Fault Tolerant State Machines

Gary Burke and Stephanie Taft

California Institute of Technology
Jet Propulsion Laboratory
4800 Oak Grove Dr Pasadena CA 91109
gary.r.burke@jpl.nasa.gov
stephanie.taft@jpl.nasa.gov

Abstract

State machines are commonly used to control sequential logic in FPGAs and ASICs. An errant state machine can cause considerable damage to the device it is controlling. For example in space applications, the FPGA might be controlling Pyros, which when fired at the wrong time will cause a mission failure. Even a well designed state machine can be subject to random errors as a result of SEUs from the radiation environment in space. There are various ways to encode the states of a state machine, and the type of encoding makes a large difference in the susceptibility of the state machine to radiation. In this paper we compare 4 methods of state machine encoding and find which method gives the best fault tolerance, as well as determining the resources needed for each method.

1 Introduction

State machines are a useful way to represent logic which is controlling a sequence of events in a digital circuit. State machines are susceptible to faults resulting from radiation. Such faults can alter the state sequence, causing harm to whatever equipment the state machine is controlling. For applications such as spacecraft control, a highly reliable state machine needs to be implemented. Various methods have been proposed to generate such a fault-tolerant state machine. The addition of fault tolerance does come with a price. They will require more resources and will operate slower than regular state machines. In this paper we examine 4 different methods of state machine encoding, comparing the size, performance, and fault tolerance of these methods.

2 State machine encoding

A state machine is a sequential circuit with feedback, used to control the sequence of digital logic. It consists of a state register which is updated on a clock. The outputs of the state register, together with inputs, are used in the next state logic to determine the next state, which will be loaded into the state register on the next clock.
Additional logic takes the state register value and optionally the inputs (depending on type of state machine) and determines a set of outputs. This logic is the 'output logic'.
The state register transitions through a sequence of defined states. Each state has a symbolic representation, for example, S0 or S1. The number of states is determined by the number of unique states needed by the circuit. The size of the state machine register depends on the number of states and the state machine encoding. The state machine encoding is the way in which the symbolic states are actually represented in the state register. As an illustration, four different state machine encodings are shown in Table 1.

State	Binary	One-Hot	Hamming 2	Hamming 3
S0	000	00000001	0000	000000
S1	001	00000010	0011	000111
S2	010	00000100	0101	011001
S3	011	00001000	0110	011110
S4	100	00010000	1001	101010
S5	101	00100000	1010	101101
S6	110	01000000	1100	110011
S7	111	10000000	1111	110100

Table 1: State machine encodings

The simplest state machine encoding is binary. This is a binary count of the state number in sequence. Table 1 shows the binary encoding for an 8-state state machine.

An alternative method of encoding states is 'one-hot'. In one-hot encoding the state encoding is such that only one bit of the state register is set for any state. One-hot is widely used in FGPGAs since it is efficient. Very little logic is needed to determine the next state. Although the state register is larger this is not usually critical in FPGAs, which are considered 'register rich' technologies. However this efficiency relies on an optimization, which

is not appropriate for high reliable logic. For high reliable circuits every state needed to be defined. Because of the large number of flip-flops the total number of states (state space) is very large, and so the resources used for a high reliable one-hot encoded state machine is large.

An alternative encoding is Hamming 2 (H2). In H2-encoding the state machine cycles through states that are guaranteed to be different by 2 bits, that is a hamming distance of 2. This means that a 'single event upset' (SEU) due to radiation can only force the state register to a don't care state. The don't care state can be defined to always bring the state machine to an idle or 'S0' state. Since no incorrect state has been entered, no incorrect outputs have been generated, although the state machine did not complete correctly. Note that this encoding is easily generated by adding a parity bit to the binary encoding. Thus, a 4000-state state machine will require only 13 bits.

To ensure the state machine completes correctly even in the case of a SEU, H3 encoding can be used. In H3 encoding there is a hamming distance of 3 between all states. This means that 3 bits need to change in any state before it will change to another legal state.
In the case of an SEU, the change in state will be to a state which is adjacent and unique to the original state. An SEU in any other legal state will not cause the state register to take on this value. Because of this property, it is possible to generate a state machine which is immune to an SEU change of state. Each legitimate state is defined as a set of values. The set of values contains of the state encoding (see table 1) and every value which has a hamming distance of 1 (adjacent) from the state encoding. For the 8-states in the example in table 1, each set actually consists of 9 values. An SEU will bring the state machine to an adjacent state. From this point the state machine proceeds to finish its sequence normally, and the SEU has no effect.
Generating the H3 encoding is not as easy as generating the H2 encoding, but we have devised an algorithm for generating H3 encoding.

There are many other encoding methods such as grey code. In this code only 1-bit changes for each transition. It is similar in performance to binary, and so is not considered in this paper. It is possible to create codes with greater hamming distance with more fault tolerance. These might be useful if multiple bit errors are expected, but are not considered in this paper.

3 Size and performance of the state machines

The implementation size of each method is significant when resources are limited, and will effect the choice of state machine. Additionally, the maximum speed of each method will be determined by the number of logic levels needed to encode the next state. This may be significant of very high speed is needed. To determine size and performance, the four different state machines were coded in Verilog, and then synthesized in a consistent way to a common target. The results are summarize in tables 2 and 3.The target FPGA used is a Xilinx Spartan 2, with a speed grade of 6.
The advantages of each method does depend on the number of states, so the synthesis was carried out for various numbers of state from 4 to 32.
To ensure consistent results, state machine optimization was turned off, and the state machine flip-flops were not replicated for fanout purposes.

State Machine Size	# Slice Flip Flops	# of 4 input LUTs	Clock Period (ns)	Max Synthesized Frequency (MHz)	Minimum Period (ns)
Binary					
4	2	7	20	272.1	3.7
8	3	15	20	178.8	5.6
12	4	25	20	129.6	7.7
16	4	38	20	122.1	8.2
24	5	50	20	109.6	9.1
32	5	96	20	94.5	10.6
One Hot					
4	4	10	20	238.2	4.2
8	8	20	20	194.8	5.1
32	12	31	20	173.0	5.8
16	16	41	20	148.9	6.7
24	24	63	20	148.9	6.7
32	32	237	20	68.6	14.6

Table 2: Size and performance of binary and one-hot state machines

State Machine Size	# Slice Flip Flops	# of 4 input LUTs	Clock Period (ns)	Max Synthesized Frequency (MHz)	Minimum Period (ns)
Hamming 2					
4	3	8	20	226.6	4.4
8	4	22	20	133.5	7.5
12	5	41	20	124.5	8.0
16	5	49	20	117.8	8.5
24	6	84	20	91.5	10.9
32	6	107	20	87.3	11.5
Hamming 3					
4	5	15	20	162.8	6.1
8	6	42	20	117.4	8.5
12	7	55	20	105.0	9.5
16	7	71	20	102.6	9.8
24	9	91	20	88.7	11.3
32	9	137	20	83.5	12.0

Table 3: Size and performance of H2 and H3 state machines

The resources used were summarized by the number of flip-flops used and the number of look-up-tables (LUTs)used.
At the low number of states the resources used by all methods are very similar. As the number of states increases, it is clear that binary encoding gives the smallest implementation both in number of of flip-flops and number of LUTs. Of the other three encodings, one-hot requires consistently more flip-flops, H3 requires consistently more LUTs. For 32 states one-hot gets anomalously worse than the other methods.
Binary encoding gives the best performance. Although it might be expected that one-hot would be the fastest, this would only be true in the optimized one-hot. Of the remaining 3 encodings, one-hot is fastest followed by H2 and H3. At the 32 state level one-hot is anomalously slower, consistent with its larger use of LUTs requiring more stages of logic.

4 Fault Generator

To test these various encodings for fault tolerance a circuit was devised which could both generate faults, and examine the effect of these faults. This circuit was designed to run on an application board containing a Xilinx Spartan 2 FPGA, with controlling switches and output LEDS. In this way a

large number of faults can be generated very quickly, and it is possible to generate asynchronous faults.

An application circuit was devised, which recognizes a 16-bit key in a serial data stream by stepping through a 16-state state machine. Five state machines are instantiated; one of each type and a reference state machine (binary) which has no faults injected.

A serial input stream will exercise the target state machines. This is implemented using by a pattern generator circuit. The pattern generator circuit generates a sequence of nearly identical 16-bit patterns, with 1 in 16 patterns being the actual key pattern. Between patterns are a number of null states to ensure all state machines are synchronized.

To test for fault tolerance , fault injection is used. This serves as a replacement for a radiation source., and is implemented using a fault generator circuit. The fault generator circuit uses a pseudorandom number generator based on an LFSR circuit. It generates a 16 bit random number (fault number) in 1 out of 16 encoding. The fault number is applied to all the targeted state machines at an interval determined by a rate counter. This rate is determined experimentally, but is chosen to be out of synchronization with the pattern generator.

In the targeted state machine the fault number is used in 3 different ways:

1: Synchronous fault
It is used to change the value of the equivalent bit in its state register for one cycle.

2: Asynchronous fault
The fault number is driven by another clock, and starts randomly through the state machines clock cycle. The fault injection is into the next state logic.

3: Asynchronous pulse
The fault number is initiated at a 'random' time and is constrained to be a short pulse a few nanoseconds long. The fault injection is into the next state logic.

Synchronous fault injection method is the easiest to setup and perform, but is the least realistic. It emulates the case that the 1 bit in the state register has an incorrect value. In practice, it is unlikely that an fault due to an SEU will be synchronized to the state machines clock. Nevertheless, it does provide a reasonable comparison between state machines. Asynchronous fault injection is more realistic, since it occurs at any time during the cycle. This represents the case that an SEU has flipped a bit in the state register at a random time.

Asynchronous pulse fault injection emulates a radiation induced pulse occurring in the next state logic. This is rare, since such an event is absorbed by the circuit loading, or is ignored if it doesn't coincide with the clock at the input to a flip-flop. Practical considerations make the pulse generated in this test circuit much longer than the pulse generated by radiation.

Although single event upsets are the most interest, the fault injector is capable of generating 2 random faults at a time in 2 out of 16 encoding. This is useful for evaluating circuits which have no or little response to single errors.

To determine how many errors are generated by the fault injection, a monitoring circuit is implemented. The monitoring circuit records errors in 2 categories

1: The reference state machine matches a pattern but the target state machine does not. This is the false negative case.

2: The reference state machine does not match the pattern but the target state machine erroneously indicates a match. This is the 'false positive' case.

5 Fault injection results

The test circuit was downloaded to the application board and run with various options. The error totals were noted. In each case the experiment was stopped when the number of errors on any state machine reached 255.

5.1 Synchronous Fault Injection

The chart Fig 1 shows the error rate for the four encoding methods. Not shown is the reference state machine which has no fault injection and thus no errors. For each state machine the four bars represent the error rate of the single fault false-negative, single fault false positive, double fault false negative and double fault false positive results, respectively.

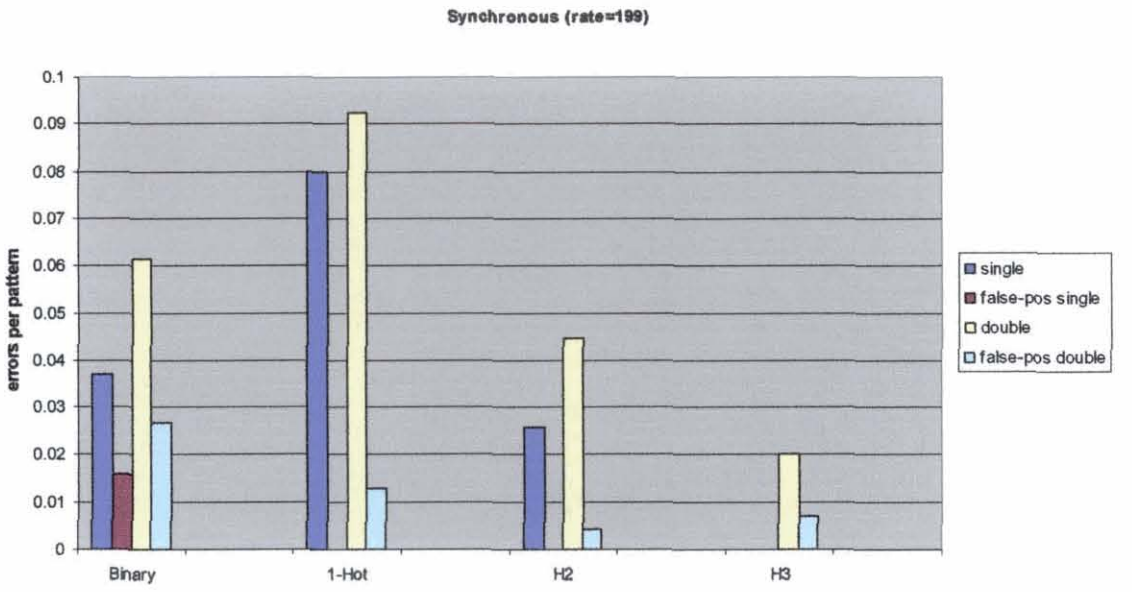

Fig 1: Synchronous Fault injection results

As might be expected the one-hot encoding is most susceptible to faults as it has the most flip-flops. It has about 2X the fault rate of binary encoding. Only the binary encoding is susceptible to single fault false positive errors. The reason is clear: binary encoding has no redundant states. A fault in the state register will force the state machine to an incorrect state, which may indicate that the key pattern is matched.

As far as single false errors are concerned, H3 has no errors at all. This is expected, since any single error will not effect the state machine. Other than H3, the H2 method has the least errors.

For double errors H3 remains the least effected. For double errors all methods give false positive errors, which is to be expected.

5.2 Asynchronous Fault injection

The asynchronous fault injection results (Fig 2) are very similar to the synchronous fault injection results. The absolute error rates are lower, since the injector is less likely to produce a fault at a critical time. The H3 method has relatively higher error rates for double faults, (compared with the last chart). It might be expected that the asynchronous injection of single faults will result in double faults effecting the H3 encoding, but in fact the H3 encoded state machine continues to have zero errors.

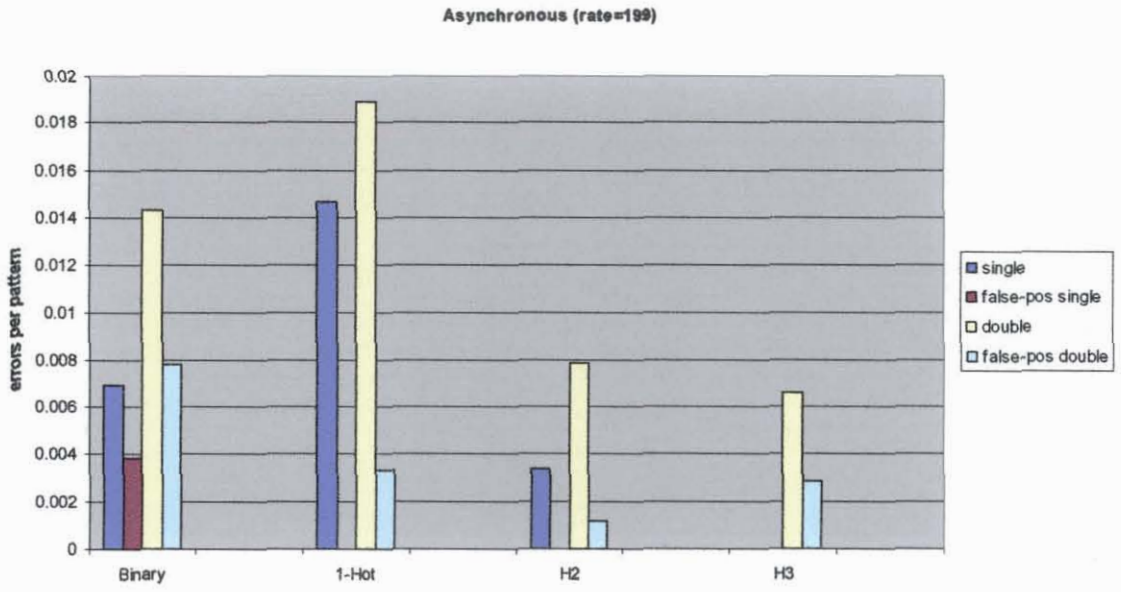

Fig 2: Asynchronous Fault injection results

5.4 Asynchronous Pulse Fault Injection

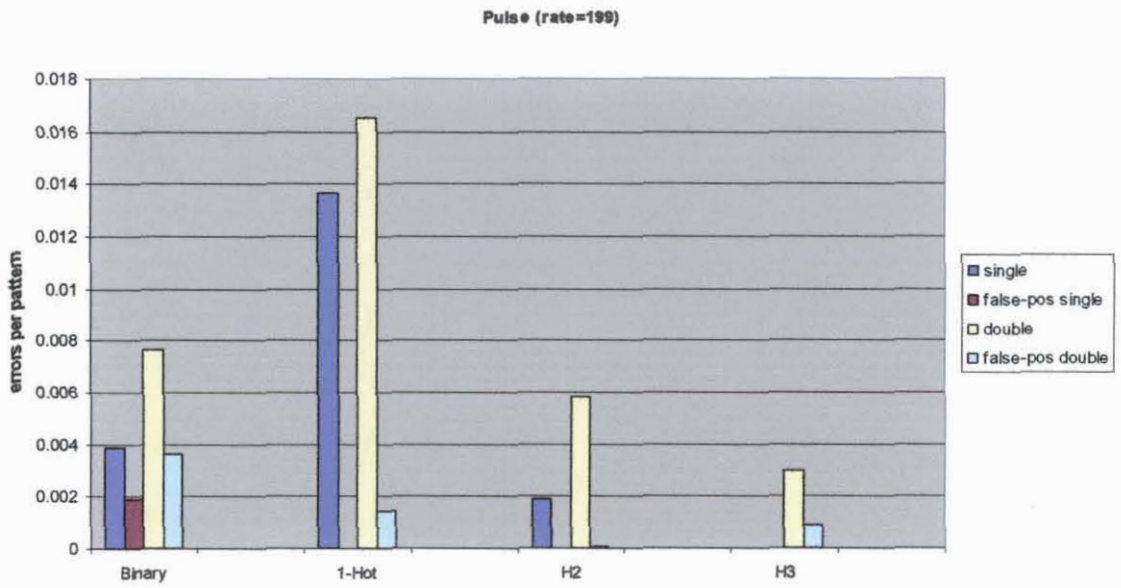

Fig 3: Asynchronous Pulse Fault injection results

The absolute values of the errors (fig 3) are less as the pulsed fault is less likely to be active at a critical time. The one-hot method performs relatively

poorly, with over 3X as many errors as binary. There are zero errors in the H3 encoded state machine for single faults, despite the asynchronous nature of the pulsed fault.

6 Conclusions

Despite the unrealistic nature of synchronous fault injection, it does give a relative measure of the susceptibility of various state machines to random faults. The Hamming-3 (H3) encoding gives by far the best fault tolerance, recording 0 errors in single fault injection tests. However it requires the most resources, and is the slowest of the encoding methods.
Hamming-2 (H2) encoding has less errors than binary encoding, and has zero 'false positive' errors.
One-hot has the most errors, due to its large number of target flip-flops. One-hot has also poor use of resources, and slow speed, although it is better than binary for false-positive errors.
For fault tolerant designs H2 is the best compromise in terms of size, speed and fault-tolerance. H3 is preferred when ultimate fault tolerance is required.

7 Acknowledgments

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration