GENISYS – A Component Inversion Engine

Kalpesh Zinjuwadia
Master’s Thesis Defense
July 25, 2005

Committee:
Dr. Perry Alexander (Chair)
Dr. David Andrews
Dr. Arvin Agah
Presentation Overview

- Introduction
- Problem Statement
- Background
- Problem Solution
- GENISYS Overview
- Data Path Generator
- Component Inversion Engine
- Test Scenarios
- Related Work
- Conclusion & Future Work
Introduction

- Importance of Testing
- Testing cost in Product Development life-cycle
- Importance of Program Inversion in Testing
- Makes Testing process Complete & Robust
- Reduces cost associated with Failures
- Motivation for Program Inversion
Problem Statement

- Test an Inner-Component of a Structural Component. It is Test Component (TC)
- Structural Component is a Black-box System
- Need to find the Dependence Link from the System Interfaces (Inputs & Outputs) to the TC
- Invert the components between the TC and System Inputs to derive the required System Inputs to test TC
- Proposed Solution: GENISYS – A Component Inversion Engine
Background – Rosetta Specification Language

- Rosetta – System-level Specification Language
- Basic constructs: Facet, Package, Domain
- General Syntax:

```plaintext
package <package_label> :: <domain> is
facet <facet_label> (parameters) :: <domain> is
    <optional local variable declarations>
    begin
    <Terms>
    end facet <facet_label>
end packet <package_label>
```
Example:

```plaintext
class INVERTER_pkg {
    static
        facet INVERTER_fct (ipt_sig:: input bit; opt_sig:: output bit) :: state_based is
            begin
                L1: opt_sig’ = not(ipt_sig);
            end facet INVERTER_fct;
    end packet INVERTER_pkg;
}
```
Background – XML

- Similar to HTML
- Widely used for Data Storage & Representation
- Flexible – User-defined custom tags
- Strict – Tags are always in pairs. Opening tag has to have a closing tag
- W3C Standard
Example:

```xml
<contact_info>
  <address>
    <block>M</block>
    <street>main</street>
    <city>Lawrence</city>
    <state>Kansas</state>
  </address>
  <phone>
    <office>1234567</office>
    <home>7654321</home>
  </phone>
</contact_info>
```
XML...

- **XML Schema**
  - Defines elements that appear in an XML Document
  - Defines legal building blocks
  - Schema is an XML Document
  - Provides information regarding attributes of a node

- **Document Object Model (DOM) Parser**
  - XML Document Parser
  - Hierarchical Tree Representation of the XML Document
  - Provides APIs for parsing/modifying the document
  - Root – Top level element of the Tree
  - Leaf – Child element
Program Inversion:

- is a computational process to derive program input values to generate a given set of output values
- is a program that computes the inverse computation of another program
- inverts the terms in the given program from last to first

Function Inversion – For a given function $F: X \rightarrow Y$, derive inverse function $F^{-1}: Y \rightarrow X$, such that,

$$F(X) = Y \Rightarrow F^{-1}(Y) = X \Rightarrow F^{-1}(F(X)) = X$$
Problem Solution

- GENISYS Tool
- Generate the Dependence Links from the System Interface to Test Component
- Determine the Component Dependence Hierarchies along the inner-components in the Structural Component
- Invert the components along the Hierarchy to derive System Vectors
GENISYS Overview

- Rosetta Specifications
- Data Path Generator
- Component Inversion Engine
- Component Dependence Hierarchy in XML
- System Vectors in XML
- Test Component Vectors

- GENISYS Inputs
- GENISYS Engines
- GENISYS Outputs
GENISYS - Introduction

- GENISYS - A Component Inversion Engine
- Two Phase Approach
- Data-Path Determination Phase
  - Determine Component Dependence Hierarchy
- Component Inversion Phase
  - Invert Components along the Hierarchy
Component Dependence Hierarchy

Levels in Hierarchy

System Input

System Output

Structural Component (Black Box)
Level in GENISYS

- Level determines the extent of interdependence between Component and TC
- Components at the same depth in Hierarchy belong to the same Level
- Component can belong to several Levels
- Driving Components:
  - Group Component in highest # Level it belongs to
- Driven Components:
  - Group Component in lowest # Level it belongs to
Level in GENISYS...

- Level 1L: Comp4, Comp6
- Level 2L: Comp5, Comp3
- Level 3L: Comp2
- Level 4L: Comp1
- Level 1R: Comp7, Comp9
- Level 2R: Comp8, Comp10, Comp12
- Level 3R: Comp11
Data-Path Determination Phase

- Rosetta Specifications
- Data Path Generator
- Component Dependence Hierarchy in XML
- Component Inversion Engine
- System Vectors in XML
- Test Component Vectors

Legend:
- GENISYS Inputs
- GENISYS Engines
- GENISYS Outputs
Data-Path Generator

- Data-path Engines Inputs:
  - Rosetta Specifications of the Structural Component
  - Test Component Vectors
- Generates two Component Dependence Hierarchies in XML:
  - Driving Hierarchy – Components directly/indirectly driving the TC
  - Driven Hierarchy – Components directly/indirectly driven by the TC
Component Dependence Hierarchy...

```
<COMPONENT_HIERARCHY file="foo.sld">
  <TEST_COMPONENT component_name="inverter">
    <DRIVING_COMPONENT component_name="amplifier1" driving_variable="volt_out1">
      <DRIVING_COMPONENT component_name="FET1" driving_variable="out_bit1">
      </DRIVING_COMPONENT>
    </DRIVING_COMPONENT>
  </TEST_COMPONENT>
</COMPONENT_HIERARCHY>

<COMPONENT_HIERARCHY file="foo.sld">
  <TEST_COMPONENT component_name="inverter">
    <DRIVEN_COMPONENT component_name="amplifier2" driving_variable="volt_out2">
      <DRIVEN_COMPONENT component_name="FET2" driving_variable="out_bit2">
      </DRIVEN_COMPONENT>
    </DRIVEN_COMPONENT>
  </TEST_COMPONENT>
</COMPONENT_HIERARCHY>
```
Feedback Loops

- Loop formed in Component Dependence Hierarchy
- Driven component directly or indirectly drives the driving component
- Solution - *Break* the last link that forms the feedback loop
- Simple & Efficient
Types of Feedback Loops

- **Self-Feedback Loop**
  
  Loop involving a single Component
  
  facet Comp1(A :: input bit; A :: output bit) :: logic is
  
  ... end facet Comp1;

- **Primary Feedback Loop**
  
  Loop involving two Components
  
  facet Comp1(A :: input bit; B :: output bit) :: logic is
  
  ... end facet Comp1;
  
  facet Comp2(B :: input bit; A :: output bit) :: logic is
  
  ... end facet Comp2;
Types of Feedback Loops…

- **Secondary Feedback Loop**

  *Loop involving more than two Components*

  facet Comp1(A :: input bit; B :: output bit) :: logic is
  ...
  end facet Comp1;
  facet Comp2(A :: input bit; C :: output bit) :: logic is
  ...
  end facet Comp2;
  facet Comp3(C :: input bit; A :: output bit) :: logic is
  ...
  end facet Comp3;
Non-Determinacy in Hierarchy

Non-Determinacy – A Parameter driven by more than one component

facet Comp1(B:: input real; A :: output real) :: logic is
...
end facet Comp1;

facet Comp2(C:: input real; A :: output real) :: logic is
...
end facet Comp2;

facet Comp3(A:: input real; D :: output real) :: logic is
...
end facet Comp3;
Component Inversion Phase

Rosetta Specifications → Data Path Generator

Data Path Generator → Component Dependence Hierarchy in XML

Component Dependence Hierarchy in XML → Component Inversion Engine

Component Inversion Engine → System Vectors in XML

System Vectors in XML → Test Component Vectors

GENISYS Inputs

GENISYS Engines

GENISYS Outputs
Component Inversion Algorithm

- Identify Invertible Components
- Assign Components to Hierarchy Levels
- Use Abstract Test Vectors of TC to perform Component Inversion for each Vector
- Invert all Components in each Level starting from Level 1L to Left-most Level in Hierarchy
- This will generate System Input parameters
Algorithm to Identify Invertible Components

- If a Component belongs to the Hierarchy,
  - it is Invertible
  - It is directly or indirectly related to TC
- If a Component doesn’t belong to the Hierarchy,
  - it is Non-Invertible
  - it is independent of TC
- Component Inversion Engine processes Invertible Components *only*
Algorithm to Invert a Component

- Populate the local information storage from Global information storage
- Invert all expression in order from last to first
- Populate local information storage with values computed after each expression inversion
- Populate Global information storage from local at the end
Conjunctive Normal Form (CNF)

- A common form of representing Boolean Expressions
- Logical AND of one or more Clauses
- A Clause consists of logical OR of one or more literals
- CNF comprises of conjunction of disjunctions of literals
- Literals comprises of variables or their complements
- Eg: \((A \lor B) \land (B \lor \neg C) \land D\)
Boolean Expression $\rightarrow$ CNF Conversion

- Eliminate the arrows
  \[(A \rightarrow B) \rightarrow \neg A \lor B\]

- Drive the negations in using De Morgan’s Law
  \[\neg (A \lor B) \rightarrow \neg A \land \neg B\]
  \[\neg (A \land B) \rightarrow \neg A \lor \neg B\]

- Distribute OR over AND
  \[(A \lor (B \land C)) \rightarrow (A \lor B) \land (A \lor C)\]
zChaff SAT Solver

- Implementation of the Chaff Solver
- Won the SAT 2002 Competition as the Best Complete Solver in both industrial and handmade benchmarks categories
- Tested on Solaris/Linux/Cygwin machines with g++ as the compiler
- Can be compiled with Visual Studio .Net under Windows
- Maintained by Zhaohui Fu, Princeton University
- Implemented in C++
zChaff SAT Solver – File Format

- CNF file name must end with .cnf extension
- Comment starts with a ‘c’ in the file
- Prelude Format
  
  \[ p \text{ cnf } \text{Num}_{\text{var}} \text{ Num}_{\text{clause}} \]

  \text{Num}_{\text{var}} \text{ & Num}_{\text{clause}} \text{ are } \# \text{ variables and } \# \text{ clauses in the expression}

- Variables are numbers from 1 to \text{Num}_{\text{var}}
- A literal can either be a variable or its complement.
- A complement is expressed as the negation of the number representing a variable.
- Eg: if 6 → x6 → -6 → ¬x6
A clause is a line of literals separated by spaces and ends with 0.

Eg: \((1 \ -2 \ 3 \ 0) \rightarrow (x1 \ V \ \neg x2 \ V \ x3)\)

A line with a single 0 terminates the CNF file.

Eg:

```
c CNF for: (x1 \ V \ \neg x2) \ \land \ (\neg x2 \ V \ \neg x3)
p cnf 3 2
1 -2 0
-2 -3 0
0
```
Component Inversion & SAT Solver

- All Rosetta expressions are Boolean
- Rosetta Expression $\rightarrow$ Boolean Expression
- Boolean Expression $\rightarrow$ CNF
- CNF expression fed to SAT Solver

E.g.,
- $F(x) = y$ is a Rosetta expression, where $F()$ is some function over unknown parameter $x$ & value of $y$ is $V$
- Find $x$, such that $F(x) = V$
- If $x$ is found $\rightarrow$ Expression is Satisfiable
- Otherwise the expression is not Satisfiable
Algorithm to invert If-Then-Else Expression

- An if-then-else expression is a boolean expression:
  \[ \text{if}(A) \text{ then } B \text{ else } C \Rightarrow (A \land B) \lor (\neg A \land C) \]
- Ensure the validity of the expression
- Traverse to the inner-most if-then-else expression
- Transform the expression to its boolean equivalent and pass it to its parent expression
- Recursively transform the if-then-else expression to its boolean till the whole expression is converted to its boolean equivalent
Algorithm to invert If-Then-Else Expression…

- Use the zChaff SAT Solver to solve the boolean equivalent of the if-then-else expression
- The result will be valid assignments for the sub-expressions of the if-then-else expression
Algorithm to invert If-Then-Else Expression…

- If (A) then
  if (B) then C else D end if
else if(E) then
  F else G end if end if;
- If(B) then C else D \(\Rightarrow (B \land C) \lor (\neg B \land D)\)
- If(E) then F else G \(\Rightarrow (E \land F) \lor (\neg E \land G)\)
- \((A \land ((B \land C) \lor (\neg B \land D))) \lor (A \land ((E \land F) \lor (\neg E \land G))\))
- DNF: \((A \land B \land C) \lor (A \land \neg B \land D) \lor (\neg A \land E \land F) \lor (\neg A \land \neg E \land G)\)
System Test Vectors

<TestData>
  <Config>
    <DataConfig>
      <Name>input_par1</Name>
      <Index>1</Index>
      <InputType/>
    </DataConfig>
    ...
  </Config>
  <TestSet>
    <TestVector>
      <Input>
        <Name>input_par1</Name>
        <Value>0</Value>
      </Input>
    </TestVector>
  </TestSet>
</TestData>
GENISYS Verification

Output Test Vectors

Rosetta Specifications

System Input Vectors

VectorGen™ DVTG Engine

Data Path Generator

Component Dependence Hierarchy

Component Inversion Engine

GENISYS Engine

Output Test Vectors

?
Testing & Example

- Inner Components
  - Trigger-based Circuit
  - Negative Trigger Circuit
  - Positive Trigger Circuit
  - Multiplexer Circuit
  - OR Gate Circuit
  - Inverter Circuit

- Rosetta Specification for all these inner components
Testing & Example...

- **Structural Component**

```plaintext
package STRUCT_COMPONENT :: logic is
    /* package body begins here */
    use INVERTER, POSITIVE_TRIGGER, NEGATIVE_TRIGGER, QUAD_MUX2X1, OR_GATE, TRIGGER_CIRCUIT;

facet STRUCT_COMPONENT ( /* interface parameters declared */
    a :: input  bit; B :: input  bit; d :: input  bit; h :: input  bit;
    G :: input  bit; x :: input  bit; z :: input  bit; W :: input  bit;
    u :: input  bit; Aa :: output bit; BB :: output bit; eE :: output bit;
    Ff :: output bit; DD :: output bit; Gg :: output bit; E :: input  bit;
    Y :: input  bit; V :: input  bit; HH :: output bit; CC :: output bit;
    zZ :: output bit ) :: state_based is
```

Inner-Components

Interface Parameters
/* locally declared variables */
   C, F, I, J, K, L, M :: bit;
/* facet body begins here */
begin
begin
   COMPONENT_1 : INVERTER(A, C);
COMPONENT_2 : POSITIVE_TRIGGER(C, B, F);
COMPONENT_3 : NEGATIVE_TRIGGER(F, D, I);
COMPONENT_4 : OR_GATE(I, F, ZZ);
COMPONENT_5 : QUAD_MUX2X1(A, B, C, D, E, F, G,
                              H, I, J, K, L, M);
COMPONENT_6 : TRIGGER_CIRCUIT(U, V, W, X, Y, Z,
                                  J, K, L, M, AA, BB,
                                  CC, DD, EE, FF, gg, HH);
end facet STRUCT_COMPONENT;
end package STRUCT_COMPONENT;
Test Scenarios – 1

(TRIGGER_CIRCUIT → TC)

<COMPONENT_HIERARCHY file="STRUCT_COMPONENT" >
  <TEST_COMPONENT component_name="COMPONENT_6" >
    <DRIVING_COMPONENT component_name="COMPONENT_5" driving_variable="J" >
      <DRIVING_COMPONENT component_name="COMPONENT_1" driving_variable="C" >
        <DRIVING_COMPONENT component_name="COMPONENT_2" driving_variable="F" >
          <DRIVING_COMPONENT component_name="COMPONENT_1" driving_variable="C" >
            ...
          </DRIVING_COMPONENT>
        </DRIVING_COMPONENT>
      </DRIVING_COMPONENT>
    </DRIVING_COMPONENT>
  </TEST_COMPONENT>
</COMPONENT_HIERARCHY>

<COMPONENT_HIERARCHY file="STRUCT_COMPONENT" >
  <TEST_COMPONENT component_name="COMPONENT_6" >
  </TEST_COMPONENT>
</COMPONENT_HIERARCHY>

Driving Comp Hierarchy

Driven Comp Hierarchy
<TestScenarios – 1...

<TestData>
  <Config>
    <DataConfig>
      <Name>A</Name>
      <Index>1</Index>
      <InputType/>
    </DataConfig>
    . . .
  </Config>
  <TestSet>
    <TestVector>
      <Input>
        <Name>A</Name>
        <Value>0</Value>
      </Input>
      . . .
    </TestVector>
    . . .
  </TestSet>
  . . .
</TestData>

<LocalVar>
  <Name>C</Name>
  <Value>0</Value>
</LocalVar>
. . .
<Output>
  <Name>ZZ</Name>
  <Value>-</Value>
</Output>
. . .
</TestVector>
. . .
</TestSet>
. . .
</TestData>

System Vectors
Test Scenarios

- INVERTER $\rightarrow$ TC
  - No Driving Component Hierarchy
  - All Inner-Components are Driven by TC
- NEGATIVE-TRIGGER $\rightarrow$ TC
  - Has both Driving and Driven Comp. Hierarchies
- Invalid Specification Scenarios
  - Feedback Loop in Hierarchy
  - Non-Determinacy in Hierarchy
Related Work

- Program Inversion – *Edsger W. Dijkstra*
- Running Programs Backwards: The Logical Inversion of Imperative Computation – *Brian J. Ross*
- A Formal Approach to Program Inversion – *Wei Chen*
- Reverse Execution of Programs – *Bitan Biswas & R. Mall*
Conclusion

- Developed GENISYS, a tool for Component Inversion
- GENISYS produces Component Dependence Hierarchies for a given Component Inter-dependence Scenario
- GENISYS processes Invertible Components to derive System Vectors
- zChaff SAT Solver is used for inverting expressions in Rosetta language
- Hierarchy and System Vectors generated in XML format
Future Work

- Allow each inner-component to be structural component itself
- Invert Functions defined in Rosetta language
- Define inner-components in Structural component
- Optimize the Component Inversion algorithm. Keep track of all the valid solutions of the SAT Solver and try the best. Computation-intensive approach
Thank You!!!
Questions?