Logic Design Pathology and Space Flight Electronics

R. Katz[1], R. Barto[2], and K. Erickson[3]
[1]NASA Goddard Space Flight Center, Greenbelt, MD 20771
[2]Spacecraft Digital Electronics, El Paso, TX 79904
[3]Jet Propulsion Laboratory, Pasadena, CA 91109

Abstract

Logic design errors have been observed in space flight missions and the final stages of ground test. The technologies used by designers and their design/analysis methodologies will be analyzed. This will give insight to the root causes of the failures. These technologies include discrete integrated circuit based systems, systems based on field and mask programmable logic, and the use computer aided engineering (CAE) systems. State-of-the-art (SOTA) design tools and methodologies will be analyzed with respect to high-reliability spacecraft design and potential pitfalls are discussed. Case studies of faults from large expensive programs to "smaller, faster, cheaper" missions will be used to explore the fundamental reasons for logic design problems.

I. Introduction

There are many aspects to building reliable space-flight systems and flying these systems remains a high-risk venture. Considerations include design/analysis, fabrication, parts, environmental modeling, test, and operations. Over the past year (1999) we have seen the failure of domestic launch vehicles (Titan, Delta, Centaur) and foreign ones (Japanese, Brazilian, and Russian). Additionally, a number of domestic scientific satellite missions have failed. The Small Explorer Wide Field Infrared Explorer (WIRE) mission was lost because of a design error [1, 2]. The Mars Climate Observer (MCO) mission failed because of a navigation error during operations [3]. The reason for the assumed loss of Mars Polar Lander (MPL) is at this time not yet determined. All of these missions were lost with none of the scientific goals met [It is noted that some science is being performed with the WIRE spacecraft, using the star tracker as an instrument]. Well over three billion dollars of flight hardware has been lost in the last 12 months alone. The increasing trend of satellite failures, prior to 1999, was discussed in [4]. Questions are publicly being asked if the pressures to cut costs and development times has increased the failure rate and are the increased failure rates acceptable in the era of "faster, better, cheaper" with many more missions being launched?

This paper will analyze and discuss failures that are a result of design errors and their root causes. It is shown in [5] that design errors account for approximately 25% of all space-borne failures, from 1962-1988, obviously a quite significant fraction of all failures. Specifically, the discussion will concentrate on design failures for digital logic circuits and their root causes. A detailed examination will be made of some SOTA design tools and methodologies. This paper aims to be of use to the practicing design engineer to improve the reliability of future designs. As such, the critical nature of this document is intended to demonstrate general principles. Specific examples, using real circuits and tools, are given. There is no generic statement made about a particular design organization, designer, manufacturer, or product. In fact, it is found that these faults are quite generic; in many cases, they appear to be timeless, having appeared continuously over many years, using multiple technologies.

In the beginning of the space age, digital systems were designed using discrete devices with no microcircuits available. Memory systems and flip-flops were quite a bit different from what we use today, with devices such as magnetorestrictive delay lines and magnetic cores used in aerospace systems. Logic, which this paper is concentrating on, was built from discrete components and the function of each element was quite straightforward. As an example, from [6], quad-redundant logic was used to ensure the long-term reliability needed for the one-year mission. AND gates, as shown below in Figure 1, were constructed from resistors and diodes; an inverter used 8 transistors. Flown later in the decade, the Apollo program used a variety of technologies. The mission computer for the Saturn V launch vehicle used discrete circuits [7]. The Apollo Guidance Computer's (AGC) central processor unit (CPU), used in both the command module and the lunar excursion module for guidance and navigation, consisted of three versions [7]. The first version, derived from a missile computer [8], used transistor-core logic for the CPU. It was redesigned later, an early adopter of microcircuits, for the Block I spacecraft, using several thousand devices, where each device was a single three-input NOR function. The Block II spacecraft used advancing technology and was constructed with dual three-input NOR functions. The Block II CPU design had increased complexity with respect to its predecessors, additional instructions were added and

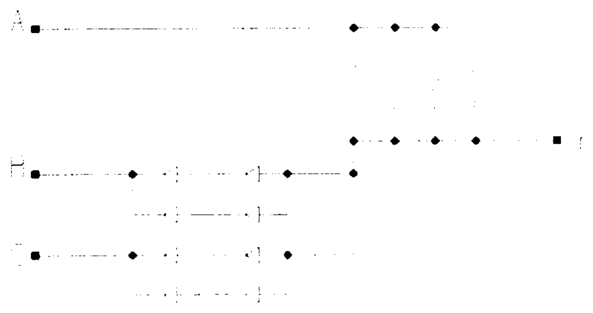


Figure 1. Quad-redundant AND gate from the Orbiting Astronomical Observatory. An inverter used 8 transistors.

some instructions, such as the multiply, were made more efficient with increased logic. Nevertheless, even with the increased complexity of the CPU over the course of the Apollo project, the number of gates to implement the CPU remained in the low thousands and the devices were conceptually very simple.

The trend of increasing complexity of logic circuits has continued both in commercial designs as well as in military and aerospace designs. In the 1970's and 1980's, microcircuit densities increased to the medium scale integration (MSI) and large scale integration (LSI) levels, corresponding to about 10^2 and 10^3 gates per chip, respectively. Today's logic designers frequently use programmable devices such as field programmable gate arrays (FPGAs) with a capacity of 10^4 gates per microcircuit, with multiple microcircuits used per board. Recently, space qualification has begun for FPGAs having on the order of 10^6 gates. Application specific integrated circuits (ASIC) devices and custom very large scale integrated (VLSI) offer larger capacities. Logic systems today have orders of magnitude more complexity, as measured by gate count, than the Apollo CPU designed over 30 years ago.

A fundamental issue is how the complexity is managed to permit reliable design despite the increased size and complexities of the functions performed. This, in fact, is a core theme of this paper. There are several ways in which complexity can be "managed."

One such method is to decompose the system into subsystems. That is straightforward and not addressed in detail in this paper. A reliable system design requires that the interfaces be well understood, the functions of the blocks are well defined and properly implemented, and that failures are handled robustly, according to mission requirements.

The second complexity handling method is provided by the use of simple computer-aided tools. Computers can be programmed to easily handle large lists, such as a netlist. By the use of the conceptually simple schematic capture program, these lists can be generated automatically. This frees the design engineer from a large, tedious, and error-prone task. There is no loss or distortion of information in using the CAE tool, when properly configured. It is noted that some tools and users rely on hidden signals to prevent clutter in the diagram and this is clearly a risk.

A third such method is abstraction, freeing the designer/analyst from having to keep all information together at once for analysis. This method will be divided into two distinct sub-methods. The first such technique is to embed the so-called *intellectual property* (IP) into a component. This, in the 1970's and 1980's, was achieved by providing devices with standard functions. In principle, the designer only needed to understand *what* the component did, not *how* it did it. As a result, the component can be placed on a circuit board and if configured and wired correctly, would add functionality to the system. Examples of these types of components include adders, counters, and microprocessors. More recently, there is the concept of "soft IP", where a description of the logic function is supplied from a third party which is fed into a logic synthesizer, producing a gate level netlist. In principle, this too requires that the designer only understand how to use the component, not understand how it works.

The second sub-method of abstraction is by allowing the designer to work at a "higher level." By the construction of various abstract layers, efficiency is promoted, as the designer is provided with a more abstract, powerful model. As an example, the designer of the OAO avionics was required to construct his own logic gates; that is no longer required. Standard functions or macros are provided giving the designer a rich, virtual set of hardware to work with. For the most part, the underlying electrical properties of the components, for a typical design, are not important and can be ignored. High-performance, optimal designs quite clearly are an exception. A CAE tool known as the logic synthesizer takes in a description of a circuit and produces a detailed design. This powerful tool can produce arithmetic functions in a single line of high level code. For example:

```
Y <= A + B;
Z <= C * D;
```

where the terms `A, B, C, D, Y,` and `Z` are multi-bit vectors, arbitrarily defined elsewhere in the description. Another feature is that abstract state machines can be designed, with the state assignment problem and next-state equations solved by the CAE software. Again, a high-level description is used, utilizing familiar and powerful programming constructs such as `CASE` and `If-Then-Else`. Clearly, these tools have the capability for high gains in efficiency. Circuits that could take hours or days for detailed design can be synthesized in minutes. In principle, these synthesized logic designs are correct-by-construction.

However, despite improved devices, methods, and tools, logic errors are still common in space-flight projects, with "bad circuits" making it into flight hardware, on the ground and on-orbit. Current space-flight electronics are expected to perform better and have higher levels of functionality as compared to designs of a decade or more before. Concurrently, we see decreased budgets and shorter development times. Designers must do more with less, obviously competing constraints. This process is not always successful and frequently not smooth.

This paper shall study the nature of logic design failures, its causes, processes, developments, and consequences. Failures are examined in detail and, layer by layer, the technologies used in the design and analysis process are peeled back, and the root causes of each failure are uncovered. That is, we shall study logic design pathology.

II. Overview of the Paper

The majority of this paper will be to provide an in-depth analysis of failures. These case studies come from industrial, government, and academic institutions. The depth of coverage is not equal for each fault as we intend to cover a wide range of problems and a detailed treatment of each is not practical. Some sections will consist mostly of general, high-level principles that may seem obvious; we include them but do not dwell on them solely to document that these mistakes, violating rules that "everyone knows," still happen regularly in the industry. The majority of the discussion will be used for detailed analysis of faults that are less than obvious. Other analyses, while perhaps simple, will demonstrate the underlying cause of the fault, indicating a process problem. Although the technical problem itself may be trivial, we need to answer the fundamental question of why these faults make it into flight hardware. It is noted that many of these problems are found in numerous systems and quite common.

Appendix A will provide a basic overview of field programmable gate array (FPGA) technology and introduce some of the concepts that factor into the faults that appear in real systems. A few architectures are discussed, providing the reader not familiar with FPGA technology a brief introduction to the technology. An exhaustive treatment of FPGAs is beyond the scope of this paper and additional information can be found in [9], which also contains extensive references. Faults have been divided into three classes. Section III will cover detailed hardware issues. Problems here range from the almost trivial to some subtle issues. CAE software is now virtually indispensable for logic design and its use, and engineer's reliance on it, is increasing. Issues with this technology will be covered in Section IV. This section includes some examples of faults that are quite difficult to spot by examination. With many circuits now represented by textual descriptions of their behavior, an understanding of synthesis and optimization technology is critical. Section V will discuss causes from a "high level" - the discussion here will be short and itself be high level, corresponding to the faults. This section shows that many "common sense" rules are still being violated and the results of this process.

III. Low-Level Hardware

A. Special Pins and Pin Terminations

Many of the logic devices in use today have a variety of special pins. They have a number of purposes: putting a device in different modes for operations such configuration, enabling debugging, programming, special voltages, etc. These pins are either unique to a device family, a manufacturer, or they may be a commercially accepted standard such as the IEEE 1149.1 JTAG specification, for boundary scan test. There may also be differences between devices in a family, or even within a device revision level. For example, some later revision levels of the RT54SX16 will include the optional TRST* pin. Many modern devices have adapted the JTAG specification, both for it's original purpose and others such as user debugging and programming support. Proper termination of these pins is critical to the reliable operation of the device. In many cases improper termination or no termination at all may result in a condition where the device functions fine in test but poses a significant reliability risk in flight. The cases below give common examples of the types of problems that are seen in flight hardware. A large amount of space is devoted to this relatively simple class of faults since its occurrence is quite high and the effects can be severe.

1. Actel MODE Pin

This pin is present in many of the Actel device families: Act 1, 2, 3, XL, DX, and MX. It is not present in the SX or SX-A families, their newest antifuse technologies. This pin, for reliable operation, needs to be held at ground. The pin is driven high for test, debug, and programming operations. Nevertheless, many systems have problems related to the improper termination of this pin.

Several projects have simply left the pin open, unterminated. They noted that the device would sometimes appear to "latch up." That is, the device would be non-functional and consume large amounts of current. The pin was left open because engineers unfamiliar with the device did not know the correct termination. In fact, the part did not latch up, as a latch up state is when a parasitic silicon control rectifier (SCR) turns and latches on, providing a low impedance path between the power supply pin and ground, thus requiring the removal of power to reset the SCR. Experimentally, devices were tested in a heavy ion beam with the MODE pin high, as part of an antifuse experiment. Under these conditions, the part would also appear to latch, consuming in excess of 800 mA of current, the current limit setting of the power supply [10].

Another engineer, after testing his design successfully for months, suddenly experienced a major failure. This coincided with a change of power supply used in the bench-level test setup. Troubleshooting went on for several weeks.

A failure investigation found that the MODE pin was tied to V_{cc} while power was applied. For this Act 3 device the Actel data book states:

```
MODE (Input) The MODE pin controls the
use of diagnostic pins (DCLK, PRA, PRB,
SDI).  When the MODE pin is HIGH, the
```

special functions are active. When the MODE in is LOW, the pins function as I/Os.

Why was the MODE pin deliberately tied high? This flight designer was not very familiar with the device and did not understand the technology. He did know that the MODE pin should be held high for *testing*. Since he was "testing" his board, he terminated the MODE pin to V_{CC} and did not differentiate between testing of the FPGA and his system. As a result, control flip-flops inside of the device, described below, powered up into different states, dependent upon the waveform of the V_{CC} signal as the power came up. After proper termination of the MODE pin to GND, the device again started functioning normally.

One vendor designed a flight subsystem that is being used on multiple missions. In their design, they unfortunately left the MODE pin open and operated the boards. Later, they discovered the improper termination and simply grounded the MODE pin and concluded that the devices were still reliable. Unfortunately, the manufacturer of the device could not guarantee the reliability of the device as its full behavior is not known under these conditions. These devices must be considered potentially overstressed.

Although just a sampling, we see that many projects have problems with either unterminated or improperly terminated MODE pins. The engineers designing the circuits did not understand the technology of the device they were using, just it's functional, logical behavior. The MODE pin plays a critical role, as we will discuss now. Figure 2 is an overview of a typical Actel device. Along with the array, which consists of I/O blocks, logic modules, and routing resources (discussed in greater detail later), there are several registers. These registers are configured such that the serial stream clocked in during test and programming operations will place the part in the desired state. The data is shifted in by properly sequencing SDI (serial data in) and DCLK (data clock), when the MODE pin is driven high. When the MODE pin is low, the registers are asynchronously driven to a safe, operational state. With the extra registers, normally hidden from the users' view, in the proper state, the device may be probed, while on the user's board. However, other modes are reserved for programming and test by the manufacturer, either in the factory or on the programmer. There are also illegal modes, and it has been demonstrated that several hundreds of mA may flow between power and ground, if certain sets of values are loaded into this register. This is consistent with the results from both the heavy ion test and users who have left the MODE pin floating. In order to ensure that the device is reliable after operation with the MODE pin high, the analysis must show that the anomalous current didn't exceed any current density limits and that no local heating could have stressed any devices. To date, this has not been done.

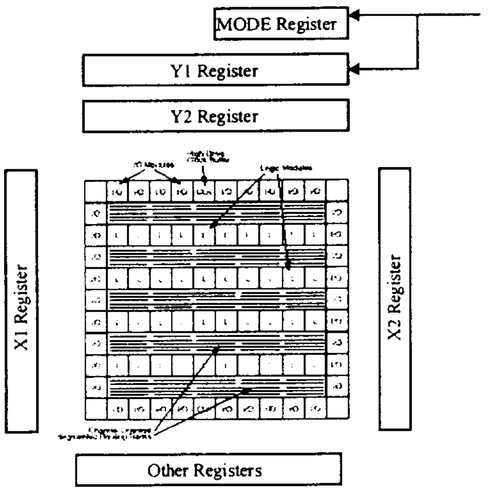


Figure 2. Overview of a typical Actel device showing array and special registers. Holding the MODE pin low keeps the special registers in a safe, operational, state.

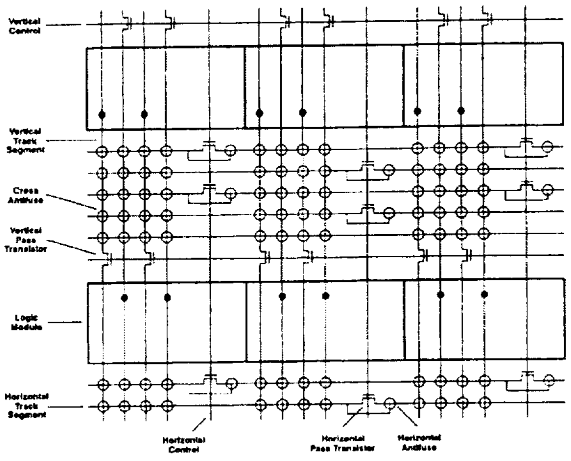


Figure 3. Close up view of a typical Actel device showing logic modules, routing resources, antifuses, and pass transistors used for test and programming operations.

Figure 3 shows a close up view of the implementation of a typical Actel FPGA. The pass transistors are used to make temporary connections and are not used in flight. The gates to these transistors are driven by the shift registers described above. If these pass transistors are enabled, which is possible if the MODE pin is high, then extra connections can be made. This can result in internal bus contention, low impedance paths between V_{CC} and ground, etc. The risk of a failure in flight is higher than during ground test. Clearly, the flip-flops in the control register will be subject to Single Event Upsets (SEUs) and the MODE pin, if it is high, will not be able to clear them. This can lead to functional failure. Once the technology and role of the MODE pin is understood, its proper termination is obvious. The casual user may misapply the device and have it work fine in test. This is an example of where verification by test is inadequate. For this reason, an ohmmeter should be used to verify that there is a low impedance path from the MODE pin to GND on every flight board.

2. *Actel SDI, DCLK Pins*

These pins, like the MODE pin described above, need special handling and are frequently configured improperly. Dealing with the MODE is quite simple: ground it. The SDI and DCLK pins can be more complex, since they can, depending upon the users design, also function as I/O pins. The user is strongly urged to check the data book for each device, as these pins are not used identically between families. Since the purpose of the paper is to help minimize errors, we reproduce Actel recommendations for termination of these pins [11].

For the A1020 series of parts, the SDI and DCLK pins are not driven. As a result, if these two pins are unused in a design (and the software tries to keep them unused), then they will be floating and may oscillate as a result of a variety of causes. It has been verified that the SDI and DCLK pins do float on a sample A1020B, designed with the 2.21 version software. I have also verified for this device that the PRA and PRB pins, when unused, are tied to a logic 0. Note that the RH1020 is based on the A1020B mask set. For ACT 2 A12XX devices all unused pins are automatically programmed as outputs which are driven low, and therefore may be left unterminated on the board. For all other Actel devices: A14XX, A12XX XL, and 32XXX DX, the unused pins are tri-stated but the pad circuitry is unpowered, which should mean that they may be left unterminated. Note that the new RH1280 radiation-hardened device is based on the A1280XL mask set.

For the MODE pin a hard jumper to ground should be placed in parallel with the resistor. Using this scheme, if a problem arises, the PROBE feature of the device would be directly usable by clipping on the ActionProbe (which provides direct access to all internal signals). As an additional reminder, PCB designs should be checked to ensure that the MODE pin is grounded. Two designs recently have been found to have these pins floating.

Since the state of the pins may be dependent on software versions, it is recommended that SDI and DCLK on all device types be terminated to ground through a 10 kohm resistor.

3. *Actel V_{PP} Pin*

The V_{PP} pin present in many of the Actel device families is dedicated for programming, using a high voltage source. One spacecraft subsystem left this pin floating in addition to not being positive that the MODE pin was grounded. The specification requires that the pin be tied to V_{CC} for proper operation. However, no functional problems were observed. Examining the device's implementation in detail, there is a diode with the anode connected to V_{CC} and the cathode connected to V_{PP}. A grounded V_{PP} would be troublesome although it appears that a floating pin should not cause any immediate trouble. Some analysis was done and breakdown tests run on sample devices, as it was a concern that the pin could "charge up" during flight. While it was felt that the circuit would probably be OK, there was no guarantee that the application would be satisfactory. Therefore, late in the program, the spacecraft was disassembled, the V_{PP} properly connected, and the MODE pin's termination verified.

4. *Xilinx MODE Pins*

There are two issues with the MODE pins, used for determining the configuration mode of many device types. In the XC4000 series, the internal pull-up resistors guarantee a logic high level. However, after the device is configured, the resistors are disconnected and the pins are floating. This can cause a failure if the device is reconfigured. Either external pull-up resistors or a modification to the bit stream can prevent this problem [12]. Additionally, problems have been reported related to the noise sensitivity of the MODE pins when internal resistors are used for termination; the resistors can have values up to 100 kΩ [13]. The use of external resistors of lower value can be used to prevent problems.

5. *Unused Inputs*

In an Actel A1020 application, the design used very few of the I/O pins for each of the devices in the system. That left a very large number of "no connects" in the subsystem. The designer, an experienced engineer, terminated the unused leads as one would for a typical CMOS device: they were grounded. This, unfortunately, led to a potential stress situation and added to the magnitude of the start-up current transient.

To properly understand this configuration, we need to understand the design software and the hardware implementation. By default, the back end software will program all unused device pins as outputs, driving a logic '0'. This would imply, from a *logical* point of view, that a hard ground of the unused I/O pin would be redundant, although the intent was to ground an unused input. However, from a *physical* point of view, we know that in

Act 1 devices inputs and outputs may actually source current during the power-on period [14]. Looking into the implementation of the A1020, we know that the device is configured by antifuses which, when programmed, make connections in the routing channels. Since the antifuses are programmed with a voltage high enough to destroy ordinary transistors, high-voltage isolation FETs are used to protect the device during programming operations. In operation, the isolation FETs are biased ON from a charge pump, whose output is higher than V_{CC}, to ensure that a good logic '1' is transmitted throughout the device. During startup, there is some time needed for the charge pump to start and to charge the large number of FETS. The basic architecture is shown in Figure 4, where we see the isolation FETs protecting input and output transistors for each module. In the case of the I/O cells, the input to the cell is not guaranteed to be properly driven, resulting in a transient situation where the output driver is not under control. This can cause an I/O module, configured either for input or for output, to source current during the start-up transient. Note that this behavior is not a fundamental nature of all antifuse FPGA devices. Act 3 devices behave differently and tend to put the outputs into a tri-state condition [14]. Some future models of SX and SX-A devices may have configurable cells, such that the pin will be driven either high or low by a resistor, during the transient.

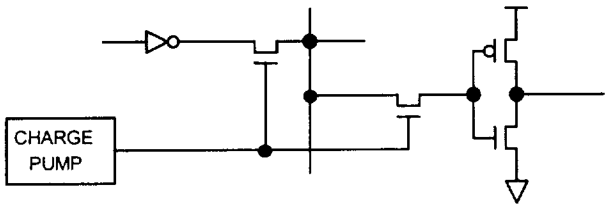


Figure 4. Charge pump and isolation FETs for an A1020 FPGA. The isolation FETs protect the transistors from damage during the programming operation, when high voltage is applied to the selected antifuse. The charge pump puts out voltage to bias the nFETs on during normal operation. During the startup transient, the device is not guaranteed to follow its truth table. For the case of an I/O module, pins configured as inputs or outputs can source current during the transient.

6. IEEE JTAG 1149.1 Pins

The IEEE Standard Test Access Port and Boundary-Scan Architecture [15] is a technology that is being put into many modern microcircuits. Commonly referred to as "JTAG," it was originally intended to support board test. In practice, it has been used for other functions such as programming, debugging, and built-in self test. The interface uses a small number of pins, TCK, TRST*, TMS, TDI, and TDO. The TRST* pin, used to hold the test circuitry in a reset state, is optional and a microcircuit can be compliant with that pin not implemented. For many devices, particularly microcircuits designed for the commercial market, the TRST* pin is not implemented, saving an I/O pin for other uses. Several problems have been seen with the use of devices implementing this standard, as discussed below.

Figure 5 shows a circuit from a flight instrument. The microcircuit being used did not implement the TRST* pin so a clock was provided to the TCK input. While the JTAG specification requires that, upon the application of power, the device's test controller enters the `TEST-LOGIC-RESET` state (which keeps the test logic in reset and the device functional), a single event upset can cause the device to enter a test mode. The JTAG specification ensures that microcircuit will move back to the `TEST-LOGIC-RESET` state after no more than 5 clocks on the TCK pin, if TMS is held high. Logically, this appears to be a good circuit.

It is known [16] that in some circuits with JTAG, very high currents can be consumed along with a loss of functionality when irradiated with heavy ions. Test data has shown that even with a relatively high frequency dedicated clock for TCK, system control may be lost and the device draws excessively high currents. For the test setup used for this experiment, the programmed current limit of 800 mA was reached. [16] gives an overview of the JTAG 1149.1 specification with an emphasis on its application to operation in the heavy ion environment.

There is another logic error in this flight circuit. A block diagram of the implementation of an I/O module in a JTAG-compliant microcircuit is shown in Figure 6. Here we see that the output driver is under control of the JTAG data path, essentially a shift register. If a false command enters the JTAG system, then an output driver may be turned on, although the user has intended this to be an input

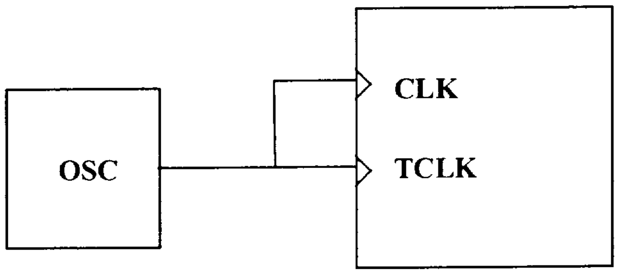


Figure 5. User circuit with a "JTAG"-compliant device without the optional TRST pin. An external clock is used as input to TCK to ensure that the TAP controller enters the TEST-LOGIC-RESET state after five TCK cycles. Logically correct, this circuit has several flaws.*

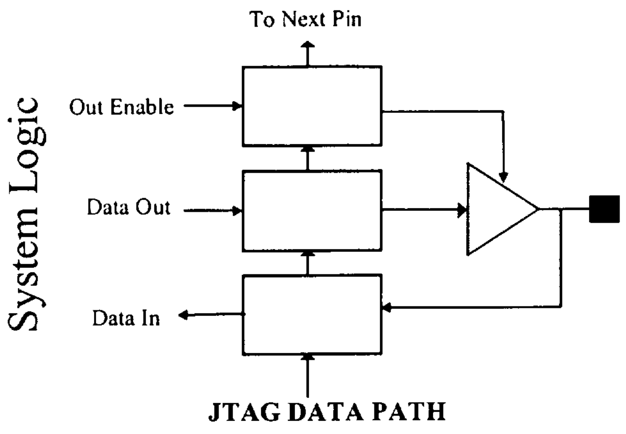


Figure 6. Block diagram of the control of a JTAG-compliant I/O module. The JTAG data path is a shift register under the control of the TAP controller. If the TAP controller is taken out of the TEST-LOGIC-RESET state by a heavy ion, garbage values may be applied to the control of the I/O module. This may result in changing a system-level input pin into an output pin.

pin. This can lead to system failure since the device, when the JTAG gets upset, can turn the CLK input into an output driving low. For most devices, this will clamp the board-level clock net close to ground, prohibiting the CLK signal from clocking the TCK pin. The system has locked up.

One potential problem with the use of JTAG-compliant microcircuits is a floating TRST* pin. When TRST* is driven low, the test controller is held in the `TEST-LOGIC-RESET`. When high, the controller is free to leave this "home" state, either by command or by an SEU. A careful reading of [15] shows that when left unconnected, the TRST* pin will behave as if driven by a logic '1'. This is another example of logic that is not testable. Since there is an internal power-on reset detector in JTAG-compliant microcircuits forcing the TAP controller to the `TEST-LOGIC-RESET` state, operation with the TRST* pin high or low is indistinguishable in ground test. For high-reliability applications, it is critical that the TRST* be held low and verified. This is critical even for applications that do not expect a significant number of SEUs. For example, one user (jet engine controller) with a JTAG-compliant microcircuit found that the TAP controller would leave the `TEST-LOGIC-RESET` state in ground test operations. The termination of the TRST* pin, when present, should be verified with an ohmmeter for every device in a flight system.

B. Clock Skew

The circuit shown in Figure 7 is one that is commonly seen in board designs as well as FPGA designs, where the clock buffer shown is a typical buffer, not a dedicated low-skew driver for FPGAs. The circuit has the general principle that two flip-flops have little delay between stages. This same structure can occur in counters and other circuits. Another variant is shown in Figure 8, where a large clock load is driven by several buffers by constructing a clock tree. Both of these circuits are flawed. They *may* operate fine on the bench for a particular unit, but may fail when the device is re-routed, another device programmed, another lot of parts is procured, or the operating conditions change. This circuit structure is typical of those designed by experienced engineers new to the FPGA technology.

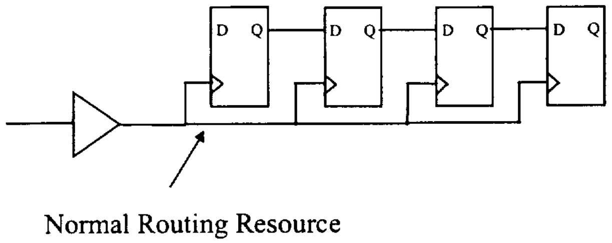


Figure 7. Shift register clocked by a local routing resource. The use of a high-skew clock can result in circuit failure from hold time violations.

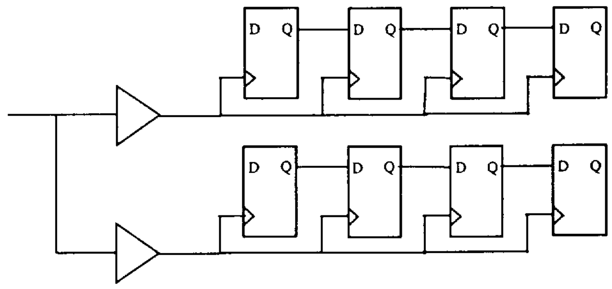


Figure 8. Long shift register with clock load driven by a clock tree. This structure in an FPGA adds clock skew from different delays to the buffers, buffer delays, and buffer loads. Buffer loads include not only the clock inputs, but the resistance of the connection elements and the amount of capacitance from a variable length routing segment.

The following sections will analyze this class of circuits and discuss the failures that are seen in flight electronics and their causes. Clock skew may also be caused by CAE software, discussed in a later section.

1. Definitions and Discussion of Terms

Clock skew is the difference in arrival time of the active clock edge between two sequentially adjacent registers. *Sequentially adjacent* registers are two registers that only have logic and/or interconnect between them. *Arrival time* is the time that the active edge "triggers" the register. Propagation delay time is referenced at an arbitrary level (voltage). Additionally, the arrival time is also a function of the slew rate of the signal that is driving the clock at the destination of the signal, as well as the receiver's logic threshold. A complete analysis will assume that the slew rate and logic thresholds will fit into a window, and as such have a maximum and minimum associated with them. In addition, these parameters, like others, can vary over the course of a mission because of the effects of operating lifetime, stress, and radiation exposure. Short term effects such as temperature and voltage must, of course be accounted for.

For proper analysis, the simple circuit shown in Figure 7 is modeled as shown in Figure 9. T_{ROUTE} and T_{SKEW} are a function of resistance, including that of antifuses (or pass transistors) and capacitance as well as loads. For antifuse-based devices, the resistance of antifuses will vary from antifuse to antifuse. The capacitance is a function of the placement of the logic modules and their routing. For reliable circuit operation, under all conditions, we must have:

$$T_{CQ} + T_{ROUTE} > T_{SKEW} + T_H \qquad (1)$$

Note that delays are a function of the trigger point of an input, which is a function of the logic threshold, a parameter that varies under different operating conditions.

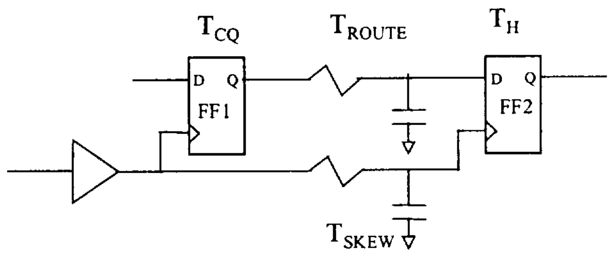


Figure 9. Circuit used for analysis of clock skew timing using local routing. A late arriving clock at FF2 can cause a hold time violation and circuit failure. A model for the circuit topology shown in Figure 1 would include the skew introduced by the extra buffer and the routing of the signal to each of the buffers.

2. Circuit Analysis with Clock Skew

This section will discuss the analysis of clock skew. As the discussion proceeds, problems from a number of projects will be analyzed. While the discussion is focused on a particular class of FPGAs, the concepts are general.

Clock skew must be calculated for each pair of sequentially-adjacent flip-flops for accurate results. It is tempting, when using a static timing analyzer, to group sets of flip-flops together. Then the analysis may be performed in one step, using the "set" as opposed to calculating the hold time for each pair of sequentially-adjacent pair of flip-flops. Clearly, this technique will be overly conservative since worst-case skew can only occur between sequentially adjacent pairs. If min and max delays are taken from this set, the min and max values may not be from sequentially adjacent flip-flops. As a result, the calculated skew may not be physically present and affect any real signals. Performing analysis on each pair of flip-flops is very labor intensive but is necessary for accurate results.

In calculating clock skew for hold time analysis, the worst-case situation occurs when the source flip-flop is clocked with the earliest possible clock and the sink flip-flop is clocked with the latest possible clock. Therefore, the clock path analysis to the source flip-flop should be calculated as a minimum. The clock path analysis to the sink flip-flop should be calculated as a maximum. Simply setting the analysis conditions to "worst-case," usually low voltage and high temperature will produce an incorrect answer. There are several reasons for this. First, by setting the conditions to maximums makes the implicit assumption that the maximum values for all delay parameters is the worst-case. Looking closer at equation (1), the worst-case occurs when T_{CQ} and T_{ROUTE} are minimums and T_{SKEW} is a maximum. Obviously, for minimums, setting the conditions to "max" will not give a proper analysis. T_{CQ} and T_H are functions of the silicon process and conditions. A closer look at T_{ROUTE} and T_{SKEW}, however, along with the model in Figure 9, shows that the minimum and maximum antifuse resistance needs to be included in the analysis. The resistance of an antifuse is a variable that falls within a range of values. Since each antifuse is individually programmed, we can not assume that the resistance of different antifuses will track.

Figures 10 [17] and 11 [18] show several antifuse resistance distributions from two manufacturers. Figure 10 is data using an ONO antifuse. To date, these have been the most popular antifuses used in space-flight systems and are used in FPGAs and PROMs. The data in Figure 11 is for an amorphous silicon antifuse, one type of metal-to-metal antifuse, which typically have close to an order of magnitude lower resistance. The use of this class of antifuse is growing with space applications in PALs, PROMs, and programmable substrates, as well as FPGAs. The resistance distributions are wide enough that they can affect critical timing for hold time. In fact, this variation has led to failures in flight hardware.

Of course, doing full min-max calculations for the entire delay paths is unrealistic. Two flip-flops, buffers, etc. on the same chip will not see large differences in supply voltages, say 4.5 VDC and 5.5 VDC. Similarly, with respect to temperature, we won't have one element at -55 °C and another at +125 °C. Therefore, a reasonable assumption is that two devices near each other operating at the same frequency with similar loads will have similar

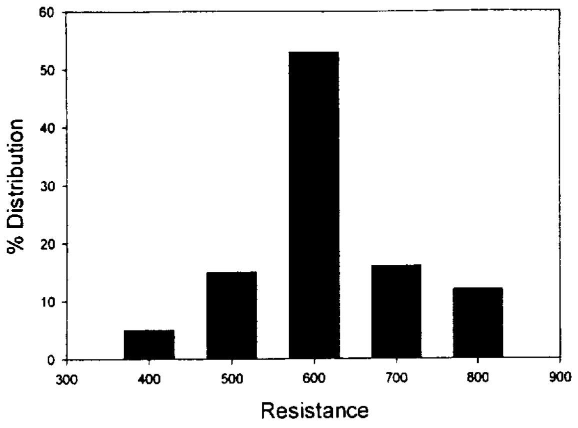


Figure 10. ONO antifuse resistance distribution for a 5 mA programming current. The worst-case analysis must take into account the range of antifuse resistance when doing min-max calculations. Antifuse resistance varies from fuse to fuse and does not "track" over a lot of parts.

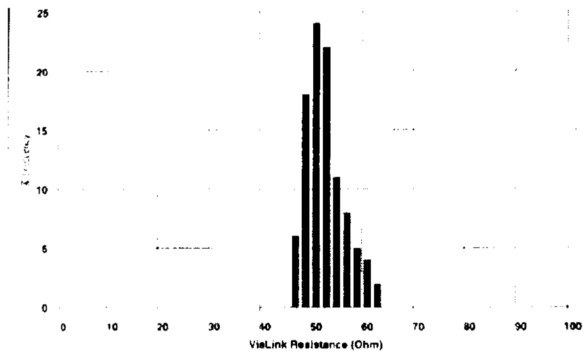


Figure 11. Amorphous silicon antifuse resistance distribution for a 0.65 μm technology.

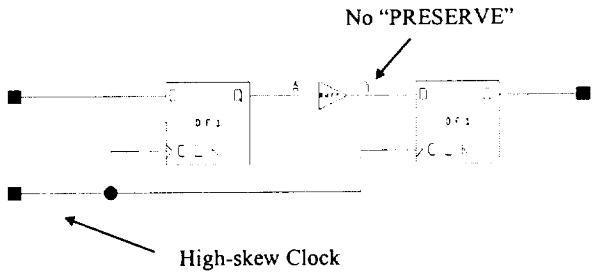


Figure 12. Attempt to eliminate clock skew problem. Careful timing analysis is needed to ensure that the added buffer will have sufficient delay to guarantee adequate hold time. Circuits using this configuration have failed because of excessive clock skew.

voltages and temperatures. Another factor in min-max calculations is process variations. It may be safe to assume that two devices on the same chip will not represent fully different corners of all acceptable process parameters. Of course, we most definitely can not assume that two devices side by side will have identical results. Parallel buffers will have differences and that must be accounted for. However, there is no database available that will give transistor to transistor variations on the same chip so the more conservative numbers must be used. Additionally, parallel buffers may have different loads, even if they appear to have the same number, based on the number of flip-flops on a net, as the Actel FPGAs use a variable length routing segment architecture, which can contribute to imbalance.

Looking again at some existing analyses, we see the need to properly select the proper operating conditions. For example, the "worst-case" conditions of low voltage and high temperature are often not the worst-case conditions for hold time. Frequently, cold temperature, high voltage, and best case processing is. Another factor is the so-called "speed grade" of the device. For Actel devices, the speed binning is done by ensuring that the binning path does not exceed a maximum specification value. As a result, particularly for military and aerospace programs, devices with a faster binning path can be marked as a slower device, as the marking is fully specified in the SMD or the SCD. The situation is similar for Xilinx devices, another supplier of space-grade FPGAs.

Figure 12 shows one technique that was used to ensure that adequate hold time was met on a high-skew clock net. Unfortunately, it is not guaranteed to work by construction. Flight hardware has failed with sequentially adjacent flip-flops separated by a logic module. This circuit will be revisited later with respect to the use of design software.

The designer/analyst must carefully analyze paths that have sequentially adjacent flip-flops, clocked on the same edge, which reside on a high-skew clock net or a clock tree. This is a slow and tedious process, and the analysis must ensure that the worst-case paths are adequately modeled. A full discussion of design techniques will not be presented here. However, in general, use of low-skew clock networks is recommended in conjunction with proper timing analysis. The hold times can be adjusted, if necessary by a variety of techniques such as component placement or appropriate setting of the clock balancing parameter [note that this feature, for A1020 and A1020A devices, is not available in current releases of software]. Opposite edge clocking can be used reliably, trading off one concern, clock skew, for another, clock duty cycle control and perhaps logic resources. Two-phase, non-overlapping clocks can also be used. The clock generator can be constructed using high-skew clock resources with a two-stage twisted ring counter clocked on opposite edges and a few gates. Then, simple latches can be used for implementing structures such as shift registers, which is very efficient in architectures such as Act 1.

3. Chip-to-Chip Applications

The above discussion analyzed clock skew within an integrated circuit. However, a circuit that looks acceptable at the board level schematics, as shown in Figure 13, can have hold time problems. The key parameters here are the $CLK \rightarrow Q_{min}$ of the source device, t_H of the sink device, and $CLK_{pin} \rightarrow CLK_{modulemax}$ of the sink device. As in the on-chip case, the worst-case is when the data moves fastest from the source chip and the clock is "late" to the sink chip. Note that the transition time of the clock driver and the logic thresholds of each device must also be included in the analysis as was done in designs using the old CD40xxB series devices, where parallel clocking was often a problem.

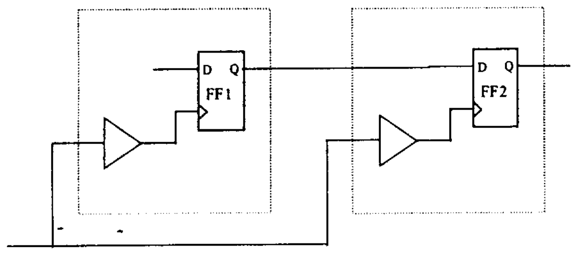


Figure 13. Clock skew between chips. Depending on the device selected, the design, and the place and route, this circuit may not guarantee proper hold times. Some architectures use dedicated I/O flip-flops and clocks to ensure hold times are met while others achieve this by circuit design. The architecture may have a programmable delay element (Xilinx) in the data path that is enabled for sequential inputs for reliability and disabled for high-speed combinational inputs. Lastly, with some devices, parallel clocking may not be reliable and the opposite clock edge, or a different clock, should be used.

The analysis and circuit designs differ, depending on the device selected. The place and route as well as the configuration of each device, if FPGAs are used, can also be a factor. For example, using Act 3 technology, there are flip-flops in the I/O modules and a dedicated clock that guarantee 0 ns hold time and reliable operation. Of course, skew between the dedicated I/O clock and the low-skew clock network used for the logic must be carefully analyzed, as skew effects are not guaranteed to be a problem. The I/O module latches in Act 2, for example, guarantee a t_{SU} of 0 ns, not a t_H of 0 ns. Some architectures, such as the CQR4000XL series, incorporate programmable delay elements in the I/O module along with I/O block flip-flops. When a sequential input is used, this delay guarantees adequate hold time; for a combinational input, the delay is bypassed, providing maximum speed. Lastly, with some devices, parallel clocking may not be reliable and the opposite clock edge, or a different clock, should be used.

C. Start-up

Circuit performance during the start-up transient can be critical for many military and aerospace applications. Information on device's behavior is sometimes not obvious and manufacturer's information on this condition is either not prominent or located in application notes. There have been failures in this area and the following sections will review circuits and provide some insight into device behavior.

1. FPGA Outputs

The circuit in Figure 14, at the schematic level, looks like a good circuit with no obvious design flaws. Modern programmable devices are CMOS and many engineers expect them to be well behaved during power-up, as CMOS circuits will function at quite a low supply voltage. For the FPGA used in this particular circuit, the A1020, the implementation of this device result in some unexpected behavior.

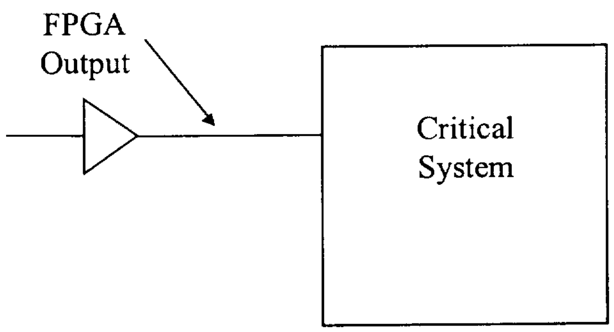


Figure 14. FPGA output start-up transient interfacing to a critical system. The behavior of an FPGA during the start-up transient can be either controlled or uncontrolled, depending on the architecture of the device and specific circuit design. A transient output can cause critical events to happen, such as switching relays, firing pyrotechnic devices, etc.

Referring to Figure 4, we see the basic connection scheme for this class of FPGA. The isolation FETs, for protection during programming are biased ON during normal operation, with the bias supplied by an internal charge pump, that takes a finite time to start and to charge the parasitic capacitance. However, as shown in Figure 15, the device outputs can be uncontrolled just after V_{CC} is applied. It is critical to note that the top two traces in the Figure 15 will not have the same waveform each time power is applied. For example, it was found that repeated applications of power for this particular circuit would result in the top two traces remaining close to zero volts. After letting the device set, unbiased, for a number of hours, the waveforms in the figure would repeat. This example shows that the design must be properly analyzed and can not be qualified by test. For critical signals controlling relays,

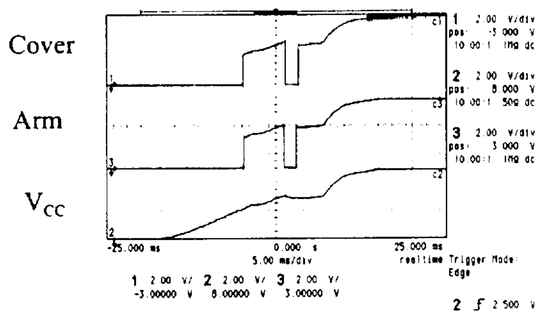


Figure 15. Startup characteristics of an A1020 FPGA with a power supply rise time of 20 msec (10% to 90%) after being powered off for 24 hours. The horizontal scale is 4 msec per division. Cover and Arm are outputs at 2V/Div and are driving high in this test run. V_{CC} is scaled at 5V/Div.

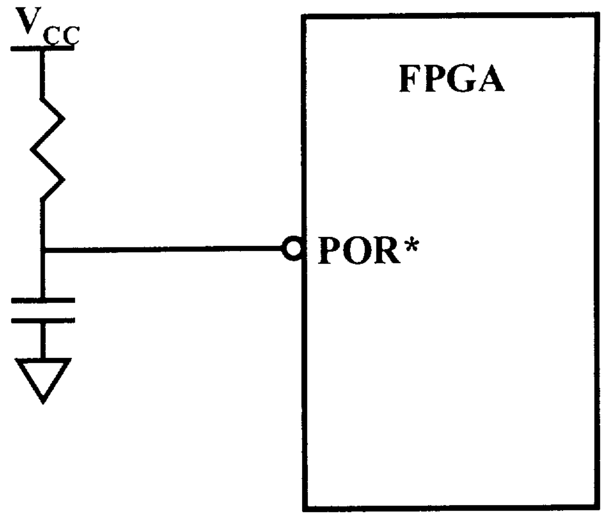


Figure 16. A typical Power-on reset circuit. Considerations for good circuit design include a time constant long enough to allow crystal oscillators and FPGAs to start, protection against capacitor discharge through the ESD protection diodes, fast response to voltage drops, transition time limitations of the inputs, and knowledge that some FPGAs may source current during the start-up transient. Additionally, many FPGA types will not respond to their inputs while they are being configured or

pyrotechnic devices, etc., a seemingly innocent circuit can cause mission failure.

2. Power-on Reset (POR) Detector

Figure 16 shows a commonly used topology for a power-on reset circuit. There are a number of considerations for this class of circuit.

First, one must consider a proper value of the time constant. As described below, in section 3, crystal oscillators often take a relatively long time to start, perhaps 50 ms or more, which is dependent on the crystal oscillator used. Secondly, the rise time of the power supply must be known and, loosely speaking, the time constant of the POR circuit must be long with respect to the power supply's rise time. This rise time typically ranges from 0.1 ms to 100 ms. Slow rise times are often desirable to limit the in-rush currents into capacitors and to meet EMC requirements. Lastly, for many FPGA devices (and this example was taken from an A1020 design) there is a finite start-time. This time is needed for the charge pump to start and to charge the large internal capacitances associated with the gates of the isolation FETs. It is necessary that the power-on reset circuit's time constant must be greater than the start time for the slowest device in the system.

Secondly, it is often good practice to place a diode across the resistor in Figure 16, with the anode at the junction where it meets the capacitor and the cathode at the V_{CC} supply. This serves two purposes. First, it helps the circuit to respond to short dropouts. Secondly, it provides a discharge path from the bulk capacitance so that the stored charge does not flow through the FPGAs input protection diodes (most models) during power down.

A series resistor is often required. Most importantly, this protects the inputs of the FPGA during capacitor discharge when power is removed or if a short (perhaps temporarily) grounds the V_{CC} bus. Some have recommended this as a way to prevent the capacitor for charging as a result of the current *sourced* by the input during the startup transient. The I/O module is not controlled during power up (see section A5) and can temporarily act as an output driving high [14]. A buffer between the analog node and the FPGA is a better solution.

Taking a closer look at the characteristics of an input circuit, it is noted that the high gain transistors often will appear to oscillate when the transition times that they see are slow. For example, many FPGAs have input slew rate requirements of 500 ns measured from 10% to 90%. Measurements have shown that many flight parts will "oscillate" under those conditions. Nevertheless, any POR circuit with a reasonable rise time, which is often more than 100 ms, will be inappropriate for most FPGA inputs. Although many FPGAs have a small amount of input hysteresis, it is insufficient. A buffer with a hysteresis input is recommended.

3. Synchronous Reset

Figure 17 shows a synchronous reset topology. One advantage of this structure is that noise or glitches will, for the most part, be filtered out since the signal is only sampled on the rising edge of the clock. A closer look at the characteristics of these components is necessary to ensure proper operation. Of course, as we have seen, the

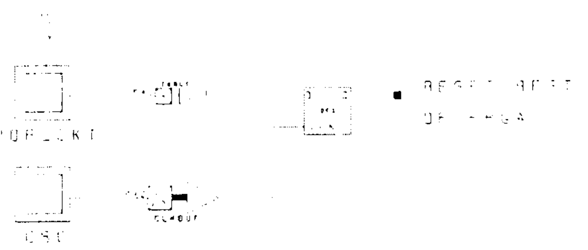


Figure 17. Synchronous reset topology. This circuit structure minimizes sensitivity to noise and glitches by avoiding the use of asynchronous inputs on the flip-flop. It will not start assert the reset until both the FPGA and the oscillator have "started."

FPGA may not follow its truth table on the application of power and proper safing must be handled at the system level. This section will focus on the characteristics of the oscillator since this circuit will obviously not assert reset internally until the oscillator starts.

As discussed previously, crystal oscillators do not start instantaneously. For examples, start times are a function of the crystal, frequency, design, and operating conditions. For the flight system analyzed, we characterized a set of flight oscillators for another parameter often not seen - the rise time of the power supply.

Figure 18 shows how the power supply rise time can affect the start time and waveforms of a Class S oscillator. This data was taken at 10°C, the estimated temperature of failure of the flight system. In this flight system the rise time of the power supply was deliberately slowed by the design engineer. The oscillator did not have a specification for start time nor was the power supply rise time shown as a critical parameter. The engineer was unaware of the start time characteristics of this device and the effects on the POR circuit were not properly analyzed.

Start time data was taken on spare oscillators and is summarized in Figure 19. The frequency of this device is low, 200 kHz, and it has a large start time, particularly for long but reasonable power supply rise times. It was shown that the start time was a linear, as a function of rise time.

Synchronous reset can be reliably used when the system level design takes into account the oscillator start time.

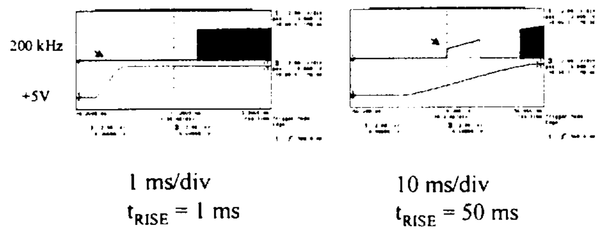


Figure 18. Flight oscillator start time samples. The effect of power supply rise time on the start time of these Class S oscillators is easily seen.

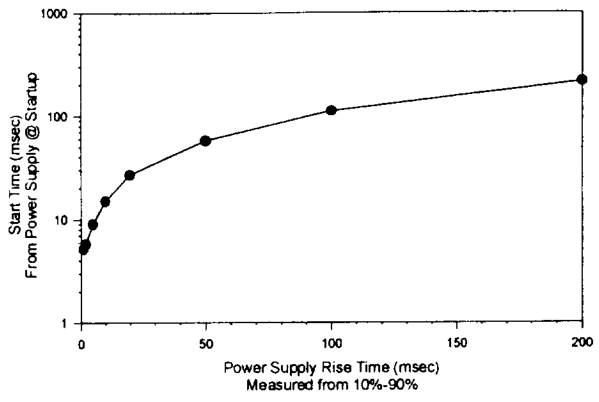


Figure 19. Summary of start time performance of a Class S flight 200 kHz oscillator as a function of power supply rise time. There exists, for this oscillator, a linear function between power supply rise time and start time.

D. Metastable States

1. Introduction

Normally a flip-flop is in one of two states; either storing a logical '1' or a '0'. These states are stable as flip-flop elements employ positive feedback. In properly designed systems, all flip-flop parameters are met and the device operates normally. The essential parameters are setup time (t_{SU}), hold time (t_W), and pulse width (t_W), with the latter applicable to clocks, presets, clears, jam loads, etc. If these parameters are not satisfied, the flip-flop may go "metastable." This would happen, for example, when an asynchronous input is fed into a flip-flop without meeting t_{SU} or t_H, or when a runt pulse is input into the clock or an asynchronous preset or clears. Despite being a well-understood phenomena, many designers are still not aware of metastability and their systems are subject to intermittent failures. These circuits are difficult to "qualify by test" since the failure rate may be low with respect to test time or an intermittent failure may be falsely attributed to another cause.

Device behavior in the metastable state may manifest itself as a device's output being a non-logic level, an output switching and then returning to it's original state, or an increased CLK $\rightarrow$ Q delay. Theoretically, the amount of time a device stays in the metastable state may be infinite. In practical circuits, there is sufficient noise to eventually move the device out of the metastable state into one of the two legal ones. However, this time period may be large with respect to the available timing slack in the circuit resulting in a system failure. Factors that affect a flip-flop's metastable "performance" include the circuit design and the fabrication process. By allowing sufficient settling time, the Mean Time Between Failure (MTBF) for a well-designed system with asynchronous inputs can be made extremely low and reduced to acceptable levels. This is possible since resolution time is not linear with increased circuit time and the MTBF is an exponential function of the available slack time. This can be seen in the following equation:

$$MTBF = \frac{e^{K2 \bullet t}}{K1 \bullet F_{Clock} \bullet F_{Data}}$$

where t is the slack time available for settling, K1 and K2 are constants that are characteristic of the flip-flop, and F_{Clock} and F_{Data} are the frequencies of the synchronizing clock and asynchronous data. By this equation, it is clear that an increase of 't' has an exponential effect on the MTBF. The constants accounting for the key characteristics of a flip-flop's metastable behavior are the size of the window (usually sub-nanosecond) and the time to exit the metastable state (which is a function of the gain-bandwith product of the device).

2. Example - MTBF Calculation

Sample results were calculated for Chip Express CX2001 technology, based on their flip-flop parameters and example in the CX Technology Design Manual. The CX2001 series uses a channeled module architecture (gate array) with each module consisting of three 2:1 multiplexors and an AND gate. This sample calculation uses a 50 MHz clock, a 10 MHz average incoming data rate, and the available extra settling time is the independent parameter. The results are shown in Figure 20.

A number of circuit topologies can be used to properly synchronize an asynchronous input. Optimally, the problem can be solved by proper synchronous system design that eliminates conditions causing metastable states. Unfortunately, that is not always possible.

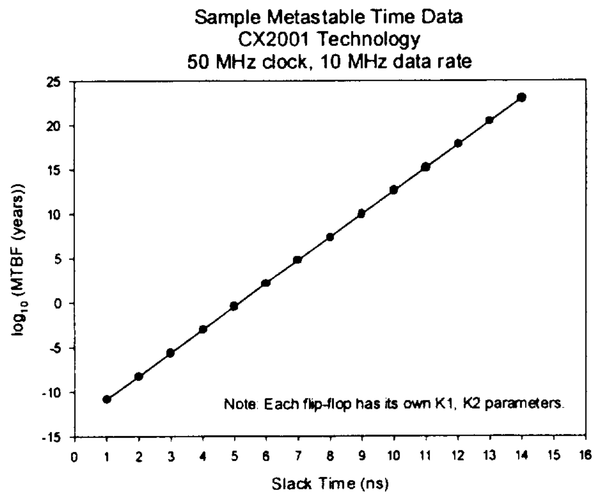


Figure 20. MTBF as a function of available slack time. System reliability is an exponential function of the settling time made available for metastable state resolution.

The data concerning metastable state performance available to the design engineer varies. Some manufacturers provide data for each flip-flop in their library. Certain vendors provide data on only a particular family while some do not support the design engineer at all. When analyzing circuit performance, careful attention should be paid to the test conditions, as temperature and voltage, for example, can significantly affect the metastable state parameters. Wide margins are recommended. Fortunately, for systems of moderate speed, modern hardwired flip-flops have excellent metastable performance, with short resolution times.

3. Example - Inadequate Synchronizer Circuit

Figure 21 below shows a common synchronizer implementation. In this situation, an asynchronous event triggers FF1 whose output is synchronized to the rest of the system, generally as an input into a finite state machine (FSM). This circuit may fail, however, if FF2's output first rises and then falls before settling into the cleared state. That transition, however, may be enough to satisfy the required pulse width of the asynchronous clear input of FF1, clearing the event's notification.

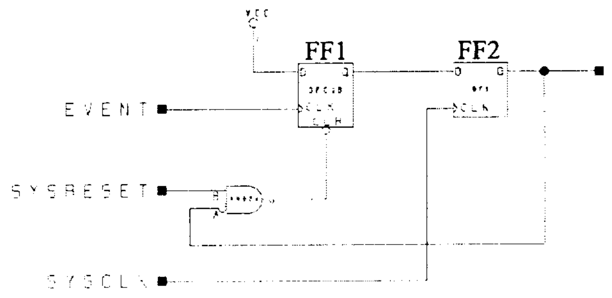


Figure 21. Inadequate synchronizer circuit design. EVENT is asynchronous to SYSCLK resulting in possible metastable behavior. The output of the synchronizing flip-flop (FF2) may clear the latching flip-flop (FF1) before FF2 is stable, resulting in a loss of the incoming event.

E. Asynchronous Circuits

1. Introduction

Most modern design methodologies preach strict synchronous circuit design techniques. Religiously, they insist on all flip-flops being clocked by the same low-skew clock and triggered by the same edge. Additionally, use of asynchronous inputs to flip-flops, such as the clear, should be limited to a single, chip-wide, reset signal.

Realistically, these goals are frequently not achievable or practical. For example, there are asynchronous signals and there are synchronizer circuits to deal with them. Often these circuits require that an asynchronous clear be used and that the clock be triggered by a local signal. For performance and/or power reasons, an asynchronous ripple counter may provide practical high-speed event counting with minimal power dissipation. Another example is the use of asynchronously sampled gray code counters by a data collection system. There are many examples when asynchronous circuits "make sense" and are perfectly reliable. However, it found, as shown in the examples below, that there are many unreliable asynchronous circuits still being designed.

2. Asynchronous Clear

The circuit shown in Figure 22 improperly uses asynchronous logic. The intent of this circuit was to generate a small pulse after the completion of two events. Each event was given a flip-flop to set. After the output pulse is generated, triggered by a third asynchronous input, at point A in Figure 22, the two flip-flops would be cleared and await the next pair of events.

First, it is clearly seen that the output pulse width consists of three gate delays. While careful analysis *may* show that that is acceptable in some technologies, it is a structure to be avoided. In general, pulse widths should be made with clocks, not gates. Note that the delays are highly dependent on the "place and route" process and the analysis must be repeated for each iteration of this algorithm. Analysis did show, however, that the small output pulse would not be guaranteed to clear *both* of the flip-flops reliably. It appears that the output of the NAND gate will reach each flip-flop clear at the same time. However, as shown in earlier sections, additional antifuses and different parasitic capacitances from variable length routing circuits can make the delays significantly unequal. In this particular case, it was not possible to prove that both flip-flops would be reliably cleared and the circuit was redesigned.

3. Asynchronous Decoding - Terminal Count

The circuit shown in Figure 23 uses the TCNT (terminal count) of a vendor-generated counter macro as clock. An examination of the macro's logic, shown in Figure 24, shows that its implementation is similar to that of the 54LS161, where there is no gating and no guarantee of a "glitchless" signal. While the 54LS161 gated the TCNT with the ENT input, the circuit in this macro is always live and is a simple combinational decode of the flip-flop's outputs. An examination of count sequences shows the problem.

$$\begin{array}{l} \longrightarrow \texttt{01111111111111} \\ \ \texttt{10000000000000} \\ \ \quad\quad \dots \\ \longrightarrow \texttt{11111111011111} \\ \ \texttt{11111111100000} \end{array}$$

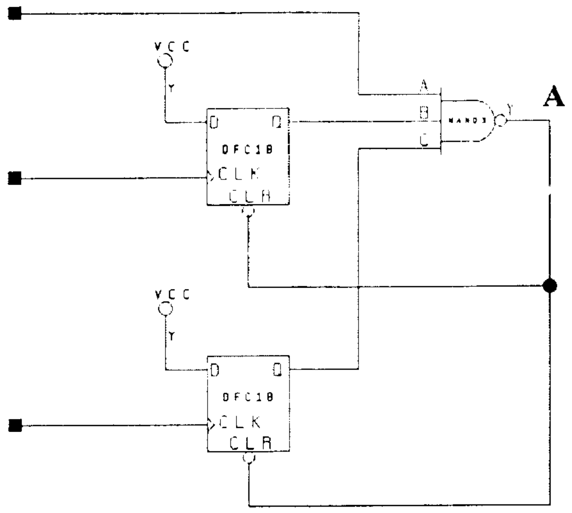


Figure 22. Improper use of asynchronous clears. The output of this circuit has a pulse width determined by propagation delays. Additionally, the pulse may have insignificant width to guarantee that both flip-flops are reliably cleared.

The outputs of the flip-flops of the counter can not all reach the decoding logic at the exact same time (idealized circuit). Certain sequences of states, such as those shown by the arrows, exist where the decoder can momentarily see "all 1's" and output a runt pulse.

There are certain logic structures that permit reliable, "glitchless" decoding. An example of this is the Johnson twisted ring counter. Note that implementations of this structure, since it does not utilize all 2^n possible states, must incorporate lockup state detection and correction for critical applications.

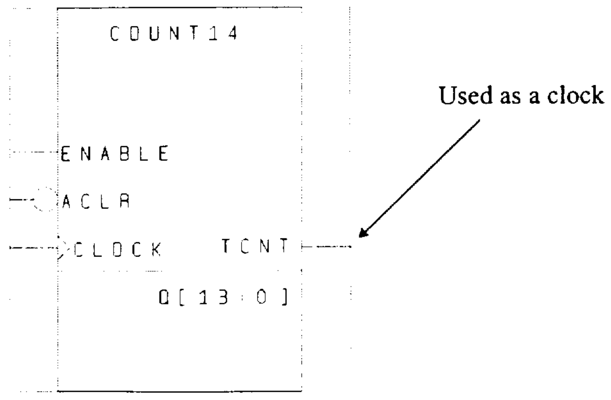


Figure 23. High level view of a manufacturer supplied macro. The use of the terminal count as a clock may result in circuit malfunction since it may "glitch."

4. Asynchronous Decoding - Binary Counter

The circuit shown in Figure 24 will fail for similar reasons as those discussed in Section 3, above. The decoding of a binary counter without qualification can result in glitches being generated. In the general case, when the counter rolls over, that is, all bits transition, any count can be momentarily decoded. The decoder's output should be used as an enable for a flip-flop. Then, the enable will effectively operate a multiplexor and select between feedback from the flip-flop (hold) or input new data. FPGA macros or hard-wired flip-flops frequently offer an enable function exactly for this reason.

Examples similar to that shown in Figure 24 have been seen using both synchronous and ripple counters and neither class of circuits could be shown to have adequate reliability.

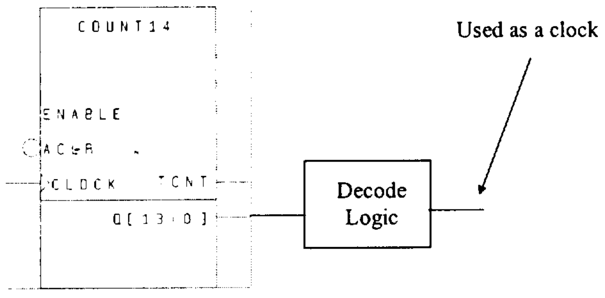


Figure 24. Asynchronous decoding of a synchronous counter.

F. Interfacing Logic Blocks

Frequently there is a requirement to interface different logic blocks, powered by independent sources. Examples include the isolation of redundant circuits, power saving modes, etc. While traditionally done with bipolar circuits as shown in Figure 25, CMOS devices are often used in the same configuration. For most powered off CMOS devices, their I/O pins present a low-impedance path from the I/O pins to the V_{DD} rail, through either protection or parasitic diodes. Clearly, this is not a desirable situation. Additionally, the two supplies would have to power up and down simultaneously to prevent one block from loading the other's supply.

Certain CMOS discrete devices are acceptable in this configuration. For example, the CD4049UB and the CD4050B have special input protection networks so that their inputs remain high-impedance with the power removed. Care must still be taken with the outputs and the use of busses and pull-up resistors. UTMC, for example, has higher speed devices intended for this application. Some of the military-specific foundries offer special I/O cells for functions such as power isolation and cold sparing.

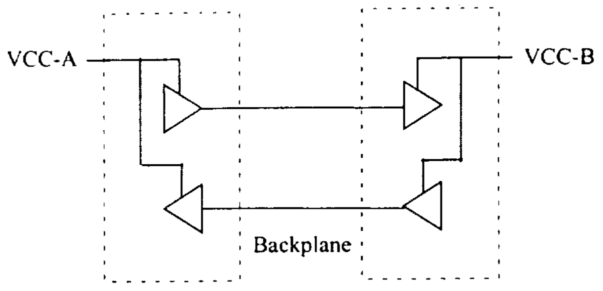


Figure 25. Interfacing logic blocks powered by individual power supplies. Most CMOS devices present a low-impedance on the I/O lines when powered off. Some new devices incorporate special circuitry for this situation.

FPGA vendors, however, typically do not. One version of the RT54SX series (under development) will support power isolation and cold sparing.

G. Interfacing Voltage Margin

Proper TTL to CMOS interfacing was a problem in the 1980's when many CMOS devices had a logic high input threshold (V_{IH}) of 70% of V_{DD} or typically 3.5 VDC. This problem still occurs. Examples include TTL-compatible oscillators and "TTL-compatible" CMOS. Note that in some CMOS devices, the definition of "TTL-compatible" is a "bit stretched," with values of 2.5 VDC seen for V_{IH}(max). Additionally, some CMOS devices have a few input pins that will have input levels which need to be driven by logic signals that effectively switch from rail to rail. This, for example, is often the case for clock signals.

The trend with new logic technologies is to move to smaller feature sizes. For a number of reasons, including reliability, this has resulted in a lower supply voltage and reduced output voltage swing. For example, for 0.35 μm feature-sized technology, supply voltages of 3.3 VDC are typically used; for 0.25 μm technology, a 2.5 VDC supply is used; and with the 0.18 μm technology that will be coming soon, a supply voltage of 1.8 VDC will be used. As such, many of the new devices have limited output voltage swings, incapable of driving inputs on existing devices with adequate noise margin. In some cases, there can be a worst-case negative noise margin. While it was relatively straightforward to design mixed technology systems with 5V CMOS circuits that have true TTL-compatible inputs, in today's circuits with low voltage CMOS devices each interface must be individually checked.

H. TMR Structure

To increase the SEU hardness of a circuit, designers can employ triple modular redundancy (TMR) to effectively harden the flip-flops [10]. An incorrect application of this technique is shown in Figure 26. In this circuit, the designer had a register that would hold a state value. While the circuit had three redundant flip-flops and a correct voter, the reliability is poor. The operation of this circuit would load a value into the three flip-flops. However, there was no provision made for "scrubbing" upsets for the TMR triplet, either automatically in hardware with a free running clock, as described in [10], or in software, by reloading the values at a sufficiently high rate. Therefore, this circuit, although having a higher MTBF due to SEUs as compared with a single flip-flop, will eventually fail.

I. Races

The circuit shown below, in Figure 27, is an example of a race. Although signals B and C are both generated synchronously from the 2 MHz clock, they re-converge on one flip-flop. At that point, t_{SU} and t_H can not be guaranteed and reliable operation can not be assured. Based on the delays in the different paths, the device may operate functionally in the laboratory and test environment with the amount of margin unknown. This circuit may fail in flight as relative delays may change as a function of life effects and degradation due to radiation.

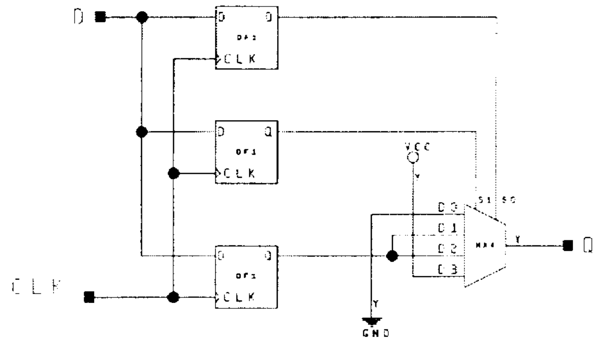


Figure 26. TMR configuration without scrubbing. This register is loaded with no provision made for scrubbing SEUs either in hardware with a free running clock and feedback or by software, by reloading the register at sufficiently fast intervals.

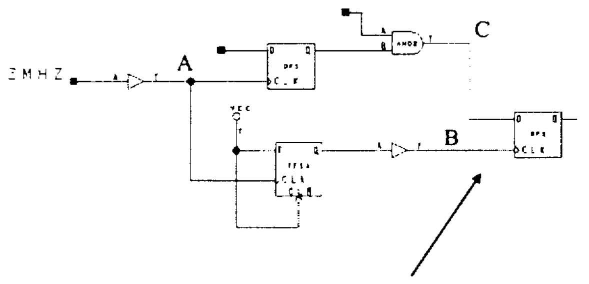


Figure 27. Example of a logic race. Signals B and C both originate synchronously from signal A, a 2 MHz clock. The two signals, however, have a race and t_{SU} and t_H can not be guaranteed to be met.

IV. Low-Level Software

A. Clock Skew Revisited

Section III-B discussed problems with clock skew from a low-level hardware perspective. We revisit this topic, showing more clock skew problems, with software as one of the contributing factors.

1. Elimination of Buffers

Referring back to Figure 12, we see the designer's intent to "solve" a clock skew problem by inserting a buffer between two sequentially adjacent flip-flops. Of course, this is not guaranteed to work. Additionally, by not understanding the design software, the implemented circuit did not match what was desired. An optimization phase in the software, added with an update to the back end design package, determined that the elimination of the buffer would leave the circuit logically unchanged and take fewer resources. Clearly, the circuit would be electrically changed and the designer was unaware of this. To force this particular set of design tools to keep the buffer in the circuit, an attribute named PRESERVE must be attached to the net driven by the buffer.

2. CAE Software Generating a Clock Tree

The following VHDL code implements a 32-bit shift register and is fine. Depending on the particular synthesizer and it's settings, it may produce a circuit with an excessive amount of clock skew. A schematic of the circuit generated by the synthesizer is shown in Figure 28, showing a structure not intended by the designer.

```
Library IEEE;
  Use IEEE.Std_Logic_1164.All;

Entity Skew Is
  Port ( Clk    : In  Std_Logic;
         D      : In  Std_Logic;
         Q      : Out Std_Logic    );
End Skew;

Library IEEE;
  Use IEEE.Std_Logic_1164.All;

Architecture Skew of Skew Is
Signal ShiftReg :
            Std_Logic_Vector (31 DownTo 0);
Begin
  P: Process ( Clk )
     Begin
       If Rising_Edge (Clk) Then
         Q <= ShiftReg(0);
         ShiftReg (30 DownTo 0)
           <= ShiftReg (31 DownTo 1);
         ShiftReg (31) <= D;
         End If;
     End Process P;
End Skew;
```

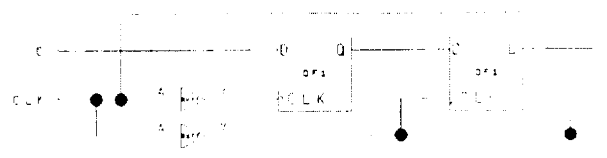


Figure 28. Logic synthesizer can generate circuits with excessive clock skew. Depending on the synthesizer and its settings, the CAE software can generate logical structures that are unacceptable. This circuit fragment is part of a shift register generated from standard VHDL code.

B. VHDL "Interfaces"

The interface to a block of logic defined by VHDL code is defined in the `Port` statement in a VHDL `entity`. This only defines the linkages between a block of code and other parts of the circuit. The behavior of the logic is defined in a structure called the `Architecture`. The section of code below, simplified from the version that appeared in a flight design, appears to be logically correct. The circuit description is written in an elegant fashion, using a Boolean value to represent the logical output of this block. This is similar to how one would program in a typical high level programming language.

```
Library IEEE;
  Use IEEE.Std_Logic_1164.All;
Entity Bool Is
  Port ( X    : In  Std_Logic;
         Y    : In  Std_Logic;
         Z    : Out Boolean     );
End Bool;

Library IEEE;
  Use IEEE.Std_Logic_1164.All;
Architecture Bool_Test of Bool Is
Begin
  P: Process ( X, Y )
     Begin
       If ( X = Y )
         Then Z <= True;
         Else Z <= False;
         End If;
     End Process P;
End Bool_Test;
```

This synthesized circuit, however, may not be logically correct. For the flight circuit, the Boolean signal was mapped to different logical values by different versions of the same VHDL logic synthesizer. A careful look at the VHDL specification and packages shows that there is no requirement to map either `True` or `False` to any particular value. By upgrading a synthesizer during a flight project, from the same manufacturer, an error was introduced since the mapping of `True` and `False` had changed. This was not documented nor was it required to be; the synthesizer still legally met the requirements of VHDL.

C. High-level Design Flow

In this section, we examine several cases where the designers relied excessively on high-level software tools. While doing this, operating at a higher level of abstraction, it was felt that there was no need to understand the lower level architecture and the details of the electronics. The CAE software would properly take care of the details.

1. Case Study 1 - Memory Controller.

The flow diagram shown in Figure 29 was the path taken in the design of a memory controller for a command and data handling (C&DH) system. Since it was a critical system on the spacecraft, there was a reliability requirement that the design be immune to SEUs at a level of 37 MeV-cm^2/mg. The designer had no experience and little knowledge of the Act 2 target technology and relied on the software tools to take care of the detailed implementation. The circuit was logically correct and was felt to be reliable.

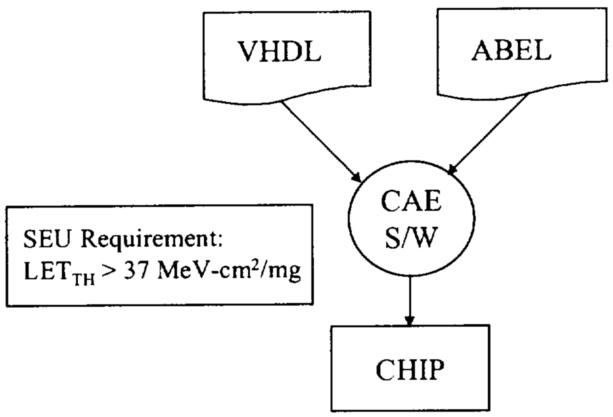
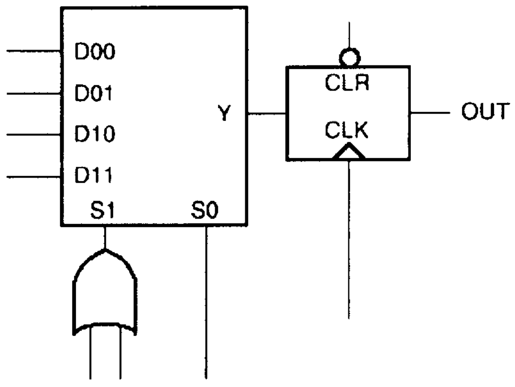


Figure 29. High-level design flow. An FPGA was designed solely using high-level design tools without knowledge of the underlying architecture nor the radiation characteristics of different structures. The software mapped the logical design onto hardware structures that could not meet the SEU requirement.

The CAE tools, which are commercial in nature, tend to map structures "efficiently." For the Act 2 architecture, there are two basic methods for flip-flop construction. The first uses a hard-wired cell, referred to as an S-Module, which is fast and compact as shown in Figure 30A. The second method uses feedback through the routing network and combinational logic structures, with a simple example in Figure 30B. This method consumes more resources and has less system-level performance.



Up to 7-input function plus D-type flip-flop with clear

Figure 30A. Hardwired flip-flop in a commercial FPGA architecture. The CAE tools chose to map flip-flops onto a hard-wired flip-flip in the Act 2 architecture for compactness and speed. This flip-flop is SEU soft.

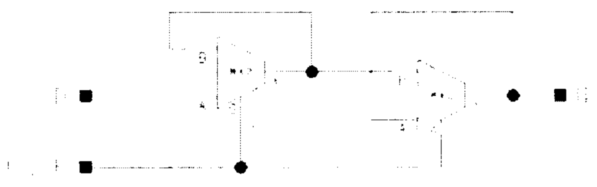


Figure 30B. Flip-flop made from combinational logic resources. A standard macro in the Act 2 architecture, this flip-flop construction is often selected for it's radiation-tolerant characteristics. The feedback for each latch goes through antifuses and the routing network.

The radiation performance, with respect to Single Event Upsets, however, differs substantially with respect to flip-flop construction. The hard-wired, S-module based devices are considered quite soft with a low LET_{TH} and a high cross-section. The routed flip-flops perform better and are considered SEU tolerant. Data for the commercial/military A1280A and the "hardened" RH1280 clearly show the difference, as shown in Figure 31. Neither flip-flop construction would meet the system's original upset requirements. For this example, either a specification change, part change, or a TMR structure would be required. With the design and analysis divorced from the physical implementation and architecture, the design was unaware of the performance of the circuit in the actual flight environment. After analysis, C-module flip-flops were used in a redesign and considered adequate for the mission.

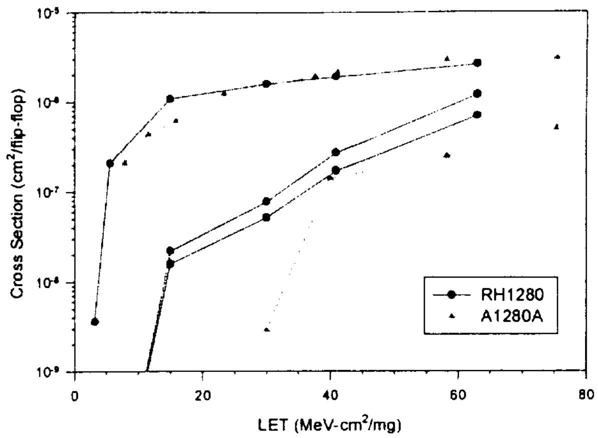


Figure 31. SEU data summary for two members of the Act 2 family. The SEU performance of two types of flip-flops are shown, with the higher cross-section and lower LET for the "hard-wired" storage elements. Flip-flops made from combinational logic using the routing network perform substantially better. Neither of the two flip-flop structures could meet the project SEU requirements.

2. Case Study 2 - Motor Controller

Similar to the discussion in the case study above, this designer relied on a software tool to replace a detailed knowledge of the target architecture. In this particular case, the target architecture was an Act 1 device with the designer familiar with an Altera device and its tools. The design flow used for this project is shown in Figure 32 and relied on a netlist translator for the detailed implementation of the A1020 circuit. No independent steps were taken to verify the correctness of the translation, with the translated netlist

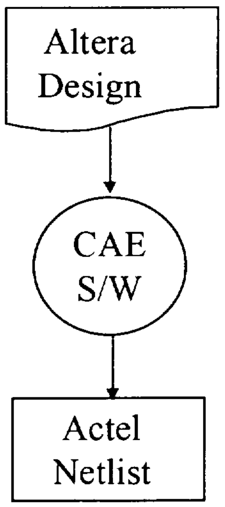


Figure 32. Design flow using netlist translation. An Altera design was mapped to an Act 1 device using CAE software. The software created logical structures unsuitable for the space flight environment.

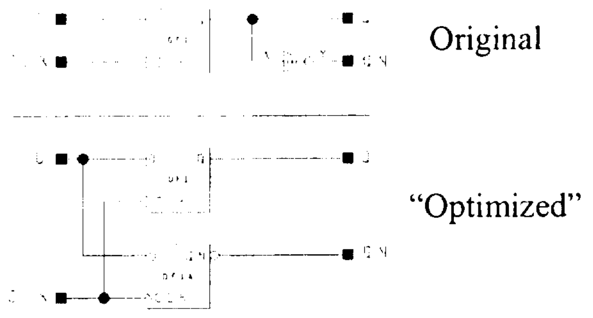


Figure 33. Flight design, before and after netlist translation. The translation software took a single flip-flop in the Altera design and mapped it into two flip-flops in the A1020 design. This circuit, although logically correct, is unacceptable since it is used both as a synchronizer and for control of high motor currents. Differing values in each of the two flip-flops resulted in an over-current condition.

directly input into the back end place and route tool.

The circuit failed and an analysis was performed. Figure 33 shows a sub-circuit, first as designed in Altera technology and below, how the CAE tool implemented it in the A1020. The output of the flip-flop, and its complement, are used as part of a driver circuit for a motor controller. Logically, in the Altera circuit (top), Q and QN must have opposite values. This is required for safety since if both outputs are asserted there will be a high current condition.

By inspection, we see that the safety provided by the original circuit is now gone. It is physically possible for both Q and QN to be simultaneously asserted. This can happen several ways. First, an unintended transient may upset the circuit. In general, designs that are sensitive to this environmental effect should be avoided. Additionally, the input to the flip-flop is asynchronous. Since the routing delay to each of the two flip-flops is different in the A1020 and t_{SU} of each flip-flop can never be identically mapped, it is conceivable that the two flip-flops will both be Q and QN will have the value of logic '1'. This is what happened in test. There is an additional concern, since this circuit was intended for a high-radiation environment, that a SEU could force both outputs to be driven high. The CAE software did not "understand" the implications of the asynchronous input for this circuit nor did it take into account SEUs. By shielding the designer from the implementation details, engineering was unaware of the risks of this circuit until a failure was observed on the flight hardware.

D. Replication and Timing Optimization

The loading on a particular signal can affect system performance if it is in the critical timing path. Another factor is where the loads are located, in particular for FPGAs which may have a relatively expensive routing network. For these nets driven by a combinational gate, one standard technique is to simply *replicate* the gate. The load will be split between the different copies and each gate will be located to optimize system goals. Similarly, for sequential logic, the output of a flip-flop may be in the critical path. Modern design software, such as logic synthesizers and back end place and route tools, may automatically replicate sequential logic. Each tool behaves differently and has settings that affect its performance. A generic schematic of this is shown in Figure 34.

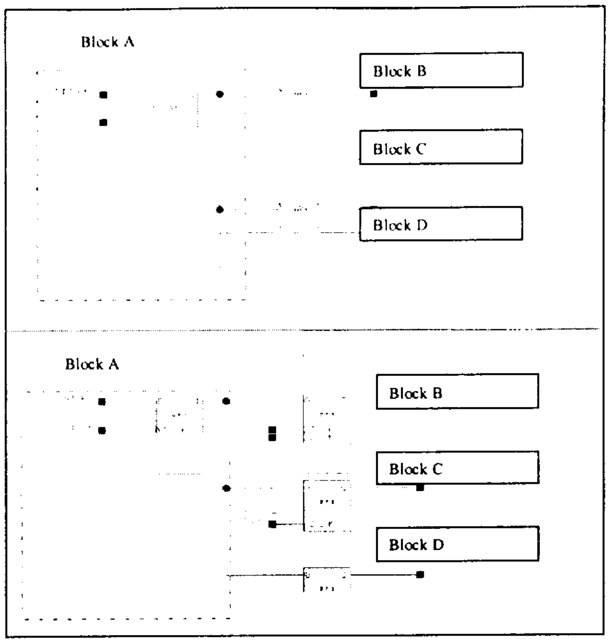


Figure 34. Logic replication to improve performance. CAE software can create multiple copies of flip-flops to improve system timing. This has been observed both in synthesis and in back end software.

As shown above, there is a *logical* equivalence between the two circuits. *Electrically*, however, they can perform quite differently, particularly in the space environment. In the case of an SEU, different parts of the circuit will see different values for the same variable. The CAE software does not put in any checking circuits for error detection or correction. Finally, since the replication does not appear in the logical definition of the circuit, either schematic or HDL code, the design engineer can not add logic for detection or correction efforts. Each tool needs to be configured properly and constrained to prevent replication when its effects could be harmful. Control of the synthesizer may have to be performed indirectly as there may not be a global command for turning off the replication feature.

E. Robust Design and Lockup States

A clear advantage of designing in an HDL such as VHDL is the ability to rapidly describe finite state machine (FSM) behavior using names for the states. This allows the

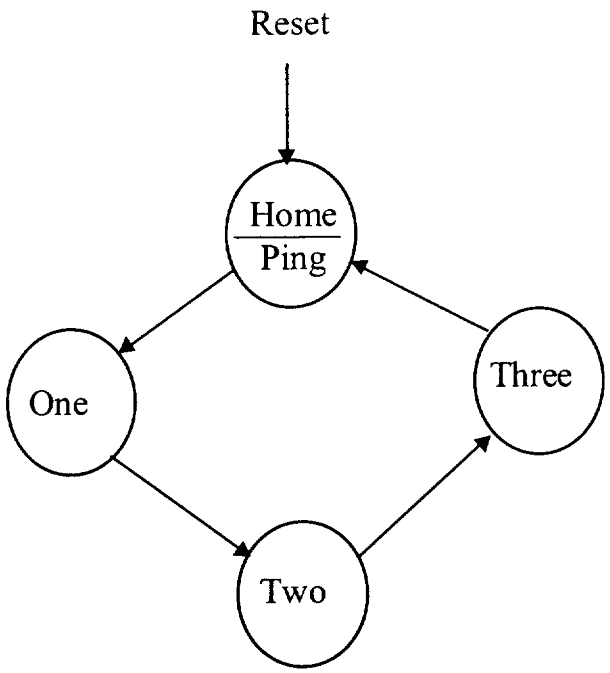


Figure 35. Simple finite state machine. CAE software may implement logic that has lockup states.

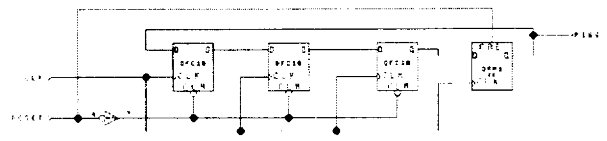


Figure 36. One-hot implementation of a simple FSM. This circuit, generated by a VHDL synthesizer, produces an state machine implementation with lockup states.

design engineer to concentrate on *what* the FSM is doing, not *how* it does thing. One of the tasks that a tool such as a VHDL synthesizer performs is called state assignment, where different physical states are assigned to the designers logical states. One popular and preferred encoding method for FPGAs is referred to as "one-hot encoding." Using this technique, there is one flip-flop for each state and exactly one flip-flop is set. An example of this, for the FSM shown in Figure 35, is shown in Figure 36. This circuit was generated by a commercial VHDL synthesizer.

Although this machine-generated circuit is logically correct, a close examination shows that electrically, for many space-borne applications, it is unacceptable. Clearly, this circuit has lockup states. These states may be entered as a result of noise or other transient or an SEU. It is not a robust circuit for a critical application.

The lockup states can be broken down into two classes. First, if the flip-flop that is set gets reset, then all flip-flops in the FSM implementation will be zero and the FSM will be stuck with no logic to transition into any valid state. Alternatively, more than one flip-flop may be set. Again, there is no logic to force the state machine into a valid state or indicate an error.

At first glance it would appear that a VHDL solution would easily cover this situation, by using the `Others` clause in the `Case` statement. However, this VHDL construct only applies to the set of logically defined states in the system; it does not apply to the set of physical states represented by flip-flops in the implementation. As such, the `Others` clause would ignore any physical error states.

This analysis can be extended and it can be shown that other FSM implementation schemes, such as binary or gray encoding, are susceptible to the same lockout problem. This is discussed in more detail in [19].

One synthesizer vendor has implemented a "safe" mode that detects an illegal state and then takes corrective action by forcing it into the home state. Logically, this *appears* to solve the problem. Electrically, an examination of the implementation revealed that the reset signal forcing a recovery is clocked on the opposite edge of the clock! This can easily lead to timing problems if the design engineer is not aware of the implementation. Again, just typing in a command to a CAE tool, telling it to make things "safe" does not guarantee a good quality circuit.

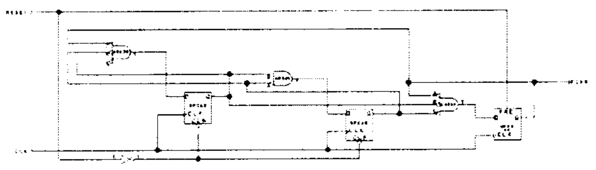


Figure 37. Modified one-hot implementation. This circuit is more robust than the implementation shown in Figure S9. Results vary as a function of synthesizer, revision level, and settings. Note that this implementation uses the all 0's state as a legal state.

The circuit in Figure 37 is in a modified one-hot topology that is more robust than the one shown in Figure 36. In this case, the state assignment algorithm used the all 0's state as a legal state with no outputs generated during that state. An examination of the next-state equations shows the improved robustness of this circuit.

Working from left to right, we denote the three flip-flops FF1, FF2, and FF3. First, for the all 0's case, FF1 and FF2 will be cleared while FF3 is set, since the three inputs to the AND gate are inverted. Additionally, FF3 will not be asserted for any other combination of flip-flop values. FF1 will only be asserted if it is already cleared (it can not stay high for two consecutive state times), FF2 is cleared, and FF3 is set. It can not be set if two or three flip-flops are set, eliminating that problem. Lastly, we examine the next-state equation of FF2. FF2 will be set if it is already cleared and if FF1 is set. The state of FF3 is a "don't care." Looking closer at the case of FF1 and FF3 being set, we see that the following will occur. First, FF2 will be set, as described above. FF1 will clear, since it's next-state equation forces the flip-flop to clock in a logic '0'. In similar fashion, FF3 will also clear. This leaves the state machine in a valid state.

It is difficult to generalize how CAE tools will perform the task of state assignment and FSM generation. The results vary based on the tool used, its settings, and even the code that is input to it. The tools are uncontrolled and their algorithms are not documented. Frequently, this results in a "try it and see what happens" approach to what is normally called engineering.

If tools are used in this fashion, their output must be understood for critical applications. Normally, design engineers will ensure that their state machines do not have lockup states. That is one of the basics taught in early logic design classes. By delegating detailed design work to CAE software, we ironically find that CAE vendors are driven by so-called "quality-of-results" which generally mean resources and speed, not the quality of the logic design itself.

F. Correct by Construction

It is tempting, when using tools such as macro generators and good HDL synthesizers, to assume that their output is "correct by construction." For HDL code, the design is often simulated first at the source code level, showing that the design is logically correct. This section is a bit general and is included as a reminder and specific analyses are not included.

However, it is felt that the synthesized netlist must be fully synthesized and subject to timing analysis. For example, using a known reliable macro generator, it has been found that under certain conditions it will produce a logically incorrect netlist, even when the target family remains unchanged. Examples include designs which fail the design rule check, outputs that are inverted, or incorrect terms, when generating linear feedback shift registers. For VHDL synthesis, errors have been reported in the inferences of latches vs. flip-flops, for example. There is enough evidence of problems in design software that full simulation and timing analysis must be performed. Additionally, the behavior of CAE tools is not always consistent throughout revision levels with "bugs" and different algorithms occasionally introduced; complete re-verification of a flight design is required when tools are upgraded.

G. Delay Generation

The circuit shown below in Figure 38 was intended to create a delay. This is in general a poor circuit and the lack of understanding of the CAE software caused the delay not to meet the designer's intent. Upgrades to the CAE software changed the default behavior. The new version would analyze the circuit, determine that the inverters were logically not required, and eliminate them. A record of this is available for export to an auxiliary file and no direct notification is given to the user. Additionally, early versions of this software would fail to report the eliminated logic to the report file. The optimizer's behavior can be over-ridden by use of an attached attribute. The designer was unaware of this.

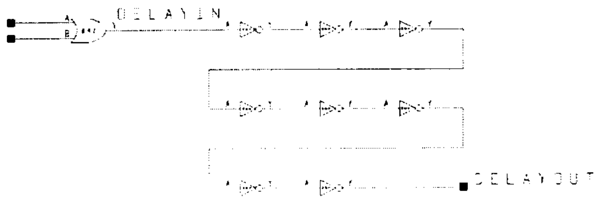


Figure 38. Delay generation. This circuit (not a particularly good one) failed to achieve the delay intended by the designer. A recent revision of CAE software changed its default behavior and the optimizer eliminated the logically unneeded gates.

V. High Level Principles, Technology

A. Specifications - General Principles

Many digital systems are designed without any specification being produced. Some causes for this are discussed below, in Section B. Obviously, this promotes miscommunications between various project personnel and makes a thorough analysis impossible. This often leads to a development by test, rather than using test to verify the analysis. For many digital systems, it is impossible to demonstrate flight-worthiness by a test program. Additionally, the test campaign must be, to a certain extent, ad hoc, since there is no specification. Similarly, for expedience, specifications are often ignored, the formalities of maintaining it get in the way, and changes are made quickly and late in the game. This often leads to problems, as the full impact of the change is not properly assessed, leading to further changes in a time frame that has grown shorter. As many of the faults documented here, this is unfortunately a quite common practice.

B. Specifications - Gate Array Operation Incorrect

The development of gate arrays is typically an expensive undertaking. The non-recurring engineering (NRE) charges are typically quite high from the foundry. The second contributor to the high cost is, in part, the first reason for high cost. The large NRE requires that there be a minimum of developmental errors, with a first-pass success a crucial goal. While an FPGA in a low-cost package has a cost of approximately $25 to $50, the NRE for a hardened gate array can be $50,000 or more. Secondly, an FPGA can be "burned" in under an hour; the "spin time for an ASIC is measured in months. First-pass success is crucial for both budget and schedule.

The reasons for the failure of this gate array are all common sense items. Yet, they did happen here and are, unfortunately, common in the industry today. A contributor to the failure was that there was no continuity of personnel. A detailed discussion of this is outside the scope of this paper, although the authors note that there are many reasons for this, many of which are manageable. Although this ASIC had a specification, changes to the specification were not frequently documented. Why did this happen? A contributor to this was another common problem, features being added and deleted during development. While some changes are inevitable during development, frequent functional changes promote problems with communications between various personnel, particularly if personnel are changing during a project. Additionally, as changes are being made, it is difficult to assess all of the impacts of the change to a complex circuit, particularly if the original designers have moved on, and subtleties are not fully documented and understood. Lastly, keeping a specification up to date is a lot of work. If the environment is such that changes are constant, then personnel are likely to put off updating the specification until things are stable. As we have witnessed, some projects never approach a stable state.

C. Specifications - FPGA Updates Delays Projects

These cases, too, is based on problems of proper specification. In the first case examined, software changes and drifting software requirements were forcing changes of FPGA requirements. Clearly, this is a system engineering and management issue. In the second set of cases grouped here, the system requirements themselves were not stable and the FPGA designs had to be updated. This environment promotes failures, mad efforts to chase down and eliminate bugs, and a constant tension. It is an environment that promotes errors. FPGA designs should work with first-pass success.

D. Reliance on Simulators - General Principles

Logic simulators started to become popular in the early 1980's to verify hardware designs prior to construction. The algorithms in the simulators rely on models, either constructed by the simulator vendor, a third party vendor, of the users themselves. Often model parameters are modified by the design organization to account for environmental conditions unique to a particular mission. Over the past several years, hardware design language (HDL) simulators have become popular, either executing VHDL or Verilog code for verification. The VHDL code, for example, will having timing parameters extracted from the model of a gate array, after the place and route operations. Thus, both logic simulators and HDL simulators can be used for functional and timing verification, in principle. The quality of the results depends on the fidelity of the models and the capabilities of the simulators. Another popular design methodology is to use the logic or VHDL simulator for functional verification and a static timing analyzer for timing verification and the determination of adequate margins.

There are several high level issues with using simulators for verification, that can lead to errors. These include the following:

- Run time limited
- Number of test vectors
- Test vector/bench generation
- Number of operating modes
- Time for modeling external circuitry
- CAE software limitations

The generation of test vectors (logic simulator) or test benches (HDL simulator) is a large amount of work. This has been estimated to be from 40% to 75% of the development effort. This, coupled with the limitations of the software, must be addressed up front in a development program.

E. Simulators - Case Study 1

An arithmetic error surfaced in a design from a latent logic error. The entire circuit was not properly simulated and the error was detected in hardware. There were two causes for this error. First, the simulator being used could only simulate for 1 ms; the circuit had a 125 ms cycle time. Secondly, a full simulation of all combinations of inputs was not practical, as there were too many combinations. Frequently, either test vector generation or test runtime can be limiting factors in a simulator-based verification effort. While 1 ms is short for a simulator's limit, modern, popular logic simulators have shown to have time limitations that were considerably less then the cycle time of the hardware under simulation. Additionally, the relative slow speed of simulation may make complete testing impractical, for some circuits.

F. Simulators - Case Study 2

For this case, a design originally built in an FPGA was converted to an ASIC. The development methodology did not have any gate level design reviews at any stage of the project and the project relied on the simulator to catch the errors. Unfortunately, the test vectors from the FPGA version of the implementation were not run on the ASIC version. Analysis of the vectors showed that the test vectors would have detected the fault present in the ASIC version of the design.

VI. CONCLUSION

This paper has discussed and analyzed a large number of failures in spacecraft systems. Clearly, the majority of these errors should have been caught earlier in the design cycle. Although not an exhaustive list of failures, we have included a select set to illustrate general principles and share experiences with designers of future logic systems.

It is noted that the space industry has seen large numbers of failures over the past few years resulting in losses of billions of dollars. Are these industry failures related to each other in any way? Is there a common root cause to these failures? Perhaps the failures are simply a statistical "blip." This is an open question and various committees and study groups are exploring the issue. It will be interesting to see what parallels, if any, exist in the failures of the space community in general and the logic design community in particular.

In discussing logic design, we grouped the failures into three general categories. These were high level principles and technology, low-level hardware, and low-level software. Many of these errors seem obvious; yet, they are seen frequently and should be discussed. Some errors are a result of the injection of the latest commercial technology, originally not designed for high reliability systems.

At the highest level, we saw many "Mom and apple pie" rules violated. For example, the lack of stable specifications, continuity of personnel, and detailed reviews remain. Reviews in particular are worthy of additional mention. Sometimes the detailed design review is skipped; other times, the review consists of perhaps an hour or so of discussion, with the reviewers being forced to "sight-read" the material in "real-time." This class of review is sufficient only for checking off a so-called *action item* that needs to be done. This can not replace a solid, detailed review. Lastly, for errors of this class, it is obvious that simulations and testing can not replace proper design and analysis. As was shown, some circuits can not be verified by test. In addition, it is difficult to cover all cases with simulations. Furthermore, the simulators and models are limited in their fidelity and often aren't a true model of the physical circuit and all of its characteristics. Understanding the limits of a simulation is critical to the proper use of this tool.

At the lower levels, both hardware and software, most of the errors have a common root cause. These errors are typically relatively simple, when properly explained. The circuits in question are not overly complex and there is little mathematics involved. Indeed, by penetrating the design, going to yet lower levels, the flaws become rather obvious. One technique for dealing with design complexity is abstraction, with the implementation of lower levels being a "solved problem." Not understanding the underlying technology is the root cause for most of the errors analyzed in this paper.

There are a number of reasons that we can postulate for these problems, which seems to be of almost epidemic proportions.

First, the logic industry tends to try to shield the engineer from the lower level technology details. This, in principle, enhances productivity and to no small extent is a function of marketing departments. For example, we see one manufacturer who claims on their world wide web site that their parts "are instantly operational on power-up." This is most definitely not the case. Another vendor marks their parts as "radiation-hardened" although they are considered radiation-soft with the MTBF for upsets in control bits is measured in hours. Synthesizer vendors promote device independence if the engineer designs his logic using an HDL and lets the synthesis tool generate the logic for him. The algorithms that the synthesis tool uses are not published or controlled. Additionally, for some newer models of FPGAs, vendors are no longer providing libraries for schematic tools; therefore engineers lose direct control of the hardware as schematic capture is not supported is being discontinued. As a result, one *must* code in an HDL and let a third-party synthesis tool generate the logic. CAE tools do not always produce good, robust circuits for high-reliability applications.

Management plays a role in today's logic design in a number of ways. First, by observation, the line manager is often not technically active nor up to date on the technology used in flight programs. Consequently, supervisory personnel often can not provide proper detailed guidance nor be able to quickly spot flaws and point out solutions. This has also been noted in other technical fields [20]. Many managers today are not promoted for technical and leadership skills. In a welding shop, one will find that the foreman is by far the most skilled welder and can quickly decide on how to solve a particular problem; for logic design, that is not the case. The leader is more than likely to be an administrator. Virtually all of the designs that we are aware of have been subjected to a number of design reviews such as the traditional PDR and CDR and sometimes a "peer review." Short presentations have replaced true independent detailed reviews. While this type of review can catch some errors, it is often insufficient, as the details of a design can not be adequately reviewed in a presentation lasting perhaps 15 minutes. Another factor is the training of logic designers. Engineers are frequently assigned positions not based on skill and experience. As a result, there are inadequately supervised engineers who are not aware of basic concepts such as metastable states and the SEU performance of the devices that they are using.

Engineers must probe farther and deeper into the technologies that they are using. The devices in use today have far more power than the devices used 30 or 40 years ago. While early digital electronics was constructed from simple components, that is not the case today, with device complexity orders of magnitude higher. Understanding the technologies and, just as importantly, the tools, is fundamental to the design and construction of reliable systems. The failure to penetrate the technologies and understand how they work, not just how to use them, is also a fundamental skill that needs to be developed. A reliance on readily available and *seemingly* powerful tools to understand and manage the lower levels of the technology will result in failure.

Lastly, there has been a lot of discussion of the effects of strategic policies such as "faster, better, cheaper" or "FBC." One can make a case that taking more time, having more thorough reviews, and running more tests will help reduce the frequency of errors. However, it is noted that large, expensive programs also have had problems. Indeed, the examples used for case studies in this paper came from the most expensive programs as well as FBC programs. Additionally, the designs came from industry, government, and academic institutions. It is felt that the problem, or the "disease," is widespread. No correlation has been found linking logic design problems to the cost of a program or the organization managing it.

REFERENCES

1. D. Branscome, et. al., **WIRE Mishap Investigation Board Report**, National Aeronautics and Space Administration, June 8, 1999.
2. R. Katz, **Small Explorer WIRE Failure Investigation Report**, National Aeronautics and Space Administration, May 27, 1999.
3. J. Oberg, "Why the Mars Probe Went Off Course," *IEEE Spectrum*, Volume 36, Number 12, December 1999.
4. M. Williamson, "Satellite In-Orbit Failures - A Rising Trend?" *Via Satellite*, December 1998, pp. 18-24.
5. W. Larson and J. Wertz, **Space Mission Analysis and Design - Second Edition**, Microcosm, Inc. and Kluwer Academic Publishers, Wirth, (1992) p. 709.
6. T. Lewis, "Primary Processor and Data Storage Equipment for the Orbiting Astronomical Observatory," *IEEE Transactions on Electronic Computers*, pp. 677-687, Dec. 1963.
7. E. Hall, **Journey to the Moon: The History of the Apollo Guidance Computer**, American Institute of Aeronautics and Astronautics, (1996).
8. R.L. Alonso, H. Blair-Smith, and A.L. Hopkins "Some Aspects of the Logical Design of a Control Computer: A Case Study," *IEEE Transactions on Electronic Computers*, December 1963, pp. 687-697
9. A. Sharma, **Programmable Logic Handbook**, McGraw-Hill, (1998).
10. R.B. Katz, R. Barto, P. McKerracher, B. Carkhuff, and R. Koga, "SEU Hardening of FPGAs for Space Applications and Device Characterization," *IEEE Transactions on Nuclear Science*, **NS-41**. pp. 2179-2186 (1994).
11. Reproduced from NASA EEE Links, Vol. 1, No. 4, October 1995
12. P. Alfke, "Two Simple Solutions for *Tricky Problems," Xcell*, **Issue 34**. pp. 54-55 (1999).
13. Xilinx, Corp., **The Programmable Logic Data Book**, p. 4-46 (1998).
14. Actel, Corp., "A Power-On Reset Circuit for Actel Devices," **FPGA Data Book and Design Guide**, pp. 3-81 to 3-82 (1995).
15. **IEEE Standard Test Access Port and Boundary-Scan Architecture**, IEEE Std 1149.1a, IEEE Computer Society (1993).
16. R.B. Katz and J.J. Wang, "Use of SX Series Devices and IEEE 1149.1 JTAG Circuitry," August 17, 1998.
17. J. Greene, E. Hamdy, and S. Beal, "Antifuse Field Programmable Gate Arrays," *Proceedings of the IEEE*, **Vol. 81, No. 7**. pp. 1042-1056 (1993).
18. Quicklogic Corp., **Databook**, page 2-4 (1996-97).
19. R.B. Katz, J.J. Wang, J. McCollum, and B. Cronquist, " The Impact of Software and CAE Tools on SEU in Field Programmable Gate Arrays," *IEEE Transactions on Nuclear Science* (1999).
20. J. Oberg, "NASA faster, cheaper, but not better," *USA Today*, December 7, 1999.
21. Katz, R., J. Wang, J. McCollum, and B. Cronquist, *Current Radiation Issues for Programmable Elements and Devices*, **IEEE Transactions on Nuclear Science,** 1998.

APPENDIX A

A. Basic FPGA Architecture

This Appendix will provide a basic overview and illustrate some of the key differences in architectures and configurations of three FPGA technologies. A detailed review of FPGA architecture is not included here and the reader is referred to [9] for additional information. Because of space limitations, the architectures and implications of different application specific integrated circuit (ASIC) technologies are not included in this section as well as other FPGA devices families that are still actively being used.

There are many features of FPGAs and ways to classify them. In this section will shall discuss the configuration/routing mechanism, the array topology, and the core logic cells for the AT6K, SX, and XQR4000XL families. These devices are the latest targeted at space flight applications.

FPGAs can be classified into re-programmable devices and one time programmable (OTP) categories. The re-programmable devices may be programmed many times and either volatile SRAM or a non-volatile EEPROM technology cells holds the configuration. Routing is achieved by having the output of the configuration memory bias an n-channel FET that either makes a connection or isolates routing tracks. SRAM cells in current FPGAs are susceptible to soft errors from either heavy ions or protons. The planned hardened version of the AT6K and the XQR4000XL are configured via SRAM cells. OTP FPGAs typically use antifuses for configuration, either an oxide-nitride-oxide (ONO) structure or a metal-to-metal configuration. Signals pass through programmed antifuses, which connect or isolate routing segments. The antifuse is normally in the off or high impedance state; programming a metal-to-metal antifuse results in a low-impedance connection, typically around 25 ohms. Radiation-hardened antifuses have been recently developed and verified in SX-series devices [21]. Configuration memories also are used to configure logic functions, input/output parameters such as drive strength, and other selectable parameters.

The FPGA core consists of logic and routing resources with different manufacturers choosing different topologies. The type of configuration memory heavily influences this decision as SRAM cells, pass transistors, and ONO antifuses all require structures built into the integrated circuit substrate. These devices may use a channeled architecture as shown in Figure A-1 or a tiled architecture (AT6K) with both horizontal and vertical channels. Metal-to-metal antifuses can be located above the logic resources with the routing channels eliminated, if a three or more metal layer fabrication process is used. This is done on the SX series, with the switches located between metal layers 2 and 3.

The logic cell structure also differs amongst the different manufacturers and is frequently updated as FPGA architectures mature. One key feature of the logic block is its granularity. The XQR series has a relatively coarse granularity, with multiple look up tables (LUTs) and outputs. The SX series, for example, is relatively fine grained, with two simple modules, the C-Module for combinational functions, and the R-Cell for storage. In general, devices with larger delays through the routing system (re-programmable devices) have a coarser grained logic cell. Modern devices such as the XQR and SX series devices also employ "direct connects" for dedicated, high-speed signaling between modules that do not pass through a programmable switch. The XQR makes use of this for carry propagation in adders, for example, with a typical delay of 300 psec per bit.

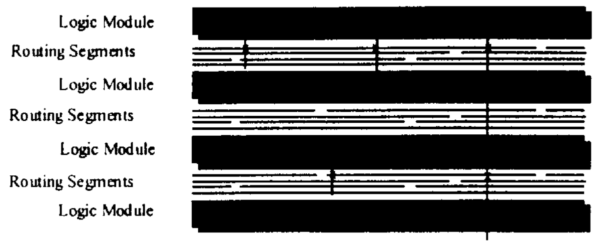
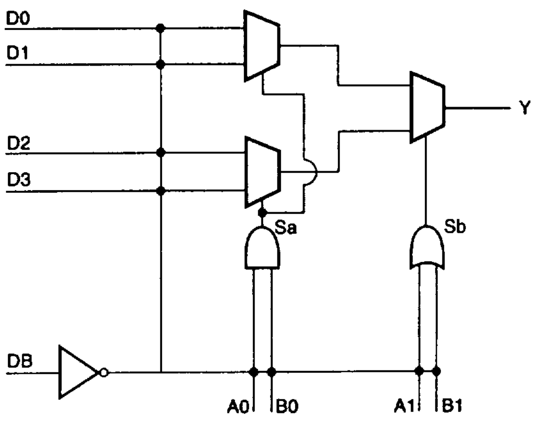


Figure A-1. Generic structure of a channeled FPGA architecture. FPGA cores consist of logic modules and routing resources with input/output blocks typically located around the edges (not shown). Other topologies include tiles, with vertical and horizontal channels and sea of modules, with the routing resources located above the logic modules.

Two basic techniques are used for the logic. In the first, used by the XQR series, small blocks of pre-loaded SRAMs are used as look up tables (LUTs) with module inputs providing the address. Additionally, these LUTs may be reloaded during device operation, providing a variety of memory configurations for the application. Not shown in the figure below is dedicated high-speed carry logic, to speed arithmetic operations. In the second basic technique, hard-wired logic is used. The AT6k uses what is essentially a half-adder function. The SX series has two types of modules, one for combinational logic based on a multiplexer, the other a storage cell with an enable function. Block diagrams for several architectures are shown below in Figures A-2, A-3, and A-4.

B.FPGA Device Usage

ASICs and FPGAs are available, or will be available, at different reliability levels, ranging from inexpensive commercial-off-the-shelf (COTS) devices to high-reliability, radiation-hardened devices as well as different devices speeds, capabilities, and configurations. This section will give an overview of these capabilities and trade-offs between different FPGA device families.



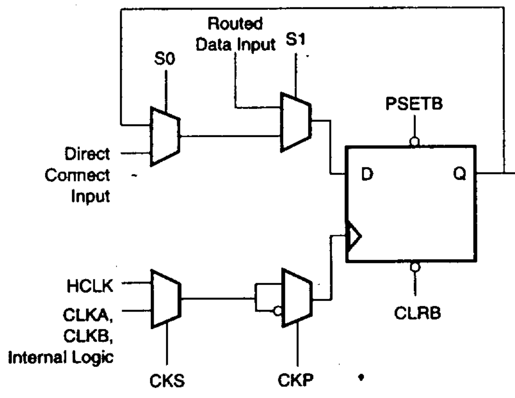


Figure A-2. SX Series C-Module (above) and R-Cell (below). The multiplexor-based C-module can implement over 4,000 logic functions. The R-cell implements a flip-flop with enable. Not shown are routing resources which connect C-Modules and R-Cells into clusters and super-clusters supporting direct connects have a routing delay oj 100 psec and fast-connects having a delay of 400 psec.

The Actel OTP devices have been the primary FPGA technology used in space flight systems to date. In general, they can be regarded as radiation-tolerant and are used for a variety of control and data processing applications. The members of the Act 1, 2, and 3 families are considered relatively small, with the largest flyable device consisting of about 20,000 "PLD Gates."

The newer, high-speed, low-power SX and SX-A series has devices in various stages of development. By modifying COTS technology, greater than 100 krad(Si) total dose performance levels have been achieved, with some SX-A prototypes exceeding 200 krad(Si). There is no single event latchup (SEL) and there is a radiation-hardened antifuse for configuration. The flip-flop performance of the basic device is considered radiation-tolerant; modified versions via software have shown radiation-hard performance levels, at the cost of some speed and logic resources. A radiation-hardened flip-flop is being incorporated into the next silicon revision. Since these are OTP devices, they obviously can

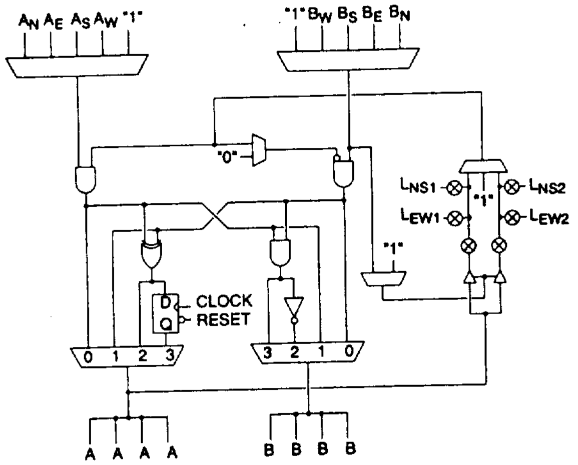


Figure A-3. Cell structure of the AT6k Series device. The four sides (north, south, east, and west) are functionally identical making each cell symmetrical. The clock and reset inputs are common to a column. The XOR and AND functions can directly implement a half-adder. Pass gates may be used as routing resources "turning" signal lines (e.g., L_{NS1} to L_{EW1}). These are shown in the right hand side of the figure.

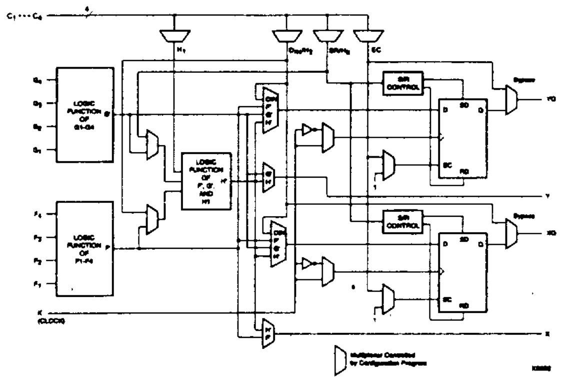


Figure A-4. Configurable Logic Block (CLB) of the XQR4000XL series. This coarse logic module provides functionality by accessing two 4-input LUTs and one 3-input LUT. The multiplexers are controlled by configuration bits loaded before operation. The LUT may be used as user memory devices. Dedicated high-speed carry logic and its direct connection to adjacent modules in the same column are not shown. Limited routing and signal buffering is possible via the "Bypass" path.

not be used in applications that require a full device reconfiguration.

The Xilinx XQR4000XL FPGAs are a subset of their commercial XC4000XL series of devices. By special processing, including a 7 µm epitaxial layer to prevent SEL, the device is usable for certain classes of space-flight data processing tasks. The XQR4036XL has shown 60 krad(Si) total dose performance and no SEL, suitable for a wide range of space science missions. The flip-flop designs in the XQR series, however, are considered radiation-soft and must be carefully applied to particular missions. The XQR4036XL has a capacity ranging from 22,000 to 65,000 gates; the largest device in the series, the XQR4062XL, ranges from 40,000 to 130,000 gates. High *effective* gate counts can be achieved by the use of LUTs as on-chip memories, as described below. Note that for SX and AT6010 devices, on-chip SRAM must be assembled from on-chip logic resources which is far less efficient at making read/write memories.

The XQR device is configured by loading SRAM cells. These cells control resources such as routing connections and parameters such as output slew rates and input thresholds. Additionally, they load small RAM blocks that act as look-up tables (LUTs) which provide a great deal of the logic power; another important logic resource in this series is the dedicated high-speed carry/borrow logic. Alternatively, the LUT may be utilized by user circuits as on-chip RAM.

The last reprogrammable device to be discussed is the planned AT6010/rhFPGA. Originally a commercial device produced by Atmel, the design is currently being transferred to a 0.8 μm, silicon-on-insulator (SOI) process at Honeywell for radiation hardening. The device will be total dose hard for most space science applications, with a performance level of 200 krads(Si) and no SEL susceptibility. Flip-flops, for both user and configuration use, will have a minimum LET threshold of 30 MeV-cm^2/mg, which is radiation-tolerant; radiation-hard levels are a design goal. These 5V devices are relatively small, consisting of from 10,000 to 20,000 usable gates.

APPENDIX B

References for Metastable State Information

There are many available references on metastable states. The list below is a partial listing of material that has been collected. Please see http://rk.gsfc.nasa.gov/richcontent/General_Application_Notes/mestablestates/MetastableStates.htm for more information.

- http://rk.gsfc.nasa.gov/richcontent/General_Application_Notes/mestablestates/xilinx_metastable_considerations.pdf
- http://rk.gsfc.nasa.gov/richcontent/General_Application_Notes/mestablestates/xilinx_metastable_recovery.pdf
- http://rk.gsfc.nasa.gov/richcontent/General_Application_Notes/mestablestates/xilinx_metastable_recovery_2.pdf
- http://rk.gsfc.nasa.gov/richcontent/General_Application_Notes/mestablestates/meta_ti.pdf
- http://rk.gsfc.nasa.gov/richcontent/General_Application_Notes/mestablestates/cypress_pldmeta.pdf
- "Flip-Flops and Metastable States," CX Technology Design Manual, Chip Express, 1997, pp. 9-18 to 9-24.
- "Metastable States," The Art of Electronics, Horowitz and Hill, 1989, page 552.
- Daniel L. Stein, "Noise-Assisted Escape from a Metastable State," http://soliton.physics.arizona.edu/~dls/1.html
- http://www.ti.com/sc/docs/psheets/abstract/apps/sdya006.htm