Table of Contents

List of Figures

List of Tables

Preface

Preface

This book has been written to serve as a general guide for the ASIC design engineer. It covers a number of important fundamentals related to design methodology relevant to application specific integrated circuits (ASICs). Although different silicon technologies are mentioned, this book does not focus on any one technology type.

It is recommended that you as an LSI Logic customer read this book thoroughly. In this book we present some of the potential considerations that may be necessary when designing your ASIC system. If this is your first ASIC design, you will most likely benefit from suggestions and comparisons made throughout this book. They should help you decide what trade-offs to make in choosing particular logic configurations for both performance and logic integrity.

In addition, test issues are discussed in order to emphasize the growing need to design testability into your logic at the beginning of your design and not as an afterthought.

Quit Table of Contents History Go Back ▶

The following issues are discussed in Chapters 1 through 7:

Chapter 1: an overview of ASICs - their uses, types, and characteristics

Chapter 2: some general design considerations - the use of cores and memories

Chapter 3: designing for speed

Chapter 4: designing for reliability

Chapter 5: designing for testability

Chapter 6: automatic tester requirements

Chapter 7: preparing test patterns for simulation

1

What is an ASIC?

An ASIC (Application Specific Integrated Circuit) is a semiconductor device designed especially for a particular customer (versus a Standard Product, which is designed for general use by any customer). The two major categories of ASIC Technology are Array-Based and Cell-Based (also referred to as Standard Cell). Array-Based ASICs configure a customer's design at the metal layers, whereas Cell-Based ASICs are uniquely fabricated at all layers of the silicon process including the diffusion layers.

CHAPTER 1 - SECTION 1
Uses of ASICs

To save chip area, ASIC technology integrates the logic and much of the memory formerly distributed among multiple ICs, thus improving reliability, optimizing PC Board space, and reducing component costs. In addition, the higher integration and smaller size results in significantly better system performance. ASICs were originally used solely to replace or consolidate TTL “glue” logic and consisted of relatively low complexity logic. Improvements in design tools and implementation software, in process technology, and in large pin count packages, now integrate much more of the logic formerly distributed among numerous ICs onto a single, very large scale System-On-a-Chip (SOC)^{TM}.

CHAPTER 1 - SECTION 2
Types of ASICs

There are two general categories of ASIC technologies: mask programmable ASICs and field programable ASICs. This book will focus on the mask programmable ASICs. In general, field programmable devices are reserved for smaller designs than are mask programmable devices. Although the turn around time for field programmable devices is less than that of mask programmable devices, the programable devices tend to offer this improved turn around time at the expense of performance. The metal lines and logic that are part of the field programmable devices are still present even after the device has been configured. So it is not uncommon to "breadboard" a design using field programmable logic to get the system design working, and then convert to a mask programmable ASIC for production purposes. The overhead in terms of both performance and die size is usually too high a price to pay to go into high volume production with field programmable devices. Some of the most popular field programmable technologies are:

- FPGA - PAL - PLA - HAL

The two most popular types of mask programmable ASICs that are being used in a great many System on-a-Chip designs today are:

- Array-Based -Cell-Based

A new emerging category of ASICs is the Application Specific Standard Product, or ASSP. This type of ASIC is discussed in more detail in Section 1.5. Don't confuse the ASSP with the term "full custom", which refers to a chip designed from scratch, but without the hooks needed to be used as an ASIC.

Array-Based ASICs

Array-Based ASICs offer the fastest customized implementation of a customer's logic onto a piece of silicon because of the small number of masks required to define the function of the chip. The diffusion layers have already been completed by the silicon foundry, and wafers are sitting in inventory waiting for the personalization provided by the customer. Most Gate Array Product Families have several “masterslice” sizes available to allow for some reasonable selection of die size that comes closest to the size needed to implement customer logic. Gate Arrays will always contain more gates than will be used to implement the custom design simply because they have been built as general purpose pre-constructed pieces of silicon, specifically so that turn around time can be fast, since only the top few layers of metal must be processed to complete a design definition. This concept is ideal for designs that will either have relatively low production volume or for designs that may need to be redone because of design changes or enhancements. Many designers will build designs in a Gate Array technology as proof of concept, and then redo the design in a Cell-Based technology before going to volume production.

Cell-Based ASICs

Cell-Based designs offer better die size optimization than Array-Based designs. A Cell-Based design consists of optimized building blocks, and approximates the performance limits and silicon efficiency of a full custom device. In Cell-Based designs, transistor geometries come in variable sizes. Components are not pre-processed (as in Array-Based), therefore cells can be located anywhere on the die. We will see later that cells are actually restricted to “cell columns” or “cell rows” soas to minimize the amount of bends that the Power and Ground metal must make. This also allows for easier “stacking” of cells if cells are placed in regular column or row structures rather than in a totally random fashion.

Application Specific Standard Products (ASSPs)

Full custom designs afford ultimate optimization, but at a very large cost in design time (with corresponding development costs). In both Array-Based and Cell-Based technologies, cells are pre-characterized - i.e., simulation models and layouts exist for each cell. A full custom design, costly and time consuming as it is, has advantages of speed and area over it's Array-Based and Cell-Based counterparts. But full custom may not always be a practical option for today's systems, which have short lifetimes and where time-to-market can be a significantly more important consideration. Application Specific Standard Products are indeed full custom chips but are available off the shelf. They have been designed by the ASIC vendor but have been defined to fit specific system requirements. They are not programmable devices.

CHAPTER 1 - SECTION 3
The Characteristics of ASICs

The remarks that follow further discuss some trade-offs of ASICs with respect to the following categories:

- Complexity
- Silicon Efficiency
- NRE
- Inventory Risk
- Design Risks
- Prototype Turnaround
- CAD / CAE Support
- Performance

Complexity

Complexity here means the number of transistors (or the amount of logic and/or memory) per given amount of area, plus the associated interconnect capability. Current Array-Based and Cell-Based chips accommodate as many as 20,000,000 usable logic gates on a single die. The term gate refers to a two-input NAND gate that consists of 4 CMOS transistors, two n-channel and two p-channel transistors. The standard “gate” is also a regular drive gate. This means that the gate can drive a typical fanout of about 16 unit loads for the given technology. You will note as we proceed that many new terms are likely to start creeping into our discussion. A standard unit load is equivalent to the amount of capacitance and current that one gate would have to drive while driving one input of another one of our basic “gates”. Array-Based technologies, such as the LSI Logic LCA500K Channel-Free Array, could implement up to about 2 million gates of random logic, up to 2 Megabits of mutliport RAM, and up to 6 Megabits of ROM. It is true that full custom designs can accommodate more gates, however Array-Based designs - especially in a Channel-Free Array technology - are capable of realizing functions that represent actual system building blocks and incorporate system memory functions on the same die. The Array-Based memories do tend to be about 5 times less dense than Cell-Based memories because they are constructed out of the gates on the masterslice. And full custom memories would provide much higher densities than do Array-Based memories. But in fact many designers who are using the Array-Based technologies to get fast turn around tend to be using very small “scratch pad” or “cache” types of memories which fit very well into the ASIC concept.

The newer Cell-Based ASICs, such as the LCBG12P Product, can incorporate about 20 million usable random logic gates, 12 Megabits of RAM, and 32 Megabits of ROM.

Silicon Efficiency

Array-Based technologies focus on fast implementation of logic integration onto a single chip, rather than on absolute highest density. Cell-Based designs allow you to get more logic onto a chip in a given area. Cell-Based designs feature transistors and routing tracks whose gradations of size are finer than those in Array-Based products. Thus Cell-Based designs use silicon more efficiently than Array-Based designs.

NRE

NRE ("Non-Recurring Engineering") charges are the costs associated with developing an ASIC. NRE is based on a number of factors. Some of these factors are: the complexity of the design, the technology chosen (# of masks required), the amount of work to be done by the customer and by the silicon vendor, the need for special cells or procedures, the type of package required, the schedule, the number of layers of metal, and so on. So NRE is not something which can be selected out of an ASIC "bluebook!" Normally, the more work the silicon vendor does and the more special the requirements, the higher will be the NRE. The more work the customer does, the lower the NRE. If the silicon vendor takes a customer design at the RTL level and does all of the synthesis, test insertion, pattern creation, simulation, static analysis, bonding diagram work, floorplanning, and static timing analysis, then the customer is likely to pay a significant NRE charge. This work is engineering intensive. If the customer does most of the aforementioned work and the silicon vendor completes the layout, makes the masks, and FABs the silicon, then the NRE will be lower since more of the engineering workload resides with the customer. Since much of the manufacturing cost is pure cost of materials (masks, wafers, test hardware, packages, etc.), this cost will be included in the NRE charge. The development of special cells, custom memory compilers, new packages, and other special requirements will all add incrementally to the NRE charges.

Array-Based designs require the fewest number of design-specific masks and therefore offer the lowest NRE to prototypes. Cell-Based designs require all masks to be generated for the chosen process and therefore the NRE charge will be higher for a Cell-Based design than for an Array-Based design.

Inventory Risks

A non-ASIC standard product, inasmuch as it can be used for a variety of purposes, has the least inventory risk. ASICs are more unique in their architecture and tend to be more focused in their uses - more customer specific. The more application specific your silicon is to your design, the more risk there is associated with building large inventories (wafers or dies). Array-Based chips have less inventory risk than Cell- based chips, at least for the vendor, because the masterslices can be used for many different customer designs. In Cell-Based technologies, all masks are customized per design and are therefore unable to be used for multiple customers. This means that if you over order silicon wafers for a Cell-Based design, you will pay the cost of all of the wafers you order, whether or not you use them.

Design Risks

The penalty for discovering a design error is higher for a Cell-Based ASIC than for an Array-Based ASIC. Mistakes after prototype fabrication in Array-Based designs usually only require that the metal mask layers be redone. On the other hand, design changes for a Cell-Based design may require that *all* masks be redone. Since masks cost over $4000 per layer in today's deep sub-micron world, the cost of materials will be much higher if a redesign is required in a Cell-Based technology. This must be balanced, however, against the tremendous advantages that Cell-Based technology brings to the ultimate die size (silicon area cost) and performance (much higher than for the Array-Based counterpart). Ironically today, as processes go more and more towards more layers of metal, a gate array design may begin to require a significant number of masks. For a 4 or 5 layer metal process, at least 10 masks would be required for the 5 metal layers and 5 via layers. In this type of process, the cost of a mask set for a Gate Array begins to increase to a point where one of the main advantages of Gate Arrays over Cell-Based designs starts to disappear.

Prototype Turnaround Time (TAT)

Designs that require a complete mask set (Cell-Based) will always require more time to manufacture than designs which use a basic set of diffusion masks and only require customization at the metal layers (Array-Based). This difference in time could be anywhere from one week to 4 weeks depending on how fast the silicon vendor can get masks from the mask shop and depending on how long the FAB cycle is for a given process. Obviously, the more layers of metal (more FAB steps), the longer the process will take. And although multiple layers of metal can help to ease the interconnect difficulties, each process step by definition will result in some yield loss. So more layers of metal does not automatically mean "better!" Also, in many cases, the silicon vendor will identify the location of the cells far enough in advance of final signoff, that the diffusion processing can actually start before the final signoff. This helps reduce the FAB time required for the diffusion layers.

EDA Tools (The old CAD / CAE Tools)

The evolution of EDA tools over the last 20 years has been mind boggling! In the 70s, vendors were struggling with the idea of automated layout. You may have seen things like high speed color plotters, digitizing tables, light tables for checking the electronic version against the hand drawings against the plots, and so on. And of course, companies were "cutting rubies" to send to mask vendors. Well, most of the layout tasks have been automated with very sophisticated place and route software, automated checking tools, and automated flows between the silicon manufacturers and the mask vendors.

Probably the biggest impact on IC design though has come as a result of the development of true design software rather than just software that translated, checked, and drew pretty pictures. Although designers were impressed with the graphical tools of the 80s such as schematic editors, waveform editors, graphical synthesis tools, and the like, it soon became obvious that to build large systems these graphical tools were too slow. The synthesis tools that have evolved, along with the ATPG tools for test, the high speed simulation tools, the static timing analysis tools, and the formal verification tools are now considered to be standard design tools.

And one of the most forward looking steps taken by many companies today, is to recognize that one company cannot create the best EDA tools for every part of the complex tasks that need to be performed to make workable chips today. Thus, many silicon vendors have begun using "3rd Party" EDA tools along with some in house software. The in-house software is usually for proprietary tools that perform a function unique to the silicon company's success. These tools are also tools that EDA vendors would not create because the market demand is not large enough.

The difficulties today are that the design sizes are becoming so large that the run times for many EDA tools is becoming prohibitive. This is spurring interest in hierarchical EDA tools.

Performance

Since the beginning of IC design, the two most critical parameters that have been used to measure the worth of new technologies have been speed and power. From a performance standpoint, both of these parameters are important. High power circuits are normally fast, but the increased power requires larger power supplies and tends to heat up the junctions on silicon chips which slows the devices. Each technology has brought some advantage to the world of ICs. However in today's most dominant ASIC technology - CMOS - high power can cause accelerated junction temperatures which can slow down speed. One way to reduce the power and still maintain speed is to develop circuits such as differential pairs that do not to switch from voltage rail to voltage rail. The SRC (Silicon Research Corp.) predicts as soon as the year 2005, the average chip voltage supply could drop to as low as 1.0 volt.

CHAPTER 1 - SECTION 4
Array-Based vs. Cell-Based Trade-offs

As a short comparison of Array-Based and Cell-Based Technologies, we can note the following:

Table 1.1: Technology Comparisons

	Gate Arrays	Cell-Based
Density	low - medium	high
Performance	low	high
Memory Capacity	low	medium
Cost	medium - high	low - medium
Turn Around Time	low	medium - high
Design Time	~ 6 - 9 months	~ 9 - 12 months

Why doesn't LSI Logic do Gate Arrays Anymore?

LSI Logic has always aspired to be a high-end ASIC company. Several years ago, when the largest feasible chips were less than 50,000 gates, design cycles could often be measured in weeks or a few months. The two or three week savings in turn-around-time for a Gate Array design was then a significant portion of the total design cycle, more than enough to offset the additional die size. Also, the smaller size of the designs and lower complexity of the process meant that both the engineering effort and mask costs were lower; hence, the NREs were lower, and both LSI Logic and our customers could afford to create ASICs targeted at lower volume systems. Lower volume means that the NRE is a large portion of the overall cost of the project, making the NRE savings available from a Gate Array more attractive. Lower volume also means that the additional die size of a Gate Array is not as important.

With today's deep-submicron processes and multi-million gate complexities come large teams of engineers for each chip, very expensive mask sets, longer total design cycles, and much higher NREs. LSI Logic and our customers are forced to consider only higher volume designs to offset the additional NRE, so the higher NRE of a cell-based design is no longer an impediment when compared to the total cost of the project. Also, the longer cycle times mean that the turn-around-time advantage of a Gate Array is no longer very large compared to the total design cycle. The higher volume of the design means that it is imperative to make the die size as small possible; the die area penalty of a Gate Array is no longer acceptable. All these factors have led LSI Logic to abandon the Gate Array concept upon which the company was founded.

CHAPTER 1 - SECTION 5
Standard Product ASIC Products (ASSPs)

One would think that many of the Standard Product companies would have a big edge over ASIC companies in the area of ASSPs. And the reason would be of course because of the wealth of chips already designed and characterized. It should be just a matter of taking off the chip I/Os and voila - an instant Core! Unfortunately, it's not as easy as all that. Most of the standard products that were designed over the last 20 years were designed as hand crafted chips. In some cases every single transistor was individually designed. And to go along with this “customization”, trade-offs were made that are now unable to be overcome. For example, many of the microprocessor chips that are available were designed over a 2 to 3 year period. But as they were designed, logic models or simulation models were not built along with them. So although the transistors and subcircuits were run through SPICE for circuit analysis, there is basically no way today to provide a model of these standard microprocessors to the Verilog and VHDL simulators. These standard products were also hand laid out in many cases. So no “auto place& route” topocell was built.

The ASSPs of today followed a different path. Many of the ASSPs were done as ASICs and then made available as stand alone functions that could be "packaged" as individual functions. And all it took to make stand alone parts out of these Cores was to put some I/Os on the boundary pins, and then package the device.

CHAPTER 1 - SECTION 6
Embedded Arrays

We have covered the fundamental features and benefits of Array-Based technology and Cell-Based technology, and we have introduced a new type of integrated product called the Application Specific Standard Product or ASSP. There is one other design concept that has existed in the ASIC world - it is called the Embedded Array approach. Although this type of product is different than either Array-Based or Cell-Based, it is not totally different than either one of these two types of ASIC. As a matter of fact, it exists as a marriage of the two technologies.

The basic concept of an Embedded Array is that a designer "embeds" either large memories or cores or both into their design and then fills in the area around the large cells with standard gate array transistors. Thus the chip is customized because the designer chooses the large cells that will be used on the design and where they will be placed, but the random logic area will be designed and laid out using standard gate array "masterslice" areas.

The big advantage of Embedded Arrays over either Array-Based or Cell-Based designs is that you get BOTH the density of cores and embedded memories from the Cell-Based products AND the turnaround time of Array-Based products. The embedded cells are floorplanned into the "custom masterslice" well in advance of final place and route so that the diffusion masks can be defined and wafers can be processed up to the metal layer steps. This turnaround time advantage comes because you are not using Cell-Based cells for the random logic portion of your design.

Of course the other major advantage of the Embedded Array concept is that if you need to make a change or if you have multiple designs that can all use the same “customized masterslice”, then you will save a significant amount of time turning the design around to get prototypes AND the cost will be significantly lower for multiple designs off the same basic masterslice.

An Embedded Array design requires the same cost as a Cell-Based design for the first run through simply because all masks are customized. But the turnaround time to prototypes is still much better than for Cell-Based designs.

So the trade-offs are that you give up some of the random logic performance vs. that available in a 100% Cell-Based design so that you can gain turnaround time and reduce overall costs. These trade-offs can offer an attractive alternative to designers that need the capability of getting their designs into silicon fast, but who also need the capacity offered by high density memories and core technology. However, it is also important to note that as the volume goes up and the performance requirements get tougher, the Embedded Array will again fall short of the advantage offered by Cell-Based ASIC technologies.

CHAPTER 1 - SECTION 7
Summary

Now that we have introduced the major categories of ASICs, we need to cover some of the methodology required to design a circuit in the ASIC world. The following Chapters cover various pieces of ASIC methodology starting with “Basic Building Blocks” in Chapter 2.

2

Design Trade-offs

Many trade-offs are going to have to be made during the course of a design project. In this Chapter we will discuss some of the trade-offs that must be considered. The trade-offs will involve giving up one benefit to gain an even greater benefit. We will consider:

- NAND logic vs. NOR logic
- The proper use of Buffers
- On chip memory vs. off chip memory
- The use of a PLL
- Hard Coded logic vs. Soft Coded logic
- Cores and Megacells
- Some limitations of Synthesis tools
- I/O Cells
- ESD Cells

Of course, these days most of the issues presented here are taken care of by the design tools, but it's still good for the designer to be aware of what's going on. Rather than suggest hard and fast rules for every design, the discussions in this chapter are meant more to make designers aware of the pros and cons of choosing a particular course of action.

CHAPTER 2 - SECTION 1
Use NAND Gates Rather than NOR Gates

Because the mobility of the electrons in N-channel transistors is approximately twice as fast as the mobility of the holes in P-channel transistors, N-channel transistors exhibit about one-half of the ON channel resistance of P-channel transistors of the same size. Figure 2.1 shows that the P-channel transistors of the NAND gate are in parallel. The P-channel transistors of the NOR gate are in series.

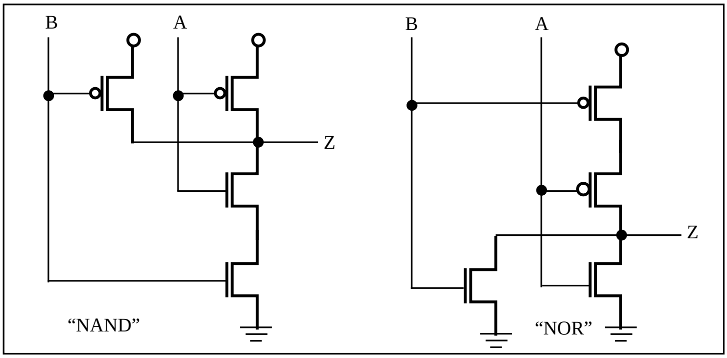


Figure 2.1
Comparison of 2-input NAND and NOR Gates

Therefore, the low-to-high and high-to-low propagation delays of the NAND gate are more nearly symmetrical than those of the NOR gate, with a resulting improvement in average speed. Again, even though synthesis tools will normally take this discrepancy into account, the engineer should be aware of it.

It is possible in a cell-based architecture to compensate for this behavior by making the P-channel transistors larger, but then the penalty is one of area rather than speed. For designs that are pad limited, where the core area of the chip is very sparsely utilized, using a larger transistor to get more balanced rise and fall times is definitely an option.

As with any benefit however, there is usually a side effect. Larger transistors have lower channel resistance and can therefore provide more current to charge or discharge capacitance. However, when input ramptimes become slow, it is possible for both the N and P channel transistors to be on for some period of time. When this happens, current can flow from the power supply to the VSS pin until one of the two transistors shuts off. This current is referred to as "crowbar" current and can cause an overall increase in chip power dissipation. See Figure 2.2 for a diagram that shows the effect of "crowbar" current.

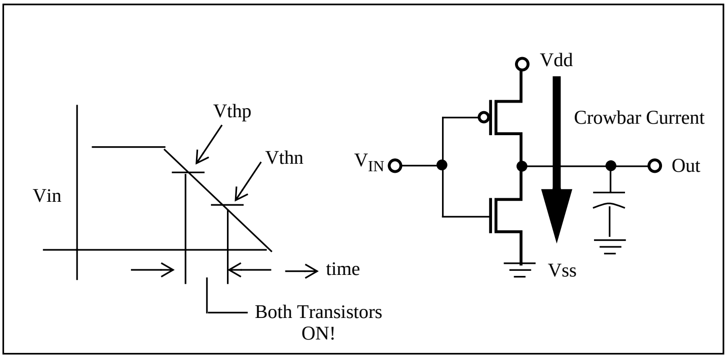


Figure 2.2
Crowbar Current

So it is suggested that a designer be cautious in the use of large output transistor cells.

CHAPTER 2 - SECTION 2
Proper use of Buffers

Buffers are required when a signal is routed either to a large number of destinations, or to a destination far away, or both. In the case of the far away destinations, the long metal run will add significant capacitance to the path parasitic model. In the case of the high fanout, the sum of the input capacitances will also cause a large capacitive parasitic component. In either case, there are several techniques which could be used to effectively "buffer" the capacitive fanout, each with its own advantages and drawbacks.

The most straightforward technique would be simply to add a large buffer at the source. But this begs the question of where the buffer should be located; in the case where a signal drives a long wire, you must ensure that the buffer is located near the source and not near the destination. Then there is the question of how large a buffer to use. Too small a buffer may do no good at all. Too large a buffer can overload the driver, introduce a significant delay in its own right in the case of a multi-stage buffer, or overdrive its output wire, causing noise and reliability problems due to metal migration. And of course we've already mentioned the problem of increased power dissipation as the buffer gets larger.

Another possibility is to add several buffers, either serially in the case of a long wire, or as a tree of buffers in the case of many destinations. In this case, take care that the delay of the buffers themselves isn't too large. Also, a tree of buffers may create skew problems unless it is carefully planned.

Be aware that, in addition to buffers created from core transistors, LSI Logic also offers the option of using I/O slot transistors as internal buffers. The good news is that you can use these very large buffers to drive many destinations from a single buffer to reduce skew. The bad news is that, in doing so, you lose the use of an I/O slot, and you must route the signal from the core logic to the edge of the die and back as shown in Figure 2.3.

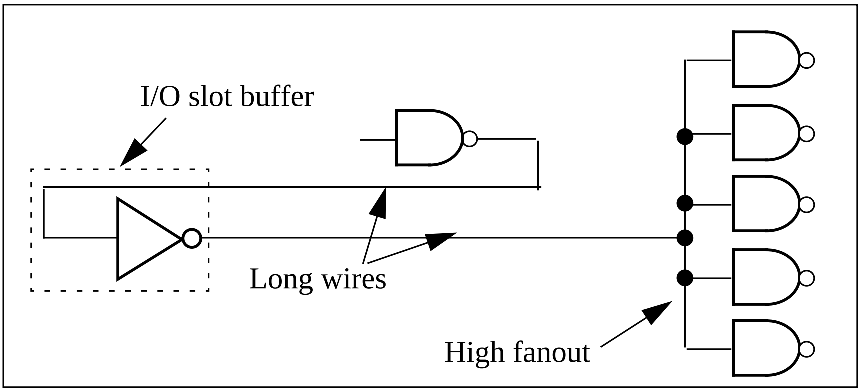


Figure 2.3
I/O Slot Used as Internal Buffer

In the LSI Logic G12 technology, the I/O slot buffer has been replaced by a dedicated “megacell” having large geometry transistors, but located within the core. You don’t lose an I/O slot, but you still need to give the buffer extra power and ground routing, which in many cases will limit its location to the periphery of the die, so Figure 2.3 would still be more or less accurate.

Since most ASIC design toolsets used today rely on feedback from the layout tools to some degree, the decision as to what type of buffer scheme to use to resolve a delay or ramptime problem, may in fact be made by the layout software given the correct user input. For example, using a series of buffers along a wire (referred to as repeater buffers) might be the overall best solution for driving a fanout which is distributed all across the chip. The wire will be long simply because the destination locations are all over the chip. But driving a long wire from a single source, no matter how big the driver is, will not be the most elegant solution. The concept of repeater buffers will allow a signal to drive destinations across the chip, and it will guarantee that no single piece of the wire in the path is larger than a certain value. This means that none of the repeater buffers in the path is likely to "see" a ramptime problem. It also means that with a smaller capacitive load due to shorter wires and smaller component fanout on any segment of the wire, the repeater buffers can be smaller cells. This may mean that internal cells are OK to use where a larger fanout may have even called for the use of an I/O buffer slot. This tradeoff means that a designer could save area and power dissipation by using repeater buffers. The key here is that since the repeater buffers would be distributed along the wire, they must have x-y coordinate information attached to them. This means that cell placement information is necessary for a tool or a human to be able to make this type of decision. See Figure 2.4 for an example of the use of repeater buffers.

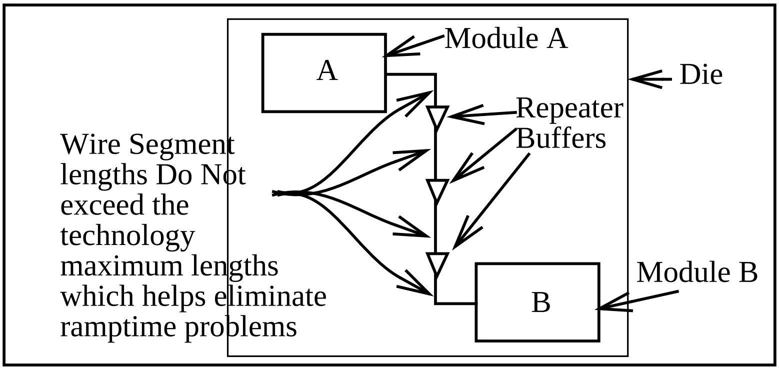


Figure 2.4
Repeater Buffers

Many of the synthesis tools used today have a feature called "IPO” or in-place optimization. The basic mechanism here is that the software can choose a larger or smaller buffer based on fanout. The difficulty is that the fanout is made up of both the destination capacitance of the load plus the wire capacitance. But the wire capacitance won't be known until the layout is complete. Unfortunately, given the long run times associated with full layouts for designs of today's size, engineers cannot always wait until the layout is finished to make many of the decisions that will affect design performance. The better ASIC toolsets provide designers with an intermediate step to get layout type of feedback without actually doing full layout. This step involves a chip floorplan. The floorplan not only allows designers a means of refining wire delay information, it can be tuned to a stage where cell placement information is very close to that of the final layout. The only step which is not performed is the actual wire routing. But given a good cell placement, the details of the actual wire length can be extrapolated with a fairly high degree of accuracy. This provides the designer with the feedback in the form of a custom wireload model which in many cases is close enough to the final routed wire model that a designer can begin to make performance decisions without actually routing the design! Then, tools like the IPO algorithm can decide whether or not to adjust the size of buffers needed to drive specific high fanout lines.

One other technique that is possible in terms of "buffering" a signal is to use the concept of split fanout. Figure 2.5 shows that instead of driving the two high fanout modules from one driving source, the driver could have been split into two buffers, where each buffer drives a more limited fanout. By taking some care during layout, any skew differences introduced by splitting the fanout can be tuned to minimize overall skew between the two signals. It is important to realize that by splitting the fanout, the cell driving the two buffers might have to be enlarged to handle a larger fanout. It is also important to realize that by splitting the fanout, the driver module now has another port added to it. This may or may not be an issue during a re-synthesis run if the original RTL code is not "told" about the new port.

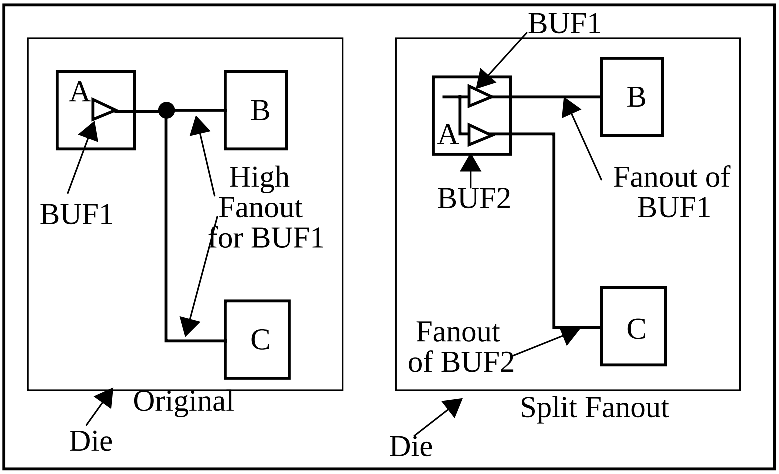


Figure 2.5
Split Fanout

In summary, some of the buffering techniques that can be used are:

- increase the driver buffer size
- insert repeater buffers in the line
- use a buffer tree to limit individual buffer fanout
- split up the fanout so that each buffer drives a reduced fanout
- use an IPO tool to adjust buffer size for load
- use an I/O buffer slot for very high fanout signals

Most designs will use some type of a buffering scheme. Many designs will use a combination of some or all of the techniques listed above.

CHAPTER 2 - SECTION 3
On Chip memory vs Off chip memory

ASIC technologies were originally designed to replace combinatorial logic which was spread out over many small TTL chips. In many of the very early systems, memory was not a big consideration.

Today almost every single chip design includes some type and amount of memory. So when does it make sense to put memory on an ASIC chip? The basic decision is based on the criticality of the design's performance. If it is crucial that the memory be right next to the logic it interfaces with, then the ideal situation is to have both pieces of the design on the same IC. But ASIC technologies do not usually support the very high memory densities one might find in DRAM chips. Even though the density of ASIC memories is growing very fast, it will still never quite catch the density that can be attained in the most dense memory technologies. One of the reasons is because ASICs still must provide for a way to build random logic. Furthermore, today's designs include some analog portions, some very high speed differential signals, very high density cores that perform specific "non-memory" functions, and so on.

So the ASIC chip has to do more than provide lots of memory. Given this fact, ASICs can complement the high density memory technologies.

It appears that many designs that are now being moved into ASIC technologies do not require lots of high density memory. Many of the memory needs are for cache memory, scratch pad memory, or for small memories local to specific logic functions. So although the amount of memory on board an ASIC chip may approach a hefty size, much of it is spread out throughout the design so that no one piece is considered "large".

In any case, memories are definitely special functions in that they are always hard coded pieces of a design. This simply means that the memory layouts are fixed once the configuration of the memory is defined. Because of the organization of the bits, words, sense amps and control logic associated with a memory it would not make sense to have "soft-coded" memories where the layout was not structured by definition.

The hard coding causes a couple of things to happen:

1. the memories usually have a fixed size and aspect ratio based on configuration;

2. the I/O pins of the memories are usually in predetermined locations around the periphery of the memory;

3. the density of the memories is usually such that no chip wires may route over the memory in order to avoid bit errors induced through crosstalk;

4. metal wires are typically much closer together inside the memories which creates a balancing act between the wires inside the memory and the wires on the rest of the chip;

5. due to the fact that memories tend to switch quite a bit of capacitance when data changes (the bit lines are long), memories can consume large amounts of power and therefore usually require their own power bus connections to the I/O power and ground pads;

Since hard coded entities like memories are densely populated areas of a design, and since they are usually hard coded to guarantee high performance or repeated performance, they tend to switch quite a bit of current during normal operation. So these entities carry their own power bus structure with them. It must be handled carefully on the layout. Today most designers get involved with the chip layout at least to the level of floorplanning. So even at this level, the location of these large hard coded entities on the chip must be carefully planned. The large power and ground buses that usually accompany the hard coded blocks can be most effectively dealt with if the large blocks are placed in a corner of the die or near one edge of the die. If this cannot be done, as in the case where there are too many of these blocks, then a special power bus matrix will essentially have to be designed to handle the current flowing to and from these blocks. These special power and ground matrices can take up significant amounts of die area if not handled properly. Figure 2.6 shows an example of potentially wasted area that results when large cells like memories are placed in non ideal locations on a die.

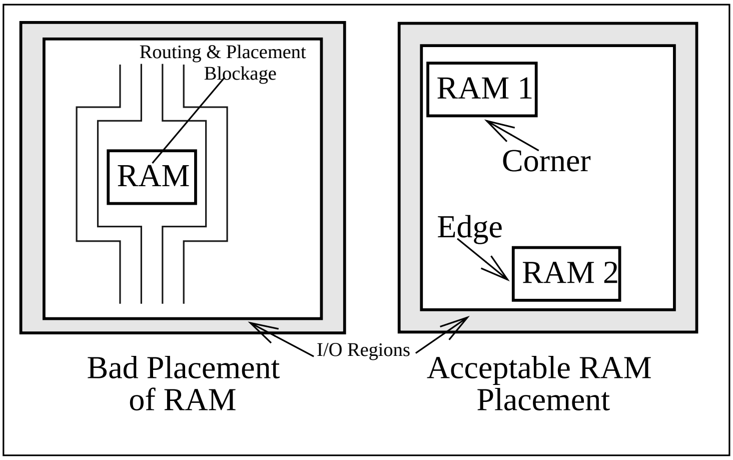


Figure 2.6
A Bad Place for a Memory

Memory blocks present other challenges for chip designers independent of their effect on chip layout. Issues such as test introduce concepts such as memory BIST logic, built in self repair, memory redundancy to allow for fixes without redoing masks, and so on. A much more extensive discussion of memory testing can be found in some of the Application Notes available in the ASKK documentation on the WEB. Testing memories that are part of a large ASIC design can be a non-trivial issue. The technique(s) used to test the memories could either add quite a bit of additional logic to the design or could complicate the test procedure required to guarantee high defect coverage on the memories.

CHAPTER 2 - SECTION 4
The Use of a PLL

In the best of all worlds, all chip designs would be totally synchronous and only have one clock signal. Alas, we live in the real world. And unfortunately, even those designs that appear to be synchronous can still suffer clock problems. Just about any scheme used to distribute the incoming clock signal around a large design will introduce serial delay into the clock signal before it actually reaches the flip flops it will be controlling. The serial delay doesn't necessarily create a big problem on chip since the skew between on chip flip flops can be minimized if care is used in the layout of the clock metal.

But the serial delay issue can cause a problem in a system where multiple chips need to be synchronized. So in effect, we need a way to synchronize the clock that reaches the flip flops on chip with the incoming clock. A very effective way to do this is to use a Phase Locked Loop module. This can basically guarantee that a designer can adjust the phase of the clock on one chip so that the overall system is synchronous. The PLL cell itself is relatively small, but it introduces an annoying issue into the design: analog circuitry! This requires that analog power and ground busses be added into the design and brought out to chip pins properly so that they do not interact with the digital power and ground busses on chip.

In general, the PLL will be able to adjust for just about any amount of serial delay that could result from long wires and clock trees, but the extra work required to handle the PLL cell correctly adds one more item to a designers list of things to do.

CHAPTER 2 - SECTION 5
Hard Coded vs. Soft Coded Logic

Hard coded logic refers to logic that has been pre-layed out before the actual place and route of the design itself begins. Some types of logic almost require hard coded layouts just due to their nature. Entities like memories are always hard coded simply because of their regular structure which cries out for a fixed layout for all of the bit cells due to their unique relationship to each other. Also, the control signals for the memory are of such a regular nature that it would be terribly inefficient to route them if the memory bit cells were layed out in a random fashion.

Generally speaking it is recommended that designers use soft coded logic blocks whenever possible. This gives layout software a lot more freedom to place cells and route metal than when hard coded logic is used simply because hard coded cells tend to act as blockages to place and route programs. This makes it more difficult to complete a chip layout since alternate locations must be used to find room to place and route the other cells and the wires in the design.

All designs will have many signals that travel some distance across the chip in order to reach all of their destination points. Given that ASIC chips have regular orthogonal “tracks” for routing wires, there are only so many routing tracks available on a given die size. So it's important to make sure that routing "space" is used wisely. When signals have to take detours around blockages, the bends or corners actually use up more die area than a straight piece of metal would, especially since changing direction usually means changing routing layers. So if possible it would be desirable not to have to bend any signal wires at all. Of course this is not possible, so the goal is to try and minimize the overall number of bends in wires as they route to and from their points of contact. The more hard coded logic on the die, the more detours signal wires are likely to have to take. This tends to cause longer wires, larger die sizes, and can create congestion in areas where it would not have existed. So the use of hard coded logic should be minimized.

Here are some cases where hard coded logic is actually required:

- memories
- need for repeatable performance
- special Cores
- macrocells

- critical paths

- datapaths

You may have a need for hard coded cells for reasons unique to your design, but it is best if you can minimize the needs for cells with fixed layouts.

CHAPTER 2 - SECTION 6
Cores and Megacells

LSI Logic has created a library of large system level building blocks which implement many industry standard functions. For some detailed information about LSI Logic's CoreWare® modules, visit the LSI Logic WEB page and specifically the section which describes the CoreWare program. It is located at:

http://www.lsilogic.com/products/coreware/coreware.html#library

The types of cores that are generally available are functions that have become very popular as building blocks for large systems. The ATM core is very popular because many systems today deal with asynchronous data which has to be moved from one place to another. The JPEG and MPEG Cores allow designers to use standard logic to handle still pictures and moving pictures without having to design logic from scratch to implement these standards. Many of the Cores handle bus interface issues for systems that have to communicate with standard bus protocols. LSI provides Cores for the PCI interface, the Fibre Channel interface, the USB interface, the Ethernet interface, and others. In fact, as new and more popular interfaces come along, LSI will introduce Cores to allow designers to incorporate these standard protocols into their system level chips.

More and more Mixed Signal applications are in vogue today. And LSI provides the most obvious Cores for the Mixed Signal interfaces: the A/D and D/A converters. And many of the new I/O cells that have been designed and are being designed, address the area of Mixed Signal interface.

Much work has been done at LSI Logic in the area of DSP and video compression. So in addition to Cores such as JPEG and MPEG, you will find video compression chips, error detection and correction chips, and new coding chips to allow for even denser compression algorithms.

But it doesn't make sense to design these large Cores from scratch. LSI provides these Cores as part of our growing Coreware library. They are meant to be used in many cases much like designers use basic low level cells - as is. In other words, the Core itself is already built as a cell. In many cases it may include a hard coded topocell which can be used by designers as a black box through the synthesis process.

Coreware modules include deliverables such as complete testbenches to exercise the Core(s), simulation shells for Verilog and VHDL simulation, timing shells for PrimeTime Static Timing Analysis, SPEF files for wire delay models, and full datasheets that include information such as static and dynamic power dissipation and switching performance parameters.

The LSI Logic FlexStream toolset includes programs which allow designers to create specialized functions for their design work such as datapath modules, which can be converted into hard coded entities that can be dropped into designs to ensure repeatability and optimized performance.

In addition to the CoreWare library, LSI has a team of Field CoreWare engineers who specialize in certain core technologies such as the Processor family: MIPS and SPARC RISC processors, the Mini-RISC and Tiny-RISC processors, and some coprocessors. Cores like the OAK Core, the ARM Core, and the HyperPHY Core have dedicated engineering personnel who are available to work with designers as applications support people to help optimize use of these cores.

CHAPTER 2 - SECTION 7
Some Limitations of Synthesis Tools

Synthesis tools have improved dramatically over the last few years. In fact, the improvements almost make one wonder if there is still a need for engineers to make critical logic decisions! Well, as it happens, even the best synthesis tools cannot know everything, and some of the things that they don't know CAN HURT YOU!

Generally speaking, the two most well known synthesis tools used by engineers are perhaps the Synopsys Design Compiler tool and the Cadence Build Gates tool. And these vendors advertise that these tools can synthesize up to about 200,000 gates in one run. No doubt the tools actually can do this, but the question remains: "is it wise to synthesize such a large piece of logic in one run"? LSI Logic recommends that designers synthesize blocks to the 30K - 50K gate range in one run. When the blocks get much larger than this, they may be creating logic which will occupy large sections of the die. If these synthesized areas are too big, then guiding the synthesis process with useful custom wireload information can be difficult.

A few of the areas that need to be considered before synthesis are the following:

- some foundry cells must be labelled as "don't use"
- memories cannot be synthesized by generic EDA tools

- cores are foundry specific and must be black boxed
- I/O cells are not synthesized
- RTL code typically cannot specify timing constraints accurately enough
- incomplete timing constraints can "mislead" synthesis tools
- initial wireload estimation techniques are usually inaccurate
- the way the RTL code is written can drive the type of logic produced, so it needs to be checked for correctness before the synthesis run.

Most often today, designers will go through a floorplanning cycle at least once to get more accurate custom wireload data which is used for a second synthesis run. And of course items such as the number of layers of metal, what type of test logic is used, and the overall clocking methodology will all help determine what the final floorplan will look like. This information is extremely valuable for the resynthesis run since it provides a physical reality check and balance for the design.

Tools are beginning to emerge that will help designers examine their RTL code, not only for synthesizability, but for design style errors which can mislead the synthesis tools and create the wrong type of logic. Within LSI, we have created a tool called **lsivega** which can be used to do RTL code reporting and generate synthesis scripts which follow the LSI Logic guidelines, and can also be used to report on critical nets such as clocks and other control signals. The tool generates design statistics and does some gate level DRC checking, and provides a schematic browser capability. The scope of this tool will grow over the next couple of years. Other tools such as the SENTE Wattwatcher tool help designers do RTL power estimation. Be prepared to see quite a bit of software that will be written to help designers look at their RTL code BEFORE going through possibly wasteful synthesis runs. We'll even be seeing RTL floorplanning tools to help get better wire estimates without having to synthesize to the gate level for a first pass.

CHAPTER 2 - SECTION 8
I/O Cells

Ten years ago, I/O cells consisted basically of input buffers, output buffers of various current drive capabilities, and three-statable and Bidirectional buffers. Today, the numbers of choices has grown beyond our ability to memorize all of the types or names. We now have mixed signal I/O with differential signal capability. We have level translators to go from one voltage level to another on chip. We have specialized GTL and NTL buffers, PCI buffers, the RAMBUS interface I/O, very high speed cells, LVTTL cells, LVDS cells, and on and on.

A designer used to have to decide which type of input threshold they needed (CMOS or TTL) and what output current drive they needed (4 ma, 8 ma, or 12 ma) and he/she was pretty much done! Today, it's important to know how the I/O signals are being used in the system soas to be able to select just the right combination of specs. to meet the pin by pin needs of the system. In addition since many systems today include JTAG logic (or boundary scan), it is necessary to guarantee that each I/O cell has appropriate JTAG logic available.

In addition to choosing the type of I/O cell to be used, it is necessary that designers plan the number and location of power and ground pads around the die so that issues such as crosstalk and signal integrity are accounted for.

And in today's designs we must deal with "analog" power and ground, cut power and ground buses, and ESD protection logic.

CHAPTER 2 - SECTION 9
ESD Cells

It has always been true that CMOS ICs in particular have needed special protection in the I/O area to prevent electrostatic damage to the very thin gate oxide regions of the die. As chips have gotten bigger, the types of I/O cells have gotten more varied, mixed voltage buses now share regions over the I/O cells, and design speeds have increased causing more simultaneous switching, and as package materials and styles have become more diverse, the ESD protection circuitry has had to evolve also to keep pace with a much larger variety of options at the periphery of the die. LSI Logic currently has two different types of ESD cells located in the I/O region of the chip. These cells provide protection for both the I/O and Core regions of the chip. Since ESD damage can occur when the chip is powered down sitting on the storage shelf, it is even necessary to protect against ESD damage between two ground pins which are normally assumed to both be at 0 volts! One would not intuitively know that protection circuitry was needed between two different ground pins on the package (VSS and VSS2). But of course when we consider that in a power down situation there may be sneak paths through oxides when an ESD voltage builds up between two ground pins, then we have to design circuitry to protect against this case. And indeed, we do keep many signals such as VSS and VSS2 separate on the die for good reasons. But when we do this, it introduces the potential for damage that is non obvious.

LSI Logic has designed two different types of ESD cells for the two major regions of the die - the core and the I/O area. And of course there are many variations of the I/O cell to accommodate the many types of I/O cells which could all possibly be tied to their own cut section of power bus.

The ESD cells occupy space and must be planned for in an ASIC design because they add necessary reliability to the chip. In this sense, they act as another type of I/O cell, since they occupy space around the edge of the die, and one must obey certain rules about spacing between ESD cells and to local power and ground pads. There must be enough of them on the die to prevent parasitic resistance from nulling out their effectiveness.

3

Design for Speed

Most of what used to be considered “designing for speed” nowadays means plugging tighter constraints into the synthesis tool. However, it is important to know what kinds of techniques the synthesis tools do automatically, since there are still occasions when an engineer must do things the old fashioned way.

CHAPTER 3 - SECTION 1
Reduce the Effect of Wire Delays

In current deep-submicron technologies, delays resulting from wirelengths have considerably greater relative effect on effective gate delay than loading due to other cells. Thus it is imperative that you: 1) properly floorplan your chip, 2) optimize drive strengths by using high-drive cells when necessary, and 3) partition your logic in such a way as to minimize the length of interconnect between cells.

The total loading capacitance is the sum of the output capacitance, the wire capacitance, and the sum of input capacitances of the driven functions. In both case (a) and case (b) in Figure 3.1, the sum of the input loads is 12; but CW2 in case (b) represents the wire capacitance to 12 places. (The term “input load” does not mean input capacitance, but it does relate to capacitance. Ultimately, LSI Logic uses the total output loading number for delay calculations.)

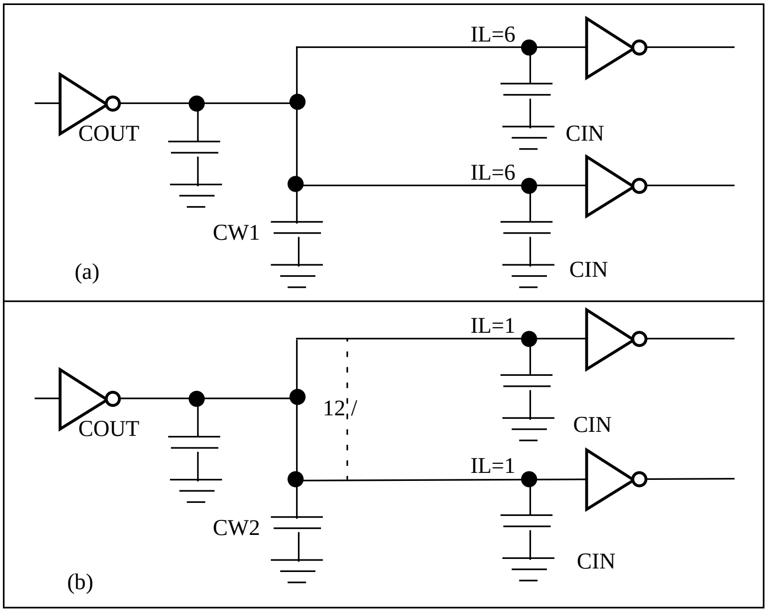


Figure 3.1

Capacitance Due to Cells vs. Capacitance Due to Wires

Interconnection trace resistance also affects delay; the LSI Logic FlexStream™ software tools perform an RC tree analysis after layout in order to guarantee very accurate post-layout delays.

CHAPTER 3 - SECTION 2

Use complex cells where appropriate

There is no such thing as a non-inverting CMOS logic gate. There are only double inverting logic gates, and a few four or six stage gates. There are, however, complex cells which perform multiple levels of logic gating with only one inversion. With so much delay in the wires, they can be faster than interconnected gates, but their size, compactness, and high pin count can cause routing congestion.

CHAPTER 3 - SECTION 3

Feed Late-Occurring Signals into the Last Logic Levels

Place early-occurring signals in the first levels of your logic and late-occurring signals in the last levels, as shown in Figure 3.2. This allows the early-occurring signals time enough to set up on the last stage. The later signal needs to propagate through only one low fanin gate.

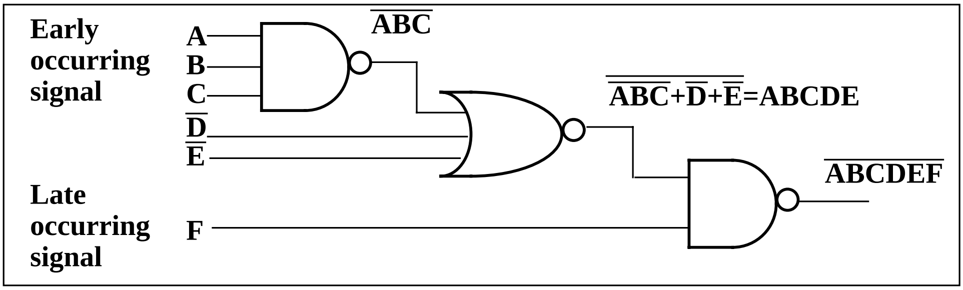


Figure 3.2
Comparison of 2-input NAND and NOR Gates

This technique can also be used to speed up certain inputs on high fan-in gates. The example in Figure 3.3 shows an 8-input NAND gate used in the critical path. The propagation delay between all inputs and the output is the same in (a). Signal H can be speeded up by replacing the ND8 gate with the structure shown in (b).

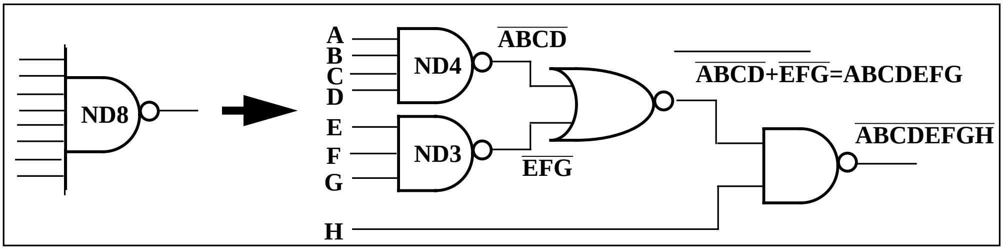
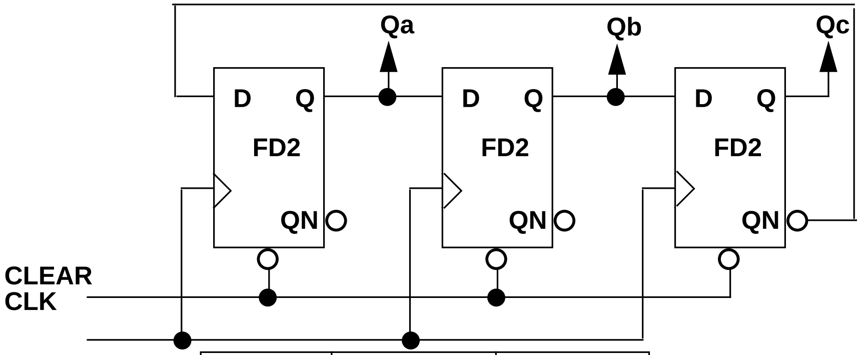


Figure 3.3
Comparison of 2-input NAND and NOR Gates

CHAPTER 3 - SECTION 4
Use Shift Counters in Place of Binary Counters

A Johnson counter is simply a shift register with the QN output of the last stage connected to the data pin of the first stage. Johnson counters are fast because there is no gating between the flip-flops in the counter. However, a Johnson counter will use more flip-flops than a binary counter. An n-stage Johnson counter gives a count sequence of 2 x n counts. Thus, for a modulo eight counter, a three-stage Johnson counter won't be adequate and a fourth stage is needed. Thus, the trade-off is speed versus gate count.



Qa	Qb	Qc
0	0	0
1	0	0
1	1	0
1	1	1
0	1	1
0	0	1

Figure 3.4

3-stage Johnson Counter with Count Sequence

CHAPTER 3 - SECTION 5
Use Duplicate Logic to Reduce Fanout and Increase Speed

You can reduce fanout and increase the speed of your circuit by allowing one of the outputs of the redundant logic to drive the critical path while the other output drives the non-critical path, as shown in Figure 3.5. Use this technique only if the timing skew introduced by separating the two paths will not create race conditions in the circuit.

This technique is useful only if the gain in speed from using duplicate logic exceeds the loss of speed that results from increasing the capacitive load of the previous macrocell.

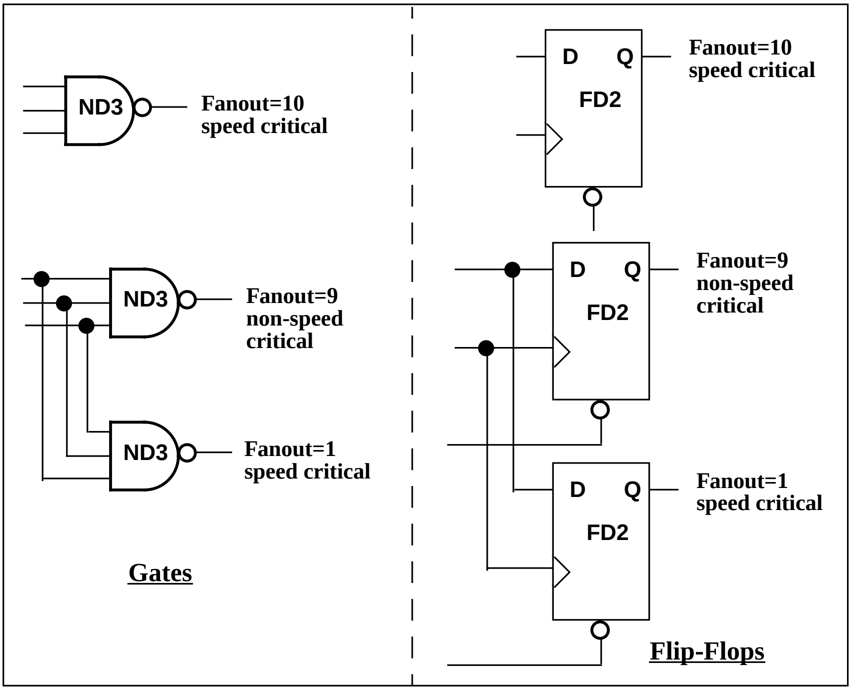


Figure 3.5
Duplicate Logic to Increase Speed

(One output signal drives a critical path; another drives a non-critical path)

CHAPTER 3 - SECTION 6
Use Input of Floorplanning

As newer technologies emerge, component speeds get faster and faster. This is good. However, die sizes are getting larger and larger and the average length of any given wire in a design can potentially be longer. Smaller geometries carry with them increased channel resistance which tends to reduce charge current. In addition to this, power supply values are dropping quickly. After almost 30 years, the standard 5.0v volt supply is now giving way to 2.5v, 1.8v, 1.2v, and yes even 1.0v! Once again this is good in that the power dissipation can be reduced. However this reduced supply voltage also has the side effect of reducing available charge current.

The above changes demand that designers take more care in the layout of their designs. The best way to manage a design layout at the design level is to be "hands on" involved with the design floorplan.

Placing design modules in the correct location on a die can significantly input performance. And in fact, floorplanning takes designers through the placement process, not only for modules and blocks, but also for components down to the cell level. Figure 3.6 shows how all placement data can help designers close timing before final routing. Notice that in Figure 3.6a, although the design has been floorplanned, since no cells have been placed, wirelength is estimated based on Block sizes and locations. However if cells "x" and "y" as shown in Figure 3.6 had been located in positions 1 and 2 or 3 and 4 as in Figure 3.6a, the "generic" wire estimate (shown by the dotted line in Figure 3.6a) could be quite far off. This before could be either overly optimistic or pessimistic, and you, the designer, don't know which is the actual value of the wire length. In figure 3.6b, all cells have been placed and the estimated wirelength between "x" and "y" will be much closer to the final value. Using

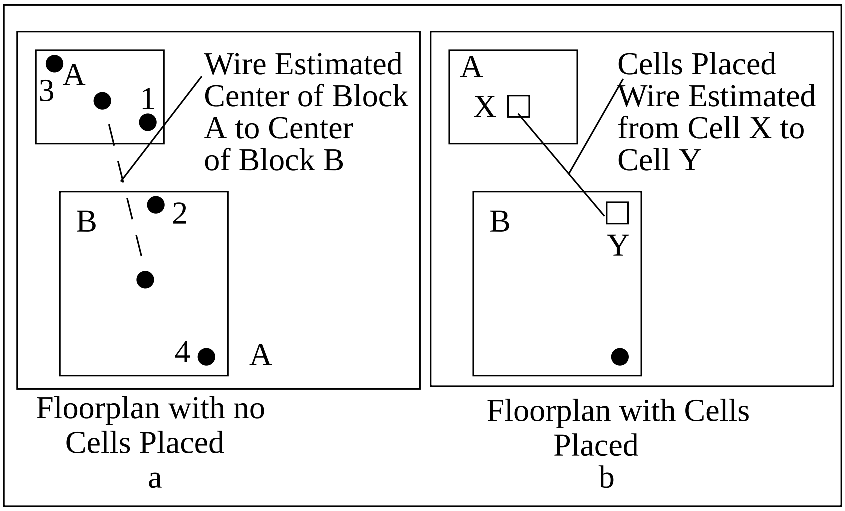


Figure 3.6
Better Wire Models After Cell Placement

the placement information, a designer can choose from a number of techniques to solve timing problems BEFORE going to final layout and detailed routing.

So Floorplanning not only helps to located the system logic on the die (Blocks A and B in Figure 3.6) but allows a designer to place all cells prior to Backend Layout.

Analyzing the results of the Floorplanning and cell placement information can allow designers to select strategies to solve timing and ramptime problems before final layout. This tends to generate fewer surprises after layout is finished.

The Customer Education Department will be producing on-line Application Notes beginning this year (2000). One of these addresses the issue of floorplanning in more detail. Suffice it to say, floorplanning is now a crucial step in the design process.

Many other considerations need to be made at the Floorplanning level in order to close timing after final layout. The locations of memories and cores, special I/O cells, the location of the clock and the flip flops it drives, the location of boundary scan logic, the location of power and ground pads and ESD cells, all contribute to Timing Closure.

A designer MUST be willing to invest time in a good Floorplan and cell placement in order for the router to complete its work and not destroy the delicate tradeoff between ever smaller geometries and bigger die sizes.

4

Designing for Reliability

This chapter discusses a number of issues which, if not properly handled, can cause designs to fail. Alternatives are presented to show comparisons between marginal logic design techniques and rigorous techniques that lead to more reliable silicon.

CHAPTER 4 - SECTION 1
Use Synchronous Design

Asynchronous circuits may cause problems, especially when (spike-prone) combinational logic is used to clock or reset storage elements. LSI Logic strongly recommends that your designs by synchronous.

Do not use asynchronous (ring) oscillators. Process variation renders their frequency unpredictable. As a result, parts cannot be tested using standard automatic test equipment. Moreover, ring oscillators introduce noise, which also reduces testing reliability.

To avoid glitches, use synchronous pulse generators. Figure 4.1 illustrates the incorrect use of an asynchronous pulse generator whose function is to provide a reset pulse whenever a positive edge occurs on its input.

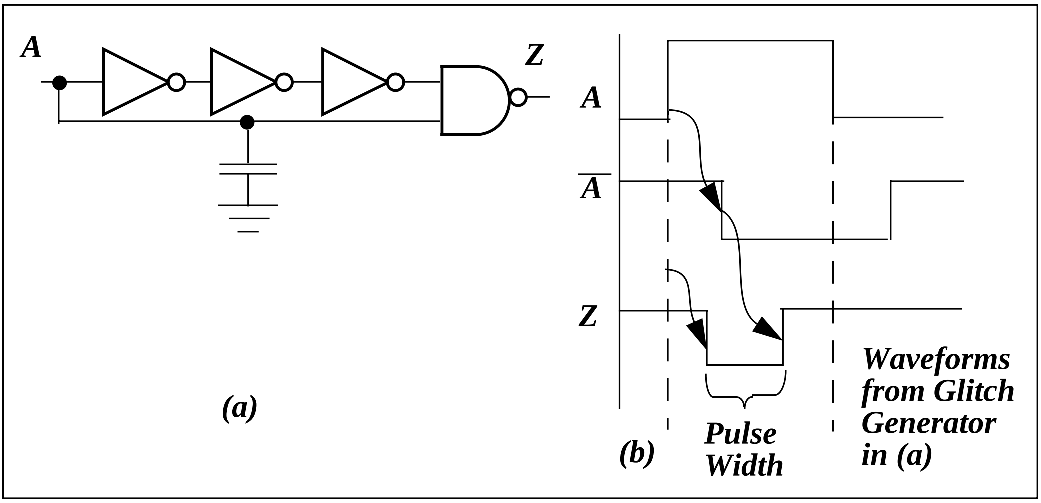


Figure 4.1
Classic Prop-Delay Based Pulse (glitch) Generator

Note that the circuit in Figure 4.1 should not be used in LSI Logic ASIC designs. The pulse width in the above circuit would appear to be equal to the cumulative propagation delay of the string of inverters. However, a distributive-capacitance model of the circuit reveals that the pulse width on the output of the NAND gate is equal to the difference in propagation delay between the inverter string and the wire beneath that inverter string. During pre-layout simulation, this circuit may perform satisfactorily. However, if the wire should be longer than expected after layout, the resulting increase in capacitance will cause the delay in that wire to increase proportionately. Increased voltage, decreased temperature, and process variation will decrease the delay of the inverter string. If the wire delay equals that of the inverter string, no output pulse will be generated; asynchronous pulse generators such as this can cause glitches. Use a synchronous pulse generator as shown in Figure 4.2 in place of an asynchronous one.

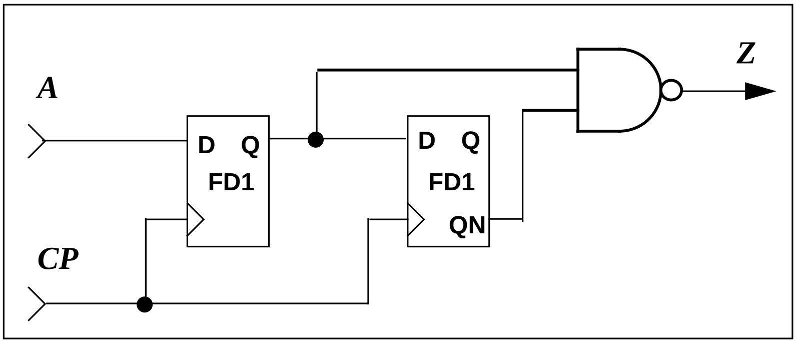


Figure 4.2
Synchronous Pulse Generator

In this synchronous pulse generator, the Z output is the decoded output or the flip-flops. This pulse will always be exactly one clock cycle long. The decoded output (Z) is used as the new clock signal (see Figure 4.3).

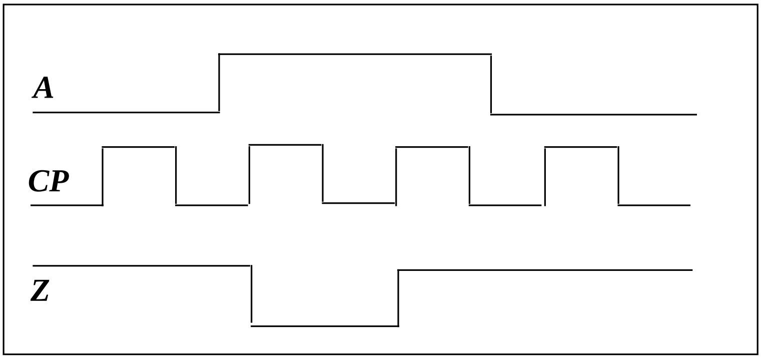


Figure 4.3
Synchronous Pulse Generator Timing Diagram

CHAPTER 4 - SECTION 2
Avoid “Glitch” Generators

An inconsistency in signal timing produces what is conventionally called a “glitch”. Figure 4.4 shows how a glitch might look compared to a signal that is behaving predictably.

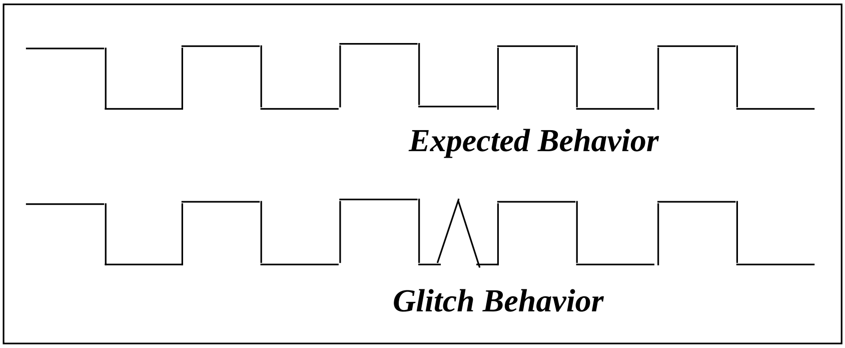


Figure 4.4
Glitch Behavior

Some circuits, such as the 2-to-1 multiplexer in Figure 4.5, invariably cause glitches.

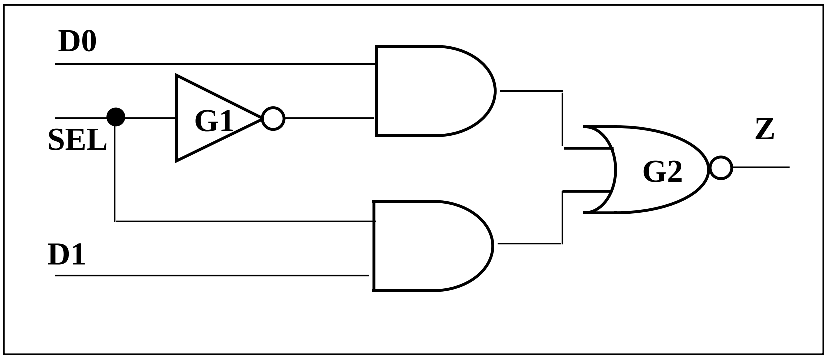


Figure 4.5
Glitch Generating Mux

Figure 4.6 is the timing diagram for the circuit in the next example. To see how the glitch is generated, assume that inputs D0, SEL, and D1 are initially high. The output therefore is low. When set makes a high-to-low transition, neither D0 nor D1 is selected for a period of time. During this time, the output Z will glitch high regardless of the states of D0 and D1. The glitch width is equal to the time it takes G1 in Figure 4.5 to propagate a 1 to its output.

Note that the low-to-high transition of SEL in Figure 4.6 will not cause a glitch on G2’s output.

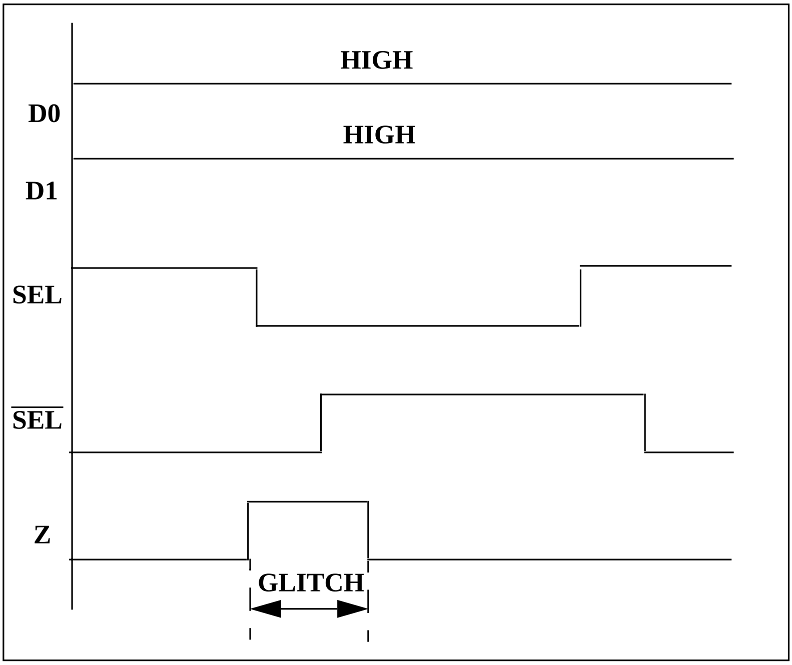


Figure 4.6
Timing Diagram of Mux Output Glitch

It is safe to use the output of a potential glitch generator to drive the data input of a synchronous network provided the output has reached its steady state before the active edge of the clock is applied. However, it is a bad practice to use such a circuit to drive a clock line, a set line, or a clear line because the potential glitch may cause a false clock signal or reinitialize internal registers. Figure 4.7 shows the 2-to-1 multiplexer driving a data input.

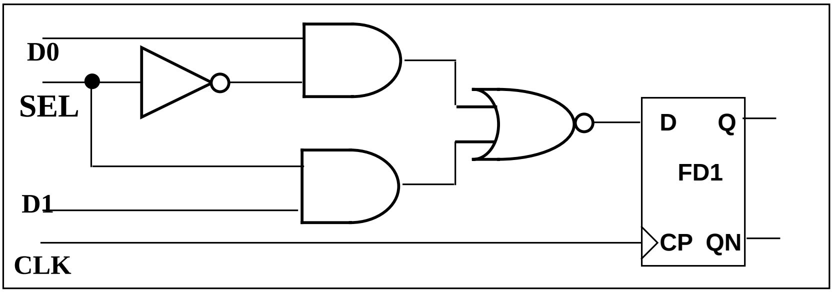


Figure 4.7
The Glitch Doesn't Matter Here

CHAPTER 4 - SECTION 3
Use Care with Gated Clocks

Designers must use great care in the use of gated clocks since improper timing of the gated signal can generate a false clock, causing the flip-flop to clock in the wrong data. The circuit shown in Figure 4.8 has a gated clock signal.

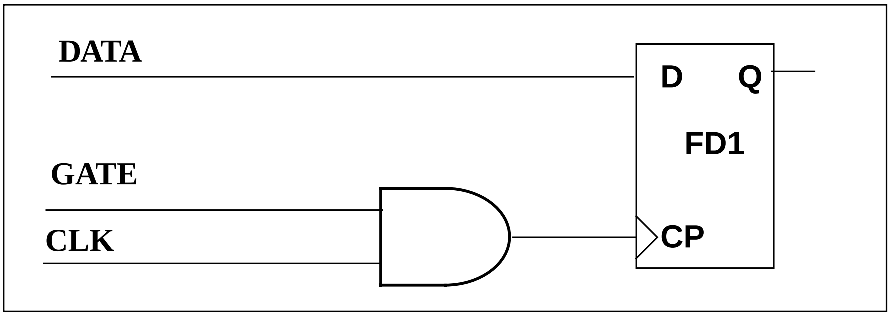


Figure 4.8
A Gated Clock

Again, gated clocks must be used with extreme caution, and require very restrictive timing tolerances between the clock and the gate signal. If you are unable to guarantee such tolerances, do not use gated clocks. Moreover, because of skew between different local clocks, gated clocks may cause hold-time violations. Interfaces between local clock signals should be carefully monitored.

The alternative design shown in Figure 4.9 is perhaps a safer implementation of the same logic function since it eliminates the gated clock. When GATE is low, the data is allowed to circulate continuously with each clock pulse. As soon as GATED goes high, new data enters the flip-flop on the next active clock edge. (Remember to allow proper setup time.)

Of course, today, one of the main advantages of gated clocks is that they can reduce power dissipation overall - sometimes considerably! So we cannot discard this design technique carte blanche, but it is a more esoteric technique that requires some design expertise.

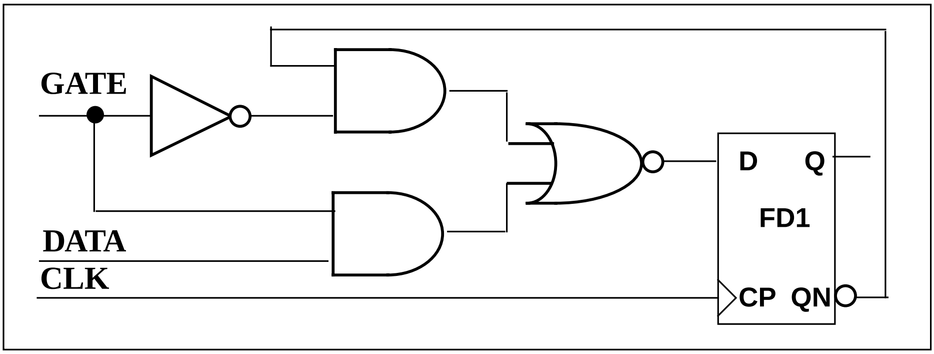


Figure 4.9
Controlling Data Instead of Clock is Safer

CHAPTER 4 - SECTION 4
Avoid Race Conditions

Race conditions occur when the same signal follows two or more paths having different delays to a common circuit element, or when two signals caused by the same event (say a clock) follow different paths to the same circuit element. (Race conditions sometimes create a condition called “reconvergent fanout.”) Avoid designs which function efficiently only if there is a predictable interval between signals traversing different delay paths. This interval is seldom predictable; pulse widths and delays vary widely due to variations in layout, processing, power supply, and temperature. Even where multiple paths have the same number of elements, variations in fanout and wire delay will cause timing differences in the different paths. As a result, a circuit that is subject to race conditions is bound to be unreliable. Examples of race conditions to be avoided are shown in Figures 4.10 through 4.13.

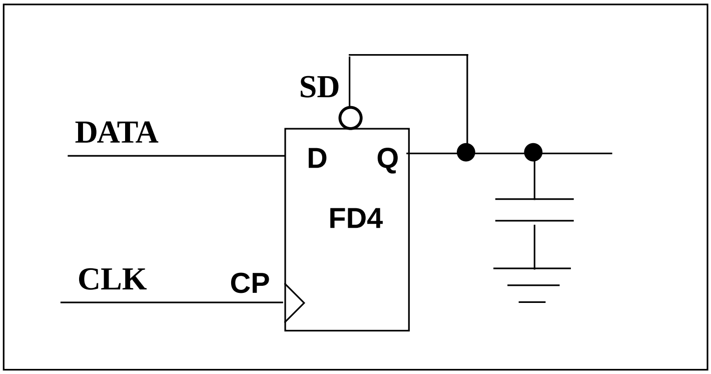


Figure 4.10
A Race Condition

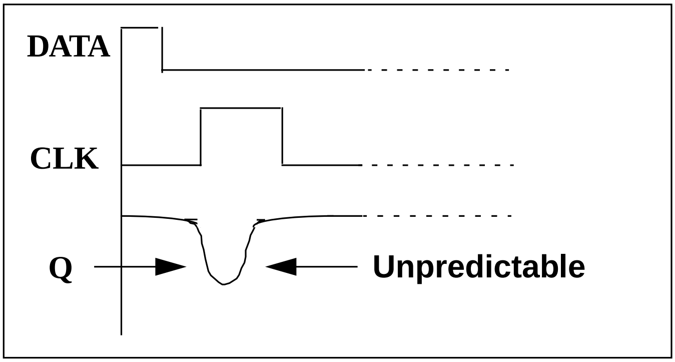


Figure 4.11
Waveforms for Figure 4.10

In Figure 4.10, the Q output sets the flip-flop when a zero is clocked into the flip-flop, creating the narrow negative pulse on the Q output as shown in Figure 4.11. The output pulse width depends on the Q output loading. Under heavy loading, the pulse may set the flip-flop, which is close to its source, but then be so attenuated by a long wire that it is invisible to its other fanouts.

Under ideal conditions, the circuit in Figure 4.12 will generate a narrow, positive pulse on each active clock edge. However, if the clock line is part of a high-fanout network with a slow ramp rate, then the rest signal may reach the flip-flop before the active clock edge, with the result that no pulse will be generated. To see how this can happen, look at the timing diagram in Figure 4.13. The inverter threshold voltage may be slightly different from the flip-flop's threshold voltage (in this case, it is slightly lower). As a result, the inverter goes low and resets the flip-flop before the CLK signal reaches the flip-flop's clock threshold.

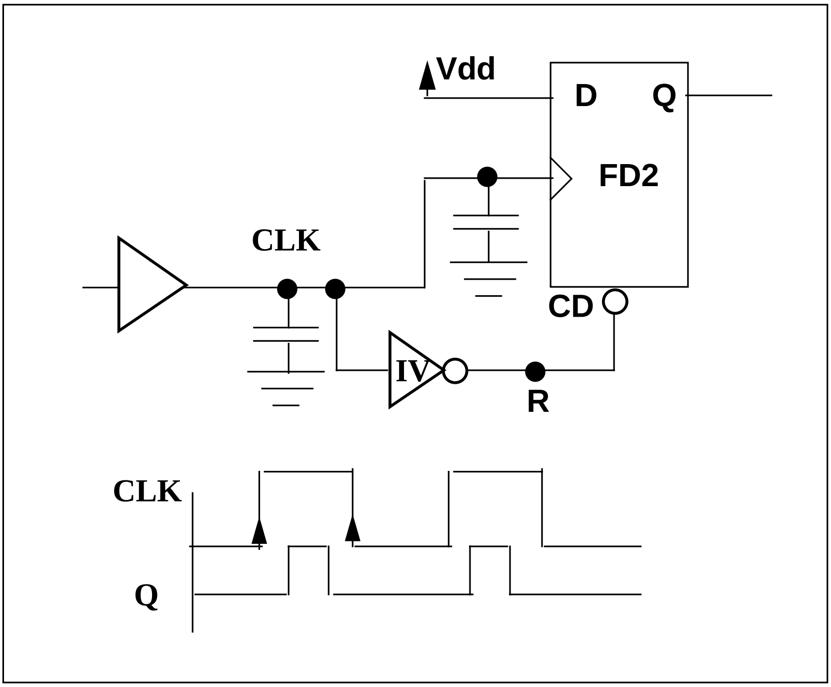


Figure 4.12
Another Race Condition

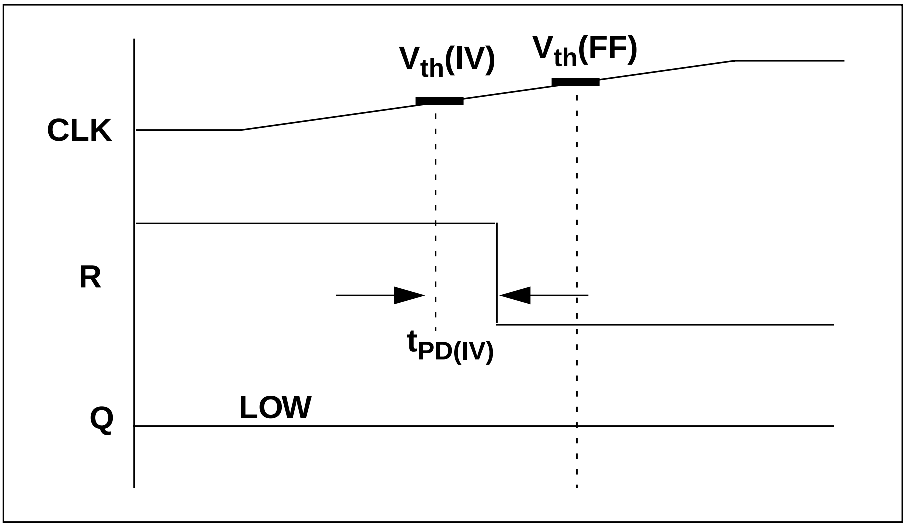


Figure 4.13
Waveforms for Figure 4.12

CHAPTER 4 - SECTION 5
Avoid Floating Nodes on Internal Three-State Buses

A floating node occurs when all of the buffers driving a bus are disabled, as shown in Figure 4.14. In general, floating nodes should not be used in LSI Logic ASICs. (Note, however, that a node floating for a period of time equal to the delay of a 2-input NAND gate with a fanout of four IV gates and two millimeters of wire is acceptable.) You can avoid floating nodes on internal three-state buses by adding a buffer to drive the bus when all of the other buffers are off. (See Figure 4.15.)

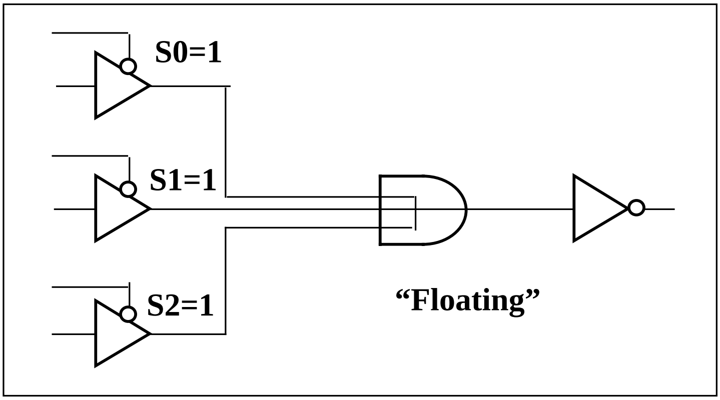


Figure 4.14
Avoid Floating Nodes

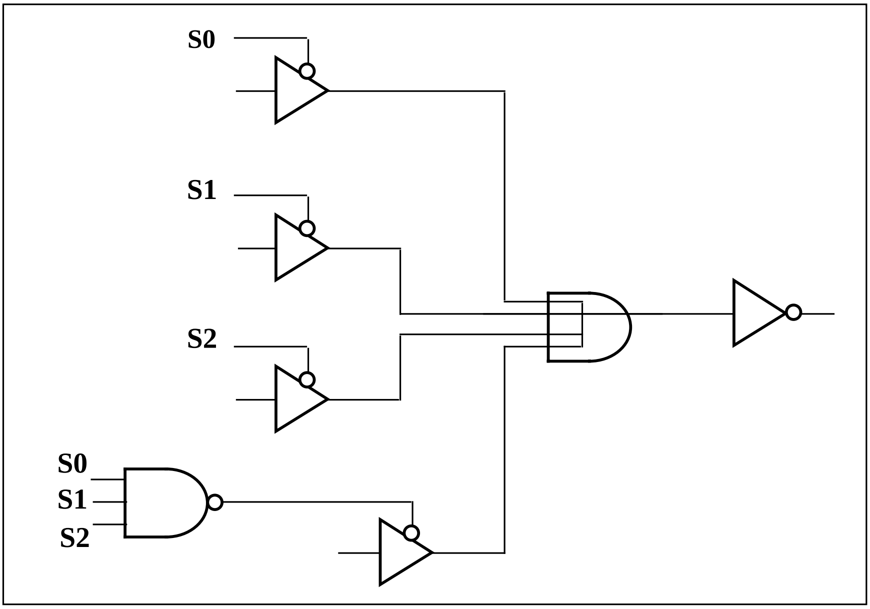


Figure 4.15
Floating Node Avoided w/ Extra Buffer/Decode

A floating node will cause static current to flow on a driven gate (such as the inverter in Figure 4.14), since the P- and N-channel devices of the inverter can be at least partially ON at the same time. A floating node may also cause the inverter gate to oscillate, generating excessive noise.

Again, to correct this situation, you can add another buffer to drive the bus, as shown in Figure 4.15.

The enable signal for the extra buffer in Figure 4.15 is equal to the NAND function of the three original enable signals. Decoding these multiple enable signals ensures that at least one buffer drives the bus at all times. Take care to control the enable signals so as to prevent bus contention. In some logic schemes, a bus may float for a brief time, but this should be minimized as much as possible. LSI Logic does not allow bus contention on our test hardware.

CHAPTER 4 - SECTION 6
Use Bidirectional Output Buffers with Pull-Up or Pull-Down Resistors to Increase the Noise Margin

Figure 4.16 illustrates the use of bidirectional output buffers with pull-up or pull-down resistors.

Pull-up and pull-down resistors tend to pull the signal away from the threshold of the pad in question making noise less likely to propagate back into a design, and thereby tending to reduce the possibility of data corruption in storage elements.

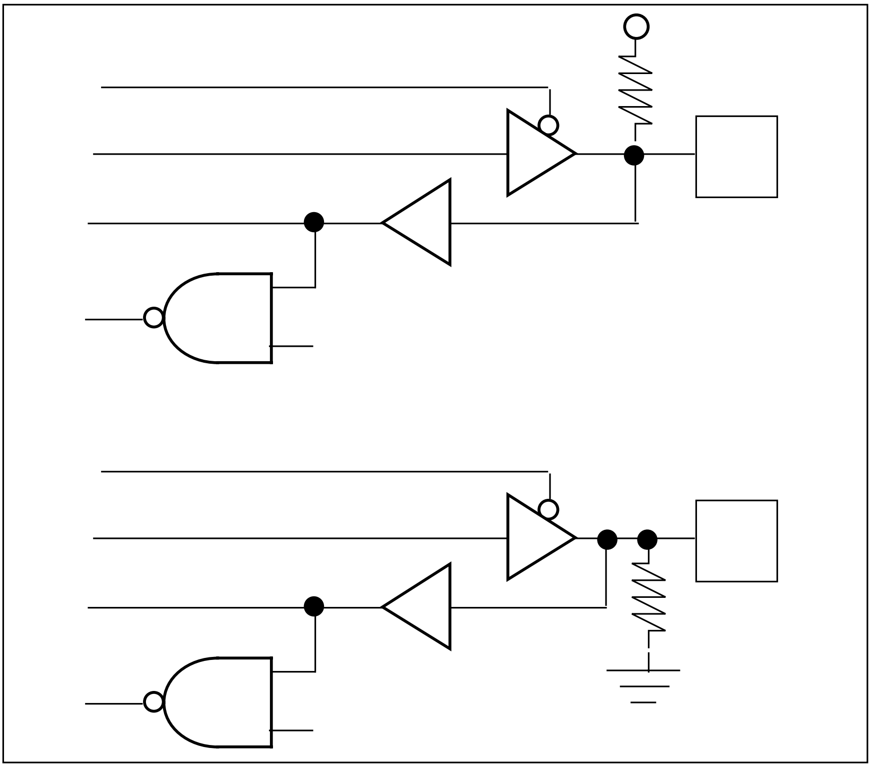


Figure 4.16
Bidirects with Pullup and Pulldown

Do not use the input half of a bidirectional buffer to drive a clock line. Doing so will result in clock problems in your design. In most LSI Logic ASICs, the typical pull-up or pull-down resistor is in the range of 80K ohms. It takes tens of microseconds for an 80K pull-up resistor to pull a pad high. During this time, any noise that occurs a the pad may propagate to the clock input of internal flip-flops, clocking in bad data. An on-chip, pull-up resistor should not be relied upon to produce a correct input state for a floating signal. That is not the purpose of these resistors. Use off-chip, high-precision resistors to set bus voltages.

CHAPTER 4 - SECTION 7
Avoid Excessive Fanout

A logic element driving many elements is said to have a high fanout and drive a heavy load. In an HCMOS circuit there is no theoretical limit to how heavy a load a gate can drive. If a gate fans out even to 1000 places, the output will eventually reach its final state, albeit after a very long delay. For practical purposes, the rise or fall or an internal node ought not to exceed about 1.0 ns under nominal conditions.

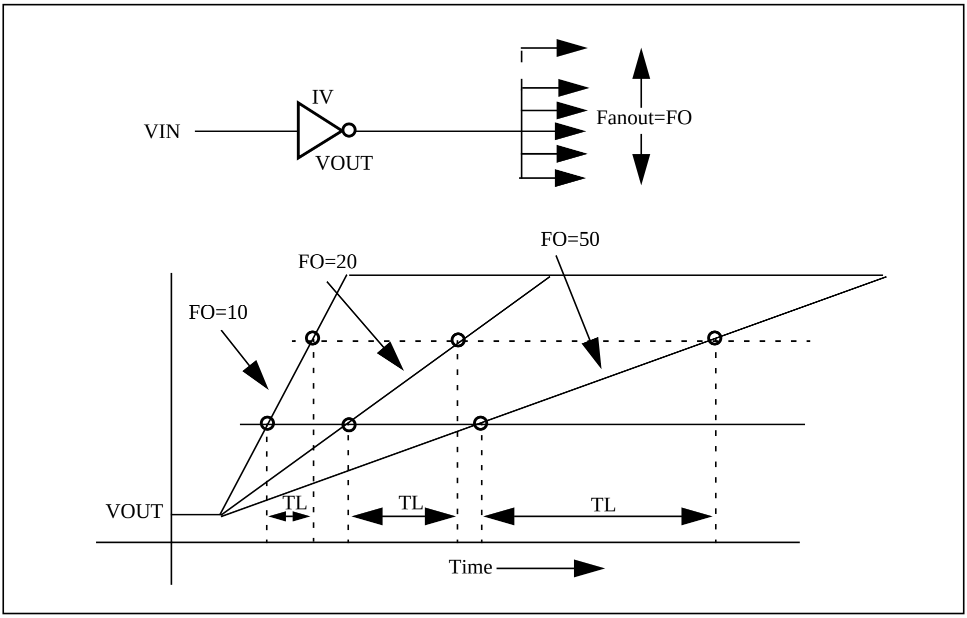


Figure 4.17
Excessive Power Consumed Due to Excessive Fanout

The graph in Figure 4.17 shows that the ramp rate of a gate becomes longer as the fanout increases. As the slope becomes longer, the driven macrocells spend more time (TL) in the active region, where both N- and P- channel transistors are ON, and consume excessive power as a result. This situation is undesirable and should be avoided. Do not exceed recommended maximum fanouts for library elements.

CHAPTER 4 - SECTION 8
Avoid "Dangerous" Decoding

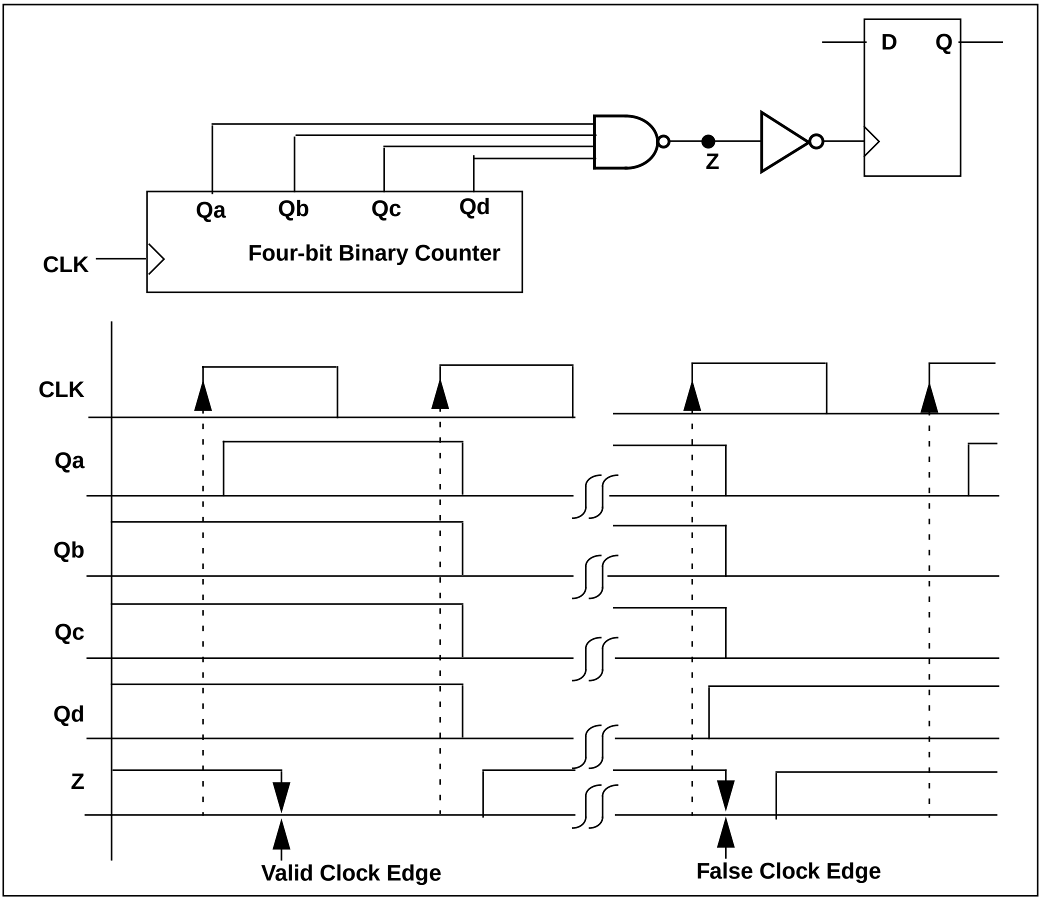


Figure 4.18
Dangerous Decoding of Terminal Count

Use either Johnson Counters or separate flip-flops to decode terminal counts. Figure 4.18 shows a procedure which, though convenient, can cause false clock signals. In Figure 4.18, the terminal count of a binary counter is detected with a NAND gate. The decoded signal is then used to drive a clock line. To avoid false clock signals, use a Johnson Counter or a Gray Code Counter instead of a binary counter; or else use separate flip-flops for the decoded signal.

In Figure 4.18 the internal loading on each of the Q outputs of the counter varies, as do their delays. As a result, the inputs to the NAND gate stabilize at different times. The output of the NAND gate can switch between a one and a zero state, creating a false clock signal, until it finally stabilizes to its steady state.

The same logic can be implemented with a Johnson counter rather than a binary counter. Figure 4.19 shows the logic diagram of a Johnson counter. Any terminal count can be decoded with a 2-input NAND gate. Since only one flip-flop changes state during a single cycle, no glitch is generated.

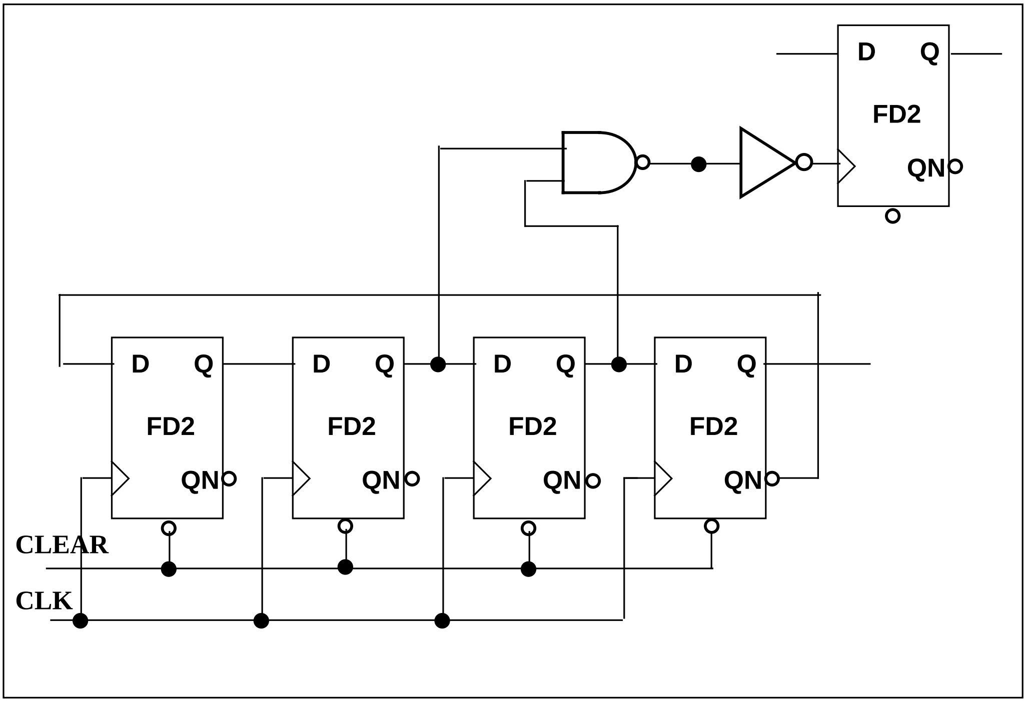


Figure 4.19
Decoding a Johnson Counter

The outputs of a binary counter can be decoded by decoding one count before the terminal count and running the decoded signal into a separate flip-flop, as illustrated in Figure 4.20. The new signal coming from the output of the single flip-flop will have a clean waveform.

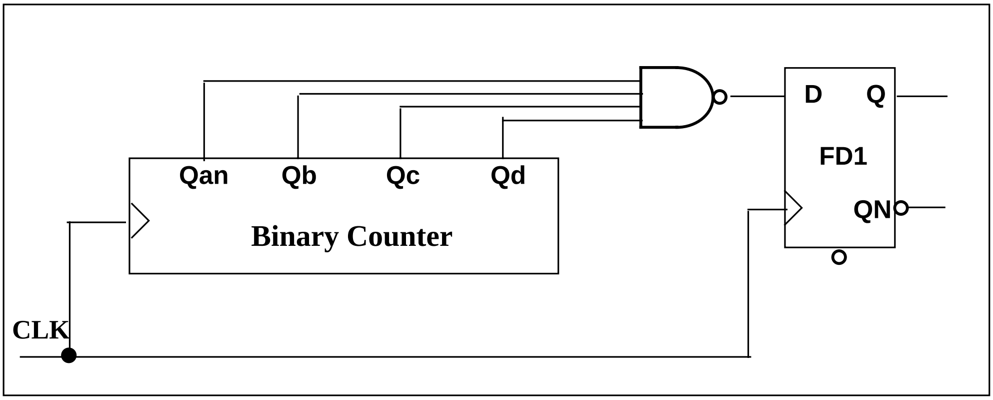


Figure 4.20
Alternate Glitch-Free Terminal Count Decode

CHAPTER 4 - SECTION 9
Do Not Use Unbuffered Transmission Gates

Transmission gates have the advantage of higher speeds into light loads, but they must be used judiciously. Input signals of the transmission gate are briefly shorted together, and when at least one of the input signals to a transmission gate multiplexer is the output of an unbuffered storage element (see Figure 4.21), this may result in corruption of data in the storage element. All LSI Logic proprietary flip-flops and latches have buffered inputs and outputs to prevent such a malfunction. Customer-designed storage devices, however, may not have this buffering protection. Do not build this reliability problem into your design. So, if you must use unbuffered storage elements, do not use transmission gate multiplexers. Conversely, if you use the transmission gate MUX's, then use buffered outputs on storage devices that feed into these MUX's.

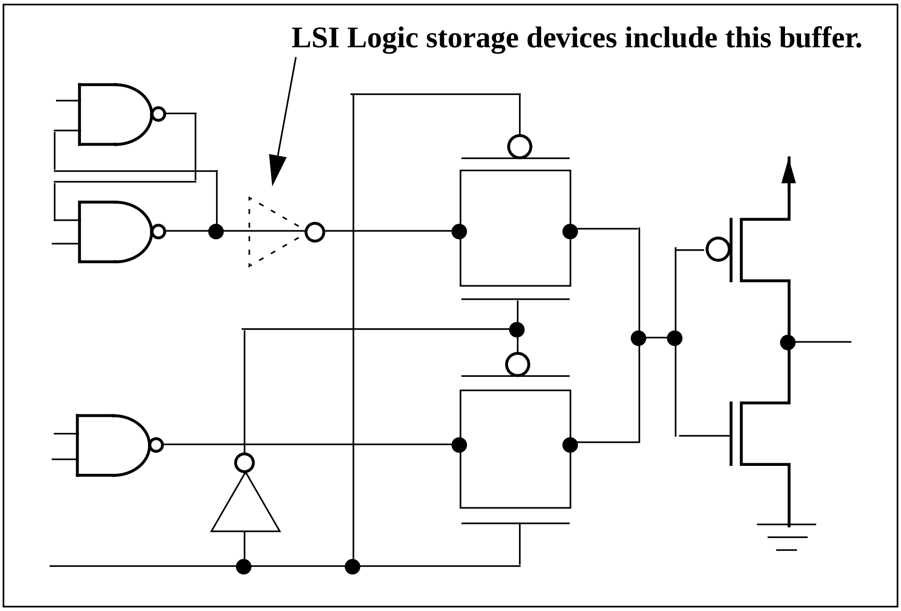
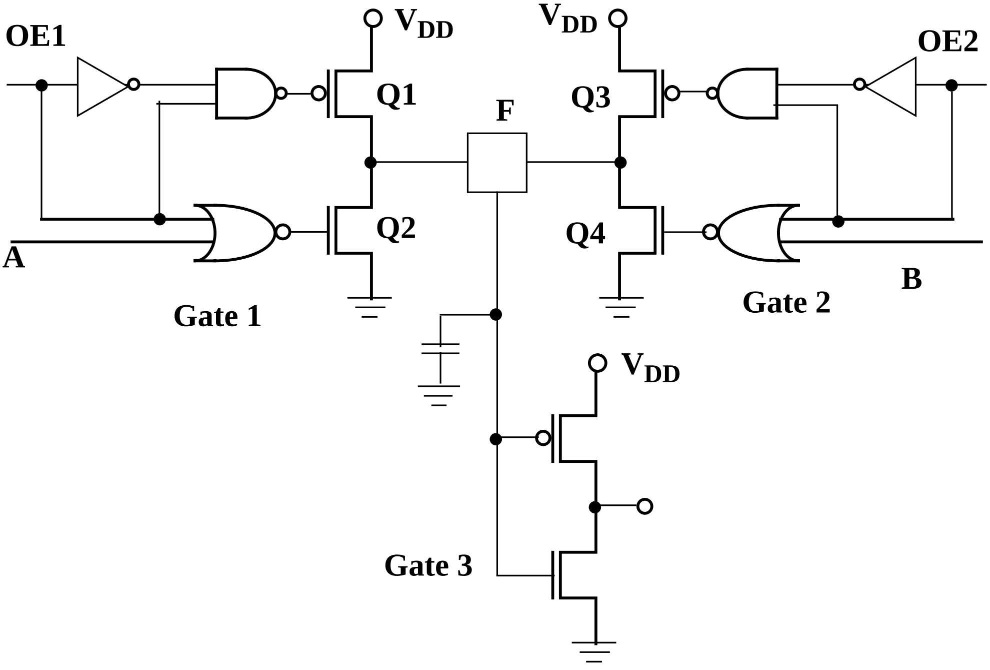


Figure 4.21
Faulty Use of Transmission Gate

CHAPTER 4 - SECTION 10
Avoid Bus Contention

Bus contention occurs when a bus is driven from different sources at the same time. There are two kinds of bus contention - internal and external. Internal bus contention can occur on internal three-state buses. External bus contention occurs when an output buffer is enabled, and is outputting a logical value that differs from that of the external driving source (see Figure 4.22, Case I).



Case 1: Gate 1 driving low; Gate 2 driving high

- **DC current path thru Q3 and Q2 to VSS (ground)**
- **Both transistors handle an “abnormal” current load.**
- **Voltage at “F” is not well-defined**
- **Output state of gate 3 is unknown**

Case II: Gate 1 driving low; Gate 2 driving low

- **Point F at “Logical 0”**
- **Useless for test purposes**

Figure 4.22
Avoid Bus Contention

In Case I of Figure 4.22, with Gate 1 driving low and Gate 2 driving high, a DC current path exists through the upper transistor of Gate 2 and the lower transistor of Gate 1 to VSS (ground). Both transistors handle abnormal current loads, and the voltage at point F is not well-defined. The output state of Gate 3 is questionable. External bus contention can be avoided by monitoring three-state control signals to determine when the external source is allowed to drive the pin.

In Case II, with Gates 1 and 2 both driving low, point F is at a logical 0, but the situation is useless for test purposes. If one of the drivers is faulty, the fault will not be detected, and weak output current drive would be asked by the other driver. Bus contention can be avoided by three-stating one buffer whenever the bus is being driven by the other buffer or external signal. Bus contention can be avoided by allowing a given bus to be driven by only one driver at any one time. Bus contention will not be allowed during testing. Therefore, bus contention must be eliminated during simulation in order to avoid fatal errors when converting simulation vectors to tester format.

CHAPTER 4 - SECTION 11
Buried Buses: Sometimes Yes, Sometimes No

“Buried” buses that route to many places tend to increase parasitic capacitance, and result in poor floorplans, inefficient chip utilization, routing problems for surrounding logic, and faulty die size estimates. Note that bit-slicing datapaths will help minimize such buses. Keep buses that route to many places at the top levels of your design hierarchy so they can be easily seen and planned for.

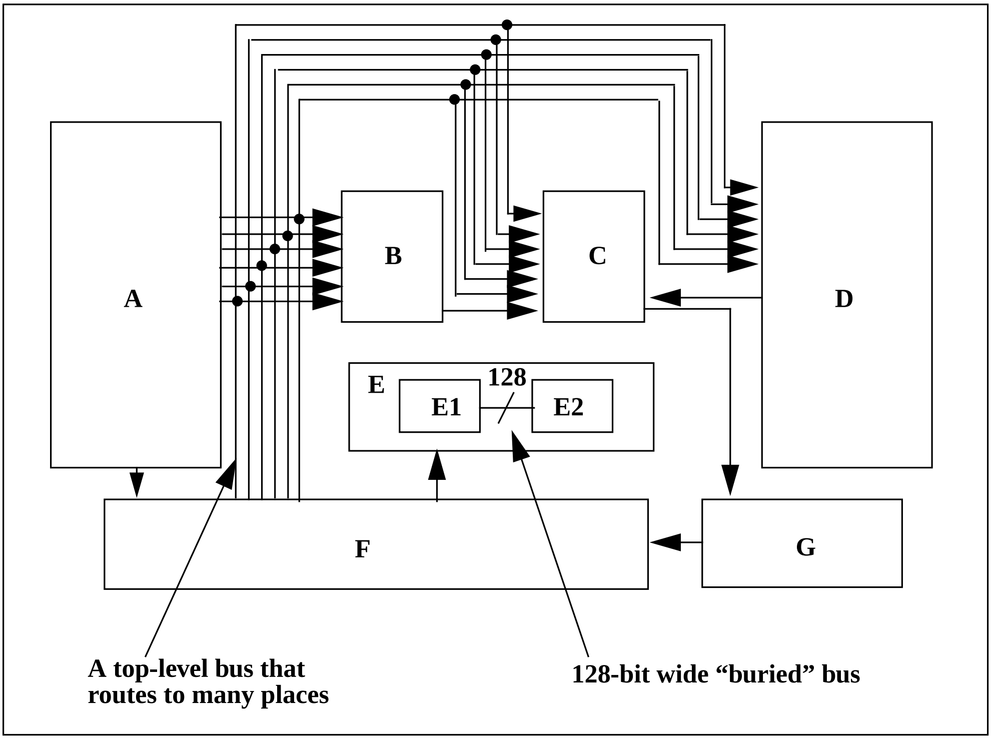

A
B
C
D
E
E1
128
E2
F
G
A top-level bus that routes to many places
128-bit wide “buried” bus

Figure 4.23
Two Types of Buses

Sometimes buses can be legitimately buried if they are local buses that basically connect one or two tightly coupled blocks. Under these circumstances it is better to bury the bus so as to maintain close placement of the tightly coupled blocks.

Figure 4.23 shows an example of each of the two types of buses.

CHAPTER 4 - SECTION 12
Follow I/O Guidelines Correctly

A generic LSI Logic rule is that every VSS pad can support only a certain number of standard output buffers (4 mA equivalents), depending on what package you are in. But this rule should not be misinterpreted. The unacceptable design in Figure 4.24 below shows 2x*N* 4 mA equivalents between the two VSS pads. In the G12 technology, it may not be unrealistic to have one V_{DD} and V_{SS} pad pair for every six to eight 4mA buffers.

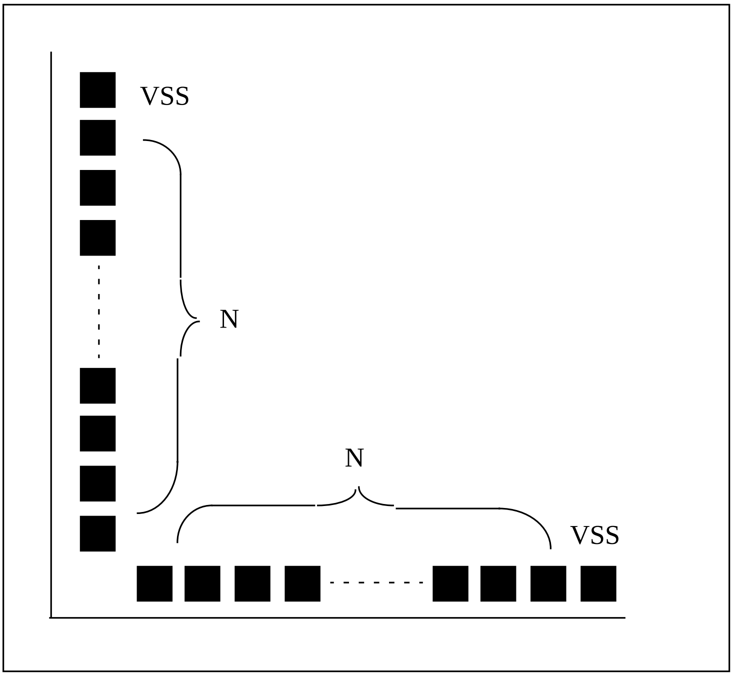


Figure 4.24
Too Many Outputs Between Power/Ground Pads

Figure 4.25 shows the proper rule interpretation - *N* 4 mA equivalents per VSS pad (either per side), and no more than *N* 4 mA equivalents between VSS pads.

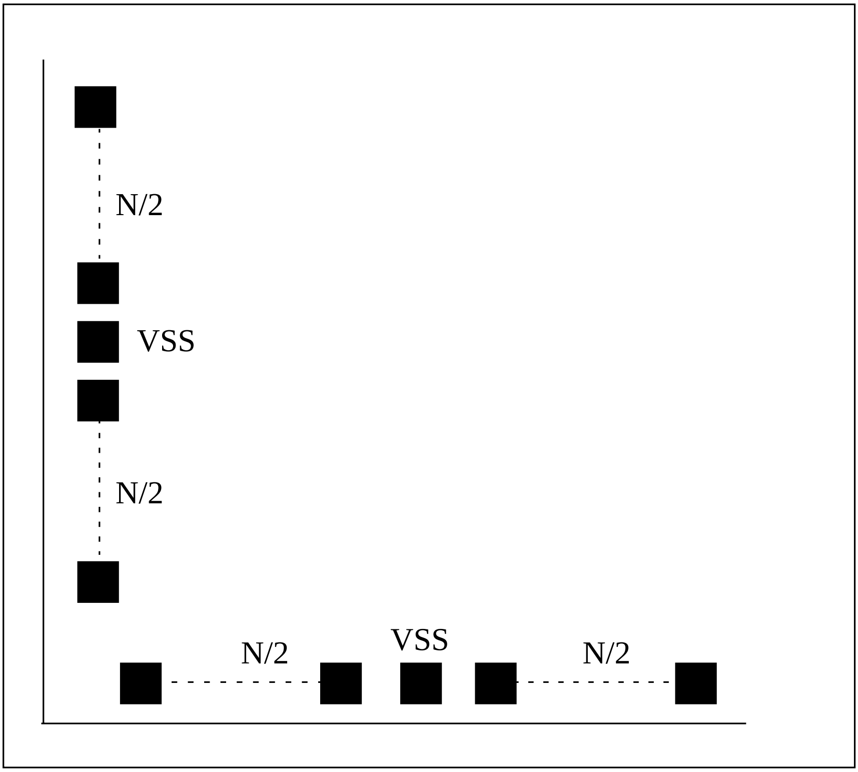


Figure 4.25
Enough Ground for the Outputs

Do not place clock signal pins, reset pins, preset pins, or other major control signals between high-drive output buffers and VSS (see Figure 4.26).

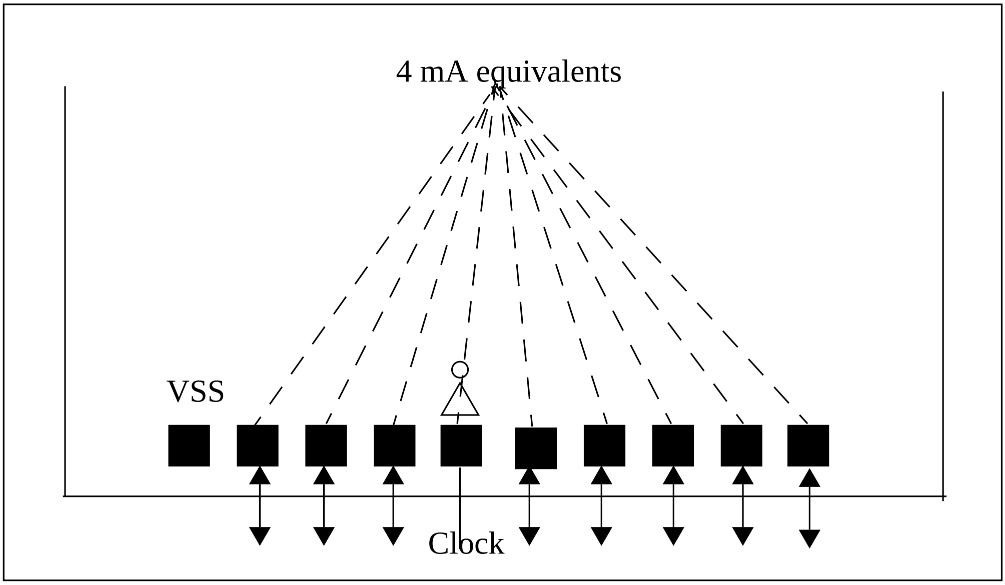


Figure 4.26
Poor Placement of a Clock

In Figure 4.27 the clock pin is properly placed, with a VSS pin between the clock pin and the high-drive output buffers, or with the clock pin between two VSS pads. Note that inputs do not cause nearly as much noise as outputs because they do not drive huge loads, so having other inputs near the clock is not a problem.

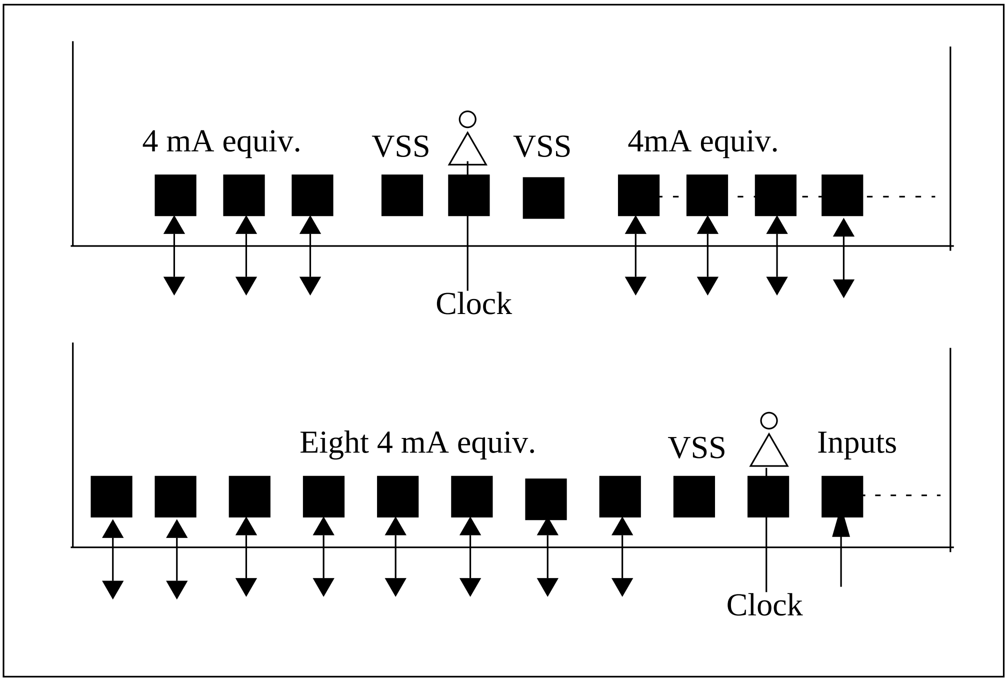


Figure 4.27
Proper Placement of a Clock

CHAPTER 4 - SECTION 13
Power Considerations

In some ASIC designs, minimizing power consumption is a high priority. In others, it is not. In any case, it is always necessary to know in advance how much power a chip will consume, since this affects the operating temperature, which in turn affects:

- ❑ Speed
- ❑ Package selection
- ❑ FIT rate

Speed

CMOS silicon chips slow down as they get hotter. Proper delay prediction must therefore take junction temperature into account. And, make no mistake: any temperature specified to any delay prediction tool is a junction temperature, not an ambient temperature. Junction and ambient temperature are related by the equation $\mathbf{T_j = (P_D)(\theta) + T_a}$, where $\mathbf{P_D}$ is power in Watts and θ (Theta) is the thermal resistance of the package in degrees C per Watt. Since speed is a function of $\mathbf{T_j}$ rather than $\mathbf{T_a}$, and the program does not know Power or Theta, the temperature cannot be ambient.

Package Selection

From the equation in the previous paragraph it is obvious that, to reduce junction temperature, you must reduce $\mathbf{T_a}$, $\mathbf{P_D}$, or θ. Perhaps the simplest of these, from a design perspective, is to select a package with a lower thermal resistance, so that heat generated by the chip flows more easily to the ambient environment. Usually, the disadvantage of a more efficient package is higher cost.

FIT rate

Many designers operate under the mistaken assumption that it is necessary only to cool a chip sufficiently for it to run at the desired speed. Unfortunately, this ignores the fact that hotter chips burn out sooner. FIT stands for Failures In Time, and is a measure of how many chips will fail for each ten thousand hours of operation. In general, a ten degree C increase in junction temperature will double the FIT rate. For a low volume design with a finite life expectancy, this may not be a problem. But for any design requiring high reliability, or any high volume design requiring moderate reliability, it may be necessary to reduce the operating junction temperature well below the rated maximum. A designer who fails to realize this will end up with a design having very high cost of maintenance and lots of angry customers. A designer who realizes this too late will end up with a more expensive system (either because of increased packaging cost [lower θ] or increased system cooling requirements [lower T_a]) and, a design that is over-specified, since it will be capable of higher frequency due to the lower operating temperature. It is therefore crucial for successful design that the appropriate FIT rate be specified early in the design cycle, and the desired operating temperature be calculated with the FIT rate in mind.

Contributing factors

CMOS circuitry consumes power in two ways: by capacitance and by resistance. The former is very frequency dependent; the latter is less so. When a CMOS circuit element changes state, it is charging and discharging capacitors. The more often the capacitors are charged and discharged (that is, the higher the operating frequency), the higher the power consumption.

There are simple equations for estimating power based on frequency, but applying them to a large system will result in a very high estimate. This is because a typical digital circuit element does not change state on every cycle. In any power estimation program, there are factors for both operating frequency and a rather nebulous “activity factor”, that is, the average number of circuit elements switching state in any given clock cycle.

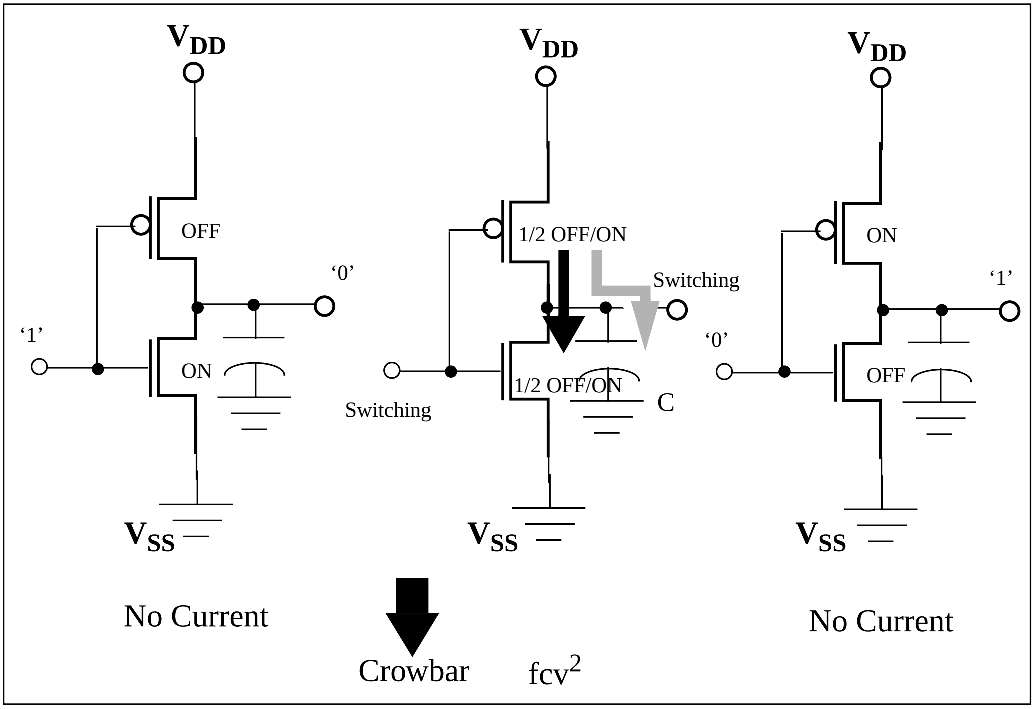


Figure 4.28
CMOS Power Consumption

There are two types of resistive power consumption in CMOS. The first is the so-called "crowbar" current, named after the habit of old-time power generation workers of deliberately causing a short-circuit with a crowbar to trip the breaker when de-powering a transmission line, to avoid dangerous power arcs in unpredictable places. When a CMOS circuit element, say an inverter, changes state, it is momentarily in the linear region, and both N-channel and P-channel devices are partially on. This creates a resistive path directly from power to ground through both transistors. The longer the ramptime on the input, the more time the device spends in the linear region. This means that one way of reducing power consumption is to ensure that there are no excessive ramptimes.

There are many reasons why all excessive ramptimes should be fixed. A long ramptime not only causes the circuit elements it drives to consume more power, it may also cause electromigration which can lead to premature and catastrophic circuit failure. In addition, a long ramptime will have a large effect on delay prediction. A long input ramptime will cause a long output ramptime on any given circuit element, so fixing one long ramptime will likely fix others as well. Also, a long ramptime may well fall outside of the characterization range of either the driving element or the driven element, which means that the delay predictor is basically making a guess with respect to these delay values.

The second type of resistive power consumption in CMOS is static power from analog (that is, non-digital) circuit elements. There may be a large number of analog circuit elements in a modern digital CMOS chip, from PLLs (phase- locked loops), to high-speed, low voltage-swing I/Os, to sense-amplifiers in high-density memories. All of these elements consume static current independent of operating frequency.

Conserving power

As mentioned earlier, reducing power consumption will make your chip run faster and last longer, may reduce the required size and complexity of both your power supply and your cooling system, and perhaps enable you to use a less expensive package. And, of course, reducing power consumption is absolutely critical in battery powered applications.

One of the easiest ways to reduce power consumption is to gate clocks; unfortunately, this can very easily cause testability problems, especially when it comes to scan and the associated ATPG (Automatic Test Pattern Generation). There are now programs available which will automatically insert circuitry that not only gates clocks, but also ensures that scan testability issues are addressed.

Another way of reducing power consumption is to enable power-hungry elements such as analog circuits and high-density RAMS only when necessary.

Power distribution

When asked to name the most important interconnection on a chip, most logic designers will immediately answer, “clock.” Actually, the answer is power and ground. Logic designers tend to take power and ground for granted; it doesn't even show up on most schematics. However, when routing an ASIC, power and ground come first.

Current LSI Logic technologies make use of a four-bus system in which power and ground for large geometry I/O transistors (Vdd and Vss) are separate from power and ground for the core (Vdd2 and Vss2). These run around the periphery of the die, with Vss and Vdd on the outside and Vdd2 and Vss2 on the inside.

The goal of routing power and ground is to ensure their even distribution to all points on the die. Unfortunately, today’s technologies don't have enough routing resources to create dedicated power and ground planes on-chip, so a certain amount of “droop” must be anticipated. Since power and ground buses are located at the periphery of the die, the greatest power droop will be concentrated in the center of the die. This is one reason why memories, which cannot tolerate as much power droop as other logic elements, should be placed near the edge of the die.

Another result of not having dedicated power planes is that power flowing into and out of the core region must all flow to the allotted Vdd2 and Vss2 pins. If care is not taken to provide a sufficient number and proper location of these pins, a great many electrons can all funnel through an insufficient cross-section of metal, which causes metal migration over time. Eventually, these metal traces will blow like a fuse. This is another example of electromigration.

Power isolation

Even when CMOS ASICs were 100% digital, it was sometimes necessary to prevent noise from high current switching areas of the die from interfering with more sensitive portions of the circuit. Until the G12 technology, all LSI Logic chips had clock drivers and output buffers sharing the same power and ground. Since this could introduce unacceptable noise into the clock line, it was often necessary to take drastic measures to isolate power and ground for the clock from power and ground for the output buffers. These drastic measures involve cutting the power and ground rings on either side of the selected circuitry and providing the isolated segment with its own power and ground pins. This becomes even more important when what we need to isolate is not merely a clock driver, but an analog circuit such as a PLL or A/D converter.

A power ring segment isolated by power cuts can also be supplied with a voltage level different from that on the rest of the die. The affected area might be supplied with a higher voltage level to increase its performance, or with a lower voltage level to conserve power, or simply to make an interface which is compatible with a more exotic system.

Power and Testability

All LSI Logic ASICs are tested for leakage current, both for reliability reasons (a chip with elevated leakage current will fail after fewer hours of operation compared to a chip with normal leakage), and as a means of defect detection. A typical leakage current measurement is on the order of 100µA. A typical high density memory can draw almost 100mA, a difference of three orders of magnitude. To ensure that structures which can draw static current can be switched off, all designs in current LSI Logic technologies must include a test pin connected to a special buffer called IIDDTN. This cell's primary purpose is to disable pullup and pulldown "resistors" (actually transistors), but it can and should also be used to provide a means of putting all circuit elements capable of drawing static current into their power-down modes.

5

Designing for Testability

CHAPTER 5 - SECTION 1
Testing Redundant Logic

Redundant logic, such as that shown in Figure 5.1, may be used to enhance reliability. However, unless the redundant logic can be tested uniquely, the intended enhancement may prove illusory. Figure 5.2 shows a testable version of this circuit.

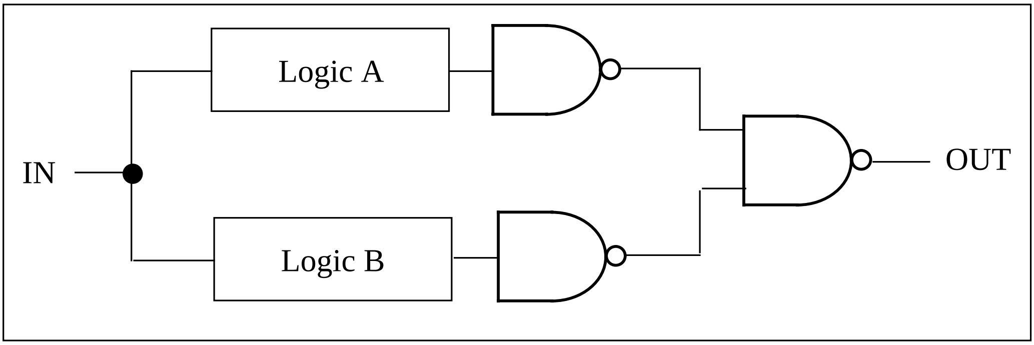


Figure 5.1
Untestable Redundant Logic

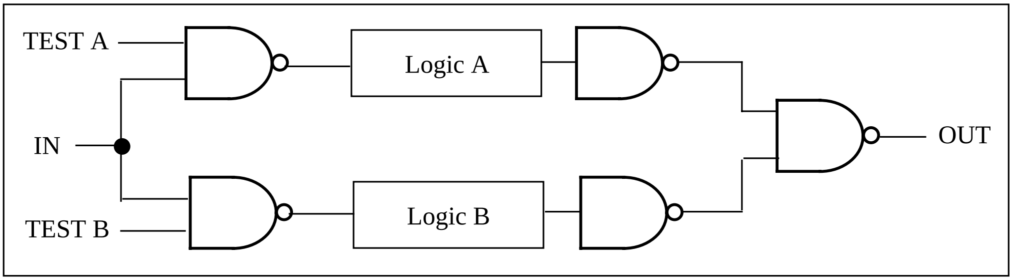


Figure 5.2
Testable Redundant Logic

CHAPTER 5 - SECTION 2
Use Combinational Logic

Combinational logic is easier to test than sequential logic. in combinational logic, the outputs are always a function of the current inputs. In sequential logic, the outputs depend on both the current inputs and the outputs of storage elements driven by inputs preceding the current ones. For example, in the multiplexer in Figure 5.3, all possible states can be exercised by presenting permutations of 1’s and 0’s on the input pins. The test vectors will not exceed 2^9 power (512), and will cover all possible detectable stuck-at faults.

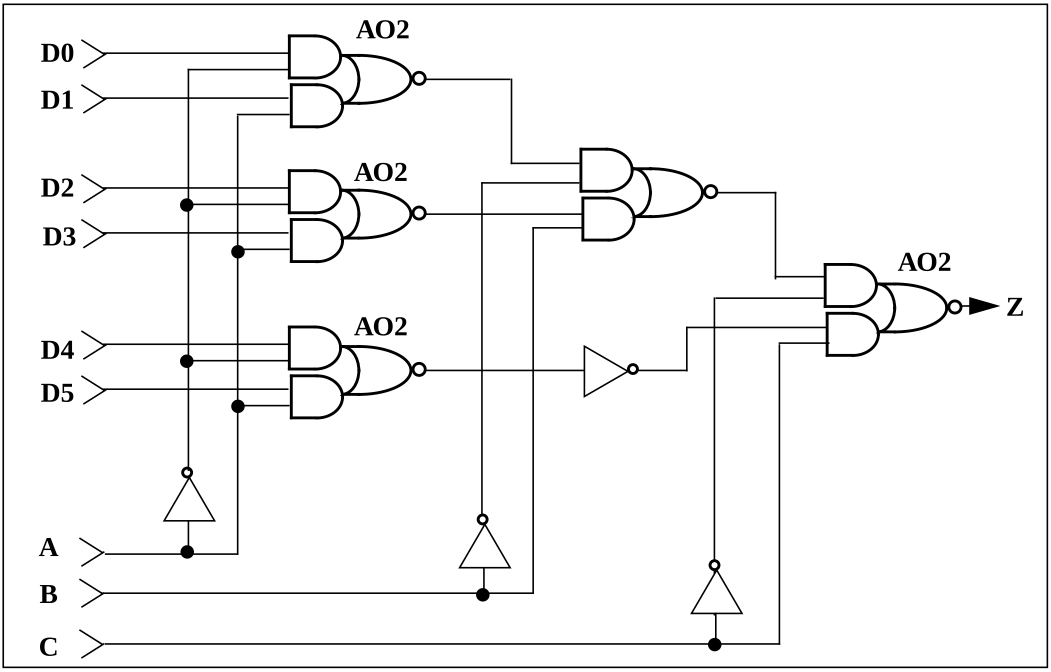
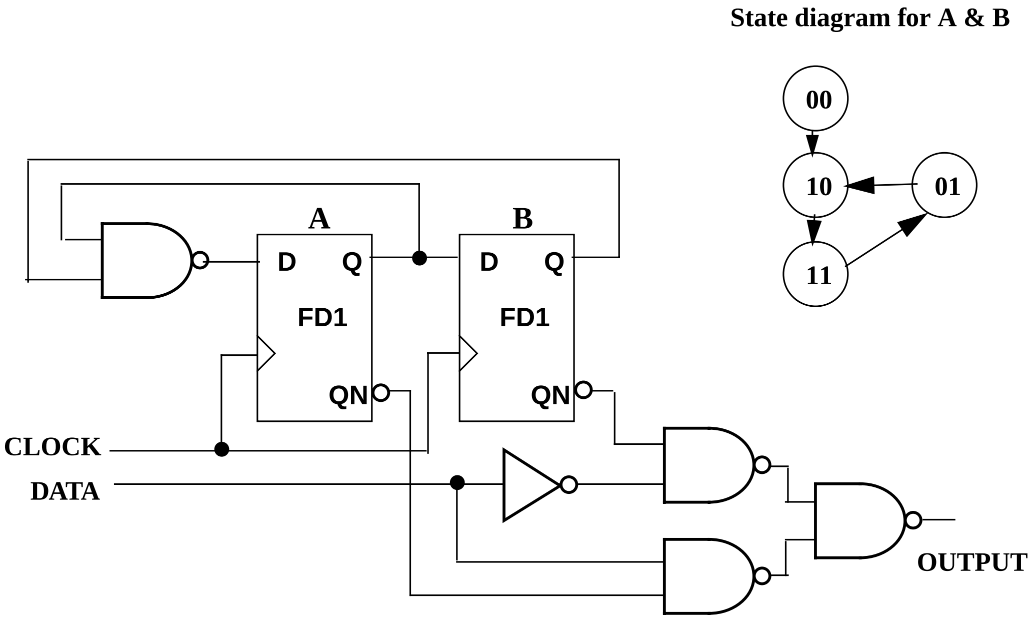


Figure 5.3
Combinational Logic is Easy to Test

CHAPTER 5 - SECTION 3
Add Test Lines to Sequential Logic

Sequential logic is more difficult to test than combinational logic because it contains storage elements. You can make sequential logic more testable by adding test lines to logically break feedback loops between flip-flops.

Figure 5.4 shows a segment of untestable sequential logic. The circuit’s initial output condition is unknown because of the feedback loop from the outputs to the inputs of the flip-flops. Therefore, input test vectors may not result in a unique set of output vectors. The four possible output sequences for the logic shown in Figure 5.4 are shown in the table below the diagram.

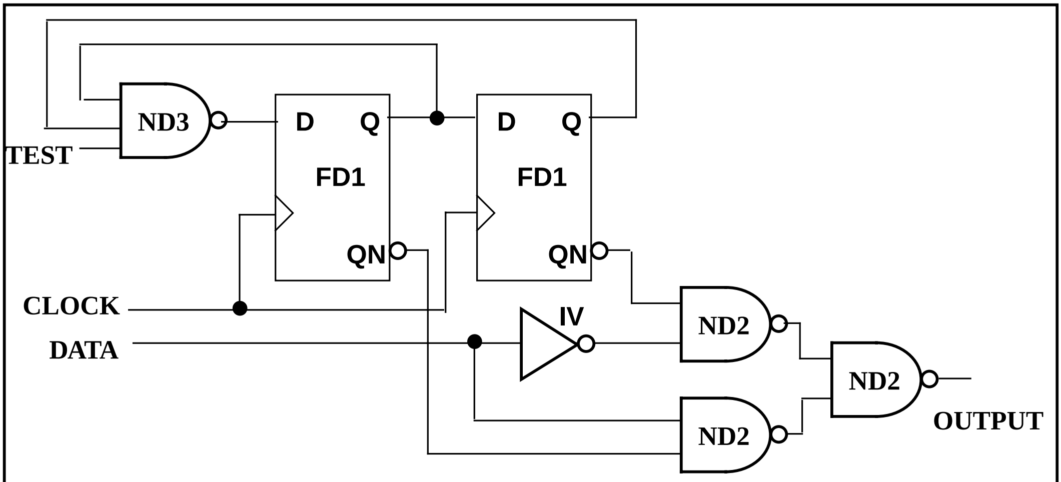


DATA	0	0	1	1	1	1	1	1	1	1	1	1	1	cont. 1’s
OUTPUT when initial A, B=00	1	1	0	1	0	0	1	0	0	1	0	0	1	repeat 001
OUTPUT when initial A, B=01	0	1	0	1	0	0	1	0	0	1	0	0	1	repeat 001
OUTPUT when initial A, B=10	1	0	1	0	0	1	0	0	1	0	0	1	0	repeat 010
OUTPUT when initial A, B=11	0	0	0	0	1	0	0	1	0	0	1	0	0	repeat 100

Figure 5.4

Untestable Sequential Logic

Figure 5.5 shows how adding a test line makes the same circuit testable. The test line logically breaks the feedback loop. Holding the test line low can produce a known output after a maximum of two clock cycles. Then the test line must be brought high. An input test vector then produces a unique output vector. The test line need not be a separate external pin on the package; you can derive it from an unused state of some other combinational logic on the chip. (If this approach is used, however, take care that your test signal is stable before you begin your test.) Note that if, in using this approach, your logic should ever pass through the unused state, the chip may be put into the test state. Design your chip in such a way that it cannot be put into test state by accident.



DATA	0	0	1	1	1	1	1	1	1	1	1	1	1	continue 1’s
TEST	0	0	1	1	1	1	1	1	1	1	1	1	1	continue 1’s
OUTPUT	x	x	0	1	0	0	1	0	0	1	0	0	1	repeat 001

Figure 5.5
Testability Added by “Test” Input (Breaking the Loop)

CHAPTER 5 - SECTION 4
Use Reset Signals to Initialize Storage Devices

Adding reset signals is one of the simplest and most effective ways to improve the testability of circuits that contain storage elements. The penalty is the additional wiring the buffering needed to drive the fanout. Storage elements require a testing strategy that puts them into known states. You can initialize storage elements by clocking in known input data, but by adding a reset signal you can reduce the number of initialization test vectors.

Since many subsystems (like the one shown in Figure 5.6) use reset signals in their normal operation, a reset signal usually involves little or no additional logic. The reset signal can be used to reset all elements whose state cannot be readily initialized by other means. For example, the divide-by-three circuit from Figure 5.4 (clock/3) at the bottom of Figure 5.6 uses the reset signal from the shift register above it.

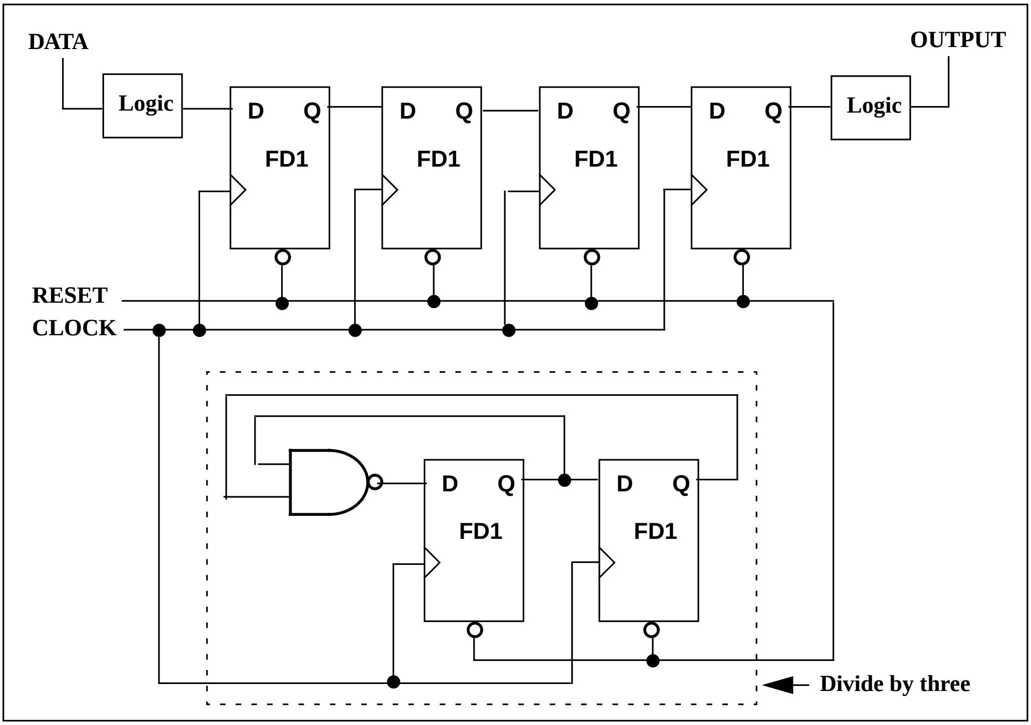


Figure 5.6
Use Reset Lines to Initialize Storage Elements

A master reset is not always required. The technique used in Figure 5.5 illustrates how the use of a 3-input NAND gate enables you to add a test line. This test line provides an alternative method of initializing the flip-flops, so that a master reset signal need not be used.

CHAPTER 5 - SECTION 5
Make Logic Observable by Multiplexing Output Pins

In Figure 5.7a, the state of the three FD1's can be observed by placing a 3-to-1 multiplexer in front of the output buffer, as shown in Figure 5.7b.

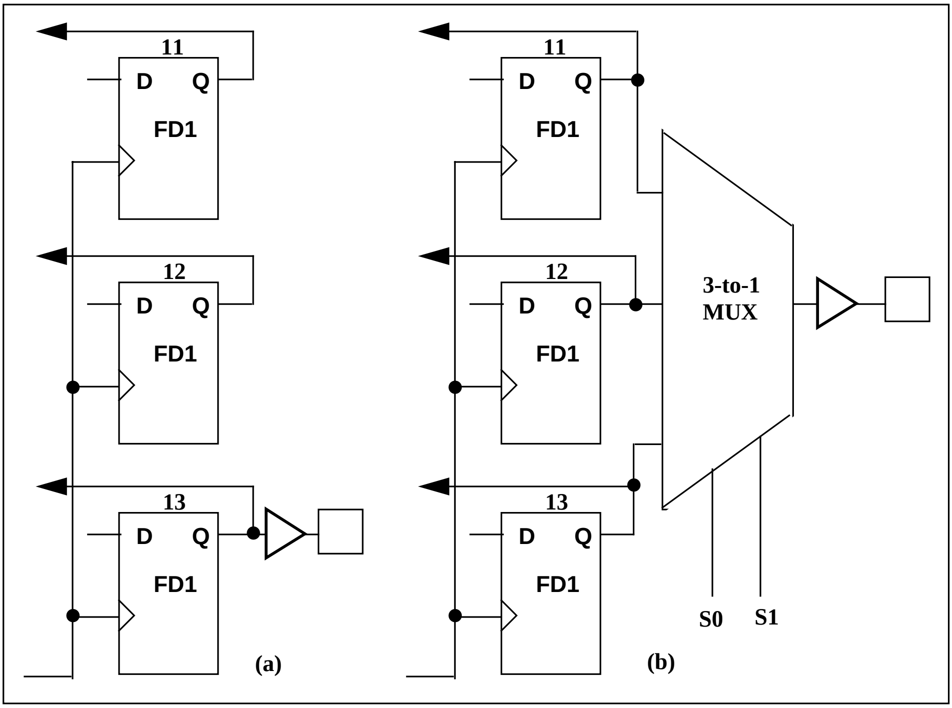


Figure 5.7
Multiplex Output Pins

CHAPTER 5 - SECTION 6
Decode Input Pins

You can make deeply buried states (e.g., internal registers) controllable for testing by decoding input pins, as illustrated in Figure 5.8. You can monitor internal registers by connecting them to package-pin test points. To increase the number of effective package pins for the purpose of monitoring internal registers (improving observability), make unidirectional pins bidirectional. If pins cannot be made available in this way, enable on-chip test logic with a combination of inputs that cannot occur in actual system operations. Never, however, put a clock signal on a bidirectional pin.

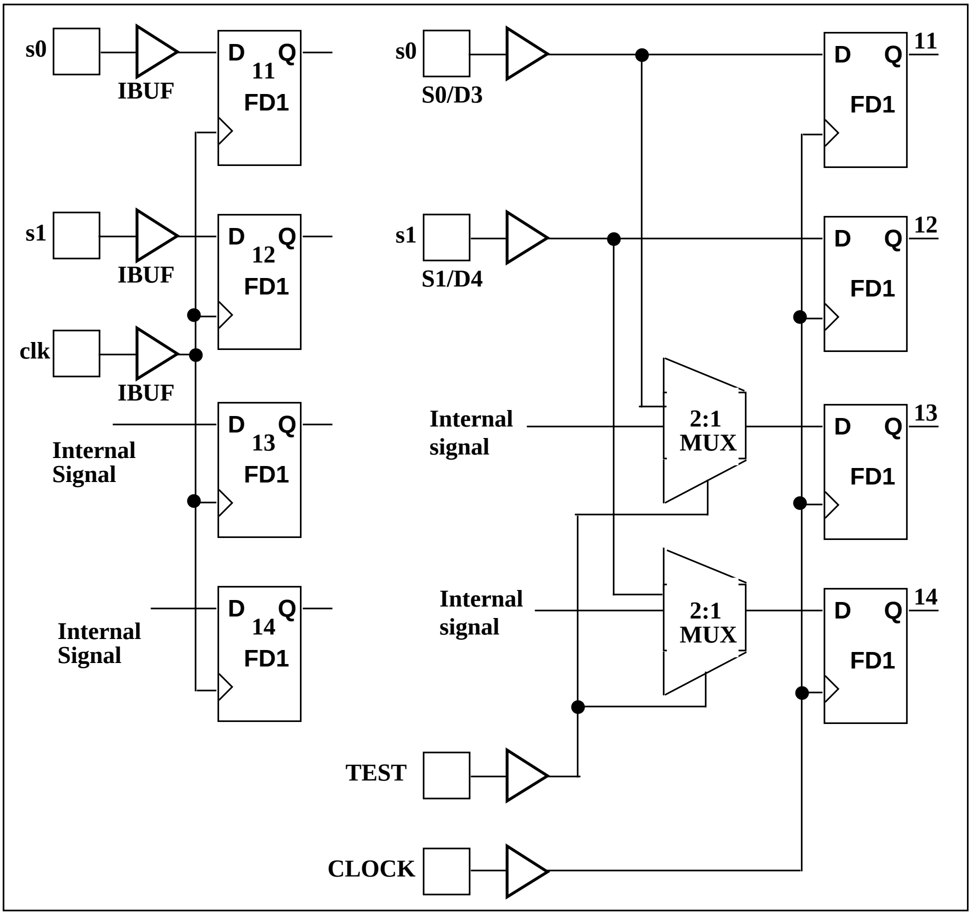


Figure 5.8
Decode Input Pins

CHAPTER 5 - SECTION 7
Break Up Long Sequential Chains

Long counters, although they may operate satisfactorily in circuit systems, can add significantly to the number of test vectors needed for proper testing. For example, the 16-state counter shown in Figure 5.9 would require 2^{16} power (65,536) test vectors merely to apply a single clock to the control circuitry. Assuming the control circuitry has 2^{12} possible states for thorough testing, about 2.7 x 10^8 test vectors would be needed to test the circuit completely.

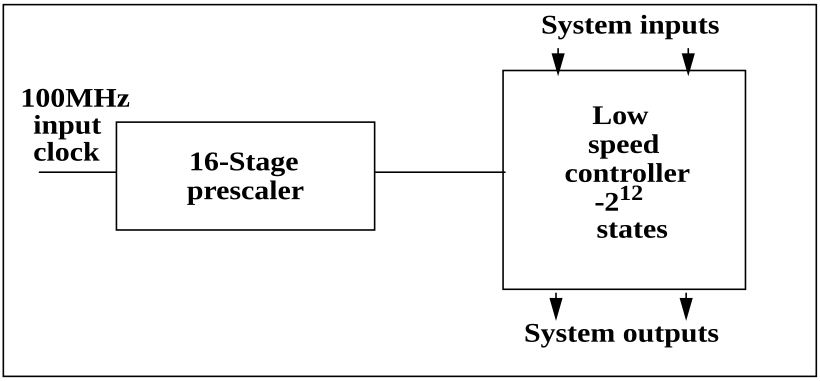


Figure 5.9

Long Counter is Difficult to Test

A practical solution is to break the 16-stage counter into two 8-stage counters, as shown in Figure 5.10. You may then test two portions of the counter and the control circuitry in parallel; 4096 test patterns are sufficient for this task.

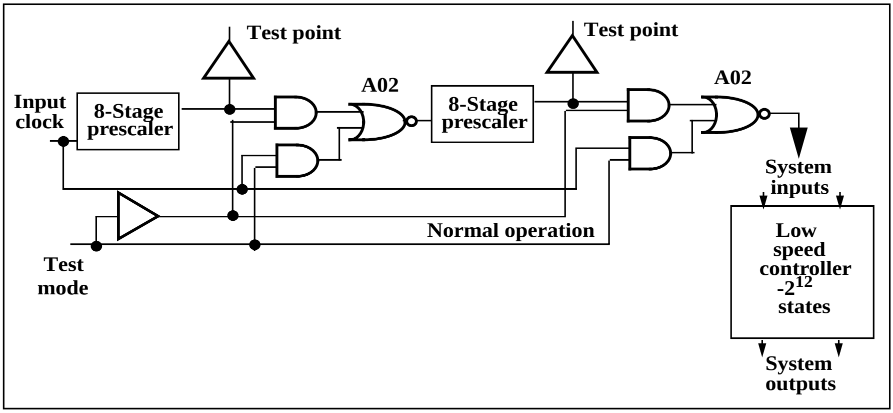


Figure 5.10

Two Short Counters are Easier to Test

CHAPTER 5 - SECTION 8

Use Level-Sensitive Design

In large designs, unavoidable skew along a clock signal may make edge-triggered flip-flops unreliable due to hold-time violations. One way to avoid this problem is to use a level-sensitive design technique. In Figure 5.11, ENA and ENB are non-overlapping, active-high enable signals. Signals are transferred into one level of the design using the ENA signal and into alternative levels using the ENB signal. In this way, the flow of data through the design can be staged and very well controlled.

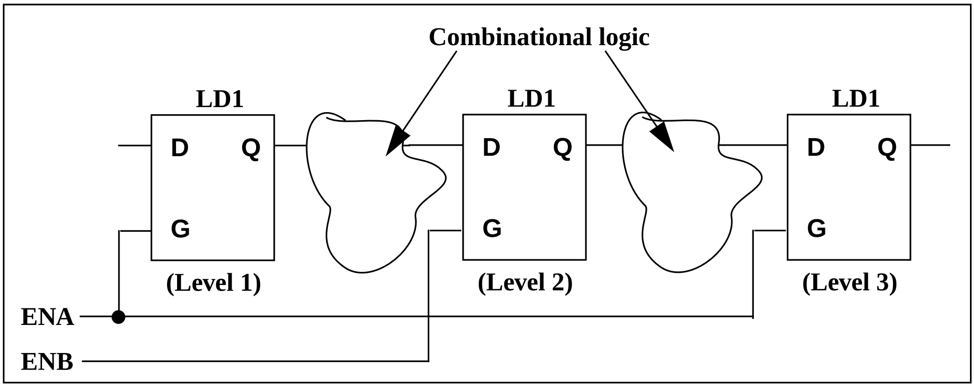


Figure 5.11
Level Sensitive Design

It is even possible to gate enable lines with complete safety (although with some loss of speed) as long as all gating signals originate with latches using the opposite enable phase (see Figure 5.12).

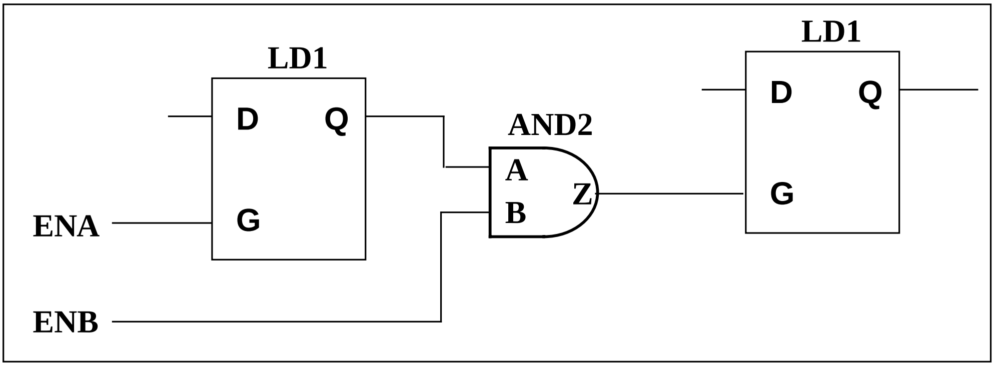


Figure 5.12
Safely Gated Enable Line

CHAPTER 5 - SECTION 9
Partition Logic into Blocks

A large design may contain deeply buried functional blocks which are both difficult to control and to observe. In such a case, it may be desirable to divide the design into easily testable blocks. These blocks may be made functionally transparent while the design is in normal operational mode. With only a slight increase in delays between blocks. Figure 5.13 and 5.14 show how this can be done.

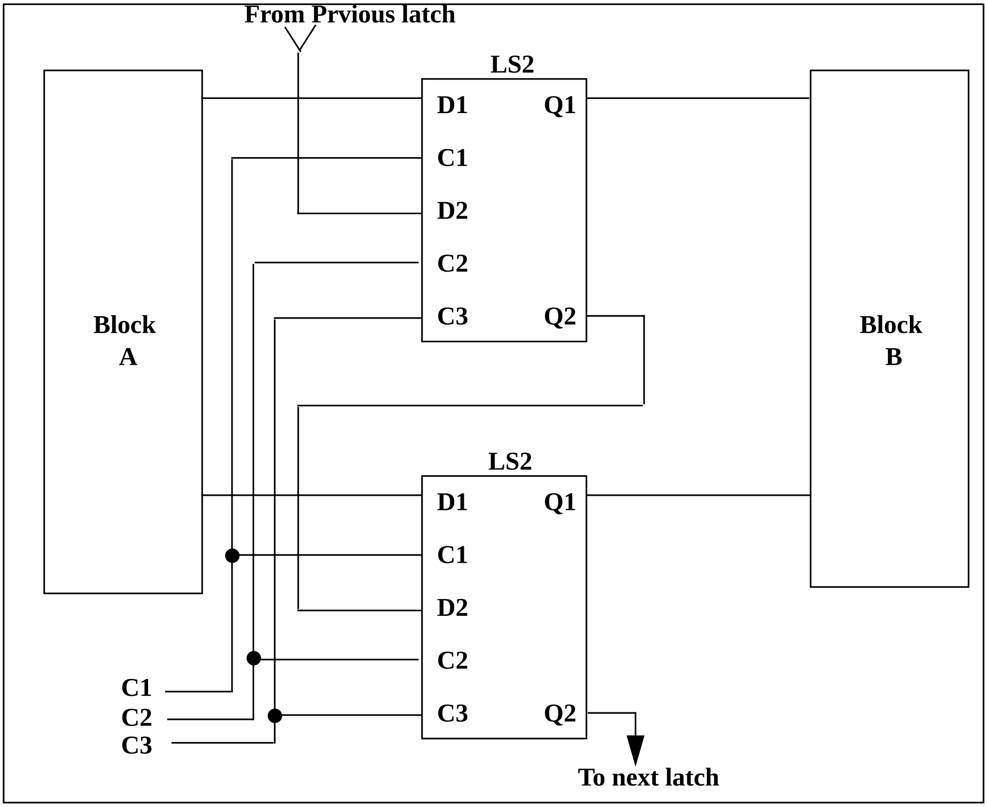


Figure 5.13
Block Level Boundary Scan

During normal operation, C1 is always high, and C2 is always low. Thus the latches are merely transparent elements. In test mode, if Block A is to be observed, C1 is held until the proper time and then brought low. Then the data is scanned out using C2 and C3 as non-overlapping enable lines. To control Block B, C1 is left low and data is scanned in as before.

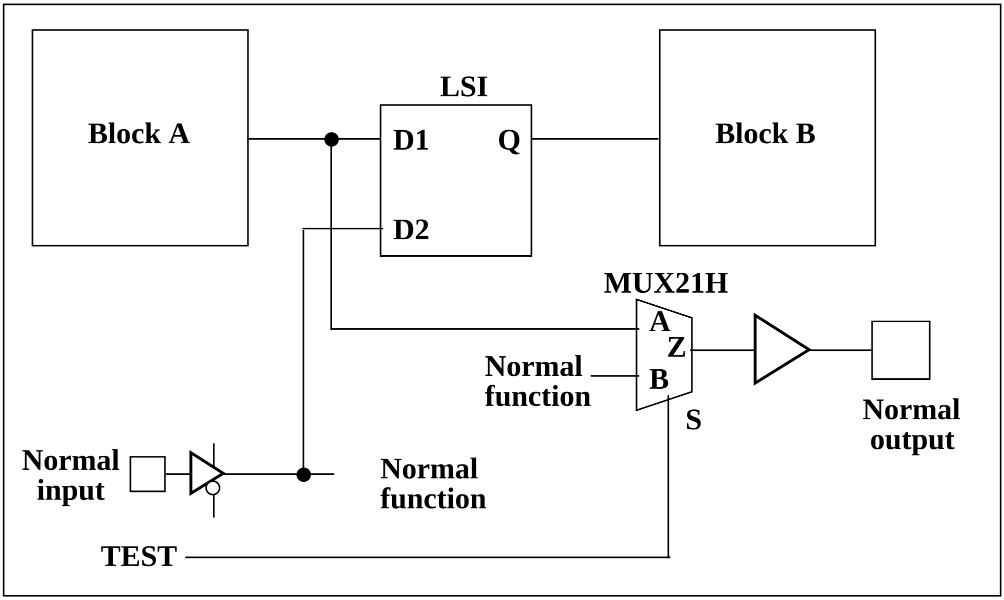


Figure 5.14
Block Partitioning Using Muxes

CHAPTER 5 - SECTION 10
Do Not Design Your Own Storage Devices

Timing Considerations

In contrast to LSI Logic defined storage devices, customer-designed storage devices do not have known layouts. Timing analysis of customer-designed storage elements is therefore unreliable. If you need a storage device not available in an LSI Logic library, ask that it be coded by LSI Logic as a hard-coded macrocell. The layouts of hard-coded macrocells are known and can therefore be accurately modelled and simulated by LSI Logic simulation software.

Figure 5.14 depicts a customer-designed storage device that has been flattened for analysis. LSI Logic layout software treats this combination of gates as a random collection of gates - i.e., not as a functional block (in this case, a D flip-flop). In other words, our software does not identify (or recognize) the gates comprising a user-defined storage device as: data inputs, clock inputs, reset, etc. It does not, therefore, analyze signal-timing, relationships among the signals in a user-defined storage device. The concept of set-up time, for example, which is essential to the analysis of a storage device, is irrelevant to gates not identified as those of a storage device. (Note that Figure 5.15 is the logic schematic of a D flip-flop.) See if you can identify the pins. The answers are at the end of this chapter.

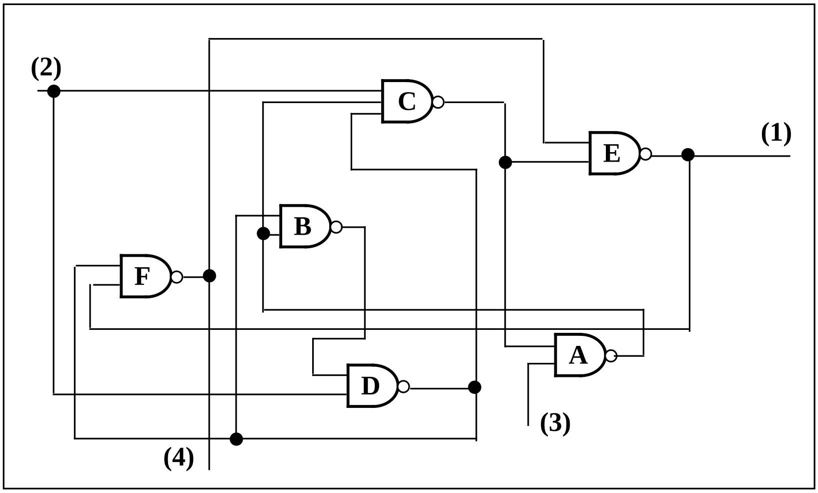


Figure 5.15
Customer-Designed Storage Device

Logic Considerations

Very often the inputs to a transmission-gate multiplexer are briefly shorted to one another. As a result, a logical 0 on one input could change the state of latched data going into another input. LSI Logic solves this problem by buffering the outputs of its hard-coded storage devices in order to make feedback paths inaccessible at the output pins of the storage element. Figure 5.16 depicts a potentially dangerous use of a transmission-gate multiplexer, on of whose inputs is an unbuffered storage device. Note that the signal on B can follow the dotted line path into the storage device, toggling the state of the latch.

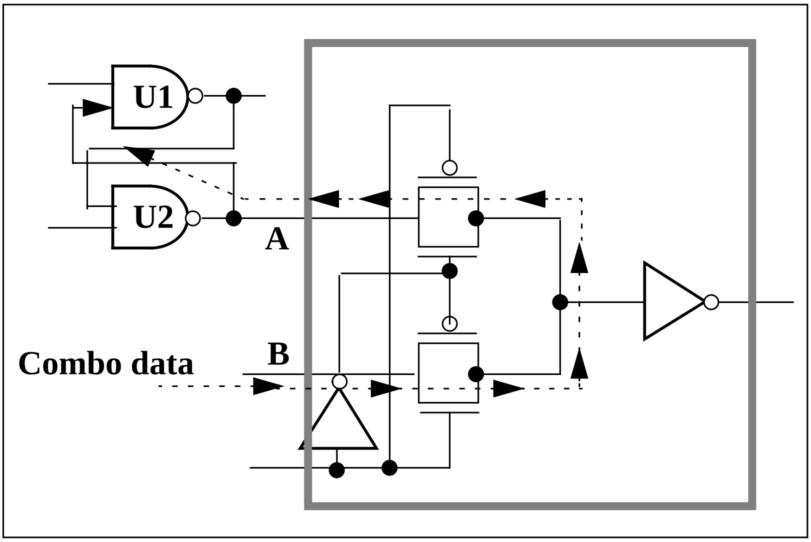


Figure 5.16
Dangerous Use of a Transmission Gate Mux

There are other compelling reasons why designers should abstain from creating their own storage devices, such as non-identical layout of logically identical storage cells, differing AC performance of logically identical storage cells, and so on.

CHAPTER 5 - SECTION 11
Use a Serial Scan Approach

Standard scan design uses a one-phase clock. Place a 2-to-1 multiplexer in front of each register, as shown in Figure 5.17.

In this case, each flip-flop receives its data input either from its normal path or from the Q output of the previous flip-flop in the scan chain, depending on the state of the test pin. The top flip-flop is an exception; its input comes from a test input pin. The bottom flip-flop's Q output goes both to its normal destination and also to a test output pin.

The circuit is placed in test mode by setting TEST to HIGH. In this mode, all the flip-flops are connected or "chained" together as a single serial shift register. This allows a serial data stream to be preloaded into the storage elements. When TEST is set to LOW, the circuit reverts to its normal operating mode and is clocked for one or more cycles to exercise the combinational logic. As a final step, the circuit should be placed in test mode again to shift out the contents of the scan chain for examination. Because scan chains can be very long for large circuits, LSI Logic uses a simulation technique that allows scan chains to be parallel loaded and unloaded during simulation. LSI Logic's lsitest and lsimts programs later convert these parallel patterns into serial patterns for actual use on the testers.

Entire PC boards may be put into test mode if properly designed. All of the individual circuits can then be read and analyzed. Serial scan design enables you to divide a large circuit into two or more independent sub-circuits for testing. You can shift out and examine the state of every storage element in your design.

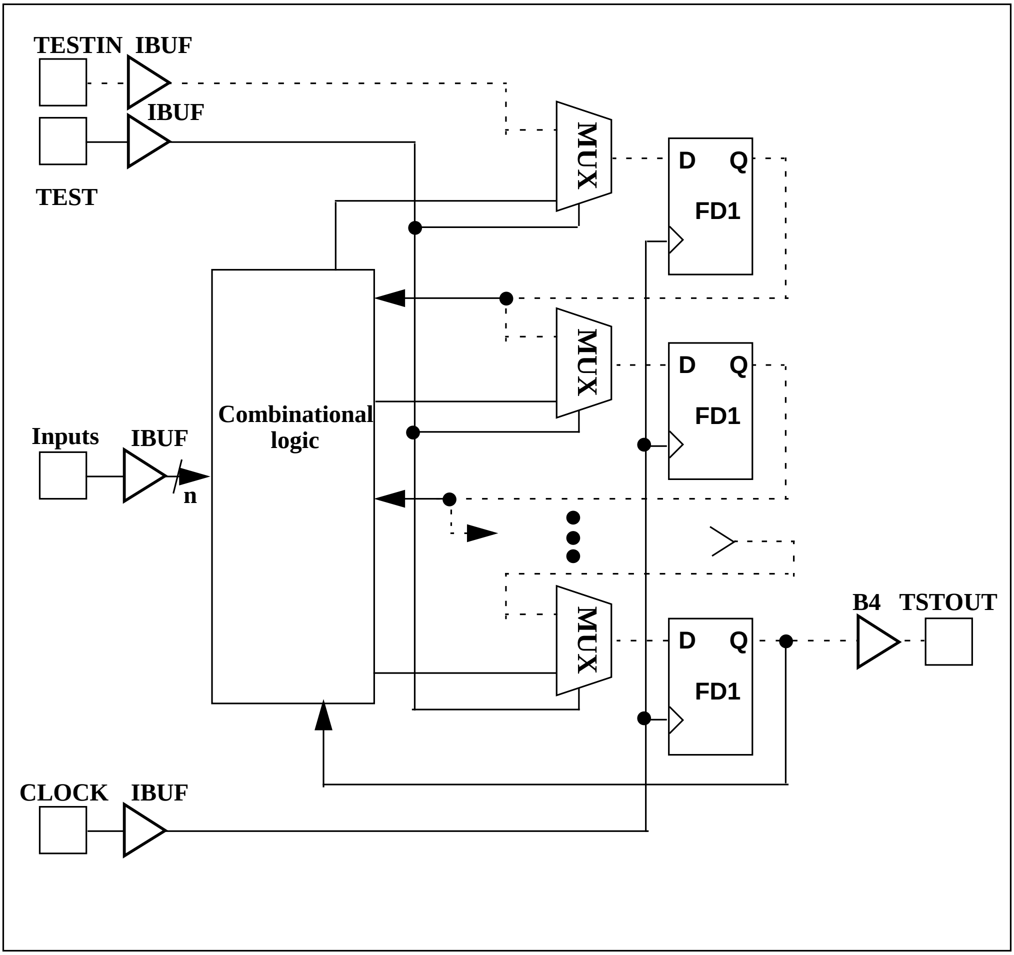


Figure 5.17
Scan Design

CHAPTER 5 - SECTION 12
Multiplex Output Pins to Make Scan Chain Observable

Pin requirements need not be increased for circuits using scan design since scan outputs can be multiplexed with regular output signals, as in Figure 5.18. Do not, however, multiplex critical path outputs.

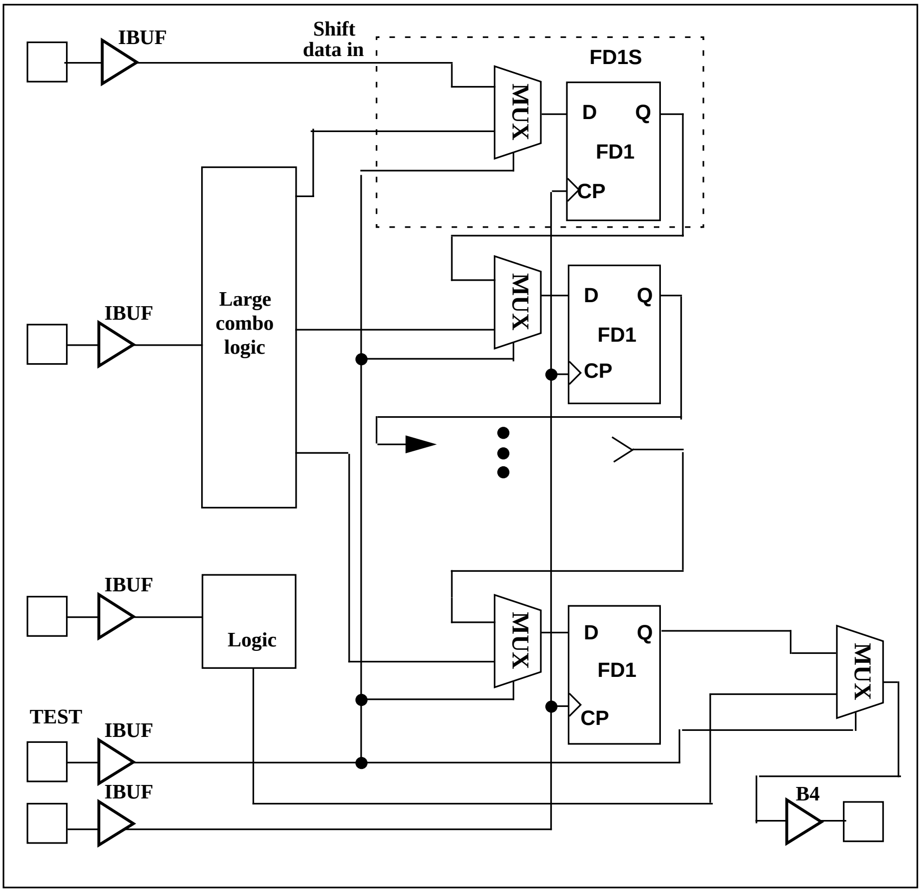


Figure 5.18
Sharing the Scan Output

Note that several scan flip-flops are available in LSI Logic libraries, (The circuitry enclosed within a dotted line in Figure 5.18 and identified as an FD1S is such a flip-flop.) Each scan flip-flop has a test input (TI) and test enable (TE) input in addition to its normal inputs. Scan test flip-flops have an "S" suffix on their macrocell names.

CHAPTER 5 - SECTION 13
Implementing the Vil/Vih (NAND) Tree

What is the NAND Tree?

The LSI Logic NAND tree is a dedicated circuit for testing each input buffer's Vil/Vih. It is formed from a special NAND gate embedded within each input or bidirect buffer. One input of the NAND gate is connected to the output of the buffer, and the other is given the port name PI. The output of the NAND gate is referred to as PO. This means that, for a non- inverting input buffer, the PI pin is active when the input signal is at 1 and inactive when the input signal is at 0. One unidirectional output buffer is chosen as the NAND tree output. This buffer is driven, through a MUX if desired soas to avoid having a dedicated NAND tree output pin, by the nearest PO pin. The corresponding PI pin is then connected another PO pin, and the process is repeated until all PO pins are connected. This leaves one unconnected PI pin, which is tied to logic 1. The result is a daisy-chain of NAND gates, all leading to a single output buffer. There is now a purely combinational path from each input buffer to a single output buffer.

Why is it important to have a combinational path?

The Vil/Vih test is performed with each input at its own Vil-max or Vih-min. This leaves no noise margin at all. Any storage element within the circuit would be susceptible to the first noise spike that came along.

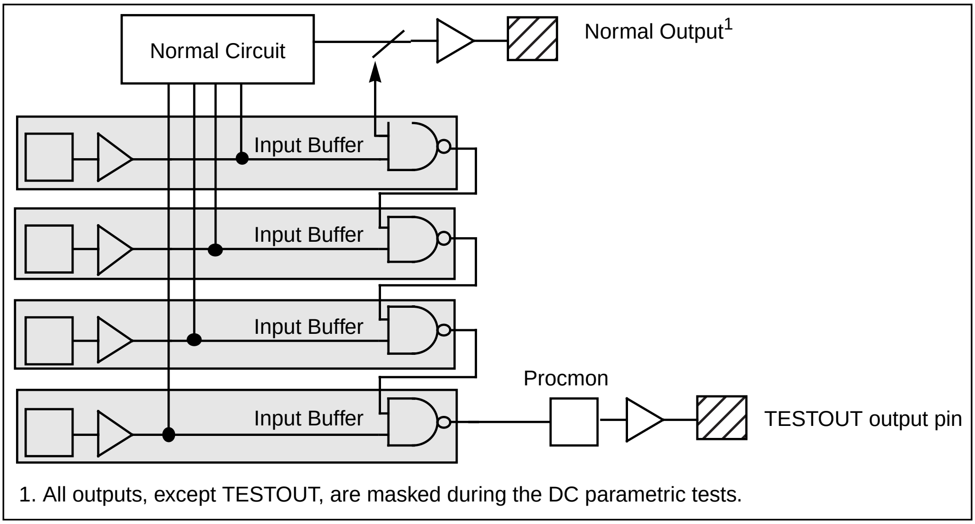


Figure 5.19
NAND Tree Structure

How is the NAND tree sequenced?

To save routing resources, it makes sense to connect the NAND tree in a sequence dictated by the bonding diagram, so that the wires between NAND gates will be as short as possible and will be confined to the edges of the die.

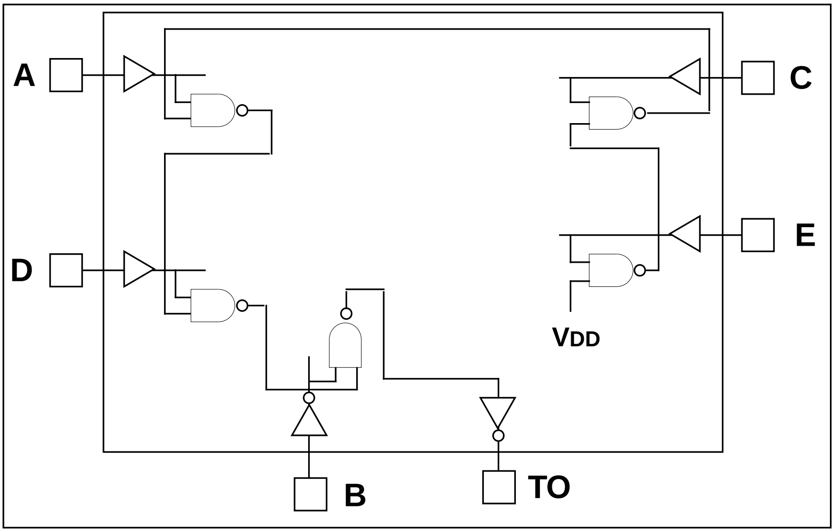


Figure 5.20
NAND Tree Sequence

How is the NAND tree tested?

In its simplest form, the Vil/Vih test is performed by setting all inputs to logic 1, then setting each to 0 in order of connectivity, starting with the buffer furthest from the output (that is, the one with its PI pin tied to 1). If there is an even number of buffers, the output is now at 1; if there is an odd number, the output is at 0. As each input goes from one to zero, the output toggles. The overall result is a function of each input properly responding to both its own Vil-max and Vih-min. Notice also that the Vil/Vih test is a static test. It is only the voltage levels that are important, not the transitions themselves, and since the circuit itself is combinational, the particular sequence of the patterns is also irrelevant. We could re- arrange the sequence so that all the patterns that cause 1 on the output came first, and all those that cause a 0 came second, and we would still have a valid Vil/Vih test.

The test must also take into account special cases such as Schmitt triggers (where direction and sequence *are* important; they have to be toggled both ways), and other special buffer types such as PCI.

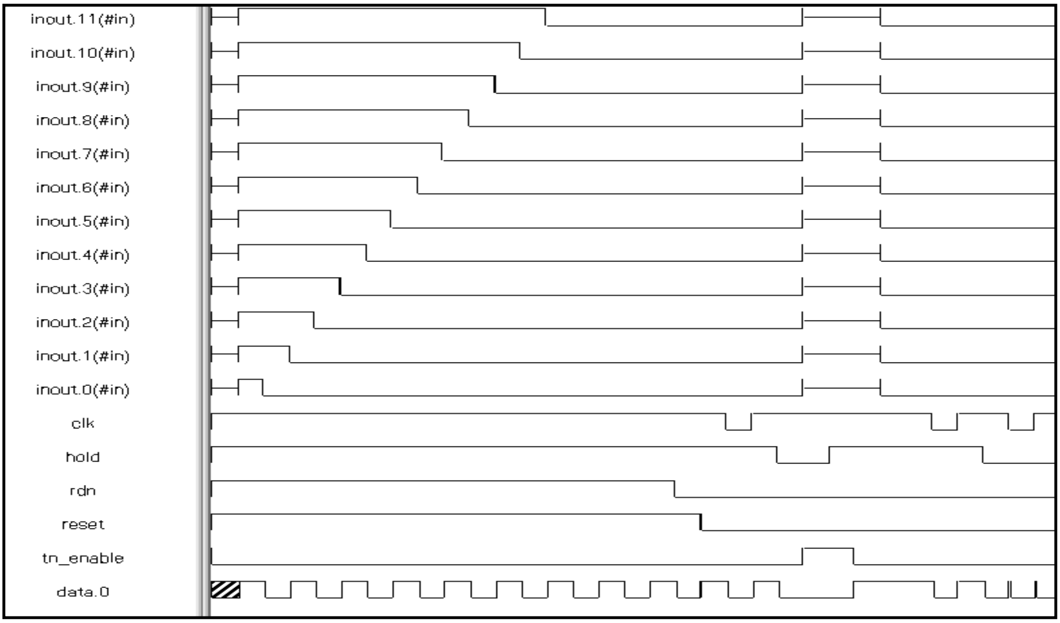


Figure 5.21
NAND Tree Test

Subtleties

The NAND tree has some surprising subtleties for something that seems at first glance to be so simple. These subtleties arise not from the NAND tree itself, but from the input buffers which drive it. The two most obvious problems are the bidirects and the MUX. How can we have a purely combinational circuit whose output is a function of all the inputs, and still guarantee that the bidirects will not suddenly go from input mode to output mode? After all, if we're testing the voltage response of the input half of the bidirect, we can't very well have it slip into output mode. But one of the other inputs that we need to toggle would have to be holding the bidirects in input mode, and if we toggle that input to test it, we risk switching the bidirects to output mode. What must happen is that the global three-state enable pin must appear in the NAND tree closer to the output than any bidirect buffer, and it's sense must be such that the bidirects are forced to input mode when the PI pin is active.

The other obvious problem is the MUX. What determines whether we're in test mode or normal mode, and how do we test that signal itself? Since most designs have an asynchronous reset pin (which is, by most conventions, active low), and most designers really don't care what the chip does while it is in the reset state, the reset state actually makes a very good Vil/Vih test mode. Just tie the reset signal to the select pin of the MUX. Ah, but wait! One of the LSI Logic design rules states that no PO pin can be unconnected. The simple solution is to drive the select of the MUX not with the Z output of the reset buffer, but with the PO output of the reset buffer. The PI pin is tied high. Testing the reset buffer for Vil/Vih is not a problem: simply toggle the reset signal when the two inputs to the MUX are in opposite states. One input of the mux (from the PROCMON “A” pin) comes from the NAND tree, the other (“N”) from the “normal” (non-test mode) circuitry. If the normal circuitry is not initialized, you'll be switching between the NAND tree and a X (unknown) state, which is not a valid test. This is yet another reason to use reset as the Vil/Vih test mode; presumably, reset will initialize the N pin.

The PROCess MONitor

OK, so now we have a daisy-chain of NAND gates driven by each input and bidirect, with a MUX at the output and all the bidirects close to the far end. What else? It turns out that, since a string of NAND gates this long has so much propagation delay, it actually doesn't hurt anything to make it even longer, which is why another test structure, the PROCess MONitor or PROCMON, is added to the end of the NAND tree. The PROCMON is a small hard-coded macro with four inputs (A, E, S, and N) and one output (Z). It consists of a special asymmetric delay chain, an exclusive NOR gate to configure the delay chain as a pulse generator, and some select circuitry to enable the E pin to select between pulse mode and inverted pass-thru mode. It even has the 2:1 MUX as its output stage (where the PROCMON's S and N pins are the S and A pins of the MUX, respectively). By using the global three-state enable signal as the mode (or “E”) pin of the PROCMON, the bidirect enable problem is handled. During the static Vil/Vih test, pulse mode wouldn't do us much good, so we must be in inverted pass-thru mode. The preferred global three-state input buffer is the ICPTNU cell. The polarity of the signals on the PROCMON is deliberately designed so that connecting the PO pin of the ICPTNU to the E pin of the PROCMON will give valid results.

There is one other important restriction in connecting the NAND tree. If proper connection of the PROCMON and the ICPTNU buffer takes care of bidirect conflicts, do we still need to worry about where the bidirects fall within the NAND tree? Well, actually, yes. Testing the PROCMON involves toggling both the ICPTNU buffer and the input buffer closest to the PROCMON. If either of the two input buffers closest to the PROCMON are bidirects, an unknown logic value may propagate to the output when the ICPTNU buffer allows the bidirects to become outputs. Therefore, the two buffers closest to the PROCMON must be unidirectional input buffers.

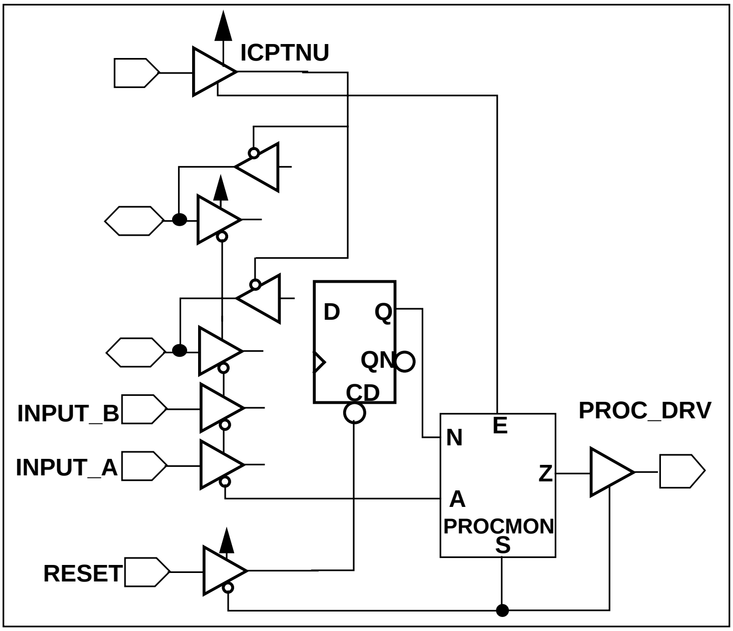


Figure 5.22
A Complete NAND Tree Circuit

The PROC_DRV

A brief soon-to-be historical note here: In the G10 and G11 technologies, there is a special PROCMON output buffer called a PROC_DRV. It has an EN pin which changes its drive strength from 4mA to 2mA. This EN pin must be (manually) connected to the same source as the PROCMON S pin so that during the Vil/Vih test, the 2mA mode is active and the noise due to the switching of the output buffer is reduced. This was important at one time because of TTL compatible input buffers. TTL buffers had a very low Vih-min, and were very susceptible to noise during the Vil/Vih test. Now, remember that the NAND tree is connected in sequence around the die. Realize also that this may mean that the input buffer at the far end of the NAND tree may be very close physically to the NAND tree output. If this input or bidirect were TTL compatible, an unfortunate thing would happen. During the first pattern of the Vil/ Vih test, the TTL compatible input buffer at the far end of the NAND tree and physically close to the output buffer goes from 1 to 0. This would send a transition all the way around the periphery of the die and cause the output to switch, which would cause a noise spike on the ground bus right next to the noise-sensitive TTL input buffer now sitting at logic 0. The noise spike causes the TTL input buffer to glitch, sending a transition all the way around the periphery of the die.... You get the idea! Since the voltage level in the G12 technology is low enough so that TTL compatibility is no longer possible, the rationale for the PROC_DRV is gone as well.

The IIDDTN

Today's technologies also have another dedicated test input buffer, called the IIDDTN. The purpose of this buffer is to turn off all of the pullup and pulldown "resistors" (actually transistors) on input buffers, and power down any other applicable structures in the I/O region. It also has another (actually identical) output which the user can connect to any other circuit elements (e.g. high density RAMs, PLLs, cores) which need to be powered down for the Idd and Iddq tests. What does this have to do with the Vil/Vih test? Well, it turns out that some input buffers have their input thresholds altered when the IIDDTN buffer is asserted. This has unfortunate repercussions if it happens in the middle of the Vil/Vih test, so if you allow the IIDDTN pin to be included at some arbitrary point in the NAND tree, and if something "downstream" has its threshold altered, the Vil/Vih test becomes much more complex, to say the least. Add to this the fact that this input buffer is purely for test and nobody really cares whether its input thresholds are a little off, and it becomes apparent that maybe the IIDDTN shouldn't even have PI and PO pins, and a declaration has been made that it is methodologically permissible to leave it completely out of the NAND tree. Unfortunately, doing so will cause errors in several FlexStream Programs. You can either leave the IIDDTN buffer out of the NAND tree and ignore these errors, make sure (in the absence of any clear documentation) that none of your input buffers are affected by the IIDDTN buffer, or carefully position the IIDDTN pin within the NAND tree such that it is “out of the way” so to speak. The best place to put it is as the second-closest buffer to the PROCMON. Then you need only ensure that the input buffer closest to the PROCMON is sufficiently simple (e.g. an IBUF) soas to be unaffected.

The PROCMON2A

Yet another wrinkle in the NAND tree saga is the fact that the G12 technology has two different kinds of transistors, High Performance (HP) and Low Leakage (LL). Thus the need for two different kinds of PROCMONs and, for designs that contain both kinds of transistors, a third, called PROCMON2A, that contains both of the other two, along with an extra input (SEL) to choose between them. In non-pulse mode, the path from the A pin through the low leakage transistors is non-inverting, to differentiate it from the high performance path and ensure that the input buffer driving the SEL pin can cause a transition at the output and thus be tested for Vil/ Vih. Since the state of the SEL pin is only important during test mode (presumably reset, as suggested earlier), the choice of which input to hook to the SEL pin is not important but, as before, it should be the PO pin from the input buffer, with the PI pin tied to logic one.

lsints

There is a FlexStream program which will automate almost all of the busy-work associated with hooking up the NAND tree and PROCMON. It is called lsints, for NAND Tree Synthesis. The only input data required for this program, other than a few rather obvious command line options, is the UCF (User Control File) file. (OK, it's a stupid set of initials, and it's kind of like saying “ATM machine” but what the heck, it's easy to remember.) The most important information in the UCF file are the connections to each pin of the PROCMON cell. These take the form of “procmon_*x*pin” where “*x*” can be *a, e, n, s, z,* or *sel*. In most cases, the program wants the name of the external signal, not the name of an internal net or cell. Further, the program assumes that specifying, for example, “procmon_spin reset” means that you want the PO pin of the “reset” input buffer to drive the “S” pin of the PROCMON. By default, the program creates a modified netlist file, Vil/Vih test pattern simulation files, and an interface file for the *lsitest* program. Future plans include the creation of a WGL file so that you won't have to bother running the simulation.

Since there are always people who want to make extra work for themselves, not to mention the occasional redesign, the program has a “pattern_only” mode which will create test patterns but not a netlist, which of course assumes that the NAND tree already exists in the circuit. There are some discrepancies between the UCF file for “full” mode and the UCF file for “pattern_only” mode. In “full” mode, the arguments for procmon_npin and procmon_zpin are identical. In “pattern_only” mode, the argument for procmon_npin must be an internal net name. Also, in “full” mode the “procmon_apin” command is ignored, but in “pattern_only” mode it is required.

Answers to Figure 5.15:

(2) = CLK

(1) = $\overline{Q}$

(3) = DATA

(4) = Q

6

Automatic Tester Requirements

CHAPTER 6 - SECTION 1
Overview

Your device will be tested at two different stages during the manufacturing process: Once at wafer probe, and once at final test after the device has been packaged. During both testing sequences, the automatic tester uses the test patterns which you created during tester functional and parametric simulations. The tester verifies the correct operation of each circuit by clocking in test patterns, and then checks to see whether the output patterns are identical to the ones predicted by simulation. The tester also verifies static and switching parametric values on the pins of the device.

Information about how the testers used at LSI Logic operate can help you to decide the best way to create test patterns for your circuit.

CHAPTER 6 - SECTION 2

Test Flow

The types of tests performed by the tester and the sequence in which they occur is shown in Figure 6.1. When LSI Logic tests a device, the tests are executed in the order shown in Figure 6.1. This method reduces test time by identifying the most common failure modes early in the sequence. Most manufacturing defects will be detected in the first three tests.

	LSI Tester Setup	Customer Functional Patterns	Customer VIL / VIH Patterns	Customer 3-State Patterns	Customer IDD Patterns
Open / Short	X				
Gross IDD					X
Gross Functional		X			
Three-State				X	
VIL/VIH			X		
PROCMON			X		
Functional		X			
Static IDD					X
VOH		X			
VOL		X			
IIL/IIH	X				
IOZ				X	

Figure 6.1
Test Flow

CHAPTER 6 - SECTION 3
Automatic Tester Configuration

LSI Logic currently uses a number of different automatic testers: These Production Testers can handle pin counts up to 784 pins, can run at speeds up to 100MHz, and can handle up to 8 scan chains. Some of the testers are also setup to provide mixed mode capability. These testers physically apply voltage and/or current to the device under test, measure the voltage and current levels on the inputs and outputs, and measure propagation delays through representative paths.

Described most simply, a tester operates in a periodic fashion. Input signals change states at the beginning of the test period. Output signals are strobed at the end of the period to determine whether the measured values match the simulated values, as shown in Figure 6.2.

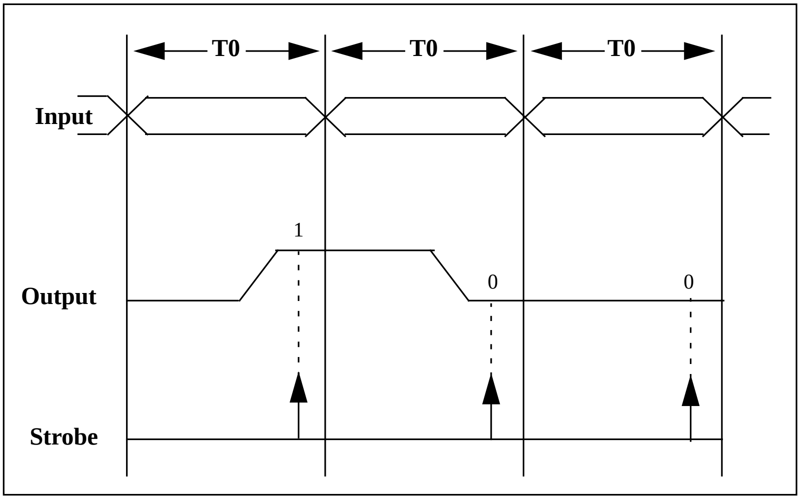


Figure 6.2
Fundamentals of Tester Operation

Figure 6.3 depicts how the automatic tester applies input patterns and measures outputs of the device under test. The shaded box at the left labelled A represents the local memory in the tester as well as the test patterns contained in the local memory. The patterns from the input columns are taken out of the local memory and applied to the device under test one pattern at a time. At the same time, the patterns from the output columns are compared with the output pins of the device under test.

A single binary bit in the local memory does not contain sufficient information for the tester to apply the correct waveform to the device under test. Therefore, a waveform generator is needed to generate the proper waveform characteristics. All LSI testers have at least six such generators, referred to as input timesets.

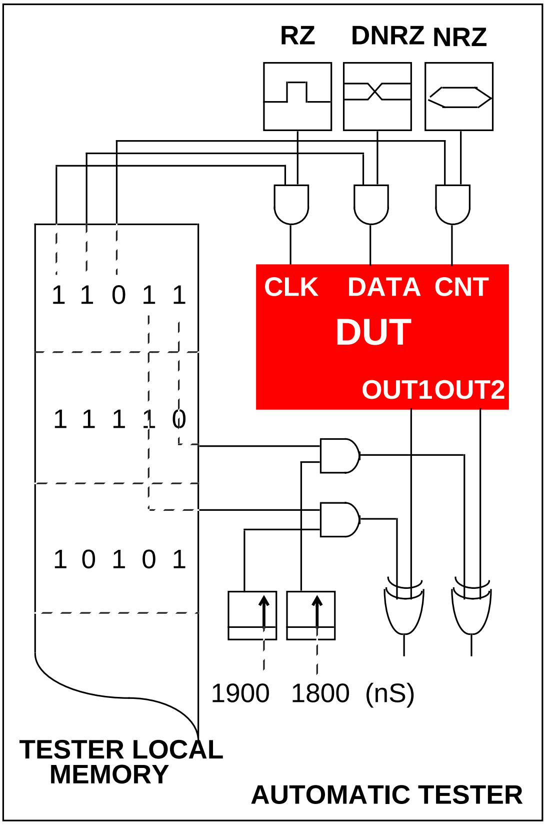


Figure 6.3

Automatic Tester Equivalent Diagram

A timeset is like the waveform generator that you'd normally use on the lab bench. Like a waveform generator; each timeset can be programmed to have a particular frequency, delay, pulse width had duty cycle. However, unlike a waveform generator, once a timeset is programmed to a particular set of timing characteristics such as frequency, you cannot change these characteristics for the duration of a given test block.

By using the combination of the binary patterns in the tester's local memory and the timing information in the input timeset, you can specify the precise input waveform to be used as input stimuli to the device under test. The six input timesets are used to replicate simulation input. Two output timesets are used to measure functionality and propagation delays.

Figure 6.3 shows the equivalent automatic tester configuration.

Up to six timesets may be used. The number of timesets used is governed by the number of groups of input pins that have identical timing characteristics. Suppose, for example, that a device has sixteen data pins, two clock pins, and three control pins. Also suppose that this device has a timing diagram like the one in Figure 6.4. If you were to apply signals to all the data pins simultaneously, then you could group all sixteen pins into a single timeset. Since the two clock pins have different delays and duty cycles, they must each use a separate timeset. Two of the control pins, CNTRL1 and CNTL2, have identical delays and therefore are grouped into a single timeset. CNTL3 has a different delay, and so it must use its own timeset.

As a result, five timesets are required for this device as shown in Figure 6.4.

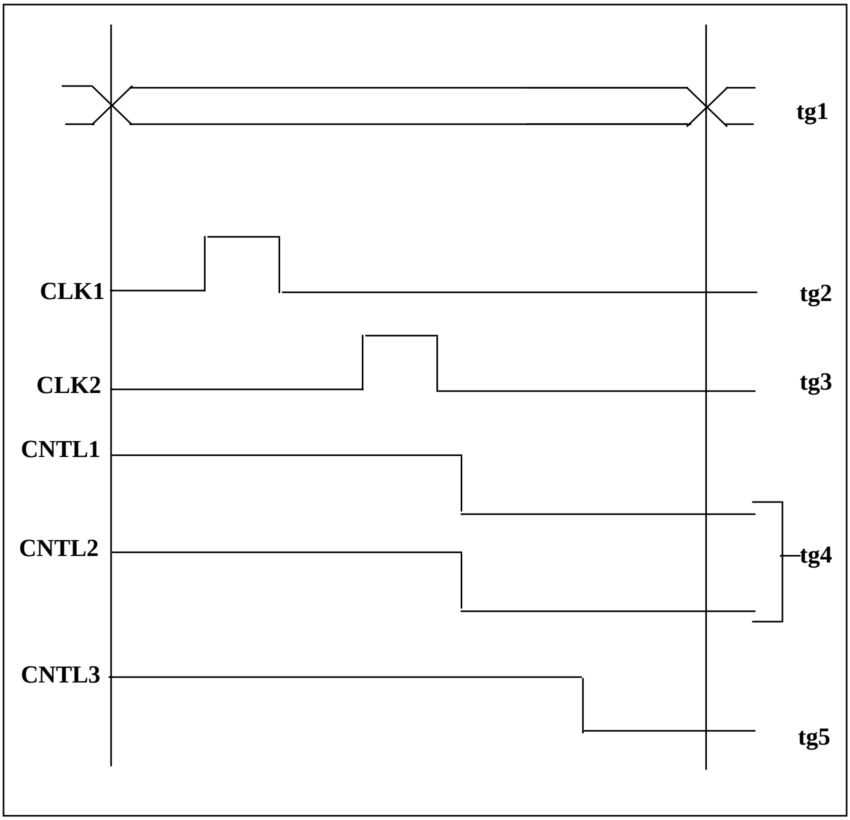


Figure 6.4

Example Showing Five Timesets

CHAPTER 6 - SECTION 4
Test Period Length

The minimum test period size must be greater than or equal to two parameters:

- 10 ns minimum (the timing constraint on the 100 MHz tester.
- the maximum propagation delay in the circuit

plus tester overhead, which will be explained later

The test period must be greater than the maximum propagation delay in the circuit because at the end of each test period, only the steady-state values of the output pins are measured. The test period must therefore be long enough so that an input change causes outputs to change state within the same test period, and not in a subsequent test period.

The maximum propagation delay is defined as the longest delay of the circuit. This is typically a path from an input pin to an output pin, as shown in Figure 6.5.

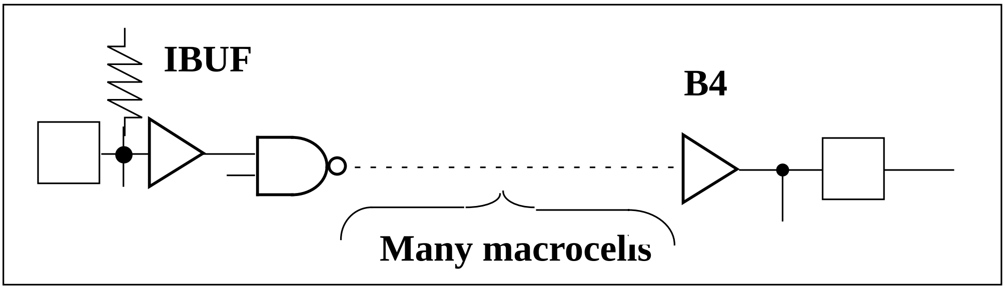


Figure 6.5
Maximum Propagation Delay Path

If the test period is not long enough to accommodate the maximum propagation delay, the circuit may be improperly tested. For example, in Figure 6.6, we purposely shortened the test period to one-half the length of that in Figure 6.2, creating twice the number of test periods and output strobes. As you can see in Figure 6.6, the state of the output will be different at the third strobe point using the new cycle time than it was at the second strobe point using the original cycle time. The circuit still operates correctly, but the choice of too small a cycle time in Figure 6.6 make it appear as if the outputs are in the wrong state. Differences between worst-case and best-case simulations could cause discrepancies like these if the cycle time was not set long enough to cover expected delays.

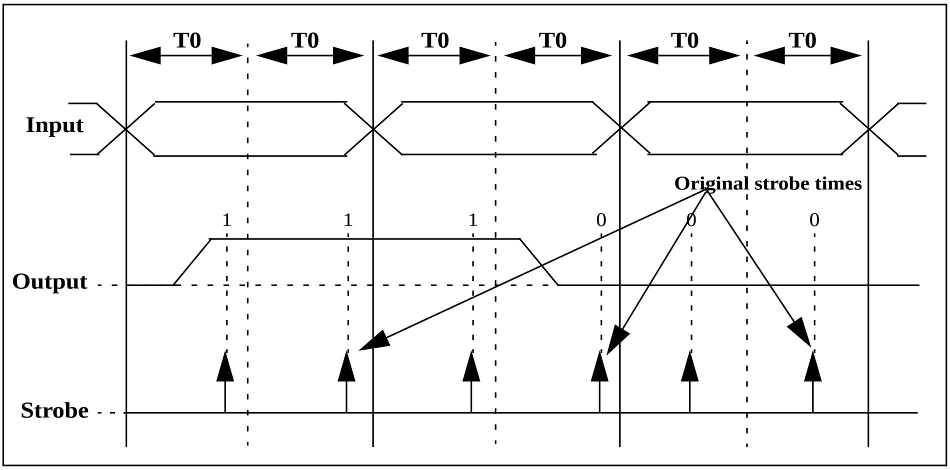


Figure 6.6
Defining a Shorter Test Period

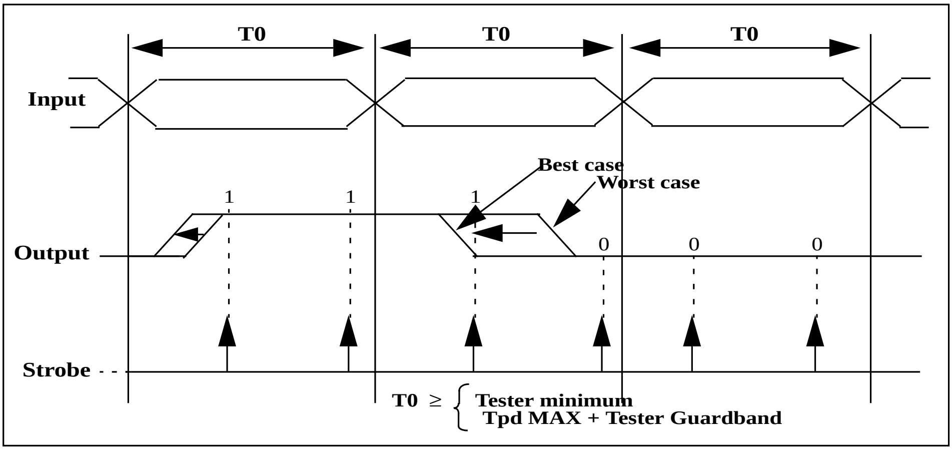


Figure 6.7
Minimum Cycle Time

In most cases, actual (manufactured) parts work faster than their simulated counterparts. The outputs will switch earlier, shifting the output waveforms to the left as shown Figure 6.7.

Notice that if the test period T0 is shortened beyond a certain point, the nominal- and best-case simulation results might be as expected, but the worst-case simulation results could fail functionally because the output could change too close to the cycle boundary or even after the cycle boundary if the cycle time is too small. The 10 ns minimum cycle time is due to the 100 MHz limitation of the tester. Although the real silicon will always be faster than the worst-case simulation prediction, reliable testing must be based on the possibility that real parts could vary enough in performance to reach the worst-case prediction. LSI Logic simulation requires that parts be tested rigorously but not so rigorously as to fail good parts that are simply closer to the worst-case limit than others.

The minimum tester cycle time can be determined by following these three steps:

1. Determine the number of nanoseconds it takes for an input signal’s active edge to change from low to high after the beginning of the tester period.

2. Determine the number of nanoseconds it takes for an output to reach steady state after the input signal’s active edge changes. Add this time to the time determined in Step 1.

3. Add about 2 ns tolerance to the time calculated above. Section 6.7, Output Timesets, explains the origin of the 2X + Y ns tolerance.

For example, in Figure 6.8, CLK is an input signal whose active edge goes from low to high 3.0 ns after the beginning of the tester period. MAT is an output that in the worst case reaches steady state 10.0 ns after the active edge of CLK. Adding another 2X + 4 ns tolerance demands a minimum test period size of 27 ns.

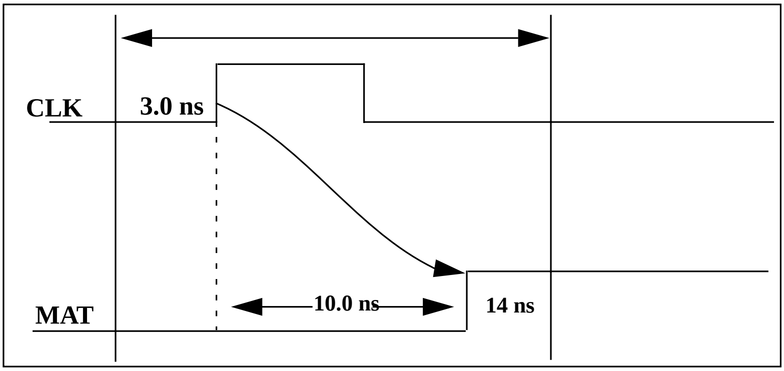


Figure 6.8
Determining the Minimum Test Period

CHAPTER 6 - SECTION 5
Input Timesets

Using input waveforms, you may specify that input signals change state either at the beginning of the test period, or after a fixed delay measure from the beginning of the test period. Alternatively, you may specify that input signals have a narrow duty cycle within the period.

Together, the six input timesets replicate a set of simulation inputs. Four types of waveforms can be used for each of the input timesets as follows:

NRZ Non-Return-to-Zero. The input changes at the beginning of the test period and stays in the same state throughout the entire test period. It is not necessary for the input to change state at every test period, but if it does, it must change state only at the beginning of each cycle. This is the preferred input type. Bidirectional pins must usually use NRZ signals.

DNRZ Delayed-Non_Return-to-Zero. The input changes state only after some fixed delay measured from the beginning of the test period. The delay must be at least X ns to satisfy our lowest cost tester hardware specifications, but no more than the length of the period minus X ns. It is not necessary for the input to change state during each test for all periods. DNRZ waveforms are now allowed for bidirectional pins.

RZ Return-to-Zero after pulsing high. The input has a duty cycle within the test period. The first edge must be at least X ns after the start of the test period. The second edge must occur no later than X ns before the end of the test period. The minimum pulse width is X ns. Note that in any period in which there is no pulse, an RZ signal must be held low. The position or delay of the pulse must be the same in every test period where it occurs. RZ waveforms are not allowed for bidirectional pins.

RTO Return-to-One after pulsing low. The input has a duty cycle within the test period. The first edge must be at least X ns after the start of the period. The second edge must occur no later

> than X ns before the end of the test period. The minimum pulse width is X ns. Note that in any period in which there is no pulse, an RTO signal must be held high. The position or delay of the pulse must be the same in every test period where it occurs. RTO waveforms are now allowed for bidirectional pins. Again all times are specified for our least expense tester.

The four types of input waveforms are illustrated in Figure 6.9. The numbers in the figure refer to the following waveform conventions:

1. Input waveforms cannot change within X ns from the end of the test period.
2. Input waveforms cannot change within X ns from the beginning of the test period.
3. Pulse width must be rater than or equal to 10 ns.
4. Dissimilar Inputs (such as clock and data pins) must not change states within Y ns of each other.

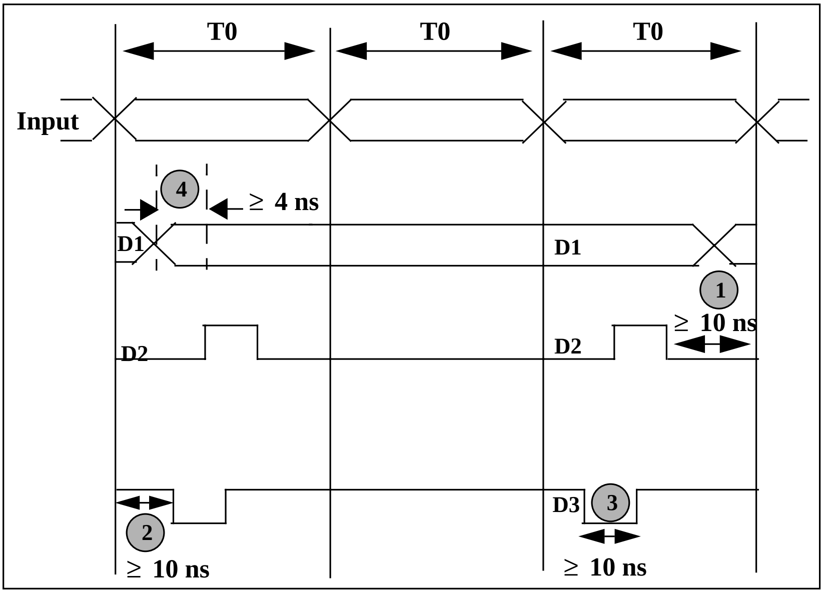


Figure 6.9
Four Types of Input Waveforms

Since they are mechanical devices, the automatic testers are neither as stable nor as precise as the simulation results. When two or more signals change at the same time, the simulation will accurately predict circuit operation, but it will not analyze the effect of any tester head skew (time difference) between the input signals. Even if you specify the input signals in the same timeset, the tester may not be able to apply them at exactly the same time. The skew between any two input pins may be up to Y ns. If the sequence of certain input signals is critical, either apply them at different periods, or separate them in different timesets with sufficient delay between them (normally X ns).

For ideal testing conditions, all inputs should be NRZ type with data pins changing on the non-active edge of the clock pins, as illustrated in Figure 6.10.

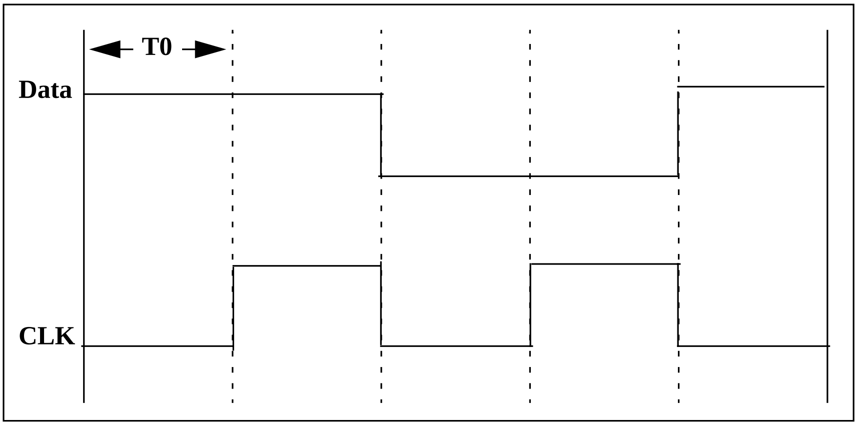


Figure 6.10
Idealized Input Test Patterns

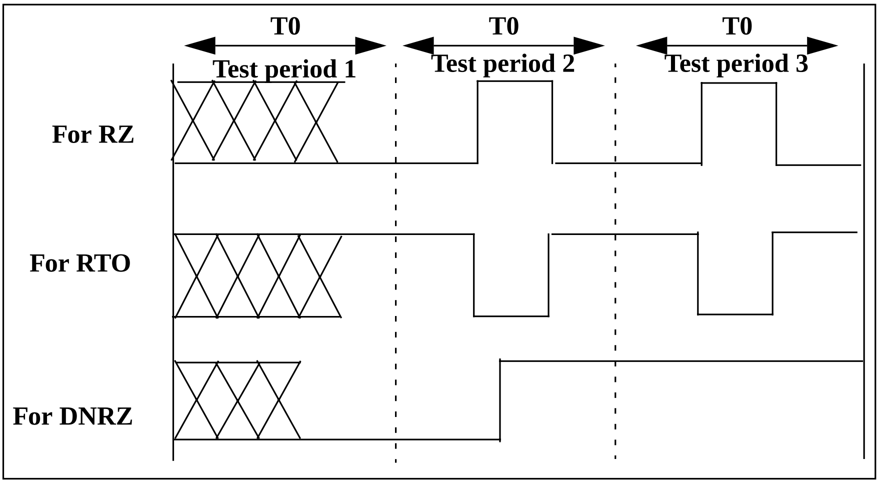


Figure 6.11
Legal Timeset Initialization

Once assigned to an input timeset, inputs cannot be changed to another timeset within the same test block.

The first test vector of each set of patterns must be an X for every input pin and a Z for each bidirectional pin. An example of legal timeset initialization patterns is shown in Figure 6.11. Undriven non-bidirectional inputs should have a value of either 0 or 1. The X or Z values are required only on the first pattern of the set (time 0).

Except for zero delay NRZ, input value changes are not legal at cycle boundaries, and different input timesets must not have the same delay value.

Bidirectional pin conflicts are not allowed during functional or three-state test patterns at any time.

In most situations, you can generate all of the test patterns using only NRZ waveforms for all input pins. However, if the number of patterns required to test the part is larger than about 32K, use DNRZ, RZ, or RTO types of waveforms for test pattern compaction.

Figure 6.12 shows a set of ten test patterns using only NRZ waveforms. In Figure 6.13, the same set of test patterns is compacted by using an NRZ waveform for the DATA in, and an RZ waveform for the CLK in. This arrangement uses only five test patterns.

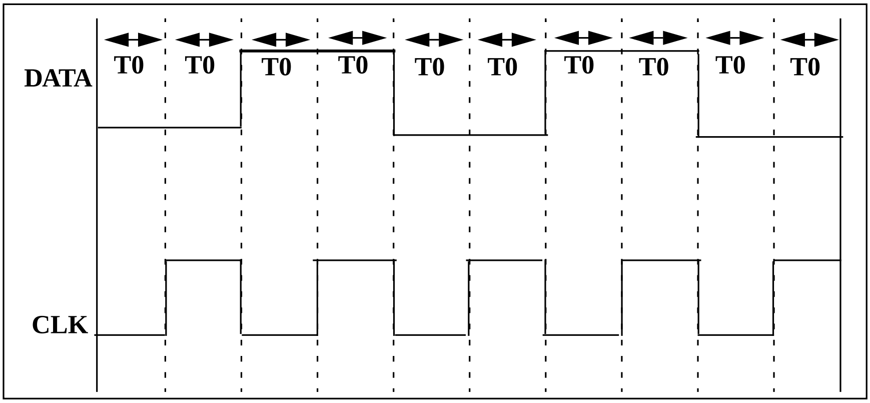


Figure 6.12
Input Test Patterns Using NRZ Waveforms Only

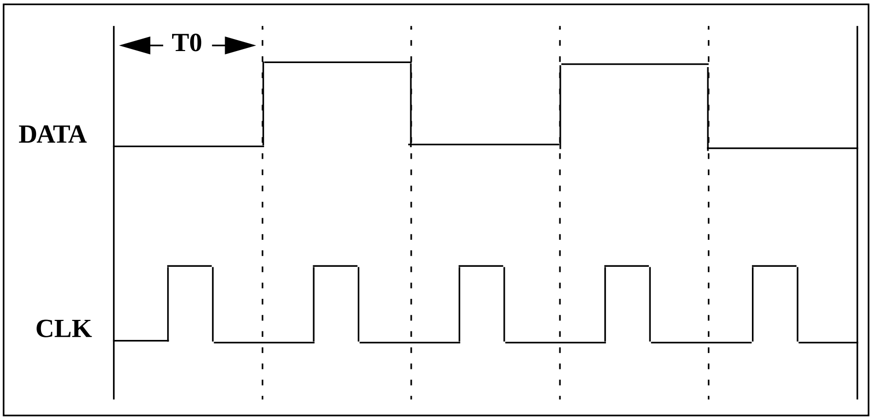


Figure 6.13
Adding RZ to NRZ Reduces the Number of Patterns

CHAPTER 6 - SECTION 6
Examples of Valid and Invalid Waveforms

Figure 6.14 shows how a set of simulation inputs is replicated by input timesets. Look at the figure to observe that Inputs DATA 0 to 7 are NRZ waveforms, CLOCK 0 is an RZ waveform with 4.0 ns delay and 4.0 ns pulse width, ENABLE 1 in a DNRZ waveform with 4.0 ns delay, and SELECT 1 and SELECT 2 are DNRZ types of waveforms with 6.0 ns and 8.0 ns delay, respectively.

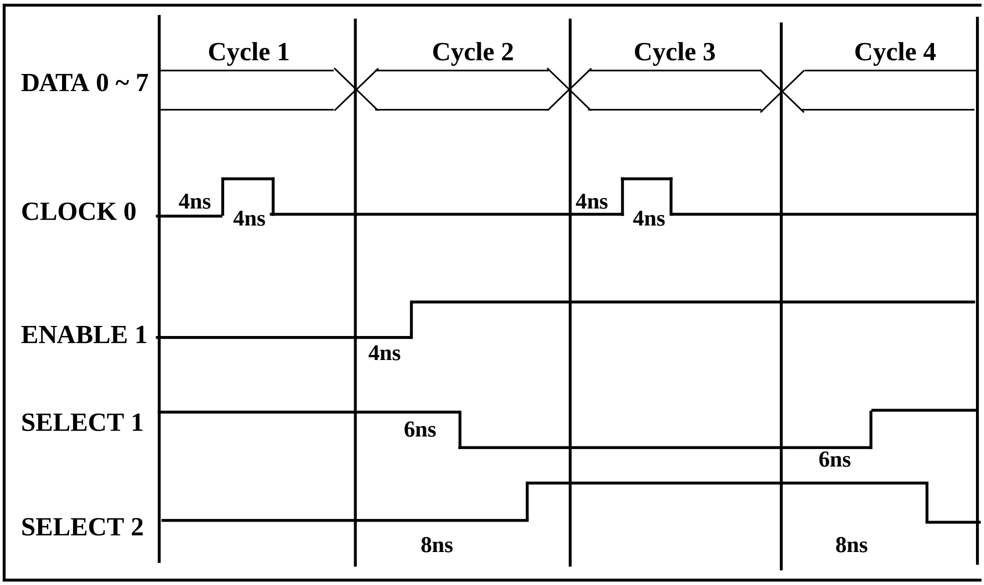


Figure 6.14
Simulation Inputs Replicated by Timing Generators

Figure 6.15 shows a set of simulation inputs that cannot be replicated by input timing generators because of the following errors:

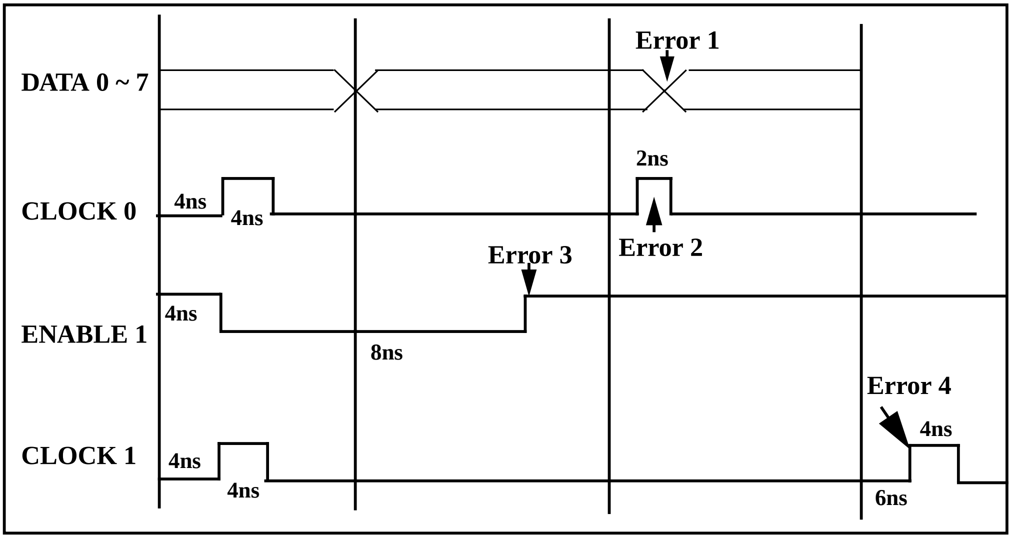


Figure 6.15
Simulation Inputs that Can't be Replicated by Timing Generators

- Error 1 - Inconsistent delay. Illegal use of an NRZ or DNRZ timeset.
- Error 2 - Inconsistent pulse width. Illegal use of an RZ timeset.
- Error 3 - Inconsistent delay. Illegal use of a DNRZ timeset.
- Error 4 - Inconsistent delay. Illegal use of an RZ timeset.

CHAPTER 6 - SECTION 7
Output Strobes

The output strobes are similar to a camera in that they capture or strobe output states at specified points in time: when the outputs reach steady state. The captured states are then compared with simulation results.

LSI Logic uses two output strobes to monitor the circuit’s output pins in order to measure functionality of your design. They are used to capture the steady-state results of the output pins at the end of each test period.

Each set of test patterns should be strobed to detect a functional state. The output strobe checks all valid output pins in every test period to capture the steady-state signal whenever the output is in a 0 or 1 state. The X and Z states, however, are not checked during functional testing. All Z states are checked during three-state testing.

The strobes must be set at the end of the test periods. The minimum test period size is equal to Tpdmax +TOL ns, where Tpdmax is the maximum propagation delay within the circuit.

The TOL tolerance for the tester is illustrated in Figure 6.16. The minimum pulse width of a timeset is X ns. The measured value is compared with the simulated value during this X ns.

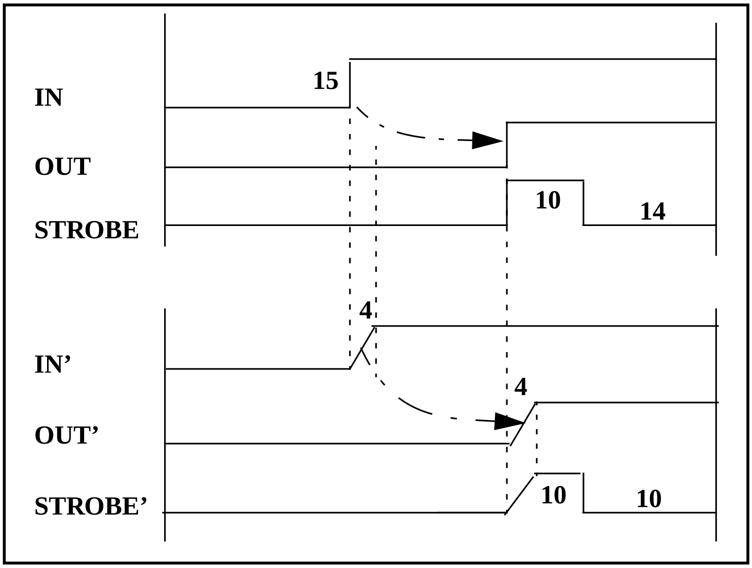


Figure 6.16
Tester Tolerance

Due to the tester’s tolerance, a timeset must not change state within X ns from either the beginning or the end of the test period. Therefore, the trailing edge of the strobe cannot be closer than X ns from the end of the test period. The accounts for 2X ns out of TOL allowed for tester tolerance. An additional Y ns is added to strobe time to compensate for the physical limitations of the tester, as explained in the next paragraph.

When simulating the chip in software, you can specify that an input signal make a low-to-high transition 15 ns after the beginning of the test period. Likewise in the tester, you can program the driver for this input pin to make a low-to-high transition 15 ns after the beginning of the test period. However, it takes Y ns for the voltage level at the output of the driver to reach the input threshold voltage level of the input buffer on the chip. This tolerance factor must also be allowed for when determining the minimum cycle time.

Wafer testing is performed at a much slower test speed due to the large inductance and capacitance of the wafer sorting hardware. Because of this, all output signals must be stable before they are strobed. All output changes caused by input changes in the same time block must be at the final steady state before they are strobed.

The above-mentioned testing requirements may mean testing your design below operational frequency. However, switching performance testing is not limited. Testing switching performance is accomplished by comparing output delays with simulation delays. Overall switching performance is based on the measurement made using the critical path. To make this comparison, use the performance strobe (TS8) to measure the delay of a representative path. In this situation, we make the assumption that the delays of all paths in the chip are within specification if the propagation delay through this path matches the delay from simulation. This assumption is valid since cells on the same silicon die and in the same package are subject to the same external voltage, temperature, and process variations. This delay times will track each other.

CHAPTER 6 - SECTION 8
Rules for Creating Time Blocks

Simulation test patterns are sometimes broken up into time blocks that run sequentially. The term time block is used in Chapter 7, **Prepare Test Patterns for Simulation**, and in this chapter. Because the terminology is used in a different context in this chapter, we will refer to time blocks used for simulation as simulation time blocks, and time blocks used for the tester as tester time blocks. You may have one or more simulation time blocks for every set of test vectors in a single tester time block. The following rules apply to tester time blocks.

1. You may create multiple tester time blocks of tester functional test patterns . A different number of test

blocks are allowed for different technologies. LSI Logic will charge additional NRE charges. Collectively, these are called tester functional patterns.

2. Additionally, one of each of the following time blocks must also be created for the tester:

    - One tester time block for tester VIL/VIH test patterns.

    - One tester time block for three-state test patterns.

    - One tester time block for IDD test patterns.

3. Each tester time block must be independent and must contain its own initialization patterns.

At the beginning of the test program, two initial vectors are always required. One defines the function of all pins at the outset of testing, and one specifies which output pins will be strobed. Each time you want to add or delete output pins from the output pins to be strobed, a new test vector is required. If your circuit contains bidirectional pins, allow two test vectors for each simulation cycle in which the function (direction) of those pins will be switched in order to avoid contention.

If a time block has more than the maximum number of test vectors allowed for the tester you have selected, you have one of two choices: either eliminate some test vectors or divide them into two tester blocks.

The test period size must be the same for each set of test vectors in a given time block. However different time blocks may have different test period sizes, input timing specifications, and output strobe timing.

Within any set of functional test patterns, all output pins must be fully stabilized Y ns prior to the beginning of the next test period. For example, if the slowest output on the chip has a delay of 9.6 ns, the test period for that particular set of patterns must be at least 9.6 ns + Y ns.

The tester time block for the VIL/VIH,IDD, and three-state patterns must be 2 uS (2000 ns).

Remember that all test simulations using test conditions are performed at 25°C, nominal VDD, and worst-case process conditions.

7

Automatic Tester Requirements

CHAPTER 7 - SECTION 1
Overview

Recall from Chapter 6 that the LSI Logic automatic testers use test patterns created during tester functional and parametric simulations. The tester verifies the correct operation of each circuit by clocking in the test patterns, monitoring the output pins, and checking them against the output patterns predicted by simulation.

The following sets of simulations should be prepared:

- system simulation
- tester functional simulation
- VIL/VIH simulation
- three-state functional simulation
- tester IDD simulation
- operationally, patterns for LSI Logic proprietary functions (i.e. cores, mixed signal, etc.)

This chapter provides guidelines you should follow in preparing simulation test patterns, and includes the following major sections:

Section 7.2	**Prepare Simulation Test Patterns**
Section 7.3	**Determine the Output Strobe Timing**
Section 7.4	**Avoid Voltage Spikes and Bus Contention**

CHAPTER 7 - SECTION 2
Prepare Simulation Test Patterns

This section contains information you should know in order to prepare simulation test patterns, including system simulation patterns, tester functional patterns, VIL/VIH patterns, IDD measurement, and three-state patterns.

Prepare System Simulation Input Patterns

System simulation input patterns are used to exercise the network under user-specified environmental conditions. The purpose is to validate the design and its performance. By analyzing the results, you will be able to determine the network's operational limits. The results of this system simulation should indicate to you whether your network behaves as you predict, and what AC performance limits exist. If the system does not perform as expected, you may have to make design modifications and re-create system simulation test patterns.

There are no waveform restrictions in system simulation, which means that input patterns may be either periodic or asynchronous. There are also no timing restrictions; the simulator can analyze a circuit at any frequency, and at a resolution as fine as 10 psec.

Input patterns may be skewed and/or the frequency of clock signals may be changed. The absence of waveform and timing restrictions in system simulation enables the design to be simulated more thoroughly than the automatic tester can test it.

System simulation should be performed in ascending hierarchical order from the most primitive module in the design to the most inclusive. This approach saves time when debugging the networks, and ensures that the foundation of the circuit is sound.

Prepare Tester Functional Simulation Input Patterns

Once the desired results are obtained from system simulation, input patterns should be created for use by the automatic tester. The tester functional simulation patterns enable the tester to verify the device's functionality, measure the propagation delay through the representative path, and measure some Static parametric values such as output voltage levels (VOL, VOH) and three-state output leakage current. Functional simulation is performed only on the top-level module, and should thoroughly exercise the network - i.e., cause every node in it to change state, or toggle, both from a 0 to 1 and from a 1 to 0 at least once. Full scan testing is used today by most designers to create high fault coverage test vectors. And the IDDQ tool in the FlexStream toolset allows designers to check their toggle coverage.

To perform tester functional simulation, it is necessary to extract a subset of the system simulation patterns and scale them down to the timing and waveform requirements of the automatic tester. The recommended way of revising system simulation patterns for use as tester functional patterns is to reduce the frequency at which patterns are applied to the inputs of the network.

Occasionally, multiple test blocks (tester memory loads) may be used to perform complete functional testing. Each block can have different timing characteristics. For example, each block of patterns can test the device with a different test period size, different input timing, and different output strobe times. Different output pins may be monitored on each block of patterns.

Tester functional input patterns must be periodic. They must also conform to one of the following types:

- Non-Return-to-Zero (NRZ)
- Delay-Non-Return-to-Zero (DNRZ)
- Return-to-Zero (RZ)
- Return-to-One (RTO)

A more specific description of recommended tester functional input waveforms may be found in Chapter 6, **Automatic Tester Requirements**.

Prepare Tester VIL/VIH Simulation Input Patterns

Tester threshold simulation patterns are used to test the circuit for input low voltage and input high voltage. These patterns enable the tester to measure the circuit for input threshold voltages. The tester VIL/VIH simulation must be performed under the following conditions: VDD =<Nominal Value>, TEMP=25 C, worst-case process, and a test period of 2000 ns. Only 0 ns delay Non-Return-to-Zero (NRZ) signals are legal for the VIL/VIH test patterns.

The Required VIL/VIH NAND Tree Test Circuit

In order to measure input voltage accurately while ignoring current spikes, every design must have built-in VIL/VIH check circuitry. Figure 7.1 shows the recommended VIL/VIH test circuit. This circuit includes a 2-input NAND gate at every input pin as well as a TESTOUT output pin. The NAND tree provides a purely combinational path from all the input pins to a single TESTOUT pin. Most LSI Logic products already have these gates built into all input and bidirectional buffers.

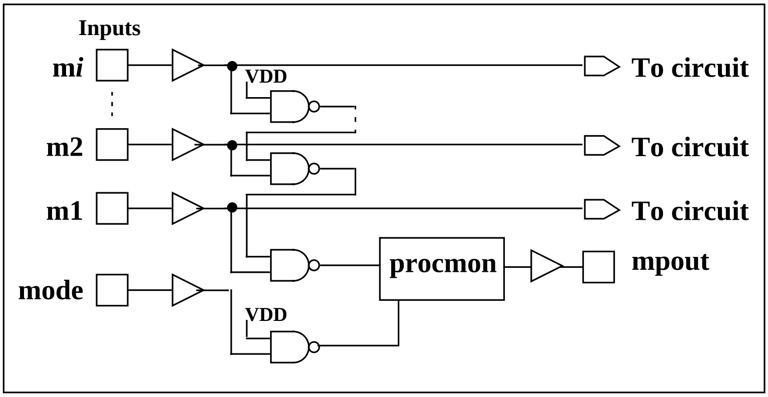


Figure 7.1
The Vil/Vih NAND Tree

The VIL/VIH test is executed with the input driver levels set at VIL, (max) and VIH (min). This leaves no noise margin for the input buffers. While switching, output buffers can generate large current spikes capable of causing the input buffers to misread voltage (and therefore logical) levels.

Recall that a typical output buffer may sink or source 4 mA in the static mode. Dynamically, these buffers generate instantaneous currents of 50 mA or more. Several output buffers switching simultaneously can generate a current spike of one or more amps. As a result, internal storage elements may be latched into an incorrect state, causing incorrect states on the output pins.

To alleviate the noise problem created when a large number of output pins switching simultaneously, only the TESTOUT pin is monitored when the VIL/VIH test us executed on the tester. All other output pins are temporarily masked. The noise generated by the simultaneous switching of a large number of output pins may cause some internal storage elements to go into the wrong state. This does not affect the tester's ability to perform the VIL/VIH test, however, as the tester is monitoring only the TESTOUT pin, but it is recommended that all other outputs accept the TESTOUT output be either three-stated or in a non-switching mode if possible to reduce noise into the ground bus during the sensitive input threshold measurements.

The VIH/VIH test patterns are specific. The first pattern must consist of 1's on the input pins for non-inverting input buffers and 0's on the inputs of inverting input buffers. Then 0's or 1's are applied according to the procedure shown in Figure 7.2 (all inputs are non-inverting in this example). Note that a circuit containing n number of input pins requires the use of at least (n + 1) VIL/VIH test patterns. To test threshold both low-to-high and high-to-low may require twice this number of vectors, but the overall vector length is short and is determined by the number of inputs and bidirect pins in a design.

For networks that do not have an extra output pin for the VIL/VIH test circuit, a regular output pin or pins may be used to multiplex a regular output with the VIL/VIH test output. Figure 7.2 shows VIL/VIH test patterns. Make sure not to use a critical path output, however, if you do intend to multiplex a functional output with this TESTOUT signal.

				TESTOUT	
M_1	M_2	M_3	M_i	i=even	i=odd
1	1	1...	1	1	0
1	1	1...	0	0	1
1	1	0...	0	1	0
1	0	0...	0	0	1
0	0	0...	0	1	0

Figure 7.2
Example NAND Tree Input Patterns

If regular input pins were used to multiplex the output for the VIL/VIH circuit, a set of supplementary test patterns would have to be added to the VIL/VIH test patterns. The purpose of these supplementary test programs would be to check those input pins that were used to multiplex the VIL/VIH test output.

The example circuit in Figure 7.1 includes only unidirectional input pins. On designs that have a bidirectional bus, running the normal parametric NAND tree pattern may cause bidirectional drivers to turn on, creating a bus conflict condition. This problem may be resolved by splitting up the NAND tree into different sections. Figure 7.3 is an example of such a circuit. In this example, bidirectional pins have been disabled with the TE pin so as to place them in the input mode.

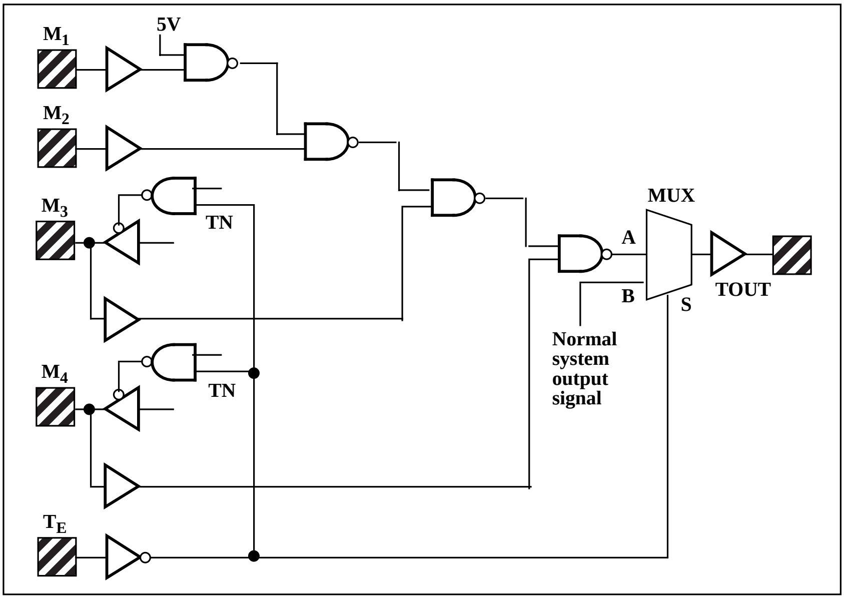


Figure 7.3
A NAND Tree with Bidirects

This allows the input section of these buffers to be tested as if it were an input buffer only. The 2-input NAND gates on the three-state control lines allow these bidirect pins to be forced into the proper state for this test. A MUX is used to direct this NAND tree output to an existing pin to eliminate the need for additional chip pins. The select line of this MUX is controlled by the TE Pin. Figure 7.4 shows the test patterns used for this example.

Only a 0 ns delay, Non-Return-to-Zero (NRZ) signal is allowed for inputs during the VIL/VIH test.

The use of a test-enable input pin in a circuit containing bidirectional pins permits writing of input test patterns as though they were for a circuit containing only unidirectional input pins.

M_1	M_2	M_3	M_i	TE	TOUT
1	**1**	**1**	**1**	**1**	**0**
1	**1**	**1**	**0**	**1**	**1**
1	**1**	**0**	**0**	**1**	**1**
1	**0**	**0**	**0**	**1**	**0**
0	**0**	**0**	**0**	**1**	**1**
Put “0” on input B of 2:1 mux				**0**	**0**

Figure 7.4

Test Patterns for NAND Tree with BiDirects

Prepare Test Patterns for the Chip IDD Measurement

Only a few patterns are needed to set up the circuit for Static leakage current (the IDD test). Test conditions should be set up in which all Static current is turned off to facilitate easy measurement of Static leakage. The test condition must eliminate the following sources of Static current:

- floating internal nodes
- internal bus contention
- a low state on an input or output pin with a pull-up resistor
- a high state on an input or output pin with a pull-down resistor
- any undriven input without a pull-up or pull-down resistor

- enabled memory sense amplifiers
- static current in Mixed Signal and Analog I/Os

The first two items mentioned above take priority over the others if all cannot be attained.

The IDD test patterns must be executed under the following conditions: V_{DD} = <Nominal>, temperature = 25° C, process = worst case, test period = 2000 ns.

Prepare Three-State Test Pattern

Three-state test patterns are used to test the three-state enable circuitry. These test patterns are typically very short, consisting of only a few vectors. Three-state functional test patterns must exercise every possible way that the three-state outputs can be three-stated.

When the automatic tester executes the tester three-state functional patterns, the output pins are monitored for the high impedance (Z) state. All output pins in the 1 or 0 state are masked.

You must execute the three-state simulation under the following conditions: V_{DD} = <Nominal>, temperature = 25° C, process = worst case, test period = 2000 ns.

To summarize, there are at least four sets of tester patterns that you must create:

1. Tester functional
2. Tester VIL/VIH
3. Tester IDD
4. Tester three-state

Additional sets of patterns may be required if you are using LSI Logic proprietary functions. Predefined simulation files are provided by LSI Logic if you choose to use them.

CHAPTER 7 - SECTION 3
Determine the Output Strobe Timing

Recall from Chapter 6 that two strobes are available. Both strobes allow a direct comparison of expected (simulated) and actual (measured) outputs. The first strobe is placed near the end of each test period to test the functional operation of most of the output pins. The second strobe can be used to get a feeling for the performance of the chip. Place the second strobe earlier in time than the first strobe within the cycle to measure a representative path delay using the remaining output pins, as shown in Figure 7.5.

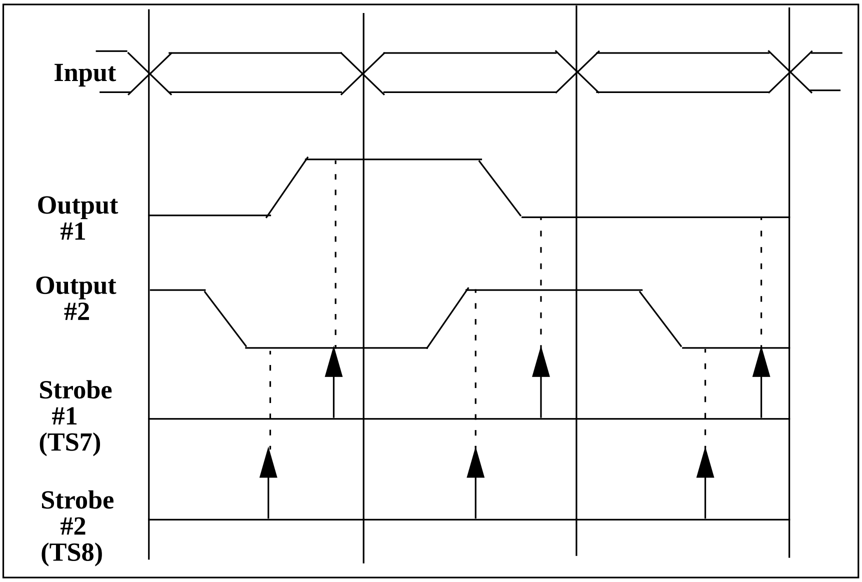


Figure 7.5
Using Output Strobe #2

When choosing a representative path for the second strobe, you should pick one of the longest paths in the design because it will contain the most internal macrocells and will, therefore, be representative of most of the circuit elements, as shown in Figure

7.6. In a shorter path, the delays being measured are really those of the input and output buffers and not the delay of the internal macrocells. An example of this is shown in Figure 7.7.

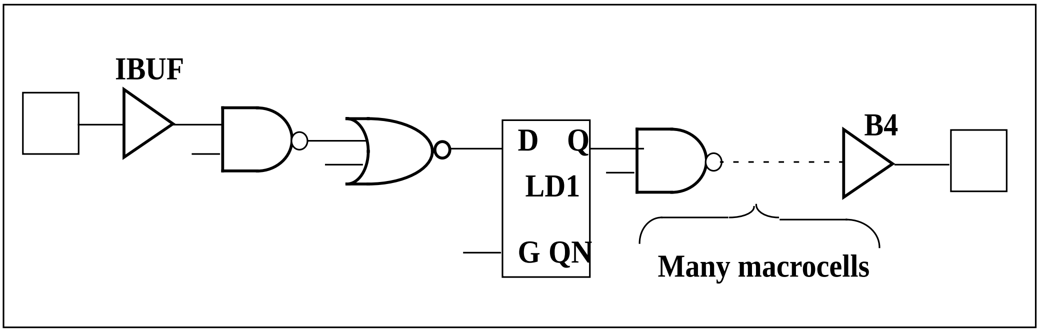


Figure 7.6
Long Path Delay, Dominated by Chip Circuitry

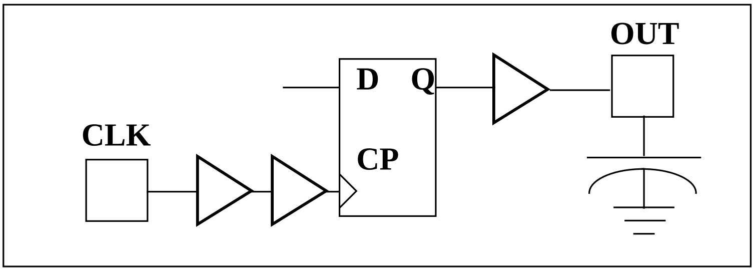


Figure 7.7
Short Path Delay, Dominated by Tester Capacitance

CHAPTER 7 - SECTION 4
Avoid Voltage Spikes and Bus Contention

This section describes information about how to avoid voltage spikes and bus contention. It is helpful to understand this information before creating simulation input patterns.

Avoid Voltage Spikes

A voltage spike, also known as a “glitch” is the abrupt transition on any node from one state to another and then back again. Voltage spikes occur when the propagation delay (or t_{PD}) of a given gate is longer than the duration of the input signal pulse width. The reason to avoid voltage spikes is that they prevent a gate from reaching either a logical 1 or a logical 0 state. Figure 7.8 depicts a 2-input NOR gate (NR2) with a low-to-high propagation delay of 0.6 ns and a high-to-low propagation delay of 0.2 ns.

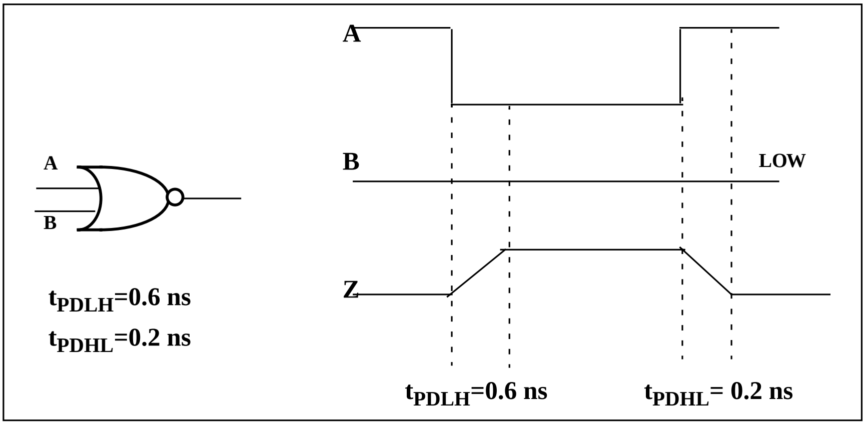


Figure 7.8
NOR Gate Output Edge Asymmetry

Figure 7.9 shows the conditions in which a voltage spike will occur on the NR2 gate in Figure 7.8.

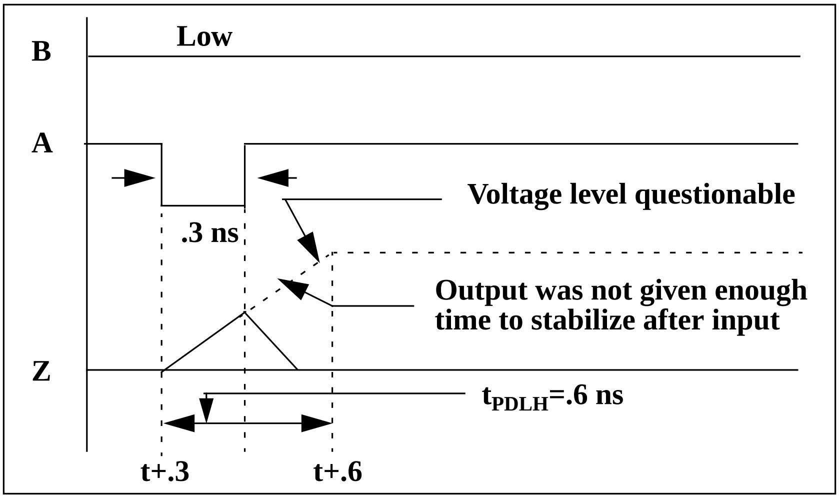

B
Low
A
Voltage level questionable
.3 ns
Output was not given enough time to stabilize after input
Z
t_{PDLH}=.6 ns
t+.3
t+.6

Figure 7.9
NOR Output Asymmetry Generating a Spike/Glitch

In Figure 7.9, input B is low and input A is high. When input A makes the high-to-low transition at time t, output Z begins to go high. Input A returns to the high state after 0.3 ns; this pulls the value of output Z back to a low, which it attains in 0.1 ns. The solid line for output Z depicts its transition back to the low state; the dotted line shows the transition output Z would have made to the high state if output A had not returned to a logical 1 at t+0.3 ns.

To prevent the voltage spike, input A must hold steady at logical 0 for 0.6 nsec or more, affording output Z time enough to change from logical 0 to logical 1, as illustrated in Figure 7.10.

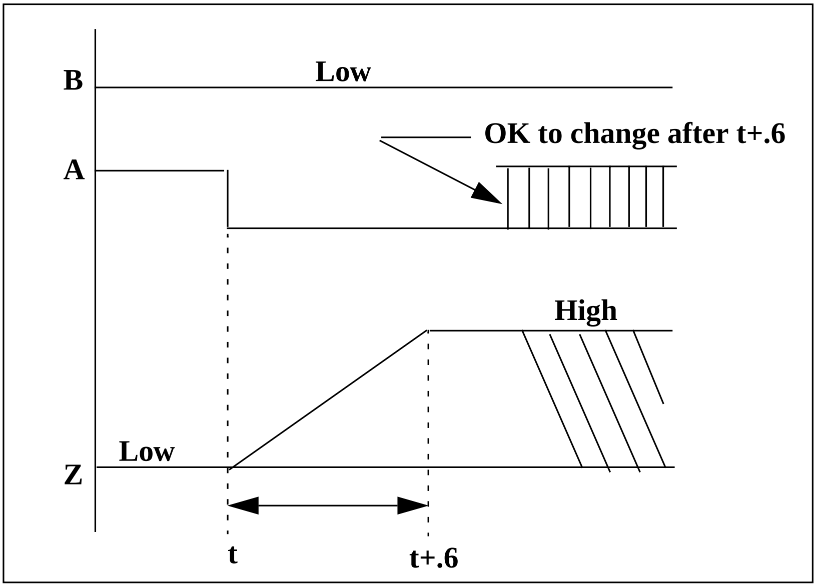


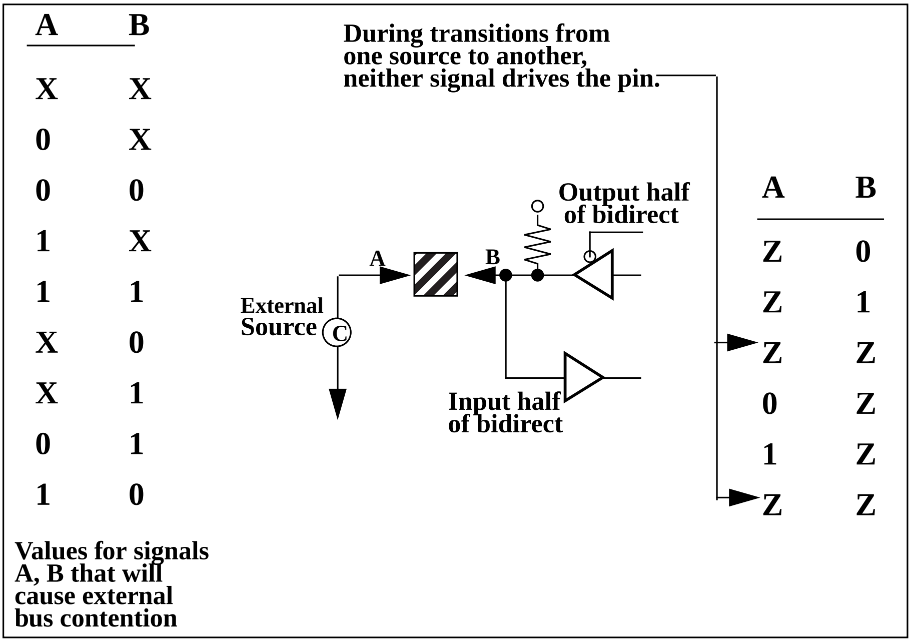
Figure 7.10
Avoiding the Glitch

Figure 7.10 shows the conditions under which a voltage spike will not occur on the same NR2 gate.

Avoid Bus Contention

Bus contention occurs when an output buffer is enabled, and is outputting a logical value that differs from that of the external driving source. External bus contention can be avoided by three-stating the output buffer whenever the bus is being driven by an external source. Conversely, when the bus is being driven by an output buffer, the external source should not be driving the pin. Bus contention is avoided by allowing only one driver to drive a given bus at a time. Buses must not be allowed to contend during any test pattern.

Figure 7.11 illustrates how external bus contention can be avoided.



During transitions from one source to another, neither signal drives the pin.
Output half of bidirect
A
B
External Source
C
Input half of bidirect

A	B
X	X
0	X
0	0
1	X
1	1
X	0
X	1
0	1
1	0

Values for signals A, B that will cause external bus contention

A	B
Z	0
Z	1
Z	Z
0	Z
1	Z
Z	Z

Figure 7.11

Avoiding Bus Contention

Signal A is created by an external source C. Signal B is generated by the output buffer. The table to the left of the drawing shows values for A and B that will result in external bus contention. Bus contention can occur when A or B (or both) are in an unknown state (X), and when A and B drive the pad at the same time, or when not enough time is allowed when switching between driving sources.

The table to the right of the drawing by contrast, show values that will not result in external bus contention. Bus contention will not occur when either A is in a high impedance (Z) state, while B is in a defined state (1 or 0), or vice-versa, or when both A and B are in the high impedance (Z) state.

8

The Beginning

The material in this book should be digested before you begin your ASIC design. Most of the techniques discussed here have been tried or used in real designs at LSI Logic. Techniques that do not work or have caused simulation problems in ASICs have been analyzed, and this book has proposed alternatives to resolve difficulties caused by the original logic configurations.

The purpose of this book is to save you valuable design time by comparing design trade-offs early in your design cycle and pointing out things that are likely to cause problems if used in your design.

Before you begin your design, assemble all of the information necessary to complete your ASIC design. LSI Logic technology design manuals, datasheets and databooks are absolutely required items in order to begin a design. Information on packaging is critical for some trade-offs that may need to be made relating to power and ground, ESD protection, or pin count. If you will be needing functions or cells for your design that are not in any existing LSI Logic libraries, you must request these functions and fells as early as possible, probably during the first technical review. Know who your

LSI Logic ASIC Customer Engineer (ACE) is from the first day you begin your design work. That person will be your technical contact throughout the duration of your interface with LSI Logic. Use them as a resource.

More on the Testing Issue

Testing is becoming a very complex issue as designs grow in size. Various test techniques have been addressed in this book, but many others not covered are available to you as a designer. You will need to partition your logic in such a way as to decide what goes on each ASIC you design and then build in logic that will allow the design to be tested most efficiently. Sometimes, a few additional test points or access points can reduce the number of tests by a significant amount. If you wish to use Memory BIST, full scan ATPG, JTAG boundary scan, or other techniques, you must discuss the ground rules early in your design with your ACE engineer. LSI Logic has a series of Application Notes in the ASKK documentation package which describe the various test techniques in great detail.

Training Classes

Training classes are available for our FlexStream Design Tools on a regular basis at our Milpitas California facility. All information related to training classes and material is available on our Customer Education WEB page located at: (Internet site)

http://www.lsilogic.com/techsupp/training/index.html

For LSI Logic personnel, you can reach a more expanded list of training classes at: (Intranet site)

http://webvision.lsil.com

All of our tools training material is downloadable from these two

WEB sites. LSI employees can access on-line classes through a PC interface using a WEB Browser or via UNIX machines which have been configured to receive video and audio media files.

Additional classes are available on-line to LSI employees at this time. These cover topics such as: Test Methodology, Packaging, ESD Protection, and so on. These will become available to customers over time as we add Internet media capability.

In the future, we will be providing expanded WEB based classes on our Internet WEB site and will also add “live” classes on the WEB via “chat room” type of access to keep travel costs down and keep providing more timely training. See our WEB pages for details.

Questions about this Book

If you should have questions about the material contained in this book, please contact the LSI Logic Customer Education Department at:

LSI Logic Corporation

1551 McCarthy Blvd., MS E-197

Milpitas, CA 95035

(408) 433-7687

One of the members of our Customer Education Department will respond to your questions. We are here to serve you and would appreciate any feedback related to this document or any other Training issues that you would like to discuss with LSI Logic.

Summary

LSI Logic wants to make your ASIC design experience a successful one. It is our philosophy that each design must work right the first time; this requires that we provide you the highest quality products, software tools, and training in the industry. Our Training WEB sites Training classes, and reference documents are available to provide you with the information necessary to make your job as easy as possible when you are using LSI Logic's ASIC products and design tools.

Our floorplan for your successful design includes:

- Attitude -
  that together we will make your design a success.
- Service -
  which we provide at every step of the design process.
- Innovative ASIC technology -
  keeping you and your products ahead of the competition.
- Customer satisfaction -
  an LSI Logic priority.

Thank you for choosing LSI Logic as your ASIC vendor.