BFIT Users Guide - v1.1

Daniel Holcomb   Wenchao Li   Sanjit A. Seshia
UC Berkeley
{holcomb,wenchaol,sseshia}@eecs.berkeley.edu
February 9, 2009

Contents

1. Introduction **2**

2. FIT Calculation **2**
2.1. $R(q,t)$ - the probability of observing different charges at different times. 2
2.2. $N(q,t)$ - the errors induced by different charges at different times 3
2.3. Determining Factors of FIT 3
2.4. Precharacterization 4

3. BFIT Algorithm **6**

4. Accuracy **7**

5. Using BFIT **8**
5.1. input netlist 8
5.2. input vectors 9
5.3. output report 9
5.4. Runtime 9

6. Results from ISCAS'89 benchmarks **10**

7 Appendix **11**
7.1 Included Benchmark files 11
7.2 BFIT source code 11

Feedback on this tool should be directed to

```
Dan Holcomb
EE Graduate Student - UC Berkeley
holcomb@eecs.berkeley.edu
http://www.eecs.berkeley.edu/~holcomb/
```

If you have any questions, comments, or suggestions for improvement, please contact me.

1. Introduction

BFIT is a part of the Verification-Guided Error Resilience (VGER) project at UC Berkeley. BFIT is a tool designed for analyzing the neutron FIT (failure-in-time, per 10^9 hours) contribution of each combinational gate in a sequential circuit.

The input is a combinational circuit netlist, and the output is the FIT of each gate in that netlist, broken down according to the various set of latches that can be upset. The generic netlist is internally mapped onto gates from the Nangate 45nm open cell library[1] for analysis[2].

The vectors applied to the circuit can randomly generated, or provided as a text file. The output of BFIT is a text file showing the FIT contribution of every gate in the circuit, with the FIT of each gate down broken down according to what set of latches it upsets. BFIT can quickly analyze thousands of input vectors on circuits exceeding 20,000 gates.

BFIT can be used as part of a methodology for efficient hardening of sequential circuits [20].

2. FIT Calculation

The FIT of any specified event is given by Eq. 1; an event is a strike to a particular gate causing an upset in some exact set of one or more latches.

$$FIT = \frac{failures}{10^9 hours} = avg\left(\frac{failures}{cycle}\right) \times \frac{cycles}{10^9 hrs} \tag{1}$$

Failures per cycle is based on knowing which strikes can cause that event, and knowing what the probability of those strikes occurring. Any neutron strike to the node can be described by a collected charge q and a time t; the space of all possible strikes to a node is then $q \times t$. A boolean valued function $N(q,t)$ takes the value 1 for strikes that cause a specified error, and 0 otherwise; different subscripts will be applied to $N(q,t)$ to denote various events. A real-valued function $R(q,t)$ represents the probability that a strike of exactly charge q will occur at exactly time t of the clock cycle. Numerically integrating over all q, t yields failures per cycle (Eq. 2) [3].

$$\frac{failures}{cycle} = \int_{q=0}^{\infty} \int_{t=0}^{t_{cycle}} R(q,t) N(q,t)\; dt\; dq \tag{2}$$

Strike Model. A Neutron strike frees charge that can be collected into some circuit node through drift. It is modeled as a current injection of the form of Eq. 3, as proposed by Freeman [7] and adopted in subsequent work [9, 17, 24].

Different magnitudes of strikes are modeled by scaling the injected current. The total collected charge q of a strike is the integral of the injected current. A strike to an NMOS diffusion will collect electrons (creating a negative voltage glitch), and the P-type diffusion of PMOS devices will collect holes (creating a positive voltage glitch)[4]. Technology-dependent parameter τ is the time constant of the injected current; τ is set to 20ps for both PMOS and NMOS devices in this work, as is extrapolated to be appropriate for sub-100nm technology by Hazucha et al. [9].

$$I(t) \propto \frac{1}{\tau}\sqrt{\frac{t}{\tau}} \exp\left(\frac{-t}{\tau}\right) \tag{3}$$

2.1. $R(q,t)$ - the probability of observing different charges at different times.

Different magnitudes of strikes occur with different frequencies; strikes producing small charges are more frequent than large charges. Large diffusion areas collect more charges than small diffusions. Based on the work of Hazucha et al. [9] (and further adopted in [24, 17]), the number of strikes per second exceeding charge q is given by equation 4.

$$R(q) = F \times A \times K \times \exp\left(-\frac{q}{Q_s}\right) \tag{4}$$

- F is the neutron flux ($= 56.5/m^2 * s$ in New York City [26]
- A is the sensitive drain diffusion area of the node collecting the charge. We assume that the diffusion of a nominal inverter is 90nm by 90nm;
- K is a technology-independent fitting parameter ($= 2.2 * 10^{-5}$ [9]);
- Q_s is a technology-dependent fitting parameter representing charge collection efficiency, and differs for NMOS (17fC) and PMOS (6.5fC) devices [9].

[1] http://www.nangate.com/
[2] In other words, we use timing and electrical models obtained from the Nangate cells
[3] note that $N(q,t)$ is essentially just selecting the bounds over which to integrate $R(q,t)$
[4] in BFIT, The polarity of the injected current accounts for this

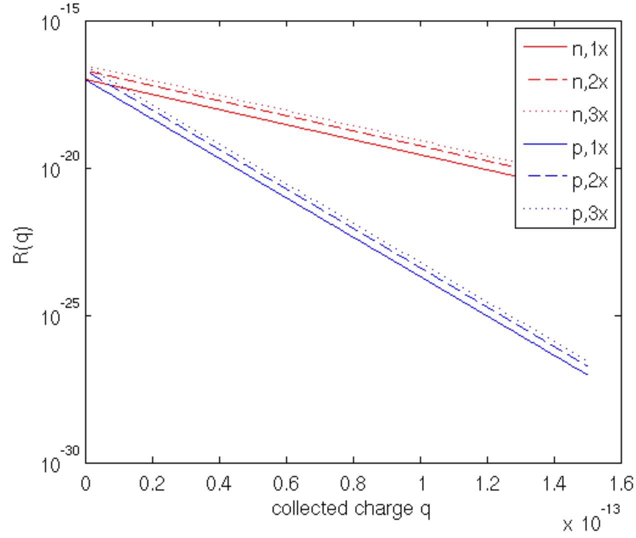


Figure 1: Strikes per second of various charges for min, 2x minimum and 3x minimum sized diffusions of each N and P type. Strikes to N-type diffusion are more frequent because of the collection efficiency parameter Q_s.

Assuming that a strike is equally likely to occur at any time in the clock cycle, the number of strikes depositing charge q occurring at exactly time t of the clock cycle is given by $R(q,t)$ in Eq. 5.

$$R(q,t) = t_{step} * \frac{dR(q)}{dq} \tag{5}$$

2.2. $N(q,t)$ - the errors induced by different charges at different times

Any event can be described by a $N(q,t)$ function; BFIT estimates the $N(q,t)$ functions for different events using logical operations over the $N(q,t)$ functions of sensitized paths. Note that the function $N(q,t)$ subsumes all masking factors: timing masked and electrically masked strikes are those (q,t) described by $N(q,t) = 0$; logically masked strikes have $N(q,t) = 0$ for all q, t.

We proceed top down, from logical events to circuit paths.

To find the number of failures in which a gate g causes exactly set E of latches to fail, subscripts are applied to Eq. 2, generating Eq. 6.

$$\frac{failures}{cycle}_{g \to E} = \int_{q=0}^{\infty} \int_{t=0}^{t_{cycle}} R(q,t) * N_{g \to E}(q,t) dt dq \tag{6}$$

For an error to be caused in exactly set E, it must be true that there is an error in each latch from set E and that all other latches are error-free (Eq. 7). Although the number of possible latch sets is exponential in the number of latches, errors are only observed in a small subset of the possible latch sets.

$$N_{g \to E}(q,t) = \bigwedge_{\forall l_i \in E} N_{g \to l_i}(q,t) \wedge \bigwedge_{\forall l_i \in L-E} \overline{N_{g \to l_i}(q,t)} \tag{7}$$

For an error to be caused in any latch l, we assume that there is one or more paths π that are solely capable of causing the error. This assumption could introduce could introduce error; Zhang et al. propose that the impact of this assumption is minimal [22] in their work.

$$N_{g \to l_i}(q,t) = \bigvee_{\forall \pi : g \to l_i} N_{\pi}(q,t) \tag{8}$$

2.3. Determining Factors of FIT

The local state of a struck gate important because it determines the area and type of sensitized diffusion, and because it determines the electrical drive strength of the gate. The type and area of sensitized diffusion factor into $R(q,t)$ and the electrical drive strength factors into $N(q,t)$.

Four important factors for determining FIT are presented here. A NAND2 gate is used for illustration.

Sensitized Diffusion Area.

The probability of observing a certain magnitude of collect charge depends on the sensitized diffusion area of that node (See Eq. 6). For a diffusion to be sensitized, it must be reverse-biased and connected to the gate output. We make the assumption that stacked devices are all upsized according to the following rule: in a double stack devices are double wide, in a triple stack they are triple wide, etc. In a NAND2 gate, a difference in sensitized diffusion area causes the 10 state to have double the FIT of the 01 state (Fig. 2). Fig. 3 shows that 01 and 10 input states have similar $N(q,t)$ functions on account of similar pull-up strength (each via a single PMOS device), meaning that the same size charge is needed to flip them.

Diffusion Type.

When a gate output is high, only NMOS diffusion will be sensitized, and when low only PMOS diffusion will be sensitized. This is significant because PMOS devices are struck less frequently (See Eq. 4 and Fig. 1). In a NAND2, the 11 state contributes the least FIT, because P-type diffusion is sensitized, and strikes to P-type diffusion are less frequent, per Eq. 4.

Electrical Strength.

When a gate is struck, the injected current charges a capacitance to create and propagate a voltage glitch. In CMOS logic, each gate output is always connected to ground or supply by a pull-up (or pull-down) path; the injected charge must be of sufficient magnitude to overcome that pull-up (-down) path in creating an erroneous voltage. The strength of this pull-up (-down) path is a function of the gate input [13, 8]. The electrical fanout of the gate being struck could also be important. In a NAND2 gate, the 00 state requires a larger strike to cause an error (See Fig. 3) , due to the stronger pull-up (via parallel PMOS devices - see Fig. 2).

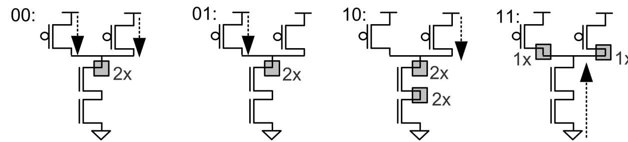


Figure 2: The input applied to a gate determines its sensitive area and its drive strength.

Path Delay.

Studies have shown that soft error-induced transient glitches can propagate through logic in sub 100nm technologies without being significantly attenuated [14, 6]. In older technologies, soft error-induced glitches typically could not propagate through more than a few gates before being attenuated away [6].

Without significant attenuation, a gate close to a latch should contribute the same amount of FIT as a gate far from the latch (all other things being equal). However, the delay of the path will shift the $N(q,t)$ function in time, since a strike propagating down a long path must come appropriately earlier relative to the clock edge (Eq. 9). Later in the manual we will show experimentally that abstraction of path delay as a timeshift to $N(q,t)$ does not induce significant error.

$$N_{\pi}(q,t) = \hat{N}(q, t - d_{\pi}) \tag{9}$$

The only significant path dependence is in the timing of the $N(q,t)$ function. For longer paths, the $N(q,t) = 1$ region is earlier in the clock cycle (Fig. 3). Since the time of a strike is uniformly distributed over the clock cycle, the time shift does not effect the failures per cycle, so the far gates contribute roughly as much failures as the close gates.

2.4. Precharacterization

If FIT does not depend on electrical properties of a path, then it depends only on local effects (input state, load capacitance). The space of possible local effects can be precharacterized.

For each gate type, input state, and load capacitance, SPICE simulation is used to precharacterize its delays, and nominal $N(q,t)$ function. Per Eq. 2, the FIT is then determined by performing a lookup to the precharacterization.

SPICE simulation shows that this lookup approach is viable. Table 2 shows that the input of the struck gate has a large impact on FIT, so it is important to exhaustively consider local effects. Table 1 shows that the path does not significantly attenuate glitches in this cell library. Table 1 shows that in all cases, the path dependence is minor; this is indicated by all of the even-numbered gates having similar FIT and all of the odd-numbered gates having similar FIT. The odds differ from the evens because they represent two different input states of the gate.

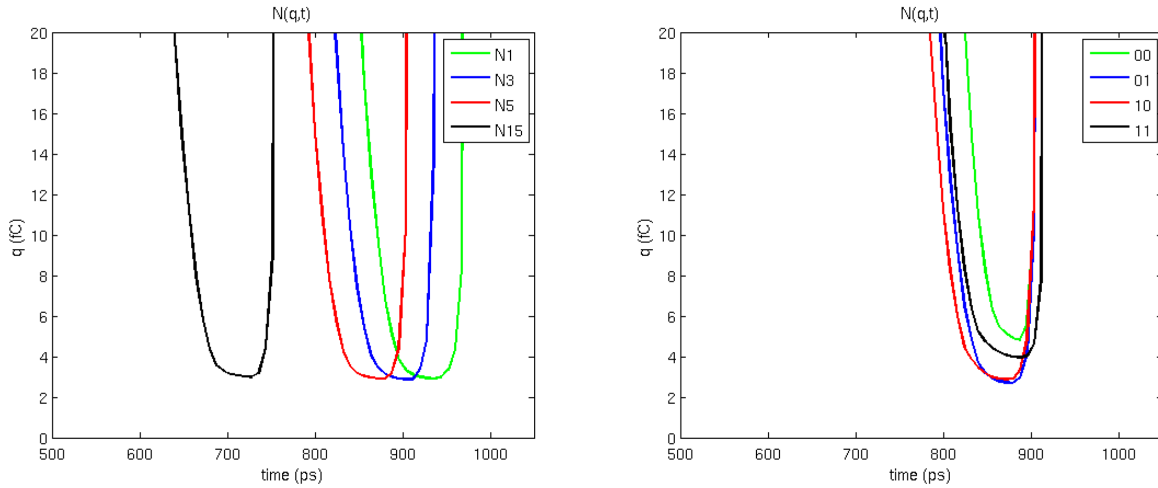


Figure 3: Gate input is more important than path properties. At left, $N(q,t)$ of NAND2_X1 gate with input 10 at distances 1,3,5 and 15 away from a latch. At right, $N(q,t)$ at distance 1 away from latch when varying input states.

Gate	C load	distance to dff																		
		18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
nand2	**0**	25	61	26	62	26	63	26	64	26	64	26	64	25	64	25	64	25	67	28
	0.40fF	27	72	27	71	27	70	27	71	28	72	28	72	28	70	28	71	29	72	28
	0.80fF	26	75	26	78	26	77	27	77	27	78	28	76	29	79	29	76	31	76	28
	1.20fF	24	82	24	81	25	82	25	82	27	82	28	83	29	82	30	84	31	80	28
not	0	10	30	10	29	10	28	10	30	10	29	10	28	10	29	10	29	10	29	11
	0.40fF	11	33	11	34	11	34	12	34	12	34	12	33	12	33	12	33	12	32	11
	0.80fF	12	36	12	36	12	37	12	37	12	37	12	37	12	38	13	37	13	35	11
	1.20fF	11	39	11	40	11	40	12	39	12	40	12	41	12	40	13	40	14	37	11
nor2	0	33	67	37	73	35	71	35	69	36	72	39	72	38	70	39	69	42	74	48
	0.40fF	32	81	36	80	36	78	37	83	40	82	43	80	45	80	45	82	49	82	48
	0.80fF	26	86	28	87	32	89	35	89	38	88	43	87	46	91	49	91	54	89	47
	1.20fF	18	93	21	91	26	96	32	96	36	97	41	98	46	98	50	97	57	95	47

Table 1: The FIT contribution of a gate does not depend strongly on distance from the latch. the units are 1e-7 FIT. When capacitance is added to the gates, the FIT diminishes somewhat further from the latch

diff	gate	input combination															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N-type	not	**36**	–														
	nand2	50	**75**	*161*	–												
	nor2	**89**	–	–	–												
	nand3	59	235	157	233	82	125	134	–								
	nor3	156	–	–	–	–	–	–	–								
	nand4	58	298	224	302	151	201	205	308	77	106	106	165	112	183	204	–
	nor4	226	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
P-type	not	–	9														
	nand2	–	–	–	29												
	nor2	–	29	12	3												
	nand3	–	–	–	–	–	–	–	40								
	nor3	–	69	33	34	29	21	10	4								
	nand4	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	42
	nor4	–	98	48	47	45	31	15	25	41	29	14	17	14	8	5	2

Table 2: the FIT rate of the different input states of each gate, when sensitized by single path and a 1000ps clock. The units are 1e-7 FIT. For each gate, the FIT contribution varies widely with input state. In general, input states that sensitize N-type diffusion cause more FIT. The particular input states that have FIT rates in bold font correspond to the states using in the test circuits in 1

3. BFIT Algorithm

To initiate analysis, BFIT parses the circuit netlist into a levelized dag. One at a time, vectors are applied to the dag, and the failure rate of a single cycle is determined. The analysis of each vector is as follows.

The input vectors are propagated through this levelized dag.

Beginning at the latches, a backtrace procedure is used to find sensitized paths (Algorithm 1). The output of the procedure is, for each pairing of gate and latch, a list of sensitized path delays from the gate to the latch. Note that in forming the list of path delays, we rely on the gate precharacterizations to specify the delay of each sensitized timing arc.

Finally, the list of sensitized paths delays is used with the gate precharacterization information to determine what set of upsets can be caused by a strike to this gate, and to determine what q, t pairings of strikes can cause the different upsets. When a particular strike is found to be capable of causing an error in two or more latches, it will contribute a multi-bit upset.

Eq. 9 shows how the $N(q, t)$ function of a path can be determined based on path delay and precharacterized local factors. Eq. 8 shows how the function can be constructed for a single latch. Eq. 1 shows that, by observing the intersections of the $N(q, t)$ functions, the strikes that can cause various multiple latch upsets are determined. Finally, Eq. 2 is used to determine the failure rate of each possible latch set upset.

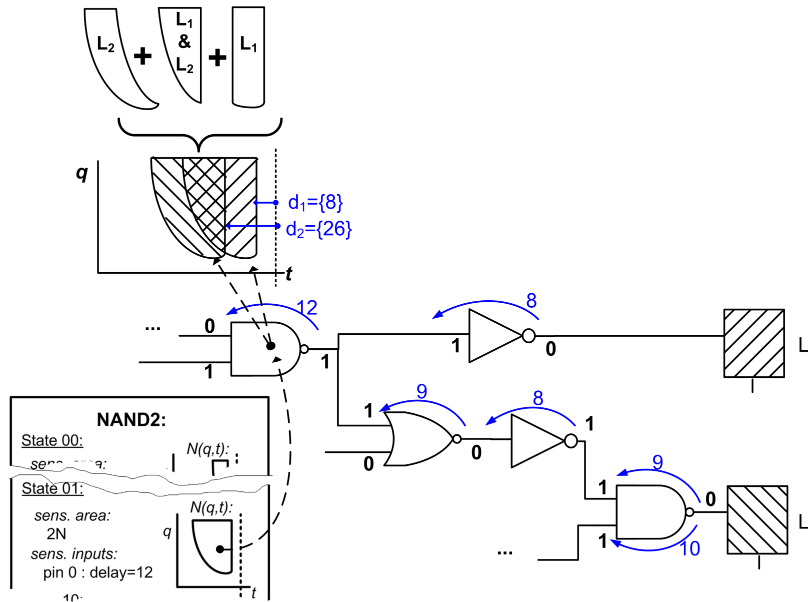


Figure 4: Dynamic programming finds sensitized paths and their delays, which are then combined with gate characterization data, to determine which q, t strikes will cause an error.

Input: Levelized dag of combinational circuit, where vertices are circuit nodes.
Nodes includes combinational gates and latches
Each vertex has a state determined by forward propagating an input vector.
Output: a list $d[n][l]$ of delays, for each pairing of n and l
initially, $d[n][l] = \emptyset$ forall $n \neq l$
initially, $d[n][l] = 0$ forall $n == l$
foreach *level of dag, starting at highest level* **do**
  foreach *node n in current level* **do**
    foreach *fanin gate f of n* **do**
      if *f is capable of flipping n* **then**
        foreach *path delay d_π of $d[n][l]$* **do**
          d[f][l].push_back($d_\pi - d_{f \rightarrow n}$)
        end
      end
    end
  end
end

Algorithm 1: The backtrace algorithm used to find sensitized paths

4. Accuracy

We compare BFIT to exhaustive SPICE simulation on two different circuits. In both BFIT and SPICE FIT share the same $R(q,t)$ function, so we are essentially comparing BFIT's estimate of $N(q,t)$ against an in-situ measurement of $N(q,t)$[5]

ISCAS85 C17

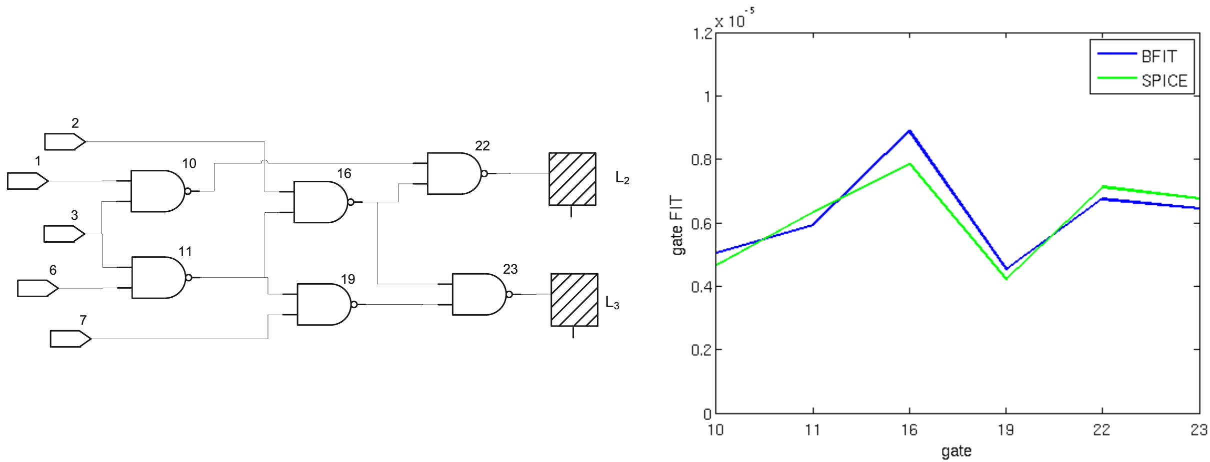
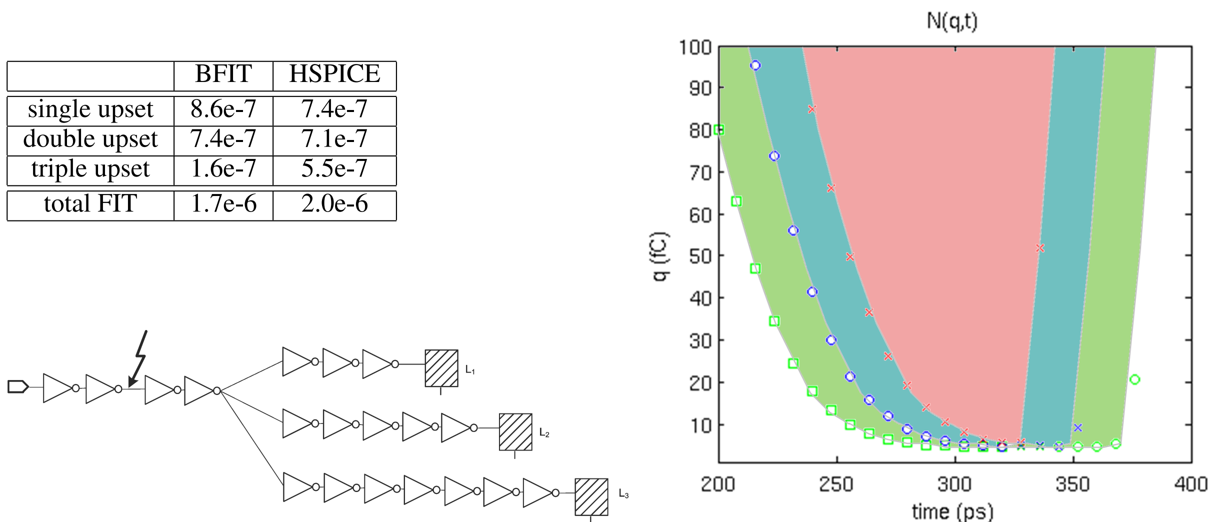


Figure 5: The average FIT contribution of each gate, BFIT and SPICE.

First we show that BFIT can accurately determine the average FIT across all 32 input vectors of each of the 6 NAND2 gates in the ISCAS 89 circuit C17 (Fig. 5). For this experiment, an 800ps clock was used, with a 16ps timestep. For each timestep, the minimum error causing strike is determined using a binary search. Getting the average FIT of a single gate requires 1600 binary searches, and several hours of simulation.

Multibit errors

In the simple three latch circuit shown in Fig 6, we perform exhaustive spice simulation for one gate, and compare this to BFIT. The same SPICE methodology is used here as above with one exception: For each timestep of simulation, we now use a binary search to find the smallest strike capable of causing one error, another binary search to find the smallest strike capable of causing two errors, and third binary search to find the smallest strike capable of causing three errors.



	BFIT	HSPICE
single upset	8.6e-7	7.4e-7
double upset	7.4e-7	7.1e-7
triple upset	1.6e-7	5.5e-7
total FIT	1.7e-6	2.0e-6

Figure 6: the $N(q,t)$ functions for all possible latch sets for the simple circuit shown. The green area has a single latch in error, the blue has two, and the red has three. The shaded areas are approximated, but the data points on the graph are accurate.

[5]but collapsing $N(q,t)$ into the single dimension of FIT

5. Using BFIT

The inputs to BFIT are a combinational netlist and (optionally) input vectors. The output is the FIT contribution of every combinational gate in the circuit, with the fit broken down by exact sets of upset latches. The syntax of a BFIT execution is given here:

./bfit **<netlist> <inputVectors> <clockPeriod> <fitReport>**

<netlist> The combinational netlist.

<inputVectors> can be EITHER
(a) the name of a text file containing input vectors, OR
(b) a number indicating how many random input vectors should be run.

<fitReport> the name of the file where the fit report will be written to.

<clockPeriod> The clock period in ps.

An example of a BFIT execution, and the resulting standard output is shown in Fig. 7. In this example, 100 random vectors are run on the netlist 'circuits/ s9234_mod.bench'; the clock frequency is 1000ps; the FIT breakdown report will be written to 'fit.Fit'.

```
holcomb@reactive:~/BFIT_rc1$ ./bfit circuits/s9234_mod.bench 100 fit.Fit 1000


------XX-XXX-X-X-X----X-----X-X---X-----XXXXXXX-----------XX-----XX-
-------XXXX-X-X-X----X-----X-X---X----XXXXXXXX-----------XX-------X-
--------XX-X-X-X----XX----X-X---X---XXXXXXXXX----------XXX----------
-------XX-X-X-XX---XXXX--X-X---X---XXXXXXXX----------XXXXX----------

         "./bfit circuits/s9234_mod.bench 100 fit.Fit 1000 "

 beginning of BFIT...

-------------------------------------------------------
| ckt             :    circuits/s9234_mod.bench       |
| fit written to  :    fit.Fit                        |
| clock cycle     :    1000ps                         |
-------------------------------------------------------

parsing circuit from file: circuits/s9234_mod.bench
levelized dag into 81 levels
gate count: 7458
                dff:    228
                input:  247
                inv:    4956
                nand2:  1377
                nand3:  61
                nand4:  45
                nor2:   431
                nor3:   30
                nor4:   83

starting to run analysis on 100 random input vectors....

...finished running all vectors

-----------------------------------
| total FIT is :  0.0717346       |
| vectors run  :  100             |
-----------------------------------


holcomb@reactive:~/BFIT_rc1$
```

Figure 7: standard output from a bfit run

5.1. input netlist

The input netlist to BFIT should be a combinational circuit in the ISCAS '.bench' format. The following types of gates are recognized: *DFF, INPUT, OUTPUT, NOT, NAND (2,3,4 input), NOR (2,3,4 input)* . By restricting BFIT to simple inverting gates,

Circuit	INPUTS	LATCHES	GATES	RUNTIME / 1k vectors	FIT
s5378_mod.bench	214	179	3232	35	3.27e-3
s9234_mod.bench	247	228	7230	68	1.06e-2
s13207_mod.bench	700	669	10277	136	1.77e-2
s15850_mod.bench	611	597	12712	207	2.24e-2
s38417_mod.bench	1664	1636	28223	451	5.62e-2
s38584_mod.bench	1464	1452	28854	1311	4.85e-2
s35932_mod.bench	1763	1728	23012	204	3.99e-2

Table 3: The BFIT runtime, per thousand vectors, of large ISCAS circuits. The clock period used for each circuit is 1000ps.

each gate corresponds to a single strikable node, so there is no need to descend below the gate level of hierarchy[6].

In the circuits subdirectory of the BFIT source, we provide the benchmark circuits that were used to generate the results of this paper. These circuits are based on the ISCAS'89 sequential benchmark, except that we have cut the sequential circuit to make it combinational, and performed some simple replacement to map the circuit onto our limited set of gates.

5.2. input vectors

The input vectors used for BFIT can either be randomly[7] generated or else supplied as a text file.

If supplied as a text file, each line should contain a single vector in ASCII, with the circuit input bits separated by spaces. The inputs are interpreted in alphabetical order; the first (leftmost) bit on any line of the text file will be assigned to the input signal that is closest to the the beginning of the alphabet. When BFIT is run, it prints out a file called inputOrdering.txt, that shows the order in which inputs have been interpreted.

5.3. output report

Aside from the general information in standard output of BFIT, the detailed report of a circuits FIT is contained in whatever file was specified by the command line argument. Figure 8 is the report generated from running the combinational version of ISCAS circuit s27.

```
holcomb@reactive:~/BFIT_rc1$ more fit.Fit

ckt: circuits/s27_mod.bench
clock: 1000ps

     G14 :  2.68641e-07 -> G6_NEXT
     G14 :  2.13757e-07 -> G5_NEXT & G6_NEXT
     G14 :  8.80027e-07 -> G5_NEXT

     G12 :  1.58168e-06 -> G7_NEXT
     G12 :  1.30921e-07 -> G6_NEXT & G7_NEXT
     G12 :  4.24908e-07 -> G6_NEXT
     G12 :  3.44801e-09 -> G5_NEXT & G6_NEXT & G7_NEXT
     G12 :  1.03958e-07 -> G5_NEXT & G6_NEXT
     G12 :  5.71491e-08 -> G5_NEXT

------------------------------
 [cut]
------------------------------
     G10 :  5.25918e-06 -> G5_NEXT


total ckt FIT: 2.7228e-05

holcomb@reactive:~/BFIT_rc1$
```

Figure 8: a FIT report.

5.4. Runtime

BFIT can run thousands of vectors in an hour on 20k gate circuits. The forward propagation of state is performed on a levelized DAG, and is thus linear in the size of the circuit. The backtrace for finding sensitized paths uses dynamic programming, and only visits each gate once per vector. However, the number of paths that must be back-propagated through each gate is potentially exponential. The runtimes for combinational versions of the sequential ISCAS circuits, is shown in Table 3.

[6] for example, and AND gate has an internal node that computes NAND, followed by an inverter, so there would then be two different sensitized (reverse biased) diffusions, and they would be of opposite type.

[7] the random number generator is given a consistent seed and will reproduce the same vectors each time is run - done this way to make results reproducible

6. Results from ISCAS'89 benchmarks

Here we present some results obtained using BFIT on the ISCAS89 circuits. These results show the significance of multiple-bit errors, and show how clock period can effect FIT rates. In our DATE 2009 paper [20], we show how BFIT can guide efficient circuit hardening.

Multiple Latch Upsets.

We find that in some circuits, multiple latch upsets can account for a significant share of errors, as high as 16 percent in the combinational version of ISCAS89 circuit s5378 (see Table 4).

circuit	bits in error					
	1	2	3	4	5	6
s5378_mod.bench	0.8388	0.0814	0.0364	0.0128	0.0080	0.0048
s9234_mod.bench	0.8625	0.0846	0.0122	0.0119	0.0031	0.0176
s13207_mod.bench	0.9379	0.0265	0.0052	0.0049	0.0035	0.0048
s15850_mod.bench	0.9148	0.0329	0.0188	0.0083	0.0066	0.0040
s38417_mod.bench	0.8812	0.0624	0.0222	0.0097	0.0083	0.0039
s38584_mod.bench	0.9538	0.0262	0.0053	0.0029	0.0019	0.0015
s35932_mod.bench	0.9960	0.0001	0.0001	0.0001	0.0002	0.0003

Table 4: The share of errors that effect multiple latches.

7 Appendix

7.1 Included Benchmark files

Because BFIT runs on combinational circuits, we convert the ISCAS89 sequential circuits [3] into BFIT-suitable combinational circuits [8]. These circuits are included in the zip file of the source code.

The modifications we make to the sequential circuit are as follows:

1. Cut the output signal of each latch. All gates driven by the cut signal are instead driven by a new combinational input of the same name; these inputs allow controllability the current state of the system. The latch output that was cut is now connected to a combinational output for observability (see Fig. 9).

2. All non-inverting gates are decomposed into constituent inverting gates. Gates with more than four inputs are broken down into multiple smaller gates.

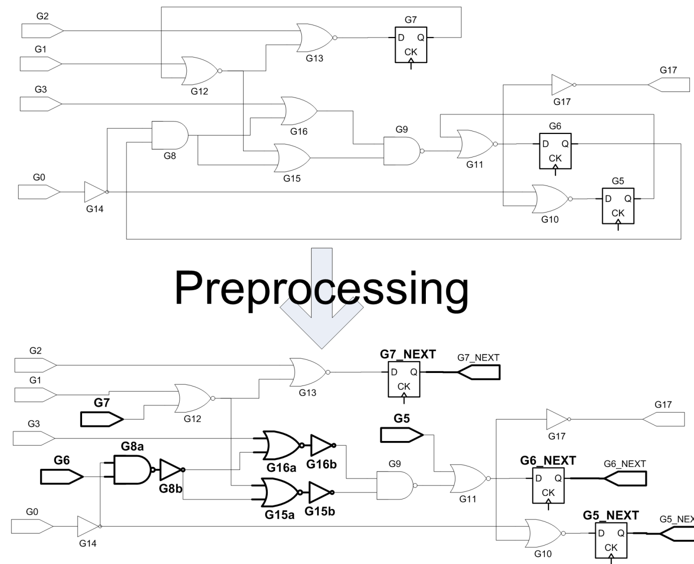


Figure 9: Preprocessing breaks the sequential circuit and decomposes non-inverting gates.

7.2 BFIT source code

BFIT is implemented in C++.

All of the source files needed to run BFIT are contained in file BFIT_1.0.zip. Example circuits are also contained in the subdirectory circuits of the zip file. It is assumed that the user has a C++ compiler, and C++ STL libraries. The list of required source files for running BFIT alpha are:

- bfit.cpp - the main code of BFIT
- bfit.h - the technology characterization. In this case, it is for the 45nm Nangate open cell library.
- gate.cpp - the gate class
- bfitUtils.cpp - the functions for manipulating the dag, estimating FIT, etc.

To compile BFIT, just do:

```
g++ -c bfit.cpp -o bfit.o
g++ bfit.o -o bfit
```

[8]ISCAS'89 circuits can be found at: http://www.cs.ubc.ca/spider/ajh/courses/cpsc513/assign-reach/ISCAS89/DATA/

References

[1] H. Asadi and M. B. Tahoori. Soft error derating computation in sequential circuits. In *ICCAD*, pages 497–501, 2006.

[2] N. Miskov-Zivanov and D. Marculescu. Modeling and Optimization for Soft-Error Reliability of Sequential Circuits. *IEEE Trans. on CAD of Integ. Circ. and Sys.*, pages 803–816, May 2008.

[3] Brglez, F.; Bryan, D.; Kozminski, K., Combinational profiles of sequential benchmark circuits. ISCAS89

[4] N. Miskov-Zivanov and D. Marculescu. Circuit Reliability Analysis Using Symbolic Techniques. *IEEE Trans. on CAD of Integ. Circ. and Sys.*, pages 2638–2649, Dec. 2006.

[5] D. Holcomb et al. Berkeley FIT estimation tool (BFIT). `http://www.eecs.berkeley.edu/~holcomb/BFIT.htm`

[6] R. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Trans. Device and Materials Reliability*, 5(3):305–316, Sept. 2005.

[7] L. B. Freeman. Critical charge calculations for a bipolar SRAM array. *IBM J. Res. Dev.*, 40(1):119–129, 1996.

[8] R. Garg et al. A fast, analytical estimator for the SEU-induced pulse width in combinational designs. *DAC'08*, pages 918–923.

[9] P. Hazucha and C. Svensson. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. Nuclear Science*, 47(6):2586–2594, Dec 2000.

[10] J. A. Rivers et al. Phaser: Phased methodology for modeling the system-level effects of soft errors. *IBM Journal of Research and Development*, 52(3):293–306, May 2008.

[11] S. Krishnaswamy, et al. On the role of timing masking in reliable logic circuit design. *DAC 2008*, pages 924–929.

[12] P. Liden, et al. On latching probability of particle induced transients in combinational networks. *FTCS 1994*, pages 340–349.

[13] L. Massengill, et al. Analysis of single-event effects in combinational logic-simulation of the AM2901 bitslice processor. *IEEE Trans. Nuclear Science*, 47(6):2609–2615, Dec 2000.

[14] D. Mavis and P. Eaton. Soft error rate mitigation techniques for modern microcircuits. *Rel. Phy. Symp.*, pages 216–225, 2002.

[15] M. Omana, et al. A model for transient fault propagation in combinatorial logic. *IOLTS 2003*, pages 111–115, July 2003.

[16] L.-S. Peh. *Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks*. PhD thesis, Stanford University, August 2001.

[17] R. R. Rao, et al. Computing the soft error rate of a combinational logic circuit using parameterized descriptors. *IEEE Trans. on CAD of Integ. Circ. and Sys.*, 26(3):468–479, 2007.

[18] S. S. Mukherjee et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *MICRO 2003*, pages 29–40.

[19] S. A. Seshia, et al. Verification-guided soft error resilience. In *DATE 2007*, pages 1442–1447.

[20] D. E. Holcomb et al., et al. Design as you see FIT: System-level soft-error analysis of sequential circuits. In *DATE 2009*, to appear.

[21] P. Shivakumar, et al. Modeling the effect of technology trends on soft error rate of combinational logic, *DSN'02*, pp. 389-398.

[22] B. Zhang, et al. FASER: fast analysis of soft error susceptibility for cell-based designs. *ISQED 2006*, pages 755-760.

[23] M. Zhang, et al. Sequential element design with built-in soft error resilience. *IEEE Transactions on VLSI*, Dec. 2006.

[24] M. Zhang and N. R. Shanbhag. Soft-error-rate-analysis (SERA) methodology. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(10):2140–2155, 2006.

[25] Q. Zhou and K. Mohanram. Gate sizing to radiation harden combinational logic. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(1):155–166, 2006.

[26] J. Ziegler. Terrestrial cosmic rays. *IBM J. Res. Develop*, 40(1):pp19–39, January 1996.