A Study of Some Flawed Adders

*** Work In Progress ***

Robert S. Boyer
and
Warren A. Hunt, Jr.

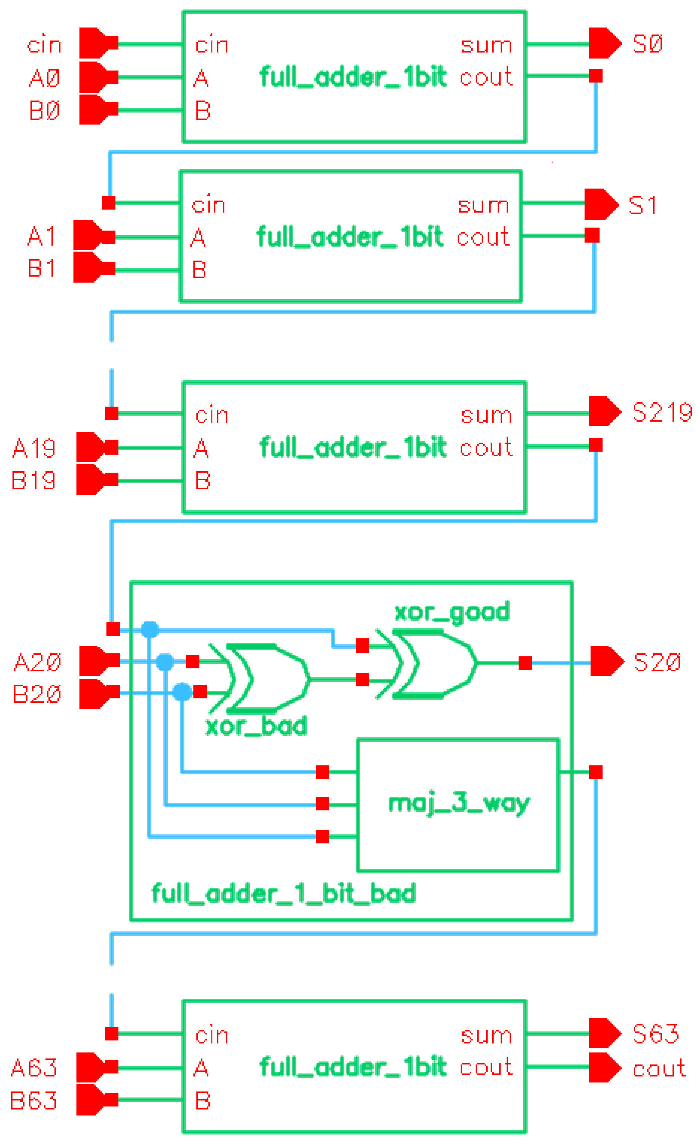
Department of Computer Sciences
1 University Station, M/S C0500
The University of Texas
Austin, TX 78712-0233

E-mail: {boyer,hunt}@cs.utexas.edu
TEL: +1 512 471 9745, +1 512 471 9748
FAX: +1 512 471 8885

Circuit Specification, Abstraction, and Reverse Engineering

- Does a manufactured circuit meet its specification?
  - Requires some kind of reverse engineering,
  - Low-level (maybe, transistor-level) analysis,
  - Higher-level specifications, and
  - Verification tools.
- When an ASIC is made ready for manufacturing,
  - technology mapping occurs,
  - synthesis and re-timing are performed,
  - test logic is added, and
  - a floorplan and layout is created.
- Subtle changes can be introduced by foundries.
  - Some circuits are added for testability, reliability, etc.
  - But, are some circuits added as Trojan horses?
- Given a transistor- or gate-level model, could we separate good changes from bad changes?
  - I started to wonder if the number of differences could be measured.
  - I wondered if the number of differences mattered.
  - And, I wondered if measured differences indicated anything.

Motivating Example: Verification of a Hardware Adder



- It should be easy to verify an adder:
  - adders have a regular structure
  - it just computes a sum.
- However, implementation flaws may still exist:
  - CAD or manufacturing flaws, or
  - Malicious changes might be made.
- To thoroughly verify an adder implementation requires:
  - netlist with transistor strengths, capacitance of wires, etc.
  - a transistor-level analyzer, and
  - a symbolic verifier.
- Could we detect a subtle change?

Circuit Verification and Measured Differences

- Generally, circuits are verified by simulation.
- We advocate symbolic verification, but even so, there may be differences that are acceptable:
  - circuits are used in a restricted environment,
  - circuits used with limited input values, or
  - approximate answers adequate
- Let's count the differences between an XOR and an OR gate.

```
   A      B         OR     XOR     Same   Different
-------------+----------------------------------
   0      0    |    0       0        1        0
   0      1    |    1       1        1        0
   1      0    |    1       1        1        0
   1      1    |    1       0        0        1
                                  --------------------
                       Total:    3        1
```

Function Representation using BDDs

- We represent binary functions as `HONS` trees.
  - The variable order is implicit.
  - The BDDs are reduced – they may terminate early.

```
        O    <-- A          ;  The simple tree representing the
       / \                  ;  disjunction of A and B:
      /   \
     T     O   <-- B
          / \               ;  The tree is represented by
         /   \              ;  (HONS T (HONS T NIL))
        T    NIL            ;  which just prints as:  (T T).
```

- We have defined functions to perform logical operations on BDDs.

```
(let* ((a (hons T NIL))      O    <--  A  -->    O
       (b (hons a a)))      / \                 / \
  (q-fn 'or a b))          T  NIL               \ /
 ==>                                   B  -->    O
(T T)                                           / \
                                               T  NIL
```

- Printing large BDDs isn't possible – too much output.

Counting when a BDD Function is 1 or 0

Given a BDD, can we count the number of times the output is 1 and 0?

```
(defun count-tip-values (x depth)
  (if (atom x)
      (mv (if x (expt 2 depth) 0)
          (if x 0 (expt 2 depth)))

    (mv-let
     (left-cnt-1s left-cnt-0s)
     (count-tip-values (car x)
                       (1- depth))
     (mv-let
      (right-cnt-1s right-cnt-0s)
      (count-tip-values (cdr x)
                        (1- depth))

      (mv (+ left-cnt-1s right-cnt-1s)
          (+ left-cnt-0s right-cnt-0s))))))
```

Using COUNT-TIP-VALUES determine the number of input combinations that produce 1 and 0 outputs.

```
(count-tip-values '(t t) 2)  ==>  (3 1)
```

Let's now produce the *difference* function between the XOR and OR functions.

```
(let* ((a (hons t nil))
       (b (hons a a)))
  (q-fn 'eqv
        (q-fn 'xor a b)
        (q-fn 'or  a b)))
```

Using COUNT-TIP-VALUES, we count the differences.

- The second argument provides a bias.

```
(let* ((a (hons t nil))
       (b (hons a a)))
  (count-tip-values
   (q-fn 'eqv
         (q-fn 'xor a b)
         (q-fn 'or  a b))
   2))
 ==>
(3 1)
```

Counting the Difference Between Two Vector of BDD Functions

When counting the differences between two, bit vectors, we compute the maximum number differences.

```
(defun count-max-tip-errors
  (x depth cnt)

  (if (atom x)
      cnt
    (mv-let
     (ones zeros)
     (count-tip-values (car x) depth)
     (declare (ignore ones))

     (count-max-tip-errors
      (cdr x) depth
      (max zeros cnt)))))
```

And when we compare a family of bit vectors to a single, specification bit vector, we compute the smallest, non-zero number of differences.

To determine the differences between two bit vectors, we compute the differences on a bit-by-bit basis.

```
(defun qv-ite-cmp (a b)
  (if (atom a)
      (if (atom b)
          nil
        (cons nil
              (qv-ite-cmp nil (cdr b))))
    (if (atom b)
        (cons nil
              (qv-ite-cmp (cdr a) nil))
      (cons
       (q-fn 'eqv (car a) (car b))
       (qv-ite-cmp (cdr a) (cdr b))))))
```

Incomparable positions of bit vectors of uneven length are assigned the maximum number of differences; i.e., `NIL`.

- We then measure the differences.

Example, Bit Vector Differences

```
      @-----------------@-----------------@------ NIL
     /                 /                 /
    /                 /                 /
   O    <-- A -->    O                NIL
  / \               / \
 /   \             /   \
T     O   <-- B   T    NIL             @-----------------@-----------------@------ NIL
     / \                              /                 /                 /
    /   \                            /                 /                 /
   T    NIL                         O    <-- A -->   NIL                O
                                   / \                                 / \
                     Answer ==>   /   \                               /   \
                                 T     O   <-- B                    NIL    T
                                      / \
                                     /   \
                                   NIL    T

      @-----------------@-----------------@------ NIL
     /                 /                 /
    /                 /                 /
   O    <-- A -->    O                 O
  / \               / \               / \
 /   \             /   \             /   \
T    NIL         NIL    T           T    NIL
```

Example, Count the Bit Vector Differences

Given the difference equations, the number of differences is shown:

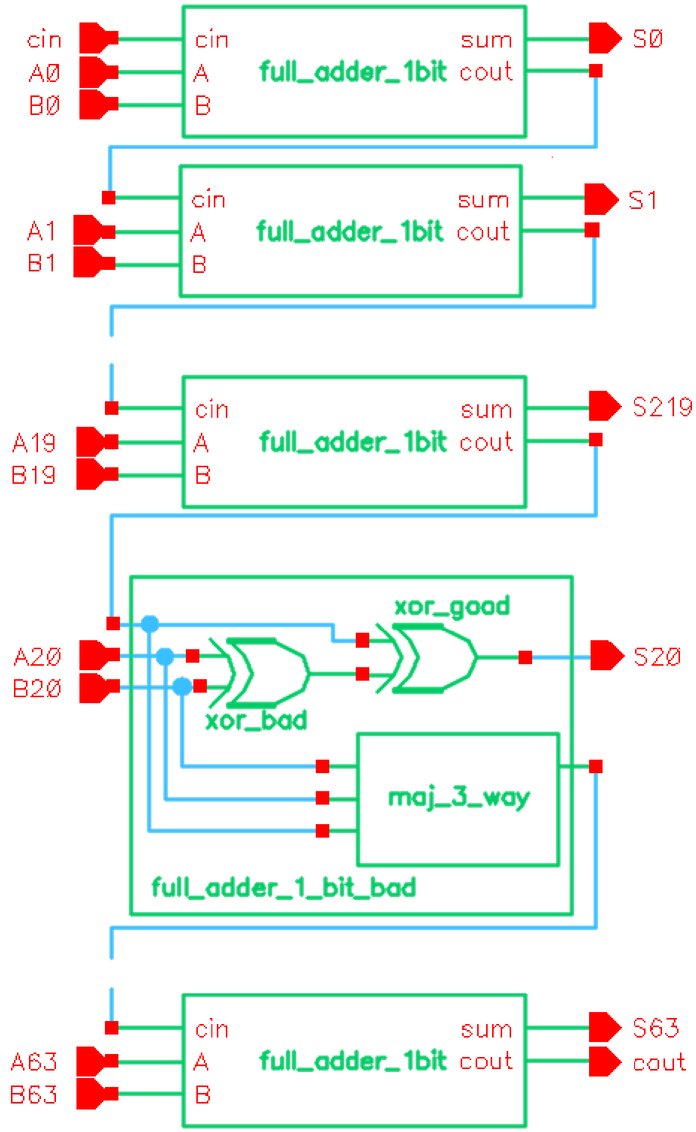
```
      @-----------------@-----------------@------ NIL
     /                 /                 /
    /                 /                 /
   O    <-- A -->   NIL                O
  / \                                 / \
 /   \                               /   \
T     O   <-- B                    NIL    T
     / \
    /   \
  NIL    T
```

```
      @-----------------@-----------------@------ NIL
     /                 /                 /
    /                 /                 /
   1                 4                 2
```

For this result, there are four differences.

When we compare the counts of many bit vectors, we *drop* bit vectors that match.

Single Gate Failures



- First Experiment – 64 bit adder.
  - Fault each two-input gate with the other 15 Boolean logic functions.
  - Measure differences.
- There are 4800 flawed adders:
  - 64 bit positions
  - 5 gates per bit position
  - 15 faulty gates per gate
  - 65 equations, 312,000 differences
- Results (for 129 Boolean inputs)
  - For one gate, replacing XOR by OR makes no difference
  - In all other cases we at least find 2^{126} differences in some bit.

Single Input-Pair Failure

Consider a 64-bit adder that returns an incorrect answer for a single pair of numbers.

- Seems like this should be easy to detect by structural means, but
  - Not if exists in purchased IP,
  - Not if embedded in an ALU, or
  - Not if a fabrication change.
- So, we use the developed machinery.

```
(defun sbv-bv-adder
  (c a b a-val b-val ans-val)
  (let
   ((bv-adder  (q-bv-adder c a b))
    (cmp-a-val (q-ite-cmp a a-val))
    (cmp-b-val (q-ite-cmp b b-val)))
   (qv-if-ite
    (q-fn 'and cmp-a-val cmp-b-val)
    ans-val bv-adder)))
```

Let's try our subtly flawed adder model. This adder has a built-in key.

```
(v-to-nat
 (sbv-bv-adder
  nil
  (nat-to-v  7 64) (nat-to-v  3 64)
  (nat-to-v  3 64) (nat-to-v  7 64)
  (nat-to-v 11 65)))
 ==> 10
```

In this case, it works fine, but...

```
(v-to-nat
 (sbv-bv-adder
  nil
  (nat-to-v  3 64) (nat-to-v  7 64)
  (nat-to-v  3 64) (nat-to-v  7 64)
  (nat-to-v 11 65)))
 ==> 11
```

We can use our counting mechanisms to determine the number of differences.

Count the Bit Vector Differences For Slightly Bad Adder

Given the difference equations, the number of differences is shown:

```
(count-tip-values-list
 (qv-ite-cmp *q-bv-adder* *sbv-bv-adder*)
 (len *all-vars*) 0)
  ==>
((:CORRECT-ANSWERS 680564733841876926926749214863536422911 :WRONG-ANSWERS 1)
 (:CORRECT-ANSWERS 680564733841876926926749214863536422912 :WRONG-ANSWERS 0)
 (:CORRECT-ANSWERS 680564733841876926926749214863536422912 :WRONG-ANSWERS 0)
 (:CORRECT-ANSWERS 680564733841876926926749214863536422912 :WRONG-ANSWERS 0)
 (:CORRECT-ANSWERS 680564733841876926926749214863536422912 :WRONG-ANSWERS 0)
 ...)
```

We can compute this answer in a few milliseconds.

But, so what?

- Is this a good test for a Trojan Horse type of flaw?
- What other tests might be tried?
- What happens on other functions?

Cone-of-Influence For Slightly Bad Adder

Using the same flawed adder specification, we can compute the cone-of-influence of the inputs for each output.

- For a good adder, the first output bit is dependant on only the input carry and the first bit of the two vectors to be added.
- For our flawed adder, every output is dependent on every input bit.
- Thus, we are investigating the *signatures* of different logic functions using these and other measuring functions.

Discussion

Using unique Boolean function representations and function memoization, we can compute the signatures of thousands of different functions in seconds.

- We actually use a one-argument counting function – it memoizes much more effectively.
- Is this capability just a novelty? Or, could it be useful?
- We find these capabilities useful for bug hunting.