fsim_logic - A VHDL type for testing of FLYTRAP

by
Joanne E. DeGroat, Ph.D.
Associate Professor
The Ohio State University
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
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
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
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 $\rightarrow$ 100,000 possible locations for fixed error. (stuck at 0, stuck at 1)
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.
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

• 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.
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,fbit_vector) RETURN fbit_vector;
  FUNCTION "OR" (l,fbit_vector) RETURN fbit_vector;
  FUNCTION "NAND" (l,fbit_vector) RETURN fbit_vector;
  FUNCTION "NOR" (l,fbit_vector) RETURN fbit_vector;
  FUNCTION "XOR" (l,fbit_vector) RETURN fbit_vector;
  FUNCTION "XNOR" (l,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", "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";
  ```
  
  14 of (total number of slides)
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
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.
**fsim_logic package declaration**

- **The declarative part (initial part)**

```plaintext
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', '0', '0', '0', '0', '0', '0'), -- 0
  ('U', 'X', '1', '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', 'H', 'W', 'X'), -- H
  ('U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X')  -- X
);
Error injection

- Error injection on logical operations
- Methodology the same as fbit

```
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" function
  CONSTANT and_table : fsim_logic_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', '1', 'X', 'X'), -- 1  
  ('U', 'X', '0', 'X', 'X', 'X', 'X', 'X', 'X'), -- Z  
  ('U', 'X', '0', 'X', 'X', 'X', 'X', 'X', 'X'), -- W  
  ('0', '0', '0', '0', '0', '0', '0', '0', '0'), -- L  
  ('U', 'X', '0', '1', 'X', 'X', '1', 'X', 'X'), -- H  
  ('U', 'X', 'X', 'X', 'X', 'X', 'X', '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";
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 replication – 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

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

24 of (total number of slides)
Example of use - errors

- Threshold reset to .9099
- Same design – exhaustive simulation
Testing of fault tolerant adder

- Threshold = 1 - no error injection
Testing of fault tolerant adder

- Threshold = .99 error injection such that there are non correctable errors.
Testing of fault tolerant adder

- Threshold = 0.70
- Most errors are noncorrectable
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

<table>
<thead>
<tr>
<th>TYPE</th>
<th>threshold</th>
<th>Time</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>---------</td>
<td>1:03.5</td>
<td>---------</td>
</tr>
<tr>
<td>fbit</td>
<td>1.00000</td>
<td>1:16.2</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>0.99990</td>
<td>1:16.5</td>
<td>20.4%</td>
</tr>
<tr>
<td></td>
<td>0.99000</td>
<td>1:21.4</td>
<td>28.2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Std_logic</th>
<th>---------</th>
<th>1:05.5</th>
<th>Vs bit 3%</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsim_logic</td>
<td>1.00000</td>
<td>1:16.0</td>
<td>16.6%</td>
</tr>
<tr>
<td></td>
<td>0.99990</td>
<td>1:16.6</td>
<td>17.5%</td>
</tr>
<tr>
<td></td>
<td>0.99000</td>
<td>1:20.5</td>
<td>23.5%</td>
</tr>
</tbody>
</table>
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
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
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
• Questions?