999  11000110000100011001111HLHL1
1069999  10000110000100001000111HLHL1
```

```
1074999  11000110000100011000111HLHL1
1079999  10000110000100001000111HLHL1
1084999  11000110000100011000111HLHL1
1089999  10000110000100001001111HLHL1
1094999  11001110000100011001111LHHL1
1099999  10001110000100000001000LHHH1
1104999  11001110000100010001000LHHH1
1109999  10001110000100000001000LHHH1
1114999  11001110000100010001000LHHH1
1119999  10001110000100000000000LHHH1
1124999  11001110000100010000000LHHH1
1129999  10001110000100000000000LHHH1
1134999  11001110000100010000000LHHH1
1139999  10001110000100000000000LHHH1
1144999  11001110000100010001000LHHH1
1149999  10001110000100000001000LHHH1
1154999  11001110000100010001000LHHH1
1159999  10001110000100000001000LHHH1
1164999  11001110000100010001111LHHL1
1169999  10001110000100000001111LHHL1
1174999  11001110000110010001111LHHL1
1179999  10001110000110000000111LHHL1
1184999  11001110000110010000111LHHL1
1189999  10001110000110000000111LHHL1
1194999  11001110000110010000111LHHL1
1199999  10001110000110000001111LHHL1
1204999  11001111000110010001111HHHL1
1209999  10001111000100000001000HHHH1
1214999  11001111000100010001000HHHH1
1219999  10001111000100000001000HHHH1
```

A.6: 0.8um Verilog Test Fixture

```
‘timescale 1ns/1ps
module test;

      reg IN1_reg ;
      reg IN2_reg ;
      reg IN3_reg ;
      reg IN4_reg ;
      reg IN5_reg ;
      reg IN6_reg ;
      reg IN7_reg ;
      reg IN8_reg ;
      reg IN12_reg ;
      reg IN9_reg ;
      reg IN10_reg ;
      reg IN11_reg ;
      reg IN13_reg ;
      reg IN14_reg ;
      reg IN15_reg ;
      wire OUT1,OUT2,OUT3,OUT4 ;

        reg     [1:28]  test_vector[1:244];
        wire    [1:28]  current_vector;
integer vector_count;

        // Current Test Vector IN11signment
        assign current_vector = test_vector[vector_count];



        // Load Test Vectors Into Memory
        initial begin
‘ifdef SD1 $sdf_annotate(“../../sdf/chip.sd1”,test.dut,,”sdf.log”,);
‘endif
‘ifdef SD2 $sdf_annotate(“../../sdf/chip.sd2”,test.dut,,”sdf.log”,);
‘endif
‘ifdef SST
  begin
    $recordfile(“verilog”);
    $recordvars;
  end
‘endif
                $display(“--> Loading test vectors...”);
                $readmemb(“../testfixtures/CHIPF.vec”, test_vector);
                $display(“--> Test vectors successfully loaded”);

        for(vector_count=1; vector_count<244; vector_count=vector_count+1) begin
        ‘ifdef FAULT
          $fs_strobe(OUT1,OUT2,OUT3,OUT4);
        ‘endif
                        // assign Test Vectors
‘ifdef FAULT
#0.1
‘endif

                        assign_inputs;
                        #9999

                        // Log Signals and Determine Errors
strobe;
                        #1
                        // Display Status
                        if((vector_count%(244/10)==0)) begin
                              $write(“%.0f percent complete...\n”,((vector_count/(244/
10))*10));
                        end
        end



$finish;
end
     CHIP  dut      (
                .OUT1(OUT1),
                .OUT2(OUT2),
```

```
                .OUT3(OUT3),
                .OUT4(OUT4),
                .IN1(IN1_reg),
                .IN2(IN2_reg),
                .IN3(IN3_reg),
                .IN4(IN4_reg),
                .IN5(IN5_reg),
                .IN6(IN6_reg),
                .IN7(IN7_reg),
                .IN8(IN8_reg),
                .IN12(IN12_reg),
                .IN9(IN9_reg),
                .IN10(IN10_reg),
                .IN11(IN11_reg),
                .IN13(IN13_reg),
                .IN14(IN14_reg),
                .IN15(IN15_reg));

task assign assign_inputs;
begin
      IN1_reg <= #15 current_vector[2] ;
      IN2_reg <= current_vector[4] ;
      IN3_reg <= current_vector[5] ;
      IN4_reg <= current_vector[6] ;
      IN5_reg <= current_vector[7] ;
      IN6_reg <= current_vector[8] ;
      IN7_reg <= current_vector[12] ;
      IN8_reg <= current_vector[13] ;
      IN12_reg <= #15 current_vector[16] ;
      IN9_reg <= current_vector[17] ;
      IN10_reg <= current_vector[18] ;
      IN11_reg <= current_vector[20] ;
      IN13_reg <= current_vector[21] ;
      IN14_reg <= current_vector[22] ;
      IN15_reg <= current_vector[23] ;
end
endtask

task strobe;
begin

  if (OUT1 !== current_vector[24]) $write("\t...ERROR vector %d @ OUT1\t%x should be
%x\n",vector_count,OUT1,current_vector[24]);
  if (OUT2 !== current_vector[25]) $write("\t...ERROR vector %d @ OUT2\t%x should be
%x\n",vector_count,OUT2,current_vector[25]);
  if (OUT3 !== current_vector[26]) $write("\t...ERROR vector %d @ OUT3\t%x should be
%x\n",vector_count,OUT3,current_vector[26]);
  if (OUT4 !== current_vector[27]) $write("\t...ERROR vector %d @ OUT4\t%x should be
%x\n",vector_count,OUT4,current_vector[27]);

end
endtask


endmodule
```

A.7: 0.8um Verilog Testfixture Test Vectors

```
10001111000000000000100011111
10001111000000000000100011111
11001111000000001000100011111
10001111000000000000100011111
11001111000000001000100011111
10001111000000000000100011111
11001111000000001000100011111
10001111000000000000100011111
11001111000000001000100011111
10001111000000000000100011111
11001111000000001000100011111
10001111000000000000100011111
11001111000000001000100011111
10001111000000000000100011111
11001111000000001000100011111
10001111000100000000100011111
11001111000100010001000011111
10001111000100000000100011111
11001111000100010001000011111
10001111000100000000100011111
11001111000100010001000011111
10001111000100000000100111111
11011111000100010001001011111
10011111000100000000001111111
11011111000100010000010001011
10011111000100000000010101011
11011111000100010000011001011
10011111000100000000000001011
11011111000100010001000001011
10011111000100000000100001011
11011111000100010001000001011
10011111000100000000100001011
11011111000100010001000111101001
10011111000100000000111101001
11011111000110010101011101001
10011111000110000010011111101
11011111000110010100111111101
10011111000110000100111111101
11011111000110010100111111101
10011111000110000101111111101
11001111000110010101111111101
10001111000100000000100011111
11001111000100010001000011111
10001111000100000000100011111
11001011000100010001000011111
10001011000100000000000011111
11001011000100010000000000111
10001011000100000000000000111
11001011000100010000000000111
10001011000100000000000000111
11001011000100010001000000111
10001011000100000000100000111
11001011000100010001000000111
10001011000100000000100000111
11001011000100010001000111100101
10001011000100000000111100101
11001011000110011001000111100101
10001011000110001001000111111101
11001011000110011001000111111101
10001011000110001001000111111101
11001011000110011001000111111101
10001011000110001001001111111101
11001111000110011001000111111101
10001111000100000000100011111111
11001111000100010001000011111111
10001111000100000000100011111111
11001101000100010001000011111111
10001101000100000000000011111111
11001101000100010001000000011011
10001101000100000000000011011011
11001101000100010001000000011011
10001101000100000000000011011011
```

```
11001101000100010001000110 11
10001101000100000000100011011
11001101000100010001000110 11
10001101000100000000100011011
11001101000100010001111111001
10001101000100000000111111001
11001101000100010101111111001
10001101000100000010011111101
11001101000100010100111111101
10001101000100000010011111101
11001101000100010100111111101
10001101000100000010111111101
11001111000100010101111111101
10001111000100000000100011111
11001111000100010001000111111
10001111000100000000100011111
11001111000100010001000111111
10000111000100000000000010111
11000111000100010000000010111
10000111000100000000000010111
11000111000100010000000010111
10000111000100000000000010111
11000111000100010001000010111
10000111000100000000100010111
11000111000100010001000010111
10000111000100000000100010111
11000111000100010001111110101
10000111000100000000111110101
11000111000100011001111110101
10000111000100001000111110101
11000111000100011000111110101
10000111000100001000111110101
11000111000100010001111111101
10001111000100000000100011111
11001111000100010001000111111
10001111000100000000100011111
11001111000100010001000111111
10001110000100000000000001111
11001110000100010000000001111
10001110000100000000000001111
11001110000100010000000001111
10001110000100000000000001111
11001110000100010001000001111
10001110000100000000100001111
11001110000100010001000001111
10001110000100000000100001111
11001110000100010001111101101
10001110000100000000111101101
11001110000110010001111101101
10001110000110000000111101101
11001110000110010000111101101
10001110000110000000111101101
11001110000110010001111101101
10001110000110000000111101101
11001111000100010001000111111
10001111000100000000100011111
11001111000100010001000111111
10001111000100000000100011111
11010000000100010001000110111
10010000000100000000000010111
11010000000100010000000001011
10010000000100000000000001011
11010000000100010000000001011
10010000000100000000000001011
11010000000100010001000010111
10010000000100000001000010111
11010000000100010001000010111
10010000000100000001000010111
11010000000100010001111101001
10010000000100000000111101001
11010000000110010010111101001
10010000000110000010011111001
11010000000110010010011111001
10010000000110000010011111001
11010000000110010010011111001
```

```
10010000000110000101111111001
11000000000110010101111111001
10000000000100000000100011011
11000000000100010001100011011
10000000000100000000100011011
11000000000100010001100011011
10000000000100000000000011011
11000000000100010000000011011
10000000000100000000000011011
11000000000100010000000011011
10000000000100000000000011011
11000000000100010001100011011
10000000000100000000100011011
11000000000100010001100011011
10000000000100000000100011011
11000000000100010001111111001
10000000000100000000111111001
11000000000100010101111111001
10000000000100000010011100101
11000000000100010100111100101
10000000000100000010011100101
11000000000100010100111100101
10000000000100000010111100101
11000010000100010101111100101
10000010000100000000100000111
11000010000100010001100000111
10000010000100000000100000111
11000010000100010001100000111
10000010000100000000000000111
11000010000100010000000000111
10000010000100000000000000111
11000010000100010000000000111
10000010000100000000000000111
11000010000100010001100000111
10000010000100000000100000111
11000010000100010001100000111
10000010000100000000100000111
11000010000100010001111100101
10000010000100000000111100101
11000010000110011001111100101
10000010000110001000111110101
11000010000110011000111110101
10000010000110001000111110101
11000010000110011000111110101
10000010000110001001111110101
11000110000110011001111110101
10000110000100000000100010111
11000110000100010001100010111
10000110000100000000100010111
11000110000100010001100010111
10000110000100000000000010111
11000110000100010000000010111
10000110000100000000000010111
11000110000100010000000010111
10000110000100000000000010111
11000110000100010001100010111
10000110000100000000100010111
11000110000100010001100010111
10000110000100000000100010111
11000110000100010001111110101
10000110000100000000111110101
11000110000100011001111110101
10000110000100000100111110101
11000110000100011000111110101
10000110000100000100111110101
11000110000100011001111110101
10000110000100000100111110101
11001110000100011001111101101
10001110000100000000100001111
11001110000100010001100001111
10001110000100000000100001111
11001110000100010001100001111
10001110000100000000000001111
11001110000100010000000001111
10001110000100000000000001111
11001110000100010001000000111
10001110000100000000000001111
```

```
11001110000100010000100001111
10001110000100000000100001111
11001110000100010000100001111
10001110000100000000100001111
11001110000100010000111101101
10001110000100000000111101101
11001110000110010000111101101
10001110000110000000011101101
11001110000110010000011101101
10001110000110000000011101101
11001110000110010000011101101
10001110000110000000111101101
11001111000110010000111111101
10001111000100000000100011111
11001111000100010000100011111
10001111000100000000100011111
```

A.8: Synthesis Logic Optimization Script

```
verilogout_no_tri=true

define_name_rules flex_migration -restricted “#[]/”

read -format verilog ./.2_to_0.8_maplib.vs
read -format verilog ../net/chip.rtl.v

current_design CHIP

link

uniquify
ungroup -all

set_dont_touch i_por
set_dont_touch GLOBAL_IN7

set_max_area 0
set_operating_conditions -max WCMIL


/* library is e8v5s */

compile -ungroup_all

report_area

change_names -rules flex_migration -verbose

write -format verilog -hierarchy -output ../net/chip.syn.v

exit
```

A.9: 0.8um Optimized Synthesized Netlist

```
module CHIP ( OUT1, OUT2, OUT3, OUT4, IN1, IN2, IN3, IN4, IN5,
    IN6, IN7, IN8, IN12, IN9, IN10, IN11, IN13, IN14, IN15 );
input  IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8, IN12, IN9, IN10, IN11,
    IN13, IN14, IN15;
output OUT1, OUT2, OUT3, OUT4;
    wire IN_IN13, IN5_S1, n136, LIN4F, IN_IN8, n135, IN4_S1, IN2_EDD, IN_IN6,
        LIN2F, IN_IN12, IN_IN7, IN_IN11, IN_IN14, IN_IN2, IN2_S1,
        IN4_EDD, IN_IN1, LIN5F, n133, IN_IN10, IN5_EDD, IN2_S2F, n134,
        IN_IN9, IN_IN4, IN_IN3, IN_IN5, IN_IN15, XC03_R, XC01_R, n96, n97, n98,
        n99, n100, n101, n102, n103, n104, n105, n106, n107, n108, n109, n110,
        n111, n112, n113, n114, n115, n116, n117, n118, n119, n126, n127, n128,
        n129, n130, n131, n132, net48, net47, net45;
    por i_por ( .zn(XC01_R) );
    it IN9_pad ( .pad(IN9), .z(IN_IN9) );
    it IN5_pad ( .pad(IN5), .z(IN_IN5) );
    it IN3_pad ( .pad(IN3), .z(IN_IN3) );
    it IN4_pad ( .pad(IN4), .z(IN_IN4) );
    it IN13_pad ( .pad(IN13), .z(IN_IN13) );
    it IN15_pad ( .pad(IN15), .z(IN_IN15) );
    rd25K i_IN2_pd ( .pad(IN2) );
    ru50K i_IN3_pu ( .pad(IN3) );
    it IN8_pad ( .pad(IN8), .z(IN_IN8) );
    it IN7_pad ( .pad(IN7), .z(IN_IN7) );
    it IN11_pad ( .pad(IN11), .z(IN_IN11) );
    it IN10_pad ( .pad(IN10), .z(IN_IN10) );
    it IN6_pad ( .pad(IN6), .z(IN_IN6) );
    it IN14_pad ( .pad(IN14), .z(IN_IN14) );
    ru50K i_IN6_pu ( .pad(IN6) );
    it IN2_pad ( .pad(IN2), .z(IN_IN2) );
    an2 U4 ( .b(n96), .a(n97), .z(n128) );
    no2 U5 ( .b(IN_IN11), .a(IN_IN9), .zn(n96) );
    no2 U6 ( .b(n136), .a(n131), .zn(n97) );
    oai21 U7 ( .b(n119), .a2(n130), .a1(LIN4F), .zn(n134) );
    or2 U8 ( .b(n102), .a(n115), .z(n101) );
    or2 U9 ( .b(n114), .a(n113), .z(n102) );
    or2 U10 ( .b(n112), .a(n111), .z(n103) );
    or2 U11 ( .b(LIN2F), .a(n108), .z(n104) );
```

```
    or2 U12 ( .b(IN_IN6), .a(n132), .z(n105) );
    or3 U13 ( .c(IN_IN10), .b(IN_IN11), .a(n136), .z(n106) );
    i1 U14 ( .a(n106), .zn(n107) );
    no2 U15 ( .b(n105), .a(n130), .zn(n108) );
    b1 U16 ( .a(XC03_R), .z(n109) );
    aoi21x2 U17 ( .b(n127), .a2(n129), .a1(n128), .zn(XC03_R) );
    b1 U18 ( .a(n134), .z(n110) );
    no2 U19 ( .b(n127), .a(n128), .zn(n111) );
    no2 U20 ( .b(n127), .a(IN_IN8), .zn(n112) );
    na2x2 U21 ( .b(XC01_R), .a(IN_IN7), .zn(n127) );
    no2 U22 ( .b(n127), .a(n107), .zn(n113) );
    no2 U23 ( .b(n127), .a(IN_IN8), .zn(n114) );
    no2 U24 ( .b(n127), .a(IN_IN9), .zn(n115) );
    na2 U25 ( .b(n117), .a(n116), .zn(n133) );
    i1 U26 ( .a(n104), .zn(n118) );
    na2 U27 ( .b(LIN5F), .a(n118), .zn(n116) );
    na2 U28 ( .b(n98), .a(n118), .zn(n117) );
    b2 U29 ( .a(n135), .z(n119) );
    na3x2 U30 ( .c(IN_IN13), .b(IN_IN15), .a(IN_IN14), .zn(n136) );
    cvdd U31 ( .z(n126) );
    itx3 U32 ( .pad(IN12), .z(IN_IN12) );
    itx3 U33 ( .pad(IN1), .z(IN_IN1) );
    ob8 U34 ( .a(n136), .pad(OUT4) );
    ob8 U35 ( .a(n119), .pad(OUT3) );
    ob8 U36 ( .a(n110), .pad(OUT2) );
    ob8 U37 ( .a(n133), .pad(OUT1) );
    oai21 U38 ( .b(n99), .a2(net45), .a1(IN2_S2F), .zn(IN2_EDD) );
    oai21 U39 ( .b(n100), .a2(net48), .a1(IN5_S1), .zn(IN5_EDD) );
    oai21 U40 ( .b(n98), .a2(net47), .a1(IN4_S1), .zn(IN4_EDD) );
    i1 U41 ( .a(IN_IN3), .zn(n130) );
    no2 U42 ( .b(LIN5F), .a(LIN2F), .zn(n135) );
    i1 U43 ( .a(IN_IN8), .zn(n129) );
    i1 U44 ( .a(n119), .zn(n132) );
    i1 U45 ( .a(IN_IN10), .zn(n131) );
    dpbrs XC04_XM1_M01 ( .sn(n126), .rn(XC01_R), .d(IN_IN4), .c(IN_IN12),
        .q(IN4_S1) );
    dpbrs XC03_XM1_M01 ( .sn(n126), .rn(n109), .d(IN5_EDD), .c(IN_IN1),
        .qn(n100), .q(LIN5F) );
    dpbrs XC02_XM1_M01 ( .sn(n126), .rn(XC01_R), .d(IN5_S1), .c(IN_IN1),
        .qn(net48) );
    dpbrs XC05_XM1_M01 ( .sn(n126), .rn(XC01_R), .d(IN4_S1), .c(IN_IN1),
        .qn(net47) );
    dpbrs XC07_XM1_M01 ( .sn(n126), .rn(XC01_R), .d(IN_IN2), .c(IN_IN12),
        .qn(net45), .q(IN2_S1) );
    dpbrs XC09_XM1_M01 ( .sn(n126), .rn(n103), .d(IN2_EDD), .c(IN_IN1),
        .qn(n99), .q(LIN2F) );
    dpbrs XC08_XM1_M01 ( .sn(n126), .rn(XC01_R), .d(IN2_S1), .c(IN_IN1),
        .q(IN2_S2F) );
    dpbrs XC06_XM1_M01 ( .sn(n126), .rn(n101), .d(IN4_EDD), .c(IN_IN1),
        .qn(n98), .q(LIN4F) );
    dpbrs XC01_XM1_M01 ( .sn(n126), .rn(XC01_R), .d(IN_IN5), .c(IN_IN12),
        .q(IN5_S1) );
endmodule
```

A.10: Pin File

```
CHIP Pin File - 8/29/2001
BASE84-  ;   Package PLCC028
PadPinSignal                    TypeComment

15IN3                       Iw/ Pull Up 50K Input Buffer, TTL,
2.NC                        XNo Connect
3.NC                        XNo Connect
46IN4                       IInput Buffer, TTL,
5.NC                        XNo Connect
6.NC                        XNo Connect
7.NC                        XNo Connect
87IN5                       IInput Buffer, TTL,
9.NC                        XNo Connect
10.NC                        XNo Connect
11.NC                        XNo Connect
128IN6                       Iw/ Pull Up 50K Input Buffer, TTL,
13.NC                        XNo Connect
14.NC                        XNo Connect
159VSS                       SSupply
```

```
16.NC                       XNo Connect
17.NC                       XNo Connect
1810VSS                      SSupply
19.NC                       XNo Connect
20.NC                       XNo Connect
2111NC                       XNo Connect
2212IN7                      IInput Buffer, TTL,
23.NC                       XNo Connect
24.NC                       XNo Connect
2513IN8                       IInput Buffer, TTL,
26.NC                       XNo Connect
27.NC                       XNo Connect
28.NC                       XNo Connect
2914VSS                      SSupply
30.NC                       XNo Connect
31.NC                       XNo Connect
32.NC                       XNo Connect
3315VSS                      SSupply
34.NC                       XNo Connect
35.NC                       XNo Connect
3616IN12                     IInput Buffer, TTL, 3X Strength
37.NC                       XNo Connect
38.NC                       XNo Connect
3917IN9                      IInput Buffer, TTL,
40.NC                       XNo Connect
41.NC                       XNo Connect
4218IN10                     IInput Buffer, TTL,
4319NC                       XNo Connect
44.NC                       XNo Connect
45.NC                       XNo Connect
4620IN11                     IInput Buffer, TTL,
47.NC                       XNo Connect
48.NC                       XNo Connect
4921IN13                     IInput Buffer, TTL,
50.NC                       XNo Connect
51.NC                       XNo Connect
52.NC                       XNo Connect
5322IN14                     IInput Buffer, TTL,
54.NC                       XNo Connect
55.NC                       XNo Connect
56.NC                       XNo Connect
5723IN15                     IInput Buffer, TTL,
58.NC                       XNo Connect
59.NC                       XNo Connect
6024OUT1                     OOutput Buffer, 8 mA.
61.NC                       XNo Connect
62.NC                       XNo Connect
6325OUT2                     OOutput Buffer, 8 mA.
6426OUT3                     OOutput Buffer, 8 mA.
65.NC                       XNo Connect
66.NC                       XNo Connect
6727OUT4                     OOutput Buffer, 8 mA.
68.NC                       XNo Connect
69.NC                       XNo Connect
7028VDD                      SSupply
71.NC                       XNo Connect
72.NC                       XNo Connect
73.NC                       XNo Connect
741VDD                      SSupply
75.NC                       XNo Connect
76.NC                       XNo Connect
77.NC                       XNo Connect
782IN1                      IInput Buffer, TTL, 3X Strength
79.NC                       XNo Connect
80.NC                       XNo Connect
813VSS                      SSupply
82.NC                       XNo Connect
83.NC                       XNo Connect
844IN2                      IInput Buffer, TTL, w/ Pull Down 25K
```

Vita

Marc Royer was born in Manchester, NH in 1974. He graduated from Manchester Memorial Highschool in Manchester, NH in 1992 and The Northfield Mount Hermon School in Northfield, MA in 1993. In September of 1993, he began studying Music Engineering Technology at the University of Miami School of Music in Coral Gables, FL in pursuit of a B.A degree. In 1995 he transferred to the Audio Engineeing program at the University of Miami School of Engineering to complete a B.S.E.E. degree. Marc obtained his B.S.E.E. degree in May 1997. In September 1997 he was employed by ASIC International, Inc. in Knoxville, Tennessee as an ASIC design engineer. ASIC International was aquired by Flextronics Semiconductor, a business unit of Flextronics International, Inc., in May 2001.