An Introduction to Application Specific Integrated Circuit (ASIC) Design

Bob Zeidman
Zeidman Consulting

1. Introduction

Once a design is committed to silicon, you want it to be right. Time, money, and your reputation can be wasted on a malfunctioning ASIC. This paper will cover the steps to take before fabrication that will minimize your chances of failure and maximize your chances of success for first silicon. These steps include how to write a specification, top-down design, simulation, test vector generation, and good procedural practices.

This paper is aimed at the engineer who is facing an ASIC design or wants to be prepared for one. Those who have never designed an ASIC will find this paper especially beneficial, and experienced ASIC designers will find this paper to be useful reference.

1.1 What is an ASIC?

An Application Specific Integrated Circuit, or ASIC, is a chip that can be designed by an engineer with no particular knowledge of semiconductor physics or semiconductor processes. The ASIC vendor has created a library of cells and functions that the designer can use without needing to know precisely how these functions are implemented in silicon. The ASIC vendor also typically supports software tools that automate such processes as synthesis and circuit layout. The ASIC vendor may even supply application engineers to assist the ASIC design engineer with the task. The vendor then lays out the chip, creates the masks, and manufactures the ASICs.

Just as a board designer does not need to have an intimate knowledge of the integrated circuits that he places on a PC board, the ASIC designer does not need such knowledge of the individual cells that are used in an ASIC design. This is not to say that no knowledge is required. Just as a PC board designer needs to know interface characteristics such as capacitive loading and trace impedance, an ASIC designer needs to understand the ASIC vendor's specifications for the particular library of cells and functions that he is using in his design.

1.2 Gate Array vs. Standard Cell

There are two varieties of ASICs, and each has its own advantages - gate arrays and standard cells. Each variety has a different architecture as shown in Figure 1. These architectural differences result in different manufacturing techniques, different costs, and different development times. Depending on your requirements, one type of ASIC will be optimal, and it is good to understand which one fits your needs before beginning the design process.

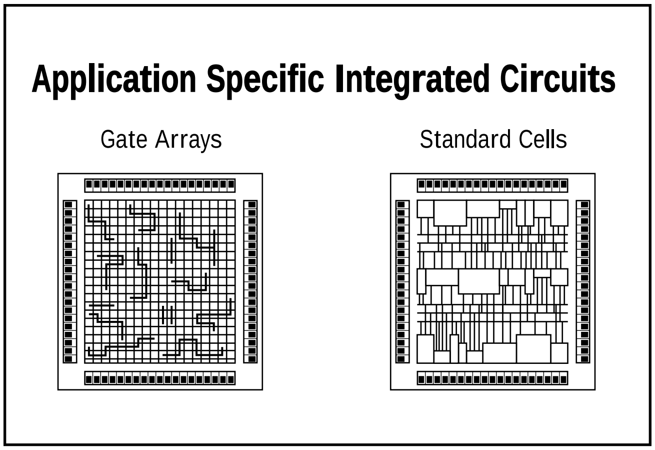


Figure 1

1.2.1 The Gate Array

The gate array consists of rows and columns of regular transistor structures. Each basic cell, or gate, consists of the same small number of transistors which are not connected. In fact, none of the transistors on the gate array are initially connected at all. The reason for this is that the connection is determined completely by the design that you implement. Once you have your design, the layout software figures out which transistors to connect. First, your low level functions are connected together. For example, six transistors could be connected to create a D flip-flop. These six transistors would be located physically very close to each other. After your low level functions have been routed, these would in turn be connected together. The software would continue this process until the entire design is complete.

The ASIC vendor manufactures many unrouted die which contain the arrays of gates and which it can use for any gate array customer. An integrated circuit consists of many layers of materials including semiconductor material (e.g., silicon), insulators (e.g., oxides), and conductors (e.g., metal). An unrouted die is processed with all of the layers except for the final metal layer that connects the gates together. Once your design is complete, the vendor simply needs to add the last metal layer to the die to create your chip.

The advantages of gate arrays is that they have a fast turnaround time. In addition, since the vendor can produce many unrouted arrays for many customers, each customer shares in some of the development cost, resulting in a lower development charge, also know as non-recurring expense (NRE).

1.2.2 The Standard Cell

The standard cell ASIC is designed using cells of transistors which are already connected together and compactly routed to form higher level functions such as flip-flops, adders, and counters. The ASIC designer connects these cells together just as he would connect TTL packages together on a PC board. The software that lays out a standard cell ASIC attempts to place these cells on the die and connect them together in as efficient a way as possible.

Since each cell consists of all of the material layers needed to produce the transistors and to connect them, and since each customer's design is different, each standard cell ASIC must be created from scratch. This results in a much longer turnaround time than for a gate array. Each mask to produce each layer is custom for each user. Therefore, customers cannot share development costs as they can with gate arrays.

The advantages of a standard cell approach is that the resulting die is typically much smaller than the equivalent gate array. For a gate array, the die size is fixed and many transistors in the array will typically not be used in the design. For a standard cell design, only those transistors that are needed are placed on the die. A smaller die results in more die per wafer, which results in a smaller cost per part. This can be a big advantage for parts which are used in high volume.

Another advantage is that standard cell ASICs can use very complex functions if those functions are available as cells in the vendor's library. Many vendors include microprocessor cores in their libraries. These cells would be very difficult to design and would take up a great deal of die area if they were implemented in a gate array.

1.3 Which ASIC type is right for you?

Which ASIC type to use depends on your project and budget. Use gate arrays when you want to hold down the initial cost, when you need fast turnaround on prototypes, and when you expect low production volumes. Use standard cells when you need to implement very complex functions and when you expect high production volumes.

Gate Arrays vs. Standard Cells

	Gate Array	Standard Cell
NRE	low	high
Per piece	high	low
Utilization	low	high
Turnaround time	fast	slow
Customizability	low	high

Figure 2

2. Design Issues

There are many design issues that are common to both ASIC design and other forms of digital design. In the next sections of this paper, we will discuss those areas that are unique to ASIC design or that are particularly critical to ASIC design.

2.1 Top-Down Design

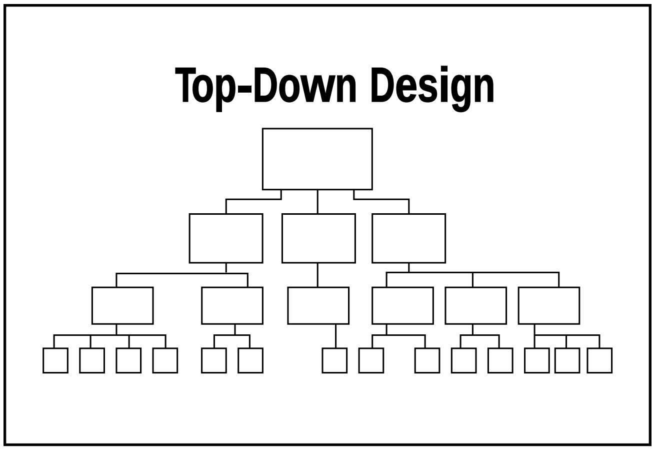


Figure 3

Top-down design is the design method whereby high level functions are defined first, and the lower level implementation details are filled in later. An ASIC schematic can be viewed as a hierarchical tree as shown in Figure 3. The top level block represents the entire ASIC. Each lower level block represents major functions of the ASIC. Intermediate level blocks may contain smaller functionality blocks combined with gate-level logic. The bottom level contains only gates and macrofunctions which are vendor-supplied high level functions. Fortunately, schematic capture software and hardware description languages used for ASIC design easily allows use of the top-down design methodology.

Top-down design is the preferred methodology for ASIC design for several reasons. First, ASICs often incorporate a large number of gates and a very high level of functionality. This methodology simplifies the design task and allows more than one engineer, when necessary, to design the chip. Second, it allows flexibility in the design. Sections can be removed and replaced with a higher-performance or optimized designs without affecting other sections of the ASIC.

Also important is the fact that simulation is much simplified using this design methodology. Simulation is an extremely important consideration in ASIC design since an ASIC cannot be blue-wired after production. For this reason, simulation must be done extensively before the ASIC is sent for fabrication. A top-down design approach allows each module to be simulated independently from the rest of the design. This is important for complex designs where an entire design can take weeks to simulate and days to debug. Simulation is discussed in more detail later in this paper.

2.2 NAND Gates

On the lowest level, most ASIC technologies are optimal if they are designed using NAND gates as opposed to other kinds of gates such as AND, OR, or NOR gates. First, NAND gates are typically more symmetric. In other words, the rise and fall times are close to equal which means you are less likely to have timing problems. Also, NAND gates are implemented with the fewest levels of transistors, making the propagation times lower than for other basic gates.

If you design your ASIC using schematic capture tools, you will produce faster parts if you use NAND gates. If you use synthesis tools to convert a high level description to a gate level design, you may want to check that the tool synthesizes into NAND gates, particularly if you are concerned about performance.

2.3 Macrofunctions - Soft and Hard

Macrofunctions are pre-defined functions which are common to many designs and can be used in blocks without understanding their internal design. They might include designs for counters, flip-flops, adders, or registers. These macros are used like ICs are used in a PC Board design. In fact, many vendors supply common 74LS part functions in their libraries of macrofunctions. They are useful because they allow the designer to integrate common higher level functionality into the design without needing to design these parts each time.

There are two types of macrofunctions - hard and soft - and it is important to know the difference in order decide when to use each type. Soft macros are high level blocks that include the low level gates that are needed to create the specific function. When the ASIC is laid out, these gates will be treated like every other gate in the design and will be placed on the die and routed. A hard macro, on the other hand, consists of a number of gates that have been placed and routed together to achieve optimal performance. The hard macro is treated like a single gate, and the relative placement of each internal gate cannot be modified.

The advantage to a soft macro is that it can be easily modified to fit your own particular need. For example, if you have a macro for an up/down counter, but you only need an up counter, you can go into the macro and take out those gates that are used for counting down. In this way, you did not need to design the counter from scratch, and you could eliminate any unnecessary functionality from the macro.

Another advantage of soft macros is that they can be lifted from one design and, with slight or no changes, be incorporated into another design, even a design using a different technology. Soft macros are easily routed in the design since the individual gates can be placed anywhere in the layout. The timing between a soft macro and the other blocks in the design can be optimized by the place-and-route software. The disadvantages of soft macros are that, like the rest of your design, the timing is not completely predictable until the entire design has been routed. This is because the timing will depend on the routing lengths.

The advantage of a hard macro is that the timing is completely predictable since the gate layout cannot change. Hard macros are usually designed to optimize performance and get the best timing possible from the process. The disadvantage is that they cannot be changed to eliminate unneeded functionality or to incorporate additional functionality. Since they are larger blocks, they may not be easily incorporated into the layout. The timing between a hard macro and the rest of the design may not be good, and hard macros are not portable to other technologies. The cells of a standard cell ASIC are hard macros. Higher level functions may be either hard or soft macros. A comparison of hard and soft macros is shown in Figure 4.

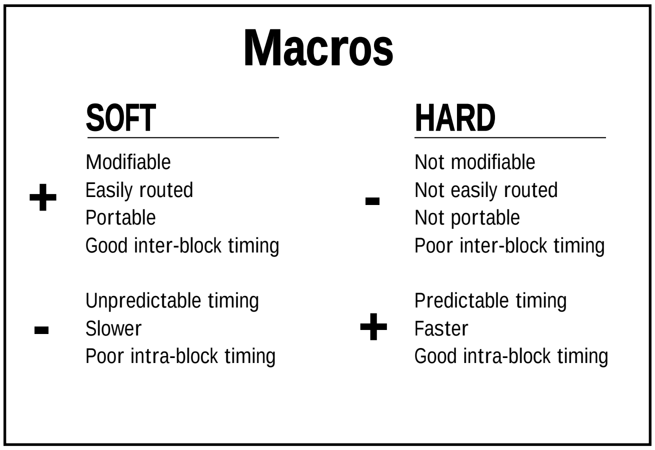

Macros
SOFT
+
Modifiable
Easily routed
Portable
Good inter-block timing
-
Unpredictable timing
Slower
Poor intra-block timing
HARD
-
Not modifiable
Not easily routed
Not portable
Poor inter-block timing
+
Predictable timing
Faster
Good intra-block timing

Figure 4

2.4 Synchronous Design

One of the most important concepts in ASIC design, and one of the hardest to enforce on novice ASIC designers, is that of synchronous design. Once an ASIC designer uncovers a problem due to asynchronous design and attempts to fix it, he or she usually becomes an evangelical convert to synchronous design. This is because asynchronous design problems are due to marginal timing problems that may appear intermittently, or may appear only when the ASIC vendor changes its semiconductor process. Asynchronous designs that work for years in one process may suddenly fail when the ASIC is manufactured using a newer process.

Synchronous design simply means that all data is passed through combinatorial logic and flip-flops which are synchronized to a single clock. No signal that is generated by combinatorial logic can be fed back to the same group of combinatorial logic without first going through a synchronizing flip-flop. Clocks cannot be gated - in other words, clocks must go directly to the clock inputs of the flip-flops without going through any combinatorial logic.

The following sections cover common asynchronous design problems and how to fix them using synchronous logic.

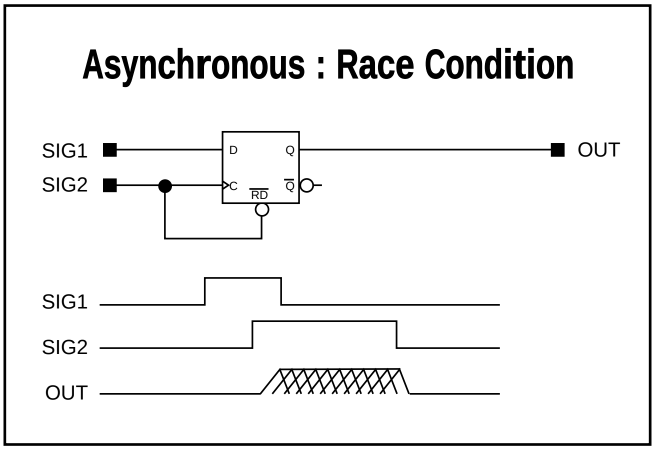


Figure 5

2.4.1 Race conditions

Figure 5 shows an asynchronous race condition where a clock signal is used to reset a flip-flop. When SIG2 is low, the flip-flop is reset to a low state. On the rising edge of SIG2, the designer wants the output to change to the high state of SIG1. Unfortunately, since we don’t know the exact internal timing of the flip-flop or the routing delay of the signal to the clock versus the reset input, we cannot know which signal will arrive first - the clock or the reset. This is a race condition. If

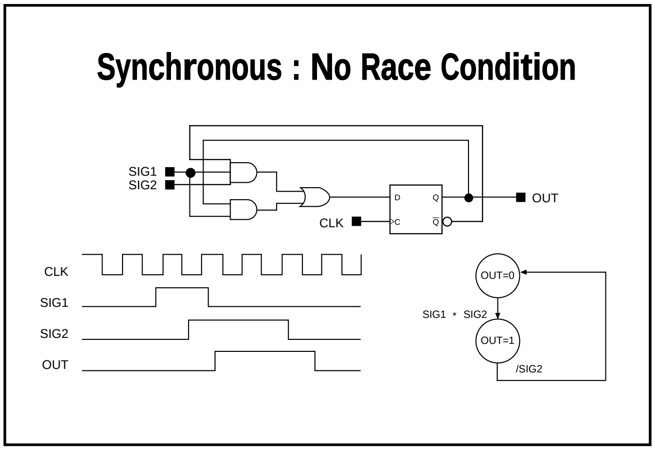


Figure 6

the clock rising edge appears first, the output will remain low. If the reset signal appears first, the output will go high. A slight change in temperature, voltage, or process may cause an ASIC that works correctly to suddenly work incorrectly. A more reliable synchronous solution is shown in Figure 6. Here a faster clock is used, and the flip-flop is reset on the rising edge of the clock. This circuit performs the same function, but as long as SIG1 and SIG2 are produced synchronously - they change only after the rising edge of CLK - there is no race condition.

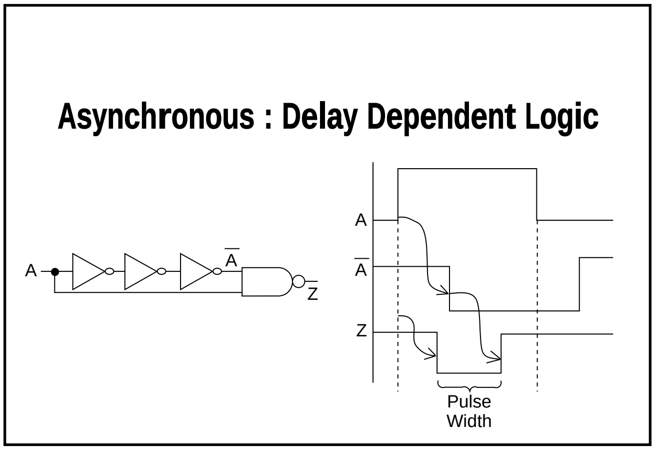


Figure 7

2.4.2 Delay dependent logic

Figure 7 shows logic used to create a pulse. The pulse width depends very explicitly on the delay of the individual logic gates. If the process should change, making the delay shorter, the pulse width will shorten also, to the point where the logic that it feeds may not recognize it at all. A synchronous pulse generator is shown in Figure 8. This pulse depends only on the clock period. Changes to the process will not cause any significant change in the pulse width.

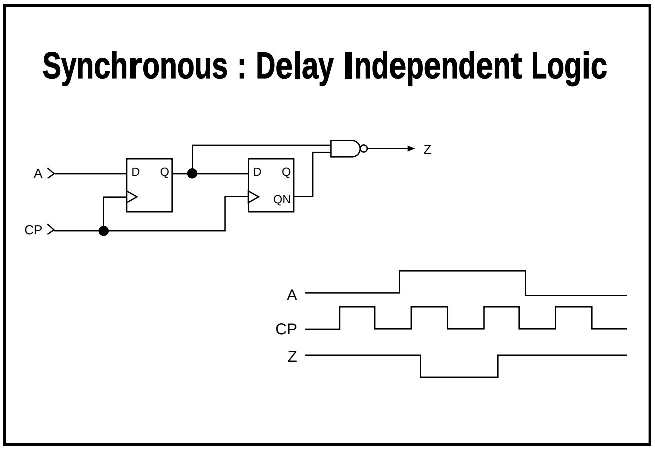


Figure 8

2.4.3 Hold time violations

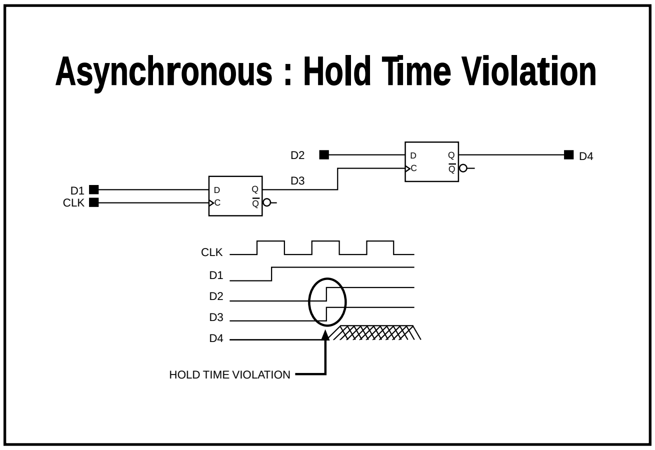


Figure 9

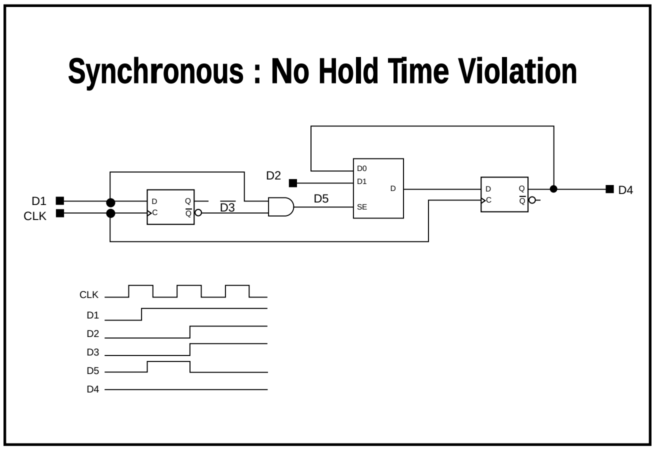


Figure 10

Figure 9 shows an asynchronous circuit with a hold time violation. Hold time violations occur when data changes around the same time as the clock edge. It is uncertain which value will be registered by the clock. The circuit in Figure 10 fixes this problem by putting both flip-flops on the same clock and using a multiplexer to either load new data or keep the previous data.

2.4.4 Glitches

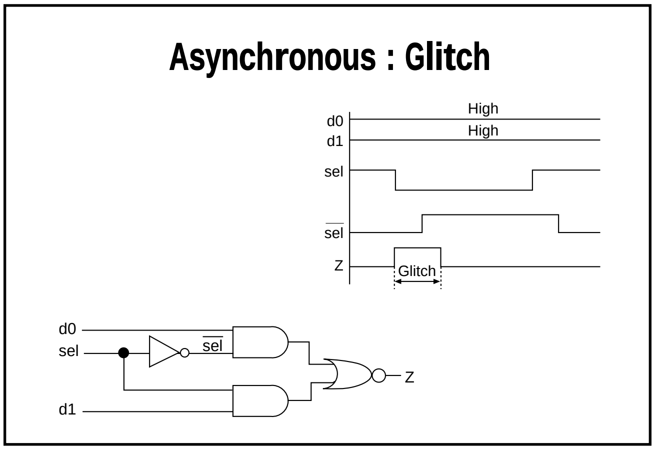


Figure 11

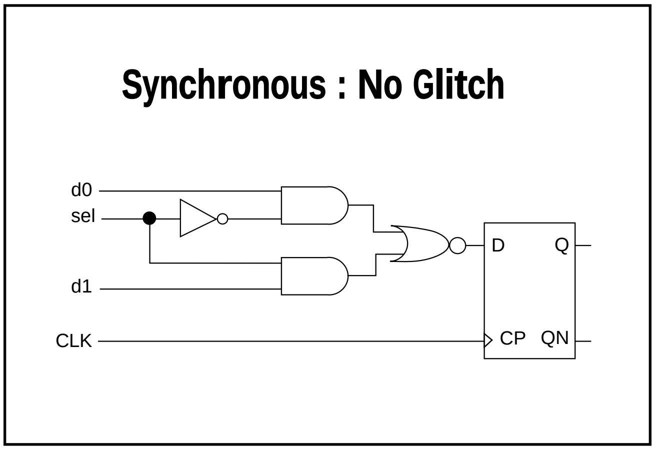


Figure 12

A glitch can occur due to small delays in a circuit such as that shown in Figure 11. An inverting multiplexer contains a glitch when switching between two signals, both of which are high. Yet due to the delay in the inverter, the output goes high for a very short time. Synchronizing this output by sending it through a flip-flop as shown in Figure 12, ensures that this glitch will not appear on the output and will not affect logic further downstream.

2.4.5 Bad clocking

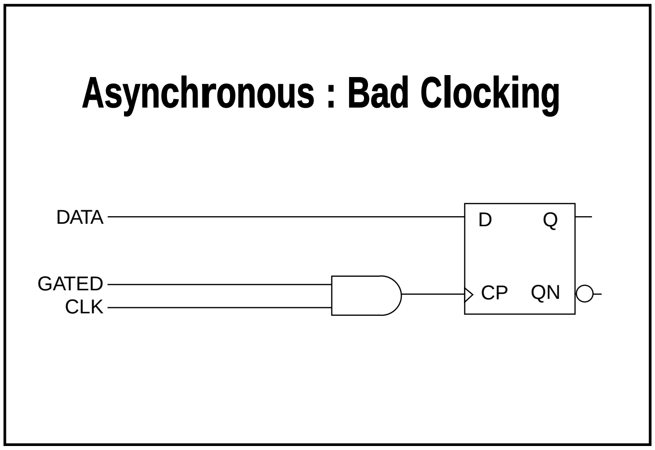


Figure 13

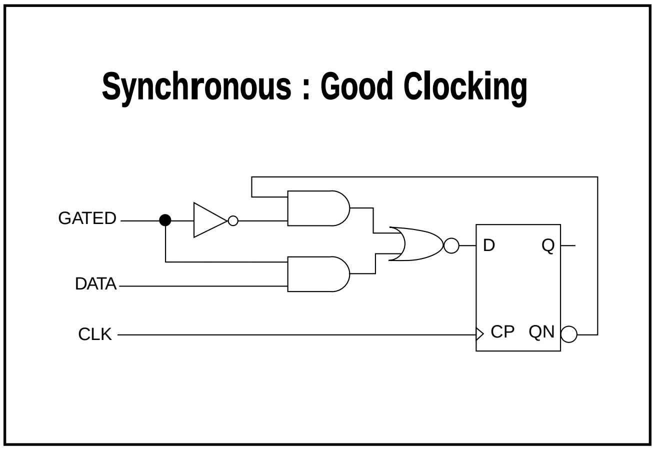


Figure 14

Figure 13 shows an example of asynchronous clocking. This kind of clocking will produce problems of the type discussed previously. The correct way to enable and disable outputs is not by putting logic on the clock input, but by putting logic on the data input as shown in Figure 14.

2.4.6 Metastability

One of the great buzzwords, and often misunderstood concepts, of synchronous design is metastability. Metastability refers to a condition which arises when an asynchronous signal is clocked into a synchronous flip-flop. While ASIC designers would prefer a completely synchronous

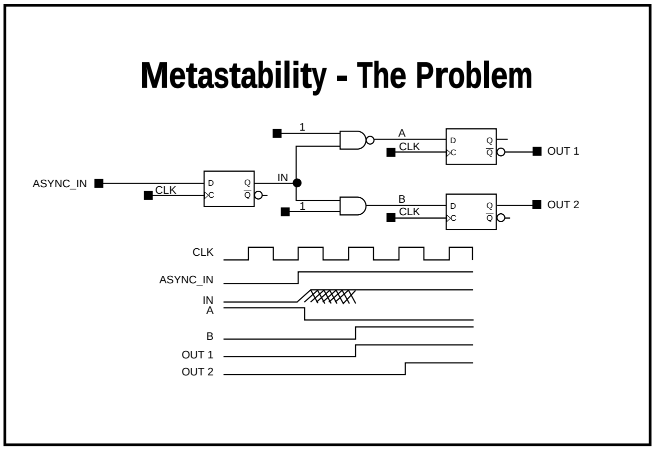

Metastability - The Problem

Figure 15

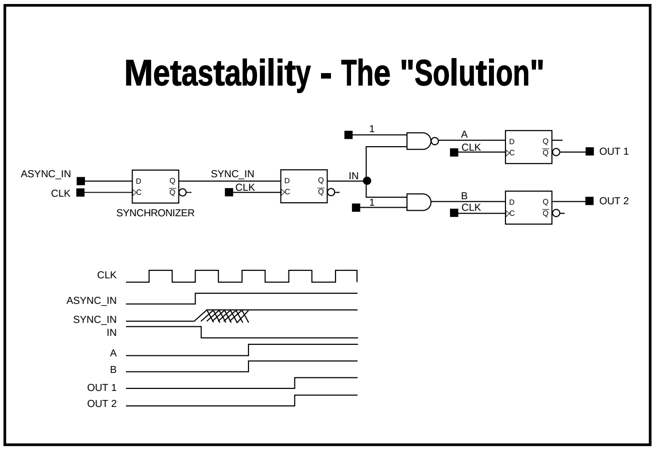

Metastability - The "Solution"

Figure 16

world, the unfortunate fact is that signals coming into an ASIC will depend on a user pushing a button or an interrupt from a processor, or will be generated by a clock which is different from the one used by the ASIC. In these cases, the asynchronous signal must be synchronized to the ASIC clock so that it can be used by the internal circuitry. The designer must be careful how to do this in order to avoid metastability problems as shown in Figure 15. If the ASYNC_IN signal goes high around the same time as the clock, we have an unavoidable ryace condition. The output of the flip-flop can actually go to an undefined voltage level that is somewhere between a logic 0 and logic 1. This is because an internal transistor did not have enough time to fully charge to the correct level.

2.4.7 Allowable uses of asynchronous logic

Now that I've gone through a long argument against asynchronous design, I will tell you the few exceptions that I have found to this rule. These exceptions, however, must be designed with extreme caution and only as a last resort when a synchronous solution cannot be found.

2.4.7.1 Asynchronous reset

There are times when an asynchronous reset is acceptable, or even preferred. If the ASIC vendor's library includes asynchronously resettable flip-flops, the reset input can be tied to a master reset in order to reduce the routing congestion and to reduce the logic required for a synchronous reset. This reset should be used only for resetting the entire ASIC and should not occur during normal functioning of the chip. After reset, you must ensure that the ASIC is in a stable state such that no flip-flops will change until an input changes. You must also ensure that the inputs to the ASIC are stable and will not change for at least one clock cycle after the reset is removed.

2.4.7.2 Asynchronous latches on inputs

Some buses, such as the VME bus, are designed to be asynchronous. In order to interface with these buses, it is necessary to use asynchronous latches to capture addresses or data. Once the data is captured, it must be synchronized to the internal clock. However, it is usually much more efficient to use asynchronous latches to capture the data initially. Unless your ASIC uses a clock which has a frequency much higher than that of the bus, attempting to synchronously latch these signals will cause a large amount of overhead and may actually produce timing problems rather than reduce them.

2.4.7.3 Other asynchronous circuits

Occasionally, circuits are needed to operate before a clock has started running or when a clock has stopped running. Circuits that generate system resets, and watchdog circuits are examples of these. Every attempt should be made to design these circuits synchronously or move them off chip and use discrete chips whose worst case and best case timing is very explicitly defined. If this cannot be done, design these circuits with care and realize that changes to the semiconductor process may make your ASIC unusable.

2.5 Floating Nodes

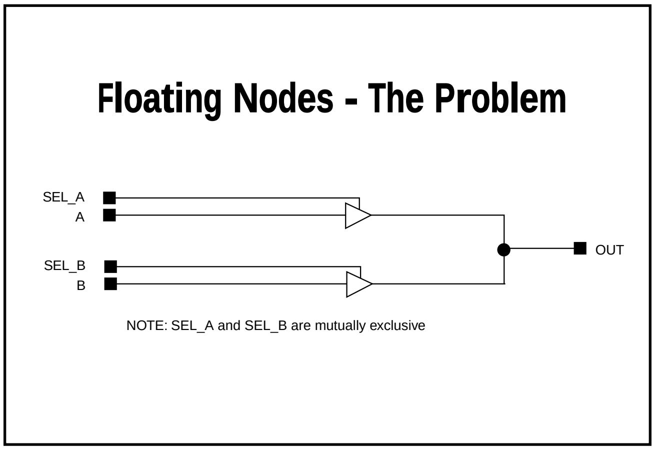

Floating Nodes - The Problem
SEL_A
A
SEL_B
B
OUT
NOTE: SEL_A and SEL_B are mutually exclusive

Figure 17

Floating nodes, or internal nodes of a circuit which are not continually driven, should be avoided. An example of a potential floating node is shown in Figure 17. If signals SEL_A and SEL_B are both not asserted, signal OUT will float to an unknown level. Downstream logic may interpret OUT as a logic 1, a logic 0, or it may produce a metastable state. In addition, any CMOS circuitry that has OUT as an input will use up power since CMOS uses power when the input is in the threshold region.

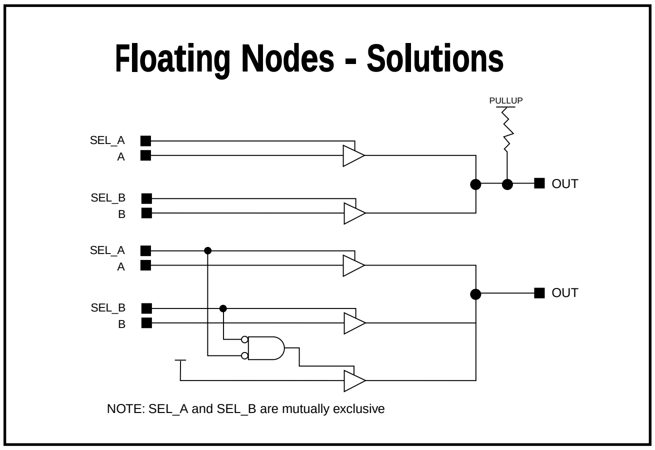

Floating Nodes - Solutions
PULLUP
SEL_A
A
SEL_B
B
OUT
SEL_A
A
SEL_B
B
OUT
NOTE: SEL_A and SEL_B are mutually exclusive

Figure 18

Two solutions to the floating node problem are shown in Figure 18. At the top, signal OUT is pulled up using an internal pull-up resistor. This ensures that when both select signals are not asserted, OUT will be pulled to a good logic level. The other solution, shown at the bottom of the figure, is to make sure that something is driving the output at all times. A third select is generated which drives the output to a good level when neither of the select signals are asserted.

2.6 Bus Contention

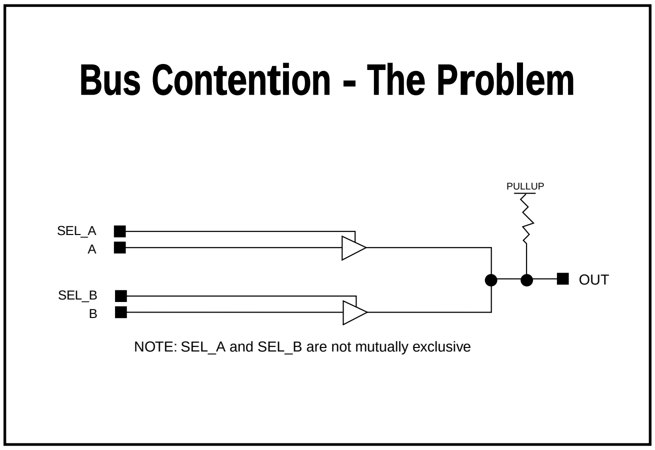

Bus Contention - The Problem
PULLUP
SEL_A
A
SEL_B
B
OUT
NOTE: SEL_A and SEL_B are not mutually exclusive

Figure 19

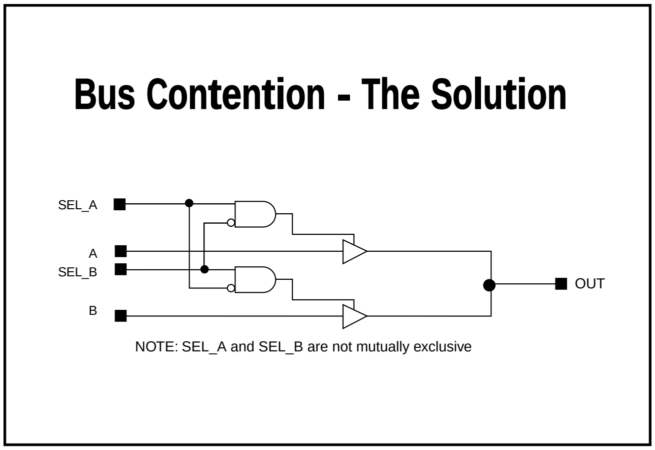

Bus Contention - The Solution
SEL_A
A
SEL_B
B
OUT
NOTE: SEL_A and SEL_B are not mutually exclusive

Figure 20

Bus contention occurs when two outputs drive the same signal at the same time as shown in Figure 19. For obvious reasons, this is bad and reduces the reliability of the ASIC. If bus contention occurs even for short times during a clock cycle, after many clock cycles the possibility of damage to one of the drivers increases. The solution is to ensure that both drivers cannot be asserted simultaneously. This can be accomplished by inserting additional logic as shown in Figure 20. The ideal solution is to avoid tri-state drivers altogether, and use muxes whenever possible.

2.7 Power and Ground Pins

For an ASIC to operate correctly, it must have sufficient power and ground pins evenly distributed around the chip so that all transistors receive good solid voltages even when sinking or sourcing the maximum current expected while in the system. The number of power pins and their distribution depends heavily on the vendor’s process, the size of the ASIC, and your system’s demands on the output drivers. For resolving this issue, you must work very closely with the ASIC vendor who can guide you.

3. Design For Test (DFT)

“Design for test” is a concept which means your ASIC is designed in such a way that testing it is easy. Test logic plays two roles. First, it helps debug an ASIC which has design flaws. Second, it can catch manufacturing problems. Both are particularly important for ASIC design because of the black box nature of ASICs where internal nodes are simply not accessible to you when there is a problem. The following DFT techniques allow for better testing of an ASIC. While not all of these techniques need to be included in your design, those that are needed should be included at design time. DFT techniques should be taken into account during the design process rather than afterwards. Otherwise, circuits can be designed that are later found to be difficult, if not impossible, to test.

One important consideration that can be overlooked, is that test logic is intended to increase the testability and reliability of your ASIC. If test logic becomes too large, it can actually decrease reliability because the test logic can itself have problems which cause the ASIC to malfunction. A rule of thumb is that test circuitry should not make up more than 10% of the logic of the entire ASIC. Similarly, if you spend more than 10% of your time designing and simulating your test logic independently of the functionality of the ASIC, then you have more test circuitry than you need.

3.1 Testing Redundant Logic

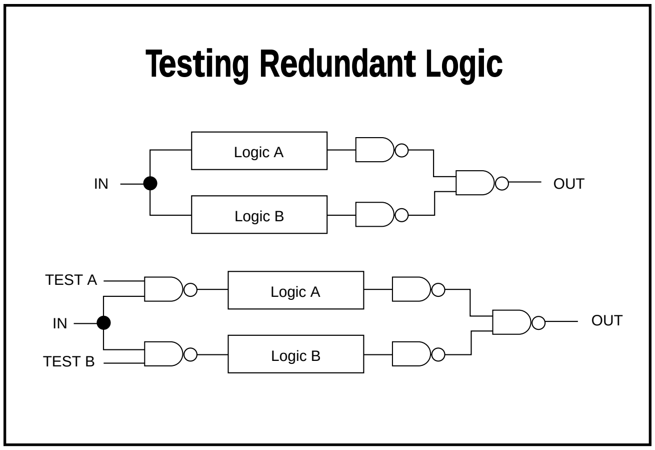


Figure 21

The top of Figure 21 shows a circuit which has duplicated logic in order to increase the reliability of the design. However, since the circuit is not testable, the effect is not as useful as it could be. The circuit on the bottom shows how test lines can be added to allow the entire circuit to be tested.

3.2 Initializing State Machines

It is important that all state machines, and in fact all registers in your design be able to be initialized. This ensures that if a problem arises, the ASIC can be put into a know state from which to begin debugging. Also, for simulation purposes, simulation software needs to start out from a known state before useful results can be obtained.

3.3 Observable Nodes

As many nodes as possible in your ASIC design should be observable. In other words, it should be possible to determine the values of these nodes using the I/O pins of the chip. On the left side of Figure 22, an unobservable state machine is shown. On the right side, the state machine has been made observable by taking each state machine through a mux to an external pin. Test signals can be used to select which output is being observed. If no pins are available, the state bits can be muxed onto an existing pin which, during testing, is used to observe the state machine. This allows for much easier debugging of internal state machines.

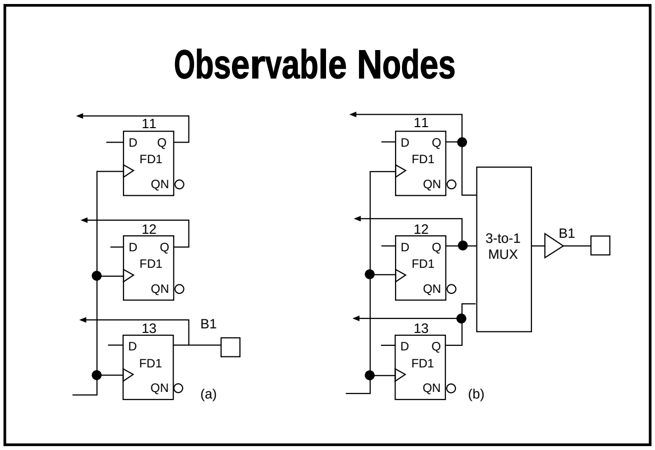


Figure 22

3.4 Scan Techniques

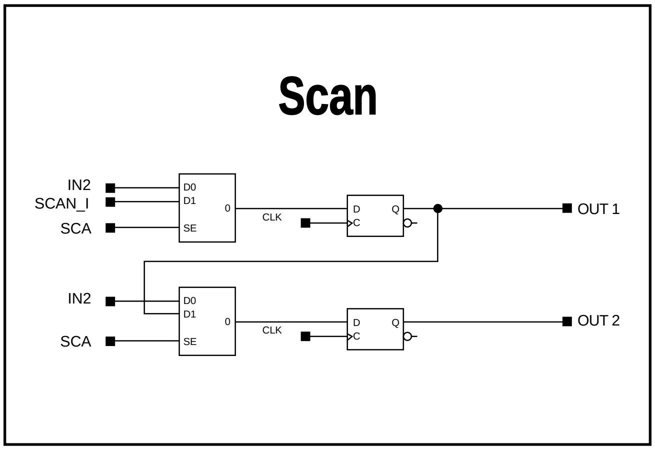
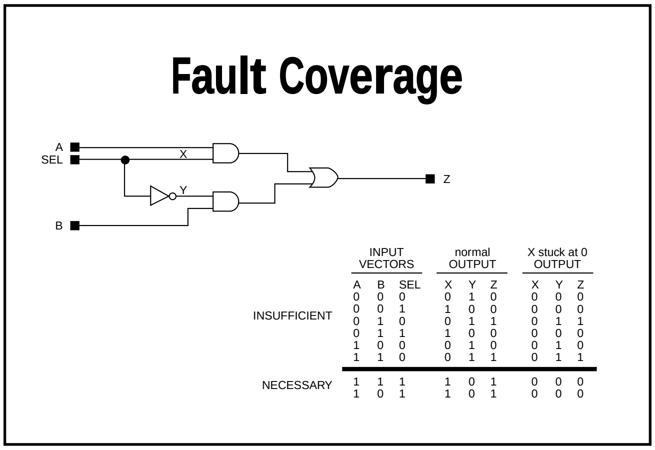


Figure 23

Scan techniques, shown in Figure 23, allow the nodes of the ASIC to be scanned out so that they can be observed externally. There are two main scan techniques - full scan and boundary scan. Full scan is extremely flexible, especially since it can also allow values to be scanned into the ASIC so that you can start it from a known state. This is particularly useful if a problem occurs only after the ASIC has been operating for a long time. A state can be quickly scanned into the ASIC which corresponds to the state which would normally be reached after a long time in operation. The drawback of scan techniques are that they require a lot of software development to support. Also, if states are scanned into the ASIC, you must be careful not to scan in illegal states. It is possible to turn on multiple drivers to a single net internally which would normally not happen, but which would burn out the chip. Similarly, outputs must be disabled while the chip is being scanned since dangerous combinations of outputs may be asserted that can harm your system. There are other considerations, also, such as what to do with the clock and what to do with the rest of the system while the ASIC is being scanned.

Boundary scan is somewhat easier to implement and does not add as much logic to the entire ASIC design. Boundary scan only scans nodes around the boundary of the chip, but not internal nodes. In this way, internal contention problems are avoided, although contention problems with the rest of the system still need to be considered. Boundary scan is also useful for testing the rest of your system, since the outputs can be toggled and the effect on the rest of the system observed.

3.5 Fault Coverage


Fault Coverage

	INPUT VECTORS			normal OUTPUT			X stuck at 0 OUTPUT		
	A	B	SEL	X	Y	Z	X	Y	Z
INSUFFICIENT	0	0	0	0	1	0	0	0	0
	0	0	1	1	0	0	0	0	0
	0	1	0	0	1	1	0	1	1
	0	1	1	1	0	0	0	0	0
	1	0	0	0	1	0	0	1	0
	1	1	0	0	1	1	0	1	1
NECESSARY	1	1	1	1	0	1	0	0	0
	1	0	1	1	0	1	0	0	0

Figure 24

Fault coverage refers to the percentage of faults that can be found when testing an ASIC. A fault is defined as a bad, or malfunctioning node. A node is the output of any gate in the design. Common faults can be “stuck-at”, “open”, or “short”. Other faults have been identified in recent years, but these seem to be the most common. A stuck-at fault involves a node that is accidentally tied to a logic 0 or logic 1 due to a manufacturing defect. An open fault is a node that is not connected to anything. A short fault refers to a node that is shorted to some other node that should not be connected. Your manufacturing tests should uncover as many faults as possible. As shown in Figure 24, however, even extensive tests may not uncover many faults. In the diagram, if node X is stuck at a logic 0, six out of eight combinations of signals A and B will still produce the correct output and never indicate a problem. To uncover this fault, it is necessary to run one of the remaining two test vectors to reveal this fault.

One hundred percent fault coverage is very difficult to obtain. As the number of nodes increases, the number of potential faults rises exponentially. Software is available to help automate this procedure. The user must decide how much fault coverage is tolerable for the particular ASIC.

3.6 Automatic Test Pattern Generation (ATPG)

Automatic test pattern generation, or ATPG, is a method of managing fault coverage. As ASICs become complex, it is often not obvious how to generate tests that improve the fault coverage. ATPG software analyzes the design and generates tests that allow you to increase the fault coverage with minimal effort.

3.7 Built-In Self Test

Another method of testing your ASIC is to put all of the test circuitry on the chip in such a way that the chip tests itself. This is called built-in self test or BIST. In this case, some circuitry inside the chip can be activated by asserting a special input or combination of inputs. This circuitry then runs a series of test on the chip. If the result of the tests does not match the expected result, the chip signals that there is a problem. The details of what type of tests to run and how to signal a good or bad chip is left up to the designer.

3.8 Signature Analysis

Signature analysis involves putting a pseudo-random sequence of ones and zeroes into the chip and noting the ones and zeroes that come out. This output sequence is referred to as the ASIC’s signature. This type of testing can be accomplished with the chip in a normal mode of operation, but is usually performed in scan mode as described above. By repeating the same pseudo-random series of bits, the resulting signature should be the same for each chip. Any chip that produces an incorrect signature is a bad chip. This type of testing is probabilistic and assumes that a pseudo-random sequence of events has a good chance of catching errors, which may not be true. However, it requires very little hardware to implement and can be used as a simple form of BIST.

4. Simulation Issues

Perhaps the most important phase of ASIC design, and the most often overlooked phase, is that of simulation. Simulation beforehand significantly increases your chance of getting working parts back from the vendor. Doing a good job at simulation uncovers errors before they are set in silicon, and can help determine that your chip will function correctly in your system.

There are two main aspects of your design for which simulation is used to determine correctness - functionality and timing. Functionality refers to how the chip functions as a whole, and how it functions in your system. An ASIC which is designed to function as an Ethernet controller may function correctly on its own. In a system that requires an ATM controller, for example, it will not work at all. It is important to look not only at the functionality of the chip as an independent design, but also to test its functionality within the system in which it will be incorporated.

The second aspect of your design which simulation examines is timing. Will your chip meet all of its timing requirements under all possible conditions? Are there any race conditions? Are the setup and hold time requirements met for each flip-flop? Do the I/O signals of the ASIC meet the timing requirements of the system? The following sections discuss ways of using timing to determine both correct functionality and correct timing.

4.1.1 Functional Simulation

Functional simulation involves simulating the functionality of a device without taking the timing of the device into account. This type of simulation is important initially in order to get as many bugs out of the device as possible and to determine that the ASIC will work correctly in your system. During the first phases of simulation, you shouldn’t be very concerned about timing because it will change as the design changes. In fact, the final timing will not be known precisely until the layout is complete. Of course you need to know initially that, in general, the timing of the ASIC process can support the speed and the I/O requirements of your design.

When performing functional simulation, a rough estimate of the amount of simulation to perform is called toggle coverage, which measures the percentage of flip-flops in the ASIC that change state during simulation. Many simulation packages will give you a number for the toggle coverage, and you should have 100 percent coverage before feeling good about the amount of simulation. This coverage can still leave many potential faults uncovered, but it signifies that each state machine has been simulated and no part of the circuit has gone unexamined.

4.1.2 Static Timing Analysis

Static timing analysis is a process that looks at a synchronous design and determines the highest operating frequency of the design which does not violate any setup and hold times. You can also use the static timing analysis software to specify a specific frequency, and the tool will list all paths that violate the timing requirements. These paths can then be adjusted to meet your requirements. Any asynchronous parts of your design (they should be few, if any) must be examined by hand.

Static timing analysis, or some sort of timing analysis must be performed immediately before layout of your ASIC. At this point, the timing numbers will be estimates that take expected trace lengths into account. After layout, timing analysis must be performed again to determine that the real chip, with real trace lengths and delays, still meets you timing requirements.

4.1.3 Timing Simulation

This method of timing analysis is growing less and less popular. It involves including timing information in a functional simulation so that the real behavior of the chip is simulated. The advantage of this kind of simulation, is that timing and functional problems can be examined and corrected. Also, asynchronous designs must use this type of analysis because static timing analysis only works for synchronous designs. This is another reason for designing synchronous ASICs only.

As ASICs become larger, though, this type of compute intensive simulation takes longer and longer to run. Also, simulations can miss particular transitions that result in worst case results. This means that certain long delay paths never get evaluated and an ASIC with timing problems can pass timing simulation. If you do need to perform timing simulation, it is important to do both worst case simulation and best case simulation. The term “best case” can be misleading. It refers to a chip that, due to voltage, temperature, and process variations, is operating faster than the typical ASIC. However, hold time problems become apparent only during the best case conditions.

4.2 Automatic Tester Issues

When an ASIC is fabricated, each production ASIC is sent through an automatic tester to determine that it is working correctly and meets both the functional specifications and timing requirements. Test vectors, which are sequences of inputs to the I/O of the chip, are used by the tester for this purpose. The vendor will guide you through the tester requirements. The following sections describe common tests for your ASIC.

4.2.1 Open/Short Test

This test tests for I/O pins that are not connected to pads on the ASIC, and for I/O pins that are shorted together.

4.2.2 Power

A power test tests the total current used by the ASIC to make sure that it is within the expected value. This ensures that there is not an internal short that is drawing too much power, and that the semiconductor process is within specification.

4.2.3 Functional Testing

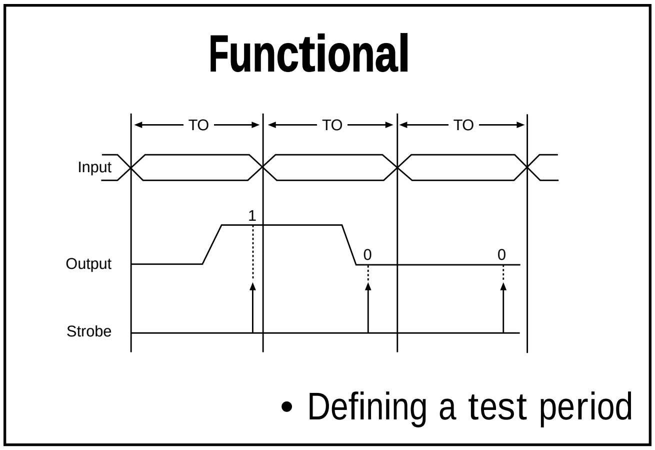


Figure 25

Functional testing determines that the functionality of the ASIC is exactly according to the specification. Test vectors are created based on a subset of the functional simulation. These vectors are then used by the tester as inputs to the ASIC. The outputs are recorded and compared to the expected outputs of the simulation. If there is a mismatch, the ASIC is flagged as bad and discarded.

Testers are designed to require a periodic application of test vectors. A clock period is defined, and new test vectors are applied to the ASIC during each period, as shown in Figure 25.

Figure 26 shows different methods of specifying when to apply test vector signals during the test period. Signals do not need to change at the beginning of the test period. They also do not need to change each period. There are four ways of specifying when

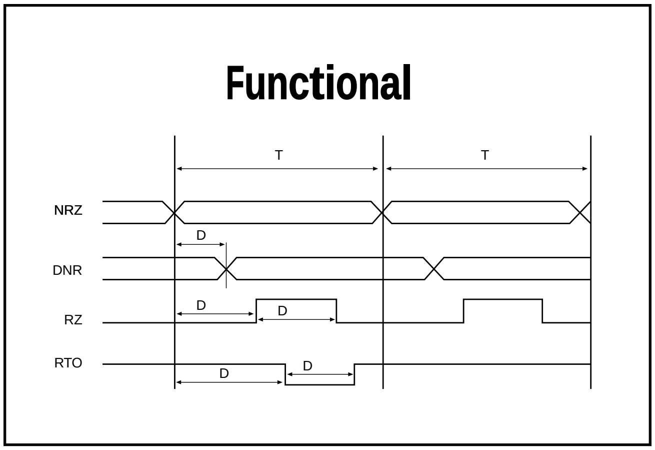


Figure 26

to change the inputs. They are as follows:

NRZ	**Non-Return-to-Zero**	A signal which is defined as NRZ changes only at the beginning of the test period if it changes at all.
DNRZ	**Delayed-Non-Return-to-Zero**	A signal which is defined as DNRZ changes at a fixed delay (D1) after the beginning of the period.
RZ	**Return-to-Zero**	A signal which is defined as RZ begins each test period at 0 value. At a fixed delay (D1) it changes to a 1 value. After another fixed delay (D2) it changes back to a 0 value. The sum of the delays (D1+D2) must be less than or equal to the test period.
RTO	**Return-To-One**	A signal which is defined as RTO is exactly like an RZ signal, except that is begins each test period with a 1 value, then goes to a 0 value, then returns to a 1 value.

Figure 27 shows a test period and a strobe. A strobe is defined at a particular time during the test period. Different testers allow different numbers of strobes. Output signals are sampled at the strobe time in order to compare their values against the expected values. It is important to define strobe points for groups of outputs such that each output that is to be sampled can be guaranteed to be stable at the strobe time.

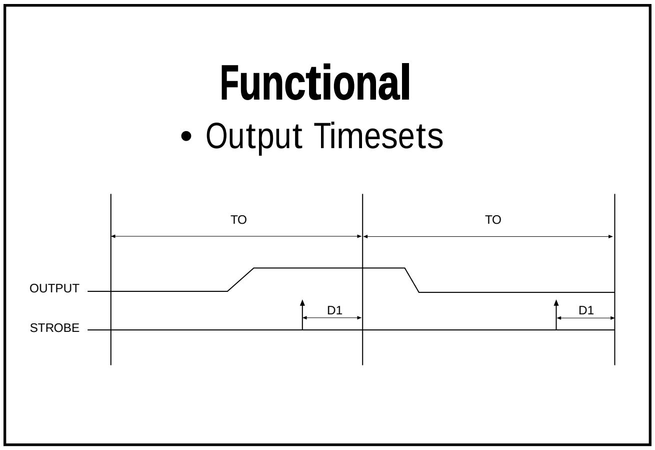


Figure 27

4.3 Three-State Functional Test

The three-state functional test puts the chip into a state where all tri-state pins are high impedance. These pins are monitored by the tester to make certain that they can, in fact, be put into a high impedance state.

4.3.1 Timing Test

The timing test is based on the timing analysis. Essentially, a simple test is used on a very long path output. The output strobe is placed at a time such that if the timing is too slow, the strobe will record an incorrect output.

4.3.2 Electrical Tests

These tests check for correct electrical characteristics to make certain that they are within the specifications of the semiconductor process. They are:

- VIL/VIH test .... Tests the correct threshold voltage of the inputs.
- VOL/VOH test .... Tests the correct voltage levels of the outputs.
- IOS test .... Tests the maximum current during a short of I/O pins.
- IIL/IIH test .... Tests the input leakage current of the inputs.
- IOZ test .... Tests the current drawn by a high-impedance output.

5. The Design Flow

This last section examines the design flow. This is the entire process for designing an ASIC that guarantees that you will not overlook any steps and that you will have the best chance of getting back a working prototype ASIC that functions correctly in your system. The design flow consists of the following steps, with design reviews occurring at the appropriate places in the process:

- Writing a specification
- Choosing a technology
- Design entry - design review
- Designing the ASIC
- Simulating - design review
- Place and Route
- Resimulating - final review
- Testing

5.1 Writing a Specification

The importance of a specification cannot be overstated. This is an absolute must, especially as a guide for choosing the right ASIC technology and for making your needs known to the ASIC vendor. As ASICs grow larger in scale, and more engineers are involved in the design, a specification allows each engineer to understand the entire design and his or her piece of it. It allows the engineer to design the correct interface to the rest of the pieces of the chip. It also saves time and misunderstanding. There is no excuse for not having a specification.

A specification should include the following information:

- An external block diagram showing how the ASIC fits into the system.
- An internal block diagram showing each major functional section.
- A description of the I/O pins including
    - output drive capability
    - input threshold level
- Timing estimates including
    - setup and hold times for input pins
    - propagation times for output pins
    - clock cycle time
- Estimated gate count
- Package type
- Target power consumption
- Target price
- Test procedures including in-system test requirements

It is also very important to understand that this is a living document. Many sections will have best guesses in them, but these will change as the ASIC is being designed.

5.2 Choosing a Technology

Once a specification has been written, it can be used to find the best vendor with an ASIC technology and price structure that best meets your requirements.

5.3 Design Entry - design review

You must decide at this point which design entry method you prefer. For smaller ASICs, schematic entry is often the method of choice, especially if the design engineer is already familiar with the tools. For larger designs, however, a hardware description language such as Verilog or VHDL is used because of its portability, flexibility, and readability. When using a high level language, synthesis software will be required to “synthesize” the design. This means that the software creates low level gates from the high level description.

At the end of this phase it is very important to have a design review. All appropriate personnel should review the decisions to be certain that the specification is correct, and that the correct technology and design entry method have been chosen.

5.4 Designing the ASIC

When designing the ASIC, remember to design synchronously and take into account the design issues that were discussed previously. These include:

- Top-down design
- Use NAND gates predominantly
- Macros
- Synchronous design
- Protect against metastability
- Avoid floating nodes
- Avoid bus contention
- Check the number of power and ground pins

5.5 Simulating - design review

Simulation is an ongoing process while the design is being done. Small sections of the design should be simulated separately before hooking them up to larger sections. Once design and simulation are finished, another design review must take place so that the design can be checked. It is important to get others to look over the simulations and make sure that nothing was missed and that no improper assumption was made.

5.6 Place and Route

The next step is to place and route the ASIC, resulting in a real layout for a real chip. This step is typically done by engineers at the vendor's facility, with input from the design engineer. This is because the vendor's engineers have more knowledge about their semiconductor processes and about the layout tools.

5.7 Resimulating - final review

After layout, the ASIC must be resimulated with the new timing numbers produced by the actual layout. If everything has gone well up to this point, the new simulation results will agree with the predicted results. Otherwise, the design will need to be tweaked and a new layout obtained. This process continues until the final simulation is within specification.

At this point, a final review is necessary just to confirm that nothing has been overlooked.

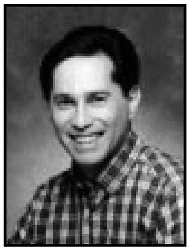
5.8 Testing

Now, prototype ASICs are manufactured by the vendor, tested in their automatic tester, and sent to you. You have the responsibility to place these prototypes in your system and determine that the entire system actually works correctly. If you have followed the procedure up to this point, chances are very good that your system will perform correctly with only minor problems. These problems can often be worked around by modifying the system or changing the system software. These problems need to be tested and documented so that they can be fixed on the next revision of the ASIC.

When the ASICs are put into production, it is necessary to have some sort of burn-in test of your system that continually tests your system over some long amount of time. If an ASIC has been designed correctly, it will only fail because of electrical or mechanical problems that will usually show up with this kind of stress testing.

6. Conclusion

This paper has intended to present guidelines for developing an ASIC based on my experience designing ASICs for a large number of companies and a large number of applications. If all of these guidelines are followed, the chances of creating a working ASIC in a short time at minimum expense is excellent. ■



Bob Zeidman is the president of Zeidman Consulting, a contract R&D firm specializing in digital hardware design and software design. He has been working in the electronics industry for over 13 years and is the author of several articles on ASIC design. He holds an MSEE from Stanford University, and a BSEE and a BA in physics from Cornell University.