Automated FSM Error Correction for Single Event Upsets

Nand Kumar and Darren Zacher
Mentor Graphics Corporation
nand_kumar{darren_zacher}@mentor.com

Abstract

This paper presents a technique for automatic error correction of finite state machines (FSMs) from single event upsets (SEUs). We present a general technique of adding Hamming error-checking bits to automatically recover from single-bit errors within the same clock cycle. Experimental results on field programmable gate array (FPGA) devices demonstrate the efficacy of the method.

1 Introduction

FPGAs are increasingly being used for mission-critical applications in hazardous operating environments (space, military, medical etc.). In these applications, the circuit must be fault tolerant, especially finite state machines (FSMs), where failures are very hard to detect [1]. Most single-bit fault-tolerant FSMs are implemented using triple module redundancy (TMR) or by using a single-error-correcting (SEC) code during state encoding [6, 7]. Typical instances of SEC employ a minimal encoding (binary, Gray, etc.) scheme to force a Hamming [2] distance of three. Commercially available FPGA synthesis tools [8, 9] only implement error recovery to a specified recovery state. They do not implement error correction. In this paper, we investigate a general technique of adding Hamming error-checking bits to any encoding style of FSMs to automatically recover from single event upsets (SEUs) within the same clock cycle. Area and delay results of using this technique on selected designs are also reported.

Section 2 introduces the FSM model and describes the effect of SEUs. Section 3 describes the use of Hamming error-checking bits to automatically correct single-bit errors. The experimental technique is presented in Section 4. Results on selected FSM examples are presented in Section 5. Section 6 discusses the current status and future work.

2 FSM Model and SEUs

An FSM is defined to be a 6-tuple [10]:

$M = (I, O, S, \delta, \lambda, s_0)$

where:
I is a finite nonempty set of inputs; O is a finite nonempty set of outputs; S is a finite nonempty set of states;
$\delta : I \times S \rightarrow S$ is the state transition function; $\lambda : I \times S \rightarrow O$ is the output function; and s_0 is the initial state.
Figure 1 shows the hardware model of an FSM. An FSM is typically modeled as a set of state registers to store the state

vector, combinational state decode for the next-state and output logic.

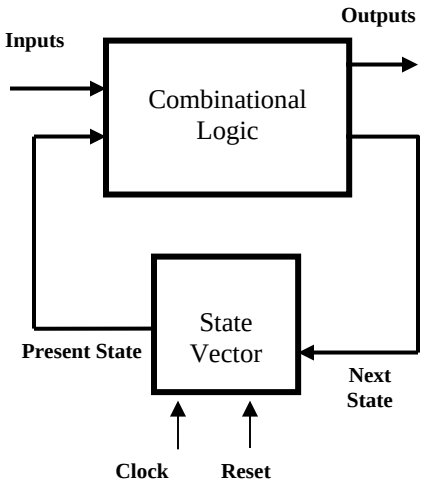

Figure 1: FSM model.

Consider an FSM with five states (S_0, S_1, S_2, S_3, and S_4), encoded using a binary or sequential encoding. The encodings are displayed in Table 1:

Table 1: Example of FSM encoding.

State	Encoding
S_0	000
S_1	001
S_2	010
S_3	011
S_4	100

If the FSM state register experienced an SEU, say in state S_0, it could place the FSM in states S_1, S_2, or S_4 depending on which bit of the state-register experienced the upset. It is hard to determine if the current state is a result of a valid state transition or the result of an SEU. It is hard to recover from such upsets because the upset actually placed the FSM in a legal state. If the FSM had transitioned from S4 to states 101 or 110, it would have been easier to detect the illegal transition and recover from the illegal state.

3 Error Correction

We use a Hamming error-correcting code [1, 2, 3, 10]. To the n encoding bits, k parity-checking bits are added to form an $(n + k)$-bit code. The location of each of the bits in the new code is assigned a decimal value starting at 1 for the most significant bit and $n + k$ to the least significant bit. The k parity checks are performed on selected bits of each encoding. The result of each parity check is recorded as 1 if an error has been detected or 0 if no error has been detected.

The number of parity bits k must satisfy the inequality $2^k \geq n + k + 1$. The parity checks are performed such that their value when an error occurs is equal to the decimal value assigned to the location of the erroneous bit, and is equal to zero if no error occurs. This number is called the *position* number. The parity-checking bits are placed in positions 1, 2, 4, …, 2^{k-1} such that they are independent of each other and encoded only in terms of the encoding bits. Considering the example in Table 1, the original encoding uses three bits (n=3), so k = 3 satisfies the above inequality. Thus, three parity bits must be added to the encoding bits to generate the error-correcting code.

The *position* numbers are shown in Table 2. From the table we observe that an error in position 1, 3, or 5 should result in a 1 in the least significant bit (c_0) of the position number. Hence, the code must be designed to have digits in position 1, 3, and 5 to have even parity. If a parity check of these bits shows an odd parity, the corresponding position number bit (c_0) is set to 1, otherwise to 0. Also, from the table we observe that an error in position 2, 3, or 6 should

result in the center bit (c_1) being set to 1 and an error in position 4, 5, 6 should result in the most significant bit (c_2) being set to 1.

Table 2: Position numbers.

Error position	Position number
	c_2 c_1 c_0
0 (no error)	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0

The parity bits are chosen as follows:
p_1 is selected to establish even parity in bits 1, 3, 5.
p_2 is selected to establish even parity in bits 2, 3, 6.
p_3 is selected to establish even parity in bits 4, 5, 6.

Table 3: Hamming code for FSM.

Position	1	2	3	4	5	6
	p_1	p_2	e_2	p_3	e_1	e_0
	0	0	0	0	0	0
	0	1	0	1	0	1
	1	0	0	1	1	0
	1	1	0	0	1	1
	1	1	1	0	0	0

Table 3 shows the Hamming code for the example FSM from Table 1.

The error location and correction is performed by computing the parity checks and determining the position number. For example, if we assume that bit zero of the encoding (e_0) of state S_0 is inverted (SEU), the new code is: 000001. The three parity checks yield the following position number bits:

4,5,6 parity check: 0 0 1; parity check is odd; $c_2 = 1$.
2,3,6 parity check: 0 0 1; parity check is odd; $c_1 = 1$.
1,3,5 parity check: 0 0 0; parity check is even; $c_0 = 0$.

The position number $c_2c_1c_0$ is 110, which means that the location of the error is in position 6. To correct the error, the bit in position 6 is inverted, and the correct encoding 000000 is obtained.

The error correction outlined in this section has been implemented to handle any encoding style and is automatically inserted for FSMs.

4 Experimental Technique

We collected seven sample FSM designs (in VHDL) with varying complexity with regards to number of states, inputs, and outputs. The FSMs (Table 4) employed a mixture of one-hot and Gray state encoding.

Table 4: Example FSM characteristics.

FSM	States	Inputs	Outputs	Encoding
fsm1	5	12	10	One-hot
fsm2	6	38	44	One-hot
fsm3	7	105	2	Gray
fsm4	9	301	35	Gray
fsm5	6 & 10	7	5	One-hot & Gray
fsm6	23	190	45	Gray
fsm7	30	196	71	Gray

We synthesized each FSM targeting the Actel A54SX72A-STD device. During synthesis, we added error correction to the FSMs. After logic synthesis, we wrote a VHDL description with the correction logic. Based on our selection of an antifuse programmable logic fabric, we chose single-voter TMR mapping for comparison. We generated a test bench to simulate the RTL description against the synthesized netlist. Both circuits were fed by a common set of 1,000 random test vectors, and on each clock cycle compared the outputs of the RTL description and the synthesized netlist.

We then repeated the simulation and introduced a series of SEUs by forcing each one of the state-bits in the synthesized netlist, one at a time, to a constant value. We compared the outputs of the RTL again and synthesized circuit descriptions. We were able to verify that the outputs were identical due to the single-bit error being corrected. We then repeated the simulation again, now forcing each one of the parity bits, one at a time; we verified the same behavior, due to the single-bit error again being corrected.

We then forced two state-bits (two parity bits) in the synthesized design and re-simulated; now, we observed differences in outputs between the RTL and synthesized circuit descriptions. The two-bit errors actually produced functional errors whereas each of the single-bit errors was corrected.

The other experiment compared the area and delay penalties of this technique with the more common usage of single-voter TMR.

5 Results

The performance penalties in terms of area and F_{max} of TMR and Hamming encoding, for the FSMs' original one-hot and Gray encodings as well as for minimal FSM encodings, are presented below.

Figure 2 compares the sequential area penalties. Notice that in general, TMR error correction results in a higher sequential area penalty; this penalty tends to be more significant when the FSM uses a one-hot state encoding.

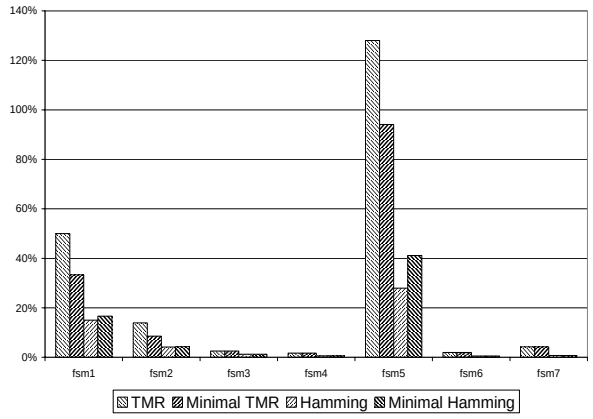

Figure 2: Sequential area penalties.

Figure 3 compares the combinatorial area penalties. Notice that in general, Hamming error correction results in a higher combinatorial area penalty; this penalty also tends to be more significant when the FSM uses a one-hot state encoding.

Also, note that in some cases, the combinatorial area actually decreased when using Hamming encoding; this could be due to the synthesis tool exploiting resource sharing on the state decode logic.

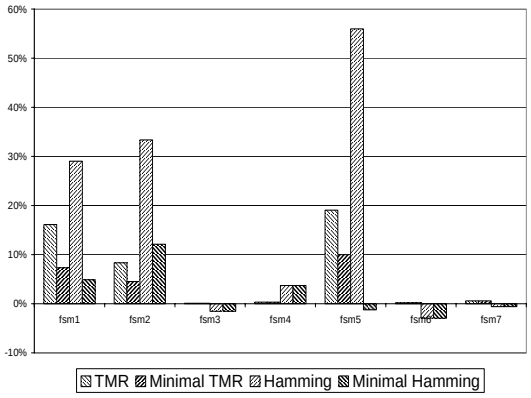

Figure 3: Combinatorial area penalties.

Figure 4 compares the aggregate area penalties. Notice that aggregate area penalties are more similar when comparing TMR with Hamming correction, though there is clear benefit in using a minimal state encoding regardless of the correction scheme used.

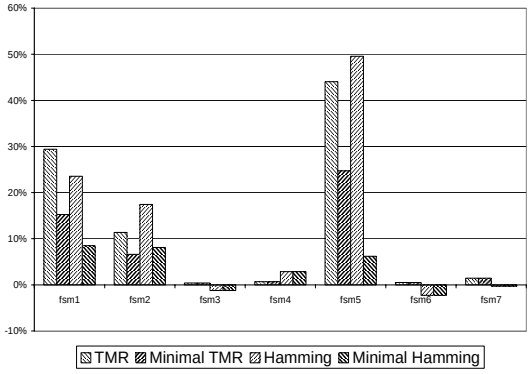

Figure 4: Aggregate area penalties.

Figure 5 compares the clock frequency penalties. Notice that clock frequency penalties are consistently greater when using Hamming correction. Using a minimal state encoding does tend to reduce the clock frequency penalty regardless of error correction strategy. Interestingly, a few of the circuits actually experienced an increase in clock frequency by using error correction.

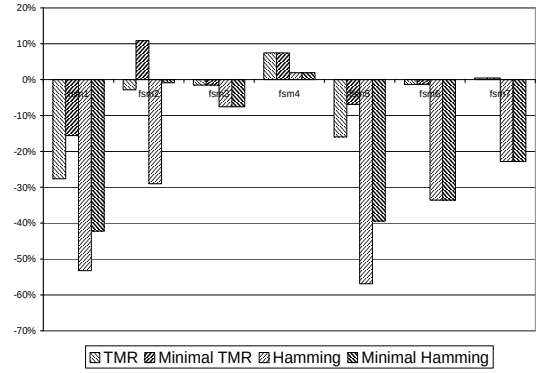

Figure 5: Clock frequency penalties.

Table 5 summarizes the area and F_{max} penalties for the various state encodings explored. TMR exhibits the highest sequential area penalty, while Hamming exhibits the largest combinatorial area and clock frequency penalties. In most cases, penalties were reduced by using a minimal FSM encoding.

Table 5: Area and F_{max} penalties.

	Chg Area			Chg F_{max}
	Seq	**Comb**	**Agg**	
TMR	29%	6%	13%	-6%
Minimal TMR	21%	3%	7%	-1%
Hamming	7%	17%	13%	-29%
Minimal Hamming	9%	2%	3%	-21%

6 Discussion

Results (Table 6-11) suggest that Hamming FSM encoding is generally an acceptable alternative to TMR, and is more area efficient, especially when using a minimal FSM encoding. However, the F_{max} penalty suggested is greater for Hamming FSM encoding than for single-voter TMR. Results also indicate that the F_{max} penalty for Hamming FSM encoding is much larger when using one-hot state encoding.

One extension to this application involves adding one or more additional Hamming parity bits to allow for detection of double or multiple event upsets. This method poses significant benefit over TMR, particularly in higher radiation environments, where a double event upset affecting two registers feeding the same voting circuit could otherwise result in a functional interrupt.

Another extension to this application is to add a metastability filter to the error detection, and use a multiplexer to force a recovery state rather than using the full correction circuit [11]. This changes the behavior from error correction to error recovery. The FSM would recover within the N clock cycles it takes for the error-detection pipeline.

This method has two clear benefits: first, the error detection circuit can now be made a false path; second, the error detection scheme can be easily applied to all encoding styles to detect even double or multiple event upsets.

References

[1] S. Leveugle, L. Martinez, "Design methodology of FSMs with intrinsic fault tolerance and recovery capabilities," IEEE Proc. EuroASIC'92, 1992, pp. 201 – 206.

[2] R.W. Hamming, "Error Detecting and Error Correcting Codes," The Bell System Technical Journal, Vol. 29, April 1950, pp. 147 – 160.

[3] D.B. Armstrong, "A general method of applying error correction to synchronous digital systems," The Bell System Technical Journal, Vol. 40, No. 2, March 1961, pp. 577 – 593.

[4] J.F. Meyer, "Fault-tolerant sequential machines," IEEE Trans. On Computers, Vol. C-20, No. 10, October 1971, pp. 1167 – 1177.

[5] R Leveugle, "Optimized state assignment of single fault tolerant FSMs based on SEC codes," Proc. 30th DAC, 1993, pp. 14 – 18.

[6] C. Bolchini, R. Montandon, F. Salice, and D. Sciuto, "A State Encoding For Self-Checking Finite State Machines," Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95., IFIP International Conference on Hardware Description Languages; IFIP International Conference on Very Large Scale Integration., Asian and South Pacific, 1995, pp. 711 – 716.

[7] R. Rochet, R. Leveugle, and G. Saucier, "Analysis and Comparison of Fault Tolerant FSM Architecture Based on SEC Codes," Proc. IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 1993, pp. 9 – 16.

[8] Precision™ Synthesis Reference Manual, Mentor Graphics Corp. 2004.

[9] Synplify Pro™ Reference Manual, Synplicity Inc. 2003.

[10] Z. Kohavi, Switching and Finite Automata Theory, McGraw-Hill Book Company.

[11] M. Berg, "A Simplified Approach to Fault Tolerant State Machine Design for Single Event Upsets," Mentor Graphics Users' Group User2User Conference, 2004.

Appendix

Table 6: Area and F_{max} for sample FSMs, in their original FSM encodings.

FSM	States	Encoding	Original			
			Area			F_{max}
			Seq	Comb	Agg	
fsm1	5	One-hot	20	31	51	114.32
fsm2	6	One-hot	72	60	132	65.75
fsm3	7	Gray	238	1742	1980	19.87
fsm4	9	Gray	467	1272	1739	25.12
fsm5	6 & 10	One-hot & Gray	25	84	109	72.98
fsm6	23	Gray	506	2186	2692	16.35
fsm7	30	Gray	427	1392	1819	24.33

Table 7: Comparing area and F_{max} for TMR mapping against original encodings.

FSM	States	Encoding	TMR							
			Area			F_{max}	Chg Area			Chg F_{max}
			Seq	Comb	Agg		Seq	Comb	Agg	
fsm1	5	One-hot	30	36	66	82.72	50.0%	16.1%	29.4%	-27.6%
fsm2	6	One-hot	82	65	147	63.9	13.9%	8.3%	11.4%	-2.8%
fsm3	7	Gray	244	1744	1988	19.57	2.5%	0.1%	0.4%	-1.5%
fsm4	9	Gray	475	1276	1751	26.99	1.7%	0.3%	0.7%	7.4%
fsm5	6 & 10	One-hot & Gray	57	100	157	61.33	128.0%	19.0%	44.0%	-16.0%
fsm6	23	Gray	516	2190	2706	16.13	2.0%	0.2%	0.5%	-1.3%
fsm7	30	Gray	445	1400	1845	24.45	4.2%	0.6%	1.4%	0.5%
							28.9%	**6.4%**	**12.6%**	**-5.9%**

Table 8: Comparing area and F_{max} for Hamming encoding against original encodings.

FSM	States	Encoding	Hamming							
			Area			F_{max}	Chg Area			Chg F_{max}
			Seq	Comb	Agg		Seq	Comb	Agg	
fsm1	5	One-hot	23	40	63	53.46	15.0%	29.0%	23.5%	-53.2%
fsm2	6	One-hot	75	80	155	46.67	4.2%	33.3%	17.4%	-29.0%
fsm3	7	Gray	241	1715	1956	18.37	1.3%	-1.5%	-1.2%	-7.5%
fsm4	9	Gray	470	1319	1789	25.61	0.6%	3.7%	2.9%	2.0%
fsm5	6 & 10	One-hot & Gray	32	131	163	31.46	28.0%	56.0%	49.5%	-56.9%
fsm6	23	Gray	509	2122	2631	10.86	0.6%	-2.9%	-2.3%	-33.6%
fsm7	30	Gray	430	1384	1814	18.78	0.7%	-0.6%	-0.3%	-22.8%
							7.2%	**16.7%**	**12.8%**	**-28.7%**

Table 9: Area and F_{max} for same sample FSMs, in minimal FSM encodings.

FSM	States	Encoding	Minimal			
			Area			F_{max}
			Seq	Comb	Agg	
fsm1	5	Binary	18	41	59	112.36
fsm2	6	Binary	70	66	136	59.44
fsm3	7	Gray	238	1742	1980	19.87
fsm4	9	Gray	467	1272	1739	25.12
fsm5	6 & 10	Binary & Gray	17	80	97	70.09
fsm6	23	Gray	506	2186	2692	16.35
fsm7	30	Gray	427	1392	1819	24.33

Table 10: Comparing area and F_{max} for TMR mapping against minimal encodings.

FSM	States	Encoding	Minimal TMR							
			Area			F_{max}	Chg Area			Chg F_{max}
			Seq	Comb	Agg		Seq	Comb	Agg	
fsm1	5	Binary	24	44	68	94.87	33.3%	7.3%	15.3%	-15.6%
fsm2	6	Binary	76	69	145	65.9	8.6%	4.5%	6.6%	10.9%
fsm3	7	Gray	244	1744	1988	19.57	2.5%	0.1%	0.4%	-1.5%
fsm4	9	Gray	475	1276	1751	26.99	1.7%	0.3%	0.7%	7.4%
fsm5	6 & 10	Binary & Gray	33	88	121	65.25	94.1%	10.0%	24.7%	-6.9%
fsm6	23	Gray	516	2190	2706	16.13	2.0%	0.2%	0.5%	-1.3%
fsm7	30	Gray	445	1400	1845	24.45	4.2%	0.6%	1.4%	0.5%
							20.9%	**3.3%**	**7.1%**	**-0.9%**

Table 1: Comparing area and F_{max} for Hamming encoding against minimal encodings.

FSM	States	Encoding	Minimal Hamming							
			Area			F_{max}	Chg Area			Chg F_{max}
			Seq	Comb	Agg		Seq	Comb	Agg	
fsm1	5	Binary	21	43	64	64.96	16.7%	4.9%	8.5%	-42.2%
fsm2	6	Binary	73	74	147	58.91	4.3%	12.1%	8.1%	-0.9%
fsm3	7	Gray	241	1715	1956	18.37	1.3%	-1.5%	-1.2%	-7.5%
fsm4	9	Gray	470	1319	1789	25.61	0.6%	3.7%	2.9%	2.0%
fsm5	6 & 10	Binary & Gray	24	79	103	42.46	41.2%	-1.3%	6.2%	-39.4%
fsm6	23	Gray	509	2122	2631	10.86	0.6%	-2.9%	-2.3%	-33.6%
fsm7	30	Gray	430	1384	1814	18.78	0.7%	-0.6%	-0.3%	-22.8%
							9.3%	**2.1%**	**3.1%**	**-20.6%**