



Field Programmable Gate Array

Application Handbook

1992

ASIC Products



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilised to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessary performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any licence, either express or implied, is granted under any patent right, copyright, mask, work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Texas Instruments products are not intended for use in the life support appliances, devices or systems. Use of a TI product in such applications without the written consent of the appropriate TI officer is prohibited.

Copyright © 1992, Texas Instruments Incorporated

TRADEMARKS

ACT, ACTIVATOR, ACTION LOGIC, ACTIONPROBE, ALES, APS, PLICE are trademarks of ACTEL corp.

ABEL is a trademark of DATA/O corp.

Apollo is a trademark of Hewlett Packard.

CUPL is a trademark of Logical devices inc.

LOG/IC is a trademark of ISDATA incorp.

Mentor Graphics is a trademark of Mentor Graphics inc.

PGADesigner and PLDesigner are trademarks of Minc inc.

PGA is a trademark of International Business Machines.

Sparc, Sparcstation and SUN are trademarks of Sun Microsystems inc.

Viewlogic and Viewsim are trademarks of Viewlogic system inc.

Pal is a trademark of Monolithic Memories inc.

Gal is a trademark of Lattice Semiconductor Corp.

LCA is a trademark of Xilinx corp.

MAX, APLUS is a trademark of Altera corp.

Mach is a trademark of AMD corp.

Synopsys id a trademark of SYNOPSYS.

Verilog is a trademark of Cadence inc.

Texas Instruments acknowledge the trademarks of other organisations for their respective products or services mentioned in this document.

CONTENTS

SECTION 1 OVERVIEW

1.1 Why Field Programmable Gate Arrays are revolutionising logic design in the 90's	1.5
1.2 Benchmarking Field Programmable Logic Devices	1.19

SECTION 2 FPGA ARCHITECTURE

2.1 FPGA Device Architecture	2.5
2.2 FPGA Future Products	2.15

SECTION 3 FPGA CAE PLATFORMS

3.1 FPGA Design Flow With Viewlogic On PC	3.5
3.2 FPGA Design Flow With Mentor On Apollo	3.17
3.3 FPGA Design Flow	3.25
3.4 FPGA Design Flow With Viewlogic On Sun	3.31

SECTION 4 ACTION LOGIC SYSTEM

4.1 Action Logic System (ALS) Overview	4.5
4.2 Configuring Workview For FPGA Libraries	4.15
4.3 Customising Workview menus For TI Action Logic System	4.19
4.4 Understanding The ADL Netlist	4.27
4.5 Back Annotated Simulation For The Field Programmable Gate Arrays	4.33
4.6 Using The ALS Timer For Static Timing Analysis	4.39
4.7 Critical Path Analysis For Field Programmable Gate Arrays	4.51
4.8 Understanding How ALS Place and Routes	4.63
4.9 Using ALS Debug	4.75
4.10 Action Probe For Functional Debug	4.83

SECTION 5 LOGIC SYNTHESIS

5.1 Logic Synthesis For Field Programmable Gate Arrays	5.5
5.2 Action Logic Enhancer and Synthesis - ALES	5.15

SECTION 6 FPGA ASIC MIGRATION

6.1 Migrating Programmable Logic To ASIC Including Automatic Test Pattern Generation	6.5
6.2 Texas Instruments FPGA To ASIC Options	6.15
6.3 Migration Design Rules For Testability	6.25
6.4 Recommendations For Concurrent Design With FPGA and ASIC Using A Hardware Description Language	6.33

SECTION 7 FPGA PROGRAMMING AND TEST

7.1 Programming Field Programmable Gate Arrays: Manufacturing Considerations and Options	7.5
7.2 How Texas Instruments Test Field Programmable Gate Arrays	7.13
7.3 Programming and Verifying FPGA Antifuses With The Activator	7.19
7.4 Speed Enhanced FPGA's	7.25

SECTION 8 FPGA APPLICATION EXAMPLES

8.1 Incorporating a Keypad Decoder and Display into an FPGA	8.5
---	-----

SECTION 9 APPENDIX

9.1 TPC10 Series Pin Loadings	9.5
-------------------------------------	-----



SECTION ONE OVERVIEW

SECTION 1.1

WHY FIELD PROGRAMMABLE GATE ARRAYS ARE REVOLUTIONIZING LOGIC DESIGN IN THE 90'S

Dung Tu
Texas Instruments, Germany
Technical Marketing

INTRODUCTION

If the car industry had made the same progresses as the electronics industry in the last two decades then a typical car today would have the size and the weight of a box of matches, could drive several ten thousands of kilometer per hour, use one liter gasoline per 1000 km and cost perhaps 100\$. This comparison may appear exaggerated, it shows however excellently the key driving forces of the development in the electronics industry: miniaturization, performance, power consumption, cost and time-to-market.

The microelectronics revolution was enabled because significant improvements were achieved in many areas: silicon process technologies, semiconductor components, design tools, as some of the most important. Obviously, the invention of the microprocessor, the availability of affordable, high density memory chips and the progresses in logic products, which include general purpose logic products (GPL), programmable logic devices (PLD) and application specific IC (ASIC), have played the key roles. This paper will discuss the issues of logic implementation by investigating the architecture, applications, benefits and drawbacks of the different options, from GPL through PLD, FPGA to ASIC.

GENERAL PURPOSE LOGIC

In the 1970's, when the Transistor Transistor Logic IC's were introduced to the market, these were simply the product for digital design engineers. TTL's were used in nearly every application. Over the years, general purpose logic products are built in all kinds of process technologies: Standard TTL, Low Power Schottky (LS), Advanced Low Power Schottky (ALS), Standard Schottky (S), Advanced Standard Schottky (AS), FAST (F), HCMOS, Advanced CMOS Logic (ACL), BiCMOS (BCT) and Advanced BiCMOS Technology (ABT).

Figure 1 illustrates, as an example, the evolution of the popular octal bus tranceiver 245. Compared with the bipolar Low Power Schottky LS245, the Advanced BiCMOS technology ABT245 is 3x faster, the drive capability is 2.6x higher but its power consumption is reduced by a factor of 18.5. This example demonstrates the great progresses which are achieved in the process technology.

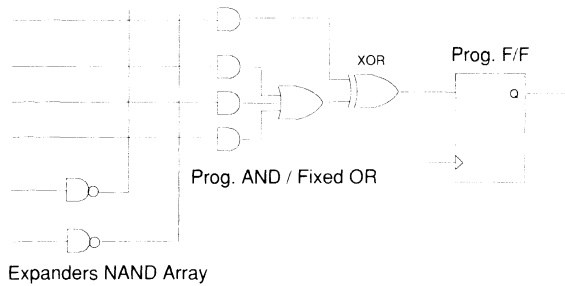


Figure 1. Process technology progress

Traditionally, GPL products can be classified into four groups: gates, flipflops, Medium/Large Scale Integration (MSI/LSI), and bus functions. With the exception of the bus functions where GPL products cannot be beaten due to their excellent drive capability, speed and cost, other GPL products do not provide the required complexity and flexibility for today new designs so they are being replaced by programmable logic devices and gate arrays. New GPL products will be mainly for wide bus functions (16 to 20 bits) using advanced BiCMOS technology.

PROGRAMMABLE LOGIC DEVICES

At the end of the 1970's, PAL (Programmable Array Logic) was introduced to the market. This product has a programmable AND array which feeds into a fixed OR array (figure 2.). The first PAL was the 16L8 - a combinatorial, 20-pin PAL with 8 outputs in bipolar technology. Instead of using several GPL logic chips like LS00, LS04, etc., to build, for example, an address decoder, a designer could put all these gate functions into one PAL. This gain of complexity versus GPL is one advantage of PAL.

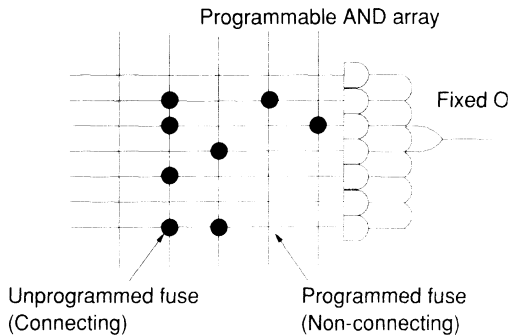


Figure 2. Programmable Array Logic (PAL)

However, what was really a big innovation, was the fact that PAL's are user-programmable. It was the first time that a normal designer can design and program an application specific chip on his own within a very short time. Determined by the speed improvements of the microprocessors, where the clock rate has grown from 4.7 MHz in the first IBM PC to 33 MHz of today PC-386 or even 50 MHz in some workstations, PAL's are becoming faster and faster. Today, 5ns PAL's are clearly the product of choice for glue logic when high speed is required. 3ns will be available soon.

At around the same time as PAL, another PLD product was taken to the market which is called Programmable Logic Array (PLA). PLA has a programmable AND array which feeds into - in contrast to PAL - a programmable OR array (figure 3.).

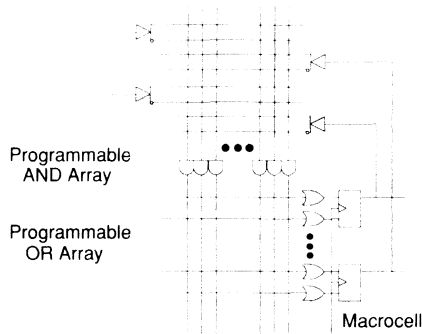


Figure 3. Programmable Logic Array (PLA)

In terms of architecture, PLA provides more flexibility than PAL because the number of product terms are not fixed but programmable. There are products with PLA architecture, the sequencers, which allow users to program up to 100 product terms to one output. This feature makes PLA suitable for complex state machine applications where the state decoding requires a lot of product terms.

Despite of the more versatile architecture and the fact that it was introduced at the same time as PAL, PLA has not been a successful product. Due to its two programmable arrays, PLA was more difficult to design with and to program. The lack of a good design tool and programming equipment in the past was one of the main reasons for the failure. This shows that good architecture without good design tool is useless for the users. Today, there are good design tools for PLA so that sequencers which allow up to 60 MHz system clock can be used for complex controller applications. Using PAL instead of PLA for such designs would require many feedback loops which increase the delay significantly.

While speed and 'quiet' signal edges are the advantages of bipolar PAL's, their disadvantages are clearly the high power consumption and not configurable output cells. The outputs of a 16L8 can be used only for combinatorial logic and cannot be configured as flipflops. The opposite is true for a 16R8. This forces the users to hold inventory of several different PAL types. These are the reasons why Generic Array Logic (GAL) (figure 4.) was taken to the market.

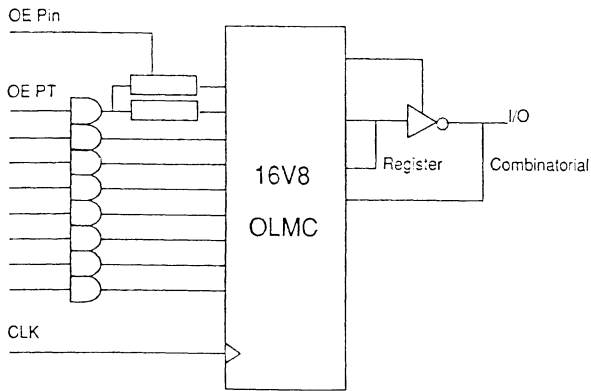


Figure 4. Generic Array Logic (GAL)

GAL's have output macrocells which can be configured as combinatorial or registered logic. Thus, a single GAL like the 16V8 can replace 24 different PAL types. GAL's are built in CMOS which reduces the power consumption by 2x compared to bipolar PAL's. However, one problem of GAL's is the fast edge rate which can cause a lot of noise on the board. Therefore GAL's cannot replace bipolar PAL's in old designs; they can only be used for new designs where the board is built to work with the fast edges.

While GAL's have the same pin counts as PAL's (20 and 24), the Erasable Programmable Logic Devices (EPLD) provide pin count from 20 up to 68 pins. Improved macrocell - the register is programmable as D, T, JK or SR flipflop, up to four independent clocks, half the power consumption of GAL's, zero standby power, higher pin counts, packed with an attractive low cost and easy to use development system are the concept for the success of EPLD's. EPLD's are not fast, but due to the low power consumption they are very successful in telecom and industrial applications. The biggest EPLD, the EP1810, which is equivalent to four 16V8 in one 68 pin package is the first successful approach to make a big PAL. However, EPLD's still have the typical programmable AND/fixed OR arrays. The problem of this architecture is the low gate utilisation.

Figure 5. shows a macrocell which is used as an address decoder. The macrocell has 8 product terms but only one of them is used. The macrocell has a dedicated flipflop which is bypassed. On the other side, when an application really needs more then 8 product terms, the user must use feedback loops which slow down the design significantly.

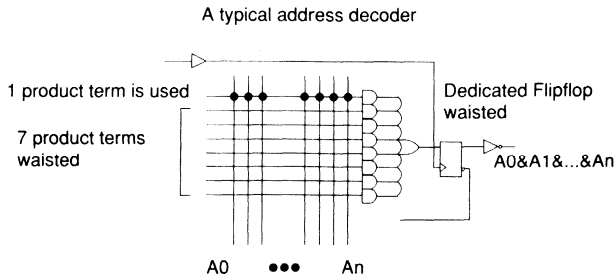


Figure 5. Macrocell as an address decoder

To overcome these problems, new products like MAX (Multiple Array Matrix) or MACH (Macro Array CMOS High-density) are taken to the market in the last two years. These products are different in details, in general however, they can be considered as an approach to combine both PAL and PLA in one product which can be called 'multilevel arrays'.

Figure 6. depicts the MAX architecture. The macrocell has only three product terms instead of eight. When more than three are required, additional NAND gates from a so called expander array can be switched to the macrocell. The macrocell itself has AND/OR structure so that it can emulate PAL. However, because the output of the OR gate is connected to a XOR, the combination AND/OR/XOR can be configured as a NAND. Thus, the NAND expander term plus the (as NAND) configurable macrocell build a NAND/NAND structure which is equivalent to AND/OR. Because both expander and macrocell arrays are programmable MAX emulates also the PLA architecture.

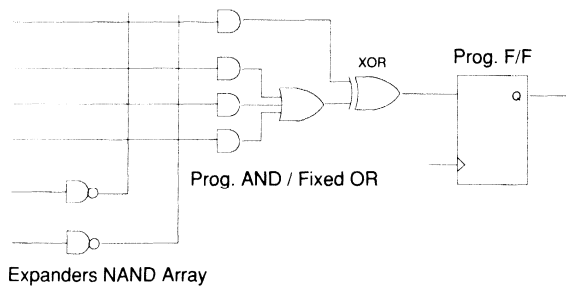


Figure 6. Multiple Array Matrix (MAX)

Instead of expanders, MACH devices use a logic allocator to allocate the 64 available product terms to the macrocells. To a certain extent, MACH can be considered as multiple-22V10 devices. Compared to EPLD's, MAX and MACH are a step forward in terms of better gate utilization. However, because the macrocell is a relatively big building block it would be difficult to make more than several hundred macrocells (flipflops) on one chip with an acceptable yield. The macrocell based MAX and MACH are apparently rather big PLD's than programmable gate arrays.

FIELD PROGRAMMABLE GATE ARRAYS

Field Programmable Gate Arrays belong to the most exciting logic products today. Three products with different architectures are available on the market: Logic Cell Array (LCA) from Xilinx, FPGA from Actel/TI/Masushita/HP and Electrically Reconfigurable Array (ERA) from Plessey. Toshiba has announced shortly a product which has the same architecture as ERA.

The driving forces for the development of field programmable gate arrays are basically the same as for the whole electronics industry but time-to-market and density are apparently the two key elements. Compared to classic gate arrays, the FPGA chip price is 10x higher. The user gets, however, an user programmable chip with a density which is ten to 100 times higher than PAL's and which allows him to make a gate array on his own within a week, without paying any non-recurring engineering charges (NRE). This reduces the risk for new designs significantly which can ultimately result in considerable cost reduction. For low volume, FPGA's are in fact very cost efficient. The concepts of LCA, FPGA and ERA show the different approaches to achieve these goals.

Figures 7a-7c show the architectures. The logic core of LCA is called Configurable Logic Blocks (CLB); for the interconnection between the CLB's, programmable interconnection points (PIP) and switch matrix are used. The configuration data must be loaded from an external EPROM to a SRAM based configuration memory.

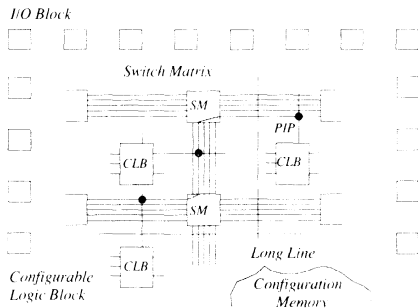


Figure 7a. Logic Cell Array (LCA)

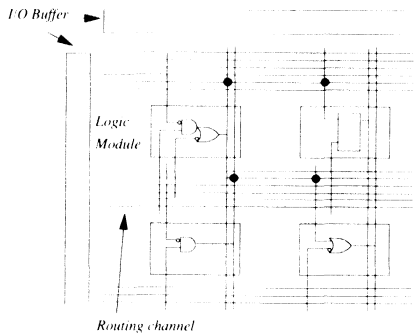


Figure 7b. Field Programmable Gate Array (FPGA)

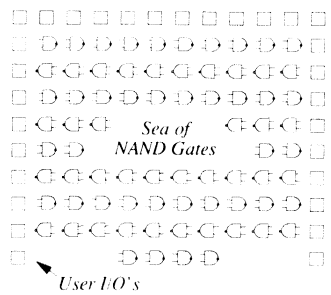


Figure 7c. Electrically Reconfigurable Array (ERA)

FPGA from Actel/TI has channeled gate array architecture with logic modules as building blocks. Non-volatile antifuses and routing channels are used for the interconnection. In contrast to FPGA, Plessey ERA has sea-of-gates architecture. The logic core is a sea of two-input NAND cells. Local interconnect lines are used to connect adjacent cells, long busses are used to make connections across the chip. Configuration data is stored in a SRAM memory which is part of the cells.

Figures 8a-8c show the LCA CLB, FPGA logic module and ERA basic cell. In terms of granularity, LCA CLB is the biggest and most complex building block, ERA cell is the smallest and FPGA logic module is in between.

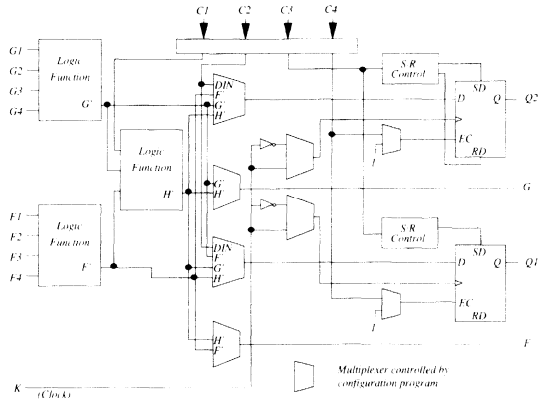


Figure 8a. Configurable Logic Block (CLB)

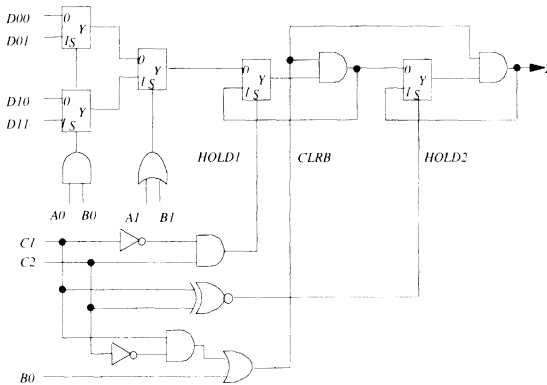


Figure 8b. FPGA Logic Module

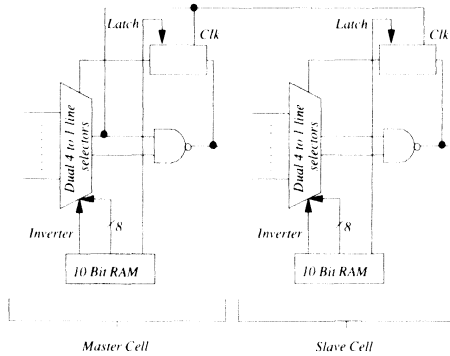


Figure 8c. ERA basic cell

It is well known that a big building block can be very efficient for applications which fit into the architecture but it can suffer otherwise from the I/O limitation problem and is more difficult to migrate to ASIC. On the other hand, a small building block leads to a high nominal gate count and is most 'gate array like' but it requires a lot of routing resources; if these are not available the utilisation is very low and place & route can become a big problem. Today, channeled architecture is the compromise which has the highest utilisation and the best routability.

Using gate count to quantify the density of programmable logic devices is not a trivial issue. Depending on the gate utilisation and the applications the number of usable gates can differ quite a lot from the nominal gate count. In fact, the total number of usable flipflops and I/O pins, which is shown in figure 9. can be used to get a rough idea about the real resources of a PLD product. Complex PLD's like MAX, MACH can provide a maximum of 300 flipflops while Field Programmable Gate Arrays can implement up to about 1000 flipflops

PLD RESOURCES

I/O PINS - MAX FLIPFLOPS (NOM.)

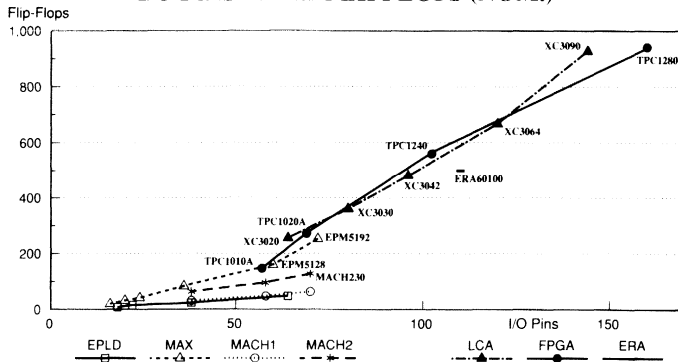


Figure 9. PLD resources

The ERA architecture is aimed at high nominal gate count to prototype gate arrays. The utilisation is however less than 30%. To be successful, the problem with routing resources and place & route must be solved. LCA and FPGA provide a complete product spectrum from 1k to about 8-10k gates. The small parts can be used for production volume up to 5k or maybe 10k units per year, the high gate count parts are useful for prototyping gate arrays.

Benchmarking FPGA's and complex PLD's is very controversial. JEDEC has defined standard tests which can be used to make a fair comparison between the various field programmable devices.

Good design tools are very essential for FPGA's. A development system for FPGA consists normally of two parts: a universal CAD environment for design entry/simulation and a product specific tool for place & route, extract timing and programming. Place & route is clearly the most important part of the tool. It can be 100% automatic and is accomplished within 30 minutes or it can run for one or two days and part of the design is not routed or it can take even weeks when the job must be done 100% manually. Due to the high complexity - the smallest FPGA can replace 10 PAL's - back-annotation simulation after place & route is very helpful to verify the timing. Because a design needs normally several modifications before it is functional, a long cycle time for place & route and back-annotation simulation has a very negative impact on the time-to-market.

The support of logic synthesis tools is also important for FPGA's. Logic synthesis improves not only the productivity of the designers but can also optimize the speed of the design. It allows a technology independant design methodology and is useful for migrating FPGA to ASIC.

ASIC

According to a forecast, in year 2000 40% of the total semiconductor in use world-wide, including microprocessors and memory chips, will be ASIC's. There is no doubt that gate arrays and standard cells have become an essential tool for making electronics systems. Without ASIC's, products like PC chip sets, ISDN, digital TV, video games, etc. are not feasible. ASIC's are used for applications with several hundred gates as well as 200k gates. If the volume is high enough, CMOS gate arrays are an unbeatable solution both in terms of performance and cost. At the high end, BiCMOS gate arrays up to 150k with true sandwich CMOS-bipolar-CMOS cells, which allow 300ps internal gate delay, fast compiler memories/datapaths (3-6ns SRAM) and more than 300 I/O's provide high performance system solutions on one or two chips. However, traditional gate level design for such complexity would be very inefficient; logic synthesis with behavioural models or VHDL is the tool of choice.

SUMMARY

Logic designers have never had in the past a bigger choice of possible solutions for a specific task as they have today. Each technology has specific benefits and drawbacks. GPL provides the best bus interface functions, bipolar PAL's the highest speed for glue logic. CMOS GAL's are cost efficient, complex multilevel-array PLD's can replace several PAL's. However, the future belongs to FPGA's and classic gate arrays. Because continuous progresses on process technology can be expected, FPGA's with combined advantages of channeled architecture and sea-of-gates in submicron CMOS/BiCMOS technology, tripple level metal, 100 MHz system clock, density up to 50k usable gates and cost not more than 0.05 cent/gate will be available in this decade.

Today, because the careabouts of the designers remain basically the same, the following requirements list can be useful for the decision process to select the right logic product for the right application.

REQUIREMENTS LIST

Density

- Total available gates
- Gate utilisation
- Total available flipflops
- Maximum available I/O pins
- Available packages
- User defined pinout/package
- RAM/ROM on chip

Performance

- Toggle rate
- System speed
- Gate delay
- Wiring delay
- I/O delay
- Clock skew
- Wiring skew
- Reconfigurability / Non-volatility
- Noise, simultaneous switching

Power consumption

- Bipolar / Cmos / Bicomos
- Drive Capability
- Stand-by Power / Dynamic Power

Cost

- Chip cost
- Multisources
- Price/performance of development system
- NRE
- Migration to ASIC
- Hardwired option
- Pre-programming service
- Inventory
- Multisources
- One chip / two chip solution
- Design protection

Time to market

- User programmable
- Good automatic place & route tool
- Ease to use
- Support popular CAD platforms, good simulator
- Support for logic synthesis tool
- Debug/testability
- Multisources
- Availability/delivery
- Technical support
- Training
- Design service

SECTION 1.2

BENCHMARKING FIELD PROGRAMMABLE LOGIC DEVICES

Glenn Bigger and Dan Powers
Texas Instruments, Dallas
FPGA Applications

ABSTRACT

The use of Field Programmable Gate Arrays (FPGAs) in the design of electronic products is growing at a rapid pace. However, some of the fundamental questions being asked by design engineers about FPGAs are not answered in terms that they can relate to easily. In addition, designers are inundated with marketing hype that further complicates the issues. FPGA device density and speed performance are often specified in terms that are general and unfamiliar to many design engineers. The pitfalls of the various FPGA architectures available in the market today are often left for the design engineers to discover by themselves at the most inopportune moments. This applications note addresses the FPGA speed performance and density issues with comparisons of the Texas Instruments TPC10 series and TPC12 series FPGAs with other devices currently available in the market. This article focuses on presenting data in terms that the design engineer can easily understand.

METHODOLOGY

The objective of this benchmarking analysis is to determine the performance and usable density of TI FPGAs as well as competitive FPGAs for an unbiased set of standard applications. The selection of designs for this evaluation was a difficult task. Any selection of designs chosen by a device manufacturer will justifiably incur skepticism. After all, won't manufacturers tend to choose designs which emphasize the best features of their device's architecture? In order to be credible to the designer, it is imperative that the designs are not biased towards any particular device architecture. For the benchmarks to be useful to the designer, they had to represent common functions used routinely in designs. To find the solution to this dilemma, we took a short walk back into history. In the early 1980s, the gate array market faced the same situation that the FPGA market faces today. Many different gate array manufacturers were publishing performance and density data for their products, but there were as many ways to measure these parameters as there were manufacturers. The designer had no way to compare the data presented by the different manufacturers. This confusion finally resulted in the Standard for Gate Array Benchmark Set adopted by the JEDEC Council. The document which is also referred to as JEDEC Standard No. 12, states its purpose as follows:

"The purpose of these benchmarks is to provide a common set of high level functions which serve as vehicles for comparing the performance of gate arrays implemented in any technology using any internal structure. These benchmarks effectively provide an unbiased measure of gate array vendors' ability to implement a desired complex function on a particular gate array at a known level of performance."

The goals of JEDEC Standard No. 12 are consistent with the goals defined for the FPGA benchmarking analysis, but questions remain as to the applicability of gate array benchmarks to the FPGA architectures. As is shown in Figure 1, the MSI functions selected by the JEDEC Council are basic building blocks commonly used in digital design. These building blocks are as integral to implementing a design in an FPGA as they are to implementing a design in a gate

array. As a result, using these functions to benchmark the different devices should aid the design engineer in understanding the achievable levels of density and speed performance for various FPGA architectures.

Benchmarks Evaluated

Benchmark #1	4-Bit ALU
Benchmark #2	16-Bit ALU
Benchmark #3	4-Bit Rotator
Benchmark #4	16-Bit Rotator
Benchmark #5	8-Bit D-Type Flip-Flop
Benchmark #6	8-Bit Up/Down Counter
Benchmark #7	3 to 8 Decoder
Benchmark #8	9-Bit Parity Generator
Benchmark #9	Combination of Benchmarks Combined Design

Figure 1. MSI Functions Benchmarked

DEVICES EVALUATED

The devices evaluated are shown in Figure 2. These are the standard speed grade devices available from the different FPGA manufacturers. Each manufacturer also offers faster speed grade devices.

Devices Evaluated

Texas Instruments	TPC1020A
Xilinx	XC3090-70
Texas Instruments	TPC1280
Xilinx	XC4005-7
Altera	EPM5128

Figure 2. Devices Evaluated

PERFORMANCE PARAMETERS

The data for the benchmark designs consists of the following parameters:

- Gate Array Equivalent Gate Count - The number of 2-input NAND gates required to implement the given benchmark in a TI TGC100 Gate Array.

- Benchmarks Per Device - A measure of utilization of the FPGA/EPLD determined by the number of times a particular benchmark circuit (i.e. 4-bit ALU) would fit into the device if the device resources, other than I/Os, could be 100% utilized. Some of the devices are I/O pin limited and this should be considered in an actual design.
- Internal Propagation Delay - A measure of how fast the benchmark circuit will operate inside the FPGA (does not include input or output buffer delays).
- External Propagation Delay - A measure of how fast the benchmark circuit will operate in a system environment (includes input and output buffer delays).
- Internal and external propagation delays are specified for worst case commercial conditions, which are VCC equal to 4.75 volts, ambient temperature equal to 70 degrees Celsius, and worst case process.

DATA

A representative sample of the eight designs benchmarked will be discussed in this application note. The designs chosen include a combinatorial design (4-bit rotator), a sequential design (8-bit register), and a design containing combinatorial and sequential elements (8-bit counter).

The 4 bit rotator is an example of a purely combinatorial circuit. This function generates a set of outputs which mirror the inputs, but rotated by a number of bits determined by the select inputs. As shown in Figure 3, 20 gates are required in a gate array technology to implement this function. Figure 4 demonstrates that the number of 4 bit rotators which can be implemented in each device range from 32 in the EPM5128 to 308 in the TPC1280. Of the devices benchmarked, those with a more granular architecture are capable of implementing more 4 bit rotator functions per device. Figure 5 displays the internal propagation delays and input pad to output pad propagation delays for a 4 bit rotator implemented in each architecture. The internal delays range from 5 ns in the TPC1020A to 37 ns in the XC4005-7. As could be expected, the pad to pad delays are somewhat higher at 31 ns for the TPC1020A to 59 ns for the XC3090-70.

Benchmark #3

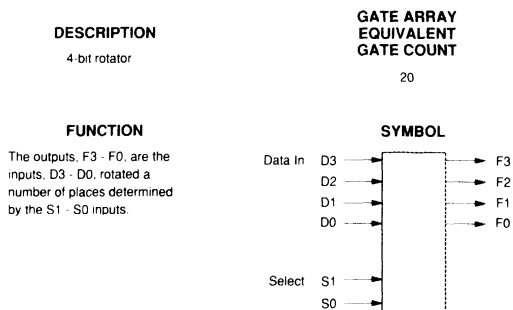


Figure 3.

Benchmarks Per Device 4-Bit Rotator - 20 Gate Array Gates

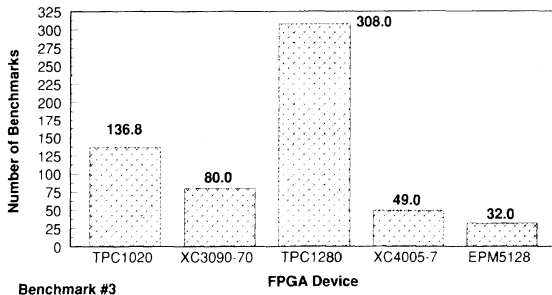


Figure 4.

Propagation Delays 4-Bit Rotator

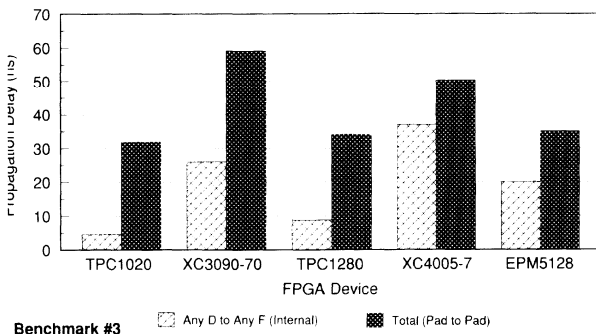


Figure 5.

An 8 bit register is used to evaluate the ability of the FPGA architectures to implement sequential logic. Figures 6, 7, and 8 display the function benchmarked, and the logic capacity and performance of each device evaluated. A total of 48 gates are required to implement an 8 bit D flip-flop register in a gate array technology. The number of 8 bit registers which can be implemented per device range from 16 in the EPM5128 to 124 in the TPC1280. Toggle frequencies of 8 bit registers actually implemented in the various architectures ranges from 33 MHz in the EPM5128 to >100 MHz in the TPC1280. However, a more realistic picture of the 8 bit register performance can be obtained from the system frequency data shown in Figure 8. Figure 9 shows the simple model used for this system. The operating frequency of the devices using the system model ranged from 20 MHz for the XC3090-70 to 33 MHz for the XC4005-7.

Benchmark #5

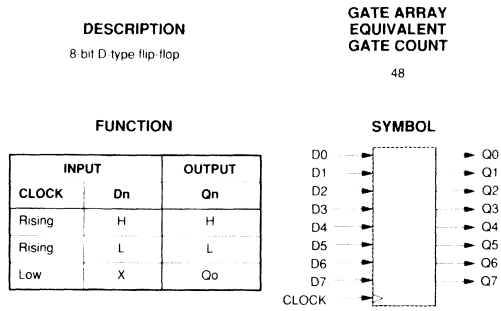


Figure 6.

Benchmarks Per Device 8-Bit Register - 48 Gate Array Gates

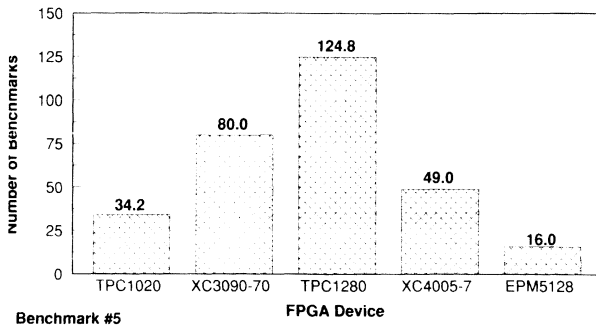


Figure 7.

Maximum Frequency 8-Bit Register

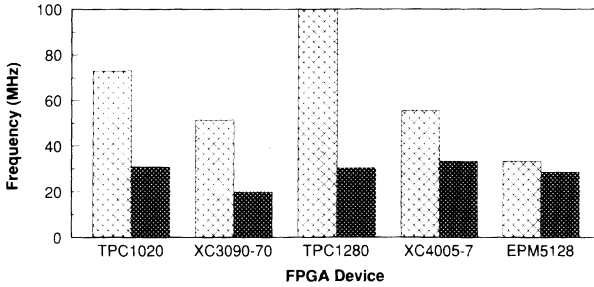


Figure 8.

The 8 bit loadable up/down counter combines the combinatorial and sequential logic capabilities of the FPGA architectures. This function is described in Figure 10, and the logic capacity and performance of each of the devices with respect to this function are shown in Figures 11 and 12. The number of 8 bit counters implemented per device ranges from 11 in the TPC1010A to 32 in the TPC1280. The performance of the 8 bit counter is based on the system clock frequency at which the counter can operate and produce a signal at the RCO signal, the longest path through the counter. The operating frequencies range from 12.5 MHz in the XC3090-70 to 20 MHz in the EPM5128.

Determining System Frequency

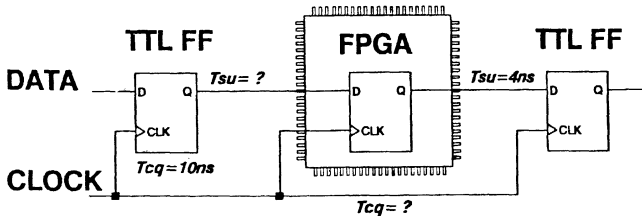


Figure 9.

Benchmark #6

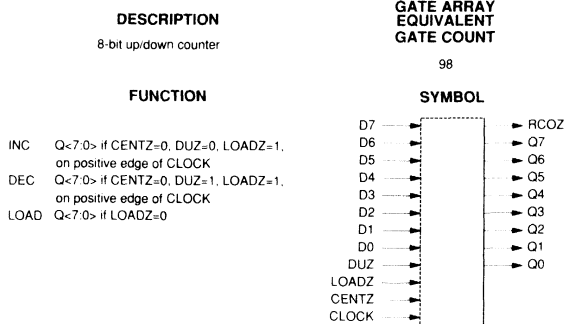


Figure 10.

Benchmarks Per Device 8-Bit Counter - 98 Gate Array Gates

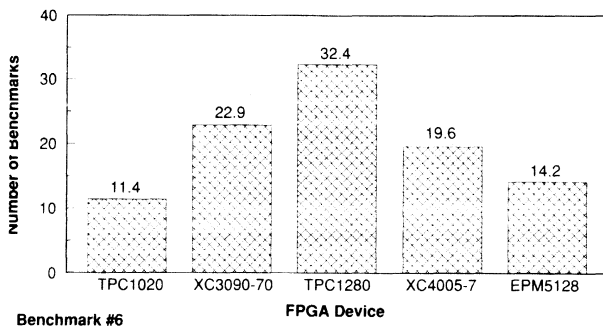


Figure 11.

Maximum Frequency 8-Bit Counter

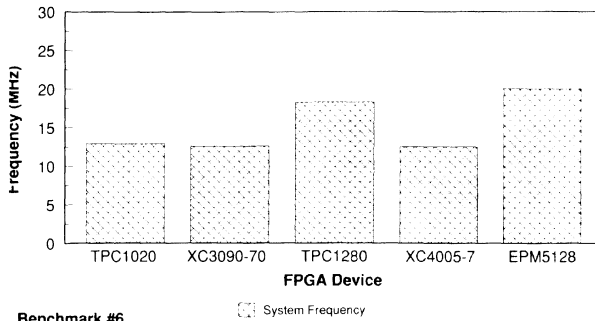


Figure 12.

SUMMARY

In today's world of digital design, the design engineer is presented with many alternatives for implementing designs. New alternatives are introduced almost daily. Not only are there more device alternatives, but for each new device, there seems to be a new set of CAE tools required. In fact, the portion of the design engineer's job before schematic capture starts, which entails choosing devices and CAE tools, has probably become more challenging than the design itself. Benchmarks are supposed to be a means to help the design engineer decide which product to purchase. However, many benchmarks are done because they enhance a particular manufacturer's device architecture. Each architecture has advantages and disadvantages in reference to other architectures. That is why the JEDEC standard benchmarks, which are comprised of commonly used MSI functions, seem to be a good place to start in evaluating various FPGA devices.

The data recorded for this benchmarking analysis show several interesting results in relation to density and speed performance. From the data, the designer can easily see the number of MSI benchmarks that will fit into the different devices. The MSI benchmarks give the design engineer a method other than gate count for evaluating device density. As shown in the data section, for each application, the TI TPC1280 provided more logic capacity than any of the other devices. The TPC1020A, Xilinx XC3090, and the Xilinx XC4005 are similar in density for most of the combinatorial benchmarks.

In the area of speed performance, the delays for the various benchmarks typically increased as the percent of utilization of the devices increased. Note the difference between internal and external delay. The input and output buffer delay can have a large impact on speed performance. Therefore, when evaluating benchmark data, the design engineer should understand whether the speed performance data includes input and output buffer delay. In addition, the system frequency at which a circuit can operate is normally much less than internal toggle frequency. The design engineer should also remember that speed performance is application specific.



SECTION TWO

FPGA ARCHITECTURE

SECTION 2.1

FPGA DEVICE ARCHITECTURE

Joel S.Lason
Texas Instruments, Dallas
FPGA Applications

INTRODUCTION

To develop a good idea of the capabilities of a Field Programmable Gate Array (FPGA), a knowledge of its architectural elements is needed. TI FPGAs have several characteristic features which define their function. These basic features are the antifuse, the logic module, the clock distribution network, routing channels, and diagnostic circuits.

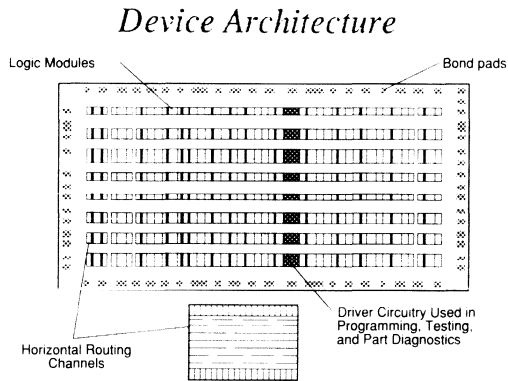


Figure 1. Device Architecture

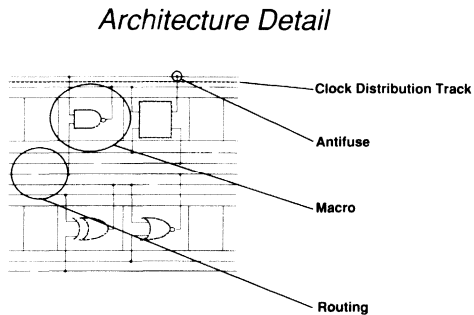


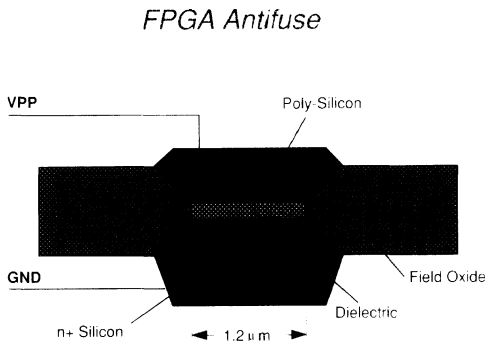
Figure 2. Architecture Detail

The two TI FPGA families are designated by the nomenclature TPC10 and TPC12. Both families share many architectural features, but the newer TPC12 family has some enhancements. A general discussion of the elements common to both devices is presented first. Then each families' unique components are covered. And finally, an example of the way that a logic module is used to implement functions is explained.

ARCHITECTURAL OVERVIEW

The Antifuse

The element which provides programmability to the device is the antifuse (Figure 3. FPGA Antifuse). An antifuse is a normally open device which becomes conductive when a high voltage pulse is applied to it. This is opposite to the action of a fuse which becomes non-conductive when a large voltage or current is applied to it. As Figure 3. shows, it consists of two conducting layers separated by a thin isolation layer. Another important feature is the small size which is up to 19 times smaller than other types of programmable elements.



Metal routing tracks run horizontally and vertically over the die and at each intersection between these metal tracks there is an antifuse. The unprogrammed resistance of the antifuse is > 100 megaohms and the programmed resistance is about 500 ohms.

Routing

As Figures 1. and 2. show, the architecture is analagous to that of a channeled gate array with routing tracks running horizontally between the logic modules and vertically over them. These tracks interconnect the macro functions implemented in the logic modules. The number of routing tracks per channel is family dependent and the segmentation of these tracks varies from device to device. Segmentation refers to the number of logic modules that a track spans before it has a break in it.

Logic Module

A prominent feature of the architecture is the rectangular array of logic modules which contain generic logic circuits. These circuits can be programmed to perform a variety of functions. Each logic module is multiplexer based. A TPC10 series device has only one type of logic module while a TPC12 series device has both the TPC10 series module and an enhanced sequential module. Sequential TPC10 series functions are implemented using combinatorial modules.

The macro functions implemented in the logic modules are generated in several ways. They come from TI supplied hard and soft macro libraries or can be user created.

Clock Distribution

High drive requirements for the clock signal are met by a dedicated clock buffer network. (Figure 4. Dedicated Clock Buffer Network). This network consists of an assigned input pin and row buffers to provide additional drive for the large fanout requirements of the clock signal. The network can connect to any logic module but only to clock and gated inputs of sequential macro functions.

Dedicated Clock Buffer Network

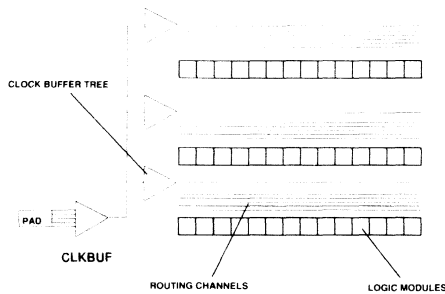


Figure 4. Dedicated Clock Buffer Network

Diagnostics

Special circuitry is built into the device for diagnostics. This circuitry consists of shift registers which allow the user to address and probe any internal node of the programmed device. Four special I/O pins are provided for this function. These pins are multiplexed so that they can be configured as diagnostic or normal I/O pins and are controlled by the Debug function of the TI Action Logic System (TI-ALS). This function allows the internal nodes to be probed while the device is operating in an actual circuit board.

I/O

I/O count depends on the device and the packaging of the device. I/O function can be programmed as either input, output, tristate, or bidirectional.

TPC10 SERIES SPECIFIC FEATURES

Logic Module

The TPC10 series logic module is shown in Figure 5. TPC10 Series Logic Module. Every logic module has 8 inputs and 1 output, and emulates the function of three 2-input multiplexers and one OR gate. Flip-flops are created by connecting a couple of modules in the appropriate circuit configuration. The small module size allows efficient mapping of logic functions and thus high device utilization.

TPC10 Series Logic Module

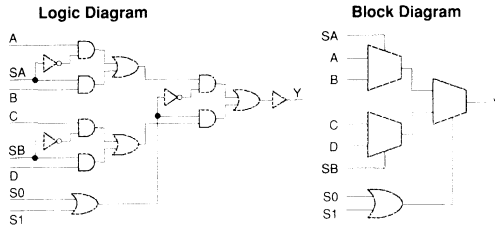


Figure 5. TPC10 Series Logic Module

Clock Distribution

A detailed diagram of clock distribution on the TPC10 series device is shown in Figure 6. TPC10 Series Clock Distribution. The TPC10 series parts have a single dedicated clock network that can hold clock skew to less than 3 ns. There are no load restrictions on the clock network and the row buffers shown help to reduce overall loading on the clock network input pin.

TPC10 Series Clock Distribution

TPC1010A

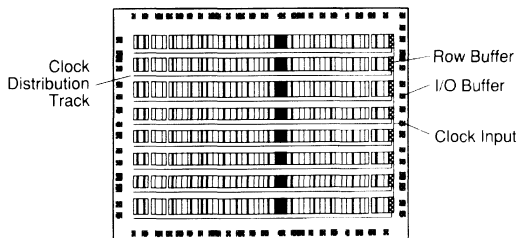


Figure 6. TPC10 Series Clock Distribution

Routing

TPC10 series devices have 25 horizontal routing tracks per channel. 22 are for logic, 1 is for clock, 1 is for VCC, and 1 is for ground. In addition, there are 13 vertical routing tracks. Tracks that extend more than a third of a row or column are referred to as long vertical tracks. These tracks can impose an additional speed penalty on the design.

TPC12 SERIES SPECIFIC FEATURES

Logic Module

Additional sequential capability has been added to the TPC12 family device in the form of a sequential module (Figure 7. TPC12 Series Architecture). On a TPC12 series die there are 624 of these enhanced modules. Each sequential module consists of the TPC10 series OR gate-multiplexer structure plus a flip-flop that is fed by the combinatorial output. There are also 608 combinatorial modules of the type that exist on the TPC10 series die. The TPC12 series combinatorial module also has an extra AND gate. Each sequential module is a complete flip-flop and a flip-flop can be made from two combinatorial modules so that total flip-flop count can be 928.

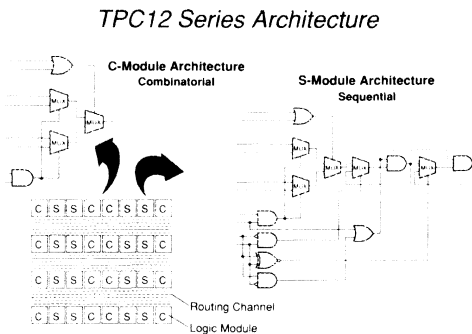


Figure 7. TPC12 Series Architecture

Clock Distribution

Clock distribution capabilities have also been enhanced for the TPC12 family. As Figure 8. TPC12 Series Clock Distribution shows, there are 2 separate clock distribution networks. Also, signals can get on the clock network either externally or internally. This means that a clock signal can be conditioned before being fed to sequential modules on the device. This conditioning could include frequency division or duty cycle modification. Clock skew can be reduced even more on the TPC12 series part because the clock signal is driven from the center of the array. Typical skew is around 1-2 ns.

TPC12 Series Clock Distribution TPC1280

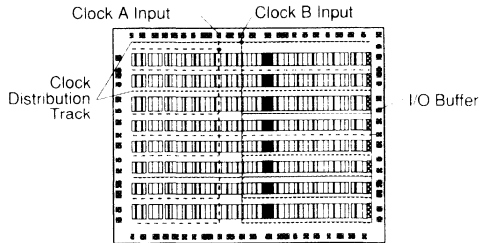


Figure 8. TPC12 Series Clock Distribution

Routing

Additional routing resources have also been added to TPC12 series parts. 36 horizontal routing tracks are available and vertical routing resources have been increased to 15.

GENERATING LOGIC FUNCTIONS FROM MULTIPLEXERS

An interesting analysis to perform is the method of implementing various logic functions using a multiplexer-based architecture. This exercise demonstrates the flexibility of this scheme but is not a necessary function for you to perform.

The signal names used in this analysis are the same as those on Figure 9.- Logic Function Implementation. It may help to refer to this figure as you read this section. The first step is to realize that the logic module is a 4-input multiplexer. Boolean simplification shows this to be true.

First, tie S0 and S1 together and call the signal B:

$$S_0 + S_1 = B + B = B$$

B is now the high address bit for the 4-input multiplexer. Then tie SA and SB together so that they become the low address bit and call this signal A. Now the equation for the multiplexer can be written as:

$$Y = A_0/A/B + A_1A/B + B_0/AB + B_1AB,$$

which is the equation for a 4-input multiplexer with signals A0, A1, B0, and B1 and address lines A and B.

Suppose the function implemented is:

$$Y = \overline{A}BC + ABC + A\overline{B}.$$

This function can be expressed in Karnaugh Map form as:

	AB	1	00	01	11	10
C	0	1	0	0	0	1
	1	1	0	1	1	1

Each column in the Map now corresponds to one AND function in the multiplexer with the AB address lines selecting the AND gate. The AND gate addressed by AB=00 has output 0 regardless of C, so its input is tied to GND. The AND gates addressed by AB=01 and 11 has output that is the same as C so its input is tied to C. The last AND gate with address AB=10 is always 1 so its input is tied to VCC. This completes the implementation of the function. The completed circuit and logic module implementation is shown in Figure 9.- Logic Function Implementation.

Logic Function Implementation

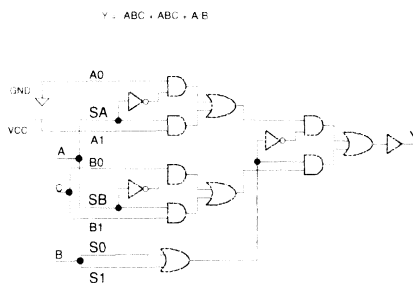


Figure 9. Logic Function Implementation

CONCLUSION

The TPC10 and TPC12 architectures provide a variety of resources which make them well-suited to logic consolidation applications. The TPC10 family is a good choice when a single high-drive clock network is needed or highly efficient mapping of logic functions is required.

The TPC12 family is useful when 2 clock networks are needed, internal access to the clock network is required, many flip-flops are used, or high speed heterogeneous combinatorial-sequential circuits are implemented. In addition, the TPC12 family meets all the needs that the TPC10 family does but offers higher density.

SECTION 2.2

FPGA FUTURE PRODUCTS

TI's longer term plan for FPGA products is set out in figure 1. The product release schedule shows how we will continue to release a major product family over each of the next three years.

FPGA Product Roadmap

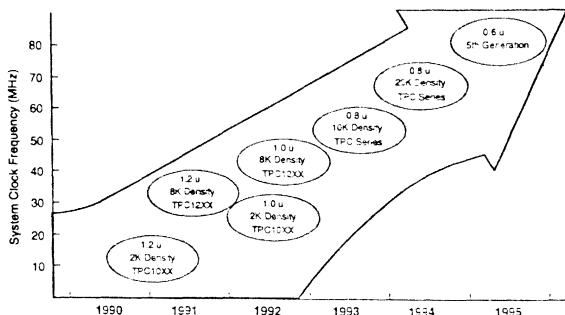


Figure 1.

Development of those future generations shown is already underway. TI will continue to invest heavily in research and development to bring subsequent devices into being to reflect changing customer demand and enhanced process capabilities.

Key performance measures for FPGAs are system speed and device capacity. As the diagram shows release plans are in place to produce devices with up to 20,000 gate array equivalent gates capable of average system speeds of over 70MHz. The primary driving force behind these improvements is the shrinking of the basic device features, and a 0.6 μ m geometry is currently being worked upon. Additionally system performance improvements are achieved through effective logic module design and useful functional features. Improvements such as those highlighted in this book as we describe the advances made in the 12 series devices over and above the 10 series will continue to be deployed as the product family extends.

Although these leading edge parameters seize the limelight we are constantly evaluating the market to see how this technology can be applied to applications requiring less complex devices. Additionally the product range is being widened with the introduction of application oriented variants alongside the basic devices. So we see alternative packages to support applications with space constraints or particular manufacturing process requirements. We have extended temperature range devices for products which operate in harsh environments, and, for time critical applications we produce or have a release schedule for higher speed variants of all the basic devices.

Easy Logic Consolidation however is not just about the devices themselves. As well as the device development we are actively working with the industry's leading CAE vendors. We will continue to enhance your productivity by providing state of the art development tools reflecting the optimum design and development methodologies and the wide range of preferences in the design community.

SECTION THREE

FPGA CAE PLATFORMS

SECTION 3.1

FPGA DESIGN FLOW With VIEWLOGIC

Doug Mackay
Texas Instruments UK
Technical Marketing

INTRODUCTION

This report describes the features of using Viewlogic on a 386DX PC for implementing a design onto a TI FPGA. A brief introduction to the design flow is given followed by a more detailed description based on a 5 bit synchronous counter application example. The simulator used is driven from its command language by a file and a comparison of both tabular and graphical output is given.

The following utilities from the Viewlogic Workview Suite of software will be described; Viewfile, Viewdraw, Viewsim and Viewwave.

VIEWLOGIC DESIGN FLOW

The basic viewlogic design flow is illustrated in figure 1.
Note that the main blocks of this are:

- Viewfile - File and Project Management.
- Viewdraw - Schematic Editor
- Viewtext - Text File Editor
- ViewSim - Interactive Logic Simulator
- Viewwave - Interactive Digital Waveform Processor

First a project directory is set using viewfile. This helps user to organize and maintain Workview's data files on a project basis. Having created the project directory the Viewdraw schematic editor is used to build the design up from the TI library of standard logic macros or user defined ones. The diagram shows that three sub-directories are created under the project for storing the sch,sym and wir files. Components, nets and labels are added building up the schematic. For large designs hierarchy can be used to make the design more manageable. Once the schematic is finished and checked for syntax errors a simulation netlist is generated ready for simulation using Viewsim.

Before the simulator is used a command file is generated which defines the format of the simulation and the stimulus being applied to the inputs. Graphical information can be described in a textual format within the file. This file could also be generated from a waveform. Having a simulation netlist and command file Viewsim is used to verify the design by performing a logical functional simulation. Both graphical and text outputs are obtainable from viewsim, the graphical output is viewed using Viewwave and the text output can be viewed using Viewtext, or any other dos editor.

After a successful functional simulation the design is ready to export into ALS. This is performed by returning to Viewdraw and selecting a MAKEADL utility which converts the workview netlist into an Advanced Design Language (ADL) netlist.

PC VIEWLOGIC DESIGN FLOW

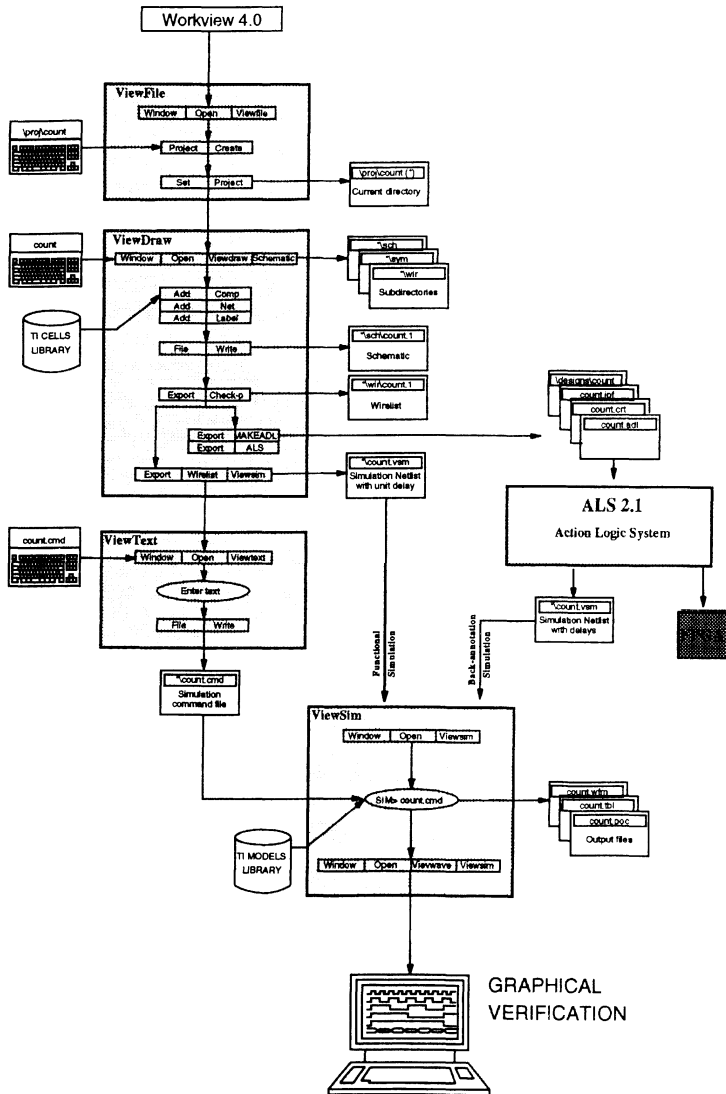


Figure 1. Design Flow

ALS is then invoked and the design is placed and routed automatically. Before programming the post-layout delays are extracted and inserted into the simulation netlist in Viewsim. This netlist is simulated again and the results compared with the pre-layout simulation. Upon successful post-layout simulation the TI FPGA can now be programmed using the TI Activator.

This method of pre and post layout simulation ensures that all the necessary steps have been taken to achieve a right-first-time design approach thus reducing the number of design iterations, and allowing the product earlier entry on the market.

CREATING A SCHEMATIC

This section will show how the schematics of the 5 bit counter application example is generated using Viewdraw.

Figure.2 shows the counter example built up from 5 d type flip-flops and combinatorial logic. No optimisation techniques have been used apart from inverted inputs to some gates. Figure 3 show the top level of the design where the counter symbol is connected to the input and output pads of the FPGA. A top down or botton up design method can be used for hierarchical designs.

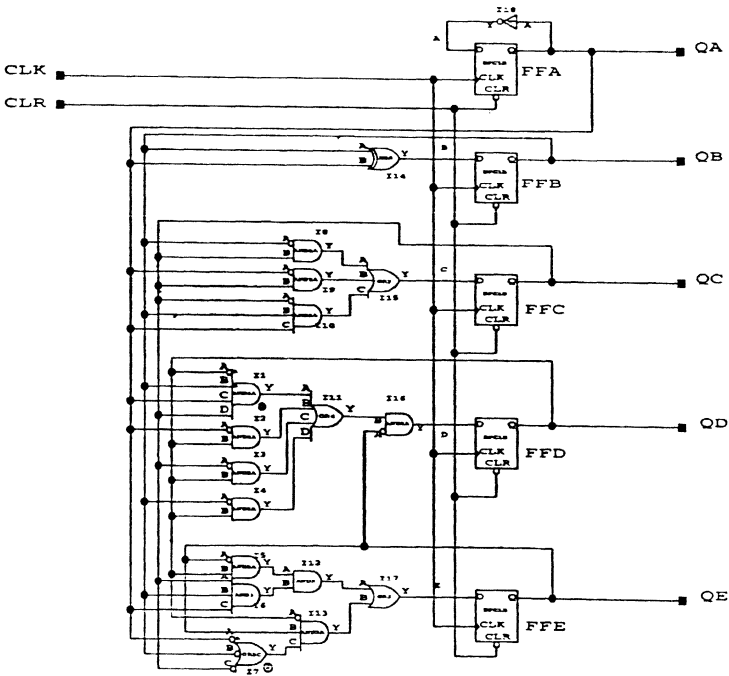


Figure 2. Counter Example

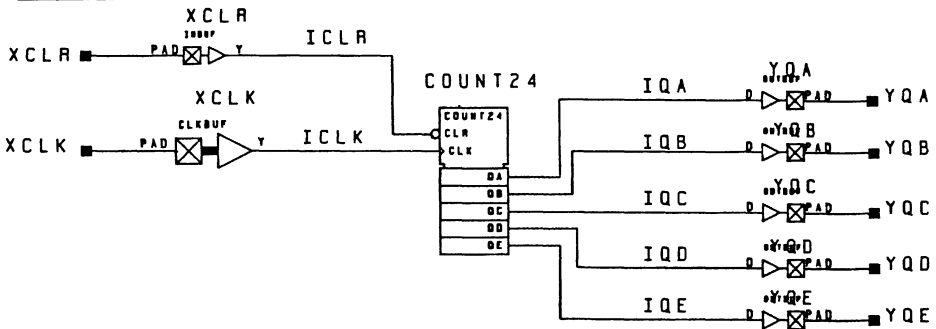


Figure 3. Top Level Schematic

For the 5 bit counter shown in Figure 2. and 3. a bottom up design approach was used.

Before starting the design you need to setup the directory structure using Viewfile, for this example \proj\count was used. Viewfile creates the necessary subdirectories and a workview.ini file. This workview.ini file is used to specify the initial setup and library search order for Workview. Viewfile can manage several projects but only one can be set to the working project.

Workview is a menu based tool where each command is selected by the middle mouse button. Figure 4 shows the interface to the user and the mouse operation when opening a schematic.

Assuming that viewfile has been used to create a working project directory Viewdraw is the Workview schematic editor. Figures 2 & 3 show the schematics which were drawn using Viewdraw.

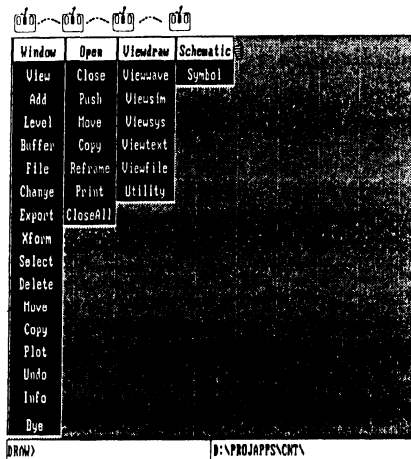


Figure 4 Workview User Interface

After opening a new schematic sheet the first part to the design process is to select the respective hard/soft macros from either the 10 series or 12 series TI FPGA library. For the schematic count 24 there are 30 hard macros used. The menu option to pick a library component is [ADD|COMPONENT]. If multiple hard macros are used then it's quicker to place one and make additional copies than selecting each component from the library.

After placing the component instances nets need to be added for continuity. The menu option [ADD|NET] will allow the nets to be drawn onto the schematic pins. The final stage in completing a schematic is to label all nets and components. This needs to be done to identify parts of the schematic easily at a later date. Omitting labels will allow workview to attach a meaningless 5 character label to the component or net.

Saving the schematic using [FILE|WRITE] will perform a simple check and create a wirelist description of the schematic.

Having completed the schematic sheet a symbol is generated allowing the sheet to be referenced higher in the hierarchy. For the example count24 the menu option [WINDOW|OPEN|VIEWDRAW|SYMBOL] was used with the same name as the schematic. Once opened a symbol body and pins were drawn, and each pin labelled with the same name given in the schematic.

To call up this schematic/symbol [ADD|COMPONENT] is selected from the menu and the symbol name given. Figure 3 shows the top level of the count design showing the user created symbol coun t24 connected to input and output pads. After the design is finished and checked a schematic wirelist is generated from the menu optio [EXPORT|WIRELIST|VIEWSIM]. This generates a file count.VSM ready for simulation.

SIMULATING A SCHEMATIC

Having generated a wirelist for simulation, Viewsim is opened in a similar way to the schematic using [WINDOW|OPEN|VIEWSIM]. Simulation commands are executed from a command file. The command file count.cmd is listed below for this design.

```
=====
| COUNT.CMD Simulation Command File for Viewsim for project count
|=====
|
| Restart
|
| Set the step size to be 50nS and a '0 - 1' 1 MHz clock
stepsize 50ns
clock xclk 0 1
|
| Clear all registers with clr active for 50nS
wfm xclr @0ns=0 @50ns=1
|
| Setup vectors ALL and QBUS
vector qbus yqe yqd yqc yqb yqa
vector all xclk xclr yqa yqb yqc yqd yqe qbus
radix dec qbus all
|
| Every 90nS print values to file TRACE.OUT
| Generate a print on change table to TRACE.POC
watch xclk xclr yqa yqb yqc yqd yqe qbus
after 90nS do(every 100ns do(print > trace.out))
break all ? do (print > trace.poc)
defaults -watch -time
|
| Generate a waveform file COUNT24.WFM for viewwave
wave count.wfm xclk xclr yqa yqb yqc yqd yqe qbus
|
| Run 26 stepsizes of simulation
cycle 26
```

Executing the command file will simulate the design using the library models for the TI FPGA. Three files are generated from this command file, a count.wfm file is written which is used by viewwave for observing and verifying the results, two trace files have been generated to show simulation vectors.

The trace.poc shows a print on change type of vector file where a vector is printed after a change in any signal. The trace.out file prints a vector after 90nS into the 100nS period. These are very useful for verifying large simulation and observing changes between pre and post layout simulation. They can also be written into a suitable format for use to allow easy import into a logic tester as test vectors.

Shown in the listings are the pre-layout and post-layout delays side by side. The effects of back-annotating post-layout delays can be clearly seen in the results of the POC files.

TRACE.POC Pre-Layout			TRACE.POC Post-Layout		
	XXXXXX	Q		XXXXXX	Q
	CCQQQQ	B		CCQQQQ	B
	LLABCDE	U		LLABCDE	U
	KR	S		KR	S
TIME	BBBBBB	D	TIME	BBBBBB	D
0.0ns	00XXXX	XX	0.0ns	00XXXX	XX
1.0ns	000000	00	29.3ns	000XXXX	XX
50.0ns	110000	00	31.7ns	0000XXX	XX
51.0ns	1110000	01	32.1ns	0000X00	XX
100.0ns	0110000	01	32.5ns	0000000	00
150.0ns	1110000	01	50.0ns	1100000	00
151.0ns	1101000	02	100.0ns	0100000	00
200.0ns	0101000	02	150.0ns	1100000	00
250.0ns	1101000	02	168.5ns	1110000	01
251.0ns	1111000	03	200.0ns	0110000	01
300.0ns	0111000	03	250.0ns	1110000	01
350.0ns	1111000	03	268.6ns	1100000	00
351.0ns	1100100	04	270.8ns	1101000	02
400.0ns	0100100	04	300.0ns	0101000	02
450.0ns	1100100	04	350.0ns	1101000	02
451.0ns	1110100	05	368.5ns	1111000	03
500.0ns	0110100	05	400.0ns	0111000	03
550.0ns	1110100	05	450.0ns	1111000	03
551.0ns	1101100	06	468.6ns	1101000	02
600.0ns	0101100	06	471.0ns	1100000	00
650.0ns	1101100	06	472.2ns	1100100	04
651.0ns	1111100	07	500.0ns	0100100	04
700.0ns	0111100	07	550.0ns	1100100	04

TRACE.OUT Pre-Layout			TRACE.OUT Post-Layout		
	XXXXXX	Q		XXXXXX	Q
	CCQQQQ	B		CCQQQQ	B
	LLABCDE	U		LLABCDE	U
	KR	S		KR	S
TIME	BBBBBB	D	TIME	BBBBBB	D
90.0ns	1110000	01	90.0ns	1100000	00
190.0ns	1101000	02	190.0ns	1110000	01
290.0ns	1111000	03	290.0ns	1101000	02
390.0ns	1100100	04	390.0ns	1111000	03
490.0ns	1110100	05	490.0ns	1100100	04
590.0ns	1101100	06	590.0ns	1110100	05
690.0ns	1111100	07	690.0ns	1101100	06
790.0ns	1100010	08	790.0ns	1111100	07
890.0ns	1110010	09	890.0ns	1100010	08
990.0ns	1101010	10	990.0ns	1110010	09
1090.0ns	1111010	11	1090.0ns	1101010	10
1190.0ns	1100110	12	1190.0ns	1111010	11
1290.0ns	1110110	13	1290.0ns	1100110	12
1390.0ns	1101110	14	1390.0ns	1110110	13
1490.0ns	1111110	15	1490.0ns	1101110	14
1590.0ns	1100001	16	1590.0ns	1111110	15
1690.0ns	1110001	17	1690.0ns	1100001	16
1790.0ns	1101001	18	1790.0ns	1110001	17
1890.0ns	1111001	19	1890.0ns	1101001	18
1990.0ns	1100101	20	1990.0ns	1111001	19
2090.0ns	1110101	21	2090.0ns	1100101	20
2190.0ns	1101101	22	2190.0ns	1110101	21
2290.0ns	1111101	23	2290.0ns	1101101	22
2390.0ns	1100000	00	2390.0ns	1111101	23
2490.0ns	1110000	01	2490.0ns	1100000	00

The waveform output count.wfm is viewed using the Viewwave utility. Figure 5 below shows the graphical output of this. On the left of the plot shows the signal names plotted followed by the graphical waveform of this signal. A cursor is displayed on the screen and moved by the mouse. This cursor is used to measure timing information and display the wave values in the Values column on the display.

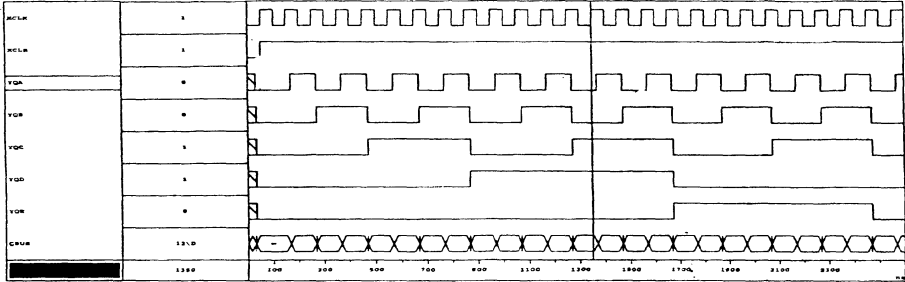


Figure 5 Simulation Waveform

This simulation was performed using the back-annotated delay information which was extracted after place and route by TI-ALS development system.

Multiple windows can be opened on the screen showing the results of the simulation and the schematic capture. An example of this is shown in figure 6. In the schematic values from the simulation are annotated into the schematic as logic values such as '1' or '0'. This is a very useful feature for debugging designs and the values of all hierarchical levels of the schematic are annotated. Figure 6 also shows the vector simulation output along with the graphical simulation output.

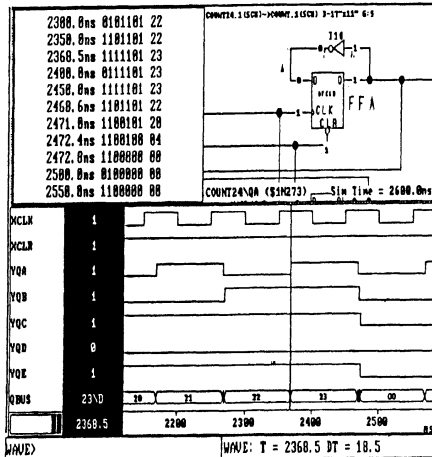


Figure 6 Workview Screen Layout

GENERATING A ADL NETLIST

Generating the ADL Netlist can either be done by executing makeadl from dos or creating a separate menu item under [EXPORT]. To do this one needs to be under viewdraw within the top level schematic.

Selecting [EXPORT|MAKEADL] will generate a new project:-
/designs/count and three files under this new directory.

```
\designs\count\count.adl  Exported ADL netlist
\designs\count\count.crt  Blank Critical path file
\designs\count\count.ipf  Initial Pin Assignment file
```

ALS then needs to be interactively used to perform validation, i/o placement, place and route and delay extraction. Post layout simulation can be performed by back annotating the extracted delays into the workview simulation netlist.

POSTLAYOUT SIMULATION

From within ALS a delay file was written containing all the net and component delays. This file is used to add delays into the simulation wirelist using workview's VSM program.

With the new simulation wirelist viewsim is used in the same way as with unit delays. A modified command file can be used generating output files with additional delays added from layout. The Viewsim simulator is a very powerful tool which allows the user to verify the design before the need to program and silicon. Then after postlayout simulation is verified the fuse file can then be used to program a TI-FPGA using an TI Activator 1 or 2.

A more detailed description to the files and procedure used in back-annotating to Viewlogic is given in the section 4.5.

DISCUSSION

This paper shows the PC Viewlogic design flow from entering a schematic to simulating with post-layout delays. A 5-bit counter example is used to show how a typical design would be implemented.

The powerful Viewsim simulator shows how easy it can be to verify large designs thus reducing the need for testing a device in the application/test-rig. With post-layout simulation and the worst and best case delay factors the designer can simulate the design under realistic conditions to observe the effect that this will have on the design. If changes are necessary they can be quickly made saving valuable time in the development of a product or system.

The suite of software, Workview, from Viewlogic is one of the most advanced PC based FPGA CAD packages and is simple and efficient to use.

SECTION 3.2

FPGA DESIGN FLOW WITH MENTOR ON APOLLO

Bryan Lawrence
Texas Instruments, UK
ASIC applications

SUMMARY

This short note is intended to describe how the MENTOR GRAPHIC(TM) CADenvironment can be used to generate a TI FPGA design. TI have developed a suite of software tools that interfaces the MENTOR CADsystem with the TI-ALS design environment, which contains all the tools necessary to generate a programmed FPGA device.

DESIGN

The FPGA development system is supplied by TI on a standard data cartridge tape and has an installation program to correctly set up the ALS/FPGA file structure.

The software runs on an APOLLO workstation under the operating system AEGIS v10.1 and above.

A user software link must then be put in place to point to this tree and the path /user/als/bin added to the csr list, which tells the operating system where to look for the programs.

When the libraries and executables have been loaded correctly, the design of an FPGA can begin.

The design to be used in this application example is a simple synchronous 5 bit counter with asynchronous clear. The schematic is given in Figure 1

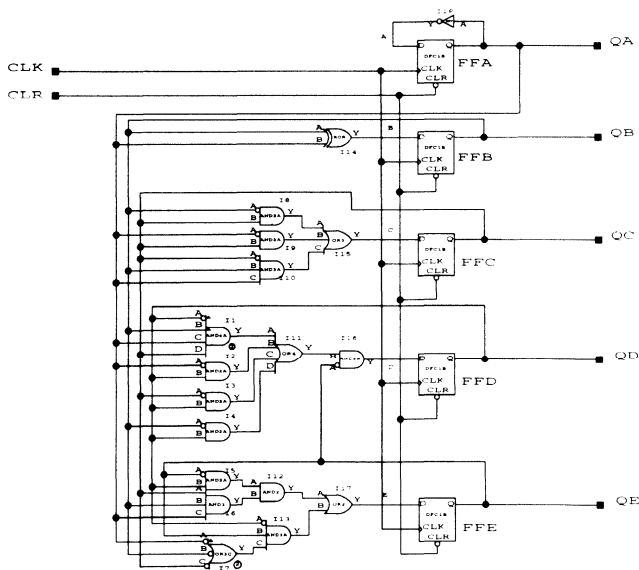


Figure 1

MENTOR NETED, the schematic capture package, is invoked and the FPGA Library is read into the NETED MENU window and this allows access to all the cells and soft macros offered by TI for design of the FPGA.

Alternatively if the Apollo system is set up to allow custom entry into neted, then a custom call e.g. neted.als would automatically set up the FPGA libraries.

A sample of the MENTOR NETED window environment is given in Figure 2. which shows primarily the FPGA library, the EDIT window and the VIEW window.

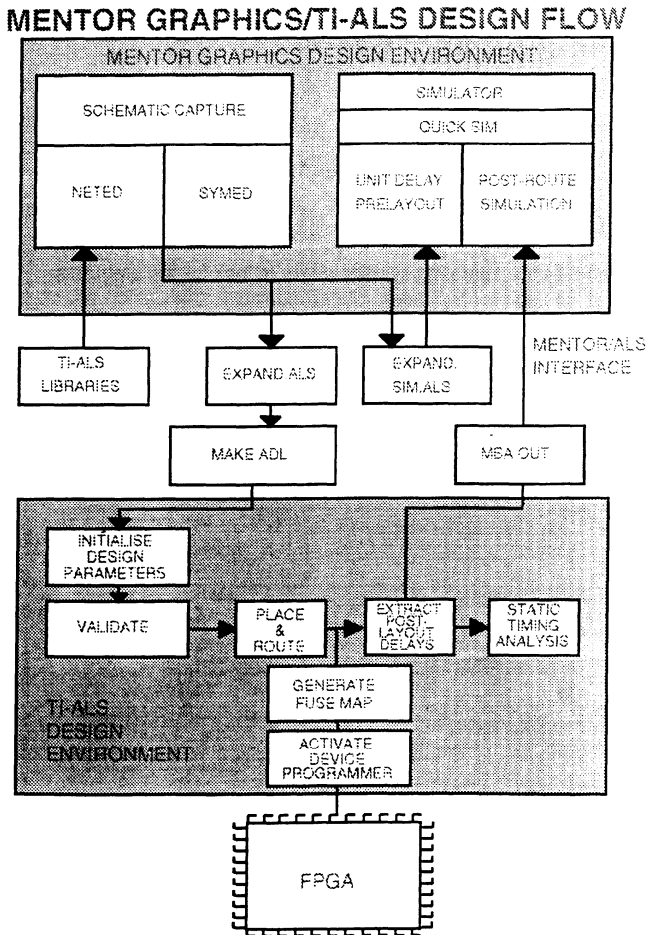


Figure 2

NOTE:

only one set of libraries should be used in a design either 10 series or 12 series, they should never be mixed.

The FPGA design environment requires that any user defined soft macro has its own symbol and this is generated using the symbol editor in MENTOR called SYMED. The symbol is required to have a special property assigned to it that allows it to be recognised by the netlist extraction tools as a user generated macro and not part of the library components.

The property LEVEL = USER is attached to the symbol body to signify this difference.

Once the design had been completed an ADL netlist description can be extracted from the MENTOR database. The ADL netlist is the description of the FPGA design in a text file format. This is done using two programs.

EXPAND.ALS <design> - This program uses the EXPAND utility of MENTOR which creates a single design database that contains all the cell interconnects to a model level required by the netlist extraction tool.

MAKEADL <design> - This program extracts a text file from the MENTOR design database of suffix .ADL. This is the netlist description of the design that interfaces to the Action Logic System.

The design can now be ported into the ALS environment.

ACTION LOGIC SYSTEM

The ALS environment is invoked by ALS <design> and it creates an ALS menu window on the Apollo. This can only be done on the node licensed to run the ALS design software.

Using the menus, the target die size, package selection and pin allocation (manual or automatic) are selected.

At this stage the delay file type is selected that will be extracted from the layout of the design and annotated to the mentor design database. The options are Military ,Industrial ,or Commercial temperature and voltage ranges,the FPGA speed rating is held at standard.

More detailed analysis can be extracted using the Static Timing Analyser which allows more specific parameters to be specified.

The ALS menu system can now be used to invoke the FPGA layout phase or alternatively|the ALS pop-up window can be dismissed and an AEGIS script file invoked in background to carry out the layout tasks.

TI have supplied an AEGIS batch program script that will validate all the design input files created during the ALS session,then assuming no errors the ALS tools will continue and place, route, optimise, extract the delay file, which contains the real interconnect delays for the design then create the programming fuse map for the design.

The program which achieves this is ALSRUN <design>.

PREPARATION FOR DESIGN VERIFICATION

Simulation of the design is done using QUICKSIM,the digital simulation package in MENTOR,and can now be carried out to verify the design timing will meet the original specification. To do this the design must be expanded into a single MENTOR database, as was

done before for the ADL netlist extraction, but includes more detailed information of the design. An ALS specific programme is supplied to do this :

EXPAND_SIM.ALS <design> - this expands the design to incorporate all the cell timing attributes for use in the simulator.

The simulation database is annotated by default with pre-layout unit delays, but TI recommend that the entire design be simulated with the real inter- connect delays before the device is programmed.

The post-layout delays are held in the delay file extracted at the layout phase of the design. The delays are annotated to the MENTOR simulation database by the use of the tool MBAOUT <design>. This program calls upon the delay parameters set up with the ALS menu and the calculated interconnect delays from layout.

The design is now ready for simulation.

SIMULATION

In order to set up the simulation environment and apply stimulus to the design MENTOR HUMAN INTERFACE MACRO LANGUAGE is recommended. This language is very simple to use and stimulus can be generated very quickly by engineers not familiar with its use, however it has powerful features for the advanced designer.

A simple example is given below for the 5-bit synchronous counter:

```
#HUMAN INTERFACE MACRO LANGUAGE

#set bus display base for hexadecimal representation radix hex

# set up bus structure for count output
DEFINE BUS COUNT QE QD QC QB
QA -COMBINE

#trace signals and look at clock -> q delays
TRACE RESET CLOCK COUNT QA QB QC QD QE

#define clock of frequency 1 Mhz and mark/space of 50:50
#all timings defined in nS
CLOCK PERIOD 1000
FORCE CLOCK 1 0 -R
FORCE CLOCK 0 500 -R

#force RESET of circuit for 3us
FORCE RESET 0 0
FORCE 1 3000

#allow circuit to run with the clock for 10uS RUN 10000

#set the waveform display window to display entire run
VIEW -ALL
```

A typical QUICKSIM display is given in Figure 3. for this design example, showing the results from the stimulus described above and how the information is presented to the designer.

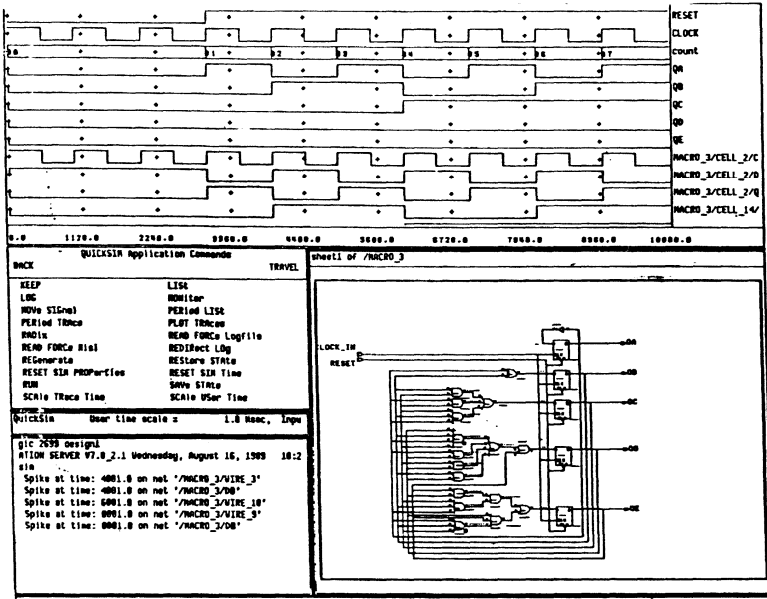


Figure 3

PROGRAMMING

To program the device developed on the APOLLO platform using MENTOR GRAPHICS and TI-ALS two options are available:

- 1: An ACTIVATOR, the FPGA programmer, driver card can be bought for the APOLLO workstation.
- 2: A separate PC ACTIVATOR system is bought along with the silicon Debugging software, called APS1, ACTIVATOR PROGRAMMING SOFTWARE.

The advantage of the PC based system is that it will allow the use of the Actionprobe and debugging features in the design lab and keep the MENTOR workstation free for other designers to use.

Conclusion

The diagram in FIG (IV) shows the flow that has been described above highlighting the software packages used and the tools within them that constitute the TI_ ALS FPGA DESIGNER.

The MENTOR/ALS design environment makes use of the workstation multitasking capabilities by the use of batch files for all the processing that has little or no interaction with the designer. This allows management of workstation time to its best advantage. The design environment matches up one of the most widely used and powerful workstation CAD systems with a new technology that can translate design ideas to silicon in hours, rather than weeks.

SECTION 3.3

FPGA DESIGN FLOW

Jim Fullerton
Texas Instruments, Dallas
FPGA Applications

INTRODUCTION

The purpose of this paper is to outline the TI Action Logic System, TI-ALS, design flow when using the Valid schematic capture and simulation software packages on a Sun Workstation. Refer to the Valid tutorials and manuals to learn how to use the Valid software.

The general TI-ALS design flow using Valid is illustrated in Figure 1. TI-ALS Valid Design Flow. ValidGED is used to enter the schematic. The schematic files may be used to create either a functional simulation netlist or a TI-ALS compatible netlist. The simulation netlist is created by using the ValidCOMPILER. The TI-ALS compatible netlist is created by using the makeadl command.

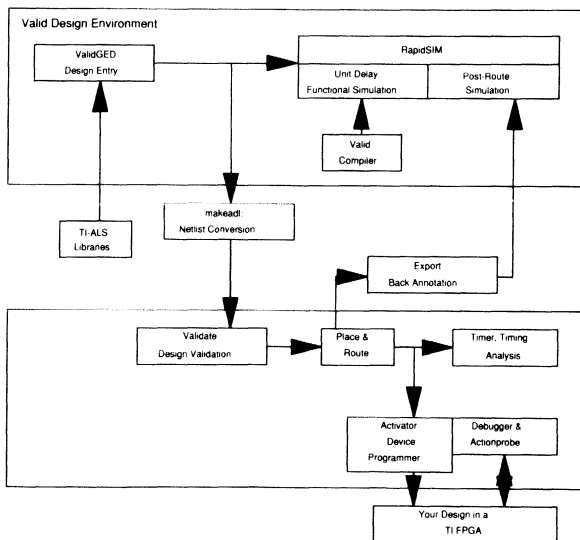


Figure 1. TI-ALS Valid Design Flow

Next, the TI-ALS Validate program is used to check the design for design rule violations, and then the TI-ALS Configure program automatically places and routes the design. The TI-ALS Timer utility can then be used for static timing analysis or a back annotation file may be created for Valid's RapidSIM simulator to run accurate system simulations. After satisfactory timing and simulation analyses, the TI FPGA is programmed using the Activator 1 or Activator 2 programming units.

Note: The Activator 1 programming unit will only interface with a 386DX or 486DX PC and can program one unit at a time. The Activator 2 programming unit will interface with a 386DX or 486DX PC, Sun workstation, and HP-Apollo workstation and can program up to four units at a time.

DIRECTORY STRUCTURE AND SYMBOLIC LINKS

The TI-ALS software searches for executable and design files in the /usr/als directory. If you wish to install the TI-ALS software in a directory other than /usr/als you must create a symbolic link from /usr/als to the directory that contains the TI-ALS software.

For example, if the TI-ALS software is installed in /home1/ti the following symbolic link must be created:

```
In -s /home1/ti/als /usr/als
```

TI-ALS/VALID LIBRARY FILES

Two TI-ALS libraries are used with Valid Logic Systems CAE design software. The libraries contain all the symbol (.body.1.n), schematic (.logic.1.n), and simulation (.sim.1.n) files necessary for schematic capture and simulation of a TI FPGA. The A1000 library is used for TPC10 designs and the A1200 library is used for TPC12 designs. The default location for these libraries is in the /usr/als/lib/a1000 and /usr/als/lib/a1200 directories.

The installation will create a symbolic link from /usr/valid/lib/a1000 to /usr/als/lib/a1000 and from /usr/valid/lib/a1200 to /usr/als/lib/a1200. The following links must be added to the /usr/valid/lib/master.lib file to reflect the current library structure;

```
'a1000' /usr/valid/lib/a1000/a1000.lib'  
'a1200' /usr/valid/lib/a1200/a1200.lib'
```

All files for the design are stored in the same design directory. The Valid software files include the startup.ged, compiler.cmd, and simulate.cmd files, as well as the symbol (.body) and schematic (.logic) files for your design. The ValidCOMPILER produces the cmpexp.dat and cmpsyn.dat files that are used to create the TI-ALS compatible (.adl) netlist file.

VALIDGED SCHEMATIC EDITOR

For each TI-ALS design, modify the startup.ged file so that either the A1000 or the A1200 library is specified for access by ValidGED. Designs cannot use TPC10 and TPC12 macros simultaneously, so only one of the libraries can be accessed in any particular startup.ged file.

Each new design must have a directory where all the TI-ALS and Valid files for the design will reside. Directory and schematic names must begin with a letter and be no longer than eight alphanumeric characters.

Note: The makeadl program will not be successful if the underscore '_' character is used in the design name.

TI-ALS Conventions for Schematic Capture Using ValidGED

These TI-ALS design conventions apply when entering a schematic:

1. The schematic must contain only components from the Valid standard library and the TI-ALS A1000 or A1200 libraries.
2. Net names that are specified by the SIGNAME command must begin with a letter. In addition to alphanumeric characters, the following characters are allowed: @, #, %, ^, &, _ , -, +, =, ~, and ?.
3. You must label all bus signals that originate from MERGE bodies in the Valid standard library with user-defined names. Use the SIGNAME command in GED to define the bus labels. You cannot leave buses from MERGEs unlabelled, because the ValidCOMPILER will create default net names that are incompatible with the TI-ALS system software.
4. You may indicate power and ground signals by attaching either the VCC and GND symbols, or the 0 and 1 signal names to a wire.
5. Only one output can drive a particular net in the design. Thus, wired-AND and wired-OR configurations are not allowed.
6. The SIZE and TIMES properties are not allowed. This ValidGED feature allows a single schematic component to represent multiple components, much like a bus represents multiple nets. The TI-ALS macro library does not permit a single component to represent multiple components.
7. Internal and external signals must pass through an I/O buffer macro prior to going on or off chip. The I/O buffers must be located in the top-level schematic of the hierarchy.
8. The EXOR and BUFF macros replace the XOR and BUF macros, respectively, in the Valid/TI-ALS library. The XOR and BUF macros are functionally identical to EXOR and BUF.

ADDING TITLE AND ABBREVIATION PROPERTIES

Valid and TI-ALS both require title and abbreviation properties. A maximum of eight characters and an abbreviation are added to each schematic. For each schematic in your design, add a drawing body with title and abbreviation properties attached.

MODIFYING TI-ALS SOFT MACROS

The TI-ALS macro libraries contain soft macros. These are higher-level logic functions created from standard hard macros. It is possible to copy the schematics and symbols of TI-ALS soft macros, modify them, and save them as user-defined macros. To do this, use the WRITE or DIAGRAM commands to create a copy of the TI soft macro. Then, you can tailor the function of the new soft macro to your design's requirements. Remember to use the CHANGE command to modify the title and abbreviation properties of the drawing body so that it matches the new macro name.

TOP-LEVEL SYMBOLS

A symbol (.body) drawing of the top-level schematic is not necessary for TI FPGA designs. The top-level symbols are only necessary when an FPGA is used in board-level schematics and simulations.

UNIT-DELAY FUNCTIONAL SIMULATION

After schematic capture, the design is ready for unit-delay simulation. Compile the design using the ValidCOMPILER, then enter the Valid simulator. The delay through each level of logic will be one nanosecond.

ADL NETLIST CONVERSION

To create the TI-ALS compatible (.adl) netlist and other TI-ALS design files, execute the `makeadl <design name>` command in the design directory. The command `makeadl` compiles the design using the ValidCOMPILER then converts the Valid netlist files `cmpexp.dat` and `cmpsyn.dat` into TI-ALS netlist (.adl), pin (.ipf), and criticality (.crt) files. The correct syntax for `makeadl` is:

```
makeadl [fam:<family_name>] <design_name>  
where <family_name> is act1 or act2
```

POST-ROUTE SIMULATION WITH BACK ANNOTATED DELAYS

After Place and Route, verifying the FPGA AC performance requires extracting physical timing delay information from TI-ALS and back annotating wire delays to the Valid simulator. After the design has been placed and routed with the TI-ALS Configure programs, you can create a back annotation file using the EXPORT command in TI-ALS. This creates the `design_name.min` file and the `design_name.max` file. These files contain the back annotated timing information of the design for the Valid simulator. To use one of these files in your simulation, edit the `simulate.cmd` file in your <design name> directory and add the line:

```
wire_delays 'design_name.min' or  
wire_delays 'design_name.max'
```

CONCLUSION

The TI-ALS system used in conjunction with the Valid tool set provides a powerful desktop FPGA design solution. This fully integrated approach offers you the ability to utilize Valid's popular design entry and design verification tools coupled with the TI-ALS system's automatic device configuration and programming features to quickly turn your design concept into programmed silicon.

SECTION 3.4

FPGA DESIGN FLOW WITH VIEWLOGIC ON SUN

Jim Fullerton
Texas Instruments, Dallas
FPGA Applications

INTRODUCTION

The purpose of this paper is to outline the TI Action Logic System, TI-ALS, design flow when using the Viewlogic schematic capture and simulation software packages on a Sun Workstation. Refer to the Viewlogic tutorials and manuals to learn how to use the Viewlogic software.

The general TI-ALS design flow using Viewlogic is illustrated in Figure 1. TI-ALS Viewlogic Design Flow. Viewdraw is used to enter the schematic. The schematic files may be used to create either a functional simulation netlist or a TI-ALS compatible netlist. The simulation netlist is created by selecting Export|Wirelist|Viewsim from the Viewlogic menus. The TI-ALS compatible netlist is created by using the makeadl command.

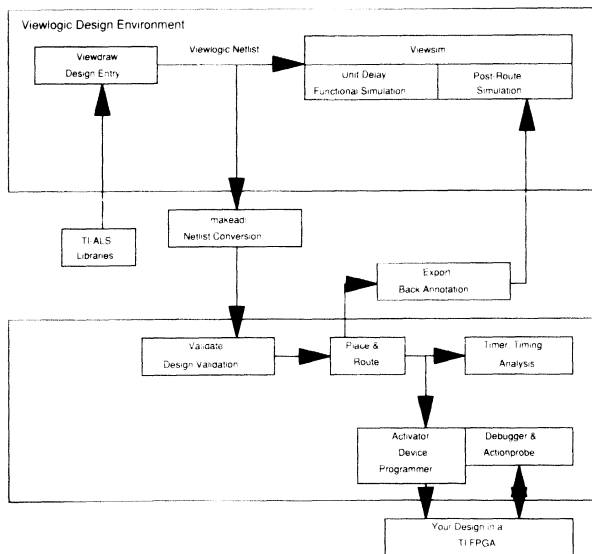


Figure 1. TI-ALS Viewlogic Design Flow

Next, the TI-ALS Validate program is used to check the design for design rule violations, and then the TI-ALS Configure program automatically places and routes the design. The TI-ALS Timer utility can then be used for static timing analysis or a back annotation file may be created for Viewlogic's Viewsim simulator to run accurate system simulations. After satisfactory timing and simulation analyses, the TI FPGA is programmed using the Activator 1 or Activator 2 programming units.

Note: The Activator 1 programming unit will only interface with a 386DX or 486DX PC and can program one unit at a time. The Activator 2 programming unit will interface with a 386DX or 486DX PC, Sun workstation, and HP-Apollo workstation and can program up to four units at a time.

SYMBOLIC LINKS

The TI-ALS software searches for executable and design files in the /usr/als directory. If you wish to install the TI-ALS software in a directory other than /usr/als you must create a symbolic link from /usr/als to the directory that contains the TI-ALS software.

For example, if the TI-ALS software is installed in the /home1/ti the following symbolic link must be created:

```
ln -s /home1/ti/als /usr/als
```

DIRECTORY STRUCTURE

The directory structure when using TI-ALS Viewlogic software is illustrated in Figure 2. TI-ALS Viewlogic Directory Structure. There is one directory for each design. Design_a and design_b are examples of these directories. The design_a directory in Figure 2. illustrates what each of these directories will look like. There are three subdirectories under the design_a directory: schematic (/sch), symbol (/sym), and wire (/wir). For every page of the design, a file resides in both the /sch and /wir subdirectories. Each file will have the name of the schematic with the page number as its extension. Similarly, for each symbol, there will be a file in the /sym subdirectory. When backing up your design you need to save the sch and sym directory files only. The wir files can be regenerated.

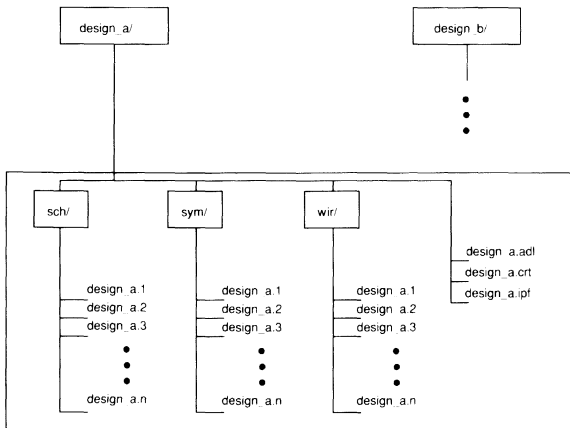


Figure 2. TI-ALS Viewlogic Directory Structure

TI-ALS/VIEWLOGIC LIBRARY FILES

The files for the TI-ALS library components are located in:

```
/usr/als/lib/a1000/cells (for the TPC10 series family)
/usr/als/lib/a1200/cells (for the TPC12 series family)
```

Designs cannot use TPC10 and TPC12 macros simultaneously.

TI-ALS/VIEWLOGIC SIMULATION MODELS

The simulation models represent the function of each TI-ALS macro in terms of Viewlogic simulation primitives. The simulation primitives for the TI-ALS library are located in:

```
/usr/als/lib/a1000/models (for the TPC10 series family)
/usr/als/lib/a1200/models (for the TPC12 series family)
```

SPECIFYING LIBRARIES IN THE VIEWDRAW.INI FILE (WORKVIEW 4.1)

The TI-ALS library directories must be specified at the bottom of the VIEWDRAW.INI file as follows:

For TPC10 Series:

```
DIR [pw].
DIR [rm] /usr/als/lib/a1000/cells
DIR [rm] /usr/als/lib/a1000/models
DIR [rm] /workview_path/builtin
```

For TPC12 Series:

```
DIR [pw].
DIR [rm] /usr/als/lib/a1200/cells
DIR [rm] /usr/als/lib/a1200/models
DIR [rm] /workview_path/builtin
```

Where /workview_path is the pathname of the Workview directory.

UNIX ENVIRONMENT REQUIREMENTS

You must alter the UNIX .cshrc file to add the Workview directory to the search path as well as to set the Workview environment variable as follows:

```
setenv WDIR workview_path/standard
```

Where workview_path is the pathname of the Workview directory.

ADDING POWER AND GROUND

To add power or ground signals in the schematic, use the VCC or GND symbols.

MODIFYING TI-ALS SOFT MACROS

The schematics and symbols of the TI-ALS soft macros can be copied, modified and saved as user-defined macros by using the Viewlogic File|WriteTo command. This command will save the schematic and symbol to another file. The new symbol must have the invisible attribute LEVEL=SOFT removed. Remove the LEVEL=SOFT attribute as follows:

1. Edit the new symbol.
2. Change all of the attributes to VISIBLE
3. Select and delete the text, LEVEL=SOFT
4. Save the edited symbol.

ADDING PINS TO THE DESIGN

Add pins to the design by using the I/O buffer macros, which you should place on the top level schematic. Then add a net segment to the pad of the I/O buffer and label it, as shown in Figure 3. Adding Pins to a Viewlogic Design. The name of the net attached to the pad corresponds to the net label. This net label is used by T1_ALS for I/O assignment. If all of the I/O buffers are not placed on the top-level schematic, you must create a top-level symbol for the design.

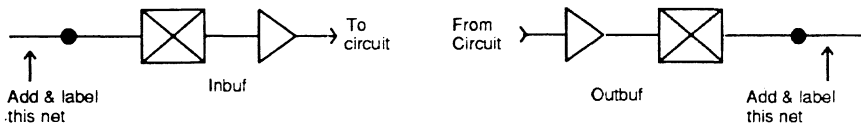


Figure 3. Adding Pins to a Viewlogic Design

TOP-LEVEL SYMBOL

For the I/O buffers on lower levels, you must pull the dangling nets up to the top-level schematic. The pin names of the symbol must match the signal names of the dangling nets on the I/O buffer macros.

Note: Using a top-level symbol may create more work for the designer because both the symbol and the schematic must change if the I/Os change.

HIERARCHY - SHEETS VS. SYMBOLS

A multiple page design is composed of more than one sch file. For a multiple page design, each sheet is treated as part of a top level schematic and is not considered as a hierarchy level. A design using user defined symbols or soft macro symbols is hierarchical.

FUNCTIONAL SIMULATION

After you have entered the schematic, a functional simulation netlist, design_name.vsm, may be created by selecting Export|Wirelist|Viewsim from the Viewlogic menus. The delay through each macro is one nanosecond.

If you are performing a functional simulation following a timing simulation, delete the design_name.var file before performing the functional simulation. This file must be removed because it will modify the intrinsic delay.

CREATING THE TI-ALS NETLIST FROM THE SCHEMATIC

After you complete your schematic, exit the Viewlogic software, and from the UNIX system prompt in the design directory type the following command:

```
makeadl [fam:<family_name>] <design_name>
where <family_name> is act1 or act2
```

The makeadl command will use the wir files for the design to create three files: the .adl netlist file, and templates for the criticality (.crt) and pin (.ipf) files. These files will be located in the design_name directory.

BACK ANNOTATING TIME DELAYS

After the .adl netlist has been created, use the TI-ALS software to place and route the device. After place and route, invoke the export program from the UNIX system prompt by typing, export <design_name>. Export will create a timing file, <design name>.dtb, in both the <design name> directory and the workview directory. Export will also create a <design name>.vsm file that contains post-route delay information for Viewsim.

The back annotated delay file contains min/max delay information for the voltage, temperature, and speed grade which are specified in the TI-ALS Export menu. The VIEWSIM.INI file should be edited to change the delay typ (typical) command to delay max or delay min.

VIEWLOGIC ANOMALIES

When selecting the TRIBUFF macro, be sure to type two fs; otherwise, the Viewlogic TRIBUF macro will be selected.

CONCLUSION

The TI-ALS system used in conjunction with the Viewlogic tool set provides a powerful desktop FPGA design solution. This fully integrated approach offers you the ability to utilize Viewlogic's popular design entry and design verification tools coupled with the TI-ALS system's automatic device configuration and programming features to quickly turn your design concept into programmed silicon.

SECTION FOUR

ACTION LOGIC SYSTEM

SECTION 4.1

ACTION LOGIC SYSTEM (ALS) OVERVIEW

Mohan Maheswaran
Texas Instruments, UK
Technical Marketing

The TI-ALS consists of software and programming hardware that operates on 386 personal computers or Apollo or Sun workstations running popular CAE systems such as Mentor, Valid, Viewlogic or Orcad (refer Table 1.). The ALS enables the process of customising Field Programmable Gate Arrays to be quick and easy without the lengthy waiting periods associated with most routing tools in the industry.

Design Process Overview

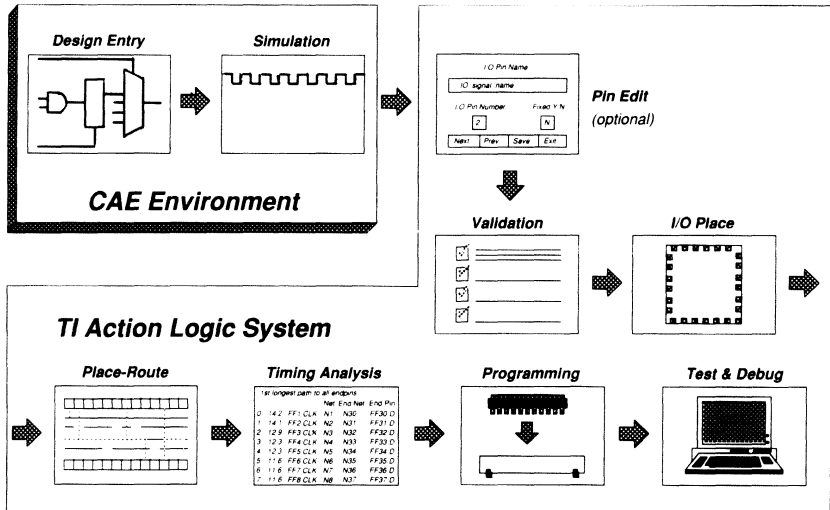


Figure 1. CAE Environment and TI Action Logic System

SCHEMATIC CAPTURE AND SIMULATION

is performed in the host environment utilising the TI TPC series macro library which consists of 583 of the most frequently used logic consolidation macros. Within the host environment both unit delay and post routing simulations can be carried out.

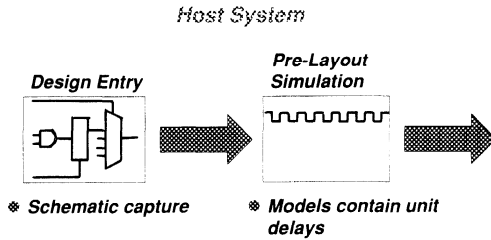


Figure 2. Schematic Capture And Simulation

MANUAL PIN ASSIGNMENT (optional)

allows I/O's to be manually assigned to device package pins. Alternatively they can be automatically assigned.

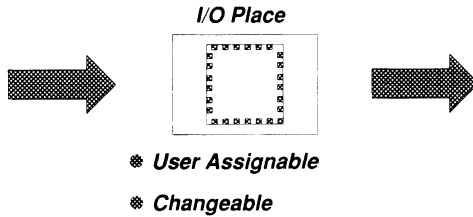


Figure 3. Pin Assignment And I/O Placement

VALIDATION

is an electronic design rule check that assures design parameters are met so errors in the schematic can be quickly corrected before proceeding.

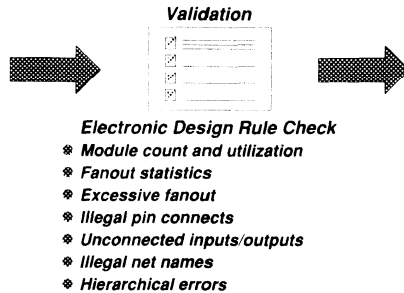
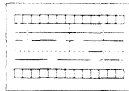


Figure 4. Validation

PLACE AND ROUTE

places the functions and routes interconnects between logic modules and I/O pins. This is accomplished automatically and optimised for your design with special attention paid to critical paths. i.e. Manual placement of components or routing of interconnects is not necessary. Back annotation exports post layout delays back to the host environment for device simulation.

Auto Place-Route



- * **Optimizes macro placement**
- * **Picks shortest interconnect net**
- * **85%-90% array utilization**
- * **100% automatic**
- * **Typically, less than 1 hour runtime**
- * **Calculates segment delays**
- * **Critical path weighting**
- * **Back-annotation of wiring delays**

Figure 5. Place And Route

STATIC TIMING ANALYSIS

displays the timing characteristics of the FPGA design for inspection of all paths. Postlayout net delays are summarised and automatically ported to the timer for specific circuit timing analysis.

Static Timing Analyzer

10 longest paths in all outputs					
Path	Net	Delay	Load	Pin	
1	4.0	FF3 C14	Net	A30	FF3D D
2	4.0	FF3 C14	Net	A21	FF3D D
3	2.8	FF3 C14	Net	A20	FF3D D
4	2.8	FF3 C14	Net	A22	FF3D D
5	2.3	FF4 C14	Net	A23	FF3D D
6	2.3	FF5 C14	Net	A24	FF3D D
7	1.8	FF2 C14	Net	A25	FF3D D
8	1.8	FF3 C14	Net	A26	FF3D D
9	1.8	FF3 C14	Net	A27	FF3D D
10	1.8	FF3 C14	Net	A28	FF3D D

- ❖ **Inspects paths and delays**
- ❖ **Determines path delays**
- ❖ **Optimizes path delays**
- ❖ **Annotates critical paths**
- ❖ **Identifies worst-case path for any pin**
- ❖ **Textual format**
- ❖ **Back-annotation of wiring delays**

Figure 6. Static Timing Analysis

PROGRAMMING

is made easy by using the TI ACTIVATOR. The TI ALS reads the fuse file and the activator hardware applies programming pulses in sequence. The target antifuse is verified and a check on surrounding antifuses assures correct device functionality.

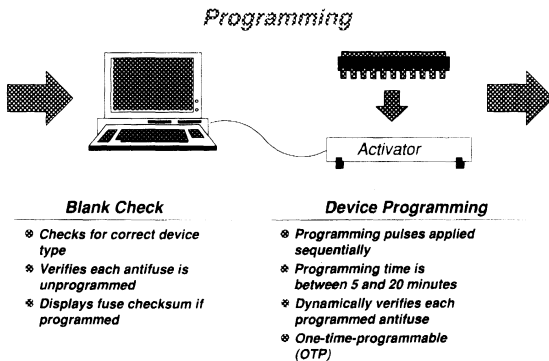


Figure 7. Programming

TEST AND DEBUG

permits verification of a device in the development system environment or target system. 100% observability of all on chip functions is provided without generating test vectors by using two built-in test pins to address multiple nodes simultaneously. These two built-in test pins can later be used as I/O's. In-circuit test and debug are accomplished using an Actionprobe which can address any internal node to determine its logical state. Blowing security antifuses protects against reverse engineering.

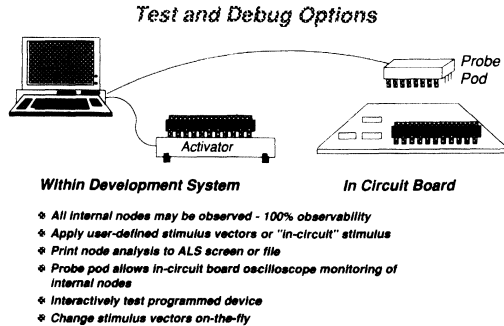


Figure 8. Test And Debug

The complete TI ALS design flow and associated file structure is shown in Figure 9.

ALS DESIGN FLOW

MAKEADL
Output

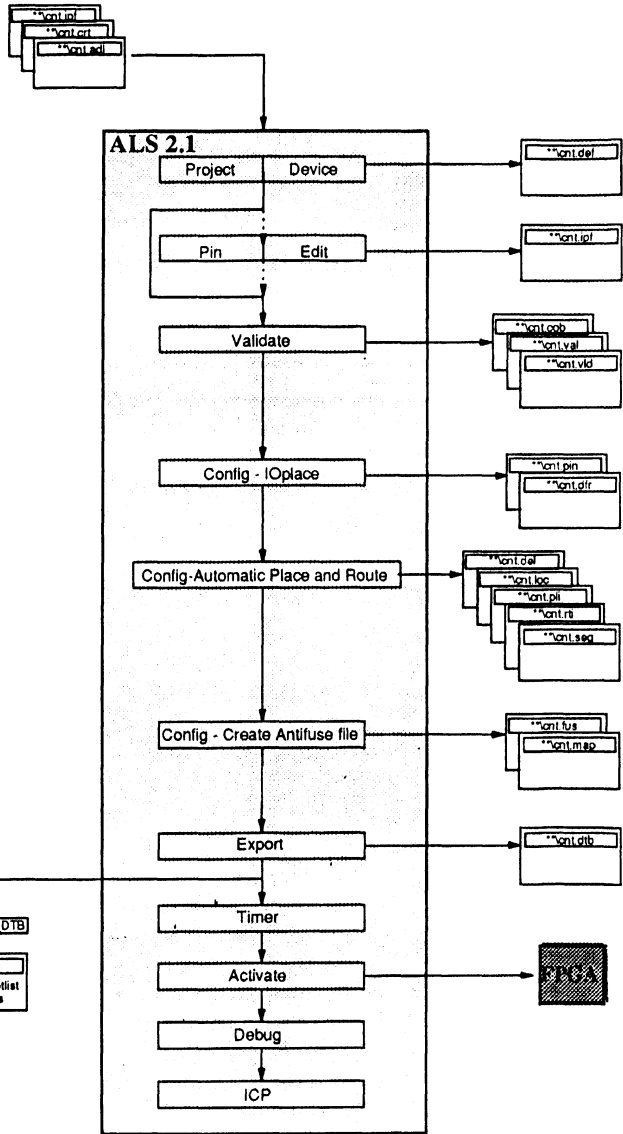


Figure 9. TI ALS Design Flow

In summary and as an introduction to the following technical papers figure 10. summarises some of the key features and benefits of the TI ALS.

TI-ALS Design System

Features	Benefits
<input checked="" type="checkbox"/> Multiple netlist inputs	<input checked="" type="checkbox"/> Uses familiar CAE environment (Viewlogic, OrCAD, Mentor, Valid)
<input checked="" type="checkbox"/> Fast auto place and route	<input checked="" type="checkbox"/> Completes in under 25 minutes for 1200 gates
<input checked="" type="checkbox"/> Static timing analyzer	<input checked="" type="checkbox"/> Summarizes post-layout net delays
<input checked="" type="checkbox"/> Functional test and debug	<input checked="" type="checkbox"/> Allows analysis "in circuit board" or "in programmer box"
<input checked="" type="checkbox"/> 100% internal signal observability	<input checked="" type="checkbox"/> Simplifies in-system analysis
<input checked="" type="checkbox"/> Back-annotation to host environment	<input checked="" type="checkbox"/> Provides accurate post-layout simulation
<input checked="" type="checkbox"/> User can control pin placement	<input checked="" type="checkbox"/> FPGA pin placement can be driven by board level constraints rather than board layout being controlled by FPGA constrictions




Figure 10. TI-ALS features and Benefits

TI ACTION LOGIC SYSTEM CONFIGURATIONS						
HARDWARE PLATFORM	LIBRARY/CAE HOST ENVIRONMENT	TI SYSTEM PART NUMBER	SOFTWARE SUPPLIED†		HARDWARE SUPPLIED‡	
			CAPTURE	LIBRARY	PROGRAM	TEST/DEBUG
PC 386	VIEWLOGIC/ ORCAD	TPC-ALS-110*	VIEWDRAW	VIEWLOGIC/ ORCAD	ACT 1	YES
	VIEWLOGIC/ ORCAD	TPC-ALS-113*	VIEWDRAW	VIEWLOGIC/ ORCAD	ACT 1	-
	VIEWLOGIC/ ORCAD	TPC-ALS-115*	VIEWDRAW	VIEWLOGIC/ ORCAD	-	-
	VIEWLOGIC/ ORCAD	TPC-ALS-210	VIEWDRAW	VIEWLOGIC/ ORCAD	ACT 2	YES
	VIEWLOGIC/ ORCAD	TPC-ALS-213	VIEWDRAW	VIEWLOGIC/ ORCAD	ACT 2	-
	VIEWLOGIC/ ORCAD	TPC-ALS-215	VIEWDRAW	VIEWLOGIC/ ORCAD	-	-
HP/APOLLO (Supports AEGIS™ operating system)	MENTOR	TPC-ALS-035*	-	MENTOR	-	-
	MENTOR	TPC-ALS-230	-	MENTOR	ACT 2	YES
	MENTOR	TPC-ALS-233	-	MENTOR	ACT 2	-
	MENTOR	TPC-ALS-235	-	MENTOR	-	-
SUN4/SPARC station 2™	VALID	TPC-ALS-045*	-	VALID	-	-
	VALID	TPC-ALS-240	-	VALID	ACT 2	YES
	VALID	TPC-ALS-243	-	VALID	ACT 2	-
	VALID	TPC-ALS-245	-	VALID	-	-
SUN4/SPARC station 2	VIEWLOGIC	TPC-ALS-055*	-	VIEWLOGIC	-	-
	VIEWLOGIC	TPC-ALS-250	-	VIEWLOGIC	ACT 2	YES
	VIEWLOGIC	TPC-ALS-253	-	VIEWLOGIC	ACT 2	-
	VIEWLOGIC	TPC-ALS-255	-	VIEWLOGIC	-	-

†Software upgrades to support the 12-Series are available for each CAE environment.
‡Hardware may be purchased as a stand-alone item for each platform. *Presently supports the TPC10 Series only.

Table 1.

SECTION 4.2

CONFIGURING WORKVIEW FOR FPGA LIBRARIES

Glenn Bigger
Texas Instruments, Dallas
FPGA Applications

Viewdraw Initialization File

A Viewdraw initialization file, `viewdraw.ini`, is located in the directory `\workview\standard`. To configure your Workview environment for TPC10 series FPGA designs, the `viewdraw.ini` file should contain the following lines at the end of the file.

```
dir 11 \als\lib\1000\cells  
  
dir 12 \als\lib\1000\models  
  
dir 13 \workview\builtin
```

To configure your Workview environment for TPC12 series FPGA designs, the `viewdraw.ini` file should contain the following lines at the end of the file.

```
dir 11 \als\lib\1200\cells  
  
dir 12 \als\lib\1200\models  
  
dir 13 \workview\builtin
```

A `viewdraw.ini` file is supplied with both the TPC10 Series Library and the TPC12 Series Library. The `viewdraw.ini` file supplied with the TPC10 Series Library is located in the directory `\als\lib\1000` and the `viewdraw.ini` file supplied with the TPC12 Series Library is located in the directory `\als\lib\1200`. To configure your Workview environment for TPC10 series FPGA designs, the `viewdraw.ini` file located in the directory `\workview\standard` should be replaced with the `viewdraw.ini` file located in the directory `\als\lib\1000`. To configure your Workview environment for TPC12 series FPGA designs, the `viewdraw.ini` file located in the directory `\workview\standard` should be replaced with the `viewdraw.ini` file located in the directory `\als\lib\1200`.

DEFAULT DESIGN DIRECTORY

A design database is required for files created and used by the TI-ALS for your designs. The default design directory is `\designs`. For a design named Counter, the TI-ALS files created for the design will be stored in the directory `\designs\counter`. The TI-ALS documentation assumes that you are using the default database. In configuring your system according to the Installation Notes, your system will be configured for the default database. However, if you use the Viewfile option when designing with the Viewlogic Workview software, Viewfile will create a different directory structure that will not be compatible with the default system configuration.

COMPONENT SELECTION

TI-ALS supports dialog box library selection which is standard with Workview 4.0 software. The Viewdraw.ini file supplied with both the TPC10 Series Library and the TPC12 Series Library has the default setting for the dialog box as OFF. If you want to use the dialog box to select components when using Viewdraw, you can change the default setting by editing the Viewdraw.ini file and changing the DBOXON command from DBOXON 0 to DBOXON 1. The dialog box can also be enabled/disabled after invoking the Workview software. To enable/disable the dialog box:

- 1) Invoke the Workview software and execute the commands Window Open Viewdraw Schematic to open an active window.
- 2) Execute the commands Change Display Params. The Viewdraw Parameters selections will appear in the upper right corner of the display.
- 3) Move the cursor to the DBOXON ON/OFF box. Press the left mouse button then use the backspace to delete the current selection. Type either ON or OFF and then press the left mouse button. Next move the cursor to the SAVEINI box and then press the left mouse button to save the changes to the Viewdraw.ini file in the \workview\standard directory.

If the dialog box is enabled, components can be added to your schematic using the following steps:

- 1) Execute the commands Add Comp and then press the middle mouse button. The dialog box will appear in the middle of the display.
- 2) Select DIR to display the directory selections. Select the library directory. For the TPC10 Series Library, the directory is \als\lib\1000\cells and for the TPC12 Series Library, the directory is \als\lib\1200\cells. The FPGA component dialog box will be displayed with the different components. Select the desired component.

SECTION 4.3

CUSTOMIZING VIEWLOGIC MENUS FOR TI ACTION LOGIC SYSTEM

Dung Tu,
Texas Instruments, Germany
Technical Marketing

INTRODUCTION

The design environment for Texas Instruments Field Programmable Gate Arrays (FPGA) on a PC-386 consists of two parts: the TI Action Logic System (TI-ALS) and the Viewlogic Workview. TI-ALS is used for FPGA specific tasks like Validate, Place & Route, static timing analysis, etc. Viewlogic Workview is a set of CAE tools for schematic entry and simulation.

ALS and Workview work normally as two independent software packages with different user interfaces. This application note shows how to use the open Workview menu structure to integrate ALS functions or any other MSDOS command into the Workview menu panels so that these functions can be selected directly from the menu.

WORKVIEW MENU STRUCTURE

The structure of Workview menus is specified in menu source files which are stored under the subdirectory \WORKVIEW\STANDARD. You can edit these ASCII files to customize the menus for a specific application environment. There is a set of menu source files for each graphic adapter. The files with the extension MN1 are for the EGA adapter, MN5 files are for CGA and MN8 files are for VGA.

There are menu source files for Utils, Viewdraw, Viewfile, Viewsim, Viewsys, Viewtext, Viewwave and WView. When Workview is invoked the first time after the installation, Workview compiles the menu source files into a binary file MENUS.M32 which is used later on to configure the menus every time Workview is invoked. Therefore, if the menu source files are modified then the MENUS.M32 file must be deleted or renamed (as backup) so that it can be recompiled, otherwise the modifications will not be effective.

CUSTOMIZING WORKVIEW MENUS FOR ACTION LOGIC SYSTEM

Each Workview menu source file can consist of several submenus. A submenu defines a menu level and contains source code for the screen coordinates, menu text and command syntax. Assuming you want to add the following functions into a submenu of the original Workview menu panel:

VLITPC10	Make ADL netlist for TPC10 series from WIR netlist
VLITPC12	Make ADL netlist for TPC12 series from WIR netlist
ALS	Invoke ALS from within Workview
ALSRUN	Invoke a batch file to run a complete ALS flow including back-annotation but without fuse generation
VSM_DTB	Back-annotate the delays to Viewsim
NC	Invoke Norton Commander

These functions can be added basically to any submenu. A good place is for example under the Export submenu.

Figure 1 shows the Main menu and the Export submenu before and after the modifications:

Main menu	Submenu	New submenu
Export	Export	
Window	Wirelist	Wirelist
View	Check	Check
Add		VLITPC10 ---
Leve		VLITPC12
Buffer		ALS New functions
File		ALSRUN
Change		VSM_DTB
Export <---		NC ---
Xform		
Select		
Delete		
Move		
Copy		
Plot		
Undo		
Info		
Bye		

Figure 1. Workview Main menu and Export submenu

Because both Export Wirelist and Export Check in the original Export submenu do something with a schematic, the Export submenu is specified in the Viewdraw menu source file which is the VIEWDRAW.MN8 for a VGA graphic adapter. Using a text editor to look at the VIEWDRAW.MN8 file you can find the specification for the Export submenu as shown in Figure 2.

```

.
.
q_export
80 22 {
Wirelist &q_wirelist
Check check
}
.
.

```

Figure 2. Export submenu in the original VIEWDRAW.MN8 file

Figure 3. shows how to modify the Export submenu to incorporate the new functions.

```
.  
.
q_export
80 22 {
Wirelist &q_wirelist
Check check
VLITPC10 "system vlitpc10 $project"
VLITPC12 "system vlitpc12 $project"
ALS "system als $project"
ALSRUN "system alsrun1 $project"
VSM_DTB "system export1 $project"
NORTON "system nc"
}
.
```

Figure 3. Modified Export submenu in VIEWDRAW.MN8 file

Looking at the VLITPC10 line as an example, the syntax is as following:

VLITPC10: The name of the function as it will appear on the new menu
system : Workview DOS interface call
vlitpc10: The name of the batch file to make an ADL netlist for TPC10 series
\$project: The project name which will be passed as parameter to thebatch file
vlitpc10.bat

In the ALS version 2.1 which supports both the TPC10 and TPC12 series, the vlitpc10.bat file looks as follows:

```
makeadl fam:act1 %1
```

fam:act1 specifies the TPC10 series as device family

%1 will be replaced by the value of \$project, which is the name of the current project.

If you do the following steps, which are recommended for creating a new project, then the correct project name will be passed automatically to \$project:

- Use 'Window, Open, Viewfile, Project, Create' to create a new project
- Use 'Set, Project' to make the new project to the current project
- Use 'Window, Open, Viewdraw' and enter a schematic name.

It is mandatory to use the same name for the top level schematic as for the project name. That means if you have used Viewfile to create a project TOP, then your top level schematic name must be TOP.

Following are the batch files for the other menu items. All batch files are stored under \ALS\BIN.

Menu Item	Batch File
VLITPC12	<pre>vlitpc12.bat: makeadl fam:act2 %1</pre>
ALS	original als.bat file which is included in TI-ALS package
VSM_DTB	<pre>export1.bat: export tempr:com voltr:com speed_grade:std %1 Back-annotate the delays to Viewsim for commercial temperature range, commercial voltage range and standard speed grade</pre>
ALSRUN	<pre>alsrun1.bat A modified version of the original ALSRUN which does not generate the fuse map but include the back-annotation of the delays to Viewsim @echo off if x%1x == xx goto usage validate %1 ioplace %1 place %1 route %1 xtract %1 export1 %1 goto exit :usage echo Please specify the design name. echo alsrun design_name :exit</pre>

After modifying the Viewdraw menu source file and adding the batch files, the last step is to rename the file 'MENU8.M32' to a backup file. When you restart Workview you can see the message that Workview recompiles the menu files and when you click 'Export' you see the new menu items.

If you open a schematic window, enter a schematic name and click 'Export, VLITPC10' then the 'makeadl' function is invoked and an ADL netlist for the schematic is generated, using TPC10 as device family. When this process is completed and you press a key you will be back in Workview environment.

ALSRUN submenu is useful if a schematic is modified. After changing the schematic and generating a new ADL netlist, clicking ALSRUN will invoke the complete ALS flow, and back-annotate the delays to Viewsim without generating the fuse file.

SUMMARY

Customizing Workview menus to incorporate ALS functions or any useful MSDOS utility releases you from entering batch commands with long input parameters and helps to improve your productivity.



SECTION 4.4

UNDERSTANDING THE ADL NETLIST

Joel S. Lason
Texas Instruments, Dallas
FPGA Applications

INTRODUCTION

The ADL Netlist

The ADL netlist is the format by which information is passed from a host CAE system to the TI Action Logic System (TI-ALS). Its text consists entirely of readable ASCII characters. It contains information that describes a design's connectivity, I/O and components.

Although it is not necessary to understand the ADL netlist to perform FPGA design, it can be a useful tool for debugging or documenting a design. Sometimes it is easier to find circuit information or a design flaw when it is expressed textually instead of graphically. With this point in mind, a simple design example and its corresponding ADL netlist have been prepared to illustrate some features of and provide familiarity with the ADL file.

ADL Netlist Generation

The ADL file is created from the host CAE database by the MAKEADL program. The correct syntax for invoking MAKEADL is:

```
makeadl [fam:<family_name>] <design_name>
```

where <family_name> is act1 or act2.

ADL Netlist Location

The output is then placed in the design directory and the full pathname to the file is:

```
C:\designs\<>design_name>\<design_name>.adl.
```

EXAMPLE DESIGN AND NETLIST

Example Design Description

A simple design that will be analyzed is shown in Figure 1.- AP1 Top Level Schematic and Figure 2.- NANBLK Sub-level Schematic. Figure 1, contains the top level schematic of this hierarchical example. Two blocks are used to represent more detailed circuitry at a lower level. In addition, there is an AND gate that is a primitive and therefore no lower level drawing exists for this symbol.

AP1 Top Level Schematic

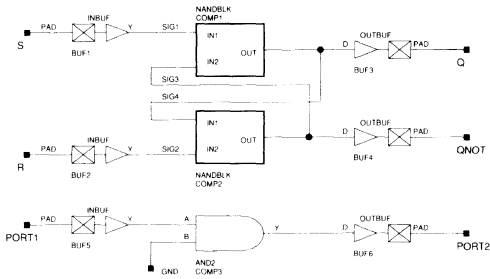


Figure 1. AP1 Top Level Schematic

Figure 2. - NANDBLK Sub-Level Schematic shows that the only logic underneath the NANDBLK SYMBOL is a NAND gate. This was done to illustrate the structure of the ADL netlist for a simple hierarchical design.

NANDBLK Sub-Level Schematic

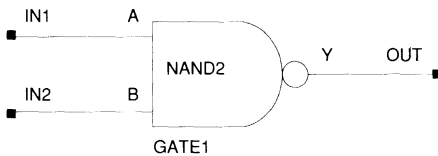


Figure 2. NANDBLK Sub-Level Schematic

Example ADL Netlist Listing

The netlist for this particular design is shown here and will now be examined. Line numbers have been added to aid in the discussion of the file. They do not exist in the actual file.

```

1 ;HEADER
2 ;FILEID ADL /designs/AP1/AP1.adl c6e91786
3 ;CHECKSUM c6e91786
4 ;PROGRAM adl2adl
5 ;VERSION 2.1
6 ;VAR DEFSYS /als/data/system.def
7 ;VAR DEFUSR /alsuser/jdoe/jdoe.def
8 ;VAR DEFDES /designs/DEMO21/DEMO21.def
9 ;VAR DEFDES /designs/AP1/AP1.def

```

```

10 ; VAR design AP1
11 ; VAR FAM ACT1
12 ; VAR ALLFAMS ACT1 ACT2
13 ; VAR ADLIB /als/data/a1000/adl04.lib
14 ; VAR DESDIR /designs/AP1
15 ; VAR AAL /designs/AP1/AP1.aal
16 ; VAR ADL /designs/AP1/AP1.adl
17 ; VAR DELETEINSTPROPS LEVEL
18 ; VAR MSGINDEX /als/data/als21.idx
19 ; VAR MSGTEXT /als/data/als21.txt
20 ; VAR PERMITBADNAMES <NOT-SET>
21 ; ENDHEADER
22 DEF NANDBLK; IN1, IN2, OUT.
23 USE ADLIB:NAND2; GATE1.
24 NET IN1; IN1, GATE1:A.
25 NET IN2; IN2, GATE1:B.
26 NET OUT; OUT, GATE1:Y.
27 END.
28
29 DEF AP1; S, R, Q, QNOT, PORT1, PORT2.
30 USE ADLIB:OUTBUF; BUF6.
31 USE ADLIB:INBUF; BUF5.
32 USE ADLIB:INBUF; BUF1.
33 USE ADLIB:INBUF; BUF2.
34 USE ADLIB:AND2; COMP3.
35 USE NANDBLK; COMP1.
36 USE NANDBLK; COMP2.
37 USE ADLIB:OUTBUF; BUF4.
38 USE ADLIB:OUTBUF; BUF3.
39 NET S; S, BUF1:PAD.
40 NET R; R, BUF2:PAD.
41 NET Q; Q, BUF3:PAD.
42 NET QNOT; QNOT, BUF4:PAD.
43 NET PORT1; PORT1, BUF5:PAD.
44 NET PORT2; PORT2, BUF6:PAD.
45 NET $1N21; COMP3:A, BUF5:Y.
46 NET $1N27; BUF6:D, COMP3:Y.
47 NET GND; COMP3:B; GLOBAL, POWER:GND.
48 NET SIG1; COMP1:IN1, BUF1:Y.
49 NET SIG2; COMP2:IN2, BUF2:Y.
50 NET SIG3; BUF4:D, COMP1:IN2, COMP2:OUT.
51 NET SIG4; BUF3:D, COMP1:OUT, COMP2:IN1.
52 END.

```

Because the design is all ASCII it is simple to read and understand. Just by looking at this file you can find several obvious types of lines which begin with DEF, USE, NET, END and the symbol ";".

Comment Lines

Lines #1-21 begin with a semicolon and all of these lines are comment lines. They contain information about the file and the conditions under which it is generated.

DEF Lines

The DEF lines define a functional block of the design. The first DEF line is line #22. This line begins the definition of the block that represents the NAND gates. The next DEF line is line #29, which defines the top level of the schematic, AP1. For simplicity these are all the blocks defined in this example, yet any number of blocks could be used.

DEF lines also call out all the I/O signals for a given functional block of the design. Line #29 shows that the I/O signals defined for the AP1 block are S, R, Q, QNOT, PORT1, and PORT2. These signals connect to various signal nets in the design.

USE Lines

Line #23 is the first example of a USE line, which tells the T1-ALS which functional blocks are used in this section. On line #23 the USE line is specifying that a NAND2 function from the standard TPC library is used and that its instance name is GATE1. Other USE lines in the design call in I/O buffers and AND2 standard library functions.

Hierarchy is illustrated by the USE lines 35 and 36. These lines specify that the circuit defined above, NANDBLK, is called into the AP1 DEF block. This differs from the other USE lines which call standard library components. The two uses of the NANDBLK definition are assigned to circuit instances COMP1 and COMP2.

NET Lines

NET lines define connections between components. Line #48 shows that a signal named SIG1 is connected to the circuit instance COMP1 and pin IN1 on COMP1. SIG1 is also connected to instance BUF1 and its output pin Y.

Line #45 shows what happens if all nets are not explicitly labeled. The system creates a default name for the NET and in this case the NET name is \$1N21. This is harder to use for debug because it contains less intuitive information.

A special type of signal is illustrated in line #47. This signal is a global signal as shown by the GLOBAL modifier on the POWER:GND pin definition.

END Lines

The last line to discuss is the END line. Its simple function is to terminate functional block definitions called out by the DEF lines.

CONCLUSION

The ADL netlist is easy to use and provides quick design insight. Checking the netlist is quicker in some instances than invoking a graphical schematic capture package, and it presents the design in a different perspective. Sometimes a different view is exactly what is needed to eliminate confusion. Because of its simplicity and utility it is a worthwhile file to understand.

SECTION 4.5

BACK ANNOTATED SIMULATION FOR FPGA'S

Mohan Maheswaran
Texas Instruments, UK
Technical Marketing

INTRODUCTION

In this paper we will look at how the designer can use the Action Logic System in conjunction with the Viewlogic environment to produce actual timing delays associated with his fpga design and can subsequently carry out a post routing simulation. The process of translating the data generated from the ALS environment to the CAE design environment is referred to as 'Back Annotation' (i.e. postlayout delay information is backannotated to the CAE simulator). In this paper we will use an 8 bit full adder as a design example (figure 1.) to describe the process of back annotating. The ability to back annotate is a very powerful feature of the ALS which when used with a good simulator can be used to verify both functionality and timing after routing the design.

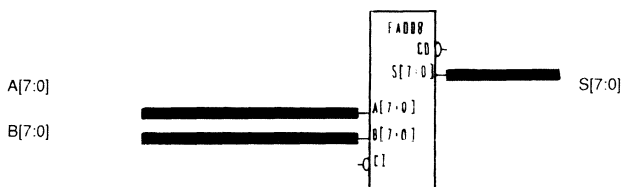


Figure 1. 8 Bit Full Adder using soft macro - FADD8

PRELAYOUT DELAY ESTIMATIONS

The first step in the design process is to carry out a 'Functional' or 'Unit Delay' simulation which will indicate whether the design is behaving as expected. Once this is established then one can begin to take a closer look at the timing of the circuit.

Essentially there are two ways, prior to routing the circuit, that the timings associated with a design, can be estimated. Firstly, a prelayout delay estimation can be made using the statistically estimated delays in the datasheet and secondly, a prelayout delay simulation can be carried out using 'scaling factors' in the Viewlogic environment. The combination of both should give the designer a good idea of whether the fpga is going to meet his design requirements and any areas that may need more careful attention before the design is routed. This is an important part of the design process as timings can be influenced significantly by the sensible assignment of pins and criticality. (Refer Paper 4.7 - Critical Path Analysis For Field Programmable Gate Arrays).

UNIT DELAY SIMULATION

The delay of any path is accumulated from a combination of a combinatorial block, a sequential block, an input and output delay and routing delays. For the calculation of the delays through any combinatorial or sequential block the number of logic levels that make up the block must first be determined. This information can be obtained from the fpga datasheet. For example the 8 bit full adder has 2 logic levels. For the purposes of functional simulation the simulator assigns one unit delay for each level of logic in the design. Thus an 8 bit add using the soft macro FADD8 would have a delay of two units (Figure 2.).

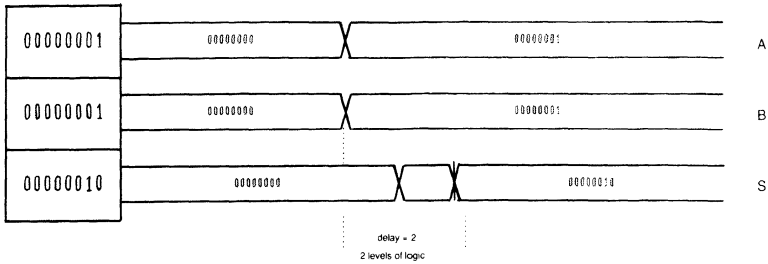


Figure 2. Unit Delay Simulation 00000001 + 00000001

POST ROUTING SIMULATION

After completing the routing of the design and exporting the delays from the als environment to the simulation environment it is now possible to carry out a post routing simulation with the back annotated delays.

The input stimulus used for simulation of the 8 bit full adder is shown in figure 3. i.e. The task is to add 00000001 and 00000001 together as fast as possible and then 00101010 and 00101010 as fast as possible.

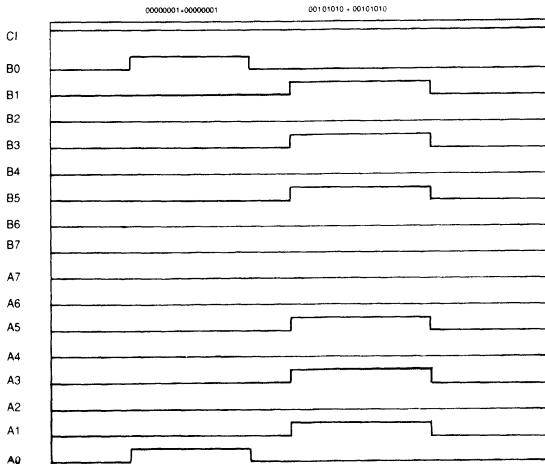


Figure 3. Input Stimuli To Full Adder

The result for the first 8 bit addition is shown in figure 4. The 8 bit addition takes just 16.6ns representing an adding operating frequency of above 60Mhz. This is the typical performance of many complex functions using the TI fpga's. The second addition occurred in a comparable time.

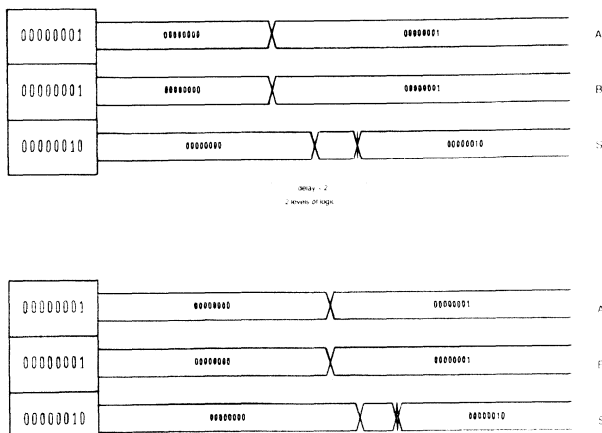


Figure 4. Post Routing Simulation 00000001 + 00000001

Having carried out a post routing simulation to prove the functionality of the design and assess the timing performance it is now good practice to try and simulate at best and worst case delays. This is possible using Viewsim's ability to adjust all the delays by some defined factor. The scaling factors that would give best and worst case timings of the circuit can be found in the TI fpga datasheet.

Debugging timing problems within a circuit can be a problem if the development tools do not facilitate this process in any way. Fortunately Viewsim has a very useful feature which allows logic levels following simulation to be seen on a schematic. Thus the simulator can be set to break on a count or an event and the resulting logic levels at the break point can be seen on the schematic (Figure 5.). Using this very useful feature it is possible to carry out a debug of the circuit without programming any devices. This is made even more useful because of the ability to 'push' down through the schematic hierarchy and see the logic levels at the logic module level. Thus, back annotation and post routing simulation are a key part of the FPGA design flow.

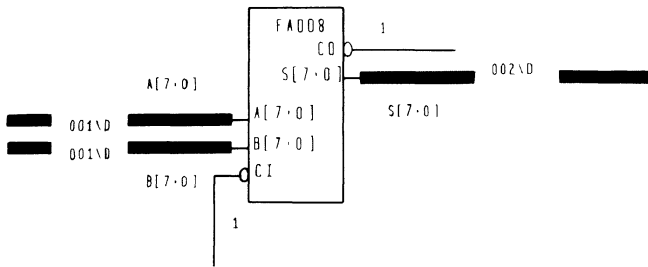


Figure 5. Post Routing Simulation Results Shown On Schematic

CONCLUSION

In summary, delays can be back annotated from the ALS environment to the Viewlogic environment and a Post Routing simulation carried out. The complete fpga system offers the designer the ability to carry out timing simulations as well as purely functional simulations. This capability offers the designer some comfort because it means he can use a design methodology that is sound and get the design right before attempting to programme a device. This is traditionally how asic designs have been carried out for several years and indeed this is the way old designs will be executed in the future.

SECTION 4.6

USING THE ALS TIMER FOR STATIC TIMING ANALYSIS

Mohan Maheswaran
Texas Instruments, UK
Technical Marketing

INTRODUCTION

The ALS TIMER is an interactive static timing analysis tool used to analyze path delays. In contrast to dynamic analysis, static analysis does not require vectors to display the timing of the design. Static timing analysis is useful for determining;

- Internal setup and hold timing checks
- Maximum operating frequency
- Input to Output delays
- Clock skew
- External setup and hold requirements

The following paper describes how the TIMER can be used to analyse the timing associated with a design. The design selected for this purpose is shown in figure 1. which will allow us to look at the timings associated with both combinatorial and sequential logic.

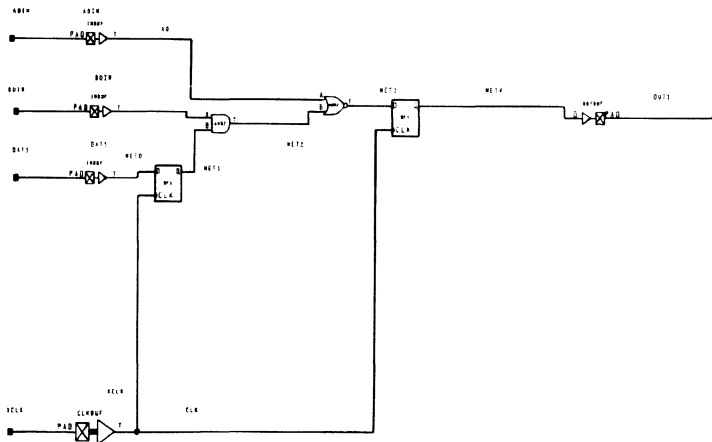


Figure 1.

ENTERING THE TIMER AND SAVING TIMER RESULTS

Prior to using the timer the design must pass through Validate and Config utilities in the ALS system. The TIMER reads the output files from the Validate and Config programs and these files are the TIMER inputs. On entering the timer the process and environment conditions must be set. The options allow the user to define worst case or best case process conditions and/or voltage/temperature conditions. The range of temperatures and voltages for commercial, industrial and military parts as represented in the timer is shown in the datasheet.

After ALS is exited, the file called /alsuser/user_name/user_name.log is written. This file contains all of the results shown on the screen during the use of the timer and must be copied to another file if the results are not to be overwritten on the next occasion the ALS is exited.

PRELAYOUT vs POSTLAYOUT

On entering the timer the user will be given the choice of Prelayout or Postlayout timing. If prelayout timing is selected then statistical estimates are made for the interconnect delays. Postlayout delays can only be seen after placing and routing the design and is generally recommended for analysis. Prelayout mode can be useful for determining and assigning net criticality.

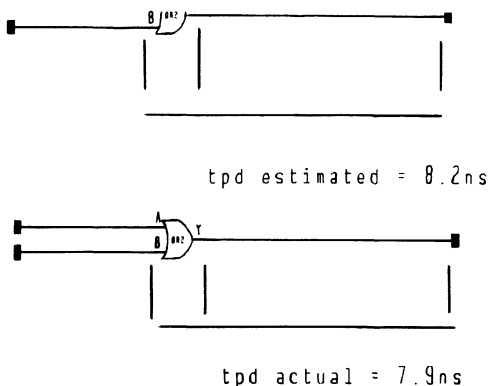


Figure 2. Prelayout vs Postlayout delays

HOW THE TIMER WORKS

The ALS timer operates by determining path delays between specified start points and end points of the circuit. The group of start points is named the 'startset' and the group of end points is named the 'endset'. The timer has default sets of starting and end points. Two of the default sets are called 'CLOCK' and 'GATED' which defines all the clock inputs as a set and all the gated outputs as a set. Thus we can, using the timer, calculate the worst case register to register delay (Figure 3.). In addition to setting the start and end points of any path the timer will rank the paths in order of delay magnitude and list them according to whether the user requests the longest or shortest delays.

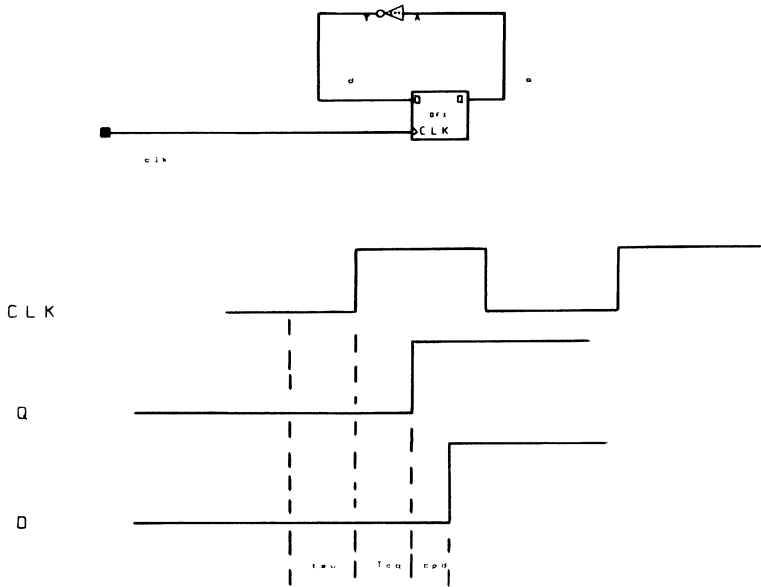


Figure 3. Register To Register Delay

In practice the most commonly used 'sets' will be the 'startset' and 'endset' but there are also other sets available such as 'breakset, tempset and stopset'. These sets are used for defining points along an asynchronous path, which needs to be broken for the timer to analyse it. An asynchronous loop is a combinatorial loop that is not broken by a flip flop stage. If the design contains asynchronous loops, the timer cannot report the delay information for the loop path.(Figure 4.).

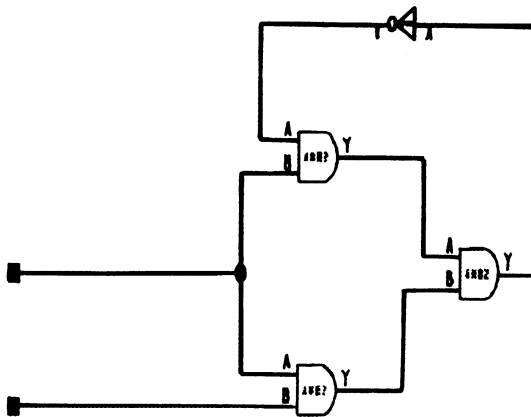


Figure 4. Asynchronous Loop

CREATING AND MODIFYING SETS

When the timer is initialized, four predefined or default sets are created. They are defined as follows (Figure 5.);

- CLOCK - All sequential macro clock inputs
- GATED - All sequential macro gated inputs
- INPAD - All input buffer pads
- OUTPAD - All output buffer pads

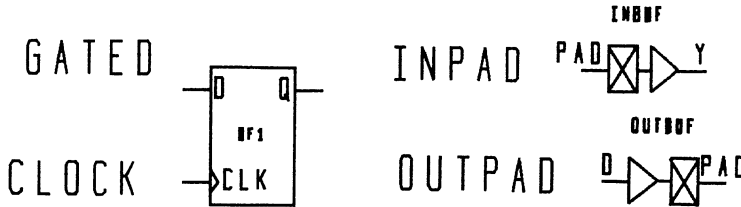


Figure 5.

The 'SET STATUS' command shows the current working sets. i.e. For our design example the current working sets are;

- ; Working startset 'CLOCK' contains 4 pins
- ; Working endset 'GATED' contains 2 pins
- ; Working stopset 'STOP' contains 0 pins
- ; Working passset 'PASS' contains 0 pins

This basically means that the startpoints are all the CLOCK pins (a pin is an input/output to a hard macro) and all the endpoints are GATED pins, which in this case are the D inputs to the flip flops. To see the actual contents of each working set the 'SET SHOWSET' command can be used as shown below;

Note: Some macro pins represent more than one load. All flip flops have some pins with two loads.

SHOWSET - Determines the content of a set. e.g. showset clock

e.g. SHOWSET CLOCK

- | ; Pin name | Net name | Macro |
|------------|----------|-------|
| ; FF2:CLK | CLK | DF1_1 |
| ; FF2:CLK | CLK | DF1_0 |
| ; FF1:CLK | CLK | DF1_1 |
| ; FF1:CLK | CLK | DF1_0 |
| ; 4 pins | | |

e.g. SHOWSET GATED

- | ; Pin name | Net name | Macro |
|------------|----------|-------|
| ; FF1:D | NET3 | DF1_0 |
| ; FF2:D | NET0 | DF1_0 |
| ; 2 pins | | |

e.g. SHOWSET INPAD

: Pin name	Net name	Macro
: DAT1:PAD	DAT1	INBUF
: A0IN:PAD	A0IN	INBUF
: B0IN:PAD	B0IN	INBUF
: XCLK:PAD	XCLK	CLKBUF_0
: 4 pins		

e.g. SHOWSET OUTPAD

: Pin name	Net name	Macro
: \$115:PAD	OUT1	OUTBUF
: 1 pins		

CHANGING SETS

The timer analysis applies to the currently defined working set. To change the current starting set or ending set simply use the commands;

```
> STARTSET SET_NAME
> ENDSET SET_NAME
```

e.g.

```
STARTSET CLOCK - sets starting point to clock
ENDSET GATED   - sets ending point to gate
```

```
STARTSET INPAD - sets starting point to input pad
ENDSET OUTPAD  - sets ending point to output pad
```

where set_name is the name of either a previously defined set or a new set. After you have defined a starting and ending set you can add or delete starting points from the set using the following commands;

```
> ADDSTART INSTANCE_NAME:PIN
> REMSTART INSTANCE_NAME:PIN
> ADDEND INSTANCE_NAME:PIN
> REMEND INSTANCE_NAME:PIN
```

MAXIMUM OPERATING FREQUENCY

In a synchronous design, the maximum operating frequency for that design is defined as the inverse of the longest register to register delay. The register to register delay is the delay from the clock of a flip flop to the data input of a flip flop, including all combinatorial gate delays between the registers and the required setup time on the ending register.

In the design example there are two flip flops. If we use the default sets of 'startset - clock' and 'endset - gated' then we see the longest delay is as follows;

: 1st longest path to all endpins					
: Rank	Total	Start pin	First Net	End Net	End pin
: 0	31.8	FF1:CLK	NET1	NET3	FF2:D

Using the expand utility of the timer we can expand the path 'ranked 0' to obtain the following breakdown showing all the delays associated with the path from the clock input of flip flop 1 to the D input of flip flop 2. (Figure 1.)

```

; 1st longest path to $FF2:D (falling) (Rank: 0)
; Total      Delay      Typ      Load      Macro      Start pin      Net name
; 31.8       6.7       Tsu       0         DF1        $FF2:D         NET3
; 25.1       6.9       Tpd       1         NOR2       $114:B         NET3
; 18.2       7.4       Tpd       1         AND2       $1126:B        NET2
; 10.8       10.8      Tcq       2         DF1        $FF1:CLK       NET1
; 0.0        0.0       Psk       4         $FF2:CLK      CLK

```

The longest register to register delay is 31.8ns giving a maximum operating frequency of 31.5 Mhz. The total delay is accumulated by combining the clock skew, the clock to q of flip flop 1, the gate propagation delays and the setup time to flip flop 2. As can be seen the delays are most significant for those macros that have more than one level of logic. At this point it is prudent to mention that this maximum frequency of operation is conservative because of the following key points:

- a) The actual delays can be heavily influenced by the manual assignment of pins and macros as well as the assignment of criticality.
- b) The use of the ALES (advanced logic enhancer and synthesiser) tool can optimise the design for both area and speed.
- c) The TIMER has been set to use the standard commercial fpga and not the fastest speed graded fpga.

INPUT TO OUTPUT DELAY

While the register to register delays give us the maximum operating frequency for the design internally within the chip, we can also use the timer to give us the input to output delays of the chip. Input to Output delay is typically used for determining the delay on combinatorial paths from input pad to output pad. We do this by setting the startset to 'INPAD' and the endset to 'OUTPAD'. We can then pick up the longest or shortest delays, as for the CLOCK to Gated, and expand on these delays.

```

; 1st longest path to all endpins
; Rank      Total      Start pin      First Net End Net      End pin
; 0         26.4      XCLK:PAD      CLK          OUT1          $115:PAD
; 1 pins

; 1st longest path to $115:PAD (falling) (Rank: 0)
; Total      Delay      Typ      Load      Macro      Start pin      Net name
; 26.4       9.8       Tpd       0         OUTBUF     $115:D         OUT1
; 16.6       5.7       Tpd       2         DF1        $113:CLK       NET4
; 10.9       10.9      Tpd       4         CLKBUF     XCLK:PAD      CLK

```

Thus we see that the 26.4ns input to output delay consists of the propagation delays through the clock buffer and output buffer and the delay through the flip flop.

SETUP AND HOLD TIME REQUIREMENTS

Internal setup time checking is done implicitly. For a synchronous design, the maximum period displayed by using a longest command will include the required setup time for a sequential macro. This setup time is essentially the time interval, for which the data must be present, immediately preceding the clock transition for the data to be recognised (Figure 3.). Thus we see that the setup time for flip flop 2 is 6.7ns.

; 1st longest path to \$1I3:D (falling) (Rank: 0)						
; Total	Delay	Typ	Load	Macro	Start pin	Net name
; 31.8	6.7	Tsu	0	DF1	FF2:D	
; 25.1	6.9	Tpd	1	NOR2	\$114:B	NET3
; 18.2	7.4	Tpd	1	AND2	\$1126:B	NET2
; 10.8	10.8	Tcq	2	DF1	FF1:CLK	NET1
; 0.0	0.0	Psk	4		FF2:CLK	CLK

If the shortest command is used then the timer will check internal hold time violations. The Hold Time is essentially the time interval immediately following the clock transition, during which the data input must be present for the input to be recognised. In this case we see that the hold time for flip flop D is 0ns.

; 1st shortest path to \$1I3:D (rising) (Rank: 0)						
; Total	Delay	Typ	Load	Macro	Start pin	Net name
; 24.5	0.0	Thd	0	DF1	FF2:D	
; 24.5	7.2	Tpd	1	NOR2	\$114:B	NET3
; 17.3	6.9	Tpd	1	AND2	\$1126:B	NET2
; 10.4	10.4	Tcq	2	DF1	FF1:CLK	NET1
; 0.0	0.0	Psk	4	FF2:CLK	CLK	

EXTERNAL SETUP TIME REQUIREMENTS

External setup time for the device is a critical timing parameter in a synchronous system design. The timing relationship between the external system clock and the external data must ensure that the internal setup time requirement of the sequential elements are met. Calculating the external setup time requirement is a two step process. First, determine the clock pad to register clock input delay. The external setup time required must satisfy the following equation:

$$\text{CLOCK PATH DELAY} - \text{DATA PATH DELAY} + \text{Tsu ext} > \text{Tsu int}$$

where Tsu int is the internal setup time requirement for a sequential macro and Tsu ext is the external setup time requirement for the data that must be provided by the system for proper device operation.

CLOCK SKEW

Clock skew is the delay variation on the clock net connecting two or more sequential elements, and is the difference between the largest delay and the smallest delay. Clock skew will depend on the number of loads on the clock circuit and the degree of clock balancing used in place and route. The clock balancing utility allows the designer to influence the placement and routing of clocked macros to minimise skew if required by balancing loads on the clock distribution network. Flip flops located on the same row will have negligible skew, but flip flops located on different rows will have skew depending on the difference in the number of clock loads on the rows. Flip flops clocked by I/O's will have skew relative to flip flops clocked by the clock driver.

The SKEWMODE option in the TIMER determines whether or not pin- clock skew time is calculated. Pin-clock skew (Psk) is a measure of the difference in delays for the clock line to reach more than one flip flop when there is a different delay path between flip- flops.

Net skew (Nsk) is calculated when flip flops on a common clock drive a non-storage macro (such as a gate). Net skew is the difference between the clock net delay or the path being examined and the smallest clock delay or the common clock network.

When timing constraints are significant, the clock skew should be considered along with the rest of the delays in the circuit. If there is a problem with clock skew, use of the load or clock balancing feature should be made.

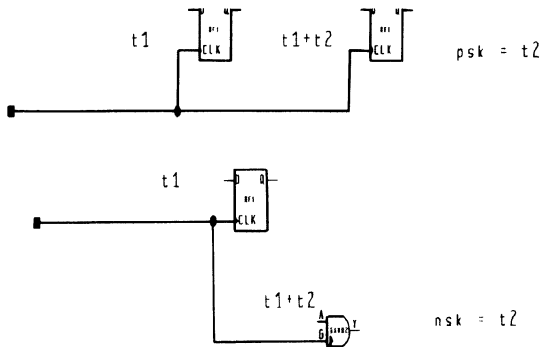


Figure 6. Clock Skew

PINS SELECTION

The timer sums delays along a path, which has a single starting and ending pin. Both the starting and ending point are macro inputs. For I/O buffers, the pad is the starting point for INBUFFS and the ending point for OUTBUFFS and TRIBUFFS. The pad of a BIBUF can be either a starting point, ending point or both. Thus by selecting the start and end points the timer will calculate the delay.

OTHER TIMER OPTIONS AND UTILITIES

The discussions in this section have been based on a synchronous design that has no user defined macros but there are also timer utilities to aid in analysing timings associated with asynchronous circuits and user defined macros. We will briefly discuss these utilities here.

NAMES AND LEVEL

NAMES specifies the name of a pin from a higher level in the design hierarchy than the level found in the TPC10 series macro library, making pins easier to reference. When NAMES is selected ON you do not need to refer to the pin's full path name. Only the user created macro name and its corresponding pin are needed. When NAMES is OFF, you must reference pins down to the TPC10 macro library level. The LEVEL option will allow the user to set one of three options; HARD, SOFT or USER. The TIMER will display the delay information for the entire soft or user defined macro if SOFT or USER are selected or break down the internal macro paths to show the internal delays at hard macro level.

OTHER SET COMMANDS

STOPSET When an asynchronous loop is detected, the TIMER issues a warning. Asynchronous loops must be broken for the TIMER to analyse the delays. This is facilitated in the TIMER by using a STOPSET. By creating a STOPSET containing a point in an asynchronous loop, TIMER is able to complete the analysis for the circuit.

BREAKSET When asynchronous loops are detected, a special set is generated that contains a possible breakpoint. This set is called BREAK_i, where *i* is an integer. If three asynchronous loops are detected, then the BREAKsets are created automatically (BREAK₁ etc.); one for each loop. The easiest way to add these break sets to the stopset is to use the 'ORSTOP' command.

> ORSTOP BREAK1

All of the required break points are then merged to the stopset, and any of the analysis commands can be executed.

TEMPSET In addition to break sets, TIMER also generates temp sets (temp1etc.), one temp set for each asynchronous loop. If you want to chose your own break points, you can display the contents of the temp set by using the showset command command to help determine which point to use and then use the addstop command to add the points to the stop set.

CONCLUSION

The ALS Timer is a comprehensive timing analysis tool which offers the ability to carry out a complete static timing analysis of both sequential and combinatorial circuits. It should be used in conjunction with a simulator to further enhance the design flow and with careful digital design practices all the necessary timing information can be extracted for a design.



SECTION 4.7

CRITICAL PATH ANALYSIS FOR FIELD PROGRAMMABLE GATE ARRAYS

Anne Phillips, Dan Powers, Mary Werling
Texas Instruments, Dallas
Applications

INTRODUCTION

Device speed, or timing, is a critical aspect of system design. A realistic estimate of the achievable system speed is often required early in the design phase to avoid waste of valuable design time. System speed, of course, depends on the operation of all system components. This application note describes how to estimate the timing constraints of a field-programmable gate array (FPGA) design. Remember: the FPGA is just one component within the design - other devices also affect the system speed.

METHODOLOGY

Three things are needed to estimate system speed for a TPC10 Series FPGA design:

1. TPC10 Series Family Data Sheet available from Texas Instruments.
2. TPC10 Series Data Sheet Supplement included at the end of this paper (PIN LOADING) (appendix).
3. Knowledge of the design's critical path (The critical path is entirely design dependant).

This application note begins by defining terms associated with estimating system speed (critical path, logic levels, and fanout). It then provides examples of estimating the propagation delay of a combinatorial path and the system speed of a sequential path.

When following the procedures outlined in this application note, remember that all data sheet delays are typical delays. To determine timing delays for a particular application, always consider the voltage and temperature conditions and derate the typical data sheet values accordingly.

CRITICAL PATH

What is a critical path? It is a path in the design which must meet certain critical timing requirements in order for the system to function properly. A critical path can be composed of any combination of hardwired or softwired macros, and input and output buffers for either combinatorial or sequential logic paths. This application note addresses each element of the critical path separately.

LOGIC LEVELS

After the critical path has been identified, the next step is to determine the hard or soft macros that will be used to implement the critical path from the TPC10 Series Family Data Sheet and to note the respective logic levels for each of those hard or soft macros. The terms "levels of logic" or "logic levels" refers to the number of logic modules that an input must cross to reach the output.

Hard macros have either one or two logic levels. The filled triangle symbol on the inputs of hard macros denotes two levels of logic. Hard macros without the triangle have only one level of logic. Figure 1. shows an example of one- and two-level hardwired macros. For a two-level hard macro, the logic module interconnection is always consistent and the timing characteristics

remain unchanged.

A soft macro is a collection of hard macros configured to make a more complex function. Logic levels for soft macros are listed in the TPC10 Series Family Data Sheet. Soft macros can have many logic levels.

Logic Levels

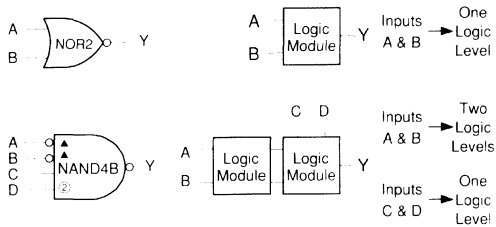


Figure 1

FANOUT

Fanout is the sum of the pin loads for all macro inputs connected to the output of a particular macro. The pin loading for every hard and soft macro is listed in the TPC10 series Data Sheet Supplement at the end of this paper (Pin Loading). Figure 2. illustrates the fanout of an element in a design.

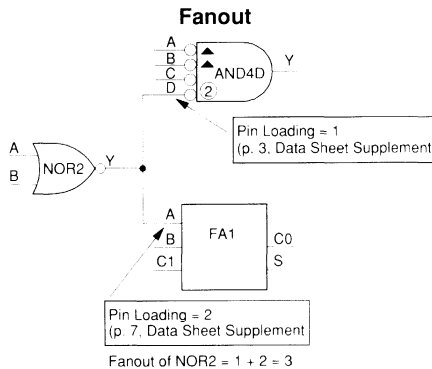


Figure 2

EXAMPLE 1 - COMBINATORIAL PATH

DELAY CAUSED BY A SINGLE-LEVEL HARD MACRO

With the help of the TPC10 Series Family Data Sheet and the Data Sheet Supplement (Pin Loading), typical delays caused by hard macros can be easily determined. In the example shown in Figure 3, the NOR2 hard macro is used.

LEVELS OF LOGIC

As a first step, the designer determines the number of logic levels for NOR2, the hard macro. The critical path is through the A input of NOR2. By looking at the NOR2 symbol in the Data Sheet you can see that there is no triangle on the A input. This indicates:

Logic levels = 1

FANOUT

Next, the fanout for NOR2 is determined. In this design (Figure 3.), the output of NOR2 feeds the A input of NAND4A and the A input of AND2. Page 3 of the Data Sheet Supplement (Pin Loading) reveals that the A input of the NAND4A macro has a pin loading of 1 and page 2 reveals that the A input of AND2 also has a pin loading of 1.

Figure 3.

Simply adding the pin loading of all inputs tied to the output of NOR2, or $1 + 1$, provides the fanout. In this case:

Fanout = 2

The TPC10 Series Family data sheet contains the table with information on the single-level logic module hard-macro switching characteristics. This information is needed to determine the typical delay caused by NOR2. According to that table, a hardwired macro in a critical path with one level of logic and a fanout of 2 has a delay of 5.8 ns. This value includes a statistical delay estimate for the routing between the NOR2 output and the NAND4A input.

NOTE

Many FPGA vendors do not include delays due to routing in their data sheets.

DELAY CAUSED BY A DOUBLE-LEVEL HARD MACRO

LEVELS OF LOGIC

The critical path is through the A input of the NAND4A hard macro. By looking in the Data Sheet, you can see that there is a triangle on the A input of the NAND4A. This denotes that there are two levels of logic through the A input of this hard macro.

Logic levels = 2

FANOUT

Next, the fanout for NAND4A is determined. In this design (Figure 4.), the output of NAND4A feeds the A0 input of MCMPC4. Page 12 of the Data Sheet Supplement (Pin Loading) reveals that the A0 input of MCMPC4 has a pin loading of 3. Therefore, the fanout of the NAND4A is 3.

Fanout = 3

The Data Sheet contains a table with information on double-level logic module hard macro switching characteristics. It shows that a hardwired macro with two levels of logic and a fanout of 3 has a delay of 10 ns.

Double-level Hard Macro Delay

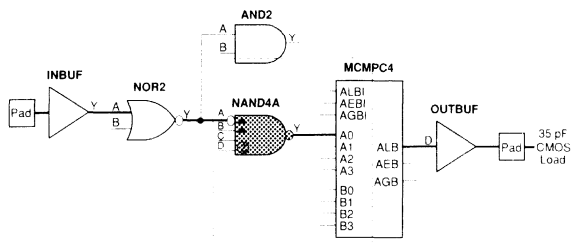


Figure 4.

DELAY CAUSED BY A SOFT MACRO

LEVELS OF LOGIC

The levels of logic for a soft macro are again given in the Data Sheet. The MCMPC4 soft macro has four logic levels.

FANOUT

It is impossible to determine the load on each of the hard macros within a soft macro without examining the soft macro schematic. Since the soft macro schematics are available only in the CAE environment, the load on the internal hard macros will have to be estimated.

1. Calculate the delay of n-1 levels of logic, where n is the total number of logic levels in the soft macro. A conservative estimate of three loads should be used to calculate the delay of n-1 levels of logic. The delay for the n-1 levels is:

$$\text{delay} = (n-1) * 6.2$$

where 6.2 ns is the delay for a single-level hard macro using critical routing and a fanout of 3.

2. Determine the load on the output of the macro. The delay on the nth level of logic is simply the delay of a single-level hard macro using critical routing and having the appropriate fanout.

3. Add the values from steps 1 and 2 to obtain the total delay of the soft macro.

The TPC10 Series Data Sheet shows that the soft macro used in this example (Figure 5.) has (n) = 4 logic levels.

1. The calculation for the delay of the first three levels of logic (n-1 levels) is:

$$(n-1) * 6.2 = (4-1) * 6.2 = 18.6 \text{ ns}$$

2. Determine the fanout for the fourth logic level in the MCMPC4 soft macro by looking at page 5 of the Data Sheet Supplement (Pin Loading) to find the pin loading on the D input of the OUTBUF macro. The fanout is 1.

Determine the delay for the fourth level of logic by looking at page 9 of the TPC10 Series Family Data Sheet. It shows the single-level logic module hard-macro switching characteristics for a critical path with a fanout of 1. The delay for the fourth logic level is 5.4 ns.

3. Find the total delay by adding the delay of the first three levels, 18.6 ns, to the delay of the fourth level, 5.4 ns. Thus, the total delay in the MCMPC4 soft macro is:

$$\text{Total delay} = 18.6 \text{ ns} + 5.4 \text{ ns} = 24 \text{ ns}$$

Soft Macro Delay

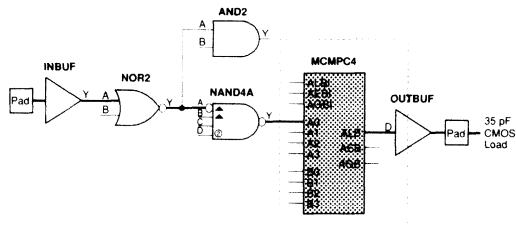


Figure 5.

DELAY CAUSED BY AN INPUT BUFFER

Input buffer delays (and bidirectional input buffers) are calculated much like hard macro delays (Figure 6). The only difference is that the propagation delay for the high-to-low transition is always used for critical paths. High-to-low transition delays are chosen because they are always greater than low-to-high transition delays as specified in the data sheet.

The Data Sheet lists the delay for an input buffer with a fanout of 1 as:

$$\text{Input buffer delay} = 6.9 \text{ ns}$$

Input Buffer Delay

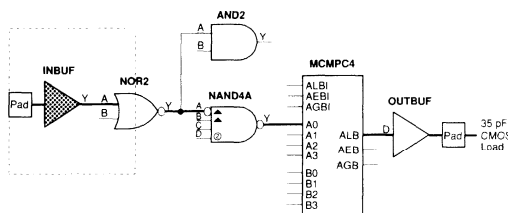


Figure 6.

DELAY CAUSED BY AN OUTPUT BUFFER

Output buffer delays (and three-state and bidirectional output buffers) are specified in the TPC10 Series Family Data Sheet for driving CMOS or TTL loads.

The output buffer delays are characterized for a load capacitance of 50 pF. To calculate a propagation delay for an output buffer with a load other than 50 pF, the delta tpd factors from the data sheet must be used.

The delay specified for the critical path evaluation should be the greater of the low-to-high propagation delay, or the high-to-low propagation delay.

The following equation calculates output buffer delay with a load capacitance other than 50 pF.

$$\text{tpd}(\text{actual}) = \text{tpd}(50\text{pF}) + [\text{delta tpd} * (\text{C}(\text{actual}) - 50)]$$

The delay caused by the output buffer shown in Figure 7 is determined with the help of further information from the TPC10 Series Family Data Sheet.

$$\text{tphl} = (3.9) + [.03 * (35 - 50)] = 3.5 \text{ ns}$$

$$\text{tplh} = (7.2) + [.07 * (35 - 50)] = 6.2 \text{ ns}$$

Output buffer delay = 6.2 ns

Output Buffer Delay

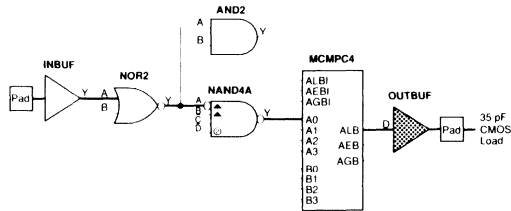


Figure 7.

COMBINATORIAL CRITICAL PATH

The components of the critical path have been identified and the delays have been calculated. The next step is to determine the cumulative timing delays of all elements in the critical path (Figure 8).

The critical path delay for a combinatorial path is the sum of all delay elements in the path. Note that the delay caused by the AND2 macro has not been added because AND2 is not in the critical path.

$$\text{INBUF} + \text{NOR2} + \text{NAND4A} + \text{MCMPC4} + \text{OUTBUF} \quad 6.9 + 5.8 + 10 + 24 + 6.2$$

$$\text{Total delay} = 52.9 \text{ ns}$$

The total estimated delay is a typical value. To account for temperature and voltage factors the typical delay is derated with the derating factors shown on page 8 of the data sheet. For example, to obtain the estimated worst-case delay for a design under commercial operating conditions the typical delay value is multiplied by 1.54.

The following calculation is for the example of a combinatorial path used in this application note:

$$\text{Worst case delay} = 52.9 \times 1.54 = 81.5 \text{ ns}$$

Combinatorial Path Delay

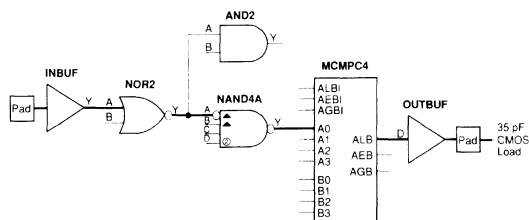


Figure 8.

EXAMPLE 2 - SEQUENTIAL CRITICAL PATH

The technique for analyzing a sequential critical path differs from the technique used for combinational paths. Setup and hold characteristics have to be considered and the critical path is broken into segments.

Hold time is the time for which a signal must be held at a specified state after the clock transition. The value is specified as 0 ns for TI FPGAs.

Setup time (tsu) is the time for which a signal must be maintained at a constant valid state before the clock transition. Setup times for both flip-flops and latches are given in the Data Sheet.

Propagation delay (tpd) of flip-flops and latches also must be taken into consideration when evaluating the achievable system speed of a sequential path. Propagation delays for flip-flops and latches are also given in the Data Sheet.

Sequential paths must be broken into segments for critical path analysis:

1. Inputs to clocked macros
2. Clocked macros to clocked macros
3. Clocked macros to outputs

The example in Figure 9 is divided into three segments. The first segment is between the input buffer and the first clocked macro. The second segment is from one clocked macro to another. The third segment is from the last clocked macro to the output buffer. The segments are compared and the one with the greatest delay determines the achievable system speed for the device.

With information in the TPC10 Series Family Data Sheet and the Data Sheet Supplement (Pin Loading), delays for the three segments are quite easily determined.

Delay caused by segment 1:

$$t_{\text{seg 1}} = t_{\text{pd}}(\text{inbuf}) + t_{\text{su}}(U1) = 9 + 3.9 = 12.9 \text{ ns}$$

Sequential Path Delay

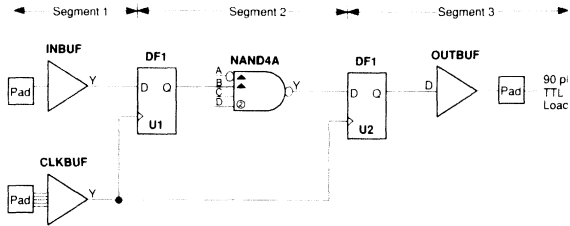


Figure 9.

Delay caused by segment 2:

$$t_{\text{seg 2}} = t_{\text{pd}}(\text{U1}) + t_{\text{pd}}(\text{NAND4A}) + t_{\text{su}}(\text{U2}) <T> = 5.4 + 9.2 + 3.9 = 18.5 \text{ ns}$$

Delay caused by segment 3:

$$t_{\text{seg 3}} = t_{\text{pd}}(\text{U2}) + t_{\text{pd}}(\text{outbuf}) = 5.4 + 7.3 = 12.7 \text{ ns}$$

In the example shown in Figure 9, segment 2 has the greatest path delay with 18.5 ns, thus, it determines the critical path delay. Again, this is a typical value. To obtain the estimated worst-case delay, multiply by 1.54.

$$\text{Worst-case delay value} = 18.5 \times 1.54 = 28.5 \text{ ns}$$

System clock frequency is calculated with the following simple equation:

$$\text{frequency} = 1/\text{delay}$$

$$\text{frequency} = 1/28.5 \text{ ns}$$

$$\text{frequency} = 35 \text{ MHz}$$

In this application note, the clock buffer delay has been completely neglected. This is a reasonable assumption for many applications. Generally, the clock buffer delay becomes a consideration only if its delay exceeds the largest segment delay. This is possible in register or latch intensive designs where the clock network is heavily loaded. In such situations, the clock buffer delay could be the limiting factor.

SUMMARY

This application note describes how to estimate the device speed of a Texas Instruments TPC10 series FPGA by calculating the delay caused by the various elements in the design's critical path. Examples of sequential and combinatorial critical paths are used to show how the calculations are performed. The result of the calculations provides an estimate of the speed with which the FPGA is likely to perform. Similar calculations can be done for the TPC12 Series FPGAs.

ALTERNATE METHOD - 10 NS RULE

A quick method to estimate achievable system speed is to assume 10 ns delay for each hard macro and 10 ns/logic level for each soft macro in the path.

PREFERRED METHOD

The most accurate way to estimate achievable system speed for a TI FPGA is to capture the critical path and analyze it using the TI Action Logic System.

SECTION 4.8

UNDERSTANDING HOW ALS PLACE & ROUTES

Dung Tu,
Texas Instruments, Germany
Technical Marketing

INTRODUCTION

Ti Field Programmable Gate Arrays (FPGA) are based on channeled array architecture consisting of an array of logic modules and interconnect channels that contain horizontal metal lines. There are also metal tracks running vertically over the logic modules. Each horizontal-vertical track intersection contains an antifuse which can be programmed to form a low resistance path between the intersecting metal tracks.

Among the vertical tracks there are Long Vertical Tracks (LVT) and module output tracks. This application note shows how you can interpret the files which are generated by the Action Logic System (ALS) during the Automatic Place & Route (APR) to get a better understanding about the array architecture and how the routing tracks are used by the APR.

LOGIC MODULE VERTICAL TRACKS

Each TPC10 series logic module has eight inputs and one output as shown in Figure 1. The eight inputs are arranged so that four inputs are available to the routing channel above the module, and four inputs are available to the channel below the module. The module output is available to four routing channels, two channels above the module and two channels below the module.

Figure 1. shows also one Long Vertical Track. While the module output track is dedicated to one module, the LVT can be used by any module.

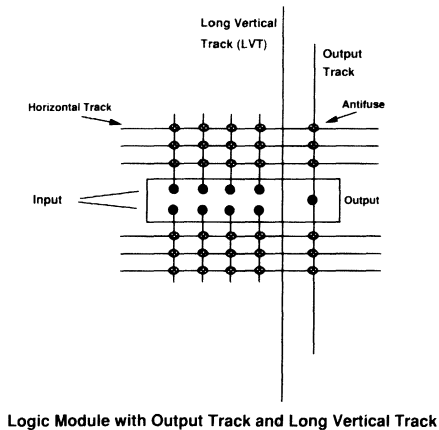


Figure 1.

EXAMPLE CIRCUIT

Figure 2. shows the example circuit - called LONGLINE - which is used to investigate the way the APR works. The circuit does nothing useful. The basic idea is to look at a signal path which consists of an input buffer at one end, an output buffer at the other end and anything in between which is built with one logic module - e.g. a simple inverter. This signal path is implemented in several different ways, using manual pin assignment, to see how the APR does the placement and routing. We assume we use a TPC1010 in 68-pin PLCC package.

In the first case, path 1, manual pin assignment is used to force the input pin to the top side of the array (pin 9) and the output pin to the bottom side (pin 27).

For the second case, path 2, we let the APR do the automatic pin assignment.

In the third case, path 3, we force the input again to the top side (pin 61) and the output to the bottom side (pin 43) but we add a buffer between the inverter and the output buffer. Figure 3. shows the pinout of a TPC1010 in 68-pin package.

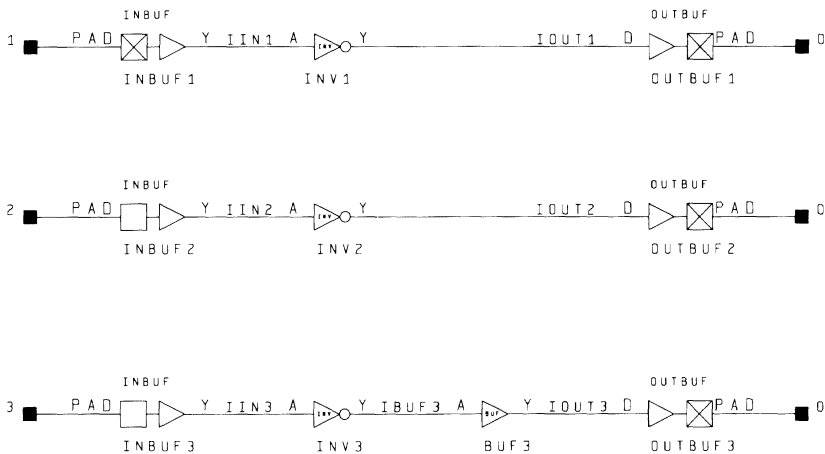
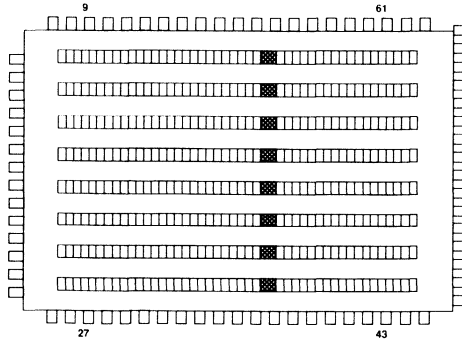


Figure 2. Evaluation circuit



TPC1010 pinout, 68-pin package

Figure 3.

PIN ASSIGNMENT, PIN FILE

During Place & Route, the Action Logic System (ALS) generates several files which are important for understanding how the APR works.

There are two files which show the locations of the pins. The .IPF file shows only the pins which are fixed by the manual Pin Edit. The .PIN file shows the locations of all pins.

```

DEF LONGLINE.
NET IN3; ; PIN:61.
USE ; INBUF3; FIX.
NET OUT3; ; PIN:43.
USE ; OUTBUF3; FIX.
NET IN1; ; PIN:9.
USE ; INBUF1; FIX.
NET OUT1; ; PIN:27.
USE ; OUTBUF1; FIX.
END.

```

Figure 4. .IPF file shows the fixed pins

```
DEF LONGLINE.  
NET IN2;;  
  PIN:67. <--- unfixed, assigned by APR  
NET IN1;;  
  PIN:9,  
  FIX:"".  
NET OUT2;;  
  PIN:68. <--- unfixed, assigned by APR  
NET IN3;;  
  PIN:61,  
  FIX:"".  
NET OUT1;;  
  PIN:27,  
  FIX:"".  
NET OUT3;;  
  PIN:43,  
  FIX:"".  
END.
```

Figure 5. .PIN file shows all fixed and unfixed pins

Looking at the .IPF file (Figure 4.) we can see that for path 1 and 3 the input and output are fixed at the pins on the top and the bottom of the array, respectively, as we intended.

The .PIN file (Figure 5.) shows that for path 2, where we used auto pin assignment, ALS has used pin 67 as input and pin 68 as output - two pins which are adjacent to each other.

LOCATION FILE, PLACEMENT FILE

Figure 6. illustrates the location file (.LOC) of the design. The location file shows where the APR has placed the components. For example, the input buffer INBUF1 is placed in a logic module which is at column number 6, row number 6, and the location is fixed by the user.

```
DEF LONGLINE.  
USE ; INBUF2;  
XY:27%6.  
USE ; INBUF1; <--- Component label  
XY:6%6. <--- XY coordinate: Column # % Row #  
FIX:". <--- Fixed by manual Pin Edit  
USE ; OUTBUF2;  
XY:23%6.  
USE ; INBUF3;  
XY:37%6,  
FIX:".  
USE ; OUTBUF1;  
XY:6%1,  
FIX:".  
USE ; OUTBUF3;  
XY:37%1,  
FIX:".  
USE ; BUF3;  
XY:37%3.  
USE ; INV2;  
XY:27%7.  
USE ; INV1;  
XY:6%0.  
USE ; INV3;  
XY:37%5.  
END.
```

Figure 6. Location file (.LOC) shows the locations of the components

Figure 7. shows the XY coordinates and the floorplan for TPC1010. This device consists of eight rows and 44 columns of logic modules. The logic module in the lower left corner is labeled (X=0, Y=0) and the top right corner is labeled (X=43, Y=7).

There are a total of 9 routing channels. The bottom routing channel (below row 0) is labeled channel 0, the top channel (above row 7) is labeled channel 8.

TPC1010 FLOORPLAN

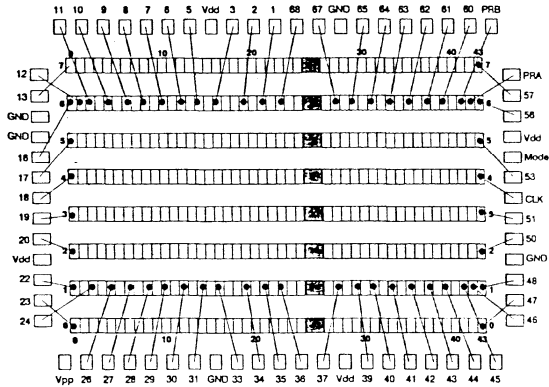


Figure 7.

The APR also generates a placement information file with the extension .PLI that informs about which nets were routed using long vertical tracks (Figure 8.).

One net of the design, net IIN1 of the signal path 1, needs a LVT for routing. This net is driven at location (X=6, Y=6) by INBUF1. The LVT runs over column 6 and spans from channel 8 to channel 0.

Average horizontal net length is 1.71429 columns.
 No nets need long horizontal tracks.
 A total of 1 nets need long vertical tracks.
 Automatic Placement completed successfully.

The design has 7 routed (excluding GND, VCC, global CLKs) nets. 6 nets need no long vertical tracks, av. horiz. length for these nets is 1.6667, av. fanout 1.0000.

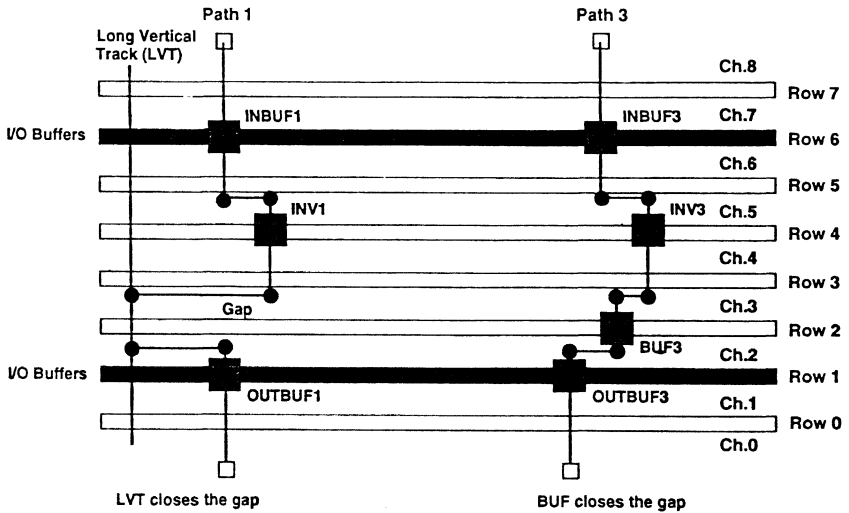
1 nets need to use long vertical routing tracks.
 1 of such tracks are of the standard type (LVT),
 av. horiz. length for these nets is 2.0000, av. fanout 1.0000.

List of all nets using standard long vertical tracks (LVT) to route:

The net IIN1 driven at location XY = (6, 6) uses an LVT.
 LVT data: column = 6, Y-span = (8, 0).
 Net data: fanout = 1, Y-spread of inputs = (0, 1).

Fig 8: Placement information file (.PLI)

Fig 9 depicts why a LVT is needed for path 1.



Long Vertical Track

Figure 9

The INBUF1 is fixed at row 6, the OUTBUF1 is fixed at row 1. Because the output track of a module spans two channels, the output of INBUF1 can be routed to channel 5, where it can be connected to the input of the inverter INV1.

The output of INV1 spans again two channels so that it can be routed to channel 3. However, because the input of OUTBUF1 can span only one channel to channel 2, there is a gap. Therefore, a LVT is needed to close this gap. In reality, the APR did require a LVT but it placed the INV1 at row 0 instead of row 4 and the LVT spans from channel 0 to channel 8.

In path 3, because an additional buffer is used, this buffer can close the gap so that no LVT is needed.

Path 2 does not have any fixed pin. In this case the APR has used two adjacent pins for the input (pin 67) and output (pin 68) to minimize the delay.

Figure 10 shows the delays which are analyzed by the Timer.

```

; 1st longest path to all endpoints
; Rank   Total   Start pin   First Net   End Net     End pin
; 0      33.1   INBUF1:PAD  IIN1       OUT1        OUTBUF1:PAD
; 1      32.9   INBUF3:PAD  IIN3       OUT3        OUTBUF3:PAD
; 2      28.1   INBUF2:PAD  IIN2       OUT2        OUTBUF2:PAD
; 3 pins

; 1st longest path to OUTBUF1:PAD (rising) (Rank: 0)
; Total   Delay   Typ   Load   Macro   Start pin   Net name
; 33.1    8.5    Tpd   0      OUTBUF  OUTBUF1:D   OUT1
; 24.6    5.9    Tpd   1      INV     INV1:A      IOUT1
; 18.7    18.7   Tpd   1      INBUF   INBUF1:PAD  IIN1

; 1st longest path to OUTBUF3:PAD (falling) (Rank: 1)
; Total   Delay   Typ   Load   Macro   Start pin   Net name
; 32.9    9.8    Tpd   0      OUTBUF  OUTBUF3:D   OUT3
; 23.1    5.5    Tpd   1      BUF     BUF3:A      IOUT3
; 17.6    6.8    Tpd   1      INV     INV3:A      IBUF3
; 10.8    10.8   Tpd   1      INBUF   INBUF3:PAD  IIN3

; 1st longest path to OUTBUF2:PAD (falling) (Rank: 2)
; Total   Delay   Typ   Load   Macro   Start pin   Net name
; 28.1    9.9    Tpd   0      OUTBUF  OUTBUF2:D   OUT2
; 18.2    7.4    Tpd   1      INV     INV2:A      IOUT2
; 10.8    10.8   Tpd   1      INBUF   INBUF2:PAD  IIN2

```

Figure 10. Path delays as analyzed by the Timer

Of the three signal paths, path 1 has the longest delay (Rank 0) due to the LVT. Path 3 is slightly faster although it contains one buffer more. Path 2 is the fastest thanks to the automatic pin assignment.

The additional delay due to the use of a LVT in path 1 can be obtained by breaking down the total path delay into its delay elements. The delay for the hardmacro INBUF from INBUF1:PAD to net IIN1 is 18.7ns. In path 2 and path 3, this delay is 10.8ns. This translates to an additional delay of about 8ns for the LVT.

Fig 11 shows the floorplan for TPC1020 in 84-pin package.

TPC1020 FLOORPLAN

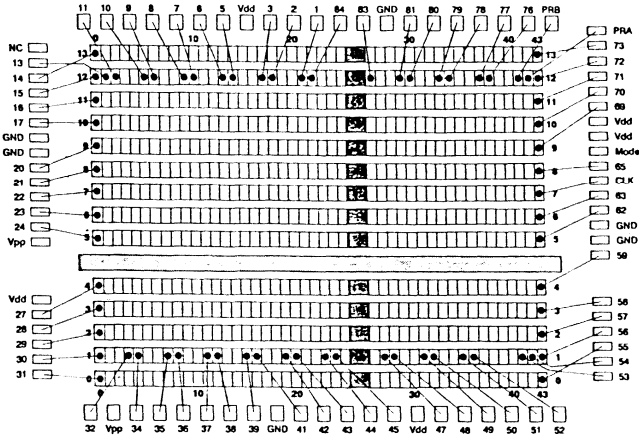


Figure 11

CONCLUSION

In most of the cases, Texas Instruments Field Programmable Gate Arrays can be placed and routed 100% automatically by the Action Logic System Place & Route software so you do not need to take care of the details of how it works. How it works however, if you want to use advanced features like manual placement control, timing optimization, etc., you can look at the files which are described above to get an idea where you can start.



SECTION 4.9

USING ALS DEBUG

Peer Uhlemann
Texas Instruments, Germany
Applications

INTRODUCTION

The TI-Action Logic System has a Debug option which allows 100% observability of every internal signal node while testing the FPGA in the activator. This complements the 'Actionprobe' (ref.4.10 Action Probe For Functional Debug) which allows 100% observability of every signal node while the FPGA is in the target board. (Figure 1.).

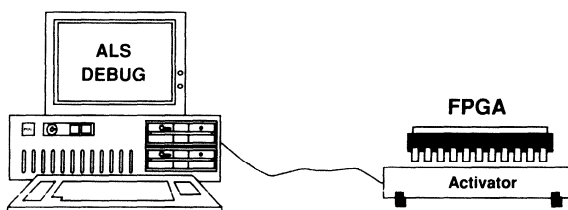


Figure 1. FPGA Debugging

This application note shows how to use Debug to test a 4-bit counter.

FUNCTIONAL DEBUG

The FPGA Debug option is a functional test to verify that the chip operates correctly once programmed. From within the TI ALS environment, using the Debug menu option, the user can force the state of the input pins and examine the response of observed output pins. The observed outputs are determined by the programmed device, rather than a simulation model as for simulation.

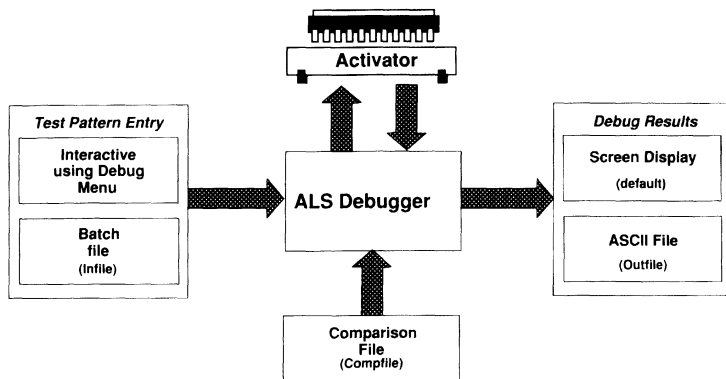


Figure 2. Debug Flow

Figure 2. illustrates the Debug flow. The Debugger is driven by commands which can be typed in from the keyboard, selected from a menu, or read from a batch file (Infile). These three types of inputs for the debugger allow the application of test vectors to the programmed FPGA. The Debug results are displayed on the screen or can be stored as an ASCII file and these results compared with the expected output values using a comparison file (Compfile). This file can be generated from a modified simulation output file. The following are prerequisites for the debug process.

- The tools only work on programmed devices.
- PROGRAM and PROBE security fuses must be unprogrammed.
- Pins SDI and DCLK are not available as user I/Os during Debug. The input to your circuit from these pins will be a logic 0 while you are in Debug mode.
- Pins PRA and PRB are not available as user I/Os during Debug.

TI-ALS DEBUG MENU

After logging into the TI-ALS system by typing ALS 'design_name' and then Debugger the following submenu is available see Figure 3.

Debugger	
Assign	Set a signal or a vector to a specified value
Low	Set a signal or vector to low
High	Set a signal or vector to high
Hi-Z	Set a signal or vector to tri-state
FAssign	File Assign: Load Input Stimulus
Step	Execute the commands previously specified
Print	Print the current value of the nodes to the screen
FPrint	Print the current value of the nodes to a file
Setup	Selects: Stimulus Vector Input File, Expected Responses Input File Output File, Load File
ICP	In-Circuit-Probe: Invokes In-Circuit Probe
View	Scroll screen
Exit	Exit to ALS Main Menu
Design cnt4	Current design name cnt4

Figure 3. TI-ALS Debug Menu

COUNTER TA161

The TA161 is a synchronous 4-bit counter with preset and clear. This counter is available as a softmacro in the FPGA library. A softmacro is a predefined block consisting of multiple hardmacros and/or other soft macros. Figure 4. shows the schematic of the 4-bit counter.

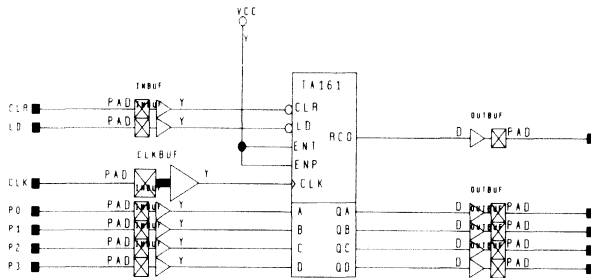


Figure 4. 4-Bit Counter TA161

The cascading inputs ENT and ENP are connected with Vcc. The load input LD is used to load a preset number (P0 to P3) to the counter. A sequence of clear, load, clock and outputs are shown in Figure 5.

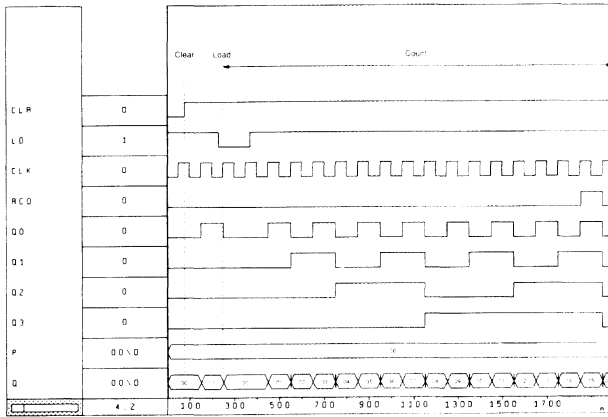


Figure 5. TA161 Waveforms

The counter does the following:

- Clear outputs to zero.
- Load to a binary zero.
(P0=0, P1=0, P2=0, P3=0)
- Count up to the outputs and reset.

CREATING A COMMAND FILE

A command file is the simplest way to apply a large number of command sequences to a device without having to re-type the commands. Command files can be created using any ASCII text editor, such as ViewText but debug commands in a command file must be enclosed in parenthesis. Alternatively commands can be entered from the Debugger menu. After creation of a command file the file can be run while in the Debugger by selecting 'Setup Loadfile' and entering the name of the command file. A sample command input file is shown in figure 6. The line numbers in the first column are included only for reference purpose.

```

1  emit "DEBUGGER DEMO FOR CNT4 DESIGN")
2  (vector P P3 P2 P1 P0)
3  (vector Q Q3 Q2 Q1 Q0)
4  (tabadd CLR LD CLK P Q RCO)
5  (infile "/designs/cnt4/cnt4.pat")
6  (outfile "/designs/cnt4/cnt4.res")
7  (comfile "/designs/cnt4/cnt4.cmp")
8  (define (clock) (I CLK) (step) (fprint) (print) (h CLK) (step)
(fprint) (print) (fcomp Q))
9  (define (clear) (I CLR) (clock) (h CLR) (clock))
10 (define (load) (I LD) (fassign P) (clock) (h LD) (clock))
11 (clear)
12 (load)
13 (repeat 15 (clock))

```

Figure 6. Command File cnt4.deb

EXPLANATION OF EXAMPLE

- Line 1: Print the text to the screen. The text must be enclosed in double quotes.
- Line 2: Vector groups the load input bits to a vector P.
- Line 3: Group the counter outputs to a vector Q.
- Line 4: The tabadd command causes the nodes or vectors (CLR, LD, CLK, P, Q, RCO) to be displayed or printed when the print (screen) or fprint (file) command are executed.
- Line 5: Infile opens an input file cnt4.pat (Figure 7) containing input stimulus which can be applied to the device with the fassign command.
- Line 6: Outfile opens a file cnt4.res (Figure 9) for writing Debugger results with fprint command. The Debugger outputs resulting data in tabular form.
- Line 7: Compfile opens a file cnt4.cmp (Figure 8) used with the fcomp command. The file contains the expected output values which will be compared with the actual output values.
- Line 8: The define command create a user macro clock which provides a pulse to the CLK input and prints all of the nodes to the outfile cnt4.res.
- Line 9: Defines user-macro, clear, which clears the counter.
- Line 10: Defines user-macro,load, which reads a load vector P from the input vector file cnt4.pat and load the counter.
- Line 11: Executes the user macro clear.
- Line 12: Executes the user macro load.
- Line 13: Repeating the macro clock 15 times, which causes the counter to count for 15 cycles.

Figure 7. Infile cnt4.pat

```
0b0000
0b0001
0b0000
0b0001
0b0010
0b0011
0b0100
0b0101
0b0110
0b0111
0b1000
0b1001
0b1010
0b1011
0b1100
0b1101
0b1110
0b1111
0b0000
```

Figure 8. Compfile cnt4.cmp

S T E P	C L R	L D	C L K	P	Q	R C O
00001 :	0	1	0	1111	0000	0
00002 :	0	1	1	1111	0000	0
00003 :	1	1	0	1111	0000	0
00004 :	1	1	1	1111	0001	0
00005 :	1	0	0	0000	0001	0
00006 :	1	0	1	0000	0000	0
00007 :	1	1	0	0000	0000	0
00008 :	1	1	1	0000	0001	0
00009 :	1	1	0	0000	0001	0
00010 :	1	1	1	0000	0010	0
00011 :	1	1	0	0000	0010	0
00012 :	1	1	1	0000	0011	0
00013 :	1	1	0	0000	0011	0
00014 :	1	1	1	0000	0100	0
00015 :	1	1	0	0000	0100	0
00016 :	1	1	1	0000	0101	0
00017 :	1	1	0	0000	0101	0
00018 :	1	1	1	0000	0110	0
00019 :	1	1	0	0000	0110	0
00020 :	1	1	1	0000	0111	0
00021 :	1	1	0	0000	0111	0
00022 :	1	1	1	0000	1000	0
00023 :	1	1	0	0000	1000	0
00024 :	1	1	1	0000	1001	0
00025 :	1	1	0	0000	1001	0
00026 :	1	1	1	0000	1010	0
00027 :	1	1	0	0000	1010	0
00028 :	1	1	1	0000	1011	0
00029 :	1	1	0	0000	1011	0
00030 :	1	1	1	0000	1100	0
00031 :	1	1	0	0000	1100	0
00032 :	1	1	1	0000	1101	0
00033 :	1	1	0	0000	1101	0
00034 :	1	1	1	0000	1110	0
00035 :	1	1	0	0000	1110	0
00036 :	1	1	1	0000	1111	1
00037 :	1	1	0	0000	1111	1
00038 :	1	1	1	0000	0000	0

Figure 9. Outfile cnt4.res

SECTION 4.10

ACTIONPROBE FOR FUNCTIONAL DEBUG

Doug Mackay
Texas Instruments UK
Technical Marketing

INTRODUCTION

The Actionprobe is a unique feature of the TI FPGA architecture which allows 100% Observability of any two internal nodes, while the FPGA device is functioning in the target system. Internal nodes toggling up to 10Mhz for TPC 10 devices and up to 50Mhz for TPC 12 devices can be observed using this system. This is achieved by programming the FPGA in the normal way and inserting the device into the Actionprobe module before being plugged into the target system. Two external pins, PRA and PRB are controlled from the TI ALS Debugger menu allowing the selected internal nodes to be observed on the pins.

ACTIONPROBE SETUP

Figure 1 shows the setup for using the Actionprobe for in-circuit diagnostics. The TI-ALS development system, Actionprobe module, Target System, programmed TI FPGA and an Oscilloscope or Logic Analyzer are required to perform the in-circuit probe function.

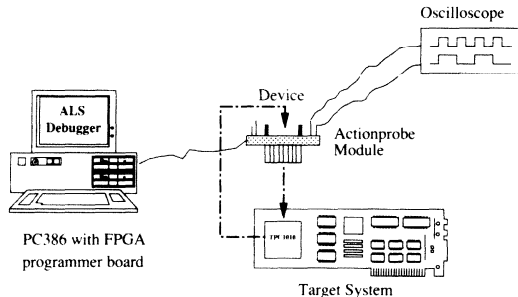


Figure 1. Actionprobe Setup

The programmed FPGA is inserted into the actionprobe module and then the module inserted into the target system. The system will still function normally as the module will appear transparent. A serial link connects the module to the TI-ALS system which is used to interface to the designer, who will be prompted for the internal node names. Once the nodes have been connected to the package pins an Oscilloscope is used on the pins to observe the waveforms. These waveforms will directly represent the node chosen but will be inverted. The actionprobe module has test pins for all the device pins allowing easy connection for observation.

Its not recommended to perform precise timing measurements with the probes as small delays will be inherent in routing the signal to the package pins.

CONFIGURING THE ACTIONPROBE

From within the Debug menu of the TI-ALS Development System the In- Circuit Probe (ICP) command is selected which brings up a small Dialog box. This prompts for the logic module pin name or internal net name for probe A and probe B. Figure 2 shows the menu entry for a simple D type Flip-Flop example.

Net CLK is selected for probe A and pin FF1:Q is selected for probe B. Selecting the OK from the dialog box will now program the FPGA device via the Actionprobe module situated in the target system. This takes about a second to program and can be reprogrammed and infinite number of times.

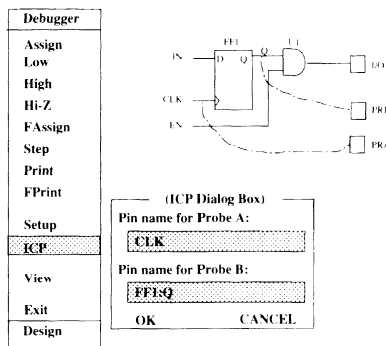


Figure 2. Configuring the Actionprobe

ACTIONPROBE RESOURCES

To obtain this unique feature of 100% observability in the target system five device package pins are required to carry out the necessary set-up and signal observation. If the features of Actionprobe are not required then four of the five pins can be configured as inputs or outputs. This is shown below:

PIN	FUNCTION	NON ACTIONPROBE ASSIGNMENT
PA	Probe A	Input
PRB	Probe B	Input
SDI	Serial Data Input	Output
DCLK	Data Clock for SDI	Output
MODE	Selects User or Debug mode	GND

A 10K pull-down resistor should be used on the PCB for SDI, DCLK and mode pin when using Actionprobe. Figure 3. shows a diagram of the location of the pins for a 68 PLCC device. This figure also shows the internal registers used to store the address of the selected net loaded via the SDI pin.

68 PLCC TI FPGA Device

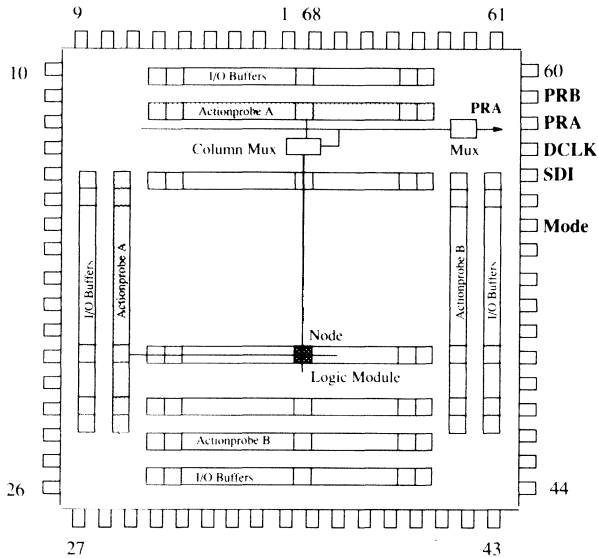


Figure 3. Resources used for 68 PLCC Device

Each probe pin has a row and column register which stores the address for the internal node selected. A sequence of multiplexers is then used to channel the internal node to the package pin for observation by an Oscilloscope or Logic Analyzer. As the address is stored in internal registers, the connection can be removed from the TI-ALS system while still retaining the address in the FPGA device.

DISCUSSION

This paper describes the unique features of the TI-FPGA allowing 100% observability after the FPGA is programmed and working in the target system. This invaluable debug tool is used when the FPGA is inserted into the target system but due to unforeseen conditions in the external environment to the FPGA the system functions incorrectly. An example of this would be asynchronous switching signals which would be impossible to emulate on a simulator.

Debugging this type of problem is very time consuming and could take a designer days/weeks to find the problem and fix it. With the Actionprobe the designer could find and fix the problem in a matter of hours, saving valuable development time and allowing the end product earlier entry into the market.

SECTION FIVE

LOGIC SYNTHESIS

SECTION 5.1

LOGIC SYNTHESIS FOR FIELD PROGRAMMABLE GATE ARRAYS

Dung Tu
Texas Instruments, Germany
Technical Marketing

INTRODUCTION

Schematic entry has been for a long time the only tool for electronics designers to design an electronic circuit or system. During the last few years, logic synthesis has become a very attractive alternative. This application note describes the design methodology trend and the available logic synthesis tools for FPGAs on a PC or a workstation.

DESIGN METHODOLOGY TREND

During the last 15 years the task of electronics design engineers has undergone significant change. In the 1970's, PCB's with a handful of TTL IC's, each with a density of ten to a maximum of 100 gates, were state of the art. The most popular equipment in the design labs was the oscilloscope. CAD tools and simulation were nearly unknown for typical designers.

At the beginning of the 1980's, a product called Programmable Array Logic was introduced to the market. This product has revolutionized the logic design methodology due to its user-programmability. It was the first time that the average designer encountered logic synthesis, i.e. describing a circuit using Boolean equations or state machine entry, then running a compiler to generate a fuse file used to program the PAL's. There are good PAL/PLD logic synthesis tools which are supported either by the PAL vendors or third parties. Simulation is however not necessary for PAL's because the devices have low complexity and the timing is normally not critical.

Around the same time ASICs appeared and ASIC design methodology became an essential tool to design complex systems. Due to the need to verify the functionality of a design, especially the timing, simulation is one of the most important steps of the design flow. A CAD ASIC design environment has typically a good simulator which is tightly coupled with a powerful schematic entry tool. This hardware oriented gate-level design approach provides an experienced designer with good visibility of his design. He can modify a node or a net and can simulate to see what happens with the signal.

However, today ASIC exist with 150k gates or more on one chip. At this level of complexity, it is almost impossible to do a design at the gate level because it is not only time consuming to design the circuit but it is also difficult to document or modify the design. These are some of the reasons why high-level hardware description languages (HDL) and especially the standardized VHDL is becoming the tool of choice for high gate count designs. On the highest level, HDL's allow description of a circuit as behavioural models. A design compiler can synthesize, optimize and map the design to a specific technology library. Design with HDL is comparable with programming using a high level language while design with schematic entry is similar to programming with a machine code. The productivity increase can be a factor of five or more.

Field Programmable Gate Arrays are high-density programmable logic devices which are intended to build a bridge between PLD's and classic ASIC's. Small FPGA's are used for production while the more complex FPGA's are useful for prototyping a design which can be built later in a gate array or standard cell.

Due to the higher complexity, the smallest FPGA can replace 10 PAL's, FPGA designs need to be simulated so that the timing can be verified. That is why FPGA design kits are normally delivered with a simulator and a schematic entry tool. On the other hand, there are many PLD designers who have experience with PLD logic synthesis and wish to use a similar approach for FPGA's. For these designers gate-level design is a step backwards. Another important reason for logic synthesis is the need to migrate an FPGA to an ASIC.

FPGA LOGIC SYNTHESIS TOOLS

There are two criteria for an FPGA logic synthesis tool. It must allow the PLD designers to use their familiar, existing PLD tools also for FPGA's. On the other hand it must be powerful enough to handle the migration to ASIC's.

In the market for high end logic synthesis tools, Synopsys VHDL, Cadence Verilog and Mentor Autologic are the main players. FPGA logic synthesis is now supported for all. The most popular PLD design tools at the low end are ABEL (Data I/O), LOG/iC (Isdata), PGA-Designer (Minc), Prologic (TI), PALASM2 (AMD) and CuPL (Logical Devices). Except for the PGA-Designer and the new ABELFPGA these tools do not support directly FPGAs. However, for historical reasons, the PDS format of PALASM2 has become a de facto standard for PLDs so the other PLD tools have an option to generate the PDS format. To enable PLD designers to use their existing design software also for FPGA, there is a tool called ALES (Action Logic Enhancer and Optimizer). ALES can take a PDS file as input, synthesize the logic and generate an optimized FPGA netlist.

FPGA LOGIC SYNTHESIS WITH ALES

The logic synthesis design flow with ALES is shown in figure 1.

ALES DESIGN FLOW

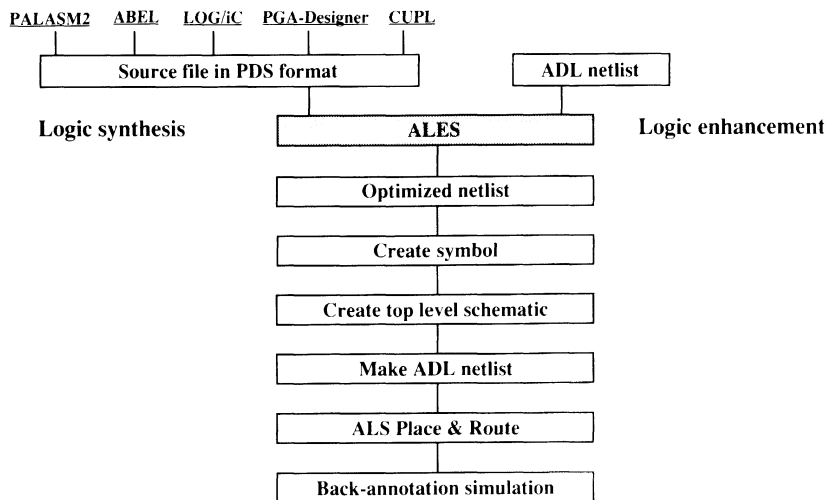


Figure 1. FPGA logic synthesis flow with ALES

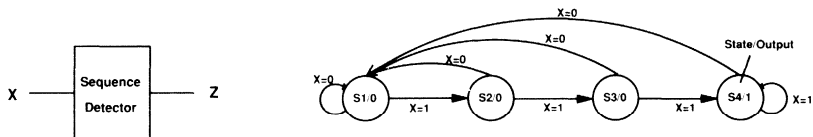
The PDS source file for a functional block which is to be synthesized can be entered as an ASCII file, using the PALASM2 syntax, or created by one of the PLD tools. ALES generates an optimized netlist for the block.

The next steps are to create a symbol for the block and a schematic for the top level. The top level schematic can consist of several synthesized blocks, mixed with normal FPGA hardmacros or softmacros. The top level schematic is then used to generate an ADL netlist for place & route as well as a wirelist so that back-annotation simulation can be done after place & route.

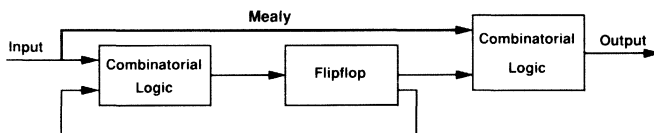
A classic text book example for a simple state machine is a sequence detector. This circuit monitors an input signal X and outputs a signal Z=1 when the input X is X=1 for three successive clock cycles.

Figure 2. illustrates the state diagram for the sequence detector, designed as a Mealy state machine. The initial state is S1 and the output is Z=0 (S1/0). If X=1, the machine moves to the next state, otherwise it stays in the same state. When in state S4 the output is Z=1, because X has been X=1 for three successive clock cycles.

SEQUENCE DETECTOR STATE DIAGRAM



- The sequence detector monitors an input signal X and outputs a signal Z=1 when X=1 for three successive clock cycles.



- The state machine is implemented as a Mealy machine, eg. the output depends not only on the state but also on the inputs.

Figure 2. State diagram sequence detector

Figure 3. shows the source file for the sequence detector using LOG/iC as the design software to describe the state machine. LOG/iC has a powerful state table syntax which allows a very efficient description of a state machine. The option 'PROGFORMAT = P-EQUATIONS' generates the PDS file for the state machine. Note that LOG/iC has generated two signals QQ1 and QQ2 automatically to store the state bits. These two signals must be drawn on the symbol but they are left open. The internal connection is included in the block netlist.

SOURCE FILES FOR ALES

Source file in LOGiC format

```

*IDENTIFICATION
Mealy Sequence Detector
Dung Tu
Texas Instruments, FPGA Application Europe
*X-NAMES
X:      ! Input signal for detector
*Y-NAMES
Z:      ! Output of detector
*FLOW TABLE
:State #, X Input value, Y Output value, Following state #
S 1.  X 1.  Y 0.  F 2.  ! The sequence detector counts
S 1.  X 0.  Y 0.  F 1.  ! the input bit stream X. It

S 2.  X 1.  Y 0.  F 3.  ! outputs a signal Z = 1 when
S 2.  X 0.  Y 0.  F 1.  ! X = 1 for 3 successive clocks

S 3.  X 1.  Y 0.  F 4.
S 3.  X 0.  Y 0.  F 1.

S 4.  X 1.  Y 1.  F 4.
S 4.  X 0.  Y 0.  F 1.
*STATE ASSIGNMENT
BINARY:
*RUN-CONTROL
PROGFORMAT = P-EQUATIONS.
*END

```

PDS format

```

TITLE Mealy Sequence Detector
PATTERN
REVISION
AUTHOR Dung Tu
COMPANY Texas Instruments, FPGA Application Europe
DATE 91.08.18 7:27:05

CHIP MEALY USER
CLK Z QQ2 QQ1 X

EQUATIONS
Z = X * QQ2 * QQ1

QQ2 := X * QQ1
      + X * QQ2

QQ1 := X * QQ2
      + X * QQ1

```

Figure 3. Source files for ALES

Figure 4. shows the top level schematic which includes the symbol 'Mealy' for the sequence detector and the input/output buffers.

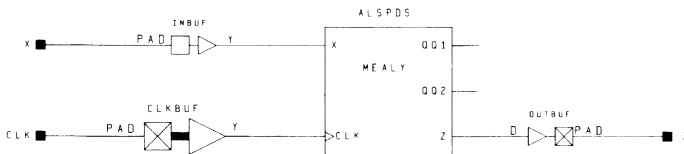


Figure 4. Top level schematic including I/O buffers

Figure 5. shows the result of the back-annotation simulation.

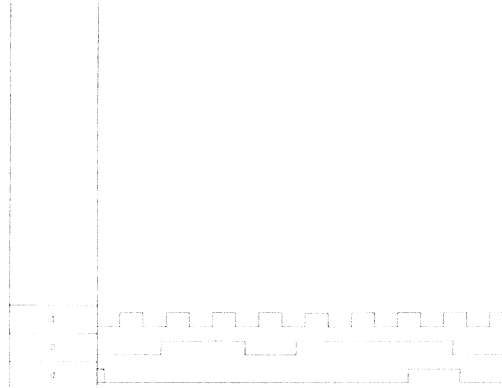


Figure 5. Back-annotation simulation

Figure 1. shows also a second interesting feature of ALES. Instead of a PDS file, the user can take an ADL netlist as input for the optimization. ALES can then enhance the logic and create a new netlist.

FPGA LOGIC SYNTHESIS WITH SYNOPSYS USING VHDL/VERILOG

Figure 6. illustrates the FPGA logic synthesis flow with Synopsys using VHDL or Verilog. The source file can be either in VHDL or in Verilog format. To map the design to FPGA, the Synopsys design compiler requires an FPGA library. The optimized design can be written out as an EDIF schematic or netlist. At this point there are two options.

The first option is to use an EDIF converter to convert the EDIF schematic either to a Viewlogic, Mentor or Valid schematic. The rest of the design flow can be done with an appropriate FPGA development kit running on the same platform.

In case of Viewlogic (the schematic conversion can be done with EDIF2VL2), the schematic can be transferred to a PC and the rest of the design flow could be PC based.

The second option is to work with the EDIF netlist and use an EDIF reader to translate the EDIF netlist to Viewlogic, Mentor or Valid netlist format. The MAKEADL tool of the ALS development system translates these netlist formats into ADL format.

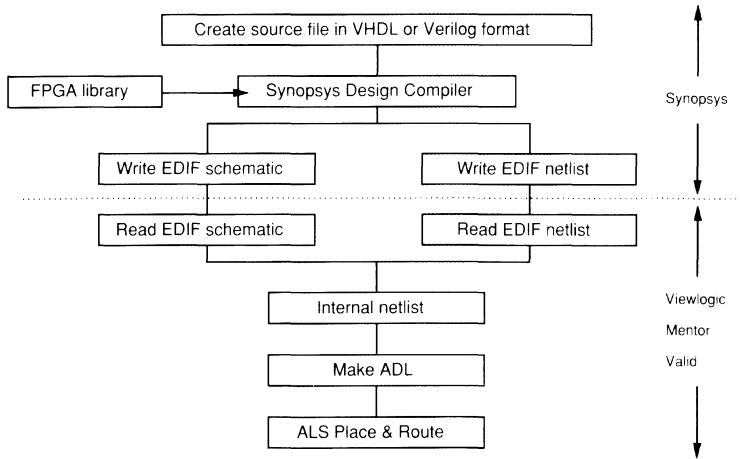


Figure 6. FPGA logic synthesis flow with Synopsys

Figure 7. shows as an example the VHDL source file and the script file for the same sequence detector as used to demonstrate the ALES flow.

VHDL SOURCE / SCRIPT FILES

VHDL source file

```
entity MEASEQ is
  port( X, CLK : in BIT;
        Z : out BIT);
end;
architecture BEHAVIOR of MEASEQ is
  type STATE_TYPE is (S1, S2, S3, S4);
  signal S, SNEXT: STATE_TYPE;
begin
  COMBIN: process
  begin
    case S is
      when S1 =>
        if X = '1' then Z <= '0'; SNEXT <= S2;
        else Z <= '0'; SNEXT <= S1;
        end if;
      when S2 =>
        if X = '1' then Z <= '0'; SNEXT <= S3;
        else Z <= '0'; SNEXT <= S1;
        end if;
      when S3 =>
        if X = '1' then Z <= '0'; SNEXT <= S4;
        else Z <= '0'; SNEXT <= S1;
        end if;
      when S4 =>
        if X = '1' then Z <= '1'; SNEXT <= S4;
        else Z <= '0'; SNEXT <= S1;
        end if;
    end case;
  end process;
  SYNCH: process
  begin
    wait until CLK'event and CLK = '1';
    S <= SNEXT;
  end process;
end BEHAVIOR;
```

Script file

```
/** Design check **/
designer = "Dung Tu ";
company = "Texas Instruments";
search_path = { /win17/fpga
link_library = tpc10.db
target_library = tpc10.db
symbol_library = tpc10.sdb
read f.vhdl measeq.vhdl
current_design = MEASEQ
write h f.db hierarchy o measeq.db
check_design > chkdns.rpt
report_design > measeq.rpt

/** Constraint: Optimize for area**/
max_area 0.0
compile
write h f.db o measeq_opt.db
report_area -cell timing > measeq.rpt
free all

/**Write EDIF schematic**/
read f db measeq_opt.db
current_design = MEASEQ
create_schematic_hierarchy size B
write h f.edif o measeq.edif
exit
```

Figure 7. VHDL source file. Synopsys script file

Figure 8. illustrates the schematic of the sequence detector which is synthesized by Synopsys.

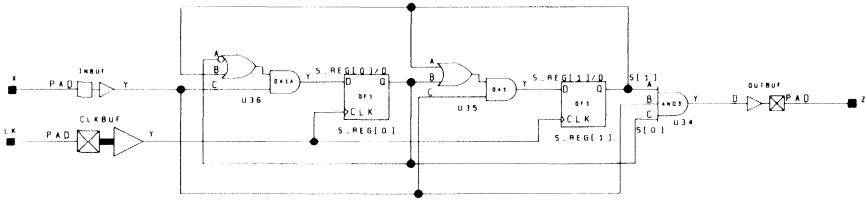


Figure 8. Sequence detector schematic

Even this simple example shows the usefulness of logic synthesis. Without this tool, the designer must use Karnaugh map techniques to synthesize and optimize the design manually, then derive the equations for the flipflops and translate these to a schematic.

SUMMARY

Logic synthesis allows you to describe a design independent of the technology so that you can concentrate on real design issues like architecture, functional blocks, etc. Depending on which technology is desirable, you can map the design later on to an appropriate technology library, either FPGA, gate array or standard cell. Due to the availability of good design tools and the standardization, VHDL is becoming the hardware description language of choice especially for high complexity designs.

SECTION 5.2

ACTION LOGIC ENHANCER AND SYNTHESIZER – ALES **An FPGA Logic Synthesis Tool for PC-386**

Doug Mackay
Texas Instruments UK
Technical Marketing

INTRODUCTION

Since the development of the Programmable Logic Device (PLD) in the late 70's we have seen how one can integrate many General Purpose Logic (GPL) parts into one PLD device and are now beginning to see the next stage in this Field Programmable Logic (FPL) evolution; The integration of multiple PLD products into a single Field Programmable Gate Array (FPGA). This is bringing the designer closer to having complete system level integration into a single FPL product. As the TI FPGA architecture is different from conventional PLD's the transition from PLD to TI FPGA is not straightforward and special design automated tools have been developed to make this transition fast and simple.

The Action Logic Enhancer/Synthesizer (ALES 1) product can automatically translate a PLD design into an FPGA netlist and optimize this design for speed and area. This allows PLD designers to continue to use their familiar design languages, state machines and boolean equations and also integrate a PLD design into a system level design captured from circuit schematics. It has another valuable benefit which allows an ADL netlist to be optimized for area or speed. This means that a design from schematic capture can be optimized to best utilize the TI FPGA architecture.

This application note will describe the basic features of the tool and its design flow followed by an application example using a 5 bit synchronous counter, which counts up to 24, designed from PALASM2 equations.

WHAT IS ALES

ALES as its title implies has two main functions; Firstly, it can Synthesize boolean equations and/or state machine functions into an optimized ADL netlist ready for place and route of the design using ALS and Secondly, it can be used to Optimize an ADL netlist into a more efficient implementation for the FPGA architecture. In either case, the designer has the option of minimizing delays or maximizing logic utilization using unique algorithms created for the TI FPGA architecture.

It is targeted at PLD designers who need a tool to interface with the popular PLD design tools in the market. ALES will accept source files from PALASM2, ABEL, CUPL, Log/IC, PGA Designer, and ADL netlist. Any one of the above design methods can be used or a combination allowing the designer to choose the best method for each portion of the design and simulate under the same environment within Viewlogic.

ALES DESIGN FLOW

Shown in Figure 1 is both the Logic Synthesis and Logic Enhancement design flows. The logic synthesis flow accepts a textual description of the design in PALASM2 from, either state machine or boolean equations. Once this has been generated, or converted from ABEL, CUPL, LOG/IC or PGA Designer, ALES is invoked with a switched set to optimize the design for area or speed.

ALES PC-386 DESIGN FLOW

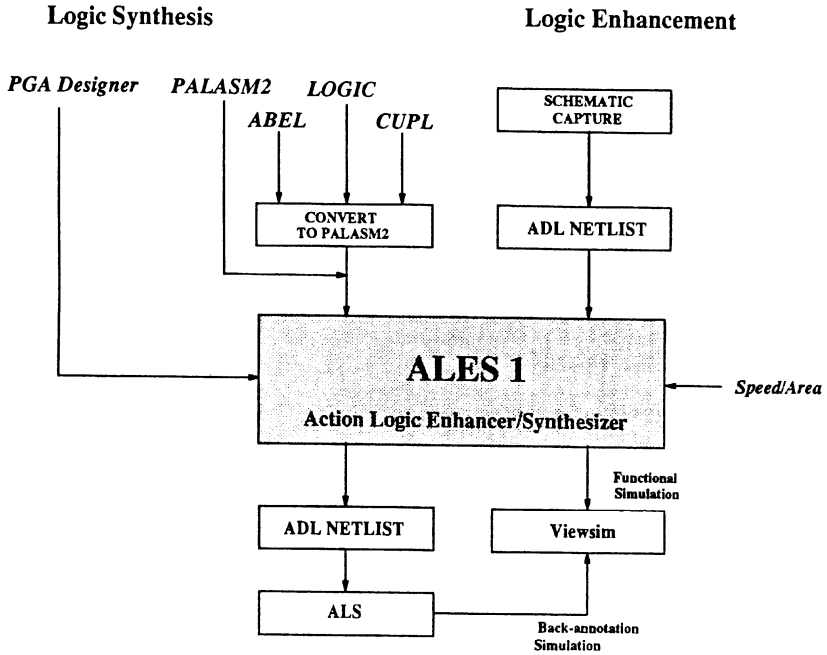


Figure 1. Ales Design Flow

ALES outputs a synthesized ADL netlist together with a Viewlogic wir file, for the PC-386 version. The wir file is used for simulation using Viewlogic's simulator, Viewsim. The ADL produced can then be imported into part of a design or solely used for place and route from within the ALS environment. This design flow will allow back-annotation using the extracted delays from the ALS environment.

The Logic optimization of an ADL netlist generated from schematic capture has a very similar design flow. A 'makeadl' utility which is given as part of the ALS system for each CAD package converts the schematic netlist into an ADL netlist. This is then used as the input to ALES with the switch set to optimize for Area or Speed. As for the PALASM2 input the output from ALES is an optimized ADL and a wirlist file for Viewlogic.

In addition to the Area/Speed option for ALES there are several switches that can be set, described in more detail in the next section.

ALES COMMAND REFERENCE

In addition to optimising for area or speed the following options are allowed when using ALES on the PC-386 platform.

ALES1 [options] <designs>

- ADL Read a TI ADL file (Logic Enhancement)
- PDS Read a PALASM2 format file (Logic Synthesis)
 - FAMILY:<f> Specifies 10/12 series family f = 1000 or 1200
 - DEL:<n> Attempts to make all paths <n> nS delay
 - PR Preserve signals from original netlist
 - CPU:<n> Limit the cpu time to n mins,(default=max=60)
 - NCL Disallow combinatorial loops
 - CM Create IO buffers automatically
 - NOBUFF:<n> Inhibit buffering on internal net <n> if fanout > 10 is detected.

More detailed information can be found in the ALES1 User Guide.

APPLICATION EXAMPLE (5 BIT SYNCHRONOUS COUNTER)

This section shows the operation of ALES1 showing the Logic Synthesis of a 5 bit synchronous counter PAL design. This counter was designed using PALASM2 (PDS) descriptive language for a standard TI PAL device 16R6.

SYNTHESIS OF A PALASM2 DESIGN

PALASM2 PDS file is an ASCII file which defines the functional operation of a programmable device. The syntax used is not case sensitive and will disregard blank spaces and blank lines. Many of the keywords in PDS are not relevant to ALES1 such as COMPANY and are ignored, a full description to the syntax required for a PDS file for ALES1 can be found in Appendix A of the ALES1 USER GUIDE.

The PALASM 2 source file used shown below has been generated to implement a synchronous counter onto a 16R6 PAL.

```

;=====
; CNT24.PDS
;=====
;
; This file is a PALASM 2 Equation Syntax File of a 5 bit
; synchronous counter which resets after counting to 24.
;
;=====
;
; Device declaration shows that a PAL of type 16R6 has been
; chosen for this design. A '/' is used for
; asserted low signals. The pin order is as per datasheet
; for the PAL. Pin 11 should be the enable signal which
; is not used and shown connected to GND.
CHIP cnt24 PAL16R6
clk /clr nc nc nc nc nc nc nc gnd gnd nc qa qb qc qd qe nc nc vcc
;
; Boolean registered equations are listed below.
EQUATIONS
qa := /qa;
qb := qa*/qb + /qa*qb;
qc := /qb*qc + /qa*qc + qa*qb*/qc;
qd := (/qc*qd + /qb*qd + /qa*qd + qa*qb*qc*/qd)*/qe;
qe := qa*qb*qc*qd*/qe + /qd*qe*/(qc+/qb+/qa);
;
; The output extension .SETF is used to provide a Asynchronous
; registered set
qa.setf = clr;
qb.setf = clr;
qc.setf = clr;
qd.setf = clr;
qe.setf = clr;
;===== END OF PALASM2 FILE =====

```

Figure 2 shows a graphical representation of the design flow used for this example. This design flow diagram shows the main tools used together with the files generated by each stage.

Having generated the necessary information for the source file ALES1 can be invoked with the required switches set. There is no need to modify the PDS file for using ALES and the reference to the 16R6 pal and its pin definition is handled by the software. For the 5 bit counter CNT24 the command to execute ALES is:

```
C:\DESIGNS> ALES1 -PDS -DEL:20 -FAMILY:1000 CNT24
```

This starts ALES1 for a PDS file cnt24.pds with the delay switch set to 20nS. This will attempt to make all delay paths in the design < 20 nS. The source file needs to reside under a directory in DESIGNS with the PDS extension.

LOGIC SYNTHESIS DESIGN FLOW

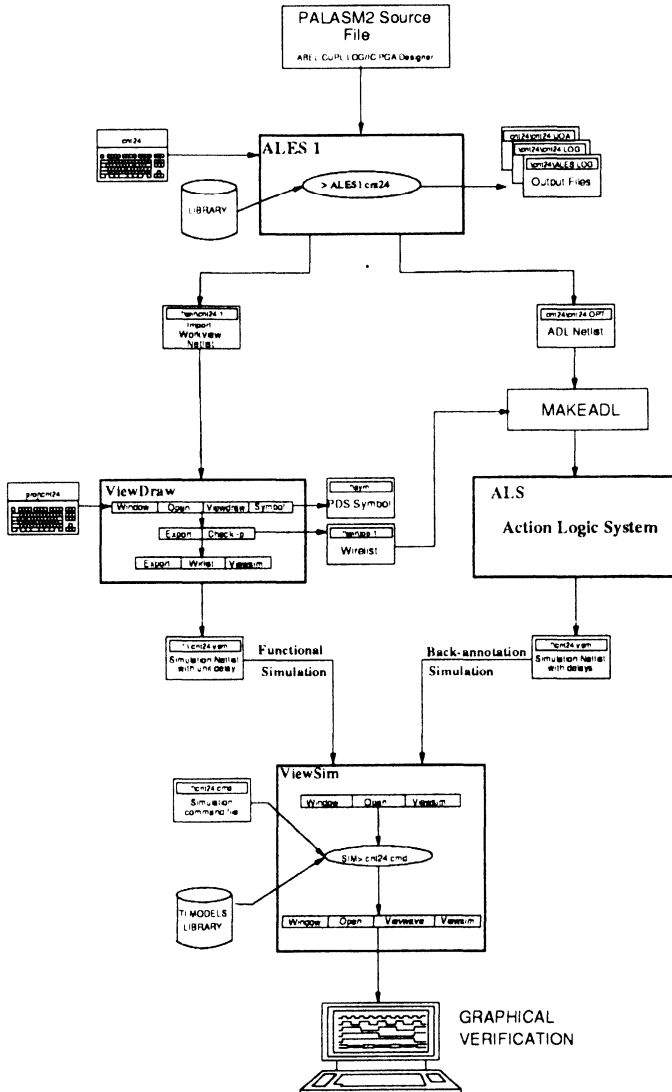


Figure 2 Logic Synthesis Design Flow

ALES1 will output four files in the design directory and a viewlogic wirelist in the WIR directory. These files are:

- design.UOA is an ADL netlist which is unoptimized.
- design.OPT is the resultant optimized ADL netlist.
- ALES1.LOG file will show all status, warning and error messages.
- design.LOG file gives a report of the ALES1 program.

The log report shows all the necessary steps that occur in the conversion and optimization phases. When the original design is translated from equations you can see that there are 73 logic modules with a maximum input to output delay of 76nS. This is then optimized with a result of 17 modules and 30nS delay on the first pass. A total of six passes is executed to try to meet the delay objective of 20nS. If the objective is not reached then the best result is saved which is 30nS for this example.

The Viewlogic wir files are then generated and saved under D:\DESIGNS\WIR directory. The wir file needs to be imported into the Viewlogic project area to verify the design by simulation. Before simulating a symbol needs to be drawn and an unattached attribute of ALSPDS which informs the MAKEADL netlist converter not to convert the wirelist for this part of the design but to use the design.OPT netlist which is also generated from ALES. Once generated this symbol can then be used in the top level of the design where input and output pads are added or it can be integrated into other parts of the design. Simulating the resultant wir file is done in the same way as if the design was created from schematic.

Once the design has been verified by pre-layout simulation MAKEADL will generate the ADL netlist required for place and route by ALS. After delay extraction by ALS and back-annotation of the delays into Viewlogic performing a post-layout simulation is the last stage of verification before an FPGA device can be programmed. An example of the PC-386 Viewlogic design flow is given in Section 3.1.

Figure 3 shows a graphical representation of the resultant wir file from ALES of the PALASM2 design.

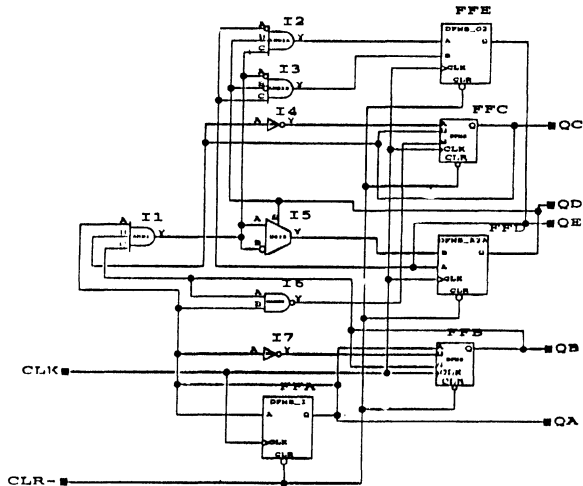


Figure 3. Counter Example

OPTIMISING AN ADL NETLIST

In addition to synthesizing a PAL into part or a full FPGA, ALES will also optimize an ADL netlist for speed and area. This option is performed by firstly generating an ADL netlist using MAKEADL. ALES is then invoked with the source file and the -ADL switch. This will then use an ADL netlist as a source file and output four files in the design directory and a viewlogic wirelist.

ANALYZING RESULTS OBTAINED FROM ALES

From the simple 5-Bit counter example used to describe the function of ALES we can see substantial reductions in propagation delay and area internal to the TI FPGA.

The results show that for this simple example a reduction of 56 Logic modules was performed by the logic synthesis tool. To obtain the same result from manual optimization would have taken a great deal longer. The results show how much logic minimization and increase in device performance is easily achievable by using ALES.

DISCUSSION

This section on Synthesizing PLD's into TI FPGA's has shown how the design tool ALES can easily be used to take existing PLD designs and integrate them into a TI FPGA. This can be partial or total integration allowing the designer to design and simulate PLD's within a schematic representing the complete system.

The example chosen was trivial but shows clearly how easily it is to use ALES for logic synthesis. The results show that a saving of 70% in area and 30% increase in speed is achievable.

An interesting statistic obtained from the results is that seventeen similar 16R6 PAL designs could be integrated into a single TPC 1010, the smallest TI FPGA in the family.

SECTION SIX
FPGA TO ASIC MIGRATION

SECTION 6.1

MIGRATING PROGRAMMABLE LOGIC TO ASIC INCLUDING AUTOMATIC TEST PATTERN GENERATION

James T. O'Connor
Texas Instruments, Dallas
Applications

ABSTRACT

This paper describes a methodology to migrate field programmable logic (FPL) to a gate array or standard cell ASIC with automated schematic translation, functional verification, static timing analysis, and test program generation. The input to the migration flow is the FPL netlist. The translation software replaces existing flip-flops with equivalent scan flip-flops and connects a scan chain. Automatic test pattern generation (ATPG) software then produces high fault coverage test patterns. Timing reports and workstation schematics of the ASIC are available post migration for timing verification and device simulation before manufacturing. This methodology reduces the execution time for FPL migration from man-months to man-days.

MOTIVATION

Among the advantages programmable logic offers for digital designs in development are:

- highly integrated, high performance functions,
- rapid functional verification, and modification,
- relatively inexpensive PC-based development tools,
- and low non-recurring engineering costs per device.

Once a design has matured, however, and moves into production, economic concerns take precedence. The cost per gate for FPL devices is typically an order of magnitude greater than for an ASIC. Conversely, an ASIC vendor's non-recurring engineering (NRE) charges for ASIC manufacturing are typically two orders of magnitude greater than the cost of programming an FPL device. If the figures in Table 1 represent typical values for NRE and unit cost per device, then the point at which the lines cross in Figure 1. represents an estimate of the minimum production quantity which justifies FPL migration to an ASIC. Where production runs are expected to exceed this minimum quantity, long-term cost savings may be achieved by integrating one or more programmable devices into a single ASIC.

	NRE	UNIT COST
FPL	\$ 100	20.00
Gate Array	\$ 10,000	2.00

Table 1. Typical Alternative Costs

TOTAL COST

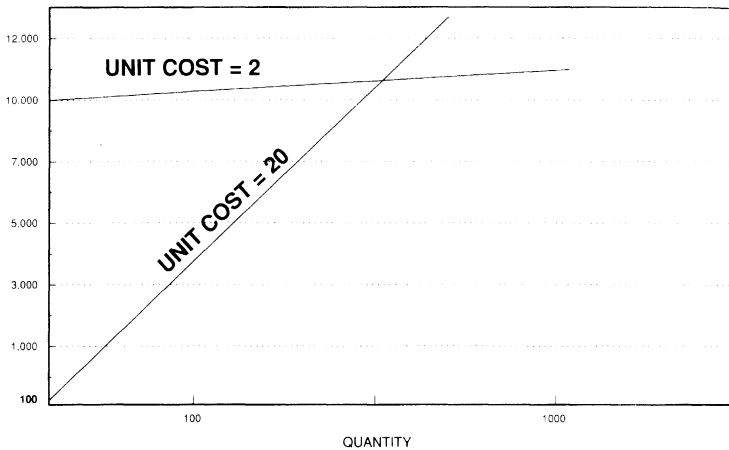


Figure 1. Minimum Production Quantity for Migration

BASIC REQUIREMENTS OF MIGRATION METHODOLOGY

An automated migration methodology must perform four basic functions. First, it must correctly interpret the functionality of the FPL device. This implies that the FPL netlist must be in a format which is understood by the translation software. Because FPL netlist formats are technology specific and non-standard, a general migration methodology requires some pre-processing to convert the netlist to a standard description language. Whatever language is used to describe the function, the FPL netlist should reference the FPL library components instantiated in the design by name. The translation software uses these names to search a reference library which describes the function, ports, and timing of these FPL components in order to correctly interpret the function of the entire design.

Second, an automated migration methodology must correctly translate the FPL function into the target ASIC technology. The translation software uses a target library, which describes the function, ports, and timing of the ASIC library components, to facilitate the mapping of functions from FPL to ASIC devices. The target library should contain components which are compatible with the FPL library in order to achieve efficient translation. For example, if an FPL netlist includes a JK flip-flop with both preset and clear, the ASIC library should have a similar flip-flop. Also, the ASIC library should have a scan cell equivalent function for each flip-flop in the library in order to properly support automatic scan chain insertion for test. Finally, the ASIC library should offer I/O buffers which meet or exceed the electrical specifications of the FPL I/O buffers.

Third, an automated migration methodology must output the translated function in an ASIC vendor's netlist format for the purpose of performing ASIC layout and timing simulations. Although the logical functions of the FPL and ASIC devices may be equivalent, technology differences and design techniques might result in different behavior or timing. Prior to migration, an FPL designer should assess whether or not the ASIC technology offers performance equal to

or better than the FPL technology. After migration, an FPL designer should perform post-layout simulations with back annotation in order to complete successful development of the ASIC.

Finally, an automated migration methodology must develop a test program for the purpose of manufacturing verification and control. If the migration flow can automatically insert test circuitry and develop test vectors with high fault coverage, then the FPL designer is relieved of the need to develop a comprehensive set of test vectors which mimic the function of the device. If critical path or output timing verification is required for manufacturing test, the FPL designer may need to generate some additional vectors using traditional ASIC development tools, and add these to the test program.

GOOD DIGITAL DESIGN PRINCIPLES APPLY

The flexibility and low iteration costs of FPL devices tend to encourage design practices which are unsuitable for automated, high-speed device testing in a manufacturing environment. Designers who are planning to migrate FPL devices to ASICs should therefore re-examine their designs with manufacturing testability in mind. The testability of a design is largely dependent on the application of a few fundamental principles of good digital design.

The basic tenet which must be satisfied in order to make any digital design testable is that all internal nodes must be controllable from the external input pins and observable at the external output pins. The insertion of an internal scan chain for test requires that the following rules be observed in order to achieve a high degree of controllability and observability.

First, the design should be completely synchronous. The device clock or clocks should be neither divided nor combined with other signals and then used as the clock or data inputs to other registers or latches. Asynchronous paths, combinatorial feedback loops, and ripple counters should be replaced with synchronous circuits.

Second, all asynchronous preset and clear signals for registers and latches should be directly controllable from external pins. Asynchronous register signals generated from other registers make internal nodes uncontrollable and thus make designs less testable.

Finally, in order to implement a scan chain successfully for ATPG, all registers and latches should be sensitive to the same edge or level of the clock. Conforming to this rule enables the scan data to propagate reliably from scan register to scan register.

The testability of a digital design is measured in terms of fault coverage. A fault is a node which cannot be changed from a high (stuck-at-1) or a low (stuck-at-0) state for a given input pattern. Fault coverage is the percentage of observable faults over all observable and unobservable faults. Following the above rules helps to maximize the fault coverage, and thus, the quality of a manufacturing test vector set developed by an ATPG program.

The following examples illustrate the benefits of practicing synchronous design techniques. Figures 2. and 3. represent an asynchronous and synchronous 3-bit counter respectively. Both circuits implement the function described in Table 2. The asynchronous counter is relatively easy to design and compact. The synchronous counter requires more effort to design and more gates to implement.

Present State			CLK	Next State		
QC	QB	QA		QC	QB	QA
0	0	0	^	0	0	1
0	0	1	^	0	1	0
0	1	0	^	0	1	1
0	1	1	^	1	0	0
1	0	0	^	1	0	1
1	0	1	^	1	1	0
1	1	0	^	1	1	1
1	1	1	^	0	0	0

Note: The ^ symbol indicates a positive edge clock.

Table 2. 3-bit counter state table.

3-BIT COUNTERS BEFORE SCAN

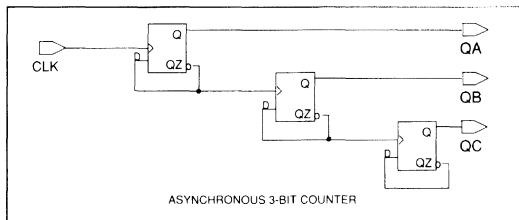


Figure 2. Asynchronous 3-bit counter

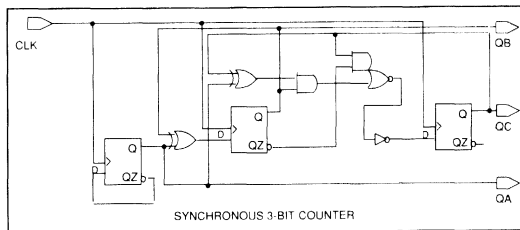


Figure 3. Synchronous 3-bit counter

Figures 4. and 5. represent the same circuits after a scan chain has been inserted. In the asynchronous 3-bit counter in Figure 4., the CLK signal goes to only one flip-flop in the circuit. Therefore, only the first flip-flop may be replaced with a scan flip-flop. If this circuit were imbedded inside a design, all of the nodes associated with the non-scan flip flops would be neither controllable nor observable by a manufacturing test program. In addition, the timing of any asynchronous speed paths from this circuit to an output pin is likely to be different between the FPL version and the ASIC equivalent.

In the synchronous 3-bit counter in figure 5., however, all flip-flops are replaced with scan flip-flops to construct a serial scan chain. All nodes in this circuit are controllable and observable in test mode as the scan vectors propagate from the test_si pin to the QC pin. Table 3. summarizes the fault coverage resulting from these scan vectors for both the asynchronous and synchronous designs. Not only is the fault coverage superior for the synchronous design, but also the problem of timing differences between the FPL and ASIC versions of the circuit due to asynchronous speed paths is eliminated. As long as the critical path is as fast or faster in the ASIC than in the FPL device, then the behavior of the ASIC is predictably the same as the FPL device.

3-bit counter	Faults Observed	Faults Unobserved	Faults Total	Fault Coverage
Async.	8	20	28	29%
Sync.	52	0	52	100%

Table 3. Fault report for counters

3-BIT COUNTERS AFTER SCAN WITH FAULT COVERAGE RESULTS

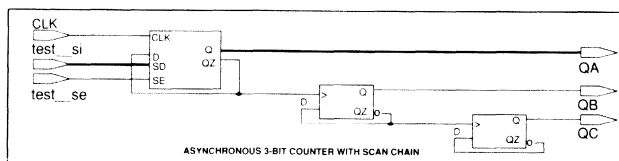


Figure 4. Asynchronous 3-bit counter with scan chain

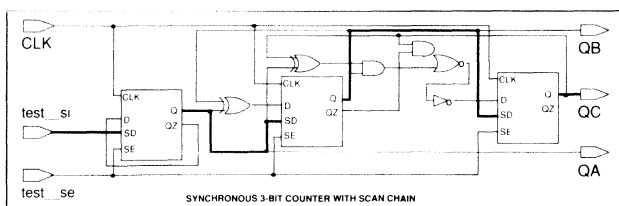


Figure 5. Synchronous 3-bit counter with scan chain

FPGA MIGRATION TO ASIC

Figure 6. illustrates an FPGA to ASIC flow which has been established at Texas Instruments. Phase I represents the technology translation and test vector generation steps of the migration flow. Phase II represents typical methodologies for layout, manufacturing, and testing an ASIC device. One of the driving concepts behind the development of this migration flow is that nothing in phase II is modified in order to complete a successful migration.

Originally, the flow was established to support the migration of Texas Instruments and Actel FPGA's. After additional development, interfaces to support migration of Xilinx X2000, X3000, and X4000 FPGA's, as well as Altera EP and EPM series EPLD's, have been added. Both a 1.2 um gate array library and a 1.0 um standard cell library are available as target ASIC technologies.

There are two inputs to the migration flow. The first input is the FPGA netlist, either an .ADL netlist for TI/Actel FPGA's, or an .XNF netlist for Xilinx FPGA's, or a .RPT file for Altera EPLD's. These netlists may come from any of the FPGA vendor supported platforms. The second input is simply a text file in electronic form, filled in by the FPGA designer, which specifies the pinout, the I/O buffer selection, and the electrical or timing characteristics of the final ASIC.

Prior to technology translation, the FPGA netlist itself is translated to a standard hardware description language, preserving all of the primitive cell calls and interconnectivity of the original FPGA. This standard HDL description is then read and translated into the equivalent ASIC function. The translation software automatically compares the ASIC function to the FPGA function to verify that the translation was performed correctly. The translation software also has the ability to optimize the ASIC design for area and speed.

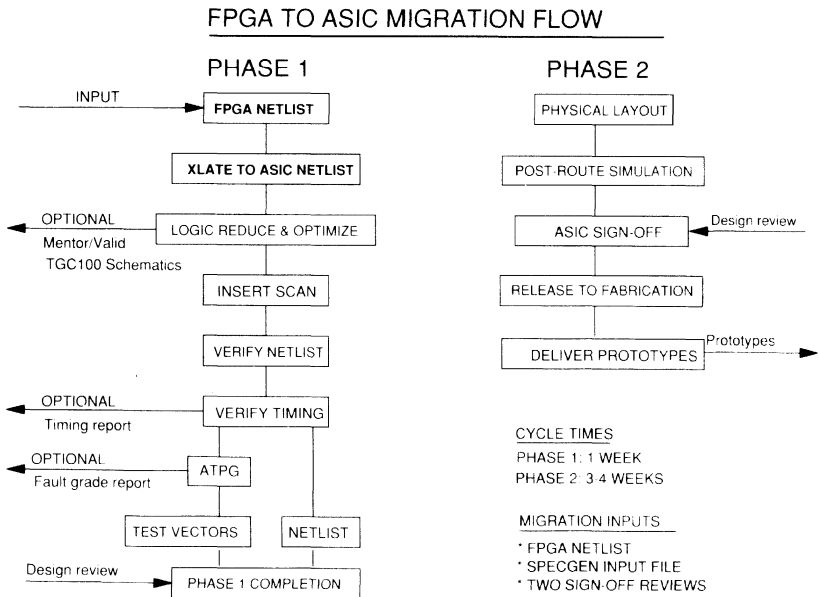


Figure 6. FPGA to ASIC Migration Flow

At this point the translation software can produce Mentor or Valid workstation schematics in order to continue the ASIC development by traditional methods of simulation and test vector development. However, another program in the migration flow can recompile the ASIC design and automatically insert an internal scan chain and develop scan test vectors, resulting in significant productivity enhancement. The scan chain is constructed by inserting a multiplexer at the data input of existing flip-flops in the design. A global scan enable signal drives all the multiplexer enable inputs, and the scan data propagates serially from the output of one scan cell flip-flop to the scan data input of the next. In addition to the multiplexed flip-flop scan methodology, this software is capable of implementing other scan methodologies, which are under development for FPGA migration.

With the scan chain in place, this program then automatically develops test vectors, compacts the vectors in order to minimize test time per device, and generates a fault coverage report. If the design conforms to good digital design practices as described above, then this program produces high quality test vectors with fault coverage typically greater than 90%. In this case the resulting test program is generally suitable for manufacturing test, and no other test vector development is required unless additional vectors are needed to test critical speed paths. If the fault coverage is low, this program generates a report which identifies all untestable nodes, and thus enables the FPGA designer to modify the logic to improve the fault coverage.

At the completion of phase I, the new ASIC circuit description and test vectors are written in Texas Instruments' internal standard formats, and then sent on for prototype layout, test, and manufacturing. The execution on phase I is accomplished entirely with specialized software which greatly improves productivity. Although actual execution time for phase 1 is design dependent, typical times range from 2 to 5 man-days with no design iterations. This compares favorably to migration methods employing traditional ASIC tools for schematic capture, simulation and test vector development, which typically require on the order of 2-5 man-months.

CONCLUSION

This paper describes an automated methodology to migrate mature FPL designs to an ASIC technology. An automated FPL migration path to ASIC technology addresses both the design manager's need for reduced design cycles, and the production manager's need for reduced long-run production costs. This methodology performs the four basic functions required for successful migration: correct functional interpretation, correct functional translation, ASIC netlist generation for layout and simulation, and ATPG. In order to produce a high-quality manufacturing test vector set with ATPG, and to ensure that the ASIC behaves the same as the FPL device, FPL designers should employ good, fundamental digital design practices in their original circuits. This methodology supports TI/Actel FPGA migration. Additional interfaces to support Xilinx FPGA and Altera EPLD migration have been developed.



SECTION 6.2

TEXAS INSTRUMENTS FPGA TO ASIC MIGRATION OPTIONS

Dung Tu
Texas Instruments, Germany
Technical Marketing

INTRODUCTION

Due to the increasing time-to-market pressure, you might wish to start a new design using a Field Programmable Gate Array for prototyping or for the first Production run and later on when the product proves successful change to an ASIC for volume production. The expected benefits from using the ASIC are lower chip price, no need for programming, tested devices, performance improvement and ultimately higher density by integrating several FPGA's to one ASIC. This application note discusses the various options for the migration FPGA to ASIC and describes in details the approach from Texas Instruments.

MIGRATION OPTIONS

There are several options today to replace an FPGA with a customized product: hardwired FPGA's, pre-programmed FPGA's and migration to ASIC.

Hardwired FPGA's typically provide a chip price reduction by a factor of 2 to 4 which may be good enough for small FPGA's but is clearly insufficient for the much more expensive larger FPGA's. Another problem is the possibility of a long lead time as a new mask set has to be produced. Looking to experience in the past, the equivalent hardwired version of PAL's - the HAL's - have not been successful.

In fact, pre-programmed PAL's - the approach from Texas Instruments to provide customers with PAL's, which are programmed according to the customer specification and tested in the factory - are a big success. Similarly, pre-programmed FPGA's can become an attractive solution for small volume. Post-programming testing is a big benefit because it improves the quality of the final devices by an order of magnitude.

For high volume, the migration to ASIC, i.e. the conversion of one FPGA to one pin-compatible gate array or the integration of several FPGA's to one gate array/standard cell is clearly the solution which offers the most from the benefits mentioned before.

There are basically three approaches to do a migration: schematic translation, netlist translation and logic synthesis using a high-level hardware description language.

At the first glance, schematic translation seems to be the most straight forward, intuitive way. Each component of the original schematic is replaced by a new component from the new technology library. This can be done manually or by software. However, because FPGA architecture is different from ASIC, a one-to-one replacement does not allow for an efficient ASIC.

The netlist translation approach maps an FPGA netlist to an ASIC netlist. The new netlist can be optimized for the ASIC.

The logic synthesis option allows a technology independant design. The design can be described using a high-level description language and mapped later on to the required silicon technologies

using the appropriate libraries. From all the possible approaches, this option is the most universal and provides not only a migration from FPGA to ASIC but also the possibility to migrate an ASIC to an FPGA.

NETLIST TRANSLATION

Texas Instruments support both the netlist translation and logic synthesis flow as shown in figure 1.

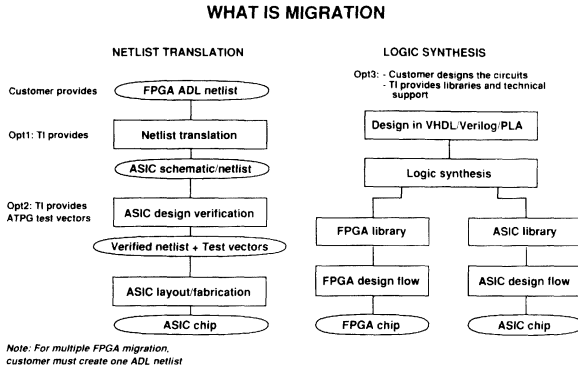


Figure 1. Texas Instruments FPGA to ASIC migration

The netlist translation flow is also discussed in the previous paper by Jim O'Connor. Figure 2. shows the service options which are available from Texas Instruments in Europe.

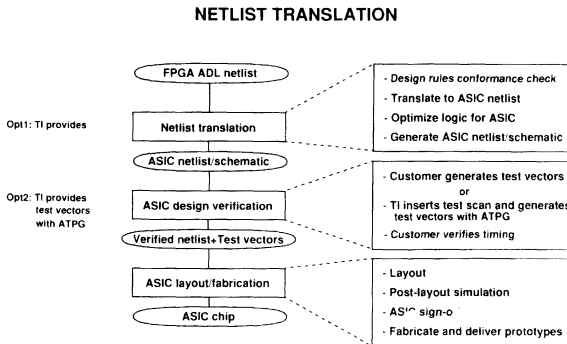


Figure 2. Netlist translation flow

Starting from an FPGA netlist in ADL format, the first step in the design flow is to check whether the netlist conforms to several design rules to ensure correct syntax, good testability and correct pinout for the ASIC.

The FPGA netlist is then translated and optimized to an ASIC netlist using TI software and Synopsys' logic synthesis tool. The ASIC can be TI CMOS gate array, TGC100 or TI standard cell, TSC700. Service option 1 provides purely a netlist translation. In addition to the netlist translation, option 2 provides test vectors for the ASIC which are generated by an automatic test pattern generator (Synopsys' Test Compiler).

ATPG requires good design-for-test practice. It releases you, however, from the most time consuming and least understood part of the ASIC design process. Manual test vector generation could consume up to 40% of the total design time for an ASIC project.

For combinational circuits, test vectors with 100% test coverage are easy to develop. Testing sequential circuits is much more complicated. Without the use of a test scan technique, a sequential circuit as shown in figure 3. is difficult to test because the present state of the sequential cells depends on their previous state. Scan design means replacing the sequential elements with a modified, scanable version that also performs a serial shift function.

There are several types of scan cell: multiplexed flipflop, clocked scan and level sensitive scan design (LSSD). Figure 3. illustrates the use of multiplexed flipflops as scan cells. The circuit has two additional inputs, Scan_enable, Scan_in, and one additional output, Scan_out. Depending on the Scan_enable input, the multiplexed flipflops work either in the normal mode or the test mode. When in the test mode, the flipflops are connected by the scan path to a shift register. The flipflops can be tested by shifting a binary pattern into Scan_in and monitoring the output at Scan_out.

Another popular scan cell is a master-slave flipflop - a double latch LSSD - which is included as hardmacro in the libraries for TI gate arrays and standard cells.

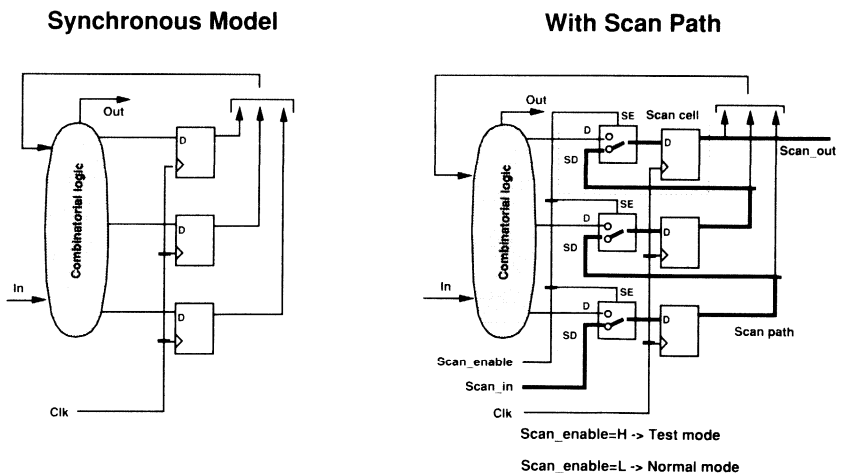
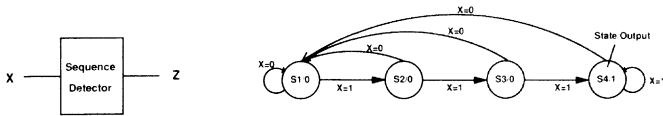


Figure 3. Scan cell, scan path

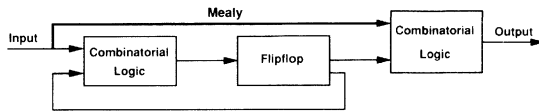
MIGRATION EXAMPLE

Figure 4. shows a simple state machine design example, a sequence detector. This circuit monitors an input signal X and outputs a signal Z=1 when X=1 for three successive clock cycles.

SEQUENCE DETECTOR STATE DIAGRAM



- The sequence detector monitors an input signal X and outputs a signal Z=1 when X=1 for three successive clock cycles.



- The state machine is implemented as a Mealy machine, eg. the output depends not only on the state but also on the inputs.

Figure 4. Sequence detector state diagram

Figure 5. shows the FPGA schematic with the back-annotation simulation using Viewlogic. Figure 6. depicts the ADL netlist.

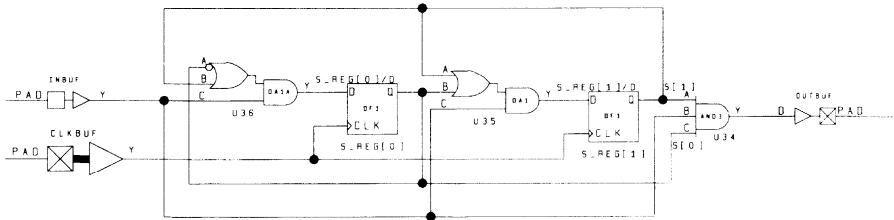


Figure 5. Sequence detector FPGA schematic

```

DEF MEASEQ; CLK, X, Z.
USE ADLIB:DF1; S_REG[1].
USE ADLIB:AND3; U34.
USE ADLIB:DF1; S_REG[0].
USE ADLIB:OA1; U35.
USE ADLIB:OA1A; U36.
USE ADLIB:CLKBUF; $1117.
USE ADLIB:OUTBUF; $1118.
USE ADLIB:INBUF; $1116.
NET CLK; CLK, $1117:PAD.
NET X; X, $1116:PAD.
NET Z; Z, $1118:PAD.
NET $1N10; U34:B, U36:C, U35:C, $1116:Y.
NET $1N11; S_REG[0]:CLK, S_REG[1]:CLK, $1117:Y.
NET $1N28; $1118:D, U34:Y.
NET S0; U36:A, U35:B, U34:C, S_REG[0]:Q.
NET S1; U34:A, U36:B, U35:A, S_REG[1]:Q.
NET S_REG0/D; S_REG[0]:D, U36:Y.
NET S_REG1/D; S_REG[1]:D, U35:Y.
END.

```

Figure 6. ADL netlist

After the netlist translation, a schematic for the same design using TI gate array TGC100 can be generated (figure 7). This circuit does not contain test scan.

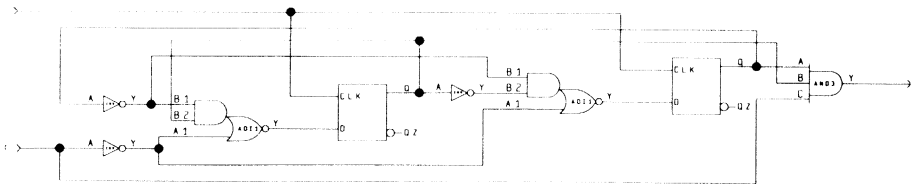


Figure 7. TGC100 schematic without test scan

Figure 8. shows the same circuit with test scan. The D-flipflops are now replaced by the multiplexed flipflops.

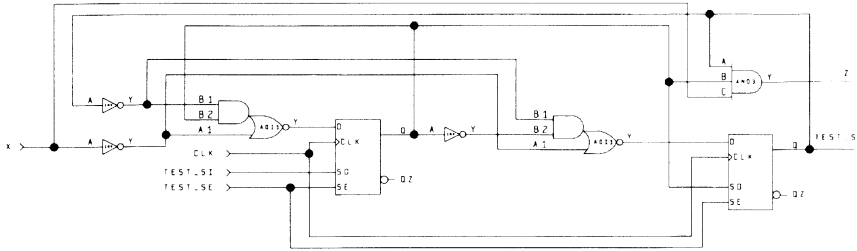


Figure 8. TGC100 schematic with test scan

Figure 9. shows the test vectors in TI Test Description Language (TDL) format. The newest version of the Synopsys Test Compiler has an option to generate the TDL format.

```

($ DATE 9/18/1991 12:43 *) SETR P:=T'YLYMYMY':($ 0')
($ TESTER: ACTV SETR P:=T'HCHMHMYL':($ 1')
($ TEST TYPE: FUNCTIONAL LIBRARY/TIMING: TGC100/COMMERCIAL *) SETR P:=T'HCL1HMVL':($ 2')
($ *) SETR P:=T'HCYOHMYL':($ 3')
($ *) SETR P:=T'HYMHMYL':($ 4')
CONNECT P_VAR=(SCANEN,CLK,SCANIN,SCANOUT,X,Z,CLR,TESTB), SETR P:=T'HCHMHMYL':($ 5')
DEFFIN=(IN 3,OUT,IN,OUT,IN 2); SETR P:=T'HCHMHMYL':($ 6')
($ PERIOD = 1000/0; *) SETR P:=T'LLMH1HL':($ 7')
CLOCK VAR=CLK; SETR P:=T'LCH1MHML':($ 8')
PATTERN=010, HOLD0= 300, HOLD1= 300; SETR P:=T'HCH1MHML':($ 9')
DELAY VAR = TESTB, SETR P:=T'HCLMHMYL':($ 10')
OFFSET = 0; SETR P:=T'LLMH0HL':($ 11')
DELAY VAR = CLR, SETR P:=T'LCL1MHML':($ 12')
OFFSET = 0; SETR P:=T'HCL1MHML':($ 13')
DELAY VAR = X, SETR P:=T'HCLMHMYL':($ 14')
OFFSET = 0; SETR P:=T'LLMH0HL':($ 15')
DELAY VAR = SCANIN, SETR P:=T'LCH0MHML':($ 16')
OFFSET = 0; SETR P:=T'HCH1MHML':($ 17')
DELAY VAR = SCANEN, SETR P:=T'HCHMHMYL':($ 18')
OFFSET = 0; SETR P:=T'LLMHLOHL':($ 19')
STROBE VAR = Z, SETR P:=T'LCHLOMHML':($ 20')
OFFSET = 900; SETR P:=T'HCLLOMHML':($ 21')
STROBE VAR = SCANOUT, SETR P:=T'HCHMHMYL':($ 22')
OFFSET = 900; SETR P:=T'LLMH0HL':($ 23')
($ *) SETR P:=T'LCH1MHML':($ 24')
($ SCSSXZCT*) SETR P:=T'HCYOHMYL':($ 25')
($ CLCC LE*) END;
($ AKAA RS*) ($ The following input(s) did not transition to L: *)
($ N NN T*) ($ CLR *)
($ E IO B*) ($ The following input(s) did not transition to H: *)
($ N NU *) ($ CLK *)
($ T *) ($ TESTB *)

```

26 test vectors are needed to test 8 gates!

Figure 9. TI Test Description Language Format

LOGIC SYNTHESIS MIGRATION FLOW

Logic synthesis allows implicitly the migration because it is a technology independent design methodology. Fig 10 shows the migration flow using logic synthesis.

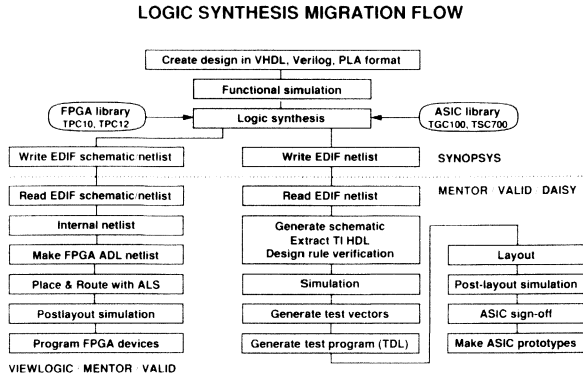


Figure 10. Logic synthesis migration flow

The design is described in a high-level language such as VHDL or Cadence Verilog. The functionality of the design is verified using a VHDL simulator or the Verilog simulator. The design is synthesized with a logic synthesis tool such as the Synopsys Design Compiler. Depending on the required technologies, either an FPGA library or an ASIC library is used for the synthesis. The result is written out as an EDIF netlist or an EDIF schematic.

An EDIF interface is required to convert the EDIF netlist to the right format for the design platform which is used to develop the FPGA or the ASIC. To make an FPGA you use the 'makeadl' utility which is included in the FPGA development kit to convert the internal netlist to an ADL netlist. This netlist is used by the Action Logic System to place & route the design. After a successful post-layout simulation you can program your FPGA chips. Similarly, if you want to make an ASIC you use the EDIF netlist as the starting point for the ASIC design flow.

Figure 11. depicts the VHDL code for the sequence detector and the script file for the Synopsys Design Compiler. To make an FPGA, you specify FPGA libraries as the link library, target library and symbol library. To make a gate array you specify the TGC100 library. If you want to make a standard cell you specify the TSC700 library. Here is the key benefit of the logic synthesis flow. The design source code remains the same, the logic synthesis tool takes care of the implementation on the gate level. If you compare the gate array schematic for the sequence detector (Figure 7.) with the FPGA schematic (Figure 5.) you can see that the TGC100 hardmacros are not exactly the same as the FPGA softmacros. Logic synthesis releases you from these details.

However, the FPGA to ASIC migration is not exactly the same as the reverse path: the ASIC to FPGA migration. When you design an ASIC you normally include scan cells and scan path in the source code. If you use the same code for the FPGA implementation the result will have a big overhead due to the test cells. A large ASIC design may require several FPGA's so the design must be partitioned. Recommendations how this can be done are best discussed in another paper.

VHDL SOURCE / SCRIPT FILES

VHDL source file

```
entity MEASEQ is
  port (X, CLK : in BIT;
        Z : out BIT);
end;
architecture BEHAVIOR of MEASEQ is
  type STATE_TYPE is (S1, S2, S3, S4);
  signal S, SNEXT: STATE_TYPE;
begin
  COMBIN: process
  begin
    case S is
      when S1 =>
        if X = '1' then Z <= '0'; SNEXT <= S2;
        else Z <= '0'; SNEXT <= S1;
        end if;
      when S2 =>
        if X = '1' then Z <= '0'; SNEXT <= S3;
        else Z <= '0'; SNEXT <= S1;
        end if;
      when S3 =>
        if X = '1' then Z <= '0'; SNEXT <= S4;
        else Z <= '0'; SNEXT <= S1;
        end if;
      when S4 =>
        if X = '1' then Z <= '1'; SNEXT <= S4;
        else Z <= '0'; SNEXT <= S1;
        end if;
    end case;
  end process;
  SYNCH: process
  begin
    wait until CLK'event and CLK = '1';
    S <= SNEXT;
  end process;
end BEHAVIOR;
```

Script file

```
/** Design check **
designer   = "Dung Tu ";
company   = "Texas Instruments";
sea:ch_path = (.win1 lib ti fpga )
link_library = tpc10.db
target_library = tpc10.sdb
symbol_library = tpc10.sdb
read -f vhdl measeq.vhdl
current_design = MEASEQ
write -h -f db -hierarchy -o measeq.db
check design > chkdns.rpt
report -design > measeq.rpt

/** Constraint: Optimize for area**
max_area 0.0
compile
write -h -f db -o measeq_opt.db
report -area -cell -timing > measeq.rpt
free -all

/** Write EDIF schematic for export to Valid, Mentor or Viewlogic **
read -f db measeq_opt.db
current_design = MEASEQ
create_schematic_hierarchy -size B
write -h -f edif -o measeq.edif
exit
```

To map to ASIC, use
gate array or standard
cell libraries

Figure 11. VHDL code and script file

SUMMARY

As a supplier for both FPGA and ASIC, Texas Instruments support the migration between these two technologies. Both netlist translation and logic synthesis are supported. The netlist translation allows the conversion of an existing FPGA design to an ASIC. Logic synthesis is however clearly the design flow of choice for new designs if they need to be implemented in different technologies.

SECTION 6.3

MIGRATION DESIGN RULES FOR TESTABILITY

Patrick D. Eyres
TEXAS INSTRUMENTS, DALLAS

FPGA's provide an excellent tool to test new design concepts or to get a product to market quickly. However, one of the principle drawbacks is cost. The ability to quickly and easily migrate to a low cost gate array would increase the attractiveness of both products. To be a viable option, this migration should require as little effort from the design engineer as possible. One of the areas that often takes the most engineering time is test vector generation. Automatic Test Pattern Generation (ATPG) is one way to reduce this time. Often, however, FPGA designs include glue logic from different areas of a PC board where testing is difficult. If a few basic design for testability rules are followed, most of these problems can be overcome.

To better understand these rules you must start with the migration flow itself. A diagram of the basic elements of the migration flow used by Texas Instruments, Inc (TI), is shown in Figure 1. A netlist is received from the customer and translated into a standard Verilog format. This netlist is then mapped or translated into the target ASIC library. Next, the design can be optimized for area or timing. After optimization, the circuit is checked for test rule violations. Finally scan flip-flops are inserted and connected into a scan chain. The next step in the flow is to generate test vectors and a fault coverage report. These vectors are translated into TI's Test Description Language (TDL) to be used for production test. Once the netlist is finished, either a Mentor or Valid schematic database can be generated and handed off to the customer for system timing verification.

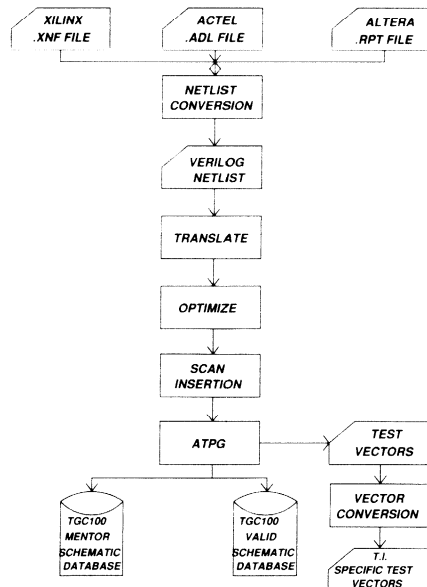


Figure 1

TI has chosen a multiplexed flip-flop scan methodology for scan implementation. This methodology simply replaces each flip-flop in the scan chain with an equivalent flip-flop and multiplexor. When the scan chain is connected the q/scan_out of one flop is routed to the scan_in of the next. The scan_enable is routed globally to all flip-flops in the scan chain. A list of advantages and disadvantages of this methodology is shown below.

ADVANTAGES

- A minimum of one additional signal pin (scan_enable) must be added. The addition of scan_in and scan_out pins may not be necessary if they can be multiplexed with design signals.
- Low area overhead. The multiplexed flip-flop is normally no more than 30% larger than the basic flip-flop. The only signal that must be globally routed is scan_enable.
- This methodology can be easily implemented using virtually any library containing a flip-flop and a multiplexor.

DISADVANTAGES

- The addition of a multiplexor in front of all flip-flops will introduce an additional setup time.
- This methodology is not well suited for designs containing latches. Latches cannot be included in the scan chain and can only be tested if made transparent.

To achieve high fault coverage with this methodology, all flip-flops must be included in the scan chain. If a flip-flop is not in the scan chain it is considered a black box by the ATPG software. None of the nodes between it and any other flip-flops will be tested. Including all the flip-flops in the scan chain, however, is not always possible. The ATPG software imposes various design rules on the designer. These rules must be understood and followed. A definite advantage can be gained by understanding and following the rules during the design of the FPGA.

A common technique will be used to correct most of the problems. The ATPG software allows the use of a Test Mode Select (TMS) pin. This pin can be asserted during vector generation, causing the circuit to go into test mode. In test mode, the design contains fewer rule violations and more flip-flops are included in the scan chain.

LATCHES

The multiplexed flip-flop methodology does not allow latches to be included in the scan chain. The preferred solution is to replace all latches with flip-flops. There are a couple of alternatives if this is not possible. The basic objective of any design change dealing with latches is to allow the enable pin to be controlled from a top level signal pin. If this is accomplished, the ATPG software will make the latches transparent during testing. Otherwise, circuitry such as is shown in Figure 2. can be added. In this example SIG1 is the normal enable for the latch and TMS is the Test Mode Select pin. During test TMS will be held to a logical 1 and the latch will be transparent. The fault coverage on the latch itself will still be relatively poor. When made transparent several of the nodes will not be toggled.

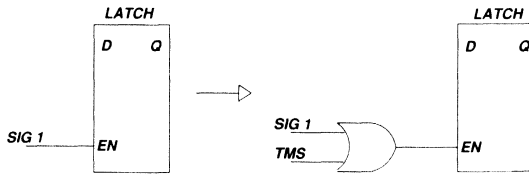


Figure 2

GATED CLOCKS

Gated clocks are rarely good design practice. They should be removed from the circuit if at all possible. Figure 3. shows a simple example of a gated clock and one possible solution. The clock of any flip-flop in the scan chain must be fully controllable from the top level signal pins. In the example, if SIG1 is generated internally CLK may not reach DFF during scan. Therefore, DFF would not be included in the scan chain. In the second case TMS is held to a logical 1 and DFF will be clocked. Any time test circuitry is added potential faults are also added. The OR-gate and one half of the AND-gate would not be tested. Our goal is to get the flip-flop added to the scan chain. If this is done, the vectors will catch faults between this flip-flop and any others.

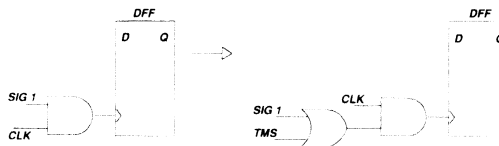


Figure 3.

CLOCK SIGNALS USED FOR DATA

Another violation is illustrated in Figure 4. This circuit shows a clock going to the data input of a flip-flop. This is poor design practice due to possible setup violations. The problem with which scan is concerned occurs if the design has two independent clocks. The ATPG software will hook all flip-flops, regardless of which clock they use, into one scan chain. All clocks in the design are then clocked together during test. If one clock goes to the data input of a flip-flop that is clocked by another, a setup violation could result. Once again the TMS pin can be used to insure the CLK signal does not get to the data pin during test mode.

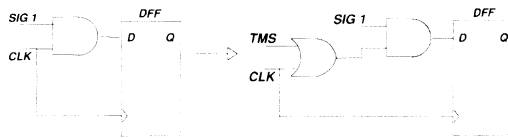


Figure 4

USING BOTH EDGES OF A CLOCK

The ATPG software expects to see all flip-flops clocked from the same edge of the clock. A problem occurs if a flip-flop clocked on the falling edge of the clock appears in the scan chain after one clocked on the rising edge. In this case, scan data is clocked into the two flip-flops in one clock cycle. The first on the rising edge. The second on the falling edge. If there are 3 flip-flops in the chain, the software expects to apply a pattern of values in three clocks. The falling edge flip-flop corrupts the chain and the software is currently not sophisticated enough to handle this problem. If you have two different clocks, you can clock one on the rising edge and one on the falling. However, for any one clock you can only use one edge. Figure 5 gives an example. Here, a multiplexor can be used in front of the flip-flop clocked on the falling edge. During test TMS selects the rising edge instead.

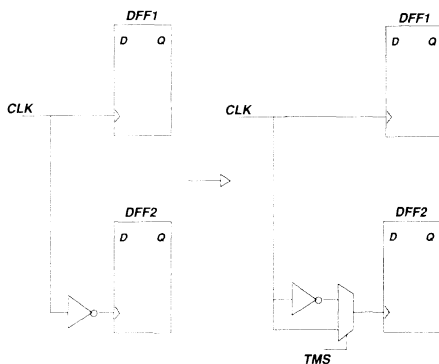


Figure 5

UNCONTROLLABLE ASYNCHRONOUS PINS

To put a flip-flop in the scan chain the software must be able to control the asynchronous signals on the flip-flops. A scan pattern should not clear or preset another flip-flop in the scan chain. The preferred design method would be to make all asynchronous pins controllable from a top level signal. An example of this is shown in Figure 6. If a logical 0 were scanned into DFF1 then DFF2 would be cleared breaking the scan chain. With the insertion of an OR-gate, the path is effectively cut during test mode.

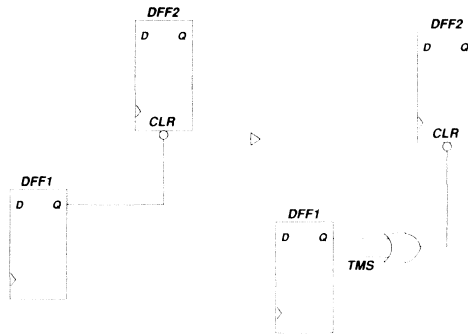


Figure 6

Q OUT USED TO CLOCK A FLIP-FLOP

The final violation is shown in Figure 7. If an internally generated signal is used to clock a flip-flop, a violation will occur. The clock of the second flip-flop cannot be controlled. It will therefore, not be included into the scan chain. All flip-flops should be clocked from a single system clock. A multiplexer can be added to the circuit as shown. During test mode the multiplexer will select the system clock. DFF2 will then be included in the scan chain.

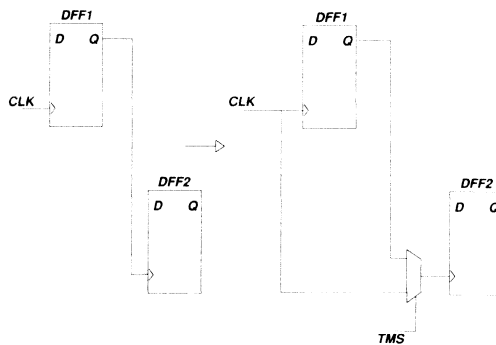


Figure 7

The techniques presented here are not the only ones that can be used. In each case you should first try to redesign any circuits that violate the ATPG rules. If this cannot be done, the TMS pin and other similar techniques can be effectively used. These techniques are more than good design practice. If all these design rules are followed, the design engineer need only hand off an FPGA netlist. The turn around time from hand off to gate array schematics can be as short as 6 days. The design will have all the needed production test vectors with no additional work needed from the design engineer.

SECTION 6.4

RECOMMENDATIONS FOR CONCURRENT DESIGN WITH FPGA AND ASIC USING A HARDWARE DESCRIPTION LANGUAGE

Peter Heusinger & Norbert Schuhmann
Fraunhofer Institute For Integrated Circuits

INTRODUCTION

Parallel design of ASIC's and FPGA's using hardware description languages and logic synthesis tools allows rapid prototyping, and the possibility of design verification in hardware, prior to the creation of masks, thus reducing the risks associated with ASIC design. While an ASIC is being produced, additional hardware and software could be developed and tested, e.g. microcontroller applications. The following paper looks at some recommendations for this design methodology.

DISADVANTAGES

The programmability and architecture of FPGA's implies some disadvantages compared to an ASIC:

- larger gate delays
- larger wire delays
- limited maximum system clock frequency
- limited gate count
- limited number of pads
- limited types of pad cells (slew rate, schmitt trigger, CMOS, TTL, driving strength)
- limited number of on chip interconnection wires (routing sources)
- limited number of sequential cells
- no oscillators on chip possible
- no analog modules on chip possible (mixed mode design)

Recommendations for parallel design:

- plan design management well
- design consistent (hierarchy, modules, signal names)
- combine specific modules of ASIC and FPGA in a common source code and select by conditional compiling (e.g. clock oscillators, NAND tree, test and scanpath cells, not necessary on FPGA's)
- choose common sequential cells on ASIC and FPGA libraries
- design fully synchronous
- use a common clock tree (use of built in clock distribution network on FPGA's is possible)
- avoid glitches and race conditions
- don't use gated clocks (hard to implement on FPGA's, waste routing resources on FPGA's)
- create all modules inside limits of FPGA's (gate count, number of pads)
- avoid bus intensive architectures (hard to route on FPGA's)
- create separate pad level containing only pad cells (ease partitioning of design)
- create separate interconnection level below pad level, containing only module instantiations and interconnections, but no primitive cells (ease partitioning of design).

RECOMMENDATIONS FOR DESIGN PARTITIONING

In the case, that the design doesn't fit into one FPGA, partitioning has to be done:

- combine pad and interconnection level to a new interconnection level
- distribute modules and related pads below the interconnection level to different FPGA's (watch for gate count and pad limits)
- insert interconnection pads for inter FPGA connections (watch for additional delays by interconnections and the interconnection pads)
- implement the master clock net as an interchip clock tree
- implement the internal clock by the built in distributed clock

CONCLUSION

This is a very brief look at some recommendations for concurrent fpga and asic design but there are other factors that could be considered in evaluating this methodology. In particular, there are many commercial advantages that have been pointed out in earlier papers. (ref.6.1 Migrating Programmable Logic To Asic and ref 6.2 TI FPGA to ASIC migration options)



**SECTION SEVEN
FPGA PROGRAMMING AND
TEST**

SECTION 7.1

PROGRAMMING FIELD PROGRAMMABLE GATE ARRAYS

Manufacturing Considerations and Options

Jerry Hughes
Texas Instruments, Dallas
Engineering, Factory Programmed Products

ABSTRACT

The use of Field Programmable Gate Arrays requires that all devices be programmed. Programming is accomplished either in-house or by outside sources. A clear understanding of the issues associated with in-house programming and available options is required prior to the decision to establish in-house programming capability. This article describes key manufacturing issues and programming options to aid in the decision making process.

Field Programmable Gate Arrays (FPGAs) provide designers with a flexible tool to reduce design cycle time, package counts, and nonrecurring engineering charges. The desk top design and programming capability allows designers to achieve the shortest time to market possible for new designs. However, the quick turn capability gained by using FPGAs creates unique manufacturing challenges, once designs move from prototype to production.

Since FPGA products must be programmed prior to use, a decision must be made to program in-house or utilize outside programming sources. This decision can only be made after careful examination of the manufacturing issues involved.

Issues that must be considered include:

- Programming time and throughput
- Maintaining surface mount component lead quality
- Handling moisture sensitive Plastic Surface Mount Components (PSMCs)
- Manufacturing efficiency

A clear understanding of these manufacturing issues is key in determining the viability of establishing an in-house programming operation.

Texas Instruments is acutely aware of the complexity in establishing an efficient programming process and offers factory programmed FPGAs (P-FPGAs) to help reduce our customer's in-house manufacturing requirements. By allowing TI to program your FPGAs, you are taking advantage of TI worldwide resources, industry leading manufacturing expertise, and quality standards. Devices are delivered fully programmed, symbolized and ready for immediate use. Consider the issues, and then consider TI P-FPGAs for your volume production requirements.

PROGRAMMING TIME AND THROUGHPUT

TI FPGA programming time is a function of the antifuse technology and fuse density. The antifuse requires multiple programming pulses to successfully form the electrical connection between logic modules. Depending on the device type, 3,000 to 20,000 antifuses are typically programmed. Figure 1 shows typical per unit programming time for TI FPGAs programmed on a TI Activator programmer.

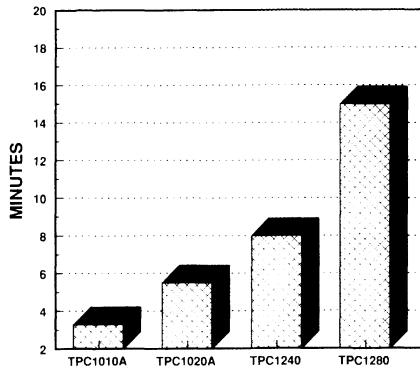


Figure 1. FPGA Per Unit Programming Time

Extended device programming time is not a critical issue for design verification and system prototyping. However, once a design is ready for production, longer cycle times may be incurred if an efficient programming operation is not implemented. Since programming cycle time increases proportionally to the quantity of units programmed, high volume production programming requirements will place a heavy burden on any programming operation.

Figure 2. illustrates programming cycle time as a function of the quantity of units programmed. The data shown is based on the use of one TI Activator 2 (gang 4) programmer.

As Figure 2. shows, programming cycle time becomes a significant factor as run rates reach 500 units. For example, the total programming time required to complete 500 units for each device type is:

- TPC1010A = 6.9 hours
- TPC1020A = 11.5 hours
- TPC1240 = 16.7 hours
- TPC1280 = 31.3 hours

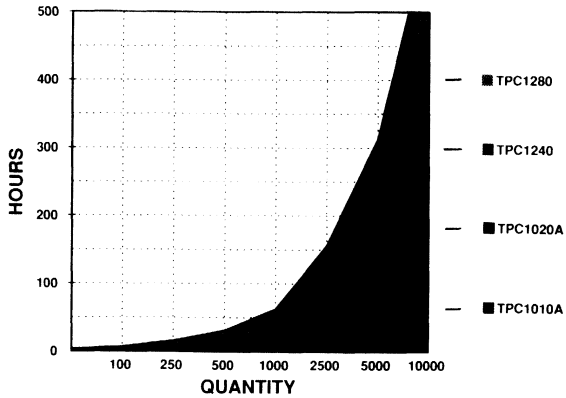


Figure 2. FPGA Programming Cycle Time

Serious consideration must be given to the effect of programming time on production scheduling and capacity. Since programming throughput is directly related to programming time, throughput will degrade as programming time increases. Therefore, a careful analysis of anticipated volume and device mix is needed to determine the amount of capacity required for an in-house programming operation. Based on a 7.3 hour shift and a 22 day month, Figure 3. shows the throughput for a one shift operation using a single Activator 2 Programmer.

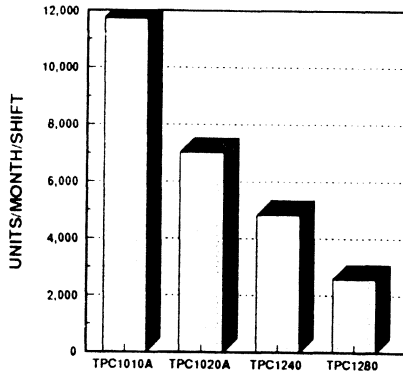


Figure 3. Activator 2 Programmer Throughput

SURFACE MOUNT COMPONENT LEAD QUALITY

The ability to maintain the lead quality of surface mount components is another key factor to consider before implementation of an in-house programming operation. Handling of surface mount components results in degradation to lead quality. The number of handling induced defects is dependent on the amount (number of process steps) and type (manual or automated) of handling. Based on TI's internal 68 and 84 pin PLCC manufacturing operation, Figure 4. illustrates the effect of device handling on lead quality. By comparison, TI factory programmed

FPGAs undergo extensive lead conditioning and inspection prior to shipment, and therefore result in significantly lower defect levels.

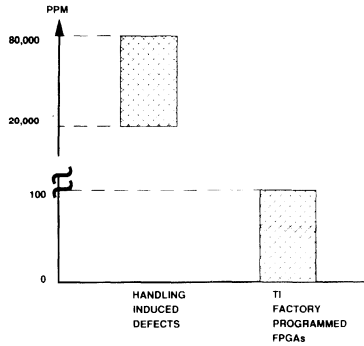


Figure 4. 68/84 Pin PLCC Defect Level—Leads

The significance of defect level to the board assembly process is shown in Figure 5. This graph indicates the probability, in percent, that at least one defective, out of spec, device will be found on a single board. The probability is a function of defect level and number of units per board. For example, the probability of one defective unit being found on a board with ten FPGAs is:

- 56% (1 in 1.8) at 80,000 ppm
- 18% (1 in 5.4) at 20,000 ppm
- .1% (1 in 1000) at 100 ppm

In contrast to the high probability of board assembly problems related to increased lead defects, using TI P-FPGAs virtually eliminates lead defects. As a result, lower cost of ownership and increased manufacturing efficiency is achieved.

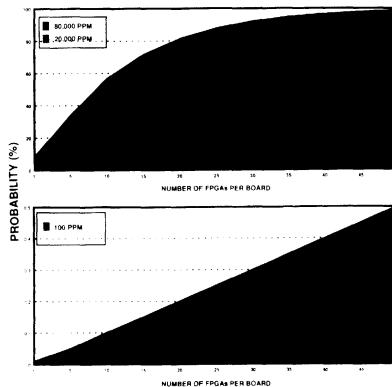


Figure 5. Defect Probability Vs. PPM Level

HANDLING MOISTURE SENSITIVE PACKAGES

A somewhat obscure, but equally important, manufacturing consideration is the required handling procedure for moisture sensitive Plastic Surface Mount Components (PSMCs). Moisture is absorbed from the atmosphere by the plastic used to encapsulate integrated circuits. Cracking of the package may occur when moisture trapped inside the plastic vaporizes during the solder reflow process. Therefore, special handling and packing procedures must be followed to prevent PSMC moisture absorption.

All TI plastic surface mountable FPGAs and P-FPGAs are baked to reduce the moisture content of the plastic package to less than or equal to 0.05% and shipped in heat sealed dry pack bags. Humidity indicator cards are sealed with the components to serve as a warning in the event of seal failure and exposure to moisture.

Moisture absorption begins immediately upon exposure of PSMCs to the environment. The amount and rate of absorption is a function of temperature and relative humidity. The maximum exposure time at 30 degrees C and 60% relative humidity is 48 hours after the bag is opened.

An in-house programming operation exposes FPGAs to the open air and must consider the exposure time limit. Production programming capacity, scheduling, and cycle time must be appropriately planned to prevent the addition of a rebake process step. In any case, bake and dry pack equipment is needed in the event maximum exposure time is exceeded or programmed devices are to be inventoried for later use.

MANUFACTURING EFFICIENCY

A decision to program FPGAs in-house results in an operation that, depending on quality and manufacturing requirements, becomes quickly complicated. In addition to programming, symbolization of programmed devices is needed to prevent different designs from becoming mixed. If pick and place equipment is to be used for PLCC packages, programmed FPGAs may need to be taped and reeled prior to use. Therefore, material must be routed to in-house tape and reel equipment or to an outside subcontractor. Regardless of the method chosen, increased cost and process complexity is the result.

A summary of the in-house programming issues follows:

- Processing requirements and capacity
- Programming equipment
- Symbolization equipment
- Handling procedures to minimize lead degradation
- Handling procedures to minimize rebake
- Handling procedures to minimize EOS and ESD damage
- Outgoing quality
- Staffing requirements
- Nonrecurring, capital, overhead, and labor costs

A decision to purchase TI factory programmed FPGAs requires consideration of the following issues:

- Cost adders
- Volume requirements
- Outgoing quality
- Leadtime

By moving programming, symbolization, and tape and reel back to TI, a substantial reduction in process steps, material handling, and defects is achieved. P-FPGAs are simply received and placed in inventory until needed for production. Figure 6 illustrates the reduction of in-house processing achieved by using TI P-FPGAs. Figure 7. shows the P-FPGA manufacturing flow implemented to provide the highest quality product possible.

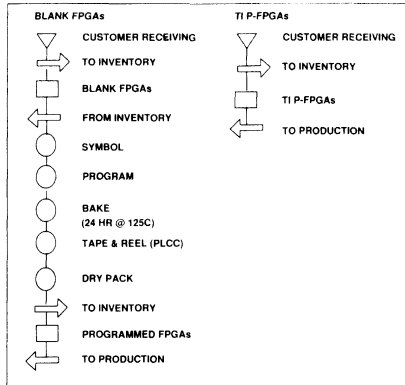


Figure 6. Blank vs. TI P-FPGA Process Steps

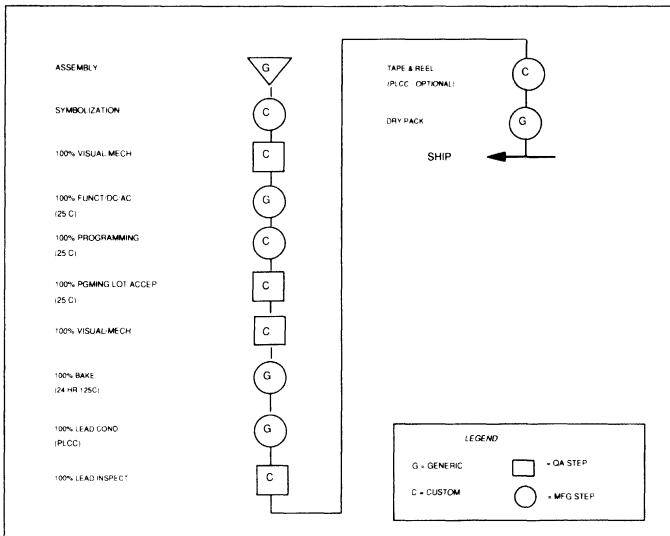


Figure 7. TI P-FPGA Factory Process Flow

SECTION 7.2

HOW TI TESTS FIELD PROGRAMMABLE GATE ARRAYS

Jim Ptasinski
Texas Instruments, Dallas
Engineering

INTRODUCTION

The 10xx and 12xx families of field-programmable gate array devices were designed with such a comprehensive and thorough set of testing circuitry, that it achieves a virtual 100% fault coverage of all the logic, prior to programming. As a result, this makes it unnecessary to functionally test the device after programming. This article will outline and describe the various design-for-testability techniques and how they are used during factory testing.

SERIAL-TEST CIRCUITRY

The internal test observability and controllability of the device is performed via a long serial shift-register chain that surrounds the array. Its length and partitioning is determined by the number of gates and the device type.

There is only one package pin, other than power and gnd, that is not programmable by you. The MODE pin is dedicated for the selection of either test/programming/debug when the pin is at a high-level, or the normal operation when it is at a low-level or gnd. The MODE pin must not be allowed to float and must be tied to gnd during normal operation.

The test data is serially-shifted into the SDI pin, one bit at a time, by clocking the DCLK pin. The subsequent test results are serially-shifted out at either the SDO pin or the MPR pins, (PRA, PRB) by also clocking the DCLK pin. Only the I/O input and output buffer tests are functionally tested at each I/O pin individually.

This shift-register is divided into several different groups to give access to the top, bottom, left, right, middle, and center of the array. The first block of registers is the mode control and once loaded, it determines the setup and sequence of events that will occur during that test. The other blocks provide each channel and column track with the ability of being forced or sensed in any combination or direction with Vpp, H, L, Z or a precharged condition. The new data or result can then be loaded back into the shift-registers, shifted-out, and read.

TEST FLOW

The factory testing flow used on the TI FPGA devices in the unprogrammed state is listed in the following table. The major test categories are shown, and each will be described separately.

- 1) pin-to-pin opens and shorts
- 2) static power-pin parametric current
- 3) serial-shift register chain functionality
- 4) channel and column tracks opens/shorts/leakage
- 5) array transistors functionality/leakage
- 6) logic modules addressing/functionality/microprobing
- 7) high-voltage stressing of antifuses and junctions
- 8) antifuses addressing and shorts tests
- 9) silicon-signature and bin-circuit programming
- 10) I/O buffers functionality/parametrics/tristate leakage

DEVICE PIN-TO-PIN OPENS/SHORTS TEST

The most basic of tests performed on any IC is the pin-to-pin opens and shorts test. This checks for assembly and handling defects and also ESD/EOS damage near the devices internal bonding pads.

The most common problems you may encounter in the field programming of FPGA's will likely be caused by mechanical damage to the package's pins or leads. The high-pin count and finer-pitch used on flat-packs and chip-carrier packages will require your close attention during socket insertion on the ACTIVATOR.

The other defects found by this test are caused by electro- static discharge into a device pin due to improper handling or from electrical over-stress of the device.

TESTING THE POWER PINS

Each of the power pins are parametrically measured for the amount of static and quiescent current the device uses, to meet the specified maximum value.

TESTING THE SHIFT-REGISTER CHAIN

The functional test of the serial-shift register chain is performed first, before any other functional tests can be valid. That is, until the shift-register is proven to pass all of its tests, no other addressing or read-back could be relied upon or even possible. Therefore this test must be 100% functional before attempting any further testing.

COLUMN AND CHANNEL TRACKS TESTING

The FPGA architecture is largely dominated with horizontal and vertical metallization that takes up a significant portion of the die area. This is one factor that gives your design a 95% gate utilization. All of these interconnection and routing tracks must be 100% tested for connectivity, opens, and every possible short first. The need for this should be fairly intuitive to you.

One of the test techniques used has been shown to be directly related to the high quality of each device. This is the ability to check for any possible leakage paths by precharging the lines. The test charges up each track, and after a predetermined time needed to maintain the level, the charge must still be high enough to allow a pass. This discharge rate therefore shows the integrity of the chip and gives a quality indication of a great deal of key process parameters.

ARRAY TRANSISTORS TESTING

The horizontal tracks are divided up in an optimum way with segmentation transistors. Each of these horizontal pass devices are in parallel with an 'H' fuse. Also the vertical tracks have many vertical pass transistors in series, along with a 'V' fuse in parallel. The gates of these transistors are turned on and off in accordance with the register sequence and the mode of test. All of these array transistors are therefore fully tested for correct switching.

You should be aware that the track testing and array testing are not entirely independent. Note how this test flow builds upon itself by using previously tested and known working paths and devices to check the next unknown level. The test vectors were designed to isolate and differentiate between many defects such as an open track or open transistor. This also applies to short and leakage problems in a track or a transistor.

LOGIC MODULE TESTING

Now we are ready to test the full functionality of each logic module available for your design. That is, we have so far made sure the entire routing, control, and test 'infrastructure' of the die is functional, and defect/leakage free.

The proven accessibility of each of the modules eight inputs and one output are now toggled thru all of its combinatorial or sequential (12xx only) truth tables and states, again via the serial-shift register.

The module microprobing and diagnostic capability is also checked at this time. This makes sure that any two module outputs throughout the array can be diagnosed by you in your board.

HIGH-VOLTAGE JUNCTION STRESS AND ANTIFUSE STRESS TESTING

Stress testing is now done in order to insure the highest possible antifuse reliability along with your field-programmable yield. High-level voltages are now used in the 12-21V range on Vpp. Each transistor that will be involved in the device programming is subjected to a junction stress for a predetermined time. This checks for possible low-voltage breakdown conditions.

The antifuse stressing is also performed using about one-half the required Vpp voltage level to actually program a fuse. This weeds out process defects that may have caused a 'weak' fuse to exist. This is very important because the key to addressing and selecting a single fuse during programming requires all of the rest to withstand Vpp/2 across them for the length of time to program on the ACTIVATOR.

ANTIFUSE SHORTS TESTING

This test is commonly known as the 'blank' test and is identical to the blank-check performed by you on the ACTIVATOR.

The previous stress 'tests' do not actually provide data for shifting-out. Those tests simply apply the stressing levels for pre-determined times, and the 'check' for any resultant shorting failure is tested out during the blank test.

SILICON-SIGNATURE AND BINNING TESTS

There are very few differences between the test flow performed on each die during wafer sort, and the final test flow on each package. However, the differences occur on these tests.

In the wafer sort flow, some words of the silicon signature may be programmed with factory-specific codes and information. This provides die traceability to the process fab, lot, and wafer. Also, a code is programmed that establishes the vpp level used. At final test or on the ACTIVATOR, this information can be reread when necessary.

The binning circuit consists of several extra testing modules that are configured during sort to produce a simple series of gates between the bin-in pin and the bin-out pin. At final test, the AC propagation delay measurements are made to 'bin' each packaged device into various speed-grade categories. Each unit is then symbolized with an add-on dash-number representing the overall performance range of each device.

I/O BUFFERS TESTING

The last group of tests performed are the functionality of each input, output, and clock network buffers. The parametric VOH/VOL and tristate leakage tests are now done also.

These are the only other tests whereby the remainder of the device pins are tested and actually used. Other than these tests, all of the previously described functional tests were performed using the 5 pins, MODE, DCLK, SDI, PRA, and PRB. The 12XX family also has an SDO pin.

CONCLUSION

This description of our FPGA testability attempted to provide you with a basic understanding of the test names, concepts, flow, and methods used during factory testing. For other details like temperature testing, actual parametric specifications, etc., refer to the datasheet.

The fully integrated test, programming, and operational architecture and circuit techniques of the TI FPGA's provide you with the highest possible quality and reliability with the preferred one-time-programmable antifuse technology.

SECTION 7.3

PROGRAMMING AND VERIFYING FPGA ANTIFUSES WITH THE ACTIVATOR

Jim Ptasinski
Texas Instruments, Dallas
Engineering

INTRODUCTION

The TI family of FPGA's, in comparison to the most common PLD's, utilize an antifuse technology instead of the popular fusible-link structure. It is this very important and fundamental difference of normally-open antifuses, versus normally-closed fuses, that allow the complete, near-perfect, design-for-testability circuitry used in FPGA's. The inherent nature of these antifuse-based FPGA architectures result in a 100% post-programming functional capability, while still having the preferred one-time-programmable feature.

This article will focus on one of the aspects that helps to achieve the highest possible yield after programming, i.e., the fuse verification testing performed during "ACTIVATE".

PROGRAMMING PROCEDURE OVERVIEW

The first step that takes place during a programming cycle in the ACTIVATOR is the attempt to read the silicon signature of each device in each socket. This insures that 1) the device was inserted properly, and 2) that the correct device type that matches the project is used. Note that on the present 10xx devices, if the security fuse option was used and also was successfully programmed, any attempt to read the silicon signature or checksum is disabled, and the display will show the false indication that the device was incorrectly inserted.

The second step, prior to the beginning of programming, is to determine that the device is 'blank', or not previously programmed. The blank test is executed using a highly parallel technique for the fastest response time. If one or more fuses are found to be programmed, the ACTIVATOR will display NON-BLANK and abort.

The blank-test makes no attempt to determine which fuses are shorted. Also, there are no provisions for reading or displaying a fuse-map as is done on many PLD-device programmers. Therefore, an unknown-design and programmed FPGA device cannot be verified or have its fuse-map extracted. This is because the place-and-route algorithm creates a unique order in the fusing sequence for every design, and without knowing the .FUS file used, the system would have no way of determining the verification paths to address each programmed fuse.

After the device passes the initial blank-test, the programming cycle starts as described below. The system will stop at the first indication of one of the failed tests performed, and an error display will appear. You are encouraged to label any failures with the fuse and test numbers at which the failure occurred, including the design_name also.

ANTIFUSE PROGRAMMING

The programming begins with the system putting the device into the programming mode and loading a fuse address. It starts with the first fuse in the .FUS design file and proceeds until all fuses and tests pass successfully. Only a single fuse is addressed and programmed at-a-time while all of the rest are biased at safe levels. The fuse can program with as few as one high-

voltage V_{pp} pulse, or may take hundreds, up to a maximum allowable value. If the limit is reached prior to the fuse being programmed, a "prog-fuse-fail" display occurs along with the failed fuse-number. The fuse number is the numerically consecutive number in the unique design .FUS file, and varies from code to code. This means that a particular fuse-number would generally never be the same fuse location on the die.

After the fuse address is applied, a lower-level V_{pp} is applied to try to measure I_{pp} current through the selected fuse. The value read should be near zero to indicate an initial open, or unblown antifuse. This is the method used for both preprogrammed and programmed fuse verification. If an initial current is read before the first programming pulse occurs, the error message "integrity check 6" will be displayed for that device.

ANTIFUSE VERIFICATION DURING PROGRAMMING

In the ACT1 family, several little-known tests take place before, during, and after each fuse is programmed. This insures the highest possible post-functional success and performance of each device.

Let's begin by describing how an antifuse is verified, or read electronically. The two most common industry methods are either logical sensing, or in the case of TI FPGA's, current sensing. Current sensing is by far the most versatile in the amount of information obtained and the control of the power each fuse is subjected with. Most PLD chip designers use a binary high and low functional output for detecting whether a fuse is programmed or not. TI FPGA's allow continuous μA current sensing of the V_{pp} pin in the ACTIVATOR during programming. This gives the system the ability to utilize many different thresholds based on the analog nature of current sensing.

The inherent variations in each fuses programming paths in the array can be overcome by optimizing the applied voltage and current levels for the various fuse types and locations throughout the device. The goal that is achieved from these techniques is controlled fuse resistance at the lowest attainable levels, to produce the highest and predictable performance and timing.

ANTIFUSE INTEGRITY TESTS '7' AND '8'

The two most important tests that are performed after each antifuse is programmed are called the 'integrity test 7' and 'integrity test 8'. Refer to Figure 1.

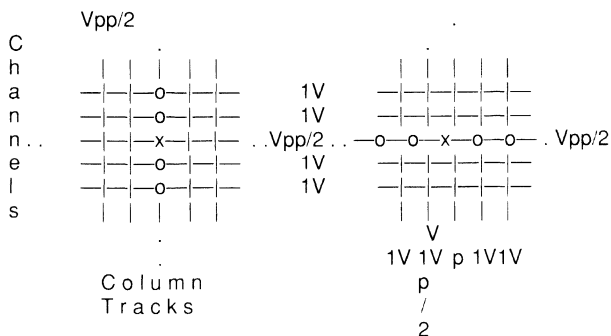


Figure 1

Before the next fuse to be programmed is addressed, the system will check that the integrity of the unblown fuses along that same column and channel track have been unaffected.

The integrity test '7' fuse verification routine rechecks to see if the fuses above and below the target fuse on the same column are still open. All of these fuses are addressed together in such a way as to measure any current flow thru Vpp. If any current flow is detected, programming will stop and the system will abort and display the integrity test and the last fuse number programmed.

The integrity test '8' is similar to the previous test '7', except it addresses unblown fuses along the horizontal channel track to the right and left of the fuse just programmed.

BROWSING YOUR .AVI FILE

During your programming cycle on the ACTIVATOR, a file is updated and stored in your design_name directory with the extension .AVI. This file records much valuable programming and verification data that had just been generated. You should use the BROWSE command or exit to DOS to read this file, especially when a failure occurs.

The .AVI file on the ACT-2 using the APS2.1v3 software revision or later, is organized with 8 columns of data following each of the fuse #'s. As you read from left to right, each of the 4 pairs of data correspond backwards to the 4 socket numbers, that is, socket#4, #3, #2, and, #1.

The first number of each pair is the final Vpp current that flowed thru that fuse. The second number of each pair is the total Vpp pulses applied to that fuse in that socket. As a typical example, 112 33 would mean that 11.2ma programmed that fuse after 33 pulses. Note that you must place the 'current' decimal point one digit to the left to read the result in milliamps.

If a 'programming fuse fail' display occurs at the last fuse number of a certain socket, the .AVI file will show 0 20000 as a possible result. When an integrity test fails, the fuse data usually appears normal. This means that one of these tests detected current elsewhere as previously explained.

The activate cycle can be restarted from or at the last fuse it stopped at the typing in 'act-fuse fuse# 0 1'. Try starting at the same fuse when it did not program at all to give it more pulses. For integrity test fails, try starting at the next fuse instead.

CONCLUSION

The purpose of this tutorial was to help you understand the inner workings of your activator and how it attempts to provide you with the highest possible field-programming quality.



SECTION 7.4

SPEED ENHANCED FPGA'S

Jim Ptasinski
Texas Instruments, Dallas
Engineering

INTRODUCTION

TI FPGA devices have the ability of sorting the relative speed performance that your system may require, into several speed bin categories, while still in the unprogrammed state.

The most common speed bins are referred to as "standard", the nominal device speed, or a "dash 1" premium higher speed grade device. A typical device will have no dash number following its device symbolization, while a "-1" will follow the device type of the higher speed grade.

WHY SPEED BINNING

Some of your system's designs, perhaps those whose board plans to consolidate most, if not all its logic into FPGAs, will occasionally find the need for speed sorted devices. That is, there will be a particularly critical path or location on the board that just needs to be 'somewhat' faster than all the rest. Once you, the FPGA designer, determine the need for dash-1 speed grade devices for your application, then this drives your purchasing to specify the higher speed bin device part number also.

THE TPC1010A and TPC1020A BINNING CIRCUITS

One of the FPGA testability features consists of a factory- programmed binning path in each die at wafer sort. This binning circuit consists of 16 modules on the TPC1010A in 2 columns of test modules on two sides of the array. The TPC1020A has 28 configured in the same manner with 14 modules on 2 sides. Also in each path there is the associated delay of one input and one output buffer. All of the bin modules are programmed into alternating inverting and non inverting buffers, each with a fanout of one.

After the binning circuit is programmed, the testing function is designed to provide an AC propagation delay measurement path from the bin input pin, BININ, to the bin output pin, PRA. The speed is then measured through the binning circuit during package level final test on each unit. After which, the sorted units are symbolized accordingly with the appropriate dash number. Note that these tests can also be performed after the array is programmed by you.

SETUP AND MEASUREMENT PROCEDURE FOR TPC10 SERIES

Step 1: Follow the normal power up of the device in a suitable designed test fixture. Connect all of the GND pins to the ground potential, and set the MODE, DLCK, and SDI initially to a VIL between 0V and 0.3V. Raise all of the VCC pins and VPP to 5.0V.

Step 2: With the MODE pin at 0V initially, raise it to 5V to put the device into the test mode.

Step 3: Load the seven bit mode register block of the serial shift register chain with the following sequence: 1011101. You do this by applying a bit, then clocking, followed by next bit until seven clocks are performed. Use a clock rate of less than 1MHZ and a setup and hold of about 0.2us. This puts the device into the binning measurement state.

Step 4: Now all you need to do is simply apply a pulse to the BININ pin and observe the output. The non inverting propagation delay can now be measured. Note that the t_{plh} and t_{p_{hl}} readings are generally not equal. The factory uses the slowest of the two reads to sort each device as an added guardband to the specifications.

TABLE 1: TPC10 SERIES BINNING PATH PIN ASSIGNMENTS

PIN NAME	PLCC			CPGA	PQFP
	44 PIN	68 PIN	84 PIN	84 PIN	100 PIN
MODE	34	54	66	E11	92
SDI	36	56	72	B11	98
DCLK	37	57	73	C10	99
BININ 1010A	9	12	NA	D2	34
BININ 1020A	7	10	13	C2	31
MPRA/BINOUT	38	58	74	A11	100

CONCLUSION

The article gives you a better understanding of what speed binning is, how it works, and what it can do for your designs. We also outline the TPC10 series bin test procedure to try on your own.

From the extensive characterization performed on FPGAs, it has been shown that the statistical correlation between binning and the majority of macro models tested has a greater than 95% confidence level.

**SECTION EIGHT
FPGA APPLICATION
EXAMPLE**

SECTION 8.1

INCORPORATING A KEYPAD AND DISPLAY INTO AN FPGA DESIGN

Patrick A Naylor
Department of Electrical and Electronic Engineering
Imperial College, London

This application note illustrates the design flow for TI FPGA systems using the simple example of a keypad to 7 segment display decoder. The issues of macro choice and timing analysis will also be considered.

It is intended that this application note may be of relevance to those wishing to familiarise themselves with TI FPGA devices and the design flow, as well as those who wish to incorporate a numeric keypad and display into a larger design.

INTRODUCTION

This application note uses a numerical keypad and 7 segment display decoder as a simple example of a circuit design using TI FPGAs. The design will be implemented on the TPC1010 device in a 68 pin PLCC package but the design principles discussed can be applied to the whole TI FPGA family since the design tools and design process are identical for all the devices in the family. The design of the keypad and decoder will first be described in section 1 and will then be developed in section 2 towards a TI FPGA implementation. Section 3 will describe the process of implementing and testing the design on a TI FPGA device and will include a brief discussion of speed critical nets, I/O assignment, timing analysis, back annotation, device programming and utilisation.

1.0 THE KEYPAD TO 7 SEGMENT DISPLAY DECODER

1.1 THE KEYPAD SCANNER

The keypad chosen for this design is of the type in which the keys are arranged into rows and columns with a connection available to each row and each column. When a key is pressed, a low impedance path is established between the row and column connections which intersect beneath the pressed key. In order to use this type of keypad, it is necessary to scan each of the columns sequentially and test each of the rows for a low impedance connection path. The design presented here uses a 3 bit ring counter which scans the keypad by applying a logic 1 to each column in turn. It has been chosen to scan the columns rather than the rows in order to minimise the required circuitry in the ring counter since there are 4 rows but only 3 columns.

The ring counter has been designed using the Workview Viewlogic tools which allow schematic capture of the circuit, incorporation of elements from the TI FPGA macro library and circuit simulation. In this case, three D-type flip-flops from the macro library were connected in cascade with Q outputs connected to the D inputs and the input clock to the circuit was buffered using the TI FPGA clock buffer. The circuit diagram for the keypad scanner is shown in figure 1. Figure 1 also shows the circuitry which performs the initialisation of the ring counter with a single logic 1. This synchronous initialisation circuit presets one of the flip-flops if all of the flip-flops are outputting logic 0. The preset signal can be written as

$$\text{PRESET} = !Q0 * !Q1 * !Q2$$

where Q0, Q1 and Q2 represent the output of each of the three flip-flops and ! indicates

negation. The action of both the initialisation circuit and the ring counter can be tested using the Workview Viewsim circuit simulator to produce a timing diagram like that shown in figure 2. In a pre-layout simulation such as this, unit delays of 1nS are used for all modules. Such delays can be scaled easily from 1nS to the typical figure of 5.4nS such that the Workview Viewsim circuit simulator can be used to give realistic pre-layout simulations. It will be shown later in this note that back-annotated post-layout simulations can also be obtained.

The circuit schematic for the keyboard scanner has been designed using macros selected from the A1000 macro library. Further discussion and examples of the choice of macros are given in section 2.1.

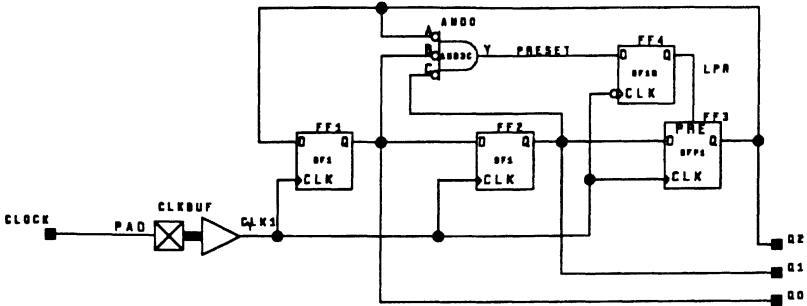


Figure 1 Circuit Schematic of the Keypad Scanner

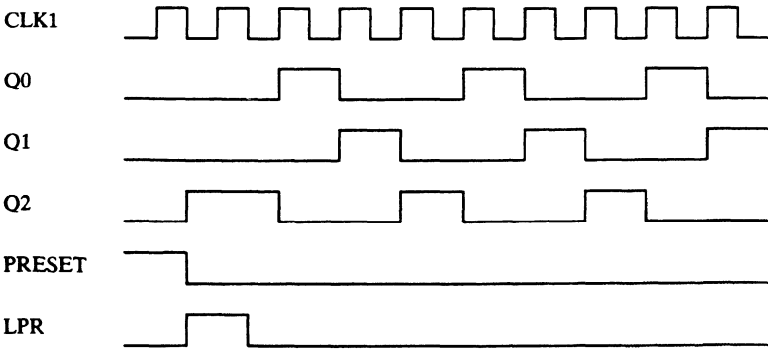


Figure 2 Timing Diagram of the Keypad Scanning Circuit

1.2 THE DECODER AND DISPLAY

The keypad scanning circuit described above has been designed using the Viewlogic tools for schematic capture and simulation which allow the design to work in an intuitive manner at the design stage with the added benefit of being able to test ideas quickly on the simulator. Under certain circumstances, the designer may wish to adopt a more formal approach based on the manipulation of Boolean equations. This second approach will be demonstrated in this section which describes the decoder circuitry.

It is necessary to decode the keypad signals in order to determine which key, if any, is being pressed. The key selection, or key pressing, process can be conveniently described by intermediate states. To do this, state "one" is defined to mean that the button marked "1" has been pressed, state "two" is defined to mean that the button marked "2" has been pressed and so forth.

There are seven keypad signals to decode, three column signals (Q0, Q1, Q2) and four row signals (D0, D1, D2, D3). The result of the decoding operation should be that the decoding circuitry enters one of the twelve possible intermediate states (zero, one, two, ... , nine, hash, star).

A combinational circuit is required to carry out this decoding operation. Before designing this combinational circuit, it is worth noting that the complexity of the circuit can be reduced significantly if it is recognised that one and only one of Q0, Q1 and Q2 can be asserted at any one time. This comes about because of the design of the keypad scanning circuit and means that, for example, if Q0 is a logic 1 it is always true that Q1 and Q2 are logic 0's. With this in mind, Boolean equations for the intermediate states can be written as follows.

one	= Q0 * D0 * !D1 * !D2 * !D3
two	= Q1 * D0 * !D1 * !D2 * !D3
three	= Q2 * D0 * !D1 * !D2 * !D3
four	= Q0 * !D0 * D1 * !D2 * !D3
five	= Q1 * !D0 * D1 * !D2 * !D3
six	= Q2 * !D0 * D1 * !D2 * !D3
seven	= Q0 * !D0 * !D1 * D2 * !D3
eight	= Q1 * !D0 * !D1 * D2 * !D3
nine	= Q2 * !D0 * !D1 * D2 * !D3
star	= Q0 * !D0 * !D1 * !D2 * D3
zero	= Q1 * !D0 * !D1 * !D2 * D3
hash	= Q2 * !D0 * !D1 * !D2 * D3

This simplifying step is one example of minimising (or optimising) Boolean Equations. In general, designers may choose to use software tools to carry out this optimisation. An example of such a tool is the ALES package which allows the designer to maximise logic utilisation in TI FPGAs for speed and area. This software also provides a synthesis tool for porting Boolean equation or state machine designs into the ALS environment. PLD design tools supported by ALES include ABEL, CUPL, PALASM, LOG/IC and PGAdesigner. ALES will accept a PALASM 2 source file generated with a text editor or generated with ABEL, CUPL or LOG/IC. For the purposes of this application note, however, any optimisation has been shown explicitly in order to aid understanding of the processes involved.

The remaining task at this stage of the design process is to arrange that the appropriate segments of the LED display are illuminated when a given key is pressed. This requires some further combinational circuitry. To help design this circuitry, the truth table which describes the

logic levels applied to the individual segments of the display for each of the intermediate states has been written out. This truth table is given below. (A, B, ... , G are the 7 segments of the LED display as shown in figure 3).

	A	B	C	D	E	F	G
one	0	1	1	0	0	0	0
two	1	1	0	1	1	0	1
three	1	1	1	1	0	0	1
four	0	1	1	0	0	1	1
five	1	0	1	1	0	1	1
six	1	0	1	1	1	1	1
seven	1	1	1	0	0	0	0
eight	1	1	1	1	1	1	1
nine	1	1	1	0	0	1	1
star	0	1	1	0	1	1	1
zero	1	1	1	1	1	1	0
hash	0	1	1	0	1	1	0

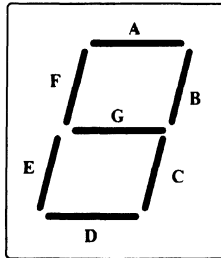


Figure 3 LED 7 Segment Display

From the above truth table, the following Boolean equations can be derived. Note that the expressions used are for the complement of each output variable as this leads to a more efficient Boolean description.

- !A = one + four + star + hash
- !B = five + six
- !C = two
- !D = one + four + seven + nine + star + hash
- !E = one + three + four + five + seven + nine
- !F = one + two + three + seven
- !G = one + seven + zero + hash

It is now possible to combine the two sets of Boolean equations above to yield

$$\begin{aligned} !A = & \quad !D0 * !D1 * !D2 * D3 * Q2 + Q0 * D0 * !D1 * !D2 * !D3 \\ & + Q0 * !D0 * D1 * !D2 * !D3 + Q0 * !D0 * !D1 * !D2 * D3 \end{aligned}$$

$$!B = \quad !D0 * D1 * !D2 * !D3 * Q2 + Q1 * !D0 * D1 * !D2 * !D3$$

$$!C = \quad Q1 * D0 * !D1 * !D2 * !D3$$

$$\begin{aligned} !D = & \quad !D0 * !D1 * D2 * !D3 * Q2 + !D0 * !D1 * !D2 * D3 * Q2 \\ & + Q0 * D0 * !D1 * !D2 * !D3 + Q0 * !D0 * D1 * !D2 * !D3 \\ & + Q0 * !D0 * !D1 * D2 * !D3 + Q0 * !D0 * !D1 * !D2 * D3 \end{aligned}$$

$$\begin{aligned} !E = & \quad !D0 * D1 * !D2 * !D3 * Q1 + D0 * !D1 * !D2 * !D3 * Q2 \\ & + !D0 * !D1 * D2 * !D3 * Q2 + Q0 * D0 * !D1 * !D2 * !D3 \\ & + Q0 * !D0 * D1 * !D2 * !D3 + Q0 * !D0 * !D1 * D2 * !D3 \end{aligned}$$

$$\begin{aligned} !F = & \quad Q0 * !D0 * !D1 * D2 * !D3 + D0 * !D1 * !D2 * !D3 * Q2 \\ & + D0 * !D1 * !D2 * !D3 * Q1 + Q0 * D0 * !D1 * !D2 * !D3 \end{aligned}$$

$$\begin{aligned} !G = & \quad Q0 * D0 * !D1 * !D2 * !D3 + Q0 * !D0 * !D1 * D2 * !D3 \\ & + !D0 * !D1 * !D2 * D3 * Q2 + !D0 * !D1 * !D2 * D3 * Q1 \end{aligned}$$

These 7 equations now represent the design of the display decoder. In the following sections the implementation of these equations will be discussed.

2.0 DESIGN FOR FPGA

2.1 CHOICE OF MACROS

2.1.1 KEYBOARD SCANNER

The TI FPGA macro library contains a large number of useful circuit elements. These include hard macros¹ such as gates, buffers, flip-flops and latches and soft macros such as counters, decoders, comparators, multiplexors, registers, multipliers, shift registers, some TTL equivalents and a UART.

In the design of the keypad scanner it would, therefore, have been possible to use a library part for the shift register required to make the ring counter. Instead, however, it was decided to build the shift register from individual flip-flops. This design choice has been made for two reasons: efficiency of device utilisation and minimisation of delays. The macro which would best fit the shift register design would be SREG4 - a 4 bit shift register with clear. This macro uses 8 logic modules compared to the 6 modules required by the 3 flip-flops hence the 3 flip-flop design makes more efficient use of resources. The SREG4 macro also uses 2 logic levels compared to the single level used in the 3 flip-flop design. Since the delays in the circuit can be estimated to a first approximation as (5.4ns x number of logic levels), then the 3 flip-flop design will insert only half the delay.

2.1.2 DECODER

The Boolean equations in section 1.3 describe the combinational circuits which are required to decode the outputs from the keypad scanner. It is possible to implement these equations directly using a combination of AND gates, OR gates and invertors. Normally, however, such an implementation is inefficient because no account has been taken of the available logic macros or the architecture of the logic module. This point is evident immediately if the AND functions are considered. The above equations require five input NAND gates of which none are available in the library (neither are they common in any other logic devices or macro libraries). It is, therefore, necessary to construct five input gates from two, three and four input gates.

The designer is faced with the question of how this should be done for maximum efficiency. To answer this question it is necessary to make reference to the logic module design as is done in the following section..

2.2 IMPLEMENTATION OF GATES WITH "FREE" INPUT NEGATION FROM 2-INPUT MULTIPLEXORS

Figure 4 shows an element of the TI FPGA logic module in the form of a two input multiplexor. The truth table shows how an output Y corresponding to the AND and OR function of the two inputs can be obtained from the multiplexor circuit. This figure also shows that the same logic module can produce AND or OR functions of two inputs when one of the inputs has an additional inversion. This additional input inversion requires no additional resources but can be obtained effectively "for free" because the logic module design employed in TI FPGAs is able to implement input inversion with a simple modification of the input multiplexors' programme. As an example, consider the cases of

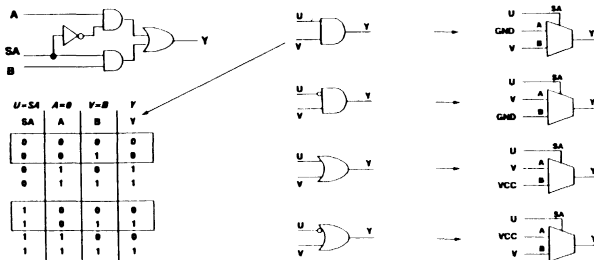
$$Y = U.V$$

and

$$Y = !U.V$$

The lines of the multiplexor truth table which are enclosed in boxes form the truth table for the AND function. These lines are selected by setting A to logic 0 and the two input signals are then fed to SA and B in order to implement the first of the two example functions. If it is now required to implement the second of the two example functions (ie AND with one input inversion) then this can be achieved by simply setting B to logic 0 and feeding the two input signals to SA and A. Thus it can be seen that no extra resources are required for the input inversion.

FPGA HARD MACROS

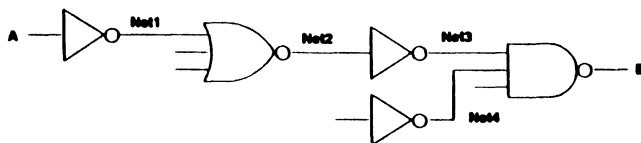


- FPGA hard macros are built from logic modules by programming the appropriate antifuses to connect the module inputs to GND or VCC.
- The logic module architecture is capable of building complex gates or flipflops with a minimum of resources.

Figure 4 Obtaining Logic Functions from the TI FPGA Logic Module

The property of "free" input inversions to gates described above can play not only a large part in increasing implementation efficiency but also in reducing the number of gate propagation delays. For example, figure 5 shows how a five gate circuit can be reduced to a two gate circuit. Since gate propagation delays often limit a circuits maximum operating frequency, the use of input inversions can be a powerful technique for speed enhancement.

Standard Logic: 5 Gates, 4 Nets



FPGA Logic: 2 Gates, 1 Net

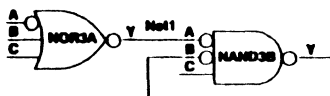


Figure 5 Circuit Reduction using Input Inversions

2.3 IMPLEMENTATION OF KEYPAD DECODER USING TI FPGA MACROS

In the design of the keypad decoder, the Boolean equations describing the required circuitry have been broken down into elements which can be implemented efficiently using TI FPGA macros. Particular attention has been paid to the use of input negation to improve efficiency.

By studying the circuits Boolean equations it can be seen that several terms occur more than once. These terms can be generated just once and can be referred to as intermediate terms I0, I1, I2 etc. in the following way.

$$\begin{aligned} I0 &= D0 * !D1 * !D2 * !D3 \\ I1 &= !D0 * D1 * !D2 * !D3 \\ I2 &= !D0 * !D1 * D2 * !D3 \\ I3 &= !D0 * !D1 * !D2 * D3 \end{aligned}$$

Referring to the macro library documentation, it can be seen that the macro AND4C will implement any of the intermediate terms completely including the necessary inversions.

It is now possible to rewrite the Boolean equations in terms of the intermediate terms.

$$\begin{aligned} !A &= I3 * Q2 + I0 * Q0 + I1 * Q0 + I3 * Q0 \\ !B &= I1 * Q2 + I1 * Q1 \\ !C &= I0 * Q1 \\ !D &= I3 * Q2 + I0 * Q0 + I1 * Q0 + I3 * Q0 + I2 * Q2 + I2 * Q0 \\ !E &= I0 * Q0 + I1 * Q0 + I1 * Q1 + I2 * Q2 + I2 * Q0 + I0 * Q2 \\ !F &= I0 * Q0 + I0 * Q1 + I2 * Q0 + I0 * Q2 \\ !G &= I3 * Q2 + I0 * Q0 + I2 * Q0 + I3 * Q1 \end{aligned}$$

These equations have been laid out such that like terms are vertically aligned. These like terms need be generated only once and, hence, further savings in circuitry can be obtained. A suitable macro for implementing the 2-input AND function is AND2.

Having considered methods for the generation of the product (AND) terms, it is now necessary to consider how the sum (OR) terms can be generated. Note that, in this particular example, the left hand side of the Boolean equations are complements and so it is actually NOR functions that have to be implemented.

The equation for !C requires nothing more than an inversion. This can be obtained implicitly from the use of a NAND gate instead of the AND gate previously indicated. A 2-input NAND macro is provided in the library as NAND2. A consequence of this inversion is that where the term occurs in other equations, the inversion must be taken into account. In this case, the only other equation which uses the $I0^*Q1$ term is the equation for !F. It will be shown shortly how the inversion can be handled.

The equation for !B requires a 2-input NOR. This is provided in the form of the NOR2 macro.

The equations for !A, and !G require 4-input NOR functions. Such a function is provided by the macro NOR4. However, this macro requires 2 logic modules and therefore uses twice the resources of a 1 logic module macro and, hence, reduces the quantity of chip resources available for implementing other circuitry. The macro NOR4A is a 4-input NOR gate which has one inverted input. This is a 1 logic module macro. This design can utilise this 1 logic module macro if one of the input terms to the macro is complemented such that, in combination with the input negation, a double complement is formed which cancels out. It can be seen that the term $I0^*Q0$ is common to the expressions for !A, and !G and so this is a convenient term to which to apply the double complement. In this design, $!(I0^*Q0)$ has been generated using the NAND2 macro. When this is done, the inversion must also be taken into account when other equations which use the $I0^*Q0$ term are implemented. In this case, the equations for !D, !E and !F are effected. It will be shown shortly how this inversion can be taken into account.

It is now helpful to rewrite the equations to include the additional inversions described above² and to move the overall inversion in each equation to the right-hand-side so as to give expressions for each segment of the display, rather than its complement.

$$\begin{aligned}
 A &= !(I3^*Q2 + !(I0^*Q0)) + I1^*Q0 + I3^*Q0) \\
 B &= !(I1^*Q2 + I1^*Q1) \\
 C &= !((I0^*Q1)) \\
 D &= !(I3^*Q2 + !(I0^*Q0)) + I1^*Q0 + I3^*Q0 + I2^*Q2 + I2^*Q0) \\
 E &= !((I0^*Q0)) + I1^*Q0 + I1^*Q1 + I2^*Q2 + I2^*Q0 + I0^*Q2) \\
 F &= !((I0^*Q0)) + !(I0^*Q1)) + I2^*Q0 + I0^*Q2) \\
 G &= !(I3^*Q2 + !(I0^*Q0)) + I2^*Q0 + I3^*Q1)
 \end{aligned}$$

It can be seen that the equation for F has had two input terms inverted and that double complements have been inserted to maintain the integrity of the equations. In order to implement the expression for F, the macro NOR4B has been chosen. This gate has 2 inverted inputs (one for each of the inverted product terms) and requires only 1 logic module.

The remaining equations to be implemented are for !D and !E. These equations have six terms, one of which has been subjected to an inversion during the above procedures. To implement these equations a combination of two gates is required. It can be seen that the terms $I1^*Q0$, $I2^*Q2$ and $I2^*Q0$ are common terms in these equations. It is more efficient, therefore, to implement these only once. The 3-input macro OR3 can be used to generate the logic OR of these 3 common terms. It now only remains to form the NOR of the output of the OR3 macro with three other terms, one of which has an additional inversion, to implement D and E. A macro which performs this NOR function with the necessary input negation has already been used in this design - NOR4A. Thus two NOR4A macros are used in conjunction with a single OR3, one to generate D and one to generate E.

Figure 6 shows the circuit schematic for the generation of the intermediate terms I0 to I3. Figure 7 shows the circuit schematic for the remainder of the keypad decoder.

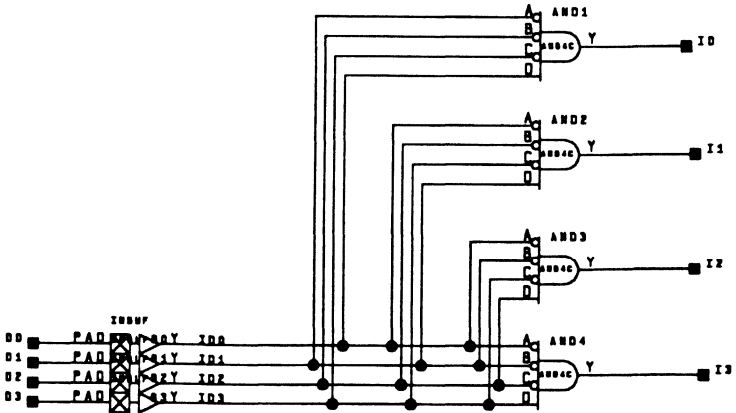


Figure 6 Circuit Schematic for the Generation of the Intermediate Terms

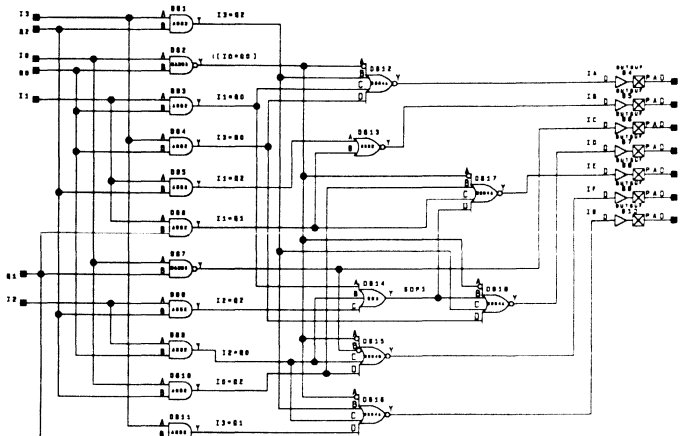


Figure 7 Circuit Schematic for the Keypad Decoder

3.0 FROM SCHEMATIC TO SILICON USING ALS

3.1 INTRODUCTION

In moving through the design flow, the designer needs to progress from a schematic representation of the circuit to a representation which is suitable to down-load to silicon. This is achieved by exporting a wire-list generated in the CAE environment to the Action Logic System (ALS). ALS is a powerful tool allowing the designer to not only programme silicon but also iterate a design in order to optimise any given criterion. More often than not, criteria of interest are speed of operation and device utilisation efficiency.

The ALS system builds a suite of design files, all of which are in ASCII text format. Each design file contains a specific type of information and is identified by a specific file name extension. These files are shown in the table below.

.ADL	Exported netlist file
.CRT	Critical path file
.IPF	Pin editor data file
.DEF	Definition file
.COB	Validator combine file
.VAL	Validation passed file
.VLD	Validation log file
.PIN	Pin assignment file
.DFR	Design for routability file
.LOC	Macro location file
.PLI	Placement information file
.SEG	Track segment file
.RTI	Routing information file
.DEL	Physical delay file
.MAP	Macro placement file
.FUS	Fuse file
.DTB	Workview delay table file

The use of these files will be discussed in the appropriate sections of the later parts of this note.

3.2 NET CRITICALITY

It is sometimes the case in circuit design that the speed of certain nets is critical to the circuit's operation. The ALS system allows the design to mark those nets which are speed critical. Four levels of criticality are supported: fast, medium, uncritical and default. If a net is marked as fast-critical the routing algorithm will use the shortest, fastest possible track layout. Up to about 5% of nets can be marked fast-critical. Marking a net as medium-critical instructs the routing algorithm not to use long vertical or horizontal tracks. Up to 15% of nets can be marked medium-critical. Uncritical and default nets are given no special consideration by the router.

Nets are marked as critical by editing the .CRT file. This file is initially blank and can be edited, according to the defined syntax, using almost any text editor. After such an editing process it is necessary to invoke the "certify" command on the file to update the file control information.

3.3 PIN ASSIGNMENT

A factor which can effect both operating speed and device utilisation is the location of I/O pins in relation to other parts of the internal circuitry. Furthermore, I/O placement is an important issue when parts of pre-existing circuitry are to be used since it is usually necessary to bring out particular signals from the device to particular locations.

Pin assignment information is stored in the .IPF and .PIN design file.

In the ALS system, both automatic and manual pin assignment are available. In the automatic method, the ALS software lays out the I/Os and assigns pins fully automatically. The software will attempt to optimise certain criteria - like minimum delay and maximum utilisation - in an overall sense. That is to say that all the elements which make up the design will perform to an approximately equal specification.

In some circuit designs, it is found that a particular part of the circuit is critical to the overall performance of the circuit. This situation can be accommodated within ALS by manually assigning I/Os in such a way as to optimise the critical elements' performance. For example, it may be possible to eliminate some of the routing delays from a speed critical path using a

combination of the .LOC design file (which describes the locations of the various circuit macros) and manual pin assignment. Manual pin assignment can also be useful when a TI FPGA design is to be incorporated into a design containing some pre-existing elements. In this case, it may be necessary, for example, to bring out particular signals from the device to particular locations on a circuit board. Manual pin assignment allows the designer to do this. However, if additional routing becomes necessary, some speed degradation should not be ruled out.

In this design it is necessary to connect 7 I/Os between the keypad and the FPGA and a further 7 between the FPGA and the display. In the case of the 4 D input I/Os, D0, D1, D2 and D3, pull-down resistors must be used in order for the scanner circuitry to function correctly. This is an example of a case in which manual pin assignment could be usefully employed since it is desirable to keep these four inputs close together so that an in-line resistor module could be used to provide the pull-down resistors.

3.4 TIMING ANALYSIS

The ALS software incorporates a comprehensive static timing analysis tool which is used to give post layout path delay information⁹ under a variety of predefined conditions - the most commonly used of such conditions being the worst-case.

In this application note, the timer will be used to investigate the maximum frequency at which the keyboard can be scanned by determining the maximum operating frequency of the keyboard scanning circuit.

3.4.1 MAXIMUM OPERATING FREQUENCY

To determine the maximum operating frequency of this, or any, synchronous circuit, it is necessary to find the worst-case delay in the circuit between clock transitions and data becoming valid on the output of latches or flip-flops. When calculating these delays, any gate delays and set-up times in the paths should be included. Delays determined in this way are known as register-to-register delays.

The ALS timer operates by finding path delays between specified start points and end points in the circuit. The set of start points is known as the "startset" and the set of end points is known as the "endset". For the sake of convenience, ALS maintains some default sets of starting and ending points. Two of the default sets are called CLOCK and GATED. The set CLOCK contains the set of all clock inputs while the set GATED contains the set of all gated outputs. So to find the maximum operating frequency of the circuit it is necessary to find the longest delay between any member of the set CLOCK and any member of the set GATED. This is done as a two-step procedure. Firstly the "startset" is chosen to be CLOCK and the "endset" is chosen to be GATED using the Set command from the timer's menu. Secondly, the longest delay is determined between these two sets using the Longest command from the menu.

These operations have been performed on the circuit and the following results obtained.

1st longest path to all endpins

Rank	Total	Start pin	First Net	End Net	End pin
0	16.2	FF4:CLK	LPR	PRESET	FF4:D
1	12.7	FF4:CLK	LPR	Q2	FF1:D
2	10.1	FF1:CLK	Q0	Q0	FF2:D
3	6.8	FF2:CLK	Q1	Q1	FF3:D

4 pins

It can be seen that in response the Longest command, the 4 longest delay paths have been

listed. The worst of these is between the CLK input PRESET net. It is very useful to be able to look in more detail at the sources of delay in any given delay path. This is supported in ALS by the Expand command.

If the path of Rank 0 is expanded the following information is listed.

1st longest path to FF4:D (rising) (Rank: 0)							
Total	Delay	Typ	Load	Macro	Start pin	Net name	
16.2	2.6	Tsu	0	DF1B	FF4:D		
13.6	3.0	Tpd	1	AND3C	AND0:A	PRESET	
10.6	6.3	Tpd	7	DFP1	FF3:PRE	Q2	
4.3	4.3	Tcq	4	DF1B	FF4:CLK	LPR	
0.0	0.0	Psk	8		FF4:CLK	CLK1	

It can be seen that 5 delay elements are listed as contributing to the overall delay of 16.2 nS. The flip-flop FF4 has both a required set-up time (Tsu) and a clock-to-Q delay (Tcq) which are 2.6nS and 4.3nS respectively. The propagation delay (Tpd) from the input of the AND gate AND0 and along the net PRESET is listed as 3.0nS. The propagation delay through the asynchronous preset input of the flip-flop FF3 and along the net Q2 is listed as 6.3nS. The sum of these delays is 16.2 nS.

Having determined the worst-case register-to-register delay path, it is now simple to calculate the maximum operating frequency F_{max} :

$$\begin{aligned}
 F_{max} &= \frac{1}{\text{worst-case delay}} \\
 &= \frac{1}{16.2 \times 10^{-9}} \\
 &= 61.7 \text{ MHz}
 \end{aligned}$$

3.4.2 INPUT TO OUTPUT DELAY

A further application to which the ALS timer can be put is the evaluation of the input to output delay of a combinational circuit. This delay time may be useful to the designer for various reasons, for example in checking that the setup times of any following sequential circuit elements are satisfied.

To evaluate the input to output delay, it is necessary to find the longest delay between any input pin and any output pin. This can be done using two further predefined sets of pins, namely INPAD and OUTPAD. The set INPAD contains the set of all input pads and the set OUTPAD contains the set of all output pads. As in the previous case of the CLOCK to GATED delays, a two-step procedure is followed to evaluate the delays. Firstly the "startset" is chosen to be INPAD and the "endset" is chosen to be OUTPAD using the Set command from the timer's menu. Secondly, the longest delay is determined between these two sets using the Longest command from the menu.

The input to output delay has been evaluated for the keypad decoder circuit and the following results were obtained for the worst-case conditions.

1st longest path to all endpins

Rank	Total	Start pin	First Net	End Net	End pin
0	67.7	G0:PAD	ID0	E	G8:PAD
1	66.4	G0:PAD	ID0	D	G7:PAD
2	64.0	G3:PAD	ID3	G	G10:PAD
3	63.9	G3:PAD	ID3	F	G9:PAD
4	62.8	G3:PAD	ID3	A	G4:PAD
5	53.5	G0:PAD	ID0	B	G5:PAD
6	52.8	G3:PAD	ID3	C	G6:PAD

7 pins

From this information it can be seen that the slowest path is from the G0:PAD (ie. the D0 signal input) to the G8:PAD (ie. the E segment output), the delay being 67.7nS. This timing analysis informs the designer of the delay between a valid D0 signal being input to the decoder and the output signals to the display becoming valid. If the segment outputs were to be latched, for example, by either an internal or external 7 bit latch, appropriate timing for the gate input to the latches could be determined.

3.5 BACK ANNOTATION

One of the ALS design files listed in section 3.1 was a .DTB file. The file is generated during the Place and Route operations of the ALS design process and contains post-layout routing delay information. This file allows the post-layout delay information to be fed back to the Workview Viewsim simulator so as to back annotate the results of circuit simulation with actual device delays. This feature of the TI FPGA design process can be extremely useful to designers allowing them to carry out not only functional simulation but also device simulation.

3.6 DEVICE PROGRAMMING

The programming of TI FPGA devices is supported by the ALS software and the ALS programmer. Programming is achieved by blowing blow-to-make anti-fuses using a programming voltage pulse. Two example programmed routes are shown in figure 8.

ROUTING

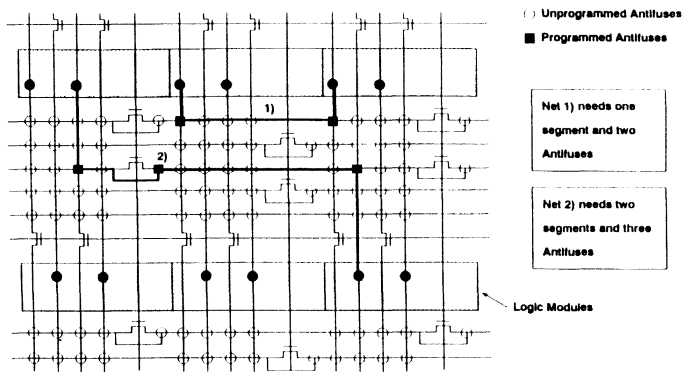


Figure 8 Route Programming

Of some interest is the method employed for addressing each individual anti-fuse. Since the device is initially full of non-connecting logic modules, it would normally be impossible to address logic modules other than those on the outside of the device. However, TI FPGA's make use of pass-transistors to perform routing during programming. Both vertical and horizontal pass transistors are provided on the device and allow a temporary connection to be made between the programming pins on the device and any anti-fuse. These are shown in figure 9.

ANTIFUSE PROGRAMMING PATH

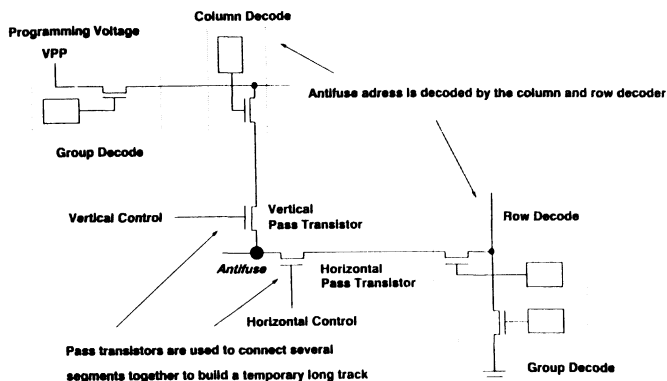


Figure 9 Pass Transistors Used for Programming

3.7 DEVICE UTILISATION

Device utilisation is reported by the ALS software in several of the ALS design files. A summary can be found in the .VLD file. For this design, the .VLD file contains the following summary information:

- VLD5300: Design uses:
 - 31 logic modules
 - 12 io modules
 - 1 clock module

92 module inputs driven by 35 signal nets with 2.62 average fanout.

The 68 pin PLCC TPC1010 device on which this implementation was done contains 295 logic modules and 57 I/Os. The implementation has, therefore, used about 10% of the logic modules and about 20% of the I/Os.

3.7 DEBUGGING AND TESTING

Once a device has been programmed it may be necessary to carry out debugging and testing. The ALS system supports two such operations, one of which is a functional debug during which the Activator hardware exercises the device under the control of the host PC, the other of which is an in-circuit debug which uses the Actionprobe diagnostic tools.

During functional debugging the inputs of the device under test can be controlled using either test vectors or using a control language and the outputs of the device can be monitored and reported to the designer. This process allows the designer to view the actual function of the device rather

than a simulation. In-circuit debugging using the Actionprobe tools is a unique feature of the TI FPGA family of devices and allows the designer to observe all internal nets using, for example, an CRO while the device is operating in-circuit. Up to two signals may be viewed simultaneously.

For details of the functional debugging control language and the use of the Actionprobe tools, see the ALS documentation.

4.0 CONCLUSIONS

A simple design has been used as a vehicle to describe the design flow for both sequential and combinational circuits using TI FPGAs. This design flow has been seen to encompass to elements of software, (i) a CAE environment for capture and simulation of the design and (ii) the Action Logic System for place and route operations as well as device programming.

It has been shown that the Workview Viewlogic tools (one of the supported CAE environments) allows designers to work both intuitively with schematic capture and the checking of ideas in simulation and also more formally by entering circuits resulting from Boolean equations in schematic form. Direct entry of Boolean equations has not been discussed in this note but is described elsewhere [2].

A degree of emphasis has been given to two important points which are specific to circuit design for TI FPGAs, (i) the choice of macros for high implementation efficiency and (ii) the use of the timer to establish maximum operating frequency and input to output delay. It has been seen that an underlying knowledge of the logic module structure in TI FPGAs and the ways in which the module is used to obtain various logical functions is a valuable asset to the designer; for example, judicious use of input negation has lead to the efficient mapping of Boolean equation descriptions onto the TI FPGA architecture in terms of both device utilisation and speed of operation.

5.0 REFERENCES

- [1] R J Landers, M G Harward, "A Benchmark Survey of Field Programmable Gate Arrays"
- [2] ALES Users Guide

1 For further information on hard and soft macros, see the FPGA Product Description documentation.

2 The reader may find the equations written in this form somewhat messy. It should be pointed out that the use of double complements in designs can often be dealt with implicitly rather than explicitly as has been done for the purposes of this example.

3 Delays can also be estimated before place and route operations have been performed. It is recommended that such pre-layout estimates are subsequently checked using post-layout analysis.

SECTION NINE
APPENDIX

**TPC10 SERIES
DATA SUPPLEMENT
PIN LOADING**

This supplement provides pin loading for the TPC10 Series 1.2µm CMOS Field-Programmable Gate Arrays. Use this supplement with the TPC10 Series Data Sheet and the Estimating System Speed Application Note to estimate manually the achievable system speed for a design implemented in a TI TPC10 Series FPGA. The index below will help you find the desired item in this supplement.

NAME	PAGE	TABLE	NAME	PAGE	TABLE
AND2	2	1	DF1B	10	39
AND2A	2	1	DF1C	10	39
AND2B	2	1	DFC1	10	40
AND3	2	2	DFC1A	10	40
AND3A	2	2	DFC1B	10	40
AND3B	2	2	DFC1C	10	40
AND3C	2	2	DFC1D	10	40
AND4	3	3	DFC1E	10	40
AND4A	3	3	DFC1F	10	40
AND4B	3	3	DFC1G	10	40
AND4C	3	3	DFE	11	43
AND4D	3	3	DFE1B	11	43
AO1	4	8	DFE2D	11	44
AO1A	4	8	DFA	11	43
AO1B	4	8	DFEB	11	44
AO1C	4	8	DFEC	11	44
AO2	4	8	DFED	11	44
AO2A	4	8	DFM	11	46
AO3	4	8	DFMA	11	46
AO4A	4	8	DFMB	11	46
AO5A	4	8	DFME1A	11	46
AOI1	4	8	DFP1	10	41
AOI1A	4	8	DFP1A	10	41
AOI1B	4	8	DFP1B	10	41
AOI2B	4	8	DFP1C	10	41
AOI2B	4	8	DFP1D	10	41
AOI3A	4	8	DFP1E	10	41
AOI4	4	8	DFP1F	10	41
AX1	4	7	DFP1G	10	41
AX1A	4	7	DFPC	11	42
AX1B	4	7	DFPCA	11	42
BIBUF	5	11	DL1	8	30
BUF	5	10	DL1A	8	30
BUFA	5	10	DL1B	8	30
CLKBUF	5	11	DL1C	8	30
CNT4A	13	54	DL2A	9	34
CNT4B	13	54	DL2B	9	34
DEC2X4	13	58	DL2C	9	34
DEC2X4A	13	58	DL2D	9	34
DEC3X8	14	59	DLC	8	31
DEC3X8A	14	59	DLC1	8	31
DEC4X16A	14	60	DLC1A	8	31
DECE2X4	13	58	DLC8A	8	32
DECE2X4A	13	58	DLCA	8	31
DECE3X8	14	59	DLE	9	35
DECE3X8A	14	59	DLE8	9	36
DF1	10	39	DLEA	9	35
DF1A	10	39	DLEB	9	35

NAME	PAGE	TABLE
DLEC	9	35
DLM	9	37
DLMS	10	38
DLMA	9	37
DLME1A	9	37
DLP1	9	33
DLP1A	9	33
DLP1B	9	33
DLP1C	9	33
FA1	7	22
FA1A	7	22
FA1B	7	22
FA2A	7	23
FADD12	7	25
FADD16	8	26
FADD24	8	27
FADD32	8	28
FADD8	7	24
GAND2	12	47
GMX4	12	48
GNAND2	12	47
GNOR2	12	47
GOR2	12	47
GXOR2	12	47
HA1	7	21
HA1A	7	21
HA1B	7	21
HA1C	7	21
ICMP4	12	51
ICMP8	12	51
INBUF	5	11
INV	5	10
INVA	5	10
JKF	11	45
JKFPC	11	45
MAJ3	4	8
MCMP16	12	52
MCMPC2	12	52
MCMPC4	12	52
MCMPC8	12	52
MX16	6	17
MX2	6	12
MX2_A	6	12
MX2B	6	12
MX2C	6	12
MX4	6	13
MX8	6	16
MX8A	6	16
MXC1	6	15
MXT	6	14
NAND2	2	1
NAND2A	2	1
NAND2B	2	1
NAND3	2	2

NAME	PAGE	TABLE
NAND3A	2	2
NAND3B	2	2
NAND3C	2	2
NAND4	3	3
NAND4A	3	3
NAND4B	3	3
NAND4C	3	3
NAND4D	3	3
NOR2	2	1
NOR2A	2	1
NOR2B	2	1
NOR3	2	2
NOR3A	2	2
NOR3B	2	2
NOR3C	2	2
NOR4	3	3
NOR4A	3	3
NOR4B	3	3
NOR4C	3	3
NOR4D	3	3
OAI	5	9
OAI1A	5	9
OAI1B	5	9
OAI1C	5	9
OA2	5	9
OA2A	5	9
OA3	5	9
OA3A	5	9
OA3B	5	9
OA4A	5	9
OA5	5	9
OAI1	5	9
OAI2A	5	9
OAI3	5	9
OR2	2	1
OR2A	2	1
OR2B	2	1
OR3	2	2
OR3A	2	2
OR3B	2	2
OR3C	2	2
OR4	3	3
OR4A	3	3
OR4B	3	3
OR4C	3	3
OR4D	3	3
OUTBUF	5	11
REGE8A	12	49
REGE8B	12	49
SMULT8	15	66
SREG4A	14	61
SREG8A	14	61
TA138	14	59
TA139	13	58

TA151	6	18
TA153	7	19
TA157	7	20
TA161	13	55
TA164	14	62
TA1691	3	55
TA181	8	29
TA194	14	63
TA195	14	64
TA269	13	56
TA273	12	50
TA280	13	53
TA3771	2	50
TRIBUF	5	11
UART	15	65
UDCNT4A	13	57
X01	3	5
X01A	3	5
XA1	4	6
XA1A	4	6
XNOR	3	4
XOR	3	4

Table 1. 2-Input Gates

	A	B	Y
AND2	1	1	0
AND2A	1	1	0
AND2B	1	1	0
NAND2	1	1	0
NAND2A	1	1	0
NAND2B	1	1	0
OR2	1	1	0
OR2A	1	1	0
OR2B	1	1	0
NOR2	1	1	0
NOR2A	1	1	0
NOR2B	1	1	0

Table 2. 3-Input Gates

	A	B	C	Y
AND3	1	1	1	0
AND3A	1	1	1	0
AND3B	1	1	1	0
AND3C	1	1	1	0
NAND3	1	1	1	0
NAND3A	1	1	1	0
NAND3B	1	1	1	0
NAND3C	1	1	1	0
OR3	1	1	1	0
OR3A	1	1	1	0
OR3B	1	1	1	0
OR3C	2	1	1	0
NOR3	1	1	1	0
NOR3A	1	1	1	0
NOR3B	1	1	1	0
NOR3C	1	1	1	0

Table 3. 4-Input Gates

	A	B	C	D	Y
AND4	1	1	1	1	0
AND4A	1	1	1	1	0
AND4B	1	1	1	1	0
AND4C	1	1	1	1	0
AND4D	1	1	1	1	0
NAND4	2	2	1	1	0
NAND4A	1	1	1	1	0
NAND4B	1	1	1	1	0
NAND4C	1	1	1	1	0
NAND4D	1	1	1	1	0
OR4	1	1	1	1	0
OR4A	1	1	1	1	0
OR4B	1	1	1	1	0
OR4C	1	1	1	1	0
OR4D	1	1	1	1	0
NOR4	1	1	1	1	0
NOR4A	1	1	1	1	0
NOR4B	1	1	1	1	0
NOR4C	1	1	1	1	0
NOR4D	1	1	1	1	0

Table 4. XOR/XNOR Gates

	A	B	Y
XOR	1	1	0
XNOR	1	1	0

Table 5. XOR-OR/XNOR-OR Gates

	A	B	C	Y
X01	1	1	2	0
X01A	1	1	2	0

Table 6. XOR-AND/XNOR-AND Gates

	A	B	C	Y
XA1	1	1	2	0
XA1A	1	1	2	0

Table 7. AND-XOR/AND-XNOR Gates

	A	B	C	Y
AX1	2	2	1	0
AX1A	2	2	1	0
AX1B	1	1	1	0

Table 8. AND-OR/AND-NOR Gates

	A	B	C	D	Y
AO1	1	1	1	NA	0
AO1A	1	1	2	NA	0
AO1B	1	1	1	NA	0
AO1C	1	1	1	NA	0
AO2	1	1	1	1	0
AO2A	1	1	2	2	0
AO3	1	1	1	2	0
AO4A	1	1	1	1	0
AO5A	1	1	1	1	0
AO11	1	1	1	NA	0
AO11A	1	1	1	NA	0
AO11B	1	1	2	NA	0
AO12A	1	1	1	1	0
AO12B	1	1	2	2	0
AO13A	2	1	1	1	0
AO14	1	1	1	1	0
MAJ3	2	2	2	NA	0

Table 9. OR-AND/OR-NAND Gates

	A	B	C	D	Y
OA1	1	1	1	NA	0
OA1A	1	1	2	NA	0
OA1B	1	1	1	NA	0
OA1C	1	1	1	NA	0
OA3	1	1	1	1	0
OA3A	1	1	1	2	0
OA3B	1	1	2	2	0
OA2	1	1	1	1	0
OA2A	1	1	1	1	0
OA4A	1	1	1	2	0
OA5	2	1	1	1	0
OA11	1	1	1	NA	0
OA12A	1	1	1	2	0
OA13	1	1	1	1	0

Table 10. Buffers

	A	Y
BUF	1	0
BUFA	1	0
INV	1	0
INVA	1	0

Table 11. I/O Buffers

	D	E	Y
INBUF	NA	NA	0
CLKBUF	NA	NA	0
OUTBUF	1	NA	0
TRIBUF	1	1	0
BIBUF	1	1	0

Table 12. 2:1 Multiplexers

	A	B	S	Y
MX2	1	1	1	0
MX2A	1	1	2	0
MX2B	1	1	1	0
MX2C	1	1	2	0

Table 13. 4:1 Multiplexer

	D0	D1	D2	D3	S1	S0	Y
MX4	1	1	1	1	1	1	0

Table 14. 4:1 Multiplexer

	D0	D1	D2	D3	S0A	S0B	S1	Y
MXT	1	1	1	1	1	1	1	0

Table 15. Other Multiplexer

	S	A	B	C	D	Y
MXC1	1	1	1	2	2	0

Table 16. 8:1 Multiplexer

	S2	S1	S0	D0-D7	Y
MX8	1	2	2	1	0
MX8A	2	2	2	1	0

Table 17. 16:1 Multiplexer

	S3	S2	S1	S0	D0-D15	Y
MX16	1	1	4	4	1	0

Table 18. 8:1 Multiplexer

	A	B	C	EN	D0-D7	Y	W
TA151	2	2	1	2	1	0	0

Table 19. 4:1 Multiplexer

	A	B	EN	C0	C1	C2	C3	Y
TA153	1	1	1	1	1	1	1	0

Table 20. 2:1 Multiplexer

	A	B	S	EN	Y
TA157	1	1	1	1	0

Table 21. Half Adders

	A	B	CO	S
HA1	2	2	0	0
HA1A	2	2	0	0
HA1B	2	2	0	0
HA1C	2	2	0	0

Table 22. Full Adders

	A	B	CI	CO	S
FA1	2	4	5	0	0
FA1A	3	3	3	2	0
FA1B	2	3	3	2	0

Table 23. Full Adder

	A0	A1	B	CI	CO	S
FA2A	2	2	3	3	2	0

Table 24. 8-Bit Fast Adder

	A0,A1	A2-A7	B0,B1	B2-B7	CI	S0-S7	CO
FADD8	2	4	3	6	3	0	0

Table 25. 12-Bit Fast Adder

	A0,A1	A2-A11	B0,B1	B2-B11	CI	S0-S11	CO
FADD12	2	4	3	6	3	0	0

Table 26. 16-Bit Fast Adder

	A0,A1	A2-A15	B0,B1	B2-B15	CI	S0-S15	CO
FADD16	2	4	3	6	3	0	0

Table 27. 24-Bit Fast Adder

	A0,A1	A2-A23	B0,B1	B2-B23	CI	S0-S23	CO
FADD24	2	4	3	6	3	0	0

Table 28. 32-Bit Fast Adder

	A0,A1	A2-A31	B0,B1	B2-B31	CI	S0-S31	CO
FADD32	2	4	3	6	3	0	0

Table 29. 4-Bit ALU

	A0,A1,A3	A2	B0,B1,B3	B2	M	CI	F0-F4	AEQB	P	G	CO	S0-S1	S2,S3
TA181	2	3	2	3	7	4	0	0	0	1	0	5	4

Table 30. D-Type Latches

	D	G	Q/QN
DL1	1	1	1
DL1A	1	1	1
DL1B	1	1	1
DL1C	1	1	1

Table 31. D-Type Latches with Clear

	D	G	CLR	Q
DLC	1	1	1	1
DLCA	1	1	1	1
DLC1	1	1	1	1
DLC1A	1	1	1	1

Table 32. D-Type Latch with Clear

	G	CLR	D0-D7	Q0-Q7
DLC8A	8	8	1	1

Table 33. D-Type Latches with Preset

	D	G	PRE	Q
DLP1	1	1	2	1
DLP1A	1	1	2	1
DLP1B	1	1	1	1
DLP1C	1	1	1	1

Table 34. D-Type Latches with Clear and Preset

	D	G	PRE	CLR	Q/QN
DL2A	1	1	2	2	1
DL2B	1	1	2	2	1
DL2C	1	1	2	2	1
DL2D	1	1	2	2	1

Table 35. D-Type Latches with Enable

	D	E	G	Q
DLE	1	1	1	2
DLEA	1	1	1	2
DLEB	1	1	1	2
DLEC	1	1	1	2

Table 36. D-Type Latch with Enable

	G	E	D0-D7	Q0-Q7
DLE8	8	8	1	2

Table 37. D-Type Latches with Multiplexed Inputs

	A	B	S	E	G	Q
DLM	1	1	1	NA	1	1
DLMA	1	1	1	NA	1	1
DLME1A	1	1	1	1	1	1

Table 38. D-Type Latch with Multiplexed Inputs

	G	S	A0-A7	B0-B7	Q0-Q7
DLM8	8	8	1	1	1

Table 39. D-Type Flip-Flops

	D	CLK	Q/QN
DF1	1	2	1
DF1A	1	2	1
DF1B	1	2	1
DF1C	1	2	1

Table 40. D-Type Flip-Flops with Clear

	D	CLK	CLR	Q/QN
DFC1	1	2	2	1
DFC1A	1	2	2	1
DFC1B	1	2	2	1
DFC1C	1	2	3	1
DFC1D	1	2	2	1
DFC1E	1	2	2	1
DFC1F	1	2	3	1
DFC1G	1	2	2	1

Table 41. D-Type Flip-Flops with Preset

	D	CLK	PRE	Q/QN
DFP1	1	2	3	1
DFP1A	1	2	3	1
DFP1B	1	2	2	1
DFP1C	1	2	2	1
DFP1D	1	2	2	1
DFP1E	1	2	2	1
DFP1F	1	2	2	1
DFP1G	1	2	2	1

Table 42. D-Type Flip-Flops with Preset and Clear

	D	CLR	PRE	CLK	Q
DFPC	1	3	3	2	1
DFPCA	1	3	3	2	1

Table 43. D-Type Flip-Flops with Enable

	D	E	CLK	Q
DFE	1	1	2	2
DFEA	1	1	2	2
DFE1B	1	1	2	2

Table 44. D-Type Flip-Flops with Enable, Preset and Clear

	D	E	CLR	PRE	CLK	Q
DFEB	1	1	3	3	2	2
DFEC	1	1	3	3	2	2
DFED	1	1	3	3	2	2
DFE2D	1	1	3	3	2	2

Table 45. J-K Flip-Flops

	J	K	PRE	CLR	CLK	Q
JKF	1	1	NA	NA	2	2
JKFPC	1	1	3	3	2	2

Table 46. Multiplexed-Input Flip-Flops

	A	B	S	CLR	E	CLK	Q
DFM	1	1	1	NA	NA	2	1
DFMA	1	1	1	NA	NA	2	1
DFMB	1	1	1	2	NA	2	1
DFME1A	1	1	1	NA	2	2	2

Table 47. Clock Buffer (CLKBUF) Interface

	A	G	Y
GAND2	1	1	0
GNAND2	1	1	0
GOR2	1	1	0
GNOR2	1	1	0
GXOR2	1	1	0

Table 48. Clock Buffer (CLKBUF) Interface

	D0	D1	D2	D3	G	S0	Y
GMX4	1	1	1	1	1	1	0

Table 49. Octal D-Type Flip-Flops and Registers

	CLK	CLR	D0-D7	Q0-Q7	PRE	E
REGEB8A	16	2	1	2	2	8
REGEB8B	16	2	1	2	2	8

Table 50. Octal D-Type Flip-Flops and Registers

	CLK	CLR	EN	D1-D8	Q1-Q8
TA273	16	2	NA	1	1
TA377	16	NA	8	1	2

Table 51. Identity Comparators

	An	Bn	AEB
ICMP4	1	1	0
ICMP8	1	1	0

Table 52. Magnitude Comparators

	An	Bn	ALBI	AEBI	AGBI	ALB	AEB	AGB
MCMP16	3	3	NA	NA	NA	0	0	0
MCMP2C	3	3	1	1	1	0	0	0
MCMP4C	3	3	1	1	1	0	0	0
MCMP8C	3	3	1	1	1	0	0	0

Table 53. Parity Checker

	A	B	C	D	E	F	G	H	I	ODD	EVEN
TA280	1	1	1	1	1	1	1	1	1	0	0

Table 54. Binary Counters

	CLR	CLK	LD	CI	P0-P3	Q0	Q1	Q2	Q3	CO
CNT4A	8	8	4	8	1	6	5	4	3	0
CNT4B	8	8	4	9	1	6	4	3	3	0

Table 55. Synchronous Counters

	LD	UD	ENT	ENP	CLR	CLK	A	B	C	D	QA	QB	QC	QD	RCO
TA161	1	NA	2	1	1	8	1	1	1	1	6	5	4	3	0
TA169	4	5	3	3	NA	8	1	1	1	1	7	6	4	4	0

Table 56. Synchronous Counter

	CLK	LD	UD	ENP	ENT	A-H	QA,QE	QB,QF	QC,QD,QG,QH	RCO
TA269	16	8	1	3	3	1	7	6	4	0

Table 57. Synchronous Counter

	LD	UD	CI	CLK	P0-P3	Q0	Q1	Q2	Q3	CO
UDCNT4A	4	5	9	8	1	7	6	4	4	0

Table 58. 2 to 4 Decoders

	A	B	E/EN	Y0-Y3
DEC2X4	4	4	NA	0
DEC2X4A	4	4	NA	0
DECE2X4	4	4	4	0
DECE2X4A	4	4	4	0
TA139	4	4	4	0

Table 59. 3 to 8 Decoders

	A	B	C	E	G1	G2A	G2B	Y0-Y7
DEC3X8	8	8	8	NA	NA	NA	NA	0
DEC3X8A	8	8	8	NA	NA	NA	NA	0
DECE3X8	5	5	5	8	NA	NA	NA	0
DECE3X8A	5	5	5	8	NA	NA	NA	0
TA138	5	5	5	NA	1	1	1	0

Table 60. 4 to 16 Decoder

	A	B	C	D	Y0-Y15
DEC4X16A	9	9	9	9	0

Table 61. Shift Registers

	CLR	CLK	SHLD	S1	Pn	SO
SREG4A	8	8	4	1	1	1
SREG8A	2	16	8	1	1	0

Table 62. Shift Register

	CLK	CLR	A	B	QA	QB	QC	QD	QE	QF	QG	QH
TA164	16	2	1	1	2	2	2	2	2	2	2	1

Table 63. Shift Register

	CLK	CLR	S1	S0	SLSI	SRSI	A	B	C	D	QA	QB	QC	QD
TA194	8	8	1	1	1	1	1	1	1	1	3	4	4	3

Table 64. Shift Register

	CLK	CLR	J	K	SHLD	A	B	C	D	QA	QB	QC	QD	QDN
TA195	8	8	1	1	4	1	1	1	1	4	2	2	2	0

Table 65. UART

	DATAIN7	DATAIN6	DATAIN5	DATAIN4	DATAIN3	DATAIN2	DATAIN1	DATAIN0
UART	1	1	2	1	1	2	2	1

Table 65. UART (continued)

	RXD	C0D	WR	RD	CS	DSR	RESETIN	CLK16X
UART	4	4	2	1	3	1	9	22

Table 65. UART (concluded)

	ENRXDATA	TXD	TXRDY	DATAOUT [7:0]	RXRDY	DTR
UART	1	1	1	0	3	1

Table 66. 8-Bit Multiplier

	A0-A2, A4-A6	A3	A7	B0,B4	B1-B3, B5-B7	P0-P15
SMULT8	6	9	14	8	4	0

TI Sales Offices

BELGIQUE/BELGIË

S.A. Texas Instruments Belgium N.V.
11, Avenue Jules Bordelhaan 11,
1140 Bruxelles/Brussel
Tel: (02) 242 30 80
Telex: 61161 TEXTBEL

DANMARK

Texas Instruments A/S
Borupvang 2D,
DK-2750 Ballerup
Tel: (44) 68 74 00
Telefax: (44) 68 64 00

DEUTSCHLAND

**Texas Instruments
Deutschland GmbH.**
Haggertystraße 1
8050 Freising
Tel: (08161) 80-0 od. Nbgst
Telex: 5 26 529 texin d
Btx: *28050#

Kurfürstendamm 195-196
1000 Berlin 15
Tel: (030) 8 82 73 65
Telex: 5 26 529 texin d
Düsseldorfer Straße 40
6236 Eschborn 1
Tel: (06196) 80 70
Telex: 5 26 529 texin d

Gildehofcenter
Hollesstrasse 3
4300 Essen 1
Tel: (0201) 24 25-0
Fax: (0201) 236640
Telex: 5 26 529 texin d
Kirchhorster Straße 2
3000 Hannover 51
Tel: (0511) 64 68-0
Telex: 5 26 529 texin d
Maybachstraße II
7302 Ostfildern 2 (Nellingen)
Tel: (0711) 34 03-0
Telex: 5 26 529 texin d

ESPAÑA

Texas Instruments España S.A.
c/ Gobelos 43,
Ctra de La Corona km. 14
La Florida
28023 Madrid
Tel: (1) 372 8051
Telex: 23634
C./Diputacion, 279-3-5
08007 Barcelona
Tel: (3) 317 91 80
Telex: 50436
Fax: (3) 301 84 61

FRANCE

Texas Instruments France
8-10 Avenue Morane Saulnier - B.P. 67
78141 Velizy Villacoublay cedex
Tel: Standard: (1) 30 70 10 03
Service Technique: (1) 30 70 11 33
Telex: 698707 F

HOLLAND

Texas Instruments Holland B.V.
Hogehilweg 19
Postbus 12995
1100 AZ Amsterdam-Zuidoost
Tel: (020) 5602911
Telex: 12196

HUNGARY

Texas Instruments International
Budapest u.42, H-112 Budapest
Tel: (1) 1 66 66 17
Fax: (1) 1 66 61 61
Telex: 2 27 676

ITALIA

Texas Instruments Italia S.p.A.
Centro Direzionale Colleoni
Palazzo Perseo - Via Paracelso, 12
20041 Agrate Brianza (Mi)
Tel: (039) 63221
Fax: (039) 632299
Via Castello della Magliana, 38
00148 Roma
Tel: (06) 5222651
Telex: 610587 ROTEX I
Telefax: 5220447
Via Amendola, 17
40100 Bologna
Tel: (051) 554004

NORGE

Texas Instruments Norge A/S
PB 106
Refstad (Sinsensveien 53)
0513 Oslo 5
Tel: (02) 155090

PORTUGAL

**Texas Instruments Equipamento
Electronico (Portugal) LDA.**
Ing. Frederico Ulricho, 2650
Moreira Da Maia
4470 Maia
4470 Maia
Tel: (2) 948 1003
Telex: 22485

REPUBLIC OF IRELAND

Texas Instruments Ireland Ltd
7/8 Harcourt Street
Dublin 2
Tel: (01) 755233
Telex: 32626

SCHWEIZ/SUISSE

Texas Instruments Switzerland AG
Riedstrasse 6
CH-8953 Dietikon
Tel: (01) 74 42 811
Telex: 825 260 TEXIN
Fax: (01) 74 13 357

SUOMI FINLAND

Texas Instruments OY
P.O. Box 86,
02321 Espoo
Tel: (0) 802 6517
Fax: (0) 802 6519
Telex: 121457

SVERIGE

**Texas Instruments
International Trade Corporation**
(Svergefilialen)
Box 30,
S-164 93 Kista
Visit address: Isafjordsgatan 7, Kista
Tel: (08) 752 58 00
Telefax: (08) 751 97 15
Telex: 10377 SVENTEX S

UNITED KINGDOM

Texas Instruments Ltd.
Manton Lane,
Bedford,
England, MK41 7PA
Tel: (0234) 270 111
Telex: 82178
Technical Enquiry Service
Tel: (0234) 223000

TI Regional Technology Centres

DEUTSCHLAND

**Texas Instruments
Deutschland GmbH.**
Haggertystraße 1
8050 Freising
Tel: (08161) 80 40 43
Frankfurt/Main
Düsseldorfer Straße 40
6236 Eschborn
Tel: (0 61 96) 80 74 18
Kirchhorster Straße 2
3000 Hannover 51
Tel: (0511) 64 80 21
Maybachstraße 11
7302 Ostfildern 2 (Nellingen)
Stuttgart
Tel: (0711) 34 03-0

FRANCE

**Centre de Technologie
Texas Instruments France**
8-10 Avenue Morane Saulnier, B.P. 67
78141 Velizy Villacoublay cedex
Tel: Standard: (1) 30 70 10 03
Service Technique: (1) 30 70 11 33
Telex: 698707 F

Centre Européen de Développement et Siège Social

Texas Instruments France
B. P. 5
06271 Villeneuve-Loubet cedex
Tel: 93 22 20 01
Telex: 470127 F

HOLLAND

Texas Instruments Holland B.V.
Hogehilweg 19
Postbus 12995
1100 AZ Amsterdam-Zuidoost
Tel: (020) 5602911
Telex: 12196

ITALIA

Texas Instruments Italia S.p.A.
Centro Direzionale Colleoni
Palazzo Perseo - Via Paracelso, 12
20041 Agrate Brianza (Mi)
Tel: 039-63221
Fax: (039) 632299

SVERIGE

**Texas Instruments
International Trade Corporation**
(Svergefilialen)
Box 30
S-164 93 Kista
Isafjordsgatan 7
Tel: (08) 752 58 00
Telefax: (08) 751 97 15
Telex: 10377 SVENTEX

UNITED KINGDOM

Texas Instruments Ltd.
Regional Technology Centre
Manton Lane,
Bedford,
England, MK41 7PA
Tel: (0234) 270 111
Telex: 82178
Technical Enquiry Service
Tel: (0234) 223000

 **TEXAS
INSTRUMENTS**