DVCon 2013
Design & Verification Conference & Exhibition

February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

fsim_logic – A VHDL type for testing of FLYTRAP

by
Joanne E. DeGroat, Ph.D.
Associate Professor
The Ohio State University

Sponsored By:

Presentation Outline

- Background – What was needed and why
- What was readily available and prior work in the area
- The fault simulation types
  - fbit and fsim_logic
  - VHDL packages for fault simulation
- Use of fbit and fsim_logic to address the need
- Conclusions

Sponsored By:

SYSTEMS INITIATIVE

Background

- What was needed and why
- Working in the area of fault-tolerant design
  - A design where the presence of an incorrect logic value is ‘corrected for’
  - Fault occurs due to
    - And intermittent fault –circuit noise, transistor threshold
    - A permanent fault – manufacture or developed over time
- The ‘corrected for’
  - Fault is not fixed
  - Fault is mitigated – through detection and redundancy a valid result is generated

Sponsored By:

Prior work in fault simulation

- During the 1980s fault simulation was a major research focus.
  - A standard set of circuits for evaluation of fault simulator.
  - Many custom simulators – nothing standardized.
- More recent work (2005) – SESC simulator – University of Illinois
  - Architectural evaluation including faults
  - Designs use shadow register to mitigate faults

Modern circuits – HDL Design

- HDLs used for design of embedded systems to complex processors.
- Desire is to have a code version of the HDL that allows fault injection during simulation.
  - Fault injection – A gate evaluation generates the wrong result – probabilistically
  - Allows for control and measurement of the number of faults injected.
  - Faults are injected at the lowest level of design – the gates.

Fault tolerant circuits

- Fault tolerant circuit design validation
  - – Correct circuit design validation
    - NO ERRORS – typical verification methodologies
    - ERRORS – some gate evaluations result in errors
  - – Circuit behavior under error
    - Single Error Detection (SED) circuits – error detected
    - Single Error Correction (SEC/DED) circuits – single errors correct for (output valid) – dual errors detected
  - – No error injection – no verification of SEC/DED design

Presentation Outline

- Background – What was needed and why
- What was readily available and prior work in the area
- The fault simulation types
  - fbit and fsim_logic
  - VHDL packages for fault simulation
- Use of fbit and fsim_logic to address the need
- Conclusions

Sponsored By:

Prior work in this area

- Work in early 1990s
  - Automatic generation of fixed error HDL models
  - Error location fixed
- Limitations of methodology
  - Large number of fixed error models
  - Limited to fixed location(s) of error
- Not practical for today's design
  - Design with 100,000 gates → 100,000 possible locations for fixed error. (stuck at 0, stuck at 1)

Sponsored By:

An alternative approach

- VHDL allows for user defined logic types
- A new type
  - Already have bit, std_logic, signed, unsigned, ... (most design in bit, std_logic)
  - Create types fbit, fsim_logic
  - Overload all operators
    - In logic operator evaluation still a table lookup
    - Add a call to a random number generator for fault injection and compare to a threshold.

Sponsored By:

SYSTEMS INITIATIVE

What the package provides

- The ability to test fault tolerant circuits
  - Error injection is uniformly distributed error injection over all logic operators.
  - Uniform distribution valid as testing operation under error conditions.
  - Number of errors injected can be adjusted to achieve useful results in reasonable simulation time.
  - Distribution of errors over entire design, not fixed location(s).
- Can test any design for behavior under fault conditions. (non-fault-tolerant design)

Presentation Outline

Sponsored By:

- Background – What was needed and why
- What was readily available and prior work in the area
- The fault simulation types
  - fbit and fsim_logic
  - VHDL packages for fault simulation
- Use of fbit and fsim_logic to address the need
- Conclusions

Package fbit

- VHDL (and System Verilog) allow for user defined types and overloading of operators.
- fbit corresponds to type bit – fbit is the fault injected version
- Modification to test circuits using fbit
  - Change type declarations from bit to fbit in architecture and ports.
  - Testbench uses type conversion to drive model.

Sponsored By:

The package declaration

- The declaration
- Logic functions overloaded
- Relational functions overloaded (=,/=,<,<=,>,>=)
- Type conversion functions

```
--------------------------------------------------------------------------------
-- Fault Simulation Package – TYPE fbit (non-resolved)
--------------------------------------------------------------------------------
PACKAGE fbit_logic IS

TYPE fbit IS ('0',  -- low
              '1'   -- high  );
--------------------------------------------------------------------------------
--unconstrained array of fbit
--------------------------------------------------------------------------------
TYPE fbit_vector IS ARRAY (NATURAL RANGE <>) of fbit;
--------------------------------------------------------------------------------
--Declare logic functions
--------------------------------------------------------------------------------
FUNCTION "AND" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "OR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "NAND" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "NOR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "XOR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "XNOR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "NOT" (l:fbit) RETURN fbit;
--------------------------------------------------------------------------------
FUNCTION "AND" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "OR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "NAND" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "NOR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "XOR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "XNOR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "NOR" (l:fbit_vector) RETURN fbit_vector;
--------------------------------------------------------------------------------
CONSTANT threshold : REAL := -.999999;
impure FUNCTION getrand RETURN real;

                                                  END fbit_logic;
```

Function Evaluation

- Table lookup used: function evaluation and error injection

```
– CONSTANT and_table : fbitlogic_table := (
–   --------------------------------------
–   --|  0    1
–  --------------------------------------
–      (‘0’, ‘0’),   -- 0
–      (‘0’, ‘1’)  );-- 1
– FUNCTION “AND” (l,r : fbit) RETURN fbit IS
–   VARIABLE val : fbit; VARIABLE rnd : REAL;
– BEGIN
–   val := and_table(l,r);
–   rnd : getrand;
–   IF (rnd>threshold) THEN
–          val:=(error_table(val)); END IF;
–   RETURN (val);
– END “AND”;
```

Sponsored By:

Error Injection

- Random number generation
  - `Impure FUNCTION getrand RETURN real IS`
  - `  VARIABLE vrandval : REAL;`
  - `BEGIN`
  - `  UNIFORM (seed1,seed2,vrandval);`
  - `  RETURN (vrandval);`
  - `END getrand;`
- The error table
  - `CONSTANT error_table : fbit_1d := (‘1’,’0’);`
- Modification of seeds – variable simulation results

Sponsored By:

Package fsim_logic

- fsim_logic corresponds to type std_logic – fsim_logic is the fault injected version
  - Also a resolved type
- Modification to test circuits using fsim_logic
  - Change type declarations from std_logic to fsim_logic in architecture and ports.
  - Testbench uses type conversion to drive model.

Sponsored By:

fsim_logic package declaration

- The declarative part (*initial part)*

```
PACKAGE fsim_logic_1 IS

TYPE fsim_ulogic IS ('0', --low
                     '1'  --high
                    );
-------------------------------------------------------------------
-- unconstrained array of fsim_ulogic
-------------------------------------------------------------------
TYPE fsim_ulogic_vector IS ARRAY (NATURAL RANGE <>) OF fsim_ulogic;

-------------------------------------------------------------------
-- Resolution Function
-------------------------------------------------------------------
FUNCTION resolved (s: fsim_ulogic_vector) RETURN fsim_ulogic;

-------------------------------------------------------------------
-- Declare fault injection fsim_logic type
-------------------------------------------------------------------
SUBTYPE fsim_logic IS resolved fsim_ulogic;

-------------------------------------------------------------------
-- Declare fault injection fsim_logic_vector type
-------------------------------------------------------------------
TYPE fsim_logic_vector IS ARRAY (NATURAL RANGE <>) of fsim_logic;
```

fsim_logic is resolved

- No errors injected on resolution
- Resolution table same as std_logic

```
CONSTANT resolution_table : fsimlogic_table := (
--    ---------------------------------------------
--    | U    X    0    1    Z    W    L    H    -
--    ---------------------------------------------
      ('U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U'),   -- U
      ('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'),   -- X
      ('U', 'X', '0', 'X', '0', '0', '0', '0', 'X'),   -- 0
      ('U', 'X', 'X', '1', '1', '1', '1', '1', 'X'),   -- 1
      ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X'),   -- Z
      ('U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X'),   -- W
      ('U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X'),   -- L
      ('U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X'),   -- H
      ('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X')    -- -
       );
```

Error injection

- Error injection on logical operations
- Methodology the same a fbit
- No error injection for ‘U’,’X’,’W’,’Z’,’-’

```
CONSTANT error_table : fsimlogic_1d :=
--     ---------------------------------------
--     | U    X    0    1    Z    W    L    H    -
--     ---------------------------------------
       ('U', 'X', '1', '0', 'Z', 'W', 'H', 'L', '-' );
```

The logic functions

- The ‘AND’ Table

```
--truth table for "and" functilon
CONSTANT and_table : fsimlogic_table := (
--    -----------------------------------------------
--    | U    X    0    1    Z    W    L    H    -
--    -----------------------------------------------
      ('U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U'),   -- U
      ('U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'),   -- X
      ('0', '0', '0', '0', '0', '0', '0', '0', '0'),   -- 0
      ('U', 'X', '0', '1', 'X', 'X', '0', '1', 'X'),   -- 1
      ('U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'),   -- Z
      ('U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X'),   -- W
      ('0', '0', '0', '0', '0', '0', '0', '0', '0'),   -- L
      ('U', 'X', '0', 'X', 'X', 'X', '0', '1', 'X'),   -- H
      ('U', 'X', 'X', 'X', 'X', 'X', '0', 'X', 'X')    -- -
       );
```

- The ‘AND’ function

```
FUNCTION "and" (l : fsim_ulogic; r : fsim_ulogic) RETURN fsim_ulogic IS
  VARIABLE val : fsim_ulogic;
  VARIABLE rnd : REAL;
BEGIN
   val := and_table(l,r);
   rnd := getrand;
   IF (rnd > threshold) THEN val := (error_table(val)); END IF;
   RETURN (val);
END "and";
```

Sponsored By:

Presentation Outline

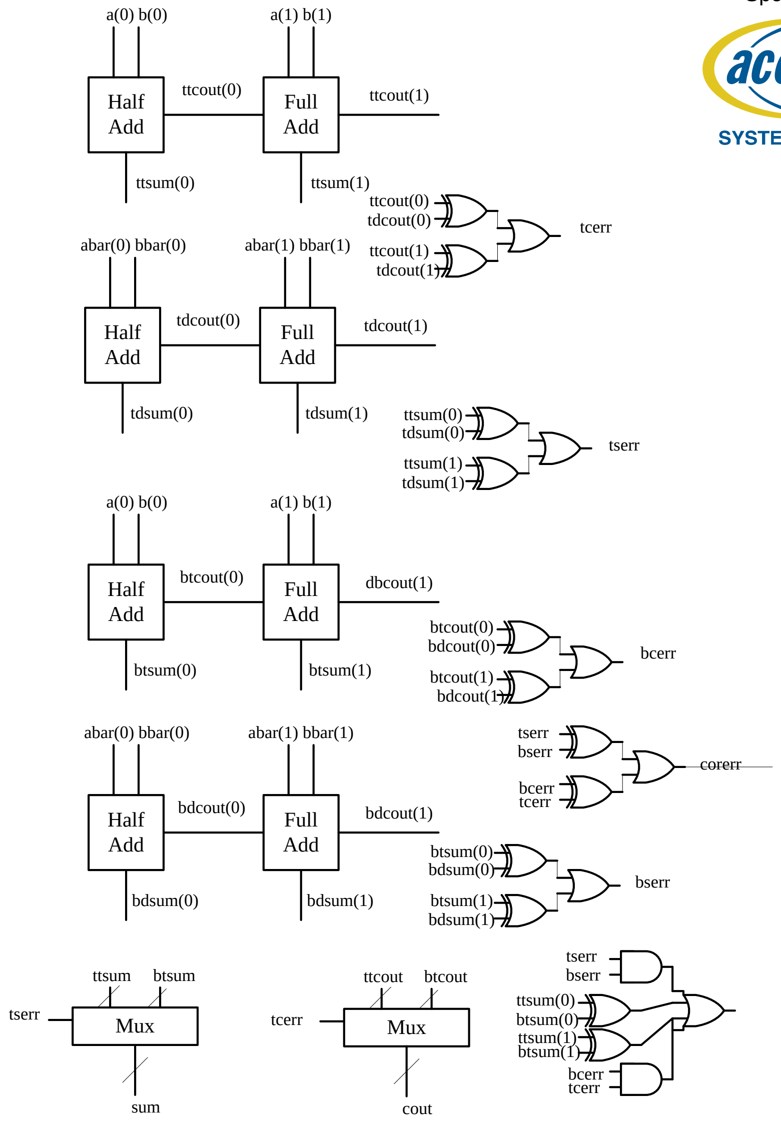
- Background – What was needed and why
- What was readily available and prior work in the area
- The fault simulation types
  - fbit and fsim_logic
  - VHDL packages for fault simulation
- Use of fbit and fsim_logic to address the need
- Conclusions

The application of package

- Fault tolerant design
- One of designs – a fault tolerant adder
- 4x replication
  - Provides Single Error Correction and Dual Error Detection (on a bit position by bit position basis)
    - 2x replication – SED
    - 3x replicaiton – SEC (TMR-many space systems)
    - 4x replication – SEC/DED
    - 5x replication – DEC (Space shuttle systems)

The adder design

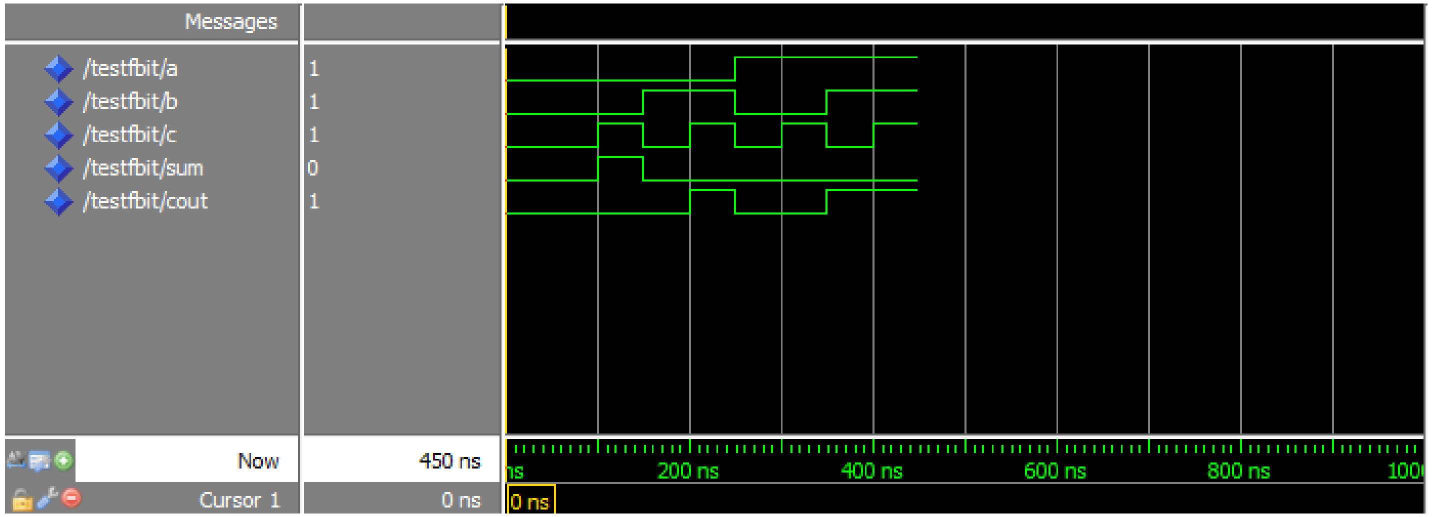
- Diagram shows 2 bit positions
- Design is duplicate dual rail
- Why dual rail?



Example of use

Sponsored By:

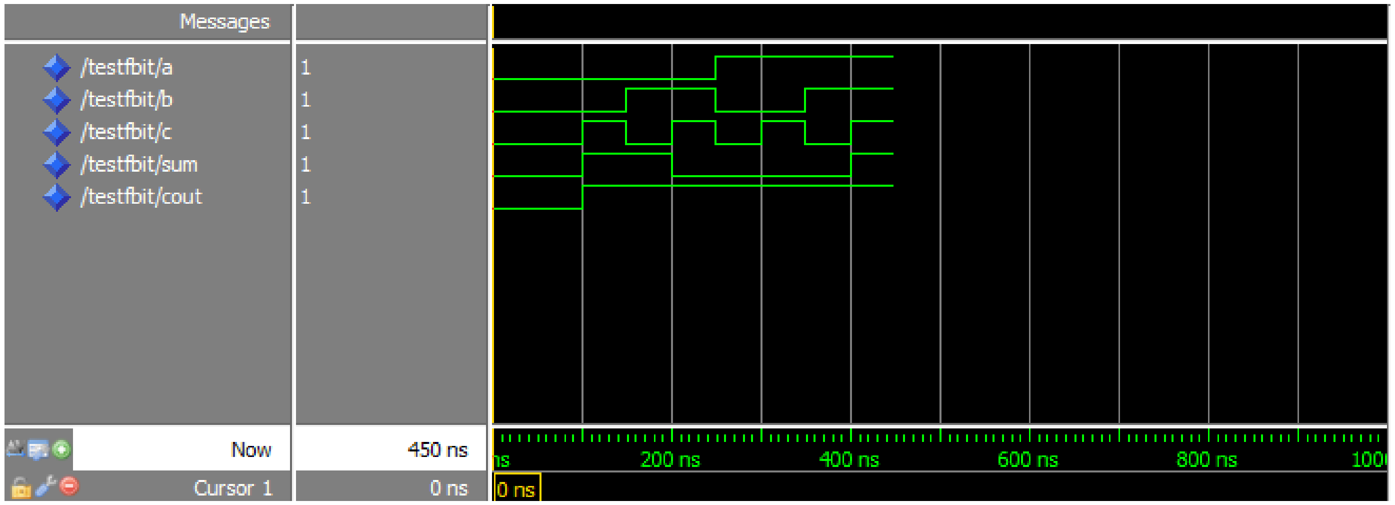
- Applied to single full adder
  - Full adder – 6 logic operation evaluations for each test transaction
  - Threshold of .9999 results in no errors



Sponsored By:

Example of use - errors

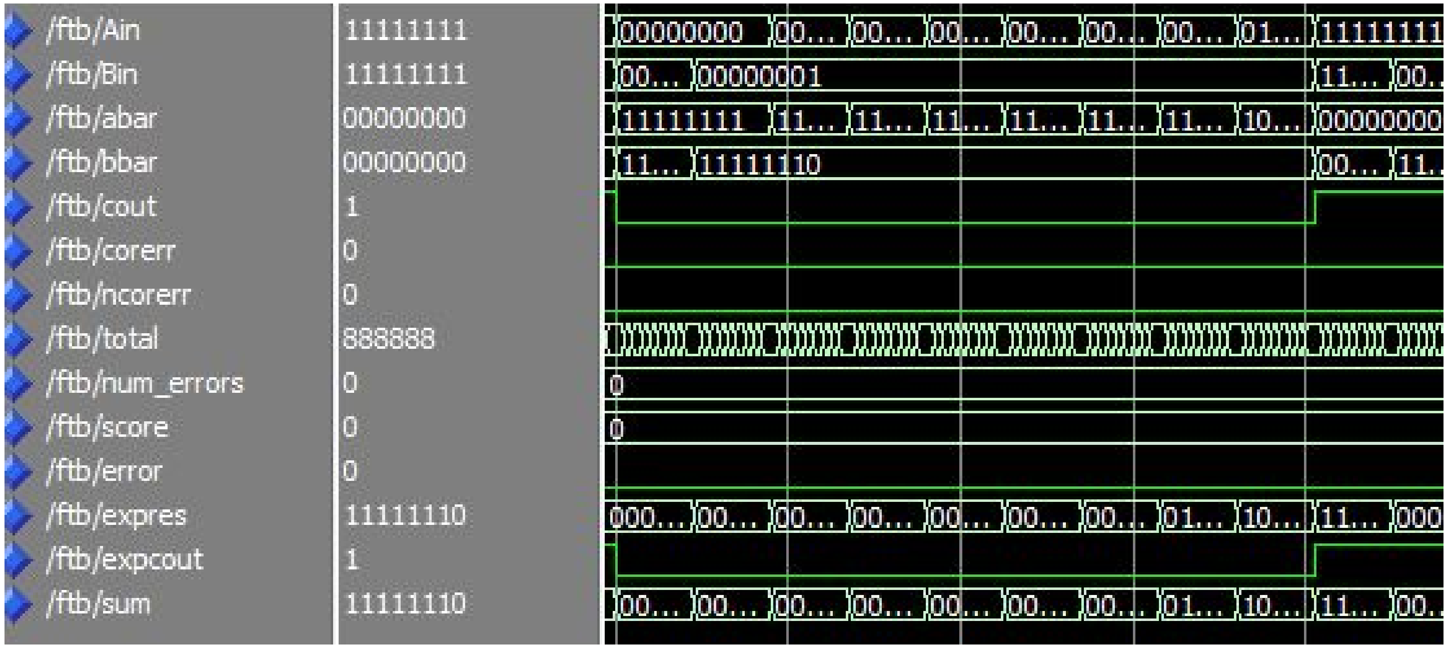
- Threshold reset to .9099
- Same design – exhaustive simulation



Messages	
/testfbit/a	1
/testfbit/b	1
/testfbit/c	1
/testfbit/sum	1
/testfbit/cout	1
Now	450 ns
Cursor 1	0 ns

Testing of fault tolerant adder

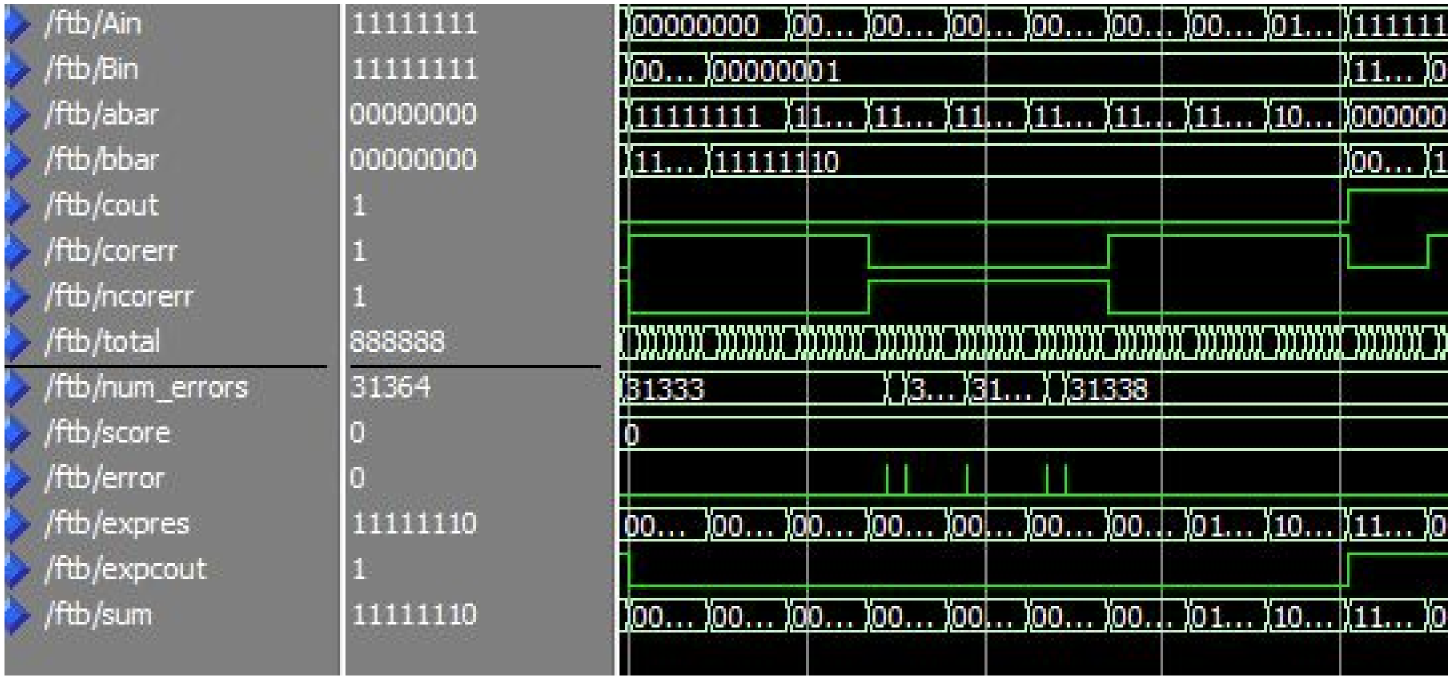
- Threshold = 1 - no error injection



/ftb/Ain	11111111
/ftb/Bin	11111111
/ftb/abar	00000000
/ftb/bbar	00000000
/ftb/cout	1
/ftb/corerr	0
/ftb/ncorerr	0
/ftb/total	888888
/ftb/num_errors	0
/ftb/score	0
/ftb/error	0
/ftb/expres	11111110
/ftb/expcout	1
/ftb/sum	11111110

Testing of fault tolerant adder

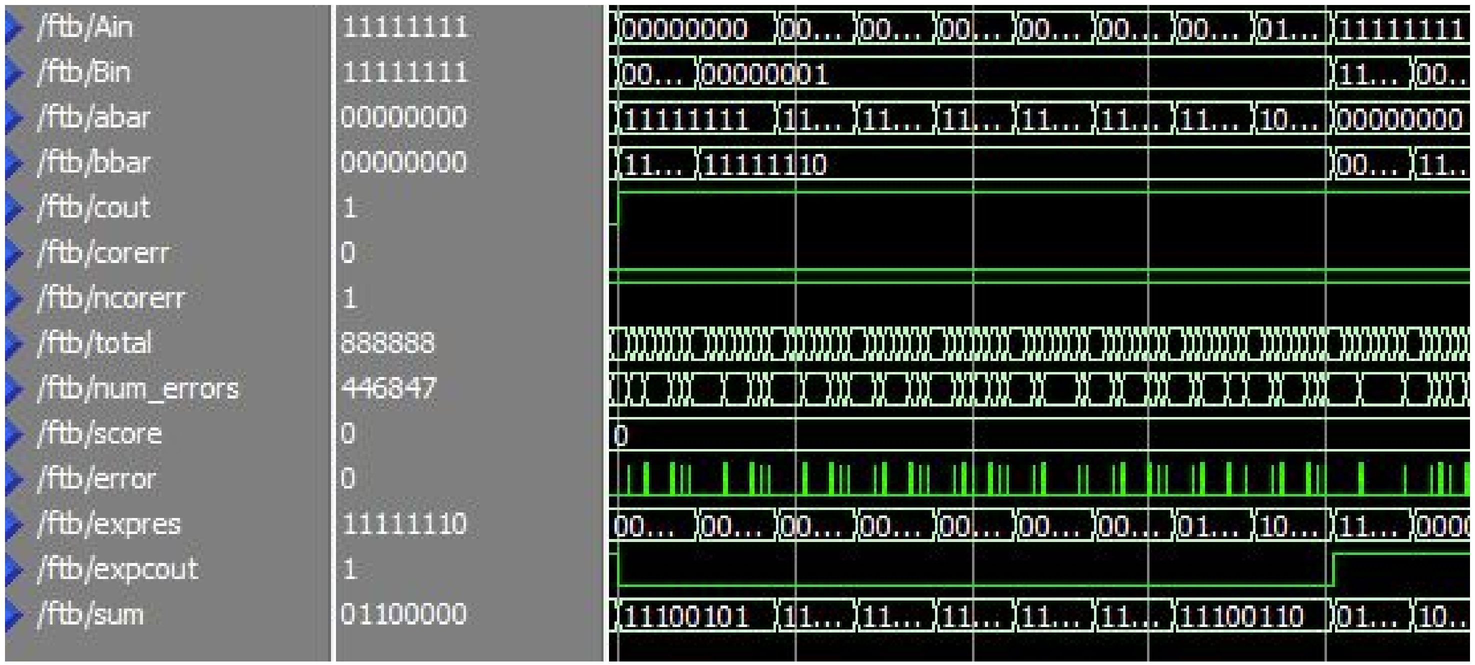
- Threshold = .99 error injection such that there are non correctable errors.



```
/ftb/Ain         11111111
/ftb/Bin         11111111
/ftb/abar        00000000
/ftb/bbar        00000000
/ftb/cout        1
/ftb/corerr      1
/ftb/ncorerr     1
/ftb/total       888888
/ftb/num_errors  31364
/ftb/score       0
/ftb/error       0
/ftb/expres      11111110
/ftb/expcout     1
/ftb/sum         11111110
```

Testing of fault tolerant adder

- Threshold = 0.70
- Most errors are noncorrectable



```
/ftb/Ain          11111111
/ftb/Bin          11111111
/ftb/abar         00000000
/ftb/bbar         00000000
/ftb/cout         1
/ftb/corerr       0
/ftb/ncorerr      1
/ftb/total        888888
/ftb/num_errors   446847
/ftb/score        0
/ftb/error        0
/ftb/expres       11111110
/ftb/expcout      1
/ftb/sum          01100000
```

Sponsored By:

Performance overhead

- Fault simulation comes at a cost
  - Simulation overhead time
- Overhead
  - Calls to random number generator
  - Comparison to threshold and error injection

The overhead numbers

- Baseline time come from TYPE bit
- Test of an 8-bit SEC/DED adder

TYPE	threshold	Time	Overhead
Bit	**---------**	**1:03.5**	**---------**
fbit	1.00000	1:16.2	20%
	0.99990	1:16.5	20.4%
	0.99000	1:21.4	28.2%
Std_logic	**---------**	**1.05.5**	**Vs bit 3%**
fsim_logic	1.00000	1:16.0	16.6%
	0.99990	1:16.6	17.5%
	0.99000	1:20.5	23.5%

Sponsored By:

Presentation Outline

- Background – What was needed and why
- What was readily available and prior work in the area
- The fault simulation types
  - fbit and fsim_logic
  - VHDL packages for fault simulation
- Use of fbit and fsim_logic to address the need
- Conclusions

Sponsored By:

Other points

- Before use package itself was verified
  - Package was verified
    - A couple of small errors were found and fixed before package was used.
- Also verified through use on simple designs
  - Individual logic operations
  - Full adder

Sponsored By:

Single gate performance

- Capability comes at a cost in simulation time and design modification
  - About 20% additional simulation time
  - Small amount of time to modify declaration types

DVCon 2013
Design & Verification Conference & Exhibition

Sponsored By:
accellera
SYSTEMS INITIATIVE

- Questions?