Highly Reliable Counters in FPGAs

MAPLD 2011

Dr Gary R Burke

Jet Propulsion Laboratory

California Institute of Technology

Reliable State Machines

Several methods exist to increase reliability of State Machines beyond TMR

- No undefined states
- Sparse state decoding
  - Hamming distance of 2
    - One hot
    - H2
- Self Correcting
  - H3
  - EDAC on state

Counters in State machines

- Counters are often used with State machines
  - Insert delay between states
    - Large values inefficient to encode into additional states
    - Delay will be eg 1000 clock cycles
  - Slow down State Machine
    - User Interface may need speed of 10Hz
    - Which is a 23-bit counter on 50Mhz clock
  - Repeat count for State machine
    - Sequence repeats a defined number of time
    - More efficient than stretching State machine out.

Counters are Vulnerable

- A Reliable State machine needs a Reliable Counter
- Most efficient Counter is Binary , that is each state is 1 more than previous state.
- The Binary Counter is vulnerable to SEUs
- Although this can be protected using SEU tolerant Flip/flops (eg TMR flip/flops) there is still some vulnerability

Counters are Weak Link

- If the delay between states is incorrect , the FSM will shorten or lengthen the time in a critical state
- A bad Clock Counter can cause the State machine to skip to the next step far too early or late
- A bad repeat count could cause FSM premature completion or case FSM to continue to cycle

Weak Link example

- A state machine turns on pulse to fire Pyro
- The length of the pulse is exactly 14.2 ms
- This length is determined by a counter counting 33Mhz clocks (468600)
- This requires a 19 bit counter
- A SEU on the most significant bit can reduce this count to 206456 ie 6.26 ms
- This can cause partial fire of the pyro

How to fix counters

- Simple solution needed
- Not to add too many f/f s, which become SEU targets
- Assume detection only is needed.
- Assume that only 1 SEU is encountered in entire count sequence.
- Need a way to validate final count value
- If its bad then FSM/system can compensate

Possible solutions

- TMR flip/flops
  - Best choice but some vulnerability remains
- Duplicate counter
  - Large overhead
  - Twice as likely to get SEU

  + Simple logic

  + Works for multiple errors
- Parity
  - May not detect change if checked at end of count
- Monotonicity check

  + Simple logic

  + low overhead

Monotonicity check

- Check that no count values were skipped
- Reached expected count in correct way
- Guarantees count value is good
  - (With previous assumptions)
- Simple implementation
  - Small auxiliary counter
  - or Extra states in FSM
  - or Software check

Auxiliary Counter

- Main counter counts in binary , every clock or under some condition
- Auxiliary counter counts in same way , but is much smaller
- For any size main counter , auxiliary counter is only 2 bits ! (single SEU only)

Why 2 bits?

- Assuming only 1 bit is changed in main counter
- Value of change is +/- 2^i
  - Where i is binary bit position
- Auxiliary counter constrained to count from 0 to 2 (and back to 0)
  - 3 states only
- The expected value of this count at the desired final count value is known (0,1 or 2)

Why it works

- Value of potential error is +/- 2^i (i=0->N-1)
- This is never a multiple of 3
  - It is always even except for i=0, which is always +/- 1;
- An error will be detected by having an incorrect auxiliary counter value at the expected main count value.
- Note this is independent of the size of the main count.

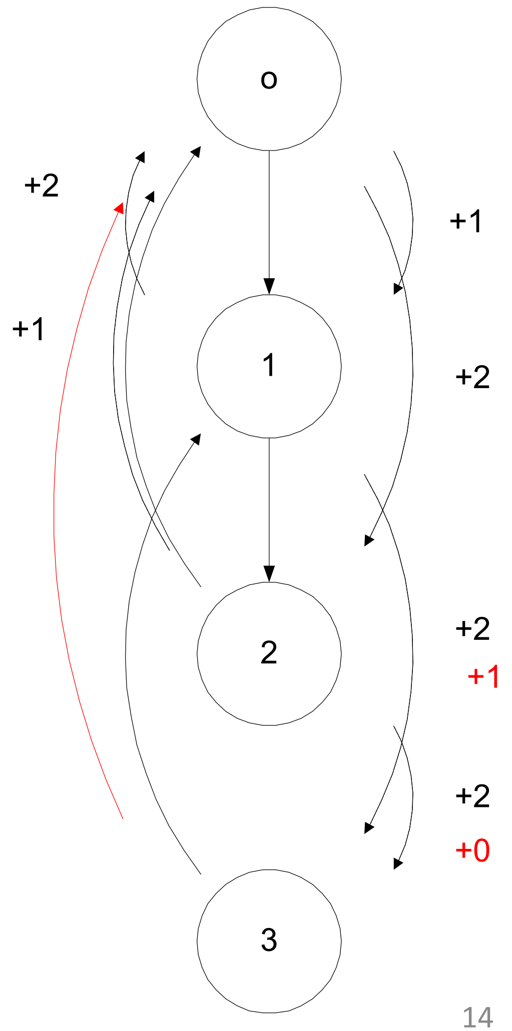
FSM implementation

- it is possible to eliminate the aux counter
- Include 3 states in a loop
- Cycle through these states under the same conditions as the main counter
- At each state check counter has reached desired value
- To be correct the count must be in the correct value, and the FSM in the correct state.

Aux counter errors

Fig.1 Aux Counter State Transition Graph

A sequence of 0,1,2 can be used. In this case a corruption of 1 bit moves the count forward by 1 or 2. This maintains the incorrect state until the end of the count. To achieve this the illegal state (3) should cause a transition to state 1. This is shown in Figure 1. The solid lines are the legal transitions. The dotted lines are the possible transitions due to an SEU. The numbers on the lines represent the increment to the aux counter as a result of these transitions. The red line and text represents what would happen if the illegal 3 state were to transition to a 0 state. Note that in this case a state of 2 could be changed to 3 , and then back to 0, with a net counter increment of 0. This would then be undetected.



multiple errors

- It is possible to tweak this to detect multiple errors , although it becomes less efficient
- If the number of detected errors is E, then for E>1 the size of the aux counter of a function of N , the number of bits in the main counter
- The size of the aux counter depends on the value of X , where X is the number is the number of states in the aux counter
  - Eg for E = 1 then X = 3
  - For E>1 then X = f(N)

maths

- The value V of the errors depends on both E and N
- $V = \sum an * 2^{\wedge} n$
- And V != mod (X)
- Elegant solution to this has not been found.
- X is always prime
- Derive the various values of X from a computer program , which tests all possible error values

Table of aux counter states

Counter Bits	E=1	E=2	E=3	E=4
8	3	19	83	173
9	3	19	83	173
10	3	23	89	307
11	3	23	89	359
12	3	29	139	557
13	3	29	233	653

Conclusions

- Protecting a counter against a single error is feasible with very little overhead , using an aux counter.
- Detecting double errors is also possible , although the size of the aux counter depends on the main counter size
- To detect more than 2 errors on large counters is possible but not competitive with other methods.