FPGA Design Strategies for the Space Radiation Environment

Melanie Berg

Radiation Effects and Analysis Group
NASA Goddard Space Flight Center –
Muniz Engineering and Technologies

Overview

- **This session will present methodologies for Reliable Design implementation**
- **Topics that will be covered:**
  - **General Design Theory**
    - **Synchronous Design Theory**
    - **Reliable Reset Circuitry**
  - **Design Theory with Respect to Single Event Upsets (SEUs)**
    - **Impact of SEUs on Synchronous Design**
    - **Design Necessities for the Space Environment**
    - **State Machine Theory**

Introduction

- **Design complexity is ever increasing**
- **Design Methodologies and Process Definitions need to be developed and followed**
- **The space environment adds complicated parameters to the design process.**
- **There are many key components necessary to compose a reliable design…**
  - **Topics chosen in this session seem to plague designers the most.**

FPGA Design

- **FPGA Design is HARDWARE Design**
- **The job of the designer is to describe the circuitry via**
  - **schematics (outdated approach) or**
  - **some form of HDL (Hardware Description Language).**
- **Misperception that HDL is similar to writing software**
  - **The electrical characteristics of the circuit are generally overlooked and designs are improperly implemented**
  - **Multilayered design process is generally not followed correctly**

FPGA Design

FPGA Design is a multilayered process

Designers should be familiar with

- **HDL (i.e. VHDL or Verilog)**
- **Reading/creating Schematics**
- **Synthesis Tools**
- **Simulation Tools**
- **Difference in Technology Libraries**

Key Ingredients for Successful and Reliable Designs

- **VHDL – looks like software… but know your technology!!!!**
- **VHDL RTL must functionally match gate level (post synthesis) for simulation purposes. This requires enforcing strict coding rules… and…** Design for Verification
- **Designer must be familiar with the synthesis tools and their interpretation of VHDL code**
  - **Combinatorial circuits vs. Sequential**
  - **Clock structures and potential skew**
  - **Proper State machine implementation**
  - **Arithmetic circuitry**
  - **Clock domain crossings**
  - **Reset logic**
  - **When to use specific Synthesis directives**
  - **Speed**
  - **Etc…**

What is the Importance of VHDL Coding Styles.

- **No Synthesis tool can be as efficient as proper Coding Style**
- **ASICS and FPGAs will be smaller and faster.**
- **Proper VHDL Coding Style is easier to verify**
- **We would like to shorten the Design Cycle… Coding Style will affect**
  - **Quality of Synthesis: drive the tool to better results**
  - **FPGA mapping or design: can take advantage of the technology**
  - **Place and Route: designs that are well thought out will have a clean route**

Coding Style Specifics - Think "Hardware"

- **Architect with comprehension of your target's features (ASIC and FPGA)**
- **Separate Combinational and Registered blocks**
- **Watch out for inferred latches**
- **Pay attention to large fan-out nets**
- **Consider how you code state machines**
- **Be careful with designing long paths of logic**
- **Be aware of when you are able to use Resource sharing**
- **Consider Simultaneous Switching Outputs**
- **Stick to well established Synchronous Design Techniques**

Design for Signal Integrity

- **Simultaneously Switching Outputs can cause ground bounce (injection of noise into the ground plane)**
- **Identify potential SSO and spread them around the package.**
- **Avoid placement of asynchronous pins (resets, enables, etc.) near SSOs**
- **Place SSOs away from clock pins/traces**
- **When possible, use low slew outputs**
- **Strategically implement coding schemes that increase output integrety: i.e. Grey Scale … careful …**
  - **output of Grey circuit is glitchy (layers of combinatorial logic) and must be registered**

Design for Signal Integrity (Cont)

- **Register all outputs – this is not a recommendation – it is a general rule.**
- **Register all inputs before usage within circuit (asynchronous or synchronous)**
- **Increased capacitive load decreases the amplitude of the ground bounce by reducing the output slew rate. However, it will slow down transfer.**
- **Stagger the SSOs by using buffers within the FPGA so that they do not switch at the same time (if I/O protocol allows – due to speed)**
- **Check the FPGA's data sheet for the "safe" number of adjacent SSO pins for the specified design**

Synchronous Design

- **Why go through the trouble?**
  - **The design becomes deterministic due to all critical logic paths adhering to discrete time intervals (clock period).**
  - **Design Tools (Simulators, PAR, Synthesis, etc…) are easier to create.**
  - **A deterministic design reduces the complexity of the verification effort.**

Synchronous Design

- **A synchronous design adheres to the following definitions:**
  - **Number of clock regions should be minimized. All DFF's that have their clock pin connected to the same clock tree (that has minimal clock skew) are considered synchronous.**
  - **Clock gating should be avoided as much as possible (trade offs for power may have to override this requirement)**
  - **Asynchronous circuitry must use proper and deterministic techniques for passing data between clock domains**
- **A synchronous design consists of two types of logic elements:**
  - **Sequential : only accepts data at clock edge**
  - **Combinatorial : will reflect function (after delay) whenever its inputs change state.**

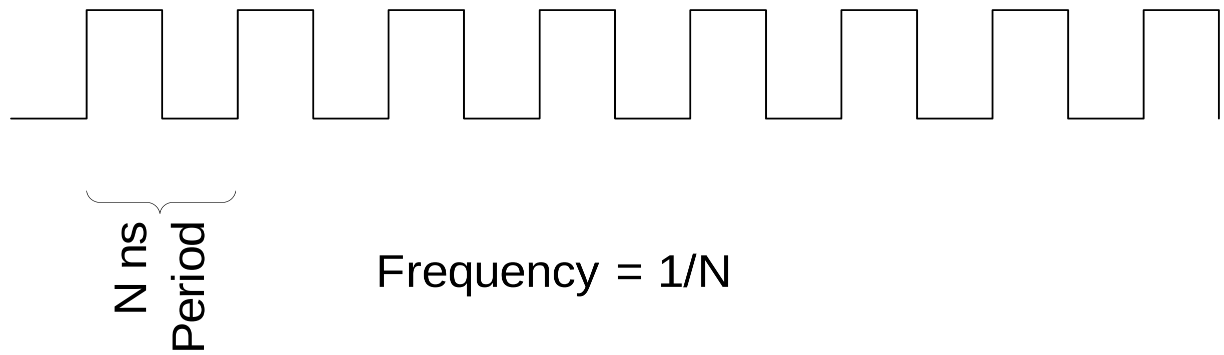
Synchronous Design (Cont)

- **Basically – a synchronous design suggests that**
  - **Every data path on the same low skew clock domain produces strictly deterministic timing analysis points**
  - **All data paths that communicate via different clock domains must contain the following characteristics:**
    - **Target domain synchronizes incoming data via a metastability filter, FIFO, or another well defined synchronization scheme**
    - **Source must hold data long enough for the target domain to synchronize**
- **Synchronization does not guarantee exact cycle that data will be available – it only guarantees that the correct data will be available within a defined range of clock cycles**

Synchronous Design: Clock

- **The clock is the heartbeat of every synchronous design**
- **It creates discrete and deterministic intervals**
- **It’s capacitive loading must be balanced (no skew)**
- **Must not enter the data path (only connect to the “clock” pin of a DFF)**

CLOCK - Input to FPGA/ASIC

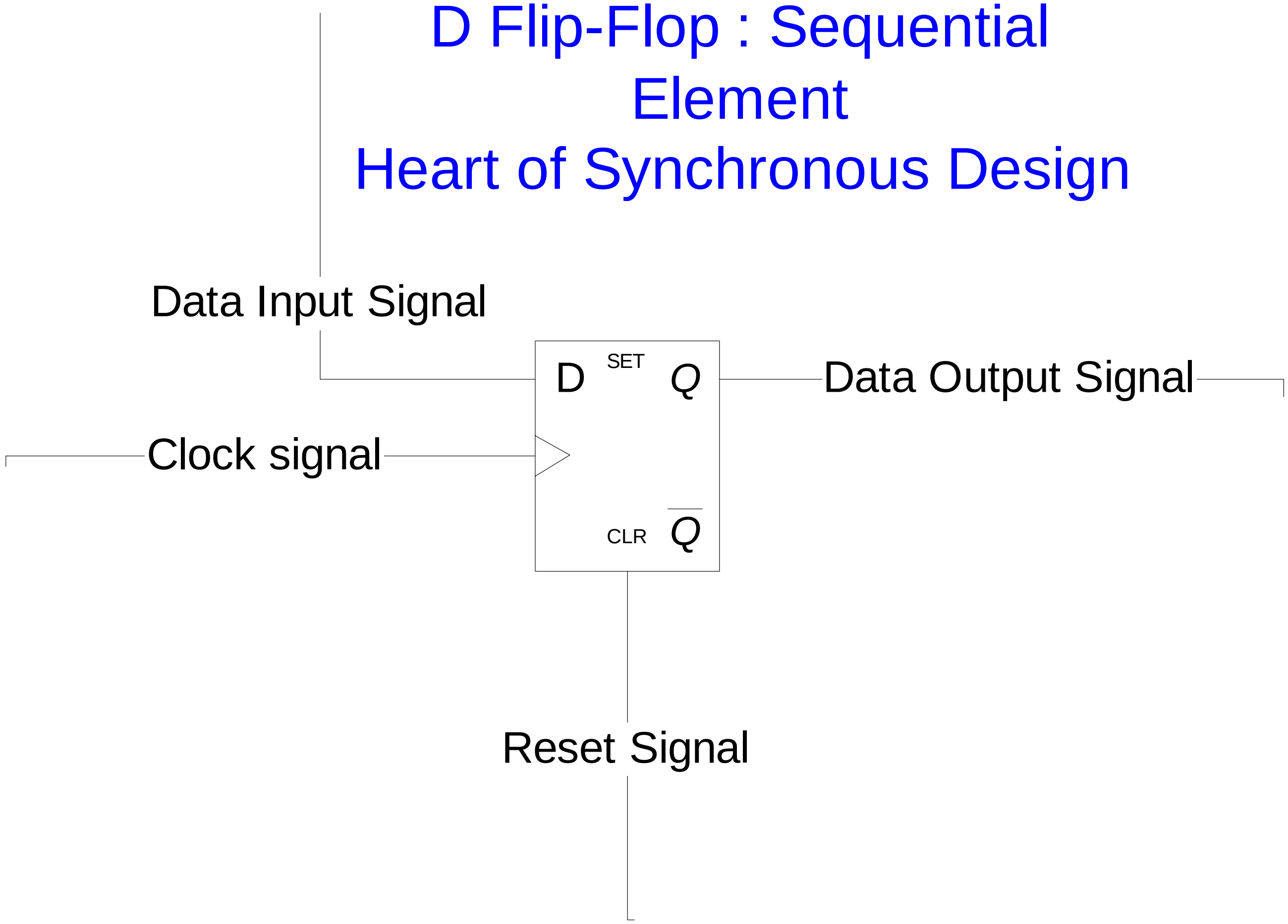


Frequency = 1/N

In a synchronous Design, The Clock Period will control

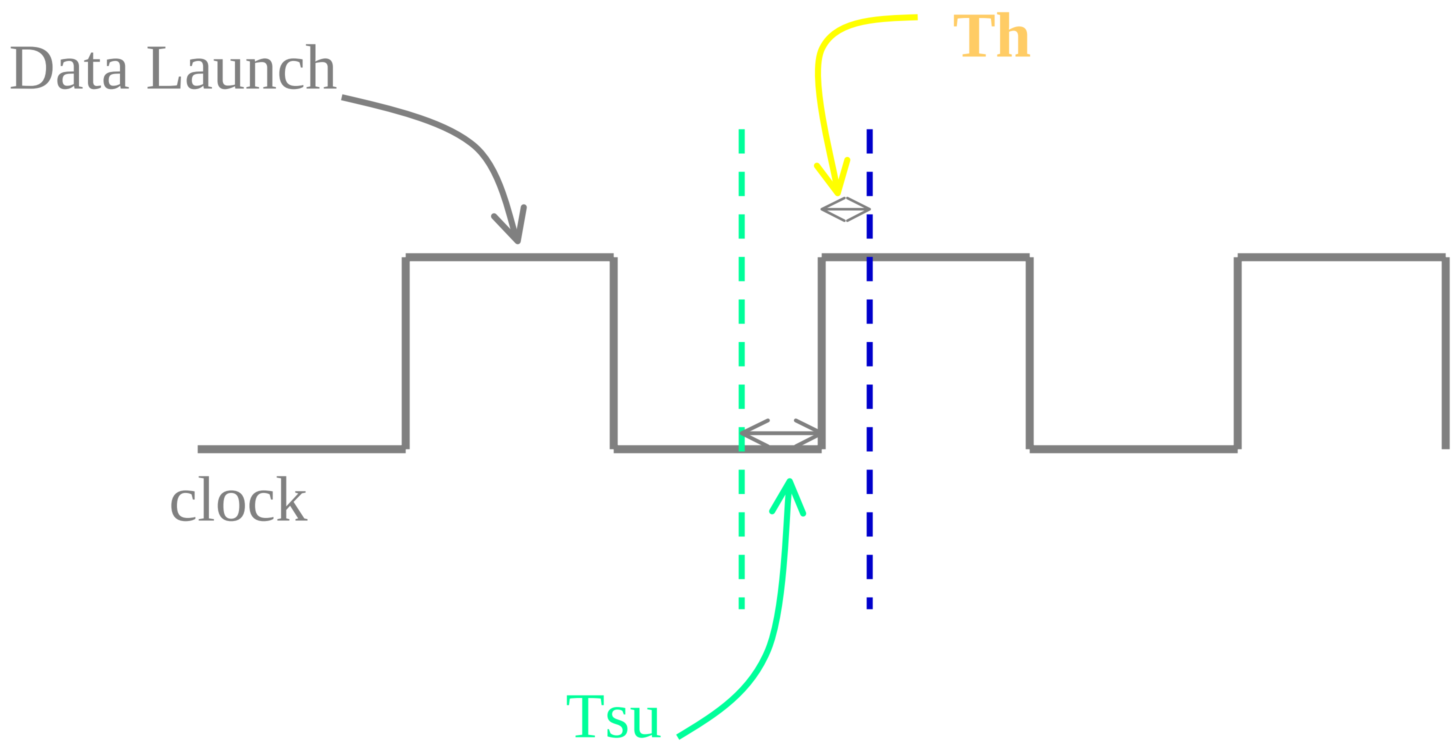
- *Amount of logic necessary to implement specified design*
- *Communication Schemes*
- *Architectural Decisions*

D Flip-Flop : Sequential Element
Heart of Synchronous Design



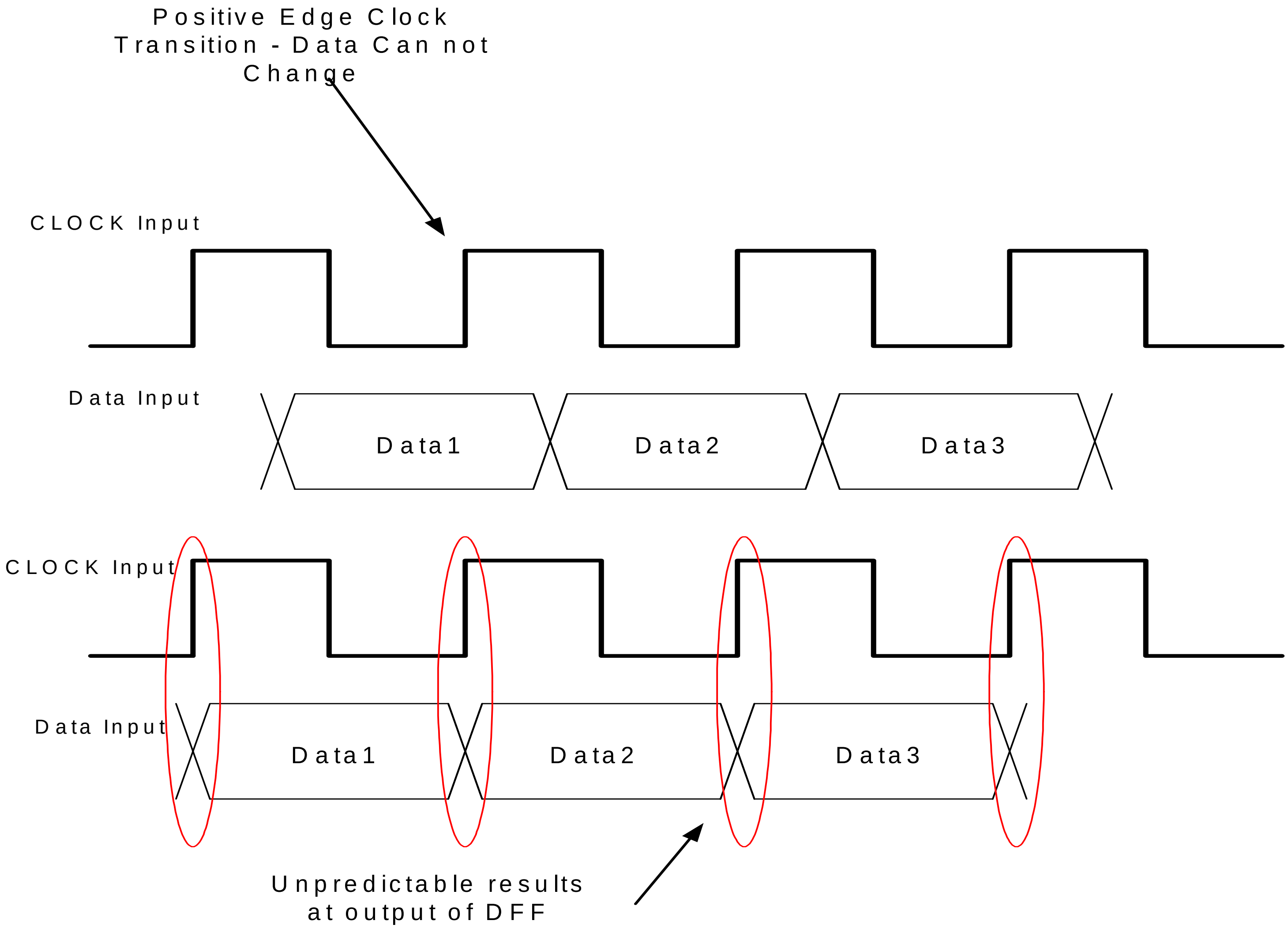
A DFF is clocked (sequential) logic where data is stored and reflected on the output at either the rising or the falling edge of a clock (following a clock to q delay).

Setup and Hold Time for DFF



Data must be stable during between Tsu and Th
Relative to the associated clock

Capturing Correct Data



Data Changing Near Clock Edge

- **Unpredictable Results:**
  - **May catch new data … but … may not capture it**
  - **Can cause a DFF to glitch or oscillate – metastability**
  - **Can cause a chain reaction of unpredictable results (state machine transitioning)**

Common Knowledge

- **This is all common knowledge and yet designers make the following common mistakes**
  - **Feed Asynchronous signals to state machines (and other DFF controlled logic)**
  - **Use multiple clock domains without synchronized filters (metastability filters or FIFOs)**
  - **Incorrectly define asynchronous domains as synchronous.**

Metastability

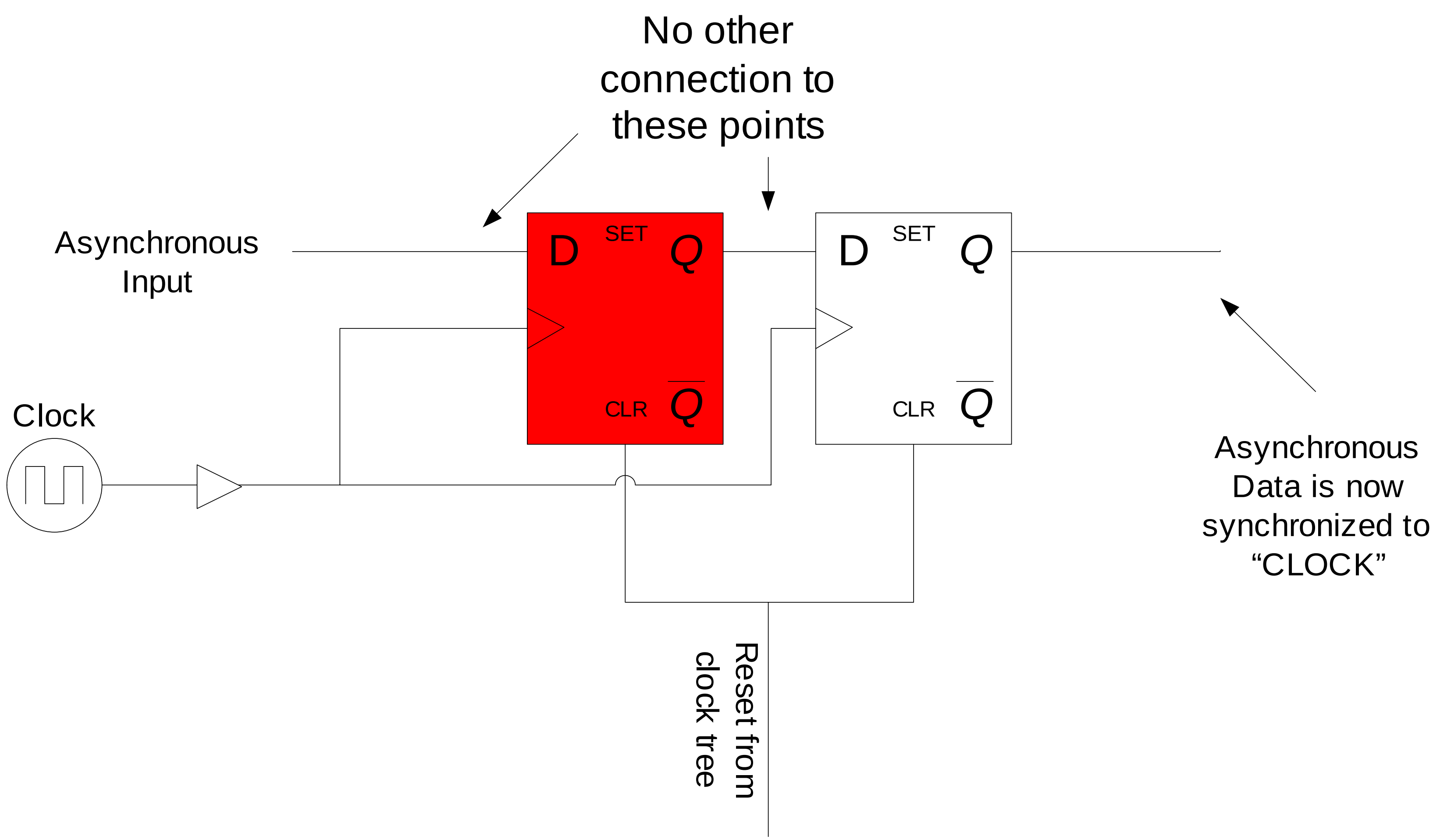
- **Problem: Introducing an asynchronous signal into a synchronous (edge triggered) system... Or creating a combinatorial logic path that does not meet timing constraints**
- **Output Hovers at a voltage level between high and low, causing the output transition to be delayed beyond the specified clk to q (CQ) delay.**
- **Probability that the DFF enters a metastable state and the time required to return to a stable state varies on the process technology and on ambient conditions.**
- **Generally the DFF quickly returns to a stable state. However, the resultant stable state is not deterministic**

Metastability Equation

$$\text{MTBF} = \frac{E^{c2*tmet}}{F0*Fd*C1}$$

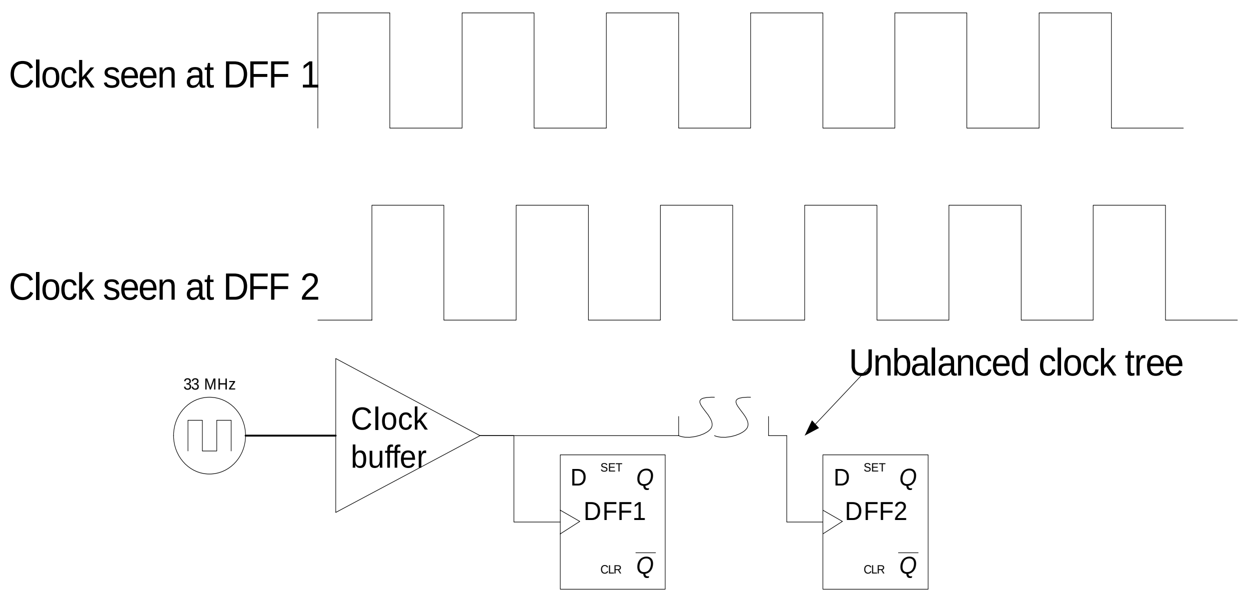
F0: Clock Frequency
Fd: incoming data frequency
C1: related to the window of susceptibility
C2: device specific constant

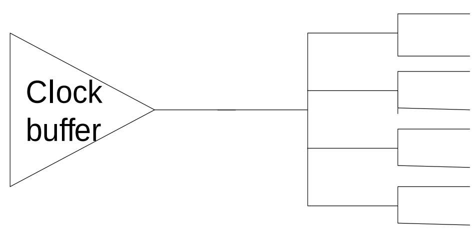
Metastability Filter



Clock Skew

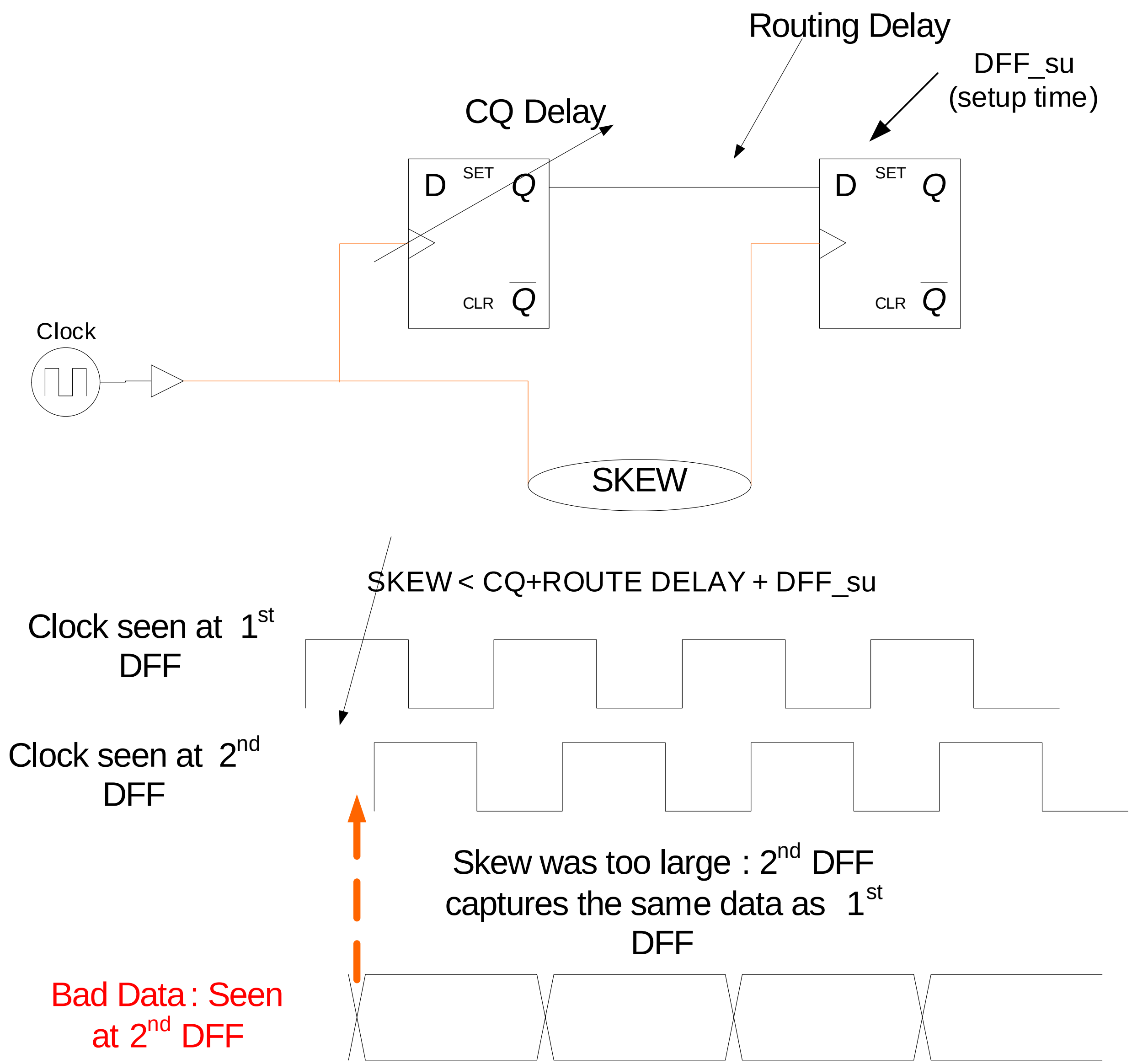
- **Skew: it is the measurement of the difference in clock arrival time seen at one DFF compared to another DFF**
- **Can cause a synchronous design to become asynchronous due to set-up and hold violations**
- **Clock tree must be balanced to avoid skew – beware of tree connections – should only be to a DFF clock pin (I.e. can not feed combinatorial logic).**
- **Designs that feed a clock that is not on a clock tree to DFFs will most likely contain unpredictable behavior.**
  - **Design Dependent**
  - **Very small number of DFFs (with combinatorial logic between them can get away with no clock tree)**





- Connections at leaves of tree must only be inputs to DFF clock pins … i.e. can not connect to an “and” gate - creates unbalanced tree.
- Connection routes must also be the same length

Clock Skew – Can cause Metastability and Unpredictability



Solution to Clock Skew Problem

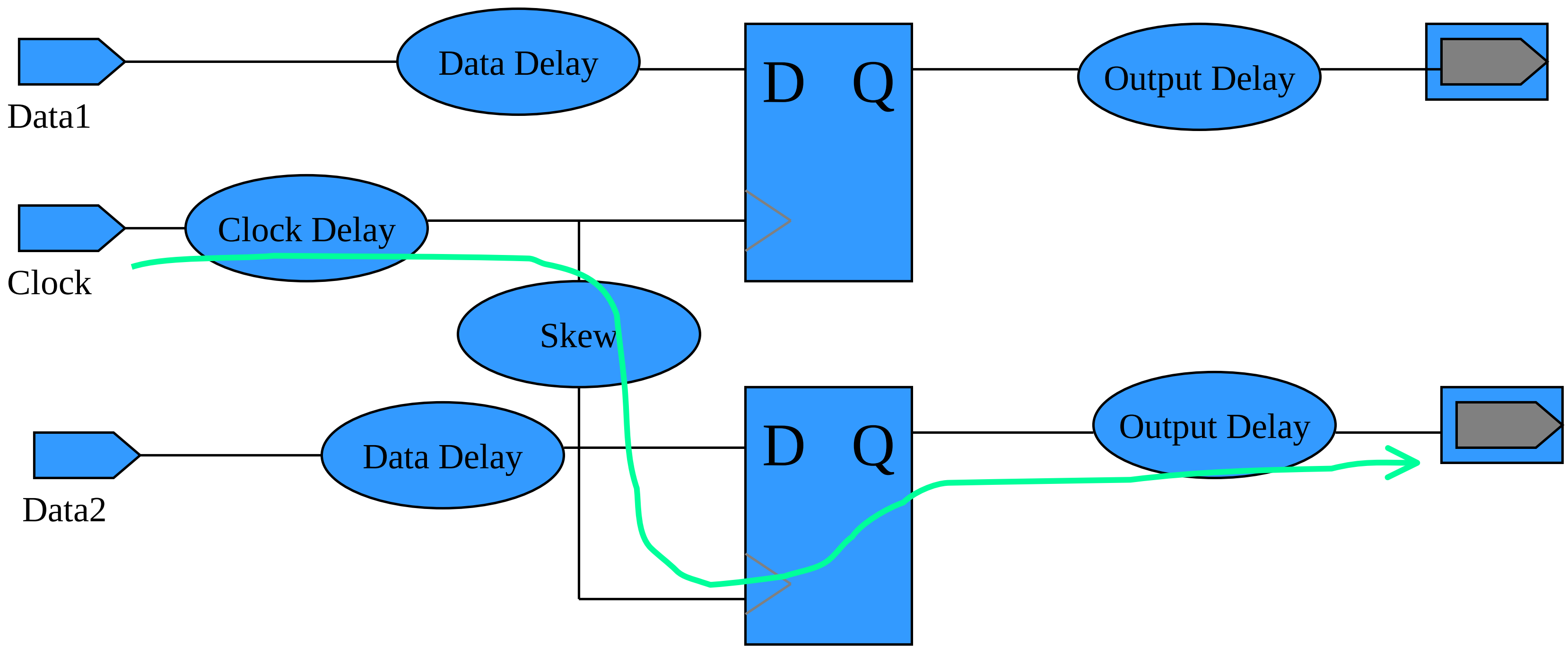
- **Use Clock trees with low skew distribution to the DFF's**
- **If not good enough… place a buffer (or some sort of delay) between the two DFF's**
- **Beware, clock trees contain points that are relative to each other (i.e. every point does not contain the same relative skew).**

Static Timing Analysis

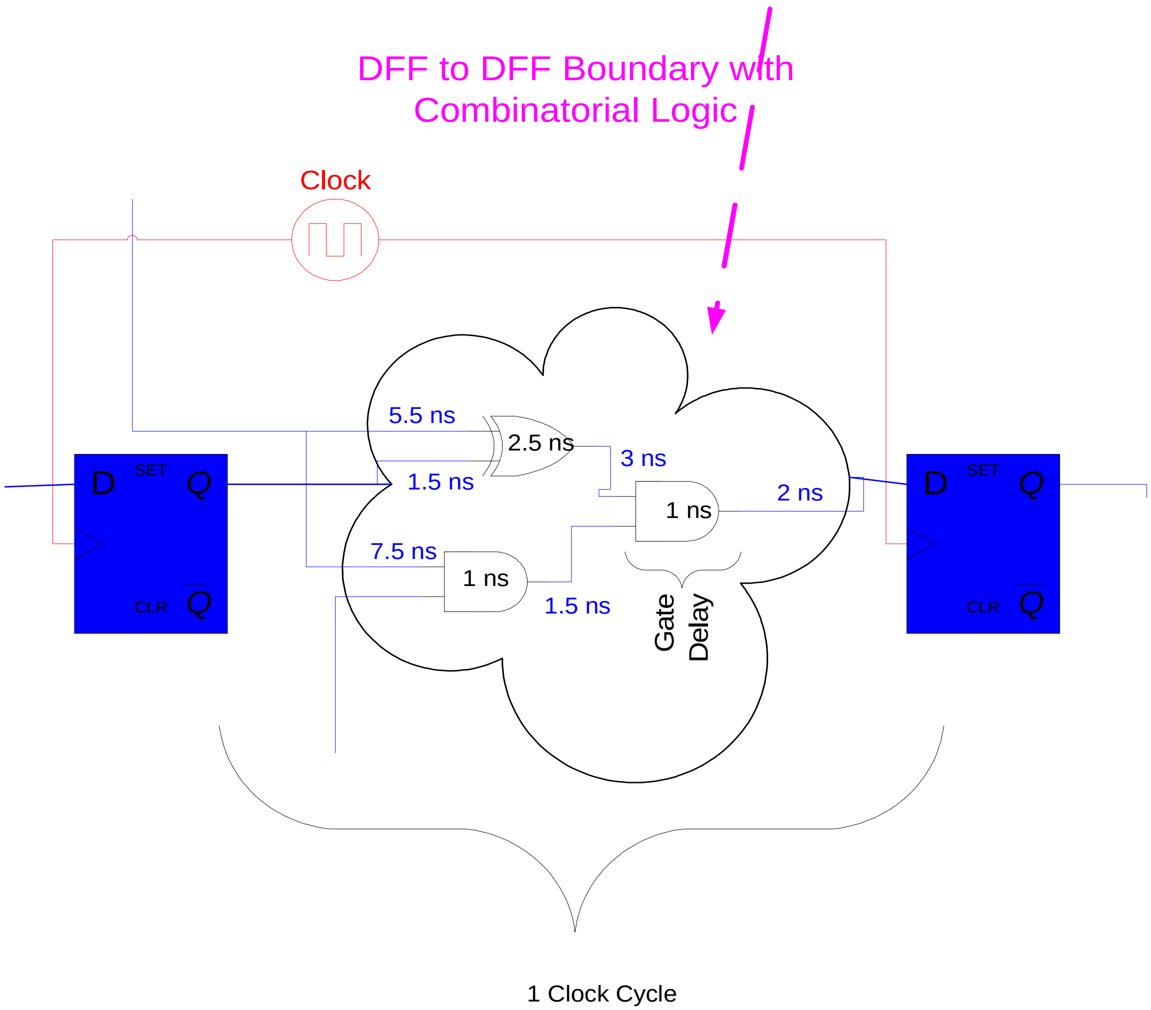
- **Concept: When will Data arrive at its associated DFF relative to the clock**
- **Every data path delay contained solely within each clock domain must be strictly deterministic**
- **Each path is defined as:**
  - **Input to DFF**
  - **DFF to DFF**
  - **Input to Output (highly not recommended design practice – inputs should pass through a DFF)**
  - **DFF to output**

Synchronous Clock Analysis –

Delays created by routing or buffer logic



Synchronous Timing Analysis



- Longest Path: 14 ns - Clock must have a period longer than 14 ns + overhead (temperature, voltage, and process variation)
- Shortest Path : 10ns

Static Timing Analysis

- **Delay of Data from its launch to its capture relative to the associated clock is calculated**
- **Data must be supplied with enough margin relative to the clock such that it will arrive at the DFF without violating DFF set-up and hold time.**
- **Best Case: Data source is derived from the same clock domain**
- **General Case (Inputs and multiple clock domain crossings): Data source must have relative (known) timing characteristics to the capture clock source. Otherwise, Data must be synchronized to the capturing domain**

Reset Circuitry

- **Within a reliable synchronous design, carefully thought-out reset circuitry is crucial.**
- **However, very often reset circuits are over-looked and the appropriate planning does not occur.**
- **Improper use of asynchronous resets has led to metastable (or unpredictable) states.**

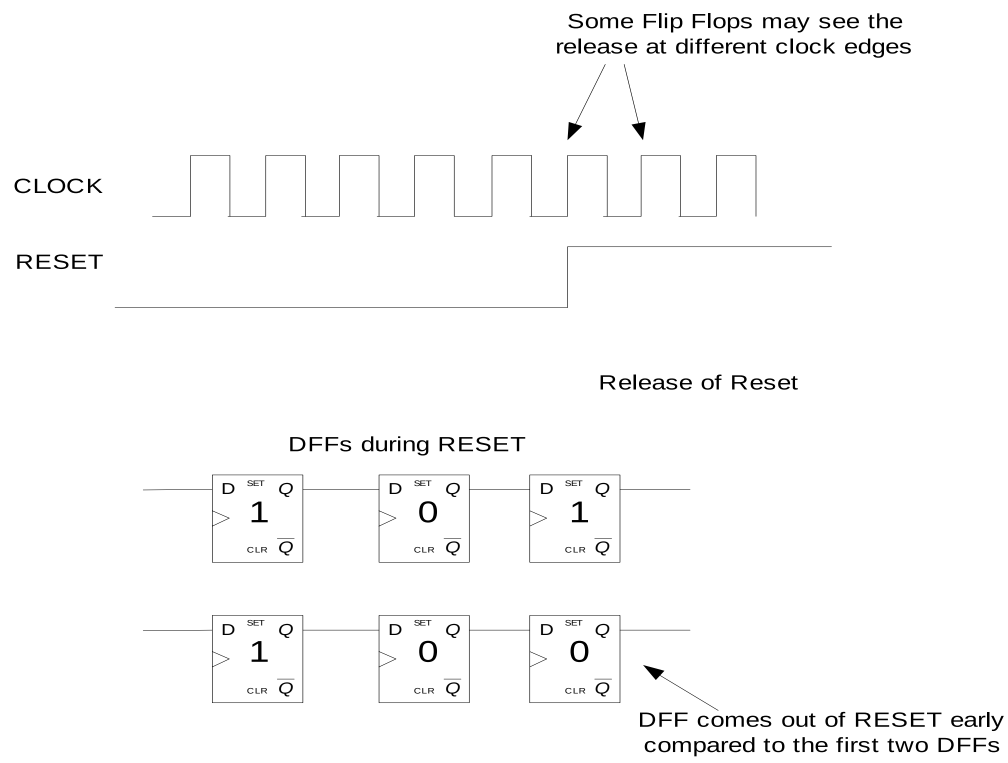
Asynchronous Resets

- **Designers will lean towards using an asynchronous reset within systems for several reasons.**
  - **Depending on the functionality of the FPGA/ASIC immediate response to a reset may be necessary.**
  - **FPGA/ASIC must respond to a reset pulse even during loss of a clock signal.**
  - **During Power Up/Down, the FPGA/ASIC outputs must be in a particular state in order to not damage other board components.**

Asynchronous Resets

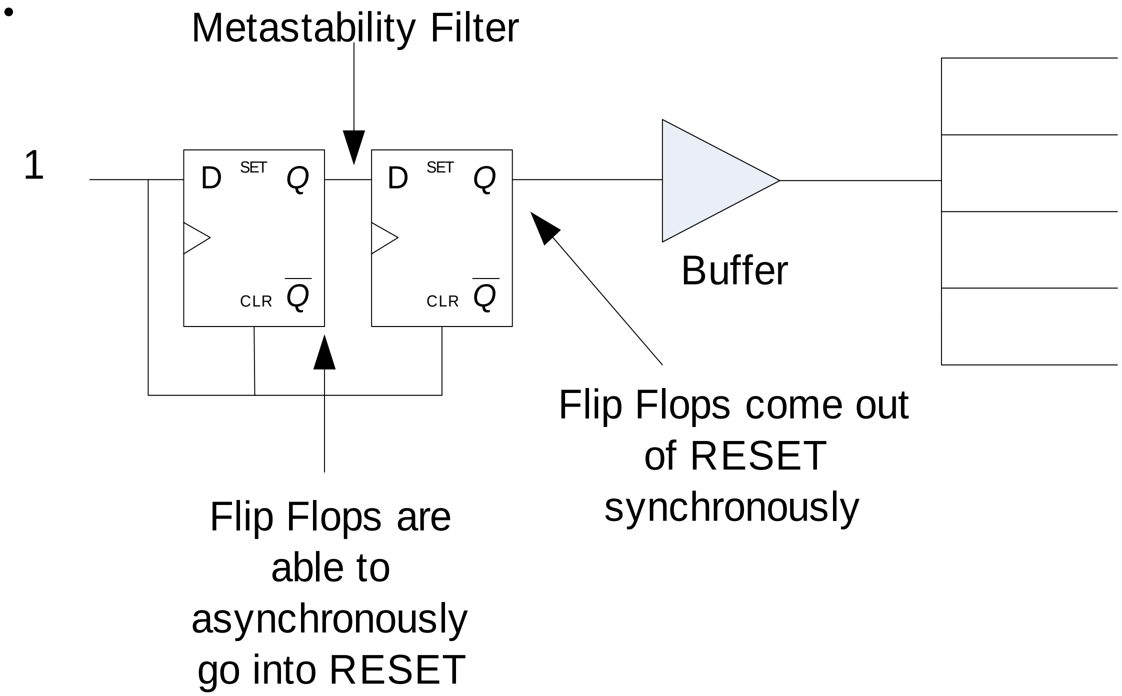
- **No problems exist as the system goes into reset due to the fact that all Flip Flops will eventually enter their reset state (i.e. a deterministic state space is reachable).**
- **The predicament occurs when the system comes out of the reset state.**
- **If an asynchronous reset signal is released near a clock edge, it is possible for the flip flops to be become metastable, or come out of reset relative to different clock edges.**

Asynchronous Resets



Asynchronous/Synchronous Resets

- **Solution: Use Asynchronous Assert Synchronous De-assert Reset circuit**
- **Such a design uses typical metastability filter theory. Diagram is Active Low.**



Asynchronous/Synchronous Resets

- **Upon the release of the reset signal, the first Flip Flop is not guaranteed to correctly catch the release of the reset pulse upon the nearest clock edge**
- **At most the next clock edge.**
- **It is also probable that the first Flip Flop will go metastable.**
- **The second Flip Flop is used to isolate the rest of the circuitry from any metastable oscillations that can occur when the reset is released near a clock edge (setup/hold time violation).**

Asynchronous/Synchronous Resets

- **Depending on the technology – ASIC vs. FPGA vs. Vendor, the designer may need to hand instantiate a high drive buffer.**
- **The output of the high drive buffer must be connected to the asynchronous reset terminal of each DFF in the system.**

Asynchronous/Synchronous Resets and Synthesis

- **There is a possibility that the synthesis tool can duplicate the second Flip Flop due to its large fan-out (not common in all technologies).**
- **The designer should check that there has not been any replication.**
- **The best approach is to have a library of modules (components) that includes a metastability filter. Within this module, place a do not replicate attribute on the second DFF to avoid incorrect realization.**

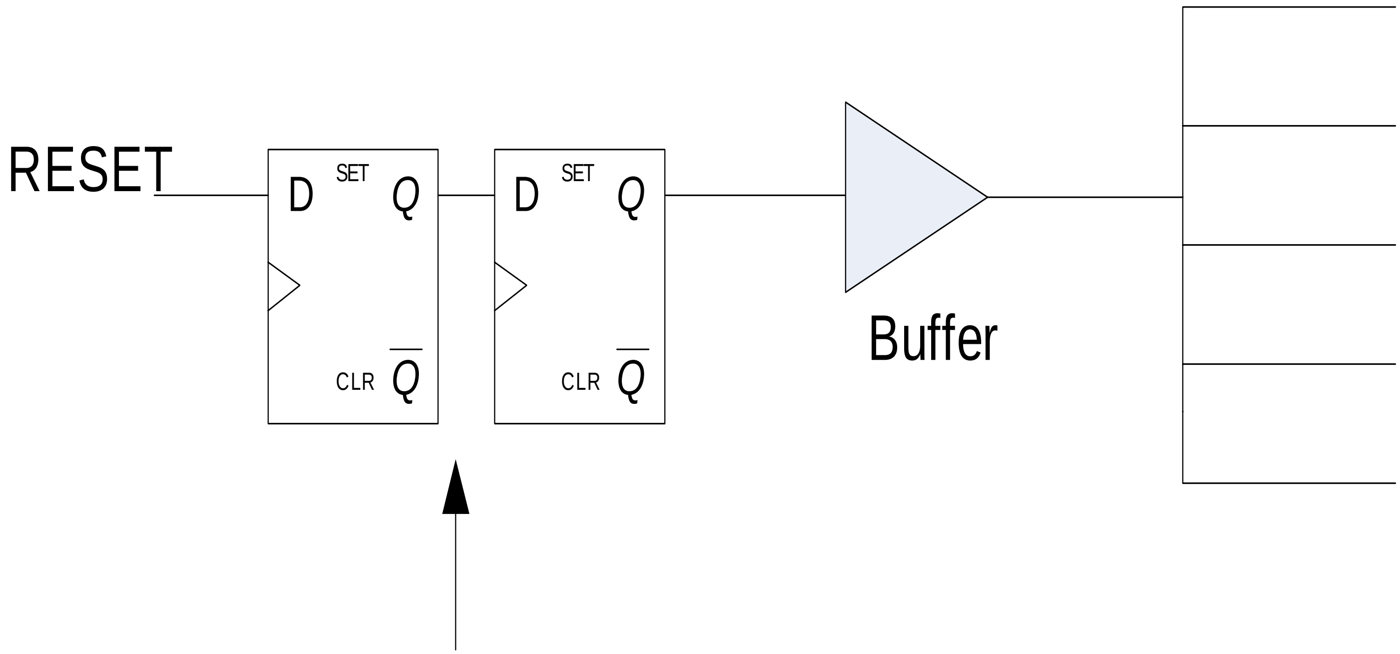
Asynchronous/Synchronous Resets Disadvantage

- **The system is very sensitive to glitches on the input reset signal and transients.**
- **The board must contain a low pass filter within the reset path before it reaches the FPGA/ASIC.**
- **I/O (or internal) Transients/Upsets are difficult to fix.**
  - **Additional filtering or mitigation (internal to FPGA) will always have at least one single point of failure and may not reduce the upset cross section.**

Synchronous Resets

- **Purely synchronous resets are very popular within the commercial industry.**
- **It is highly recommended to implement mixed asynchronous/synchronous reset circuitry for space applications**
- **However, if there are no sensitive components that the FPGA/ASIC is feeding, the synchronous approach is sufficient.**

Synchronous Resets



Metastability Filter do not connect the RESET signal to the Asynchronous DFF Reset terminals

Synchronous Resets
Advantages

- **The following are advantages of synchronous reset implementations:**
  - **The reset can predictably reach all of the DFF's in the circuit during the same clock cycle (as long as no timing violations exist).**
  - **The reset can be partitioned per module by adding an extra**
    DFF and thus reduce reset routing congestion.
  - **Extensive reset debounce circuitry can be implemented (using counters)**

Synchronous Resets Disadvantages

- **Must Have a Clock present**
- **Can potentially damage parts on the board during power up/down**
- **Can become hard to manage if it gets entangled within the data path**

Reset Circuitry Summary

- **Asynchronous Reset assert and Synchronous de-assert is the most optimal implementation**
- **When using Asynchronous assert and Synchronous de-assert, de-bounce circuitry is necessary**
- **Use Synchronous Resets if partitioning (due to critical timing) is necessary**
- **Careful system level consideration must be performed**

Synchronous Design within an Asynchronous Radiation Environment

- **Synchronous Design Theory depends on deterministic behavior**
- **Single Event Upsets (SEUs) and Single Event Transients (SET's) are considered asynchronous events**
- **Metastability and non-determinism is inevitable.**
- **Design for Hardness Methodologies have been developed to reduce the upset rate**

Three Common Mitigation Techniques

- **Localized Triple Mode Redundancy (TMR)**
- **Distributed TMR**
- **Localized DICE**

Localized vs. Distributed Mitigation

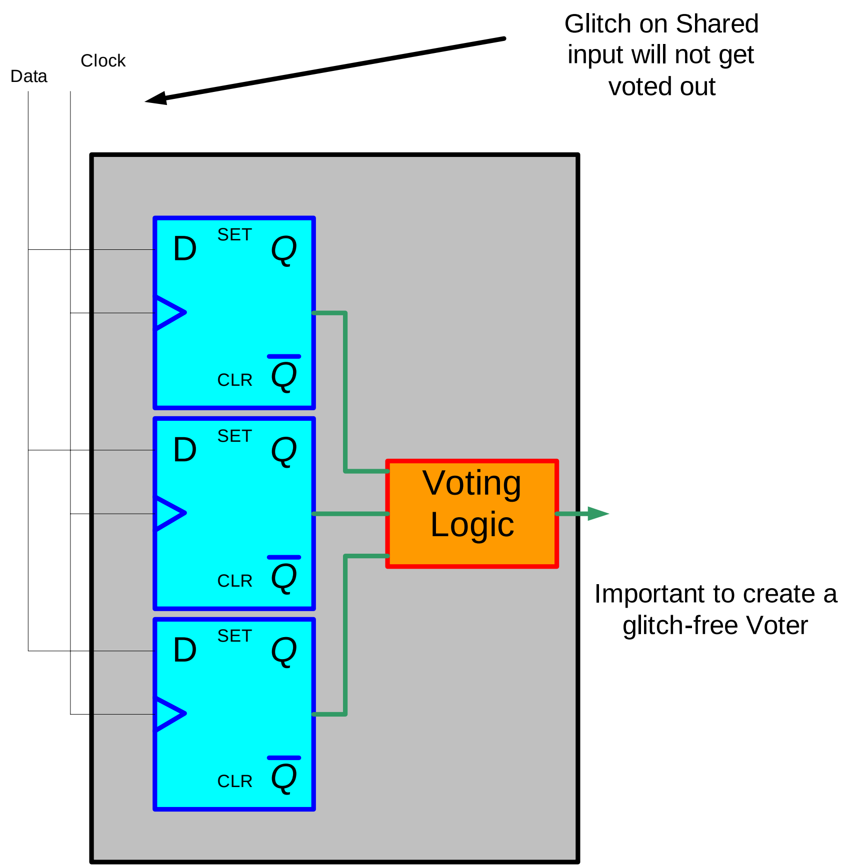
Localized

- **Mitigation occurs within one clock domain and at each DFF**
- **Data, clock, reset, and enable are shared inputs to the DFF**

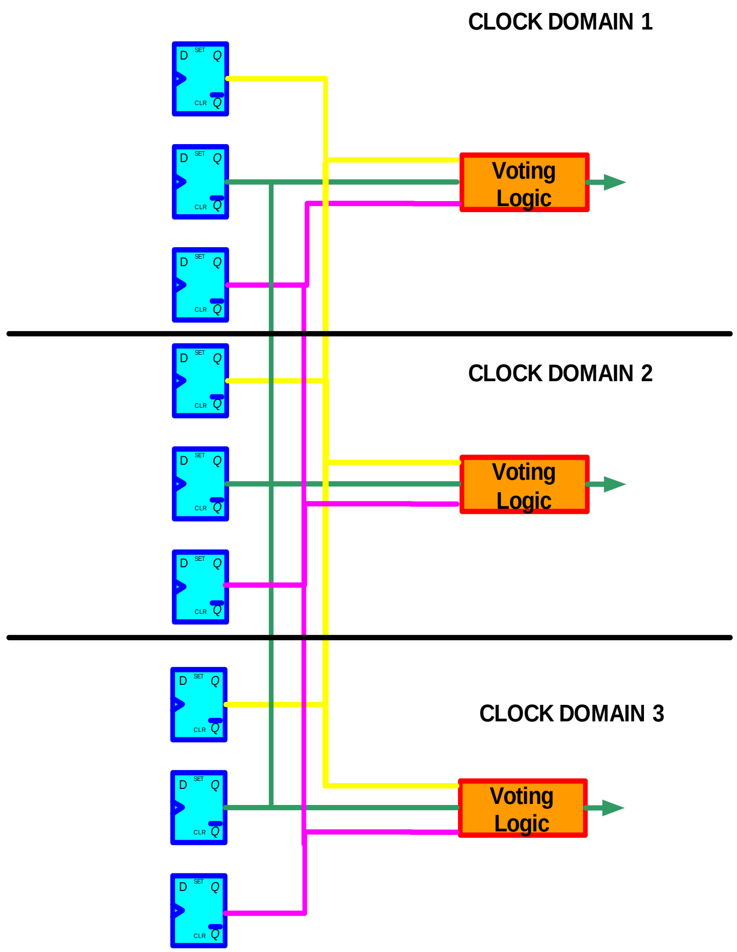
Fully Distributed

- **The entire design is tripled (I/O, clock domains, and logic)**
- **Mitigation occurs at each DFF across clock domains**
- **No shared DFF input lines**
- **Area extensive**
- **Power hungry**

Localized TMR Example: One DFF Cell



Distributed TMR Example



Antifuse FPGA Devices

- **Currently the most widely employed FPGA Devices within space applications**
- **Configuration is hardened due to fuse based technology**
- **Localized Mitigation (TMR or DICE) is employed**
- **Clock and Reset lines are hardened**

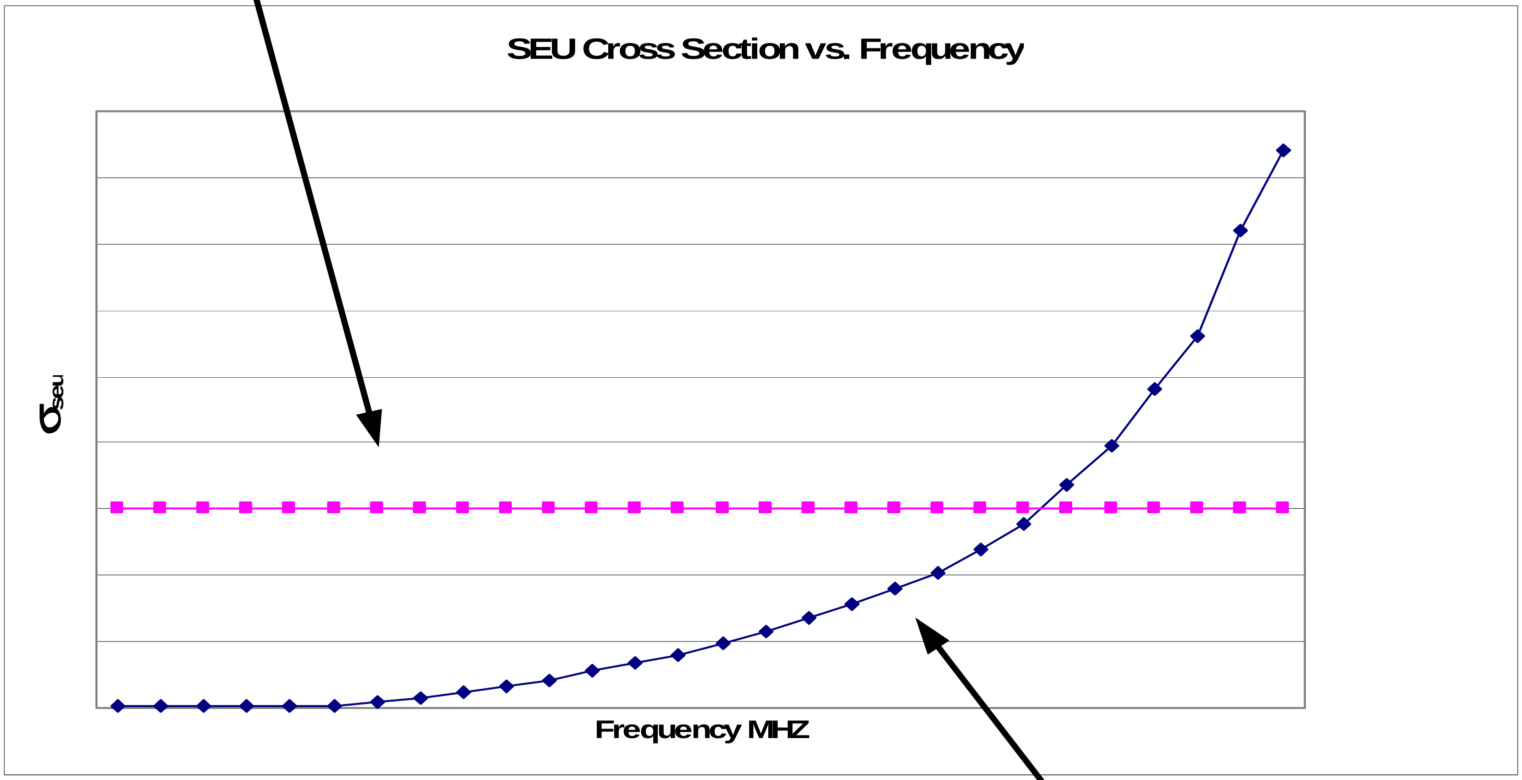
SEUs and The Antifuse FPGA Design Community

- **Design strategies have been built off the sole fact that SEUs are created from DFF radiation hits.**
- **Current Proposed Design Methodology:**
  - **Use less DFFs**
  - **Do not replicate DFF logic due to high fan-out**
  - **Use binary (or Gray) encoding state machines vs. one-hot**

- **However, as frequency increases, SET generation and encapsulation actually dominate the error cross section.**

DFF Upsets

DFF Upsets due to SEUs



DFF Upsets due to Transients

Basis of a Design Methodology

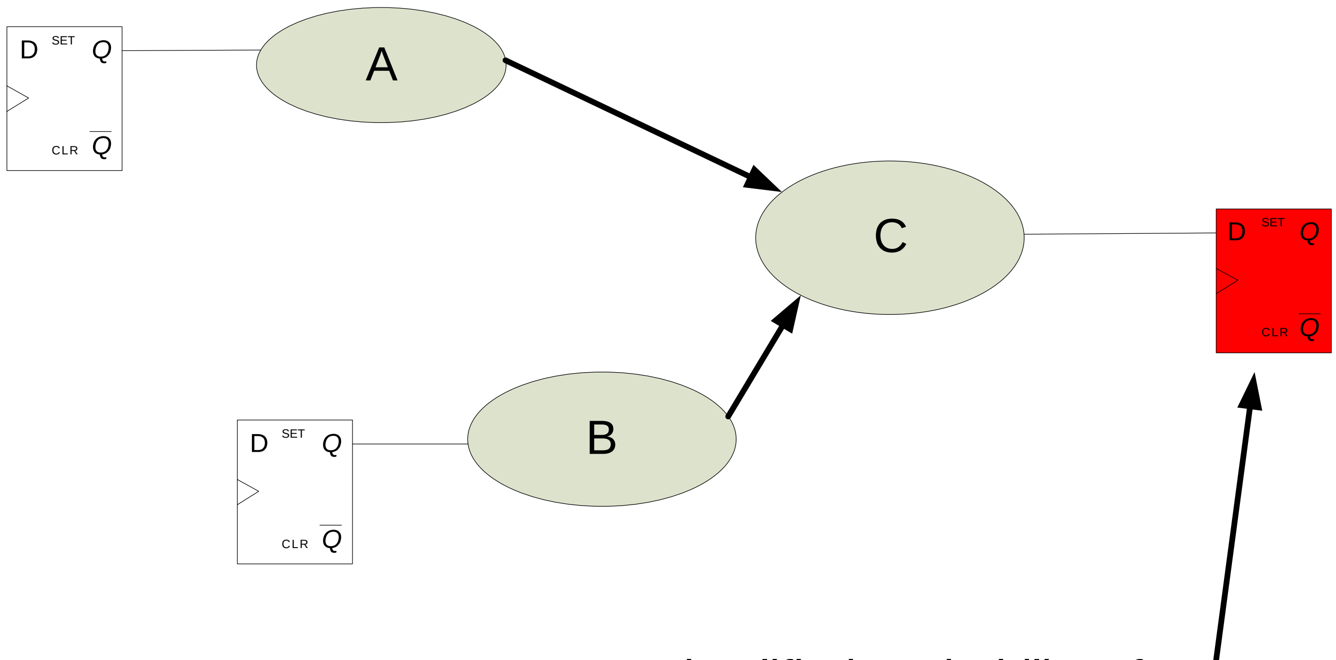
- **No mitigation is 100%**
- **Objective is to reduce the probability of SEU generation**
- **Current Hardened FPGA devices suggest that DFF nodes should have a low SEU cross section – upsets mostly due to SET's**

Design Methodology

The designer should:

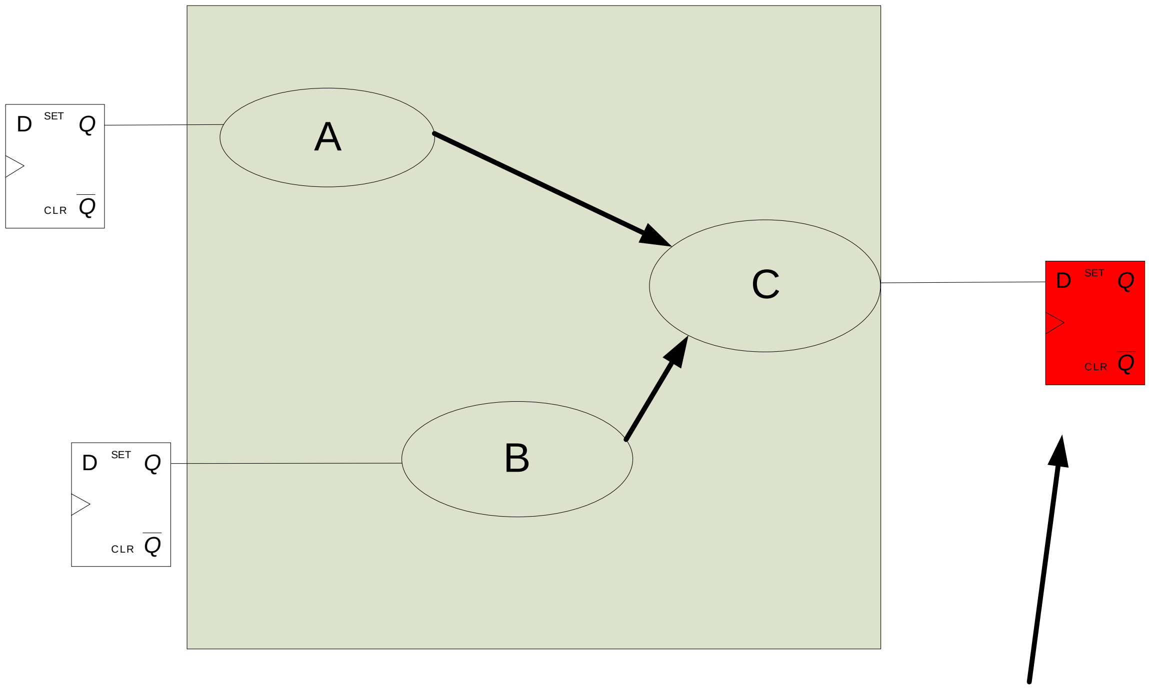
- Reduce the amount of combinatorial logic or
- Strategically add redundant logic
- Use Hardened Clock trees for clock Distribution
- Use Hardened Clock trees for reset Distribution
- Simplify logic and use lower fan-out solutions (i.e. one-hot state machines vs. binary)

Probability of Upset due to Capturing a SET



Simplified Probability of Upset :
P(A) + P(B) + P(C) + P(DFF catching SET)

Probability of Upset with Mitigation



Mitigated ABC function

Simplified Probability of Upset :
P(mitigation) + P(DFF catching SET)

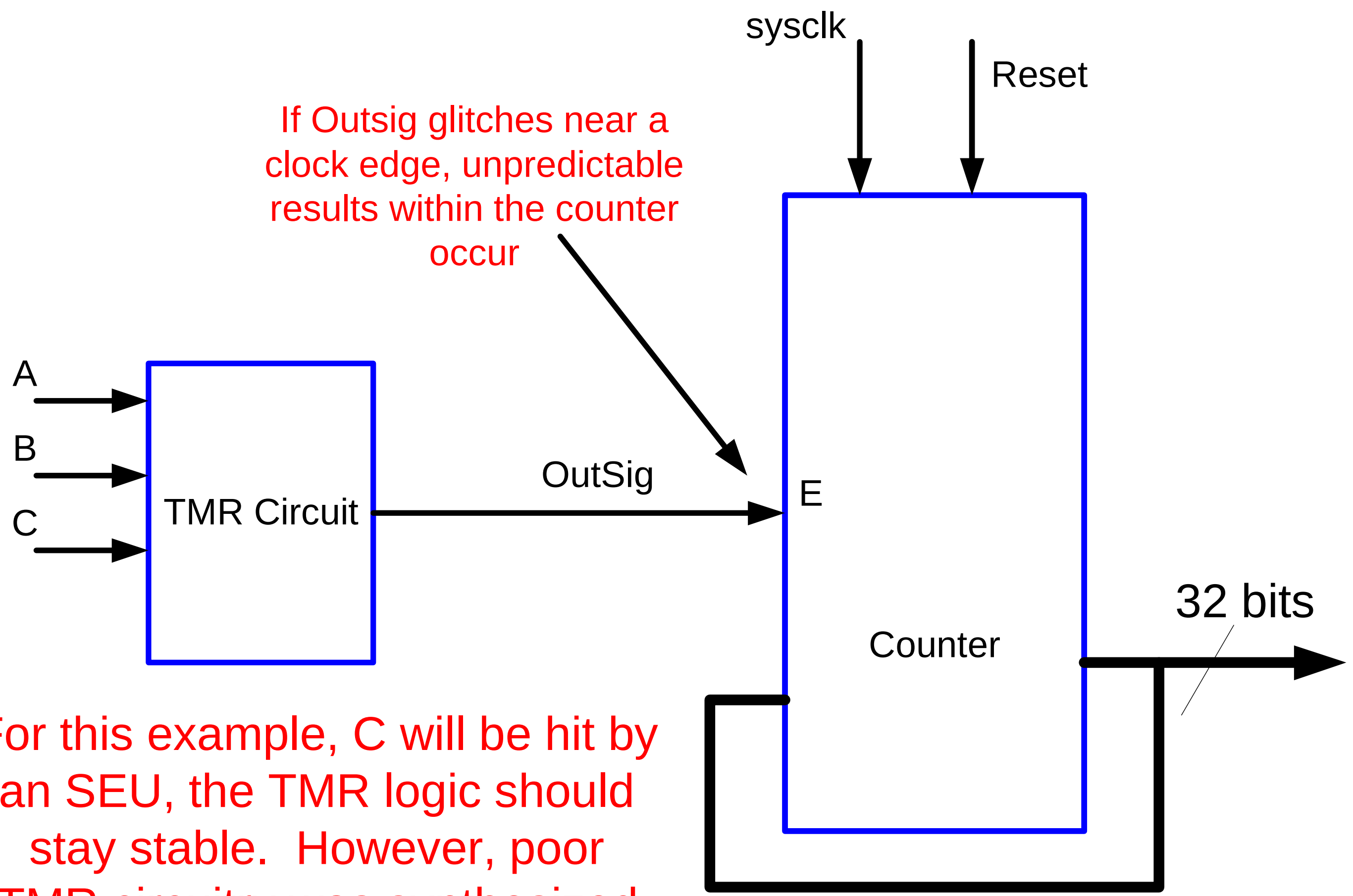
Probability of Upset with Mitigation

- **Although there is more combinatorial logic with mitigation insertion, the probability of upset is reduced**
- **Mitigation susceptibility:**
  - **Glithy mitigation: can add to cross section**
  - **Delay filtering: reduces functional cross section but has its own (overlap)**
  - **Increase in mitigation complexity can increase susceptibility**
  - **Sensitivity of last transistor in mitigation circuit (single point of failure – very low cross section)**

Mitigation and Your Synthesis Tool

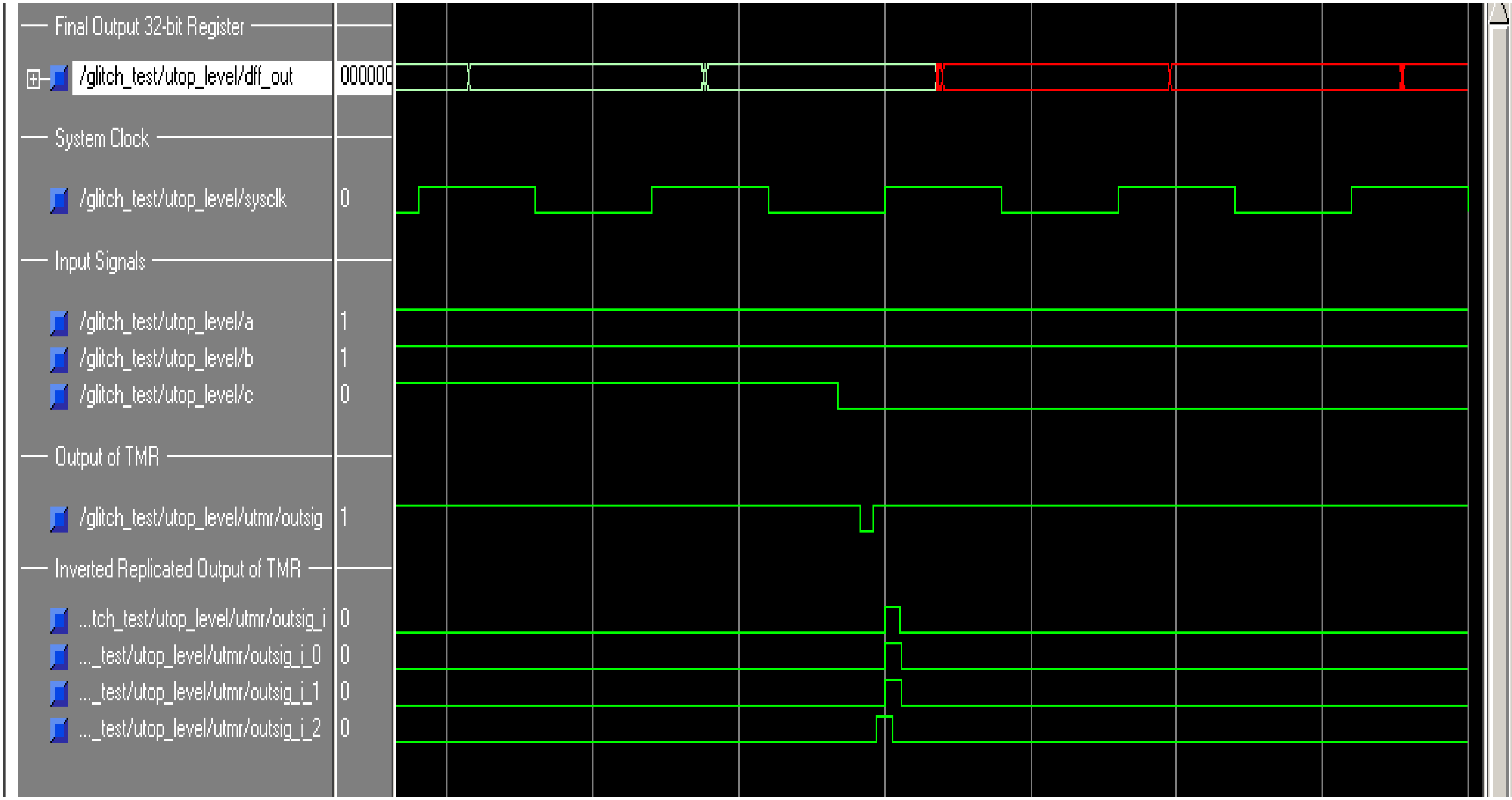
- **The objective of the synthesis tool is to reduce area**
- **The synthesis optimization algorithm will want to remove redundancy to reduce area**
  - **Don't touch directives may be necessary**
- **Designer must look at the schematic produced by the synthesis tool to verify that the mitigation has been correctly produced**

Glitches in TMR Circuitry: Example



For this example, C will be hit by an SEU, the TMR logic should stay stable. However, poor TMR circuitry was synthesized and a glitch occurs on OutSig

Glitchy TMR Circuitry Continued TMR Reaches DFFs at Separate Times



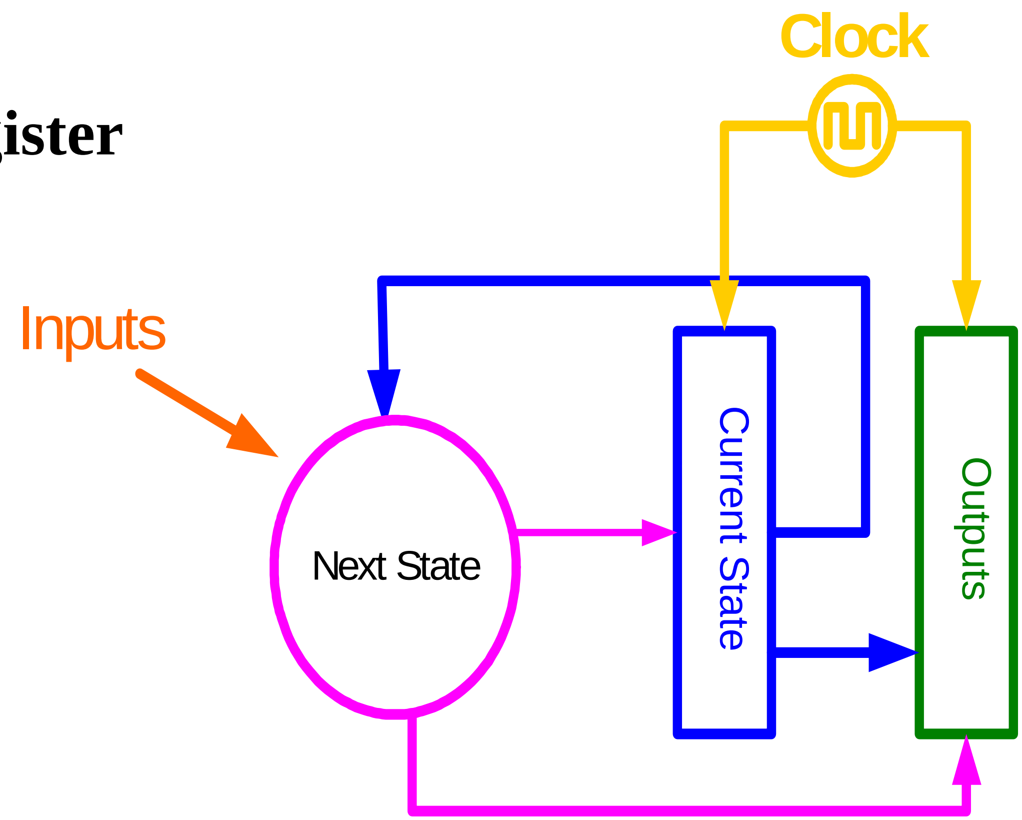
State Machine Example

Synchronous State Machines

- **A Finite State Machine (FSM) is designed to deterministically transition through a pattern of defined states**
- **A synchronous FSM utilizes flip-flops to hold its currents state, transitions according to a clock edge and only accepts inputs that have been synchronized to the same clock**
- **Generally FSMs are utilized as control mechanisms**
- **Concern/Challenge:**
  - **If an SEU occurs within a FSM, the entire system can lock up into an unreachable state: SEFI!!!**

Synchronous State Machines

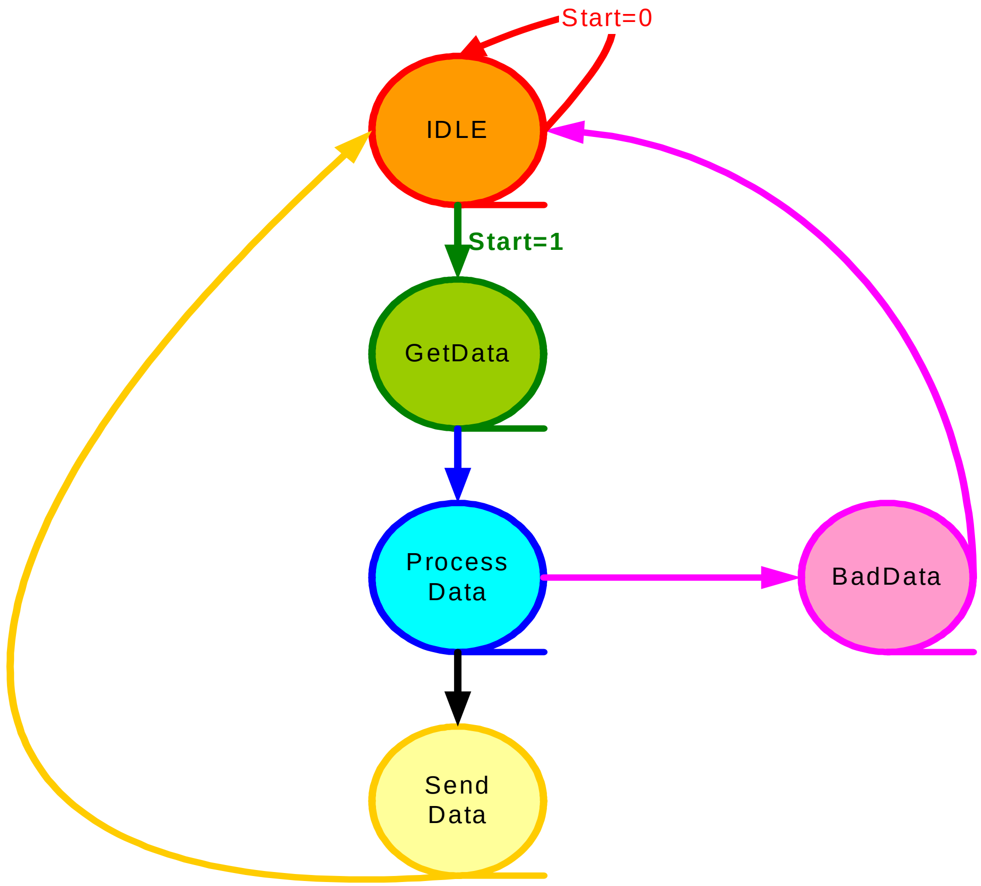
- The structure consists of four major parts:
  - Inputs
  - Current State Register
  - Next State Logic
  - Output logic



Encoding Schemes

Example:
Five states need to be mapped.
There is only one input: Start

- **Each state of a FSM must be mapped into some type of encoding (pattern of bits)**
- **Once the state is mapped, it is then considered a defined (legal) state**
- **Unmapped bit patterns are illegal states**

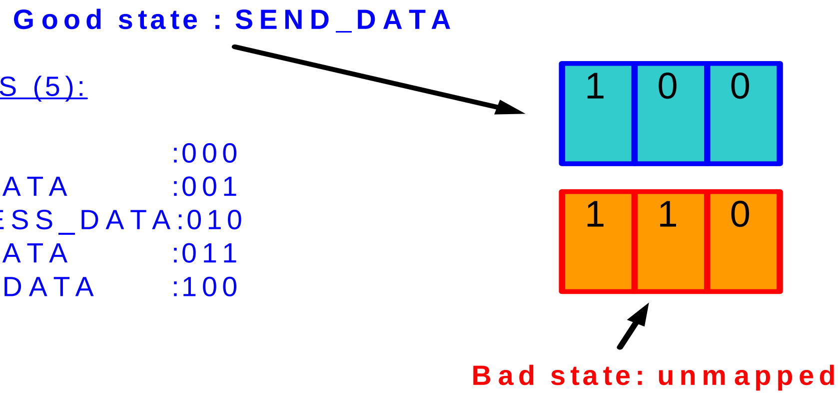


Encoding Schemes

Registers: binary encoding

STATES (5):

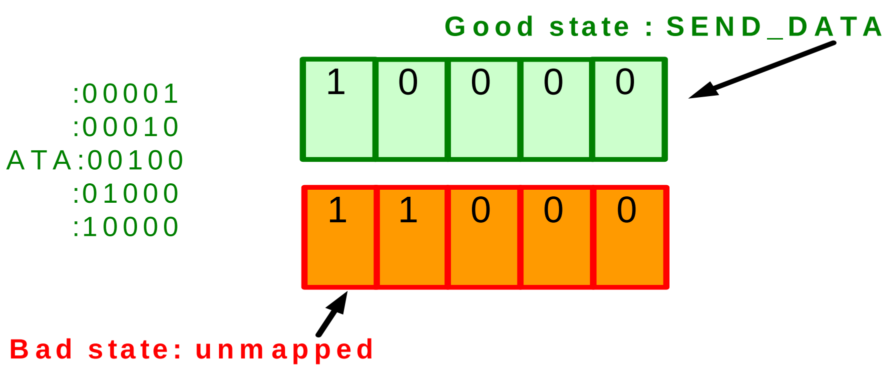
IDLE	:000
GET_DATA	:001
PROCESS_DATA	:010
BAD_DATA	:011
SEND_DATA	:100



Registers: One Hot encoding

STATES (5):

IDLE	:00001
GET_DATA	:00010
PROCESS_DATA	:00100
BAD_DATA	:01000
SEND_DATA	:10000



Safe State Machines???

- **A "Safe" State Machine has been defined as one that:**
  - **Has a set of defined states**
  - **Can deterministically jump to a defined state if an illegal state has been reached (due to a SEU).**
- **Synthesis tools offer a "Safe" option (demand from the Aerospace industry):**

  TYPE states IS (IDLE, GET_DATA, PROCESS_DATA, SEND_DATA, BAD_DATA);

  SIGNAL current_state, next_state : states;

  attribute SAFE_FSM: Boolean;

  attribute SAFE_FSM of states: type is true;
- **However…Designers Beware!!!!!!!**
  - **Synthesis Tools Safe option is not deterministic if an SEU occurs near a clock edge!!!!!**

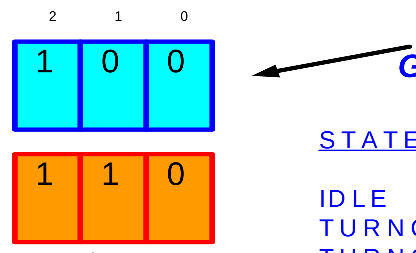
Binary Encoding: How Safe is the “Safe” Attribute?

- **If a Binary encoded FSM flips into an illegal (unmapped) state, the safe option will return the FSM into a known state. However, this is most safely implemented by use of a error detection and FPGA reset.**
- **If a Binary encoded FSM flips into a good state, this error will go undetected.**
  - **If the FSM is controlling a critical output, this phenomena can be very detrimental!**
  - **How safe is this?**

Safe State Machines???

State(1) Flips upon SEU:

Using the "Safe" attribute will transition the user to a specified legal state upon an SEU



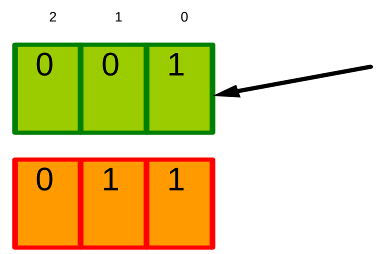
Good State

STATES (5):

IDLE :000
TURNON_A :001
TURNOFF_A :010
TURNON_B :011
TURNOFF_B :100

Illegal State: unmapped

Using the "Safe" attribute will not detect the SEU: This could cause detrimental behavior



Good State: TURNON_A

legal State: TURNON_B

One-Hot vs. Binary

- **Some suggest that Binary is "safer" than One-Hot**
  - **Based on the idea that One-Hot requires more DFFs to implement a FSM thus has a higher probability of incurring an error**
- **This theory does not apply to Antifuse hardened FPGA's working at high frequencies (> 10 MHZ)**
  - **Most of the community now understands that although One-Hot requires more registers, it has the built-in detection that is necessary for safe design**
  - **Binary encoding can lead to a very "un-safe" design**

Proposed SEU Error Detection: One-Hot

- **One-Hot requires only one bit be active high per clock period**
- **If more than one bit is turned on, then an error will be detected.**
- **Combinational XNOR over the FSM bits is sufficient for SEU detection**
- **Error Detection can be used to deal with the upset (i.e. reset FPGA)**

Summary

- **Synchronous Design Techniques should be followed to create reliable designs**
- **Think ahead – overall system consideration**
- **Understand the targeted technology - mitigation, hardened routes, areas of SEU susceptibility.**
- **Add extra mitigation where necessary (control the synthesis tool while doing so)**
- **Use hardened Clock networks for the low skew clock tree and resets**