


Jim Hoff (jimhoff@fnal.gov)



Digital Design in High Energy Physics

Outline

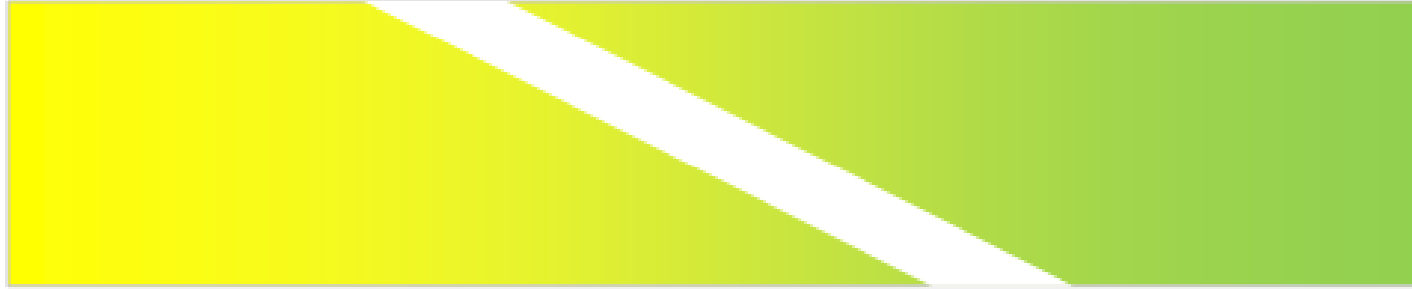
- Introduction
 - Analog vs. Digital
 - Three Pillars of Digital Design
- Digital Design in Extreme Environments
 - Taxonomy of Radiation Effects
 - Total Ionizing Dose
 - Single Event Effects
- Design (RTL)
- Verification (VRF)
- Place-and-Route (PNR)



Introduction

Analog

Digital



**For HEP Front-End Chips:
Amplification – Shaping – Discrimination - Algorithms**

In this worldview, the dividing line between analog and digital depends on what transistor regions you care about - Digital really only cares about Saturation and Cutoff and Analog is forced to care about Linear as well.

**Precision vs. Repetition:
Timing Circuits**

Timing circuits, do digital things, but they do so with such precision that it is hard to lump them in with the rest of the digital world where very general statements can be made about transistor sizes and, broadly speaking, the same digital circuit can, and is, used again and again.

Analog

Digital



- **What?** Amplifiers, Bias References
- **How?** Full-custom design from the transistor level

- **What?** Phase Locked Loops (PLLs), Delay locked Loops (DLLs), Data Converters, Drivers, High speed serializers
- **How?** A mixture of both Full-custom and Semi-custom design styles

- **What?** Serializers, Deserializers, Data Concentrators, Algorithms, Zero Suppression, Readout Architectures
- **How?** Semi-custom design from standard cell libraries that are part of a Process Design Kit (PDK)

Analog

Digital

When I started in this field, unquestionably, Analog ruled our world. Digital was told how much (read: how little) room it could use. Digital was told what metal layers it would be allowed to use.

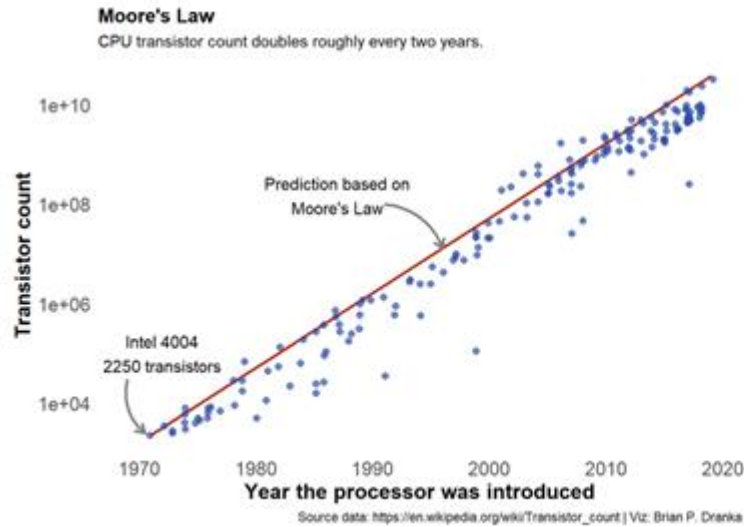
In the years since I started, the percentage of transistors and of chip area dedicated to pure digital has grown dramatically and it shows no signs of stopping.

- **W**
Re
- **H**
fro

gorithms,
res
n design
libraries
rocess

Analog

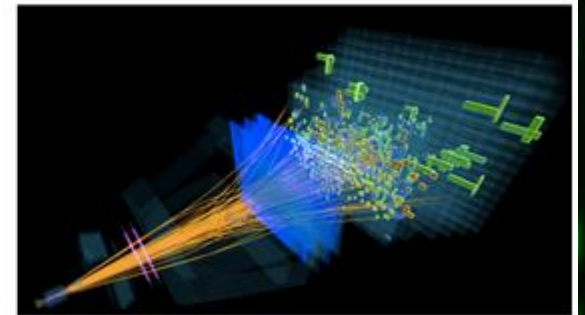
Digital



Data Explosion

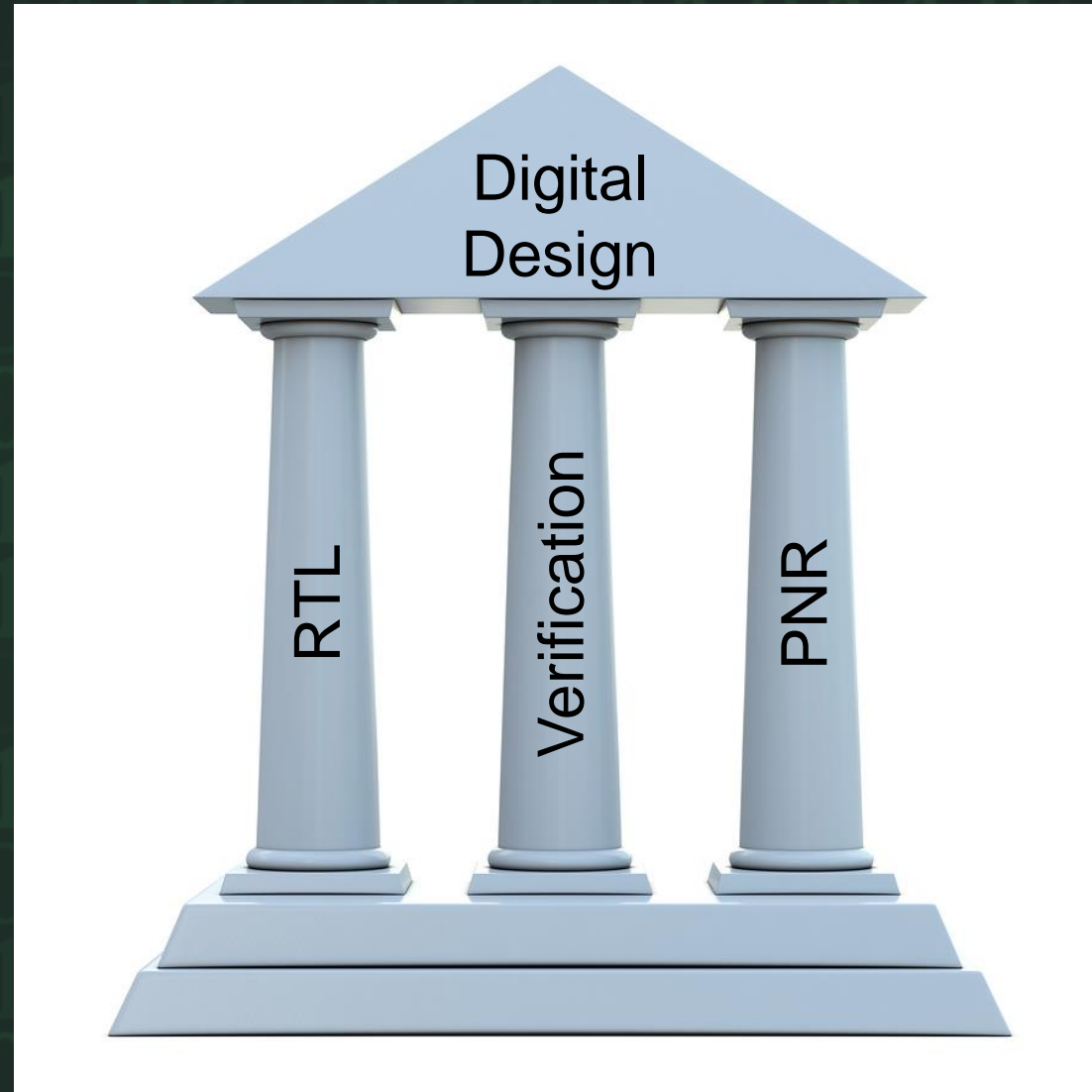
Reduced Feature Size

Improved Simulation



▸ The Three Pillars of Digital Design

RTL
Verification
Place-and-Route



The Three Pillars of Digital Design

RTL

Register Transfer Level

Verilog or SystemVerilog

Design

This is what you learned in your University classes. This is usually what everyone thinks of when they think “digital design”



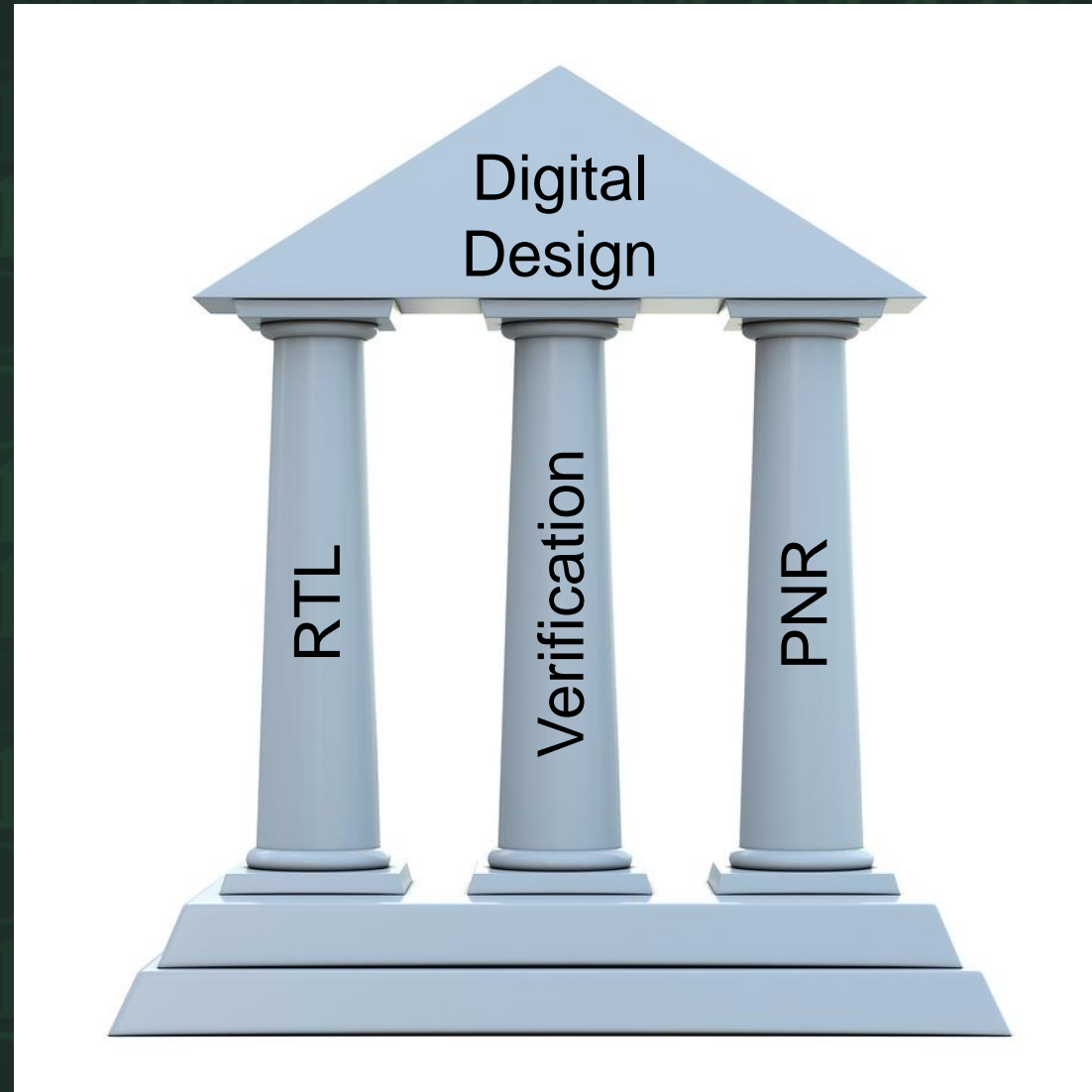
The Three Pillars of Digital Design

Verification

It is *NOT* just simulation

It is *NOT* just a *LOT* of simulations

It is a systematic, mathematical approach to design testing to cover as many aspects of a design as a group of engineers can possibly think of and (in certain versions) uses constrained randomization to (hopefully) reach corners the designers did not think of.



The Three Pillars of Digital Design

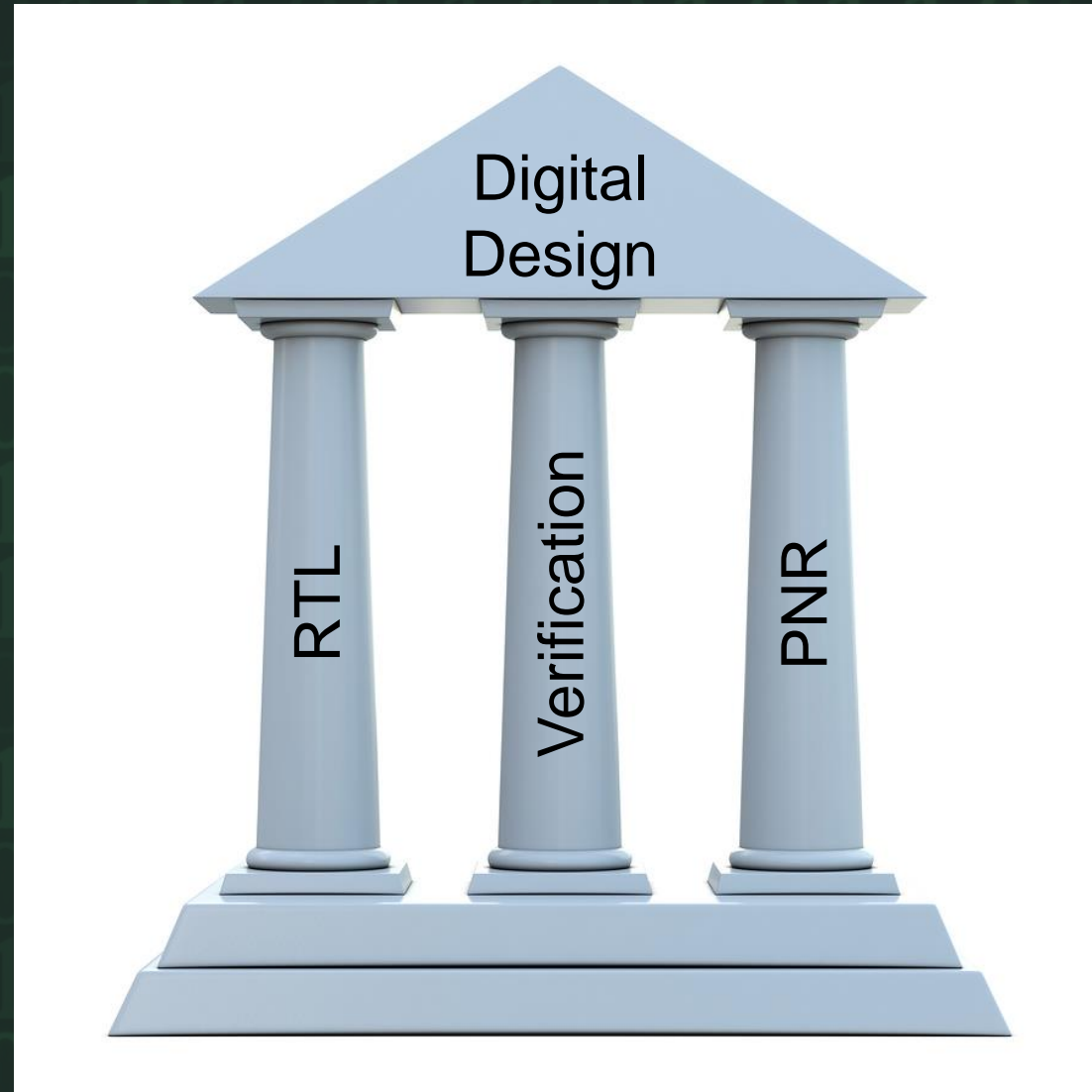
PNR

Place-And-Route

Synthesis

Digital Layout

It is every bit as complex and comprehensive as full-custom layout in Analog Design, but it is completely different



RTL-VRF-PNR Interaction

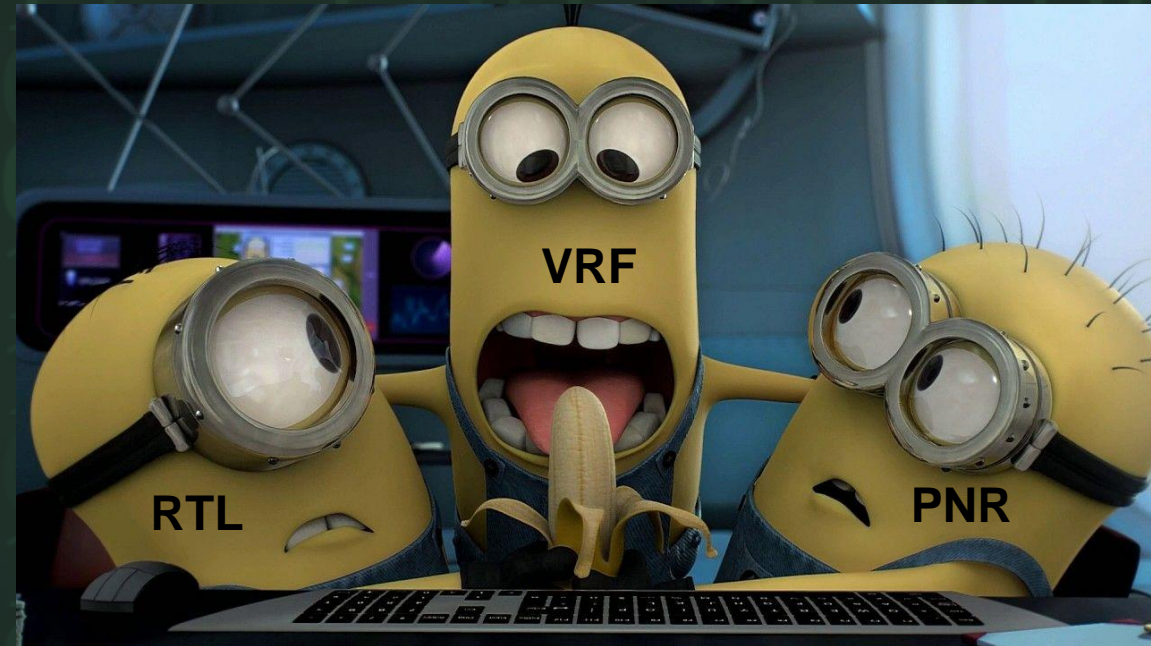
- Obviously, RTL must start first because it sets the functional goals of a projects.

...BUT...

- PNR must start first because it must set a realistic floorplan which, in turn, defines for RTL how sub-modules will talk to one another

...BUT...

- Verification must start first because it sets the verification plan which determines the different testbenches that will be developed and these testbenches will allow the RTL to be tested for their functionality.



RTL-VRF-PNR Interaction

- Verification needs RTL to finish first because Verification cannot possibly finish until the RTL is set in stone

...BUT...

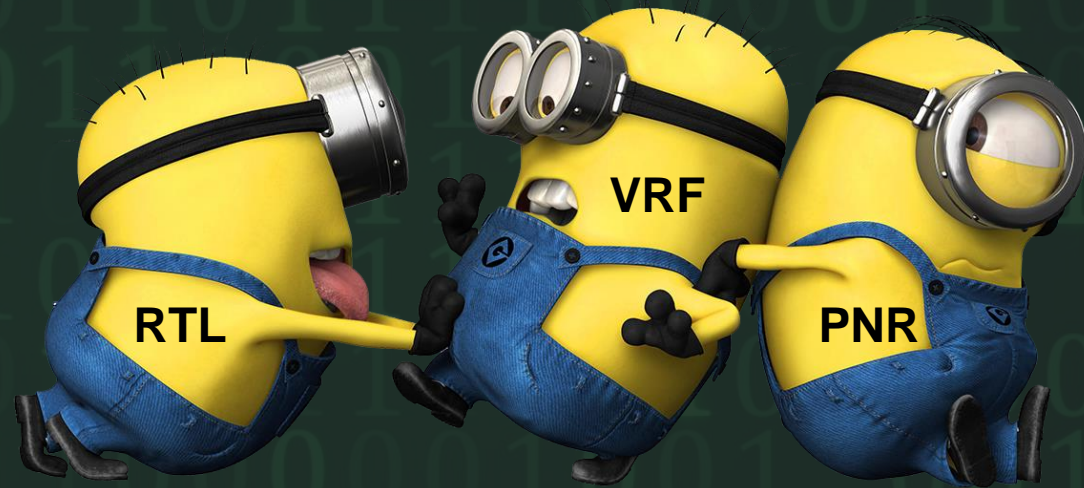
- Verification is finding problems with the RTL, so if Verification is doing its job, the RTL is changing at the same time

...BUT...

- PNR needs RTL and Verification to stop messing with its scripts

...BUT...

- Verification needs PNR to be done first because it needs netlists and SDF files to do post-layout simulations which will, of course, further alter the RTL and then the PNR



RTL-VRF-PNR Interaction

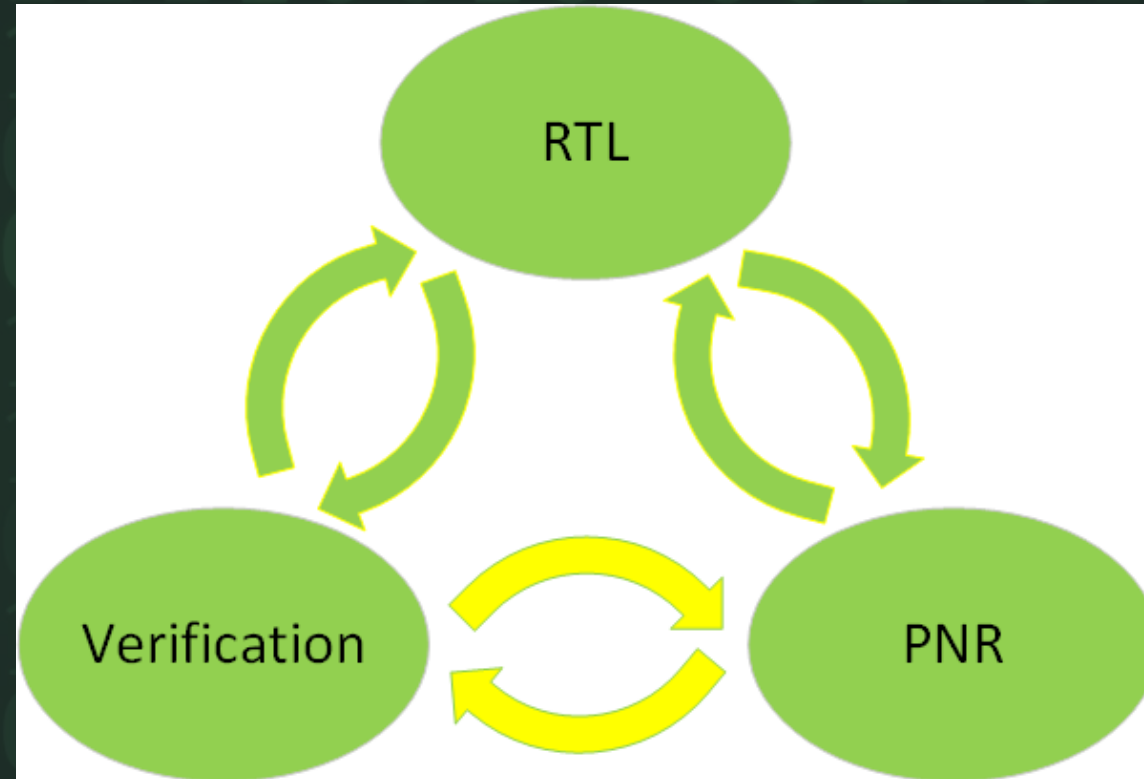
And, of course,
EVERYBODY
needs the physicists
to stop changing the
damn
specifications!!!



Constant Feedback and Successive Approximation

In the end, what we have is a system of feedback and successive approximation.

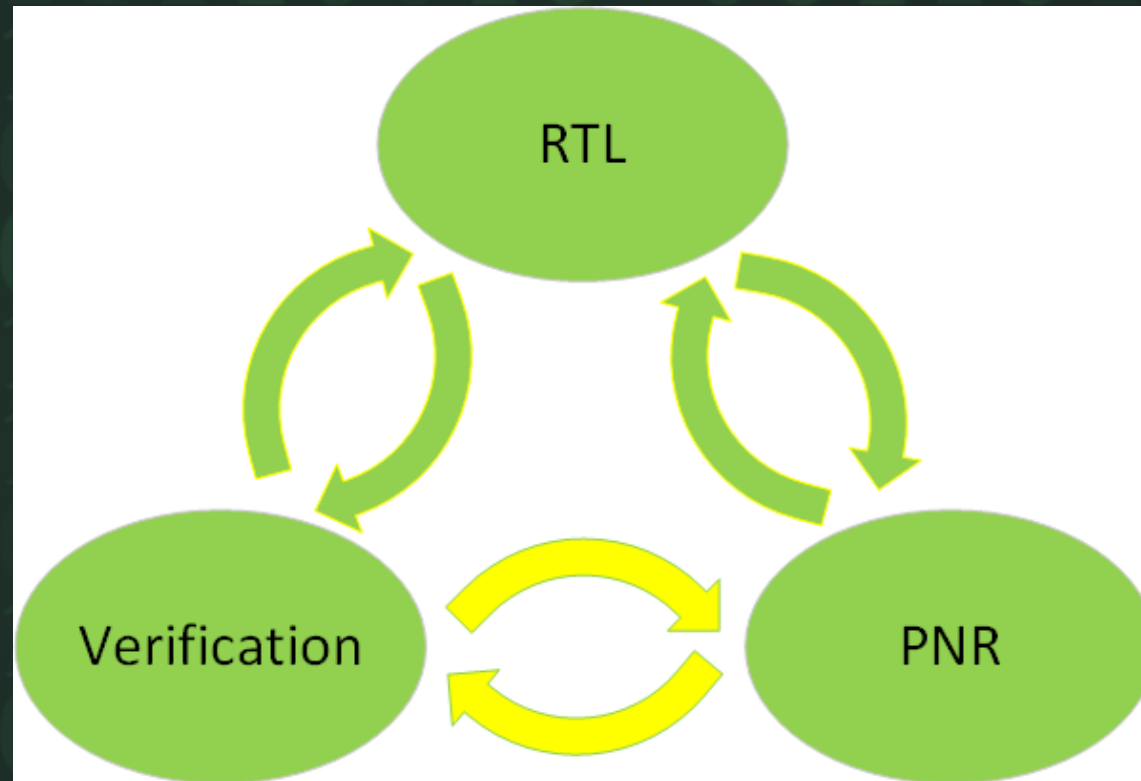
1. In a perfect world, RTL, VRF, and PNR are NOT THE SAME PEOPLE.
2. In a perfect world, everyone starts at approximately the same time. At the very least everyone is aware of the actions of their digital teammates from the start.
3. RTL and VRF are in constant feedback. RTL feeds VRF verbal and written **SPECIFICATIONS** of functionality. VRF feeds RTL a constant stream of bugs. Both must agree on a hierarchy and both must be involved in the verification plan.



Constant Feedback and Successive Approximation

In the end, what we have is a system of feedback and successive approximation.

4. RTL and PNR are also in constant feedback. RTL hierarchy is logical, but PNR hierarchy is physical. It is a virtual certainty that RTL's logic will be forced to give way to PNR's practical limitations. Later in the design process, RTL's logic will again have to give way to PNR's timing limitations.
5. The feedback from VRF to PNR is different. VRF cannot possibly finish its job until it has a final netlist and SDF (delay information) from PNR and PNR's modifications to hierarchy often force changes to VRF's hierarchy. Honestly, VRF is largely a pain in PNR's neck as it pushes for netlists and SDF files.





Digital Design in Extreme
Environments

TID Bibliography to get you started

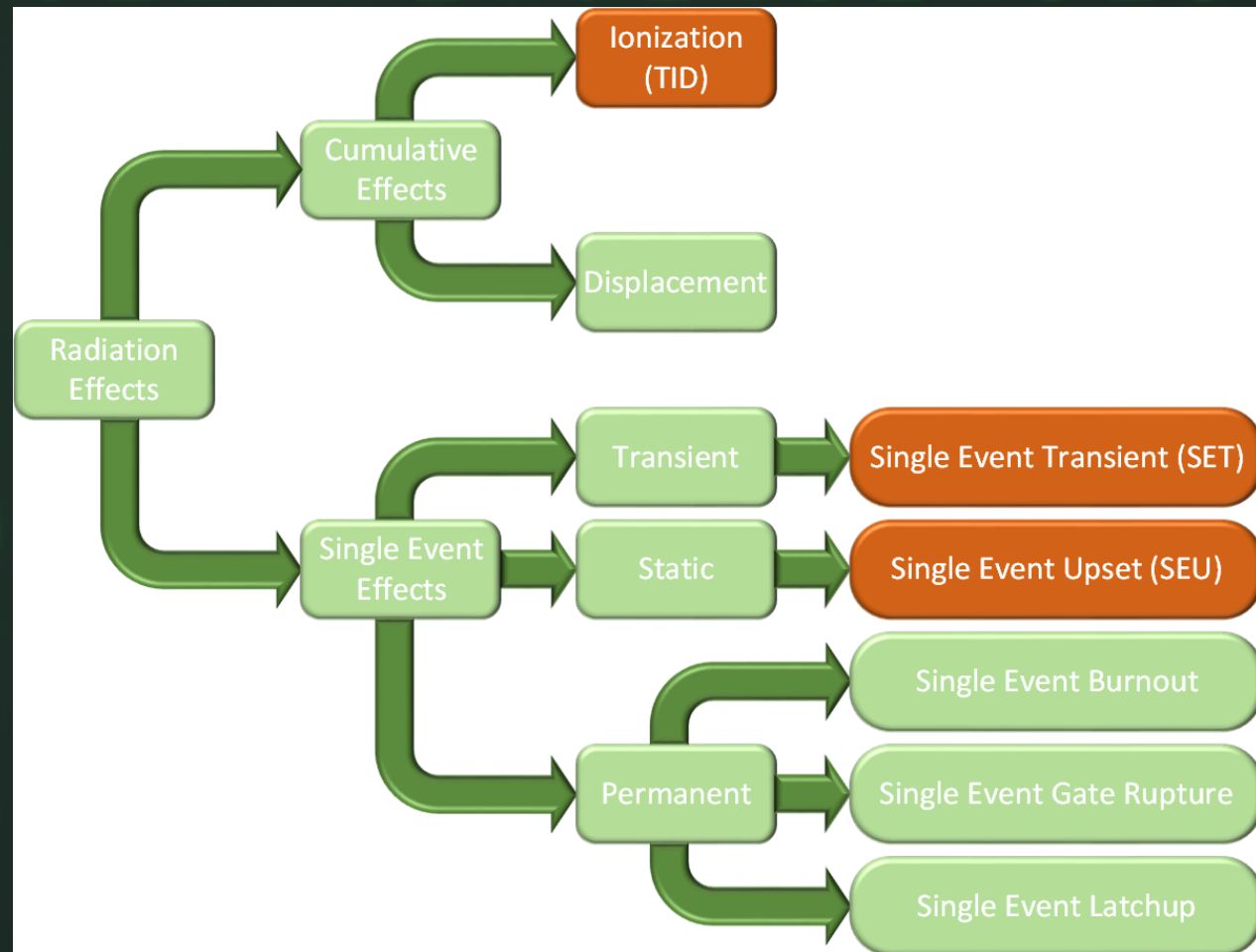
- F. Faccio and G.Cervelli, IEEE Trans. Nucl. Science, Vol.52, N.6 (2005)
pp.2413-2420
- F. Faccio et al., IEEE Trans. Nucl. Science, Vol.62 , N.6 (2015)
- M. Menouni, VERTEX 2017, ([testlrrad65 \(cern.ch\)](#))
- F. Faccio, et al., TWEPP 2015, ([TWEPP15_Faccio.key \(cern.ch\)](#))
- L. T. Clark et al., IEEE Transactions on Nuclear Science, vol. 69, no. 12,
pp. 2305-2313, (2022)
- H. Spieler, ([Radutr6.PDF \(lbl.gov\)](#))

SEU Bibliography to get you started

- S. Kulis, JINST, Vol.12, N.1 (2017) DOI: 10.1088/1748-0221/12/01/C01082
- TMRG Tool - <https://tmrg.web.cern.ch/tmrg/>

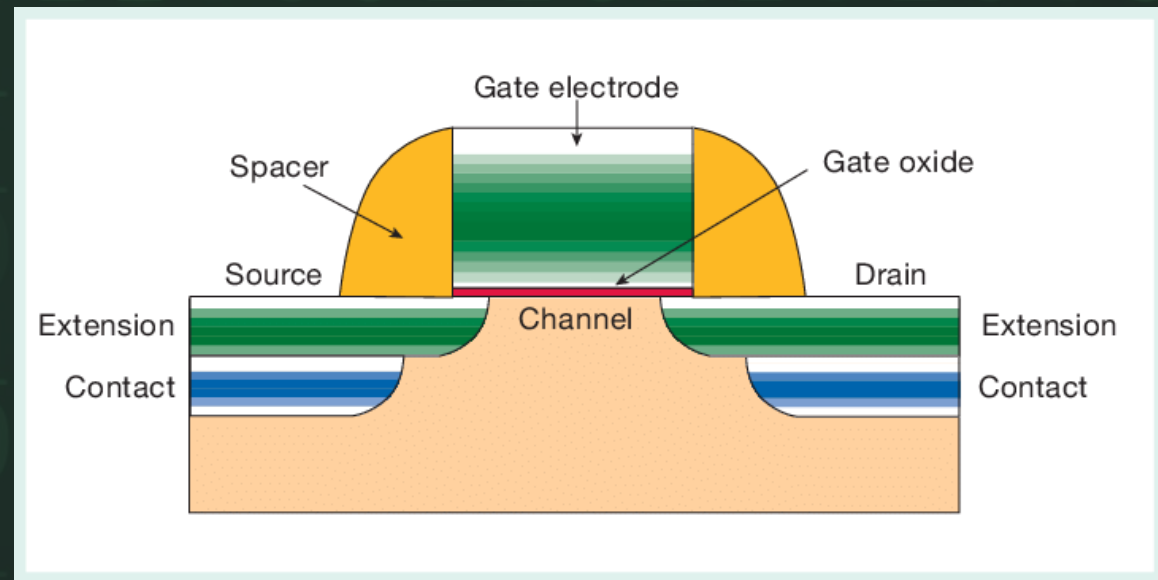
Taxonomy of Radiation Effects in Silicon Devices

As digital designers, we are most concerned with Total Ionizing Dose (TID) among the Cumulative Effects and Single Event Transients (SET) and Single Event Upsets (SEU) among the Single Event Effects (SEE).



Taxonomy of Radiation Effects in Silicon Devices

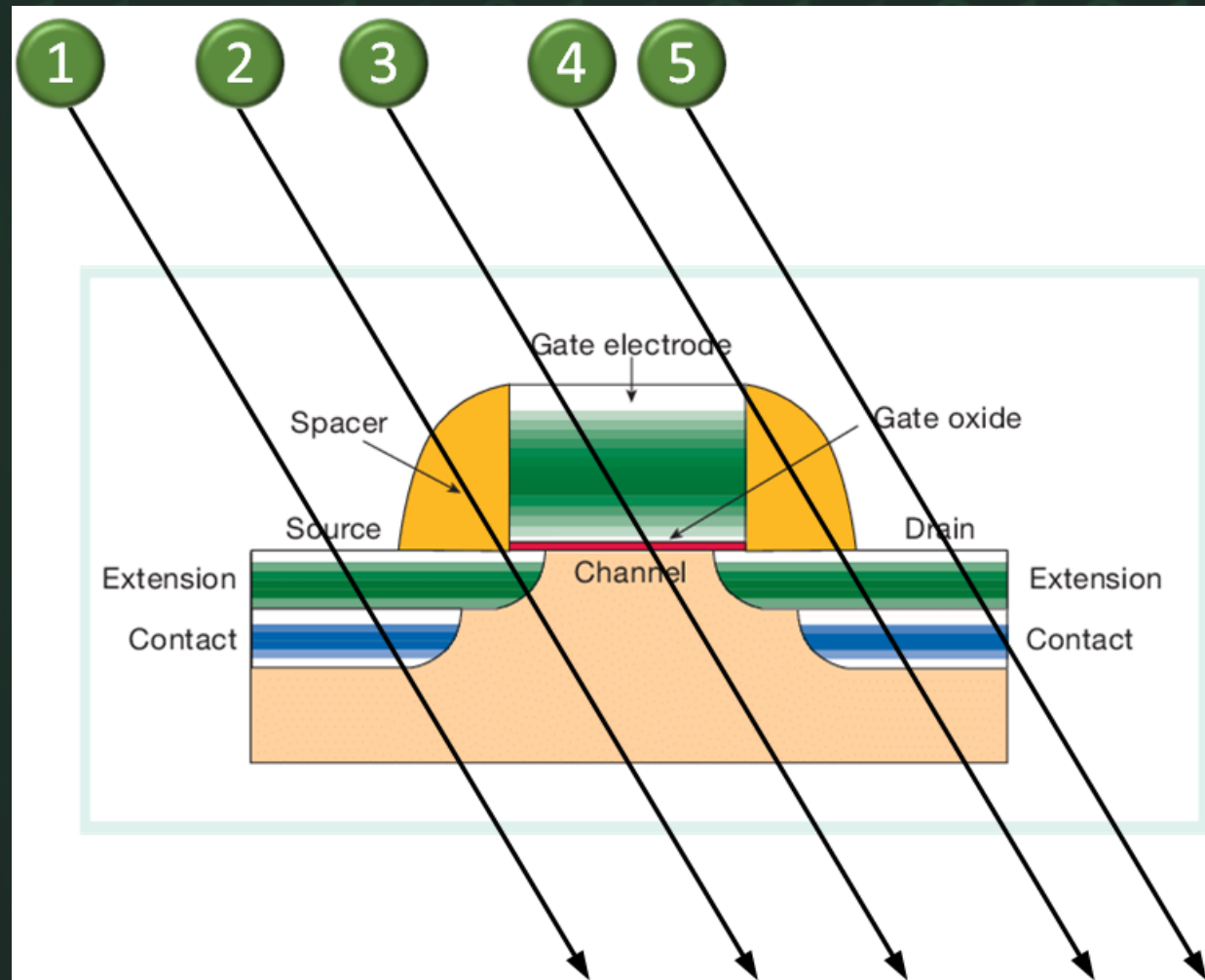
By now, we are all familiar with this cartoon cross section of a field effect transistor.



Taxonomy of Radiation Effects in Silicon Devices

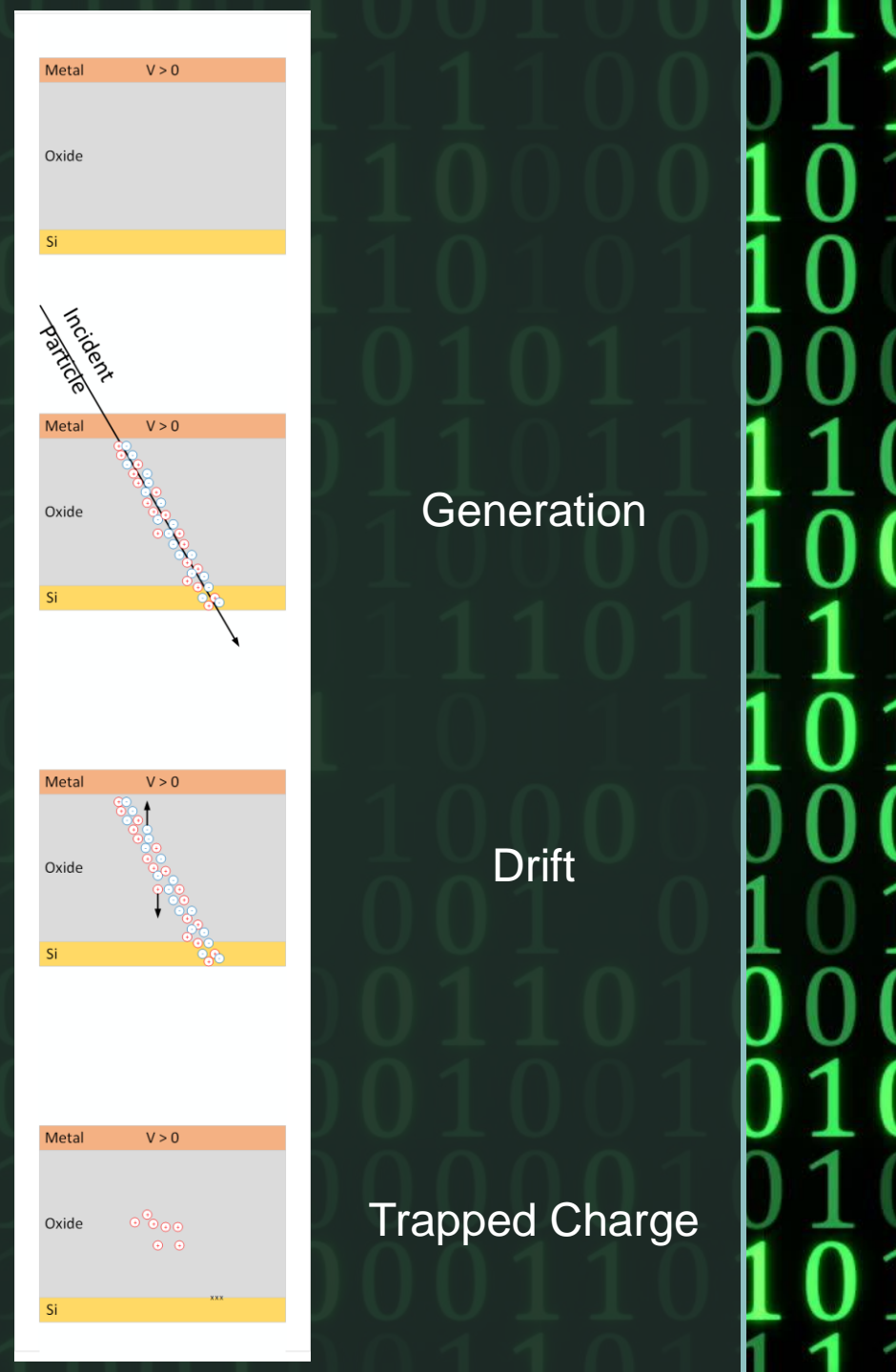
Radiation effects can be divided

- Between those caused by multiple interactions over time vs. those caused by a single interaction
- Between those that alter device models and those that affect data/states
- By WHERE your device was hit



TID: What happens when a particle hits an oxide?

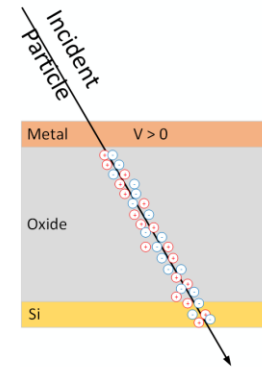
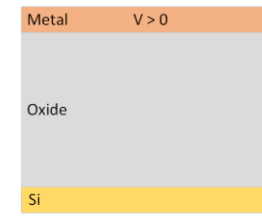
1. We generate particles through ionization
2. Under the influence of an electric field, particles drift, but mobility favors the electrons
3. The net effect is trapped charge in the oxide (and an increase in interface traps)



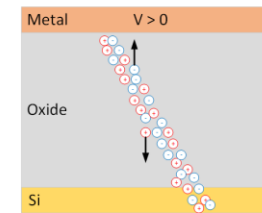
TID: What happens when a LOT of particles hit an oxide?

1. Most significantly: Threshold voltage shifts for both nFETs and pFETs.
2. Reduction in drive current
3. Increase in leakage current

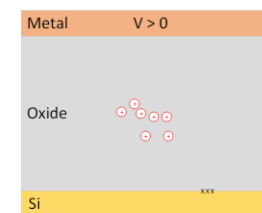
Net effect: Your models change



Generation

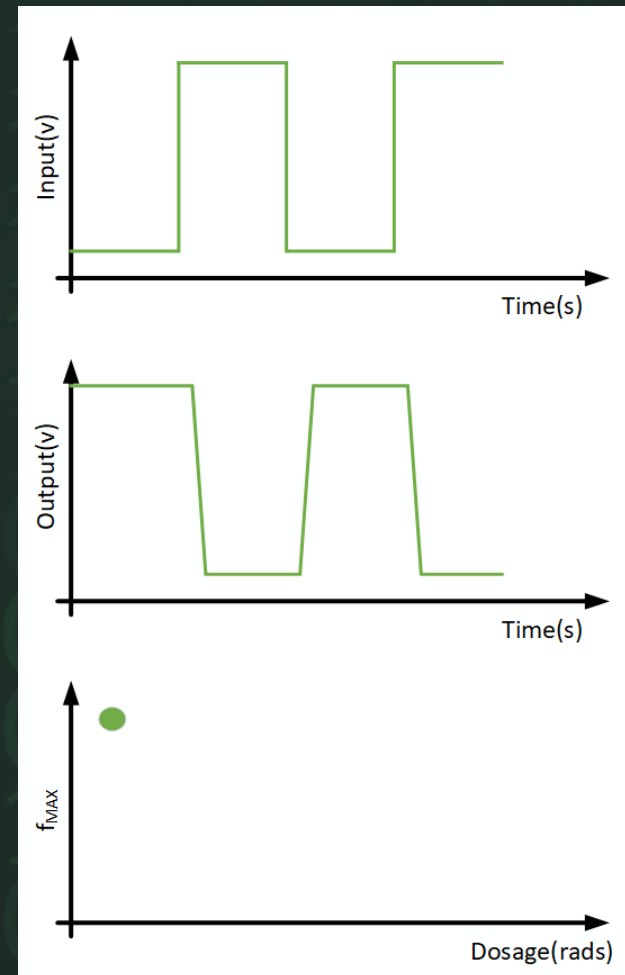
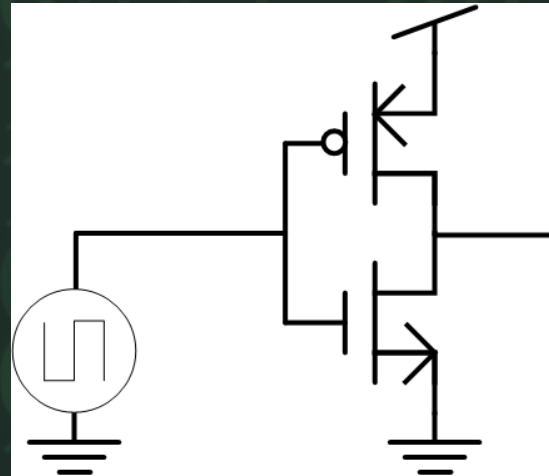


Drift



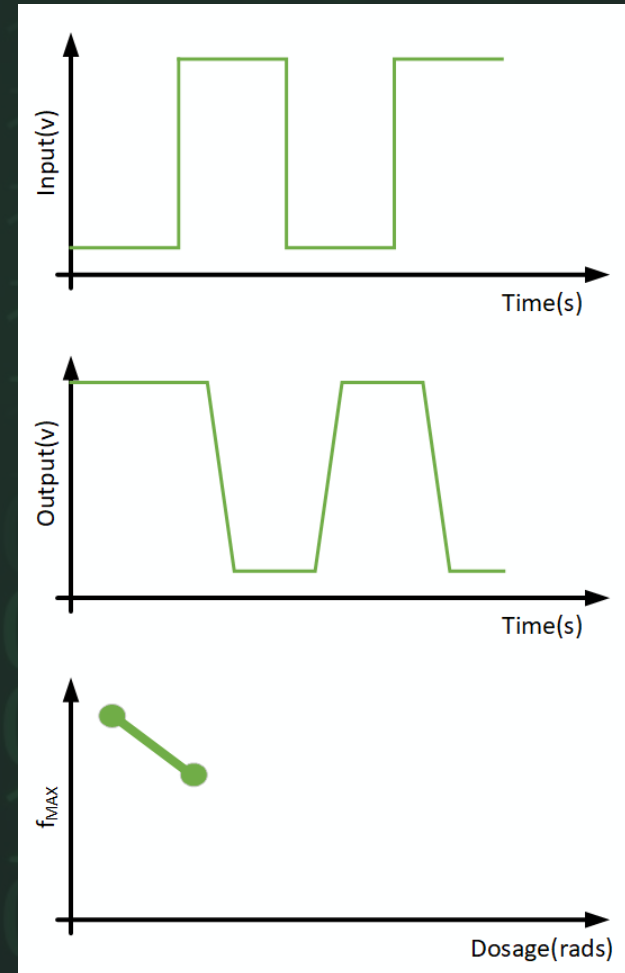
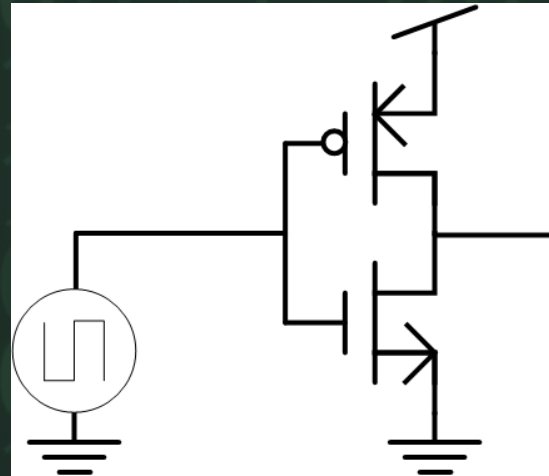
Trapped Charge

TID: What does this mean to a digital designer?

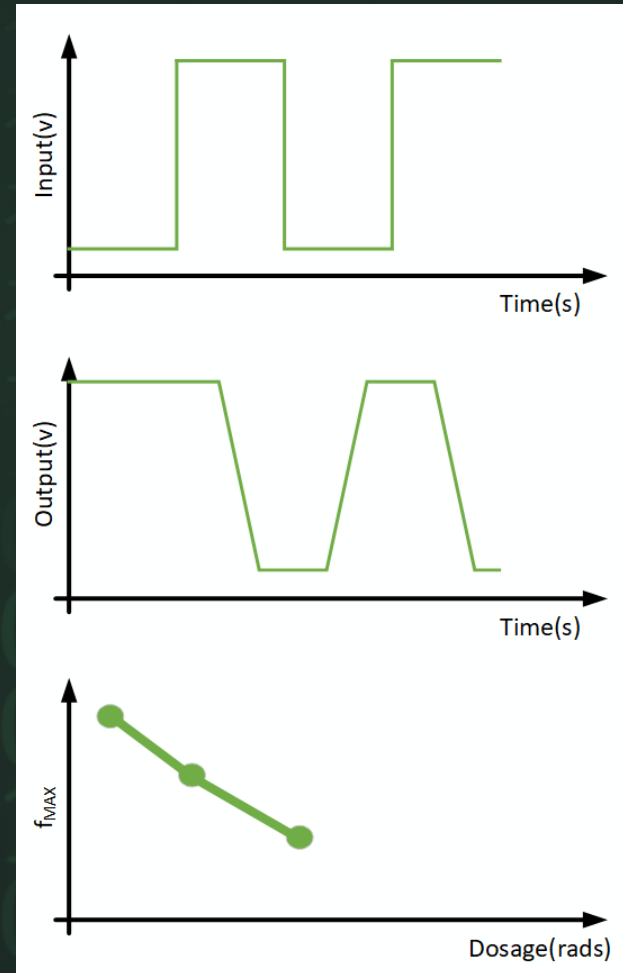
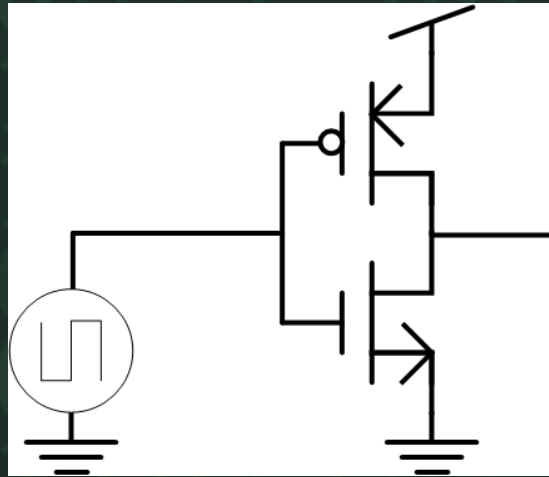


Credit: S. Kulis, <https://tmrg.web.cern.ch/tmrg/>

TID: What does this mean to a digital designer?

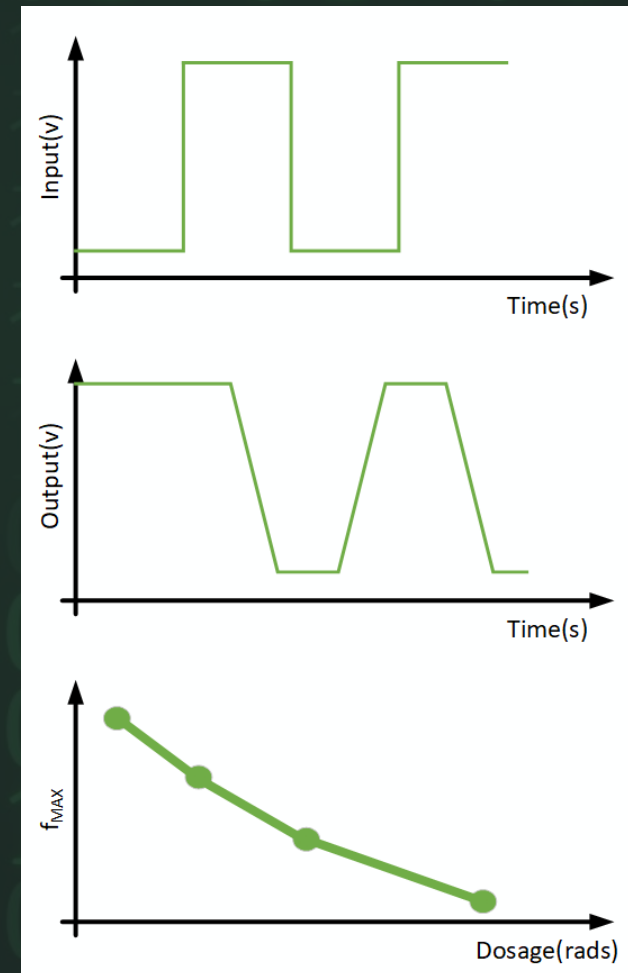


TID: What does this mean to a digital designer?



TID: What does this mean to a digital designer?

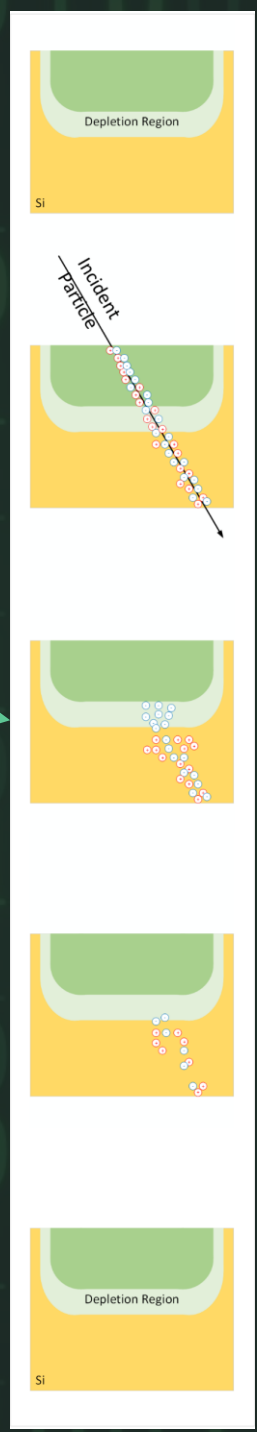
- With increasing dose, your devices slow down.
- With advanced technology nodes, the change in speed is somewhat more nuanced. For example, at lower dosages (think Space radiation levels), the speed actually has been found to increase because the nFET gets quicker before the pFET dominates. However, this is not significant for High Energy Physics.
- **GOOD NEWS:** With advanced nodes and reduced gate oxide thickness, it is harder to trap charge in the gate oxide, so we benefit from naturally improving TID resistance.
- **GOOD NEWS:** Many of the problems associated with leakage and drive current can be fixed by using transistors with larger gate widths and larger gate lengths
- **BAD NEWS:** Digital designers rarely get to choose their device dimensions



SEE: What happens when a particle hits a depletion region?

1. Once again, we generate particles through ionization
2. The electric fields within the depletion region very quickly cause drift current, resulting in a spike in current. (10s of picoseconds)
3. After this initial spike, there is a comparatively long tail that results from diffusion current outside the depletion region. (nanosecond scale)
4. After the tail, the excess charge disappears. It is an entirely transient phenomenon

The net result is the temporary deposition of charge on one of the nodes of your circuit

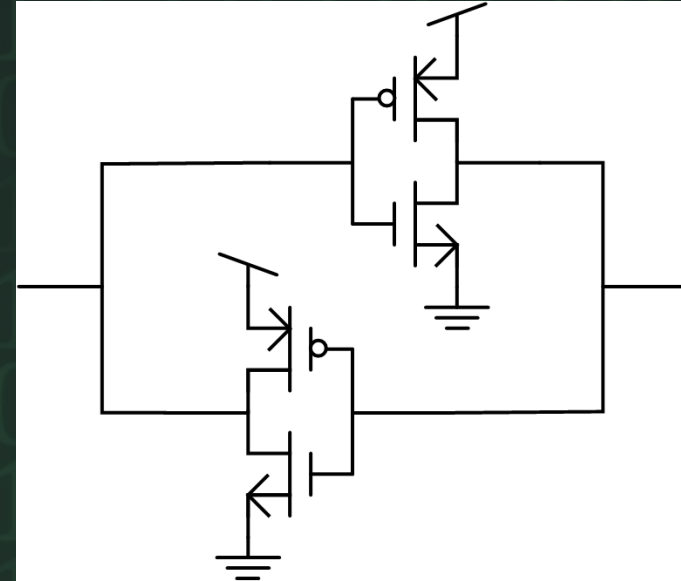


Generation

Drift

Diffusion and Recombination

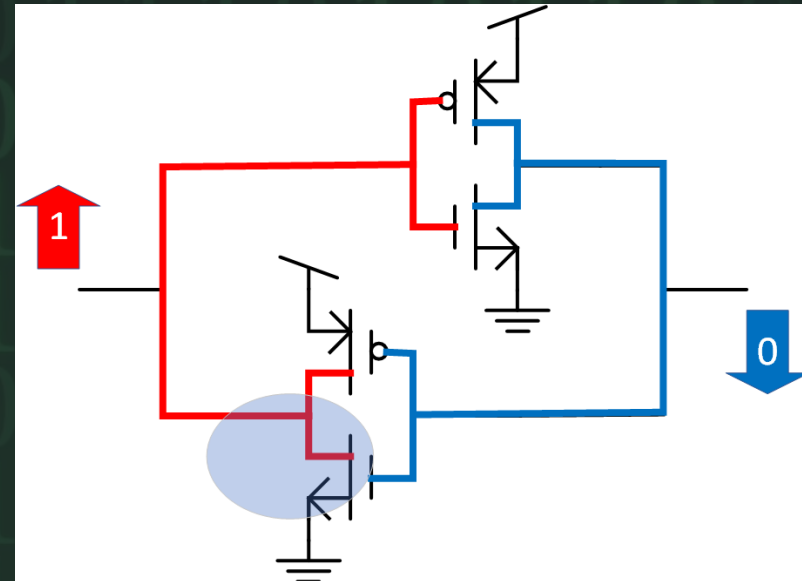
SEU: Single Event Upset



- Above is a classic 4T SRAM cell ignoring gating transistors

SEU: Single Event Upset

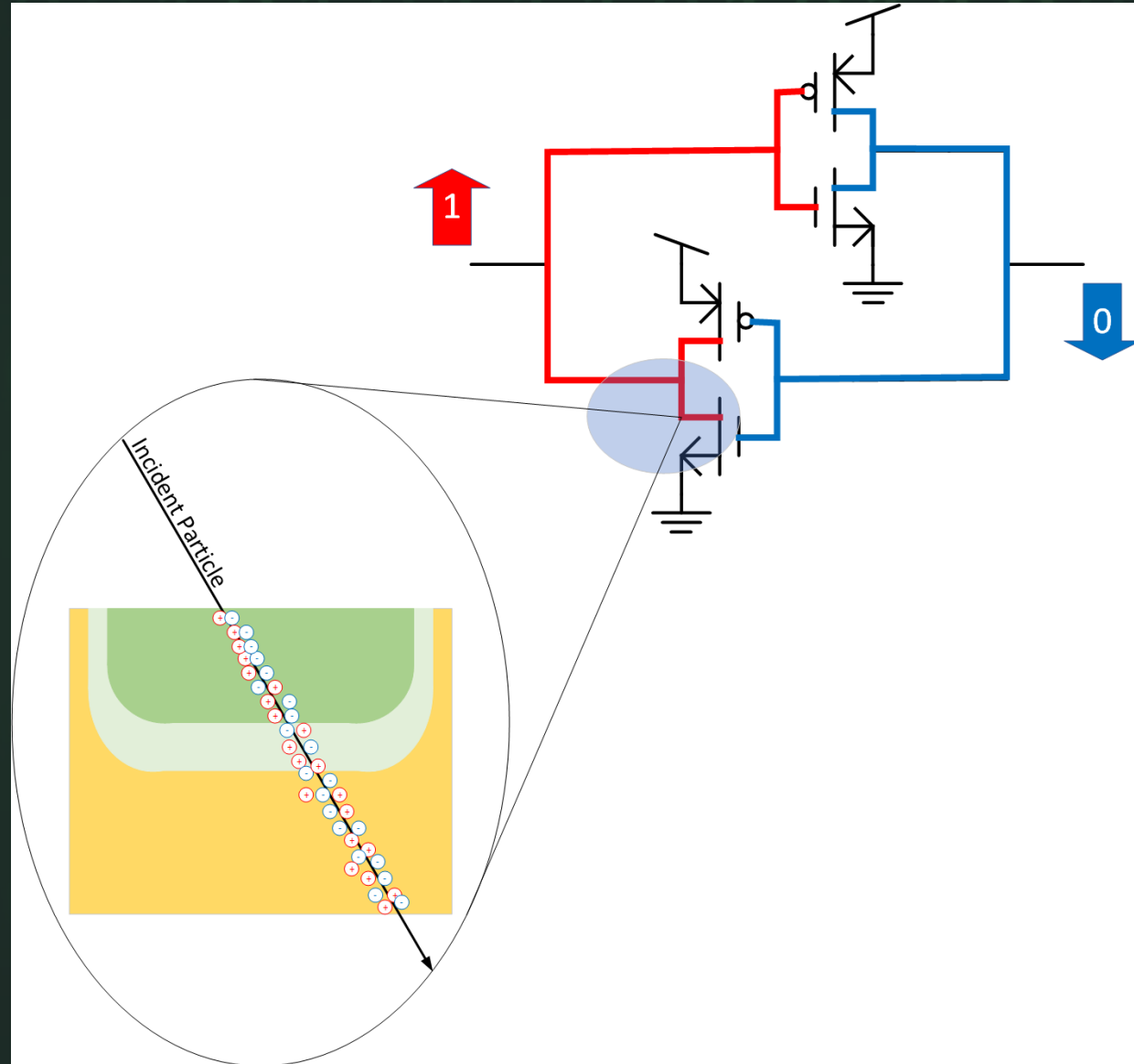
- Let us assume that it is set such that the left node is a Logical 1 and the right node is a Logical 0
- In this configuration, the left node is being pulled high by the pFET of the lower inverter.



- The nFET sits in a substrate that is grounded.
- The nFET is cut off
- The nFET's drain is shorted to the pFET's drain and consequently, the depletion region below the nFET drain (highlighted) is back-biased.

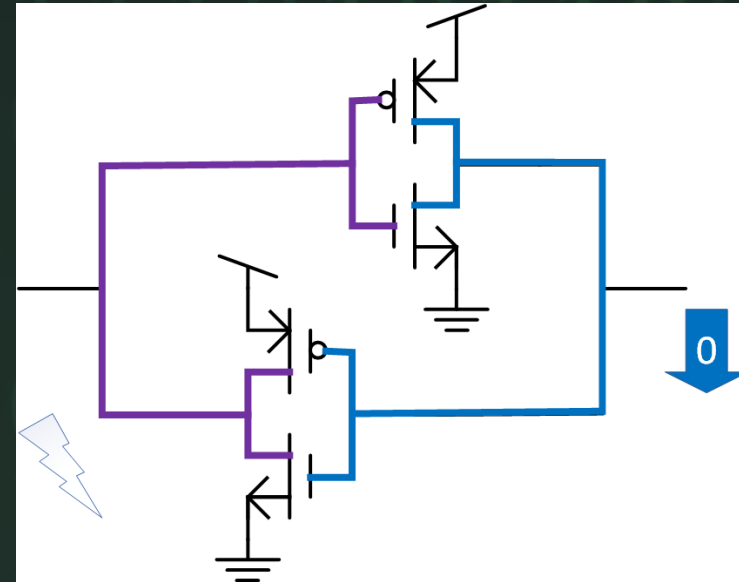
SEU: Single Event Upset

- If an incident particle strikes that specific location, as was shown in a previous slide, charge will be deposited on that node.



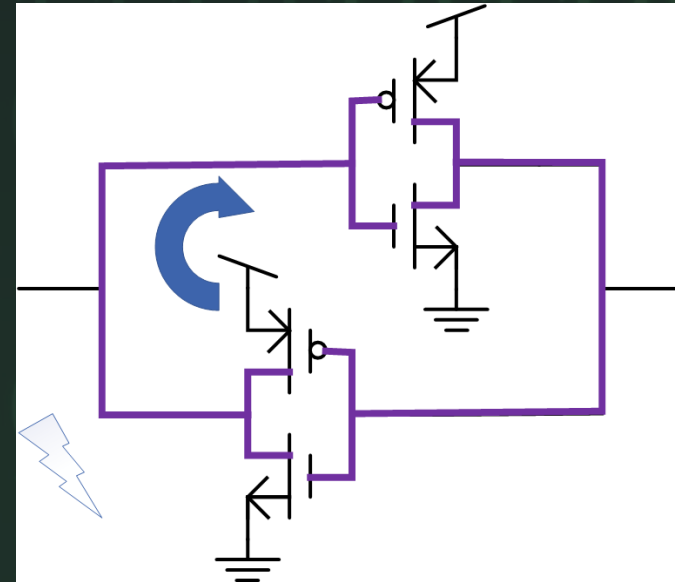
SEU: Single Event Upset

- The deposited charge can/will begin to pull the left node away from a perfect Logical 1
- How far it will pull away will depend on the nature and energy of the incident particle and the exact location of the strike



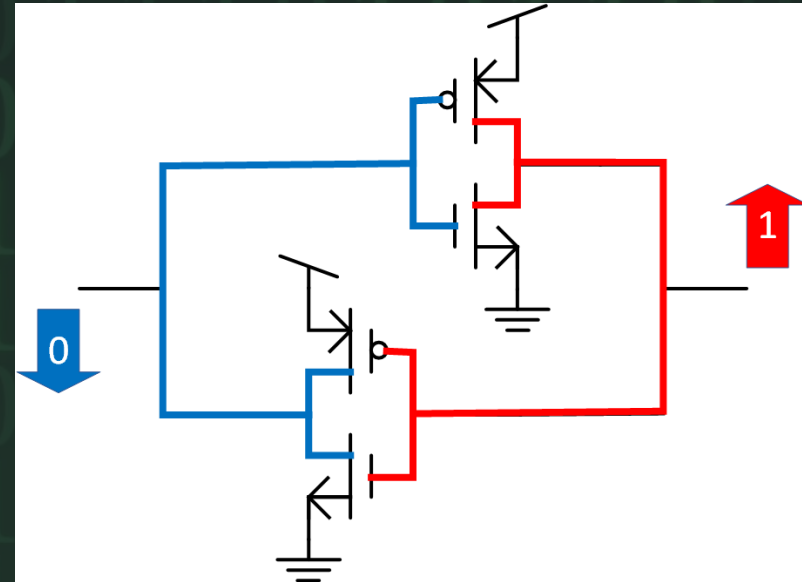
SEU: Single Event Upset

- If the charge deposition is above a certain critical value, the positive feedback of the SRAM circuit will begin to assert itself.



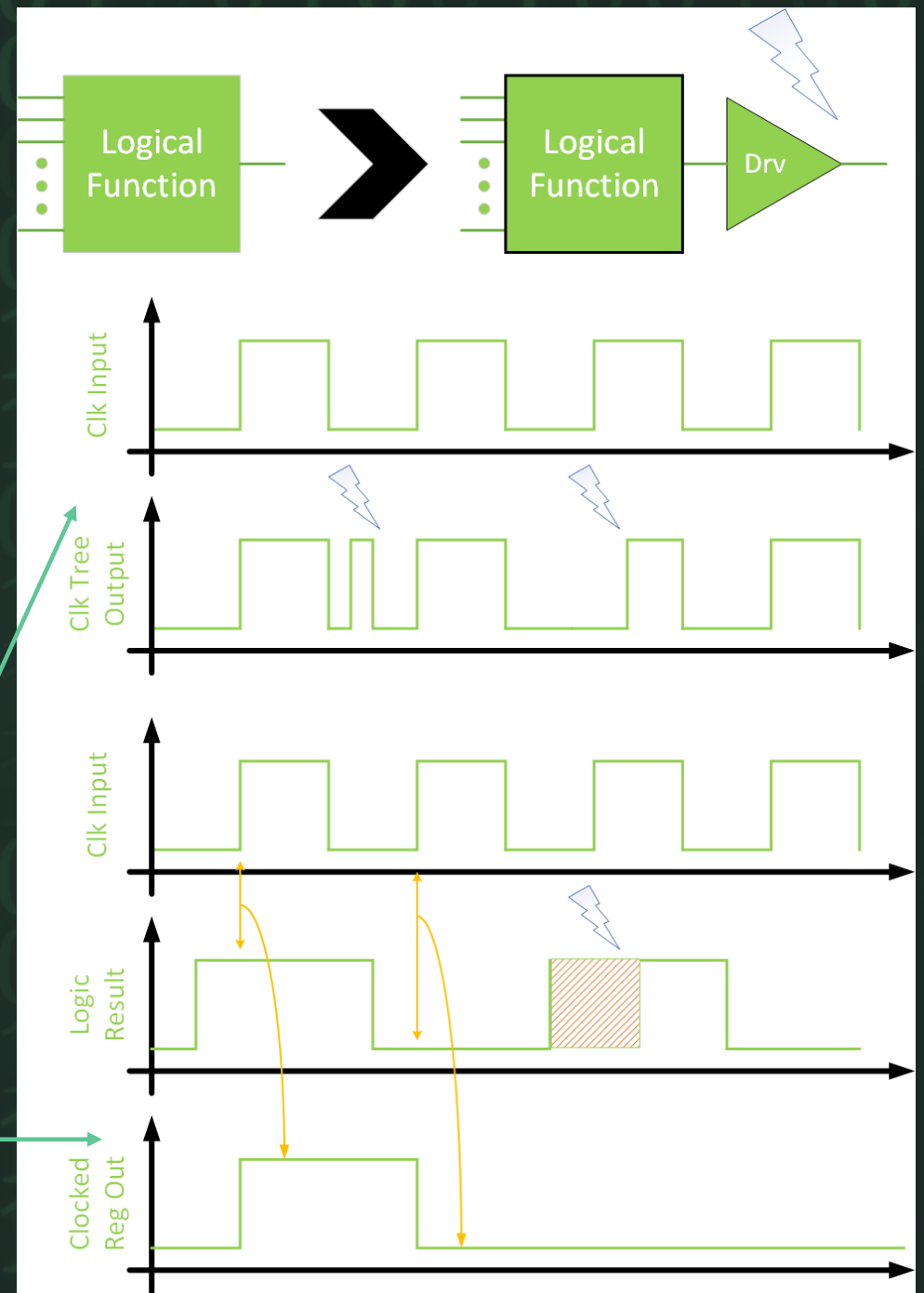
SEU: Single Event Upset

- The net result is a change in the state held by the SRAM
- A Single Event Upset can therefore be defined as a Single Event Effect in the presence of positive feedback that exceeds a critical charge deposition value.
- In the case of data in HEP systems, SEUs appear as noise in the data stream
- However, in the case of controls in HEP systems, SEUs can be severe.







SET: Single Event Transients

- Every logical function – combinatorial or sequential – can be thought to contain its function and an output driver. That output driver can be temporarily pulled away from its expected value by an incident particle
- If the logical function is a clock tree, this can result in spurious or missing clock pulses.
- In other cases, downstream registering of results can also be thrown off



What can we do about SEE?

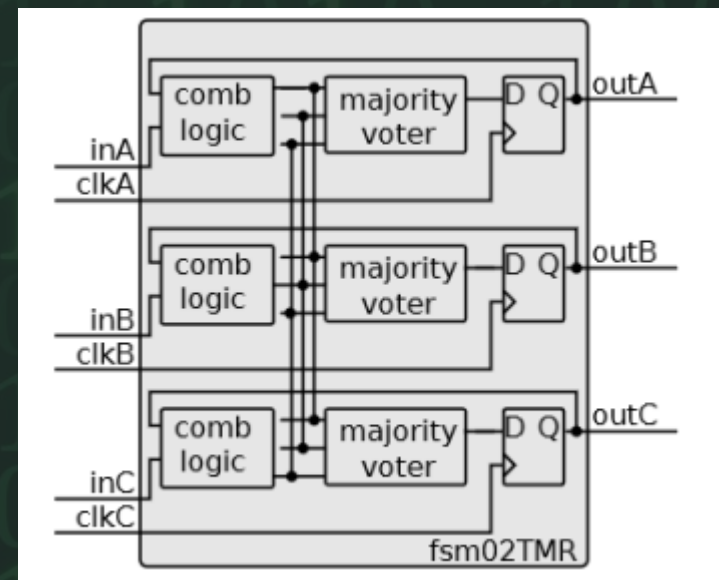
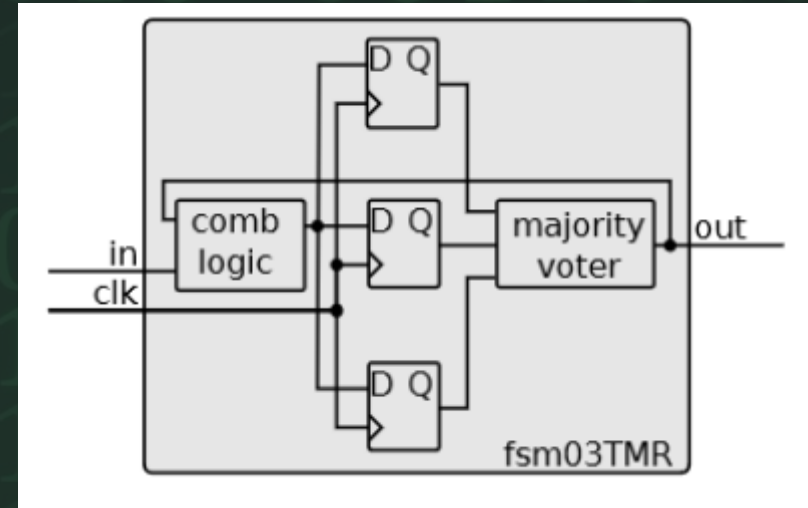
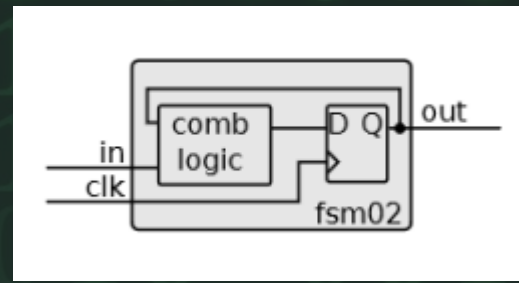
- Chose a different technology (e.g. SOI)
 - Typically, not up to the Digital Designer 
 - Increase the charge necessary to flip a node by increasing capacitance
 - Typically, not up to the Digital Designer 
 - Store data on multiple nodes (DICE, SEUSS, Whitaker Cell)
 - Heavily layout dependent and typically not available to the Digital Designer 
 - Encoding (Hamming, Reed-Solomon)
 - Triple Modular Redundancy
- Can be done with (relatively) simple RTL or PNR modifications 

TMR

- TMR is currently the most common form of SEE mitigation (RD53, HGCAL (CMS), COLDATA (DUNE))
- As the name indicates, it consists of triplicating each register
- Simply voting on the triplicated register protects against Single Event Upsets
- Full Triplication or Full TMR also protects against SET. Triplicated clocks protects against SET in the clock tree. Triplicated logic and voting protects against SET in the data path.

Triplicated registers –
“SEU-only”

Full TMR
– SEU and SET



TMR Insertion

1. “Sandeep Method” – TMR insertion after Synthesis
 1. HUGE Advantage! The RTL designer does not need to be aware that they are designing for extreme environments. If you have the RTL, you can triplicate any IP.
 2. Disadvantage – Not agile; full triplication is difficult
2. TMRG – TMR insertion after RTL
 1. RTL designer is required to add pragmas to the RTL in order for TMRG to do its job
 2. Complete agility and flexibility – full triplication; skip modules; etc.

TMR Insertion with TMRG

```
40 module packet_veto (
41     output logic packet_veto,
42     input logic [6:0] regArray_FBack_Header_eStatus,
43     input logic [15:0] I2C_RW_Veto_Pass1_Fail0,
44     //***
45     //*** Status flags
46     //***
47     input logic statusFlag_processingCRC,
48     //***
49     //*** Clocks and Reset
50     //***
51     input logic CLK,
52     input logic reset_b
53 );
54
55 //***TMRG*** TMRG insertion
56
57 //*****
58 // Hardware Package
59 //*****
60 import ECON_D_Hardware_PACKAGE::ZERO;
61 import ECON_D_Hardware_PACKAGE::ONE;
62
63 //*****
64 // Variables
65 //*****
66 logic e;
67 logic ht_msb;
68 logic ebo_msb;
69 logic m;
70 logic [3:0] address;
71 logic packet_veto_comb;
72 logic packet_veto_m1;
73 logic packet_veto_m2;
74
75 //*****
76 // Assignments
77 //*****
78
79 // ***TMRG*** These are (will become) the voters
80 wire packet_veto_m1Voted = packet_veto_m1;
81 wire packet_veto_m2Voted = packet_veto_m2;
82
83 wire reset_bVoted = reset_b;
```

```
75 //*****
76 // Assignments
77 //*****
78
79 // ***TMRG*** These are (will become) the voters
80 wire packet_veto_m1Voted = packet_veto_m1;
81 wire packet_veto_m2Voted = packet_veto_m2;
82
83 wire reset_bVoted = reset_b;
84
85 // ***TMRG*** Assign voted results to actual outputs
86 assign packet_veto = packet_veto_m2Voted;
87
88 assign e = regArray_FBack_Header_eStatus[6];
89 assign ht_msb = regArray_FBack_Header_eStatus[5];
90 assign ebo_msb = regArray_FBack_Header_eStatus[3];
91 assign m = regArray_FBack_Header_eStatus[1];
92
93 assign address = {e, ht_msb, ebo_msb, m};
94
95 assign packet_veto_comb = !I2C_RW_Veto_Pass1_Fail0[address];
96
97 // ***TMRG*** Simple sequential - Vote on flip-flop output
98 always_ff @(posedge CLK)
99     if(reset_bVoted == ZERO)
100         begin
101             packet_veto_m1 <= '0;
102         end
103     else if (statusFlag_processingCRC == ONE)
104         begin
105             packet_veto_m1 <= packet_veto_comb;
106         end
107     else
108         begin
109             packet_veto_m1 <= packet_veto_m1Voted;
110         end
111
112 // ***TMRG*** Simple sequential - Vote on flip-flop output
113 always_ff @(posedge CLK)
114     packet_veto_m2 <= packet_veto_m1Voted;
115
116 endmodule : packet_veto
117
```


TMR Insertion with TMRG

```
1 module packet_vetoTMR(  
2   output logic packet_vetoA ,  
3   output logic packet_vetoB ,  
4   output logic packet_vetoC ,  
5   input logic [6:0] regArray_FBack_Header_eStatusA ,  
6   input logic [6:0] regArray_FBack_Header_eStatusB ,  
7   input logic [6:0] regArray_FBack_Header_eStatusC ,  
8   input logic [15:0] I2C_RW_Veto_Pass1_Fail0A ,  
9   input logic [15:0] I2C_RW_Veto_Pass1_Fail0B ,  
10  input logic [15:0] I2C_RW_Veto_Pass1_Fail0C ,  
11  input logic statusFlag_processingCRCA ,  
12  input logic statusFlag_processingCRCB ,  
13  input logic statusFlag_processingCRCC ,  
14  input logic ClKA ,  
15  input logic ClKB ,  
16  input logic ClKC ,  
17  input logic reset_bA ,  
18  input logic reset_bB ,  
19  input logic reset_bC  
20 );  
21 import ECON_D_Hardware_PACKAGE::ZERO;  
22 import ECON_D_Hardware_PACKAGE::ONE;  
23 wor reset_bTmrErrorC;  
24 wire reset_bVotedC;  
25 wor packet_veto_m2TmrErrorC;  
26 wire packet_veto_m2VotedC;  
27 wor packet_veto_m1TmrErrorC;  
28 wire packet_veto_m1VotedC;  
29 wor reset_bTmrErrorB;  
30 wire reset_bVotedB;  
31 wor packet_veto_m2TmrErrorB;  
32 wire packet_veto_m2VotedB;  
33 wor packet_veto_m1TmrErrorB;  
34 wire packet_veto_m1VotedB;  
35 wor reset_bTmrErrorA;  
36 wire reset_bVotedA;  
37 wor packet_veto_m2TmrErrorA;  
38 wire packet_veto_m2VotedA;  
39 wor packet_veto_m1TmrErrorA;  
40 wire packet_veto_m1VotedA;  
41 logic eA ;  
42 logic eB ;  
43 logic eC ;  
44 logic ht_msbA ;  
45 logic ht_msbB ;  
46 logic ht_msbC ;  
47 logic ebo_msbA ;
```

```
62 logic packet_veto_m2A ;  
63 logic packet_veto_m2B ;  
64 logic packet_veto_m2C ;  
65 assign packet_vetoA = packet_veto_m2VotedA;  
66 assign packet_vetoB = packet_veto_m2VotedB;  
67 assign packet_vetoC = packet_veto_m2VotedC;  
68 assign eA = regArray_FBack_Header_eStatusA[6] ;  
69 assign eB = regArray_FBack_Header_eStatusB[6] ;  
70 assign eC = regArray_FBack_Header_eStatusC[6] ;  
71 assign ht_msbA = regArray_FBack_Header_eStatusA[5] ;  
72 assign ht_msbB = regArray_FBack_Header_eStatusB[5] ;  
73 assign ht_msbC = regArray_FBack_Header_eStatusC[5] ;  
74 assign ebo_msbA = regArray_FBack_Header_eStatusA[3] ;  
75 assign ebo_msbB = regArray_FBack_Header_eStatusB[3] ;  
76 assign ebo_msbC = regArray_FBack_Header_eStatusC[3] ;  
77 assign mA = regArray_FBack_Header_eStatusA[1] ;  
78 assign mB = regArray_FBack_Header_eStatusB[1] ;  
79 assign mC = regArray_FBack_Header_eStatusC[1] ;  
80 assign addressA = {eA,ht_msbA,ebo_msbA,mA};  
81 assign addressB = {eB,ht_msbB,ebo_msbB,mB};  
82 assign addressC = {eC,ht_msbC,ebo_msbC,mC};  
83 assign packet_veto_combA = !I2C_RW_Veto_Pass1_Fail0A[addressA] ;  
84 assign packet_veto_combB = !I2C_RW_Veto_Pass1_Fail0B[addressB] ;  
85 assign packet_veto_combC = !I2C_RW_Veto_Pass1_Fail0C[addressC] ;  
86  
87 always_ff @( posedge ClKA )  
88   if (reset_bVotedA==ZERO)  
89     begin  
90       packet_veto_m1A <= '0;  
91     end  
92   else  
93     if (statusFlag_processingCRCA==ONE)  
94       begin  
95         packet_veto_m1A <= packet_veto_combA;  
96       end  
97     else  
98       begin  
99         packet_veto_m1A <= packet_veto_m1VotedA;  
100      end  
101  
102 always_ff @( posedge ClKB )  
103   if (reset_bVotedB==ZERO)  
104     begin  
105       packet_veto_m1B <= '0;  
106     end  
107   else  
108     if (statusFlag_processingCRCB==ONE)  
109       begin  
110         packet_veto_m1B <= packet_veto_combB;  
111       end  
112     else  
113       begin  
114         packet_veto_m1B <= packet_veto_m1VotedB;  
115       end
```

Design (RTL)

RTL's Mission:
Fail Gracefully and Obviously

Failing Gracefully and Obviously

- Any knucklehead who has taken a SystemVerilog course can design a module that does what you ask it to do. That's the easy part.
- Failing Gracefully and Obviously - This is the hard part.
 - What happens when the data comes too fast?
 - What happens if your data is corrupted?
 - What happens if your (corrupted) data is out of range?
 - What happens when your control system is corrupted?
 - How quickly can you sense that something is wrong?

Failing Gracefully and Obviously

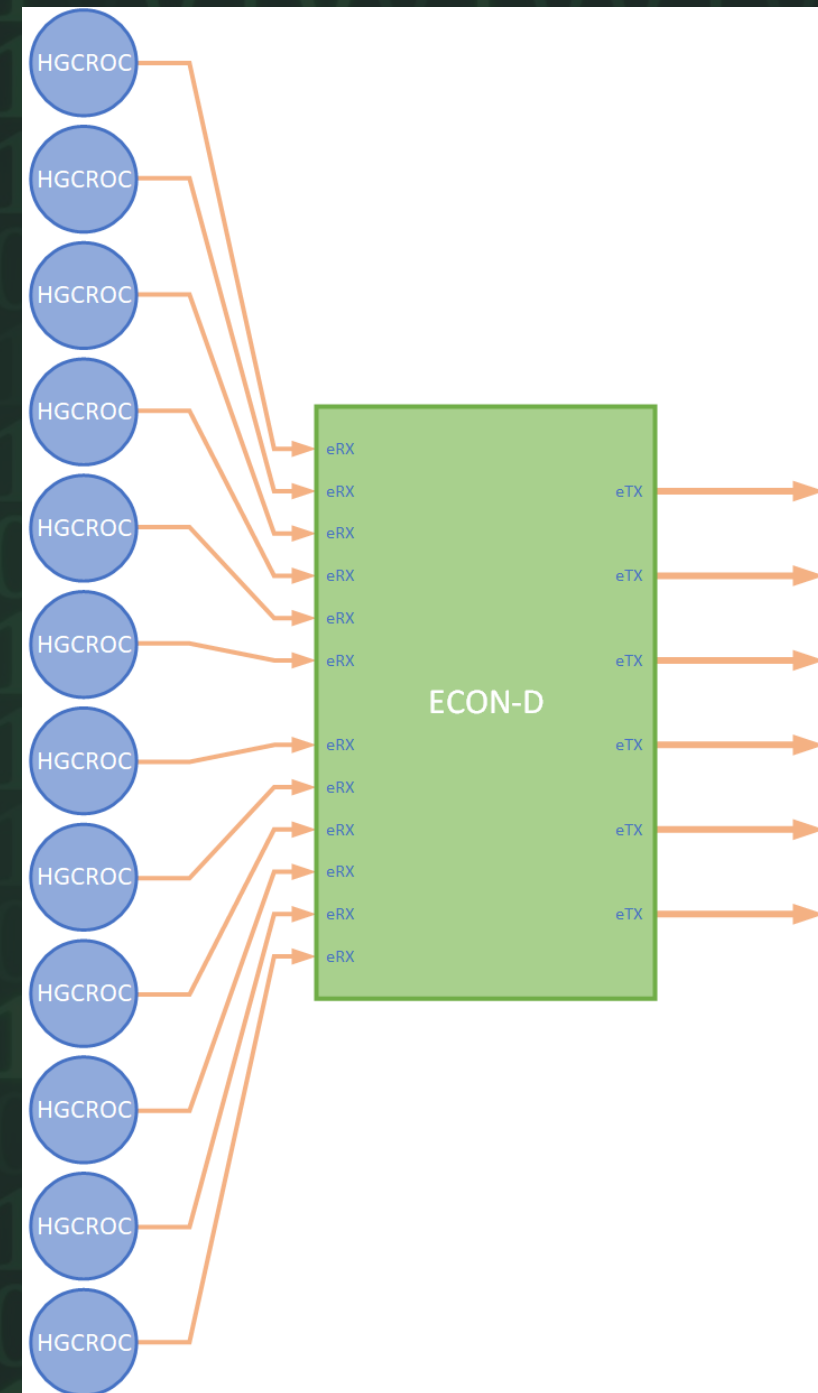
- Graceful Failure
 1. It keeps going despite failures
 2. It continuously restores itself
 3. It self-checks
 4. In the event of failure it returns to proper operation in a predictable fashion
- Obvious Failure
 1. It flags errors
 2. It notifies the backend of suspicious data

Failing Gracefully and Obviously

Here's another complication: Oftentimes, the overriding question is "Do you care about corruption?" As a general rule of thumb, we always care about corruption in control systems, but we frequently can ignore corruption in the data path. Unfortunately, history has shown that **everyone** will quickly forget that distinction and before you know it, you are implementing SEU mitigation on every signal in the chip. That costs significant power. Part of your job is keeping everyone on-task and in-agreement on what does and what does not require mitigation.

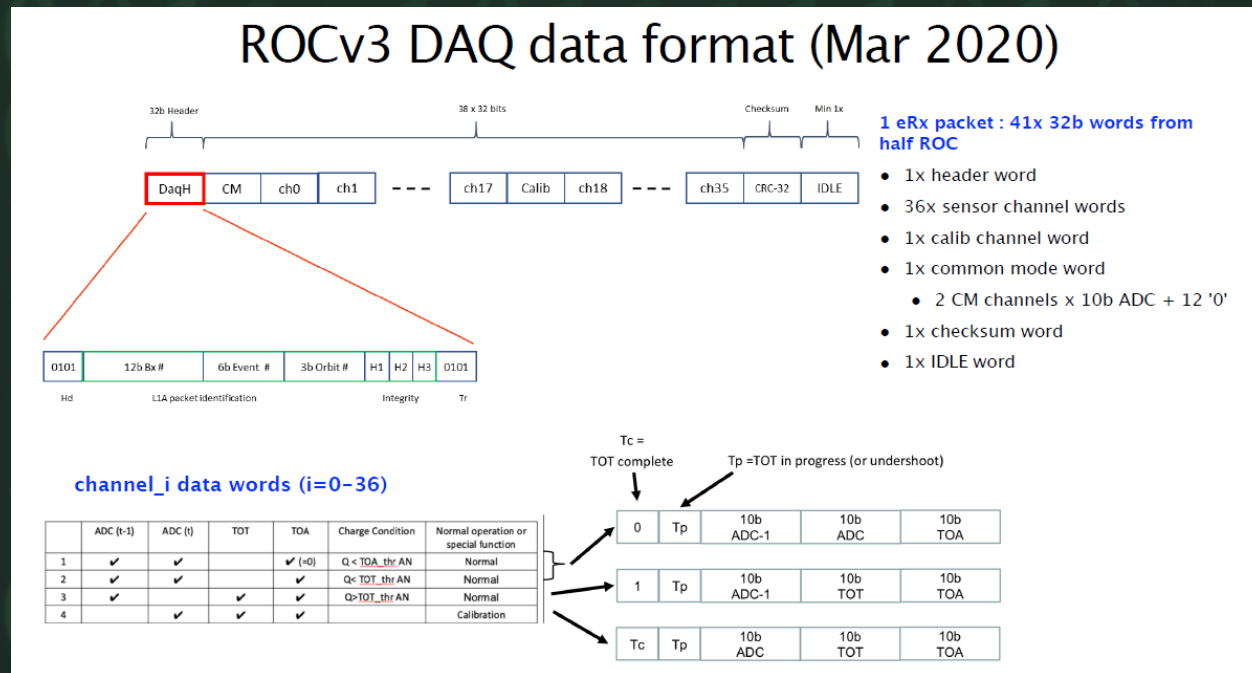
RTL Example: ECON-D

- The ECON's are Endcap Concentrators within CMS's High-Granularity Calorimeter (HGCAL) system
- Up to 12 HGCROC front end chips will feed each ECON
- Up to 6 1.28GHz outputs



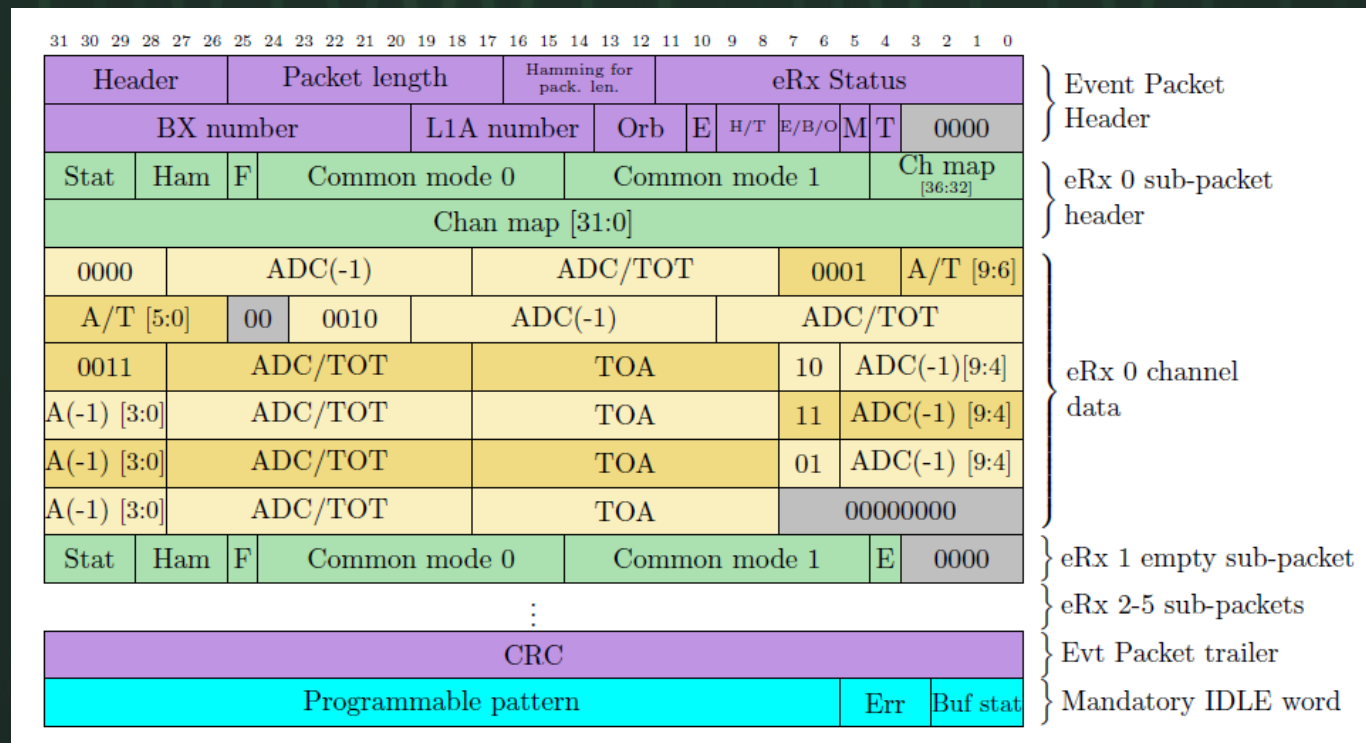
RTL Example: ECON-D

- Each HGCROC ships out a packet of 38 words plus a checksum delivered in a serial stream at 1.28GHz
- These packets will all be synchronized to one another
- *Between packets, the HGCROCs send SYNC/IDLE Words*



RTL Example: ECON-D

- Our job as data concentrator is to produce an aggregate data frame



RTL Example: ECON-D

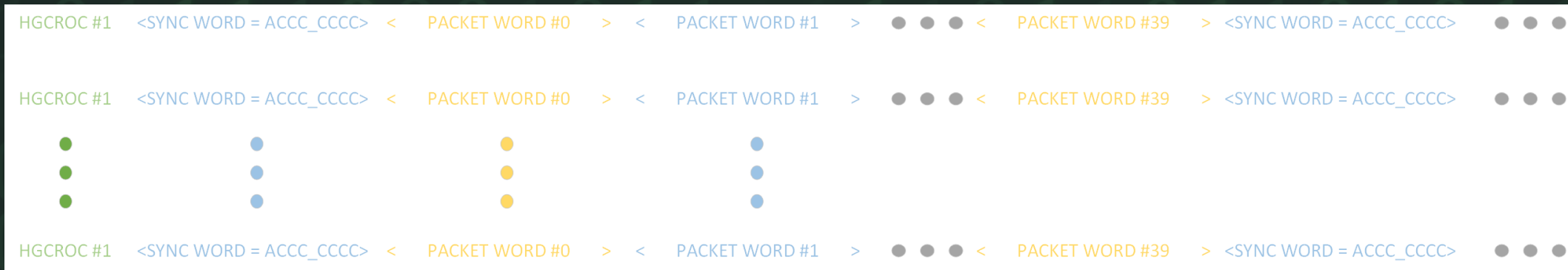
- The basic job is easy

1. Capture each packet
2. Simple arithmetic manipulation of the channel data
3. Create the output packet header
4. Assemble the output packet
5. Drive the output packet

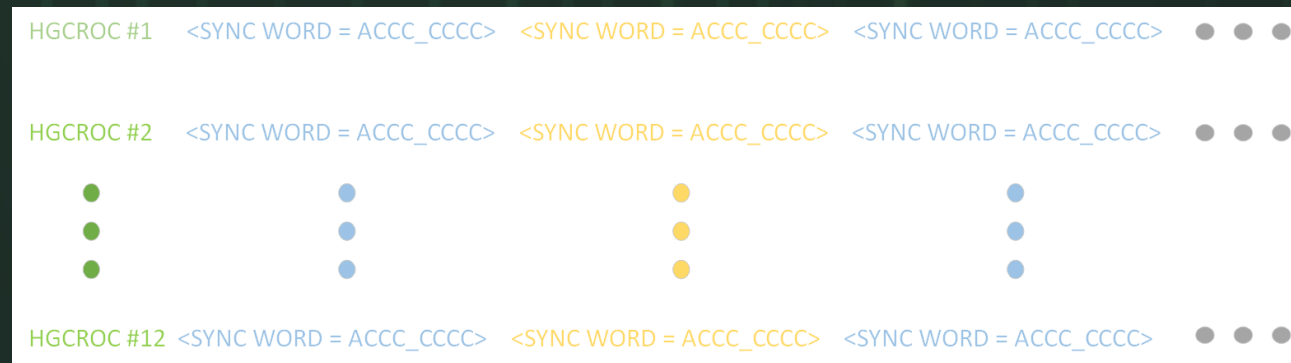
RTL Example: ECON-D

- In an extreme environment, things are not so easy. For example:
 1. How can we be certain that a packet is starting? What do we do if one HGCROC packet is not aligned with the another's?
 2. Radiation can corrupt the event, bunch and orbit numbers. How do we know what the right numbers are? How do we extract the right numbers?
 3. We can control this with one main state machine. What happens if radiation corrupts our state?

How can we be certain that a packet is starting?



1. “Where is a packet?” is logically equivalent to “Where is it not NOT a packet?”
2. Sync Words in this design are all identical. This provides **redundant information that you can use to your advantage.**
3. Solution: Find Sync Words independently for each eLink and see if all active eLinks agree.



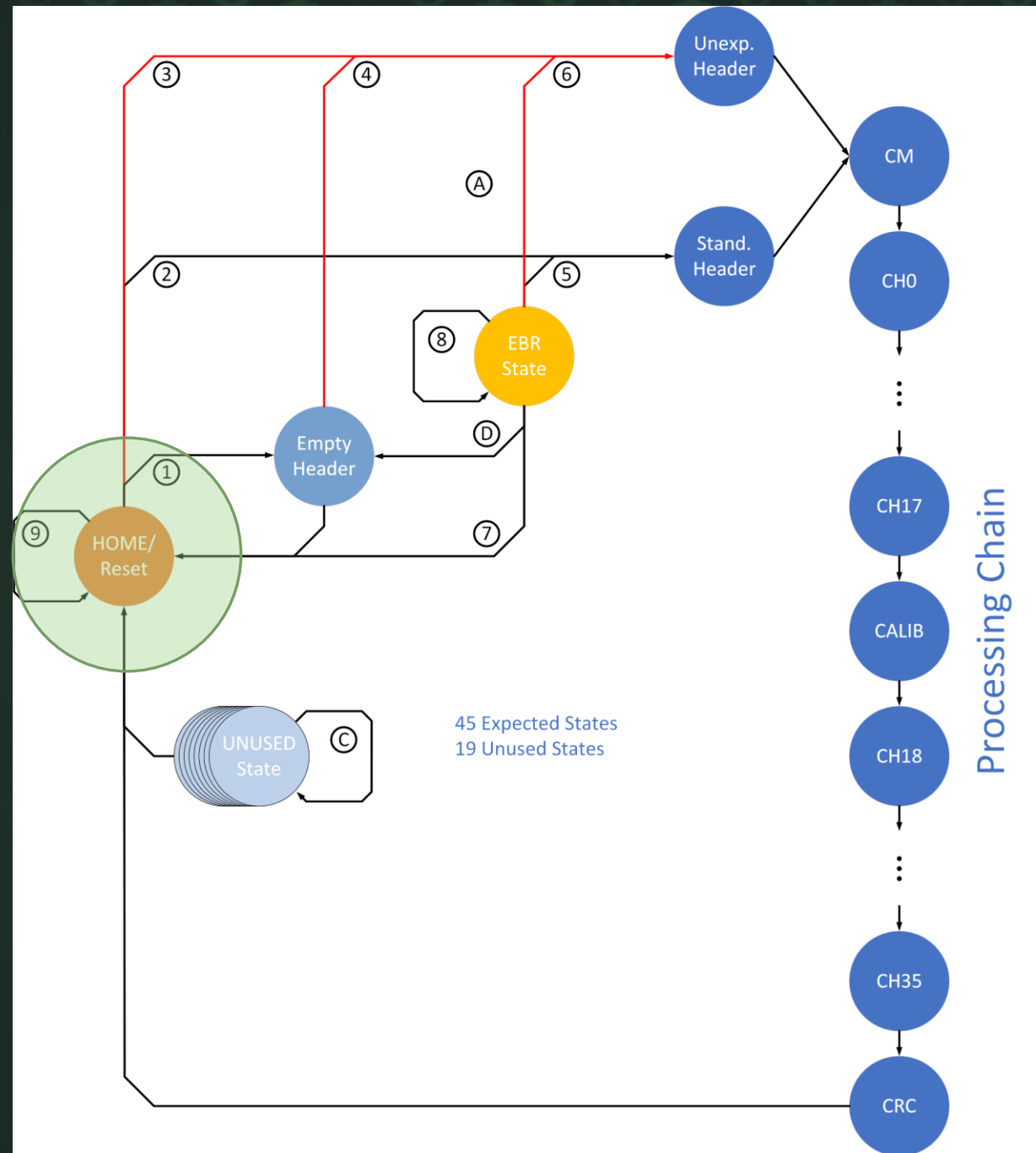
1. **Graceful Failure:** They don't ALL have to agree, set a threshold
2. **Obvious Failure:** Establish a Confidence Level based on the number of Sync Words each channel agrees upon

▴ I found a Sync Word! Who cares!?

1. I can reliably build a foundation for my state machine
2. **Graceful Failure:** if I am anywhere else in my state space and all of my Sync Word detectors are telling me that they just found another Sync Word, maybe I should be in the Home state.

Set up a counter. Set up a threshold. Set up an internal reset back to the Home State.

3. **Obvious Failure:** Set up a status register to report if you ever reset yourself back to the Home state.



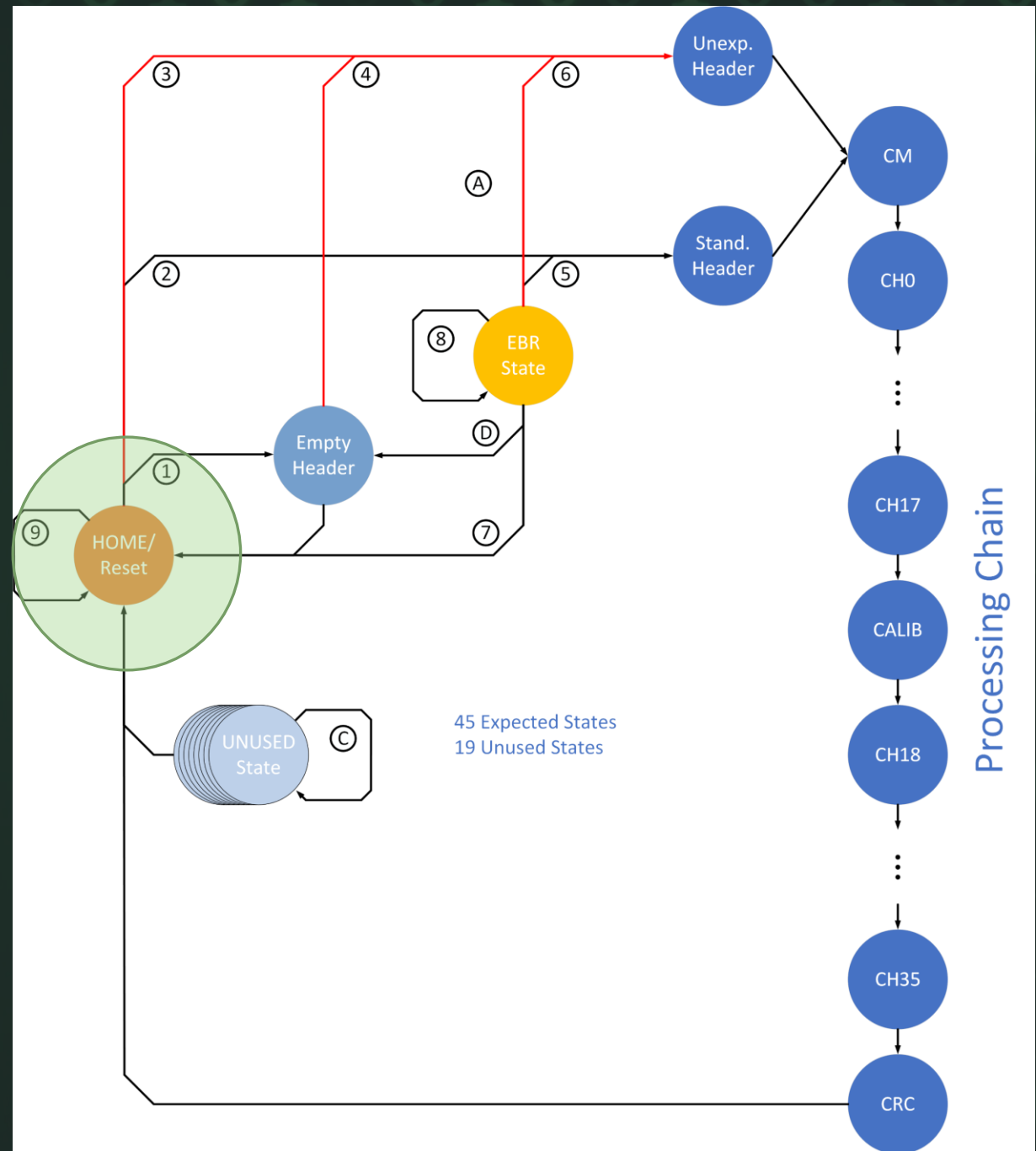
⌚ I found a Sync Word! Who cares!?

Alternately:

1. **Graceful Failure:** if I am in my HOME state and all of my Sync Word detectors are telling me that this is NOT a Sync Word, maybe I missed a packet

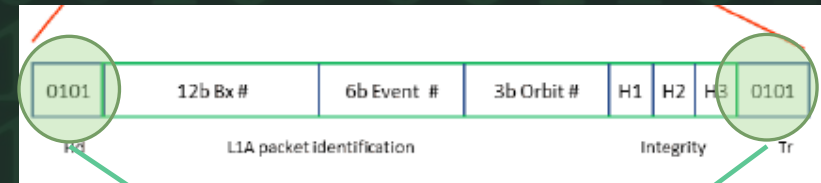
Set up a counter. Set up a threshold.

2. **Obvious Failure:** Set up a status register to report if you missed packets.



Packet Synchronization

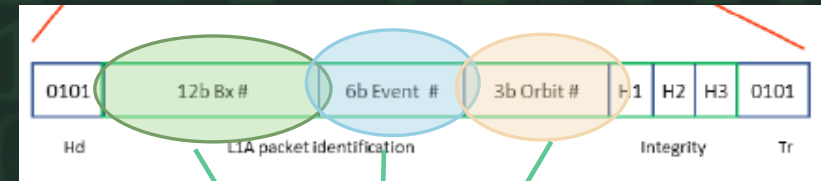
- Again: **redundant information that you can use to your advantage.** The Hdr and Trl fields are consistent.
- Graceful Failure:** They don't ALL have to agree either, set a threshold
 - Obvious Failure:** Establish another Confidence Level based on the number of Hdr and Trl fields each channel agrees upon



Present in all packet streams. Failed synchronization or SEU might result in one or two corruptions, but the majority will agree that this is a Header Word

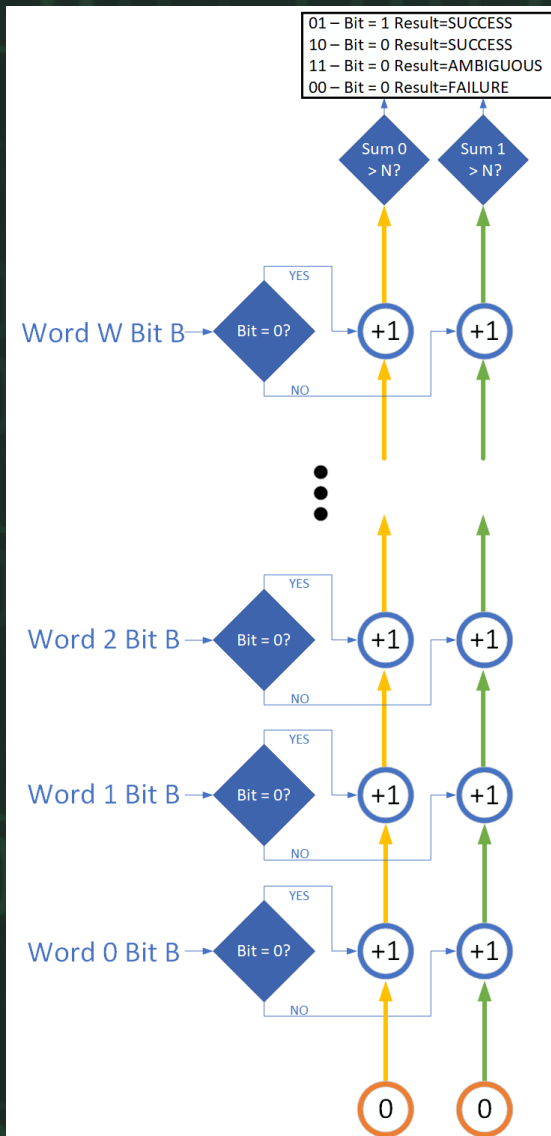
Corrupted Data: How can you tell what ought to be correct?

- Each packet contains a Bx (bunch) number, an Event number and an Orbit number (EBO).
- They ought to all be the same, but extreme environments remove that security blanket.
- You have redundant information in that all active eLinks are supposed to have the same EBO numbers, but no eLink is any more or any less likely to be the victim of the extreme environment.
- What do you do?

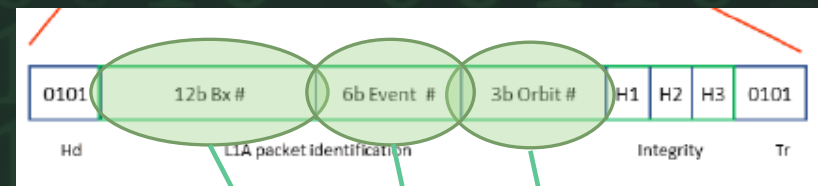


Present in all packet streams. Failed synchronization or SEU might result in one or two corruptions, but the majority will agree that this is a Header Word

Vertical Reconstruction – TMR on Steroids



Perfect
 Successful
 Ambiguous
 Failed



Should be the same in all packet streams

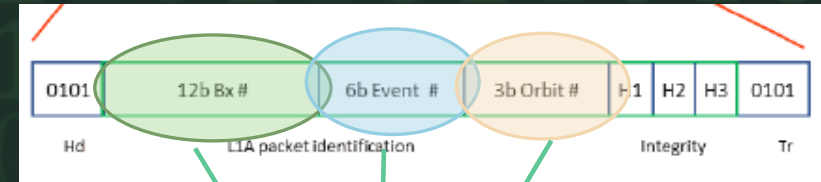
	Sum 0s	Bit_5	Sum 1s	Sum 0s	Bit_3	Sum 1s	Sum 0s	Bit_2	Sum 1s	Sum 0s	Bit_1	Sum 1s
eLink 0	0	0	0	1	0	0	0	1	1	1	0	0
eLink 1	0	0	0	2	0	0	0	1	2	2	0	0
eLink 2	0	0	0	3	0	0	0	1	3	3	0	0
eLink 3	0	0	0	4	0	0	0	1	4	4	0	0
eLink 4	1	0	1	4	1	1	0	1	5	5	0	0
eLink 5	0	0	0	5	0	0	0	1	6	6	0	0
eLink 6	0	0	0	6	0	0	0	1	7	6	1	1
eLink 7	0	0	0	7	0	0	0	1	8	6	1	2
eLink 8	0	0	0	8	0	0	0	1	9	6	1	3
eLink 9	0	0	0	9	0	0	0	1	10	6	1	4
eLink 10	0	0	0	10	0	0	0	1	11	6	1	5
eLink 11	0	0	0	11	0	0	0	1	12	6	1	6

11>Threshold?	1>Threshold?	12>Threshold?	0>Threshold?	0>Threshold?	12>Threshold?	6>Threshold?	6>Threshold?
YES	NO	YES	NO	NO	YES	NO	NO
Reconstruct 0	Reconstruct 0	Reconstruct 1	Reconstruct 0*	Reconstruct 0	Reconstruct 1	Reconstruct 0*	Reconstruct 0*
SUCCESS	SUCCESS	SUCCESS	FAILURE	SUCCESS	SUCCESS	FAILURE	FAILURE

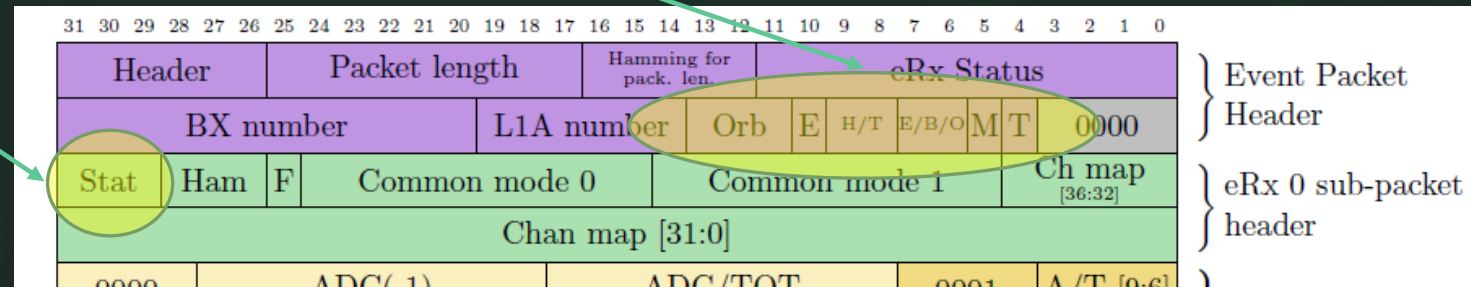
Packet Synchronization

- Vertical Reconstruction can help you to recreate what is most likely the uncorrupted data. It also lets you know how well you did at reconstructing the data

- Graceful Failure:** Failed or Ambiguous reconstruction allows you to flag suspect data and still keep going
- Obvious Failure:** Report the results of the vertical reconstruction within the output packet. Report how well each eLink did at finding the reconstructed data

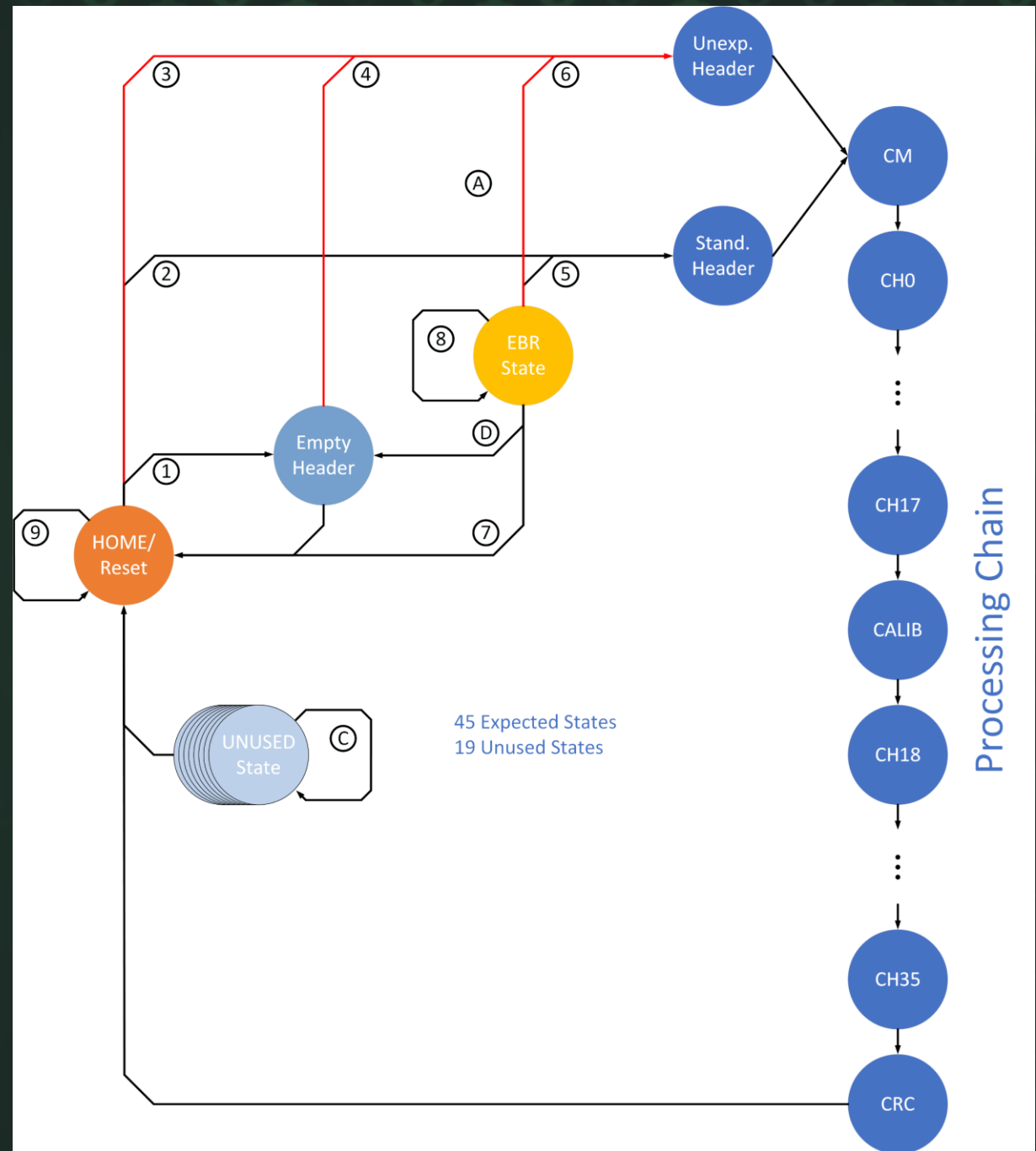


Present in all packet streams. Failed synchronization or SEU might result in one or two corruptions, but the majority will agree that this is a Header Word



What happens if radiation corrupts our state?

1. Sometimes you can define your states to advance like a Grey Code counter. That way, if the next state differs from the last state by more than one bit, you can flag it. That doesn't work here because of the branching.
2. Alternately, you can keep redundant copies of the last state to see if it is allowable for the current state to be X when the last state was Y.
3. ALWAYS: fill out the state space in your state machines and make sure that there is path back to HOME that ****WILL**** happen.





Verification

Types of Verification

- **Functional Simulation**

1. Testcase - You generate the stimulus and decide when it is injected into the RTL
2. Driver - You write the drivers or BFM's that do the injection of the stimulus based on a certain protocol
3. Monitor - You write the monitors that receive the output from the RTL
4. Model - You code a detailed reference model, a zero-time equivalent of the RTL, which produces a predicted result
5. Checkers - You write the checkers and scoreboard that compare the output from the RTL versus that from your reference model and declare Pass or Fail

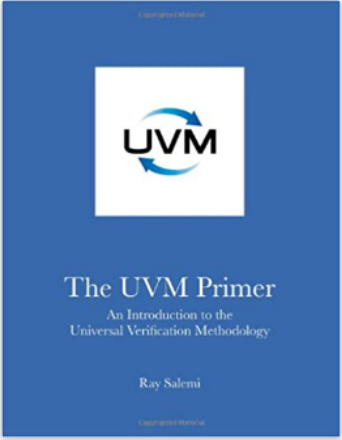
Types of Verification

- **Formal Verification** - the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics
- SEU injection framework for radiation-tolerant ASICs, a formal verification approach
 - Journal of Instrumentation
 - 2023-02-01 | Journal article
 - DOI: 10.1088/1748-0221/18/02/c02023
 - Part of ISSN: 1748-0221
 - CONTRIBUTORS: M. Lupi; A. Pulli

UVM – The Universal Verification Methodology

Books › Computers & Technology › Computer Science

[Look inside](#) ↓



The UVM Primer: A Step-by-Step Introduction to the Universal Verification Methodology 1st Edition
by [Ray Salemi](#) (Author)

4.3 ★★★★★ 79 ratings [See all formats and editions](#)






Kindle
\$47.95
You Earn: 144 pts
[Read with Our Free App](#)

Paperback
\$42.76 - \$47.95 [prime](#)
5 Used from \$42.76
14 New from \$43.95

Extra Savings 90 days FREE. Terms apply. [1 Applicable Promotion](#) ↓


The UVM Primer uses simple, runnable code examples, accessible analogies, and an easy-to-read style to introduce you to the foundation of the Universal Verification Methodology. You will learn the basics of object-oriented programming with SystemVerilog and build upon that foundation to learn how to design testbenches using the UVM. Use the UVM Primer to brush up on your UVM knowledge before a job interview to be able to confidently answer questions such as "What is a `uvm_agent`?", "How do you use `uvm_sequences`?", and "When do you use the UVM's factory." The UVM Primer's downloadable code examples give you hands-on experience with real UVM code. Ray Salemi uses online videos (on www.uvmprimer.com) to walk through the code from each chapter and build your confidence. Read The UVM Primer today and start down the path to the UVM.

[Report incorrect product information.](#)

ISBN-10	ISBN-13	Edition	Publication date	Language	Dimensions
 0974164933	 978-0974164939	# 1st	 October 23, 2013	 English	 8.5 x 0.44 x 11 inches

[See all 3 images](#)

Follow the Author

 [Ray Salemi](#) [Follow](#)

UVM – The Universal Verification Methodology

- A standardized methodology for verifying integrated circuit designs
- A class library that brings automation to the SystemVerilog language such as sequences and data automation features (packing, copy, compare) etc., and unlike the previous methodologies (e.g. OVM, e) developed independently by the simulator vendors, is an Accellera standard with support from multiple vendors: Aldec, Cadence, Mentor Graphics, Synopsys, Xilinx Simulator(XSIM)

Constrained Random Verification

- Directed Tests –
 1. Pick a scenario
 2. Make it happen on the test bench
 3. Check to see if it worked
 4. Repeat
- Constrained Random
 1. Randomize what test will be performed (reset, data-taking, start-up, etc)
 2. Within reasonable constraints of the test, randomize the DUT's configuration
 3. Within reasonable constraints of the configuration, randomize the data coming into the DUT
 4. Automate this and run until the computer network cries out for mercy

Coverage – Where “Randomization” becomes “Engineering”

- You are probably asking: How is Constrained Random any different from just running lots and lots of different scenarios? The answer lies in coverage.
- What is Coverage?
 - Toggle Coverage – Design activity; Did a particular bit flip?
 - Code Coverage – How much of your RTL Code was tested?
 - Functional Coverage – What features were tested?
- Keep doing tests until you are satisfied with the coverage.
 - Toggle Coverage and Code Coverage is easy
 - Functional Coverage is the big one and it is the playground in which Constrained Random does its work

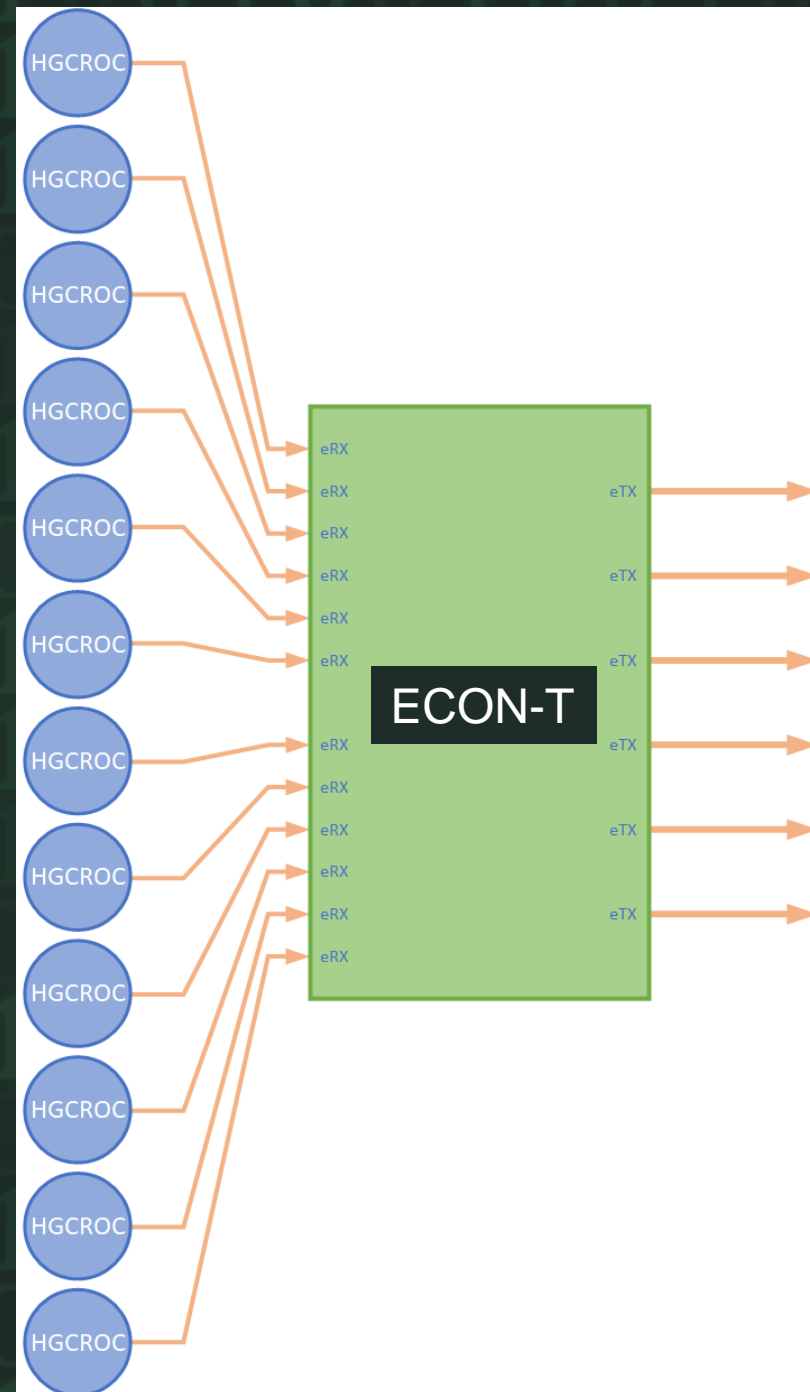
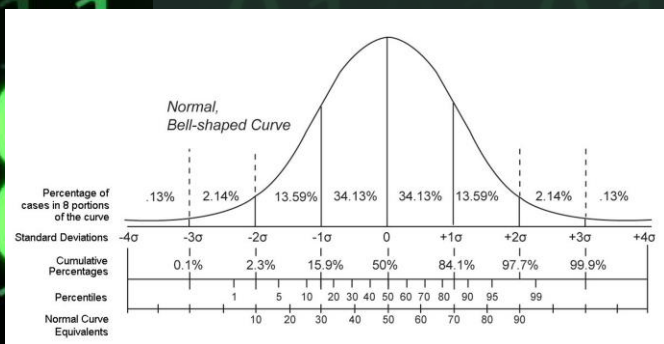
Coverage and Constrained Random Example

- In ECON-T the data words from the HGCROC consist of 4 Trigger Cell sums
- Recall:

3. Within reasonable constraints of the configuration, randomize the data coming into the DUT

The net result was that we were never seeing situations in which all the Trigger Cells were low or all the Trigger Cells were high – i.e. **OUR COVERAGE WAS INCOMPLETE!!!**

So, we **adjusted the constraints** so that we could get all low or all high.



Coverage – Where “Randomization” becomes “Engineering”

- With Constrained Randomization and Functional Coverage, simulation becomes verification.
- Verification is a systematic and mathematical approach to testing.
- The job becomes to uncover all the features of a design
 - To figure out the phase space of the features
 - To understand what features are mutually reliant and which ones are mutually exclusive
 - To ferret out all those features that must happen and all those that cannot be allowed.

Coverage – Where “Randomization” becomes “Engineering”

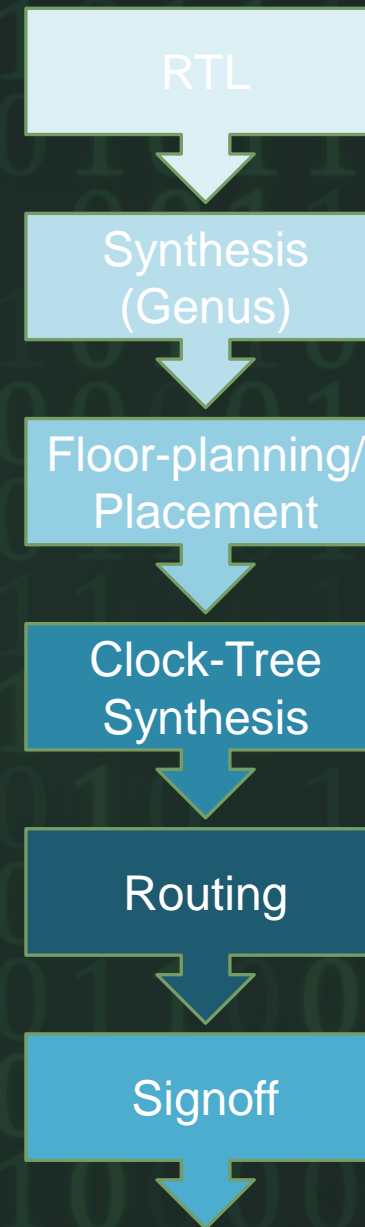
- The best part of this is that you do not need any particular skills to be involved in coverage – i.e. the Verification Plan.
 - We have divided the problem of Verification into Testbench Development (which requires a particular programming skillset) and Coverage Development (which does not).
 - The physicists, the RTL guys, the DAQ system architects, etc can all be part of Coverage Development.
- Better yet, they can all, also, be part of the Coverage Evaluation.
 - It used to be that they would ask us if the chip was ready and if it wasn't, it was our fault.
 - Now, they can ask, “How is the Coverage?” and , “Have you seen evidence of the issues that concern ME?”



Place And Route

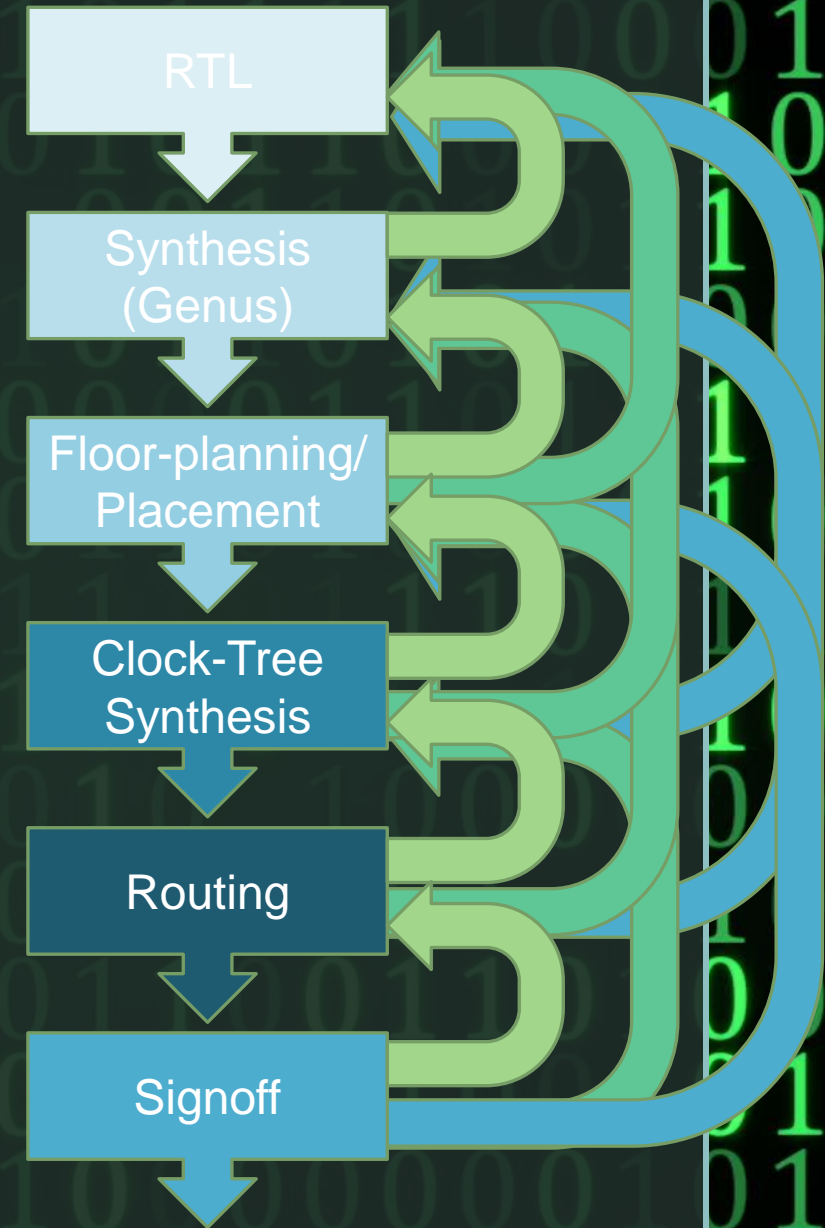
Semi-Custom Digital Design Flow

- RTL : Behavioral model of the circuit in SystemVerilog.
- Synthesis: RTL is mapped into a gate-level netlist. (SystemVerilog behavioral model to SystemVerilog structural model)
- Floor-planning: From the synthesized netlist, core area sizing, power planning, and pin assignment.
- Placement: Place Standard-cells & Macro-cells (layouts). No overlap is allowed between any two cells.
- Clock-Tree Synthesis: Clock tree is designed after placement
- Routing : The metal wires are used to connect the placed components
- Signoff: Make sure that the final structure meets the specified timing constraints

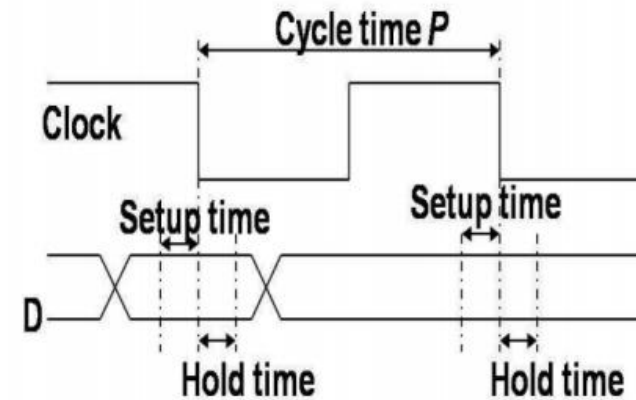
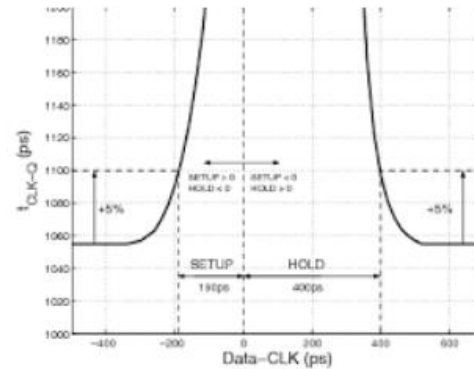
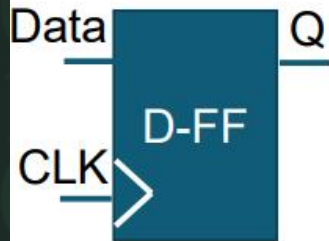


Semi-Custom Digital Design Flow

- RTL : Behavioral model of the circuit in SystemVerilog.
- Synthesis: RTL is mapped into a gate-level netlist. (SystemVerilog behavioral model to SystemVerilog structural model)
- Floor-planning: From the synthesized netlist, core area sizing, power planning, and pin assignment.
- Placement: Place Standard-cells & Macro-cells (layouts). No overlap is allowed between any two cells.
- Clock-Tree Synthesis: Clock tree is designed after placement
- Routing : The metal wires are used to connect the placed components
- Signoff: Make sure that the final structure meets the specified timing constraints

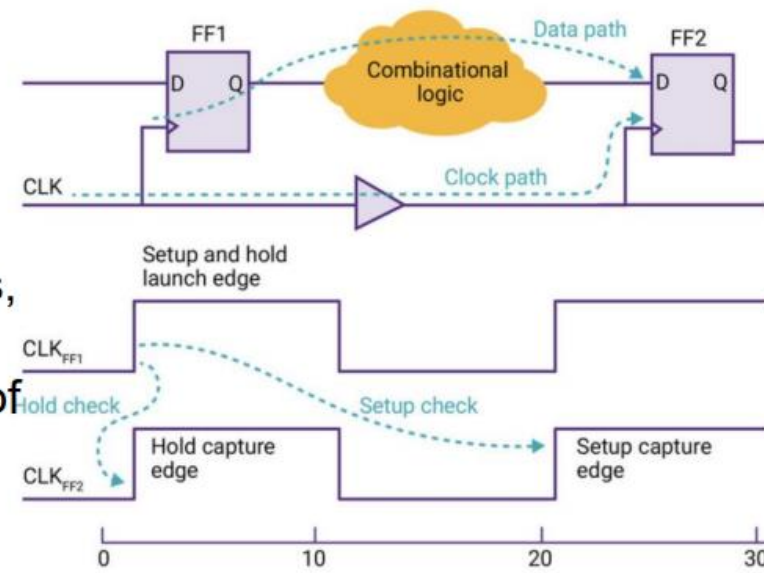


Static Timing Analysis



- ❖ Static timing analysis (STA) is a method of validating the timing performance of a design by checking all possible paths for timing violations
- ❖ STA breaks a design down into timing paths, calculates the signal propagation delay along each path, and checks for violations of timing constraints
- ❖ Algorithms determine the max and min propagation delays of the data path

Setup and hold checks



- ▶ setup slack = $(\text{cycle_time} - t_{ckq} - t_{pd} - t_{su}) - \text{clock_skew}$
- ▶ hold slack = $(t_{ckq} + t_{pd} - t_h) - \text{clock_skew}$

Thanks for listening!