Achieving the Ideal Testing on HDL Programs

Onur Kilincceker

University of Paderborn, Paderborn, Germany.

Mugla Sitki Kocman University, Mugla, Turkey.

Ercument Turk

International Computer Institute, Ege University, Izmir, Turkey.

Moharram Challenger

International Computer Institute, Ege University, Izmir, Turkey.

Fevzi Belli

University of Paderborn, Paderborn, Germany.

Izmir Institute of Technology, Izmir, Turkey.

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

Outline

1. Introduction
2. Proposed Approach: Applying Ideal Testing
3. A Demonstrating Example
4. Results
5. Conclusion

Program testing can be used to show the presence of bugs, but never to show their ***absence****.*

– Dijkstra, 1971

We prove a fundamental theorem showing that properly structured tests are capable of demonstrating the ***absence of errors*** *in a program.*

– Goodenough/Gerhart, 1975

... it is possible to use testing to formally prove the ***correctness*** *of programs ...*

– Howden, 1978

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

1. Introduction

a) VLSI Testing

b) Reminder: Ideal Testing

c) Background

a) VLSI Testing

- Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining millions of transistors into a single chip.

Integrated Circuits (IC)	Developed	Number of transistors
Small-scale integration (SSI)	In the early 1960s	<1000
Large-scale integration (LSI)	In the 1970s	<10.000
Very-large-scale integration (VLSI)	In the early 1980s	>100.000

Figure: VLSI Technology

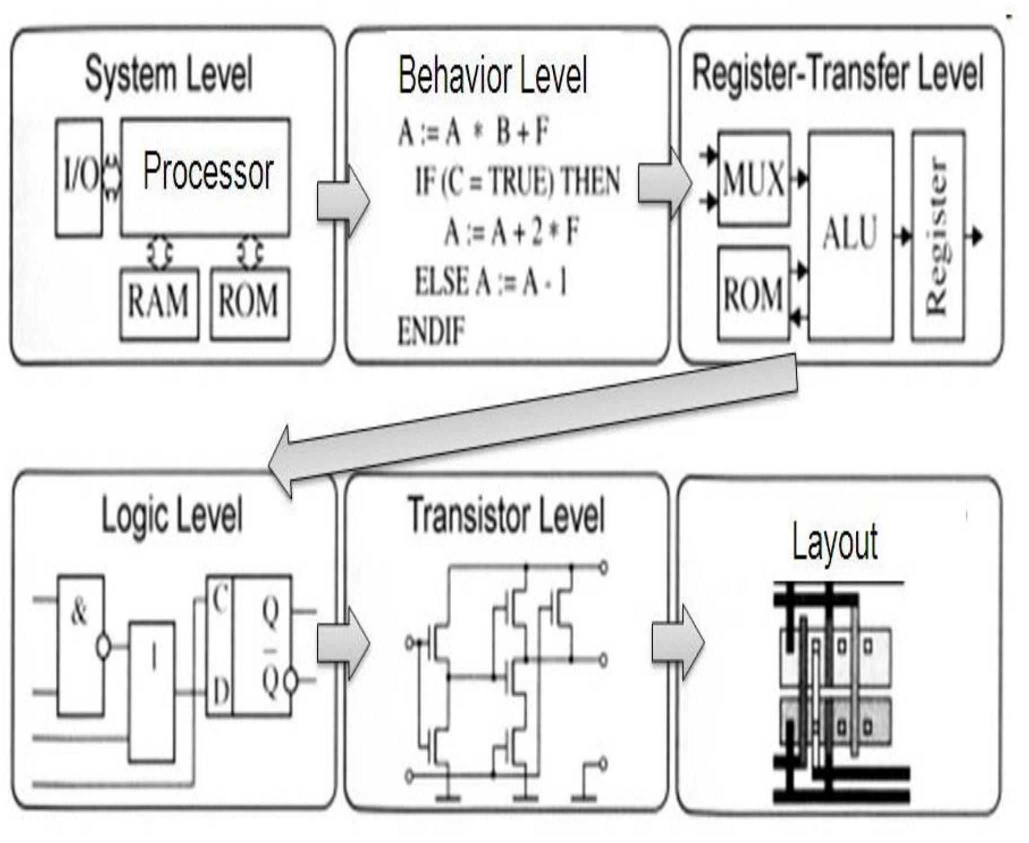
- Advances in VLSI technology have resulted in circuits with hundreds of millions of transistors and many new testing challenges.

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

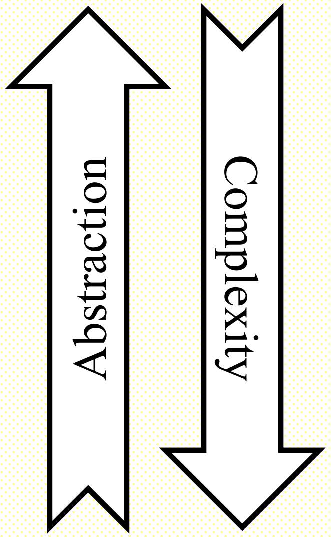
a) VLSI Testing (Cont'd)

i. Abstraction Levels in VLSI Design

- Abstraction defines how much details are included in the design.
- Testing methodologies differ based on abstraction levels.

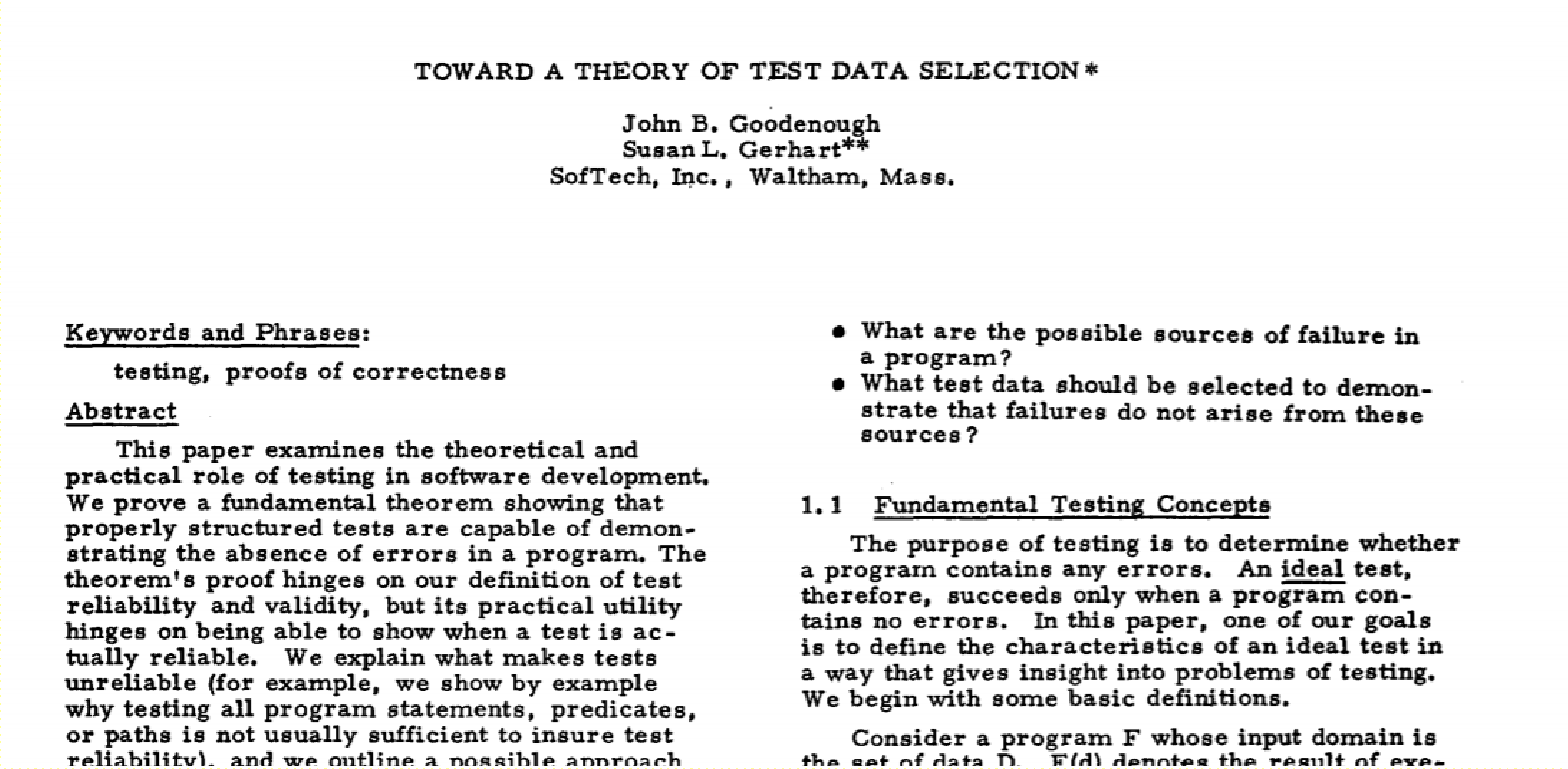


System Level
Behavioral Level
RTL Level
Gate (Logic) Level
Transistor Level
Layout Level



1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

b) Reminder: Ideal Testing*



TOWARD A THEORY OF TEST DATA SELECTION*

John B. Goodenough
Susan L. Gerhart**
SofTech, Inc., Waltham, Mass.

Keywords and Phrases:

testing, proofs of correctness

Abstract

This paper examines the theoretical and practical role of testing in software development. We prove a fundamental theorem showing that properly structured tests are capable of demonstrating the absence of errors in a program. The theorem's proof hinges on our definition of test reliability and validity, but its practical utility hinges on being able to show when a test is actually reliable. We explain what makes tests unreliable (for example, we show by example why testing all program statements, predicates, or paths is not usually sufficient to insure test reliability), and we outline a possible approach

- What are the possible sources of failure in a program?
- What test data should be selected to demonstrate that failures do not arise from these sources?

1.1 Fundamental Testing Concepts

The purpose of testing is to determine whether a program contains any errors. An ideal test, therefore, succeeds only when a program contains no errors. In this paper, one of our goals is to define the characteristics of an ideal test in a way that gives insight into problems of testing. We begin with some basic definitions.

Consider a program F whose input domain is the set of data D. F(d) denotes the result of exe-

*Goodenough, John B., and Susan L. Gerhart. "Toward a theory of test data selection." *IEEE Transactions on software Engineering* 2 (1975): 156-173.

1. Introduction
2. Proposed Approach:
   Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

b) Reminder: Ideal Testing (Cont'd)

Fundamental Theorem

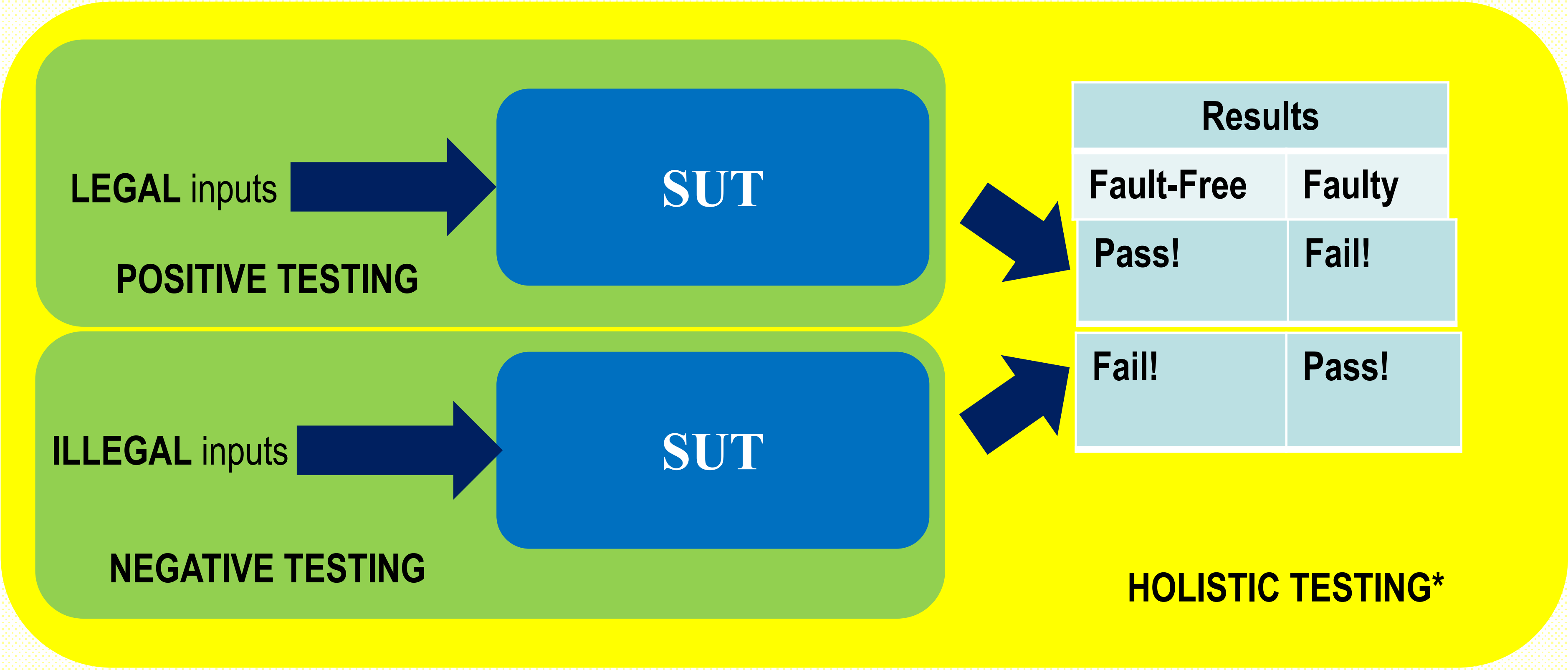
- Given: System Under Test (SUT), a Test T, and a test criterium C
  - Test *T* is *successful* if SUT delivers for any t of T a correct output (*t acceptable*), or T is fail if SUT delivers for any t of T a wrong output
  - Criterium *C* is *reliable* (*consistent*) if all tests, which satisfy C, are successful, OR, if all tests, which satisfy C, are not successful (fail)
  - Criterium *C* is *valid* (*effective*) if for each fault in SUT a t of T exists that satisfies C, and reveals the fault
- If a test selection criterium C Is reliable and valid, any test T that satisfies C is an *ideal* test.
- Fundamental Theorem: If an ideal test succeeds once, it will succeed all tests. Equally, if there is a bug in SUT, an ideal test will reveal it.

Goodenough, J. B. ; Gerhart, S. L. (1975)

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

c) Holistic Testing* (Positive and Negative Testing)

- **Positive Testing** is testing process where is system is checked against the **legal** input data.
- **Negative Testing** is testing process where is system is checked against the **illegal** input data.



Results	
Fault-Free	Faulty
Pass!	Fail!
Fail!	Pass!

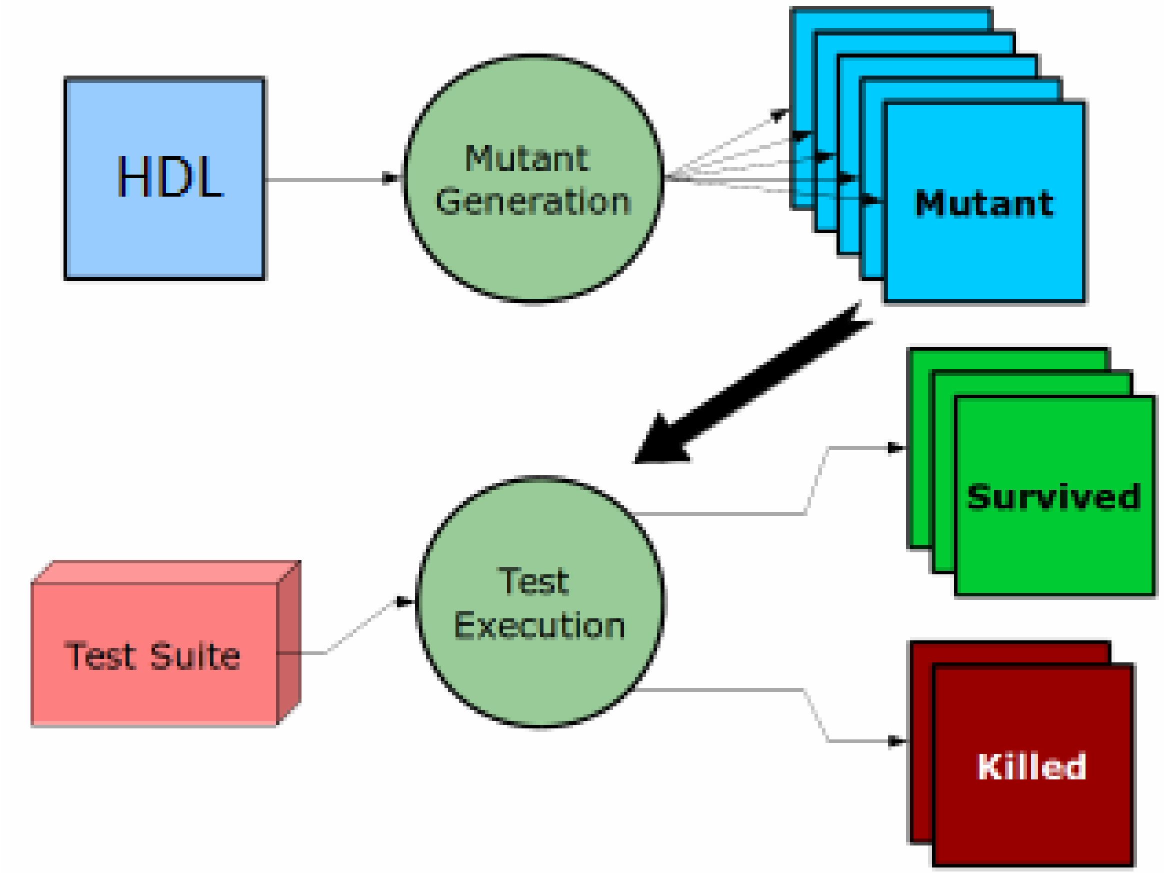
*Belli, Fevzi. "Finite state testing and analysis of graphical user interfaces." *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*. IEEE, 2001.

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

d) Mutation Testing

- Mutation Testing is a fault-based testing and widely used in software testing. It has also applied to VLSI testing.
- It requires injecting simple faults into original programs to produce set of erroneous versions, called *mutants.*
- *Mutation operators* are applied to original programs to generate mutants. There are plenty of mutation operators exist with respect to type of program syntax.
- Generated set of test sequences are executed on original programs and mutants to decide whether to kill it by comparing output results. If it isn’t killed, it is live (survived) depending on the effectiveness of test sequences or *equivalency* of mutants.

d) Mutation Testing (Cont'd)



1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

e) Background

Finite State Automata (FSA) and Regular Expression (RegEx)

- A **FSA** is a quintuple (X,Y,S; f,g) (***Mealy Automaton***) with
  $X=\{x_1,\ldots,x_m\}$: inputs, $Y=\{y_1,\ldots,y_n\}$: outputs, $S=\{s_1,\ldots,s_p\}$: states
  (X, Y, S: finite sets)
  and
  f: $XxS \rightarrow Y$ (output function),
  g: $XxS \rightarrow S$ (state transition, or next-state function)
- We will assume that the next state will be unambiguously determined, thus we study only *deterministic* FSA.
- A **RegEx** T consists of symbols connected by the operations
  - ❑ Sequence („.“(dot) that can be left out: concatenation),
  - ❑ Selection („+“: exclusive or),
  - ❑ Iteration („*“: arbitrarily often repetition, Kleene´s star operation)
  - ❑ „+“: at least one occurrence of symbols
  - ❑ λ : empty event; $\omega := \lambda^*$ (empty word)

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

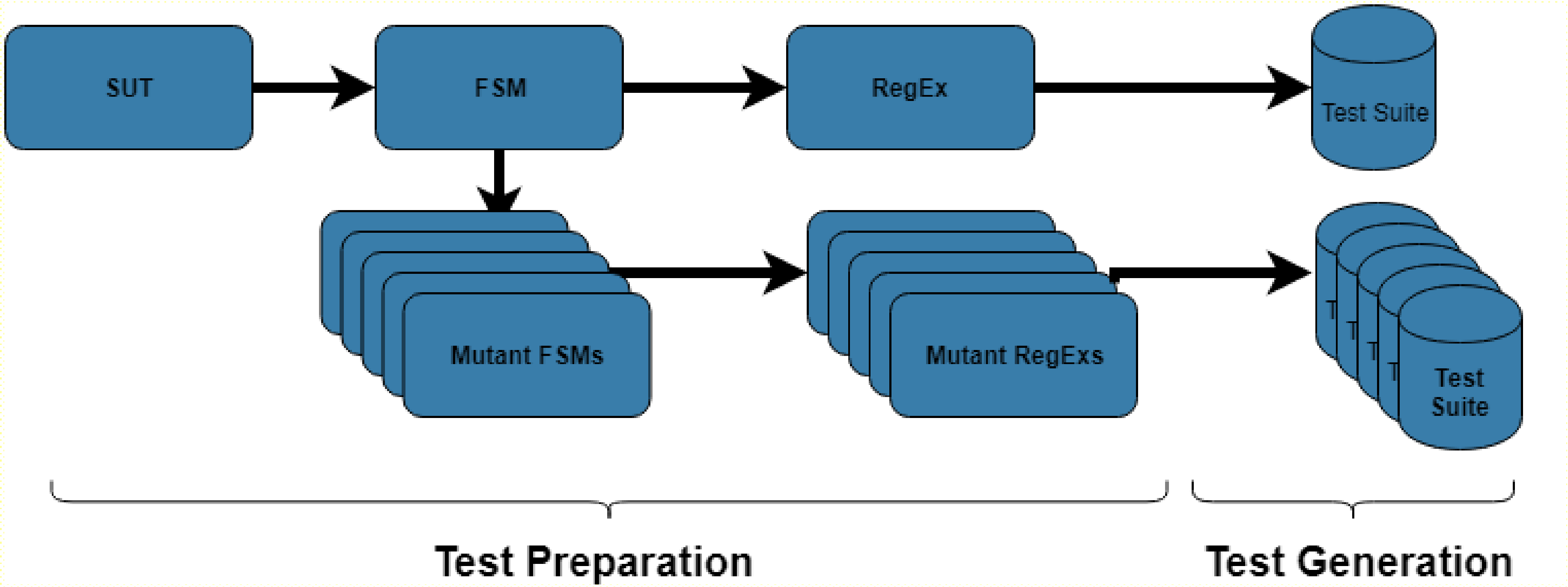
2. Proposed Approach: Extending Ideal Testing

a) Test Preparation and Test Generation

b) Test Execution and Test Selection

c) Achieving the Ideal Testing

a) Test Preparation and Test Generation



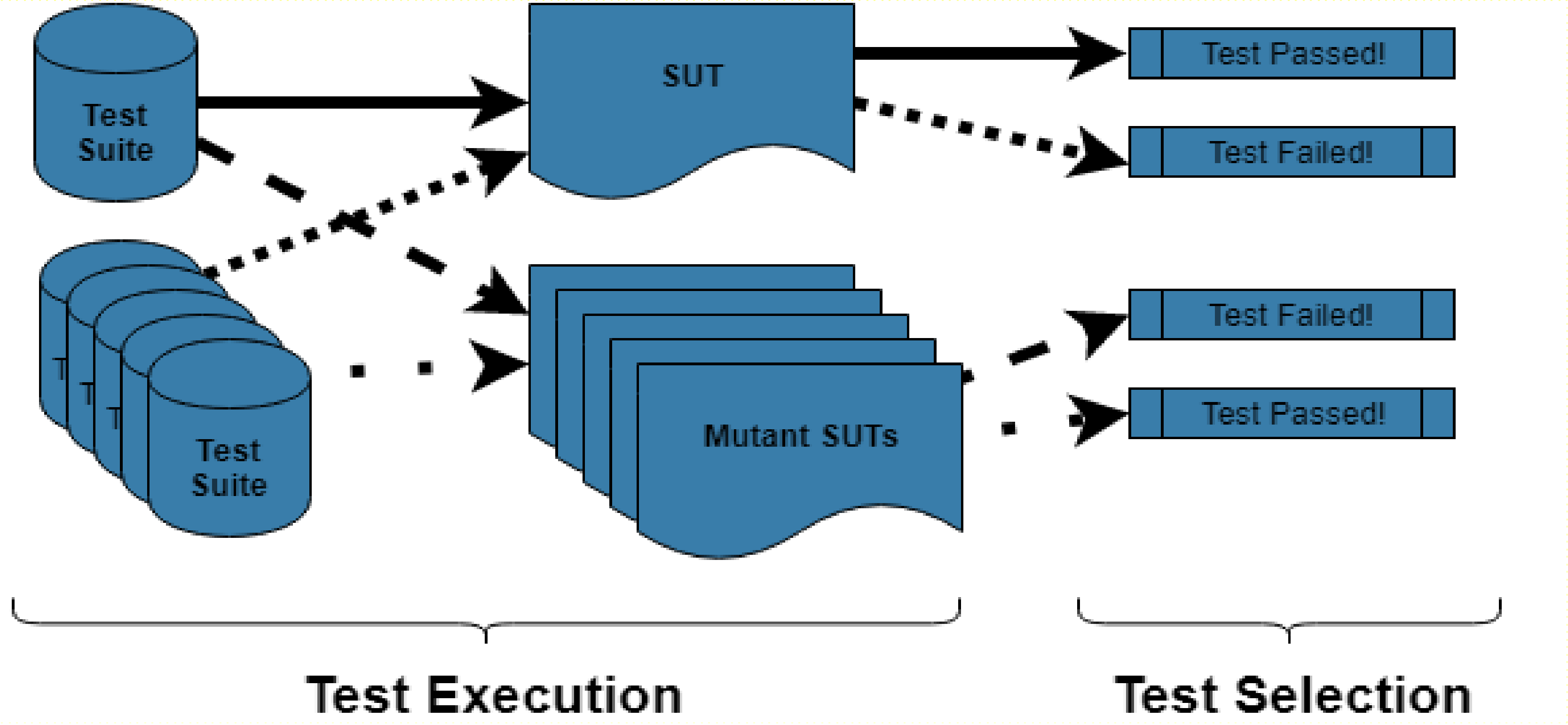
❖ Proposed approach combine holistic (positive and negative)** and mutation testing* to achive ideal testing which require definition of validity and reliability of pre-defined selection criteria

*DeMillo, Richard A., Richard J. Lipton, and Frederick G. Sayward. "Hints on test data selection: Help for the practicing programmer." *Computer* 11.4 (1978): 34-41.

**Belli, Fevzi. "Finite state testing and analysis of graphical user interfaces." *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*. IEEE, 2001.

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

b) Test Execution and Selection



1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

c) Achieving Ideal Testing

For given SUT, Criterium C1, C2, C3, and C4, defined below, are requirements of the test selection.

❖ In positive testing, test sequence ti is generated from original model.

❑ **C1 :** Select passed test sequence ti if it is applied to original SUT
❑ **C2 :** Select failed test sequence ti if it is applied to corresponding mutant SUT.

❖ In negative testing, test sequence tj is generated from a mutant model

❑ **C3 :** Select passed test sequence tj if it is applied to the corresponding mutant SUT
❑ **C4 :** Select failed test sequence tj if it is applied to the original SUT.

c) Achieving Ideal Testing(Cont’d)

- **Successful (T)** := $(\forall(t_i) \in T)OK(t_i)$ and **Fail(T)** := $(\forall(t_i) \in T) \neg OK(t_i)$
  By definition, t_i and t_j are either succesful or fail for each criterium.

- **Reliable(C)** := if all tests, which satisfy C, are successful, OR, if all tests, which satisfy C, are not successful (fail).
  By definition, C_1, C_2, C_3, and C_4 are reliable.

- **Valid(C)** := if for each error in SUT a t of T exists that satisfies C, and reveals the error
  Criterium C_1 and C_3 are **not valid** because tests, satisfying them, do not reveal any fault.
  However, Criterium C_2 and C_4 are **valid** because tests, satisfying them, reveals a fault.

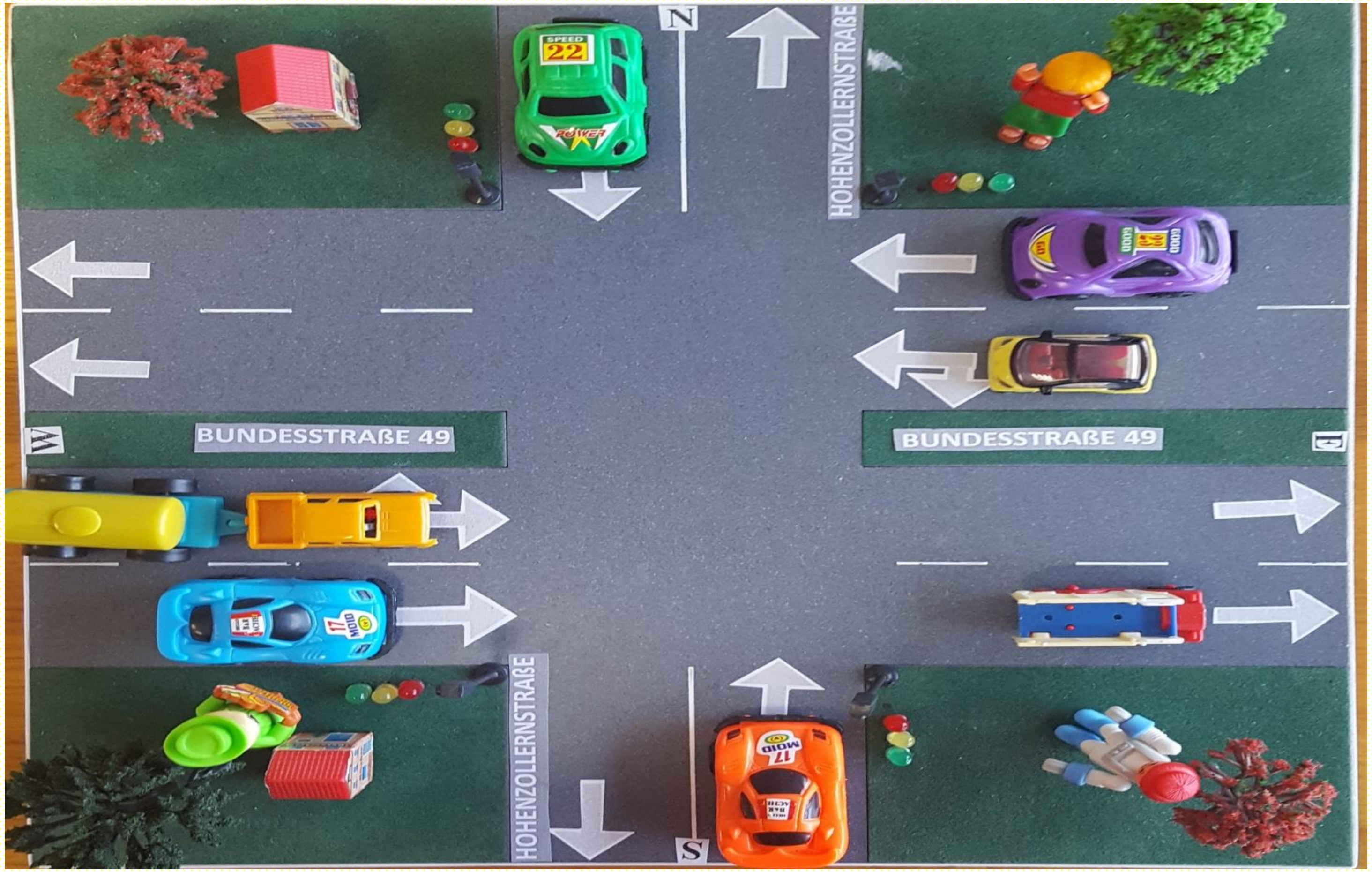
Therefore, criterium C_2 and C_4 are ***reliable*** and ***valid***, thereby tests t_i and t_j constitutes ***ideal test.***

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

3. A demonstrating Example
   a) System Under Test (SUT)
   b) Test Preparation
   c) Test Generation
   d) Test Execution
   e) Test Selection

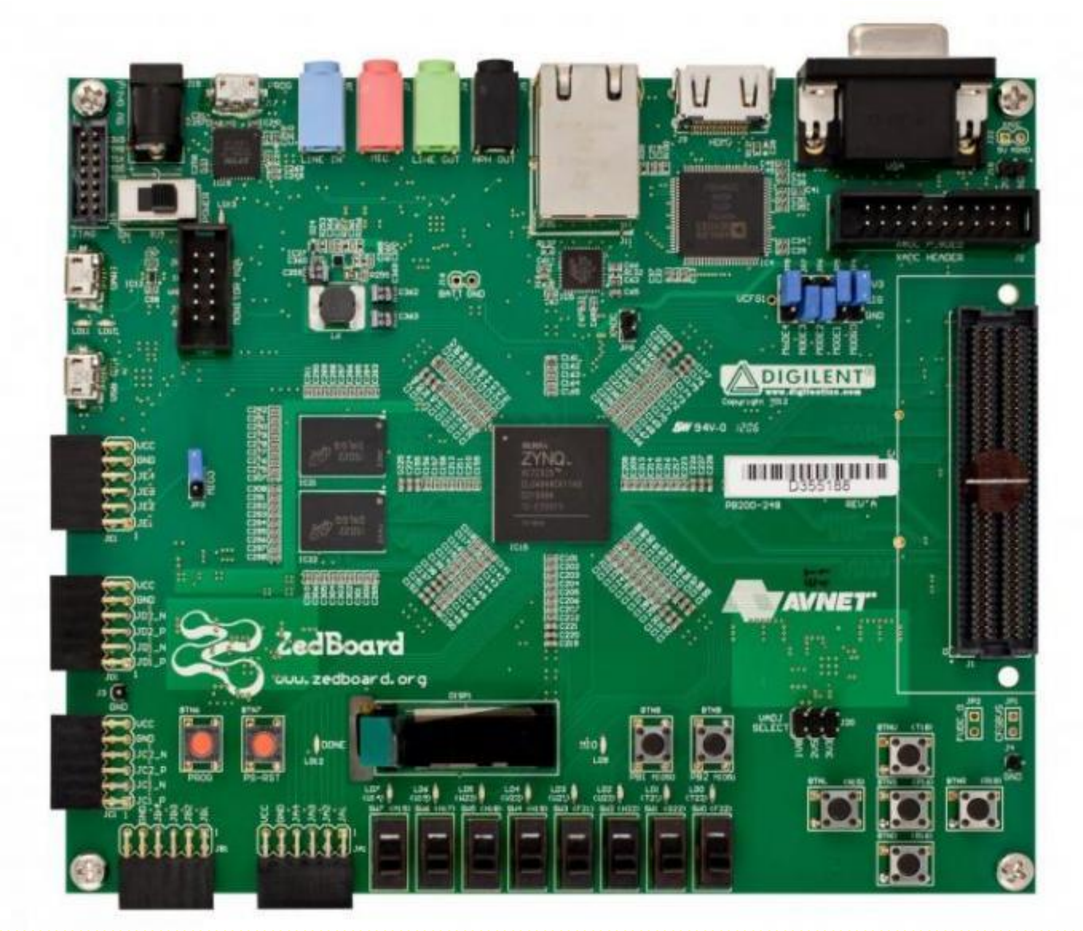
a) System Under Test (SUT): Traffic Light Controller



- We assume that there are four traffic light poles, one each at East, West, South, and North of the crossroads
- Green Light sequence: East-West-South-North

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

a) SUT: Traffic Light Controller (Cont'd)

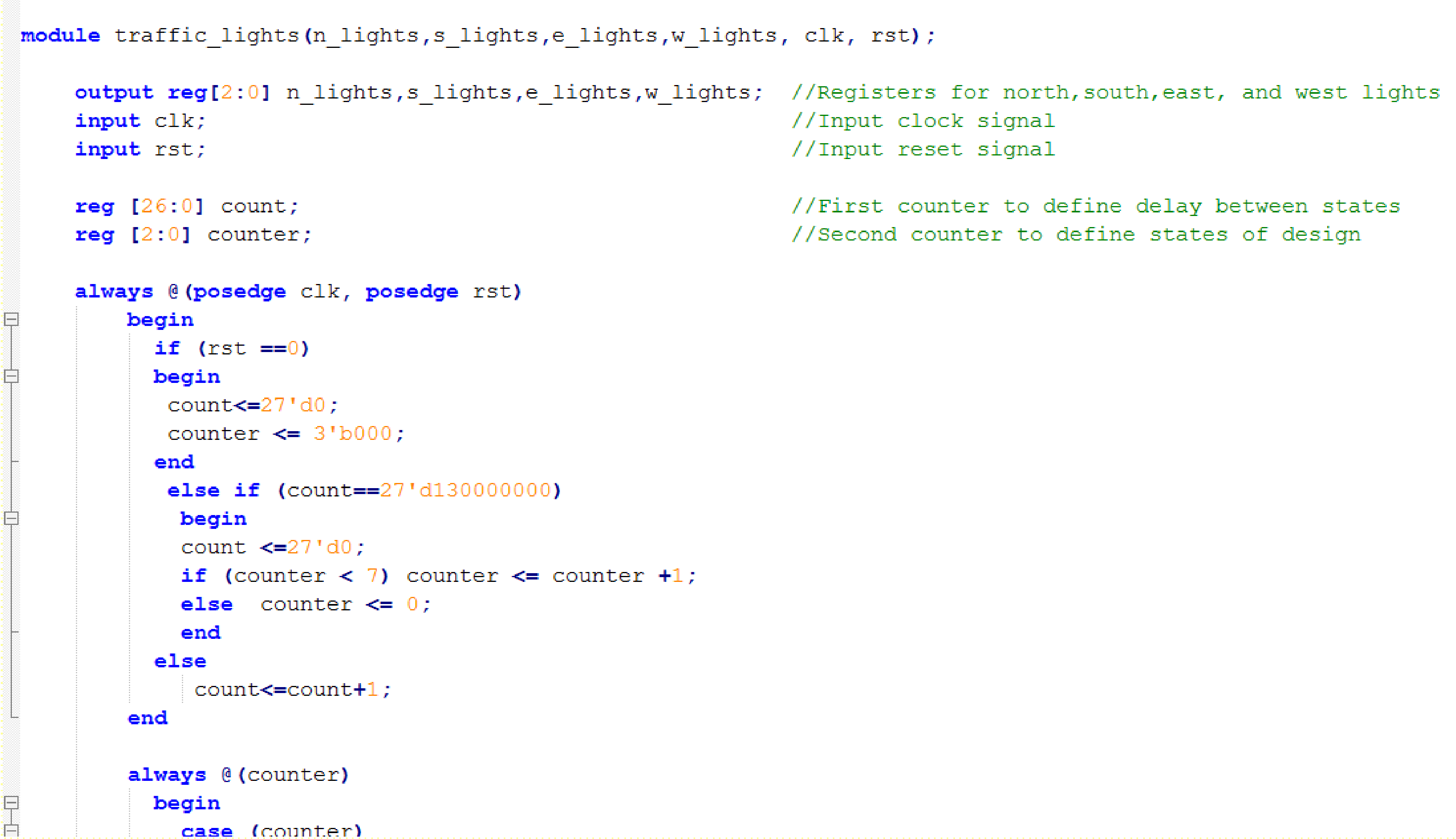


- The controller is implemented in *Verilog Hardware Description Language* (HDL) using a *Xilinx Zedboard Zynq-7000 All Programmable SoC*

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

a) SUT: Traffic Light Controller (Cont’d)

Behavioral Level: Verilog Source code of Traffic Light Controller



```
module traffic_lights(n_lights,s_lights,e_lights,w_lights, clk, rst);

    output reg[2:0] n_lights,s_lights,e_lights,w_lights;  //Registers for north,south,east, and west lights
    input clk;                                            //Input clock signal
    input rst;                                            //Input reset signal

    reg [26:0] count;                                     //First counter to define delay between states
    reg [2:0] counter;                                    //Second counter to define states of design

    always @(posedge clk, posedge rst)
        begin
          if (rst ==0)
          begin
           count<=27'd0;
           counter <= 3'b000;
          end
           else if (count==27'd130000000)
            begin
            count <=27'd0;
            if (counter < 7) counter <= counter +1;
            else  counter <= 0;
            end
          else
             count<=count+1;
        end

        always @(counter)
          begin
            case (counter)
```

- The source code contains 350 lines and nine states in an always block using case statement.

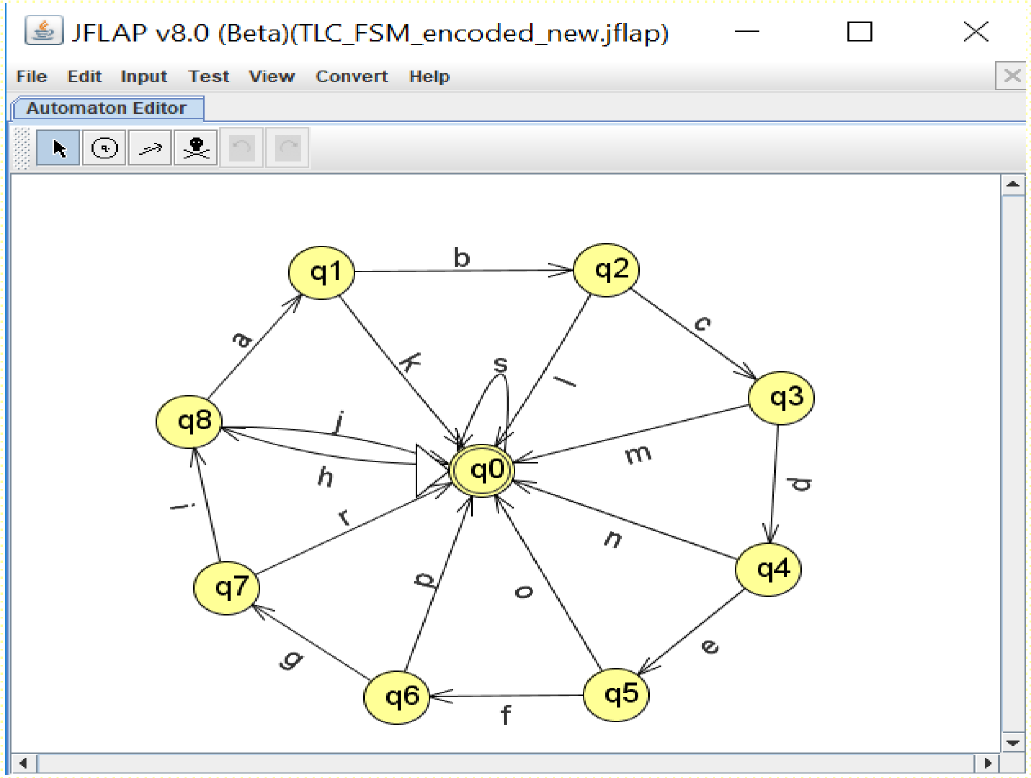
Achieving the Ideal Testing on HDL Programs

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

b) Test Preparation

Modeling Traffic Light Controller by a Finite State Automata (FSA)

(a) TLC model



(b)Machine model

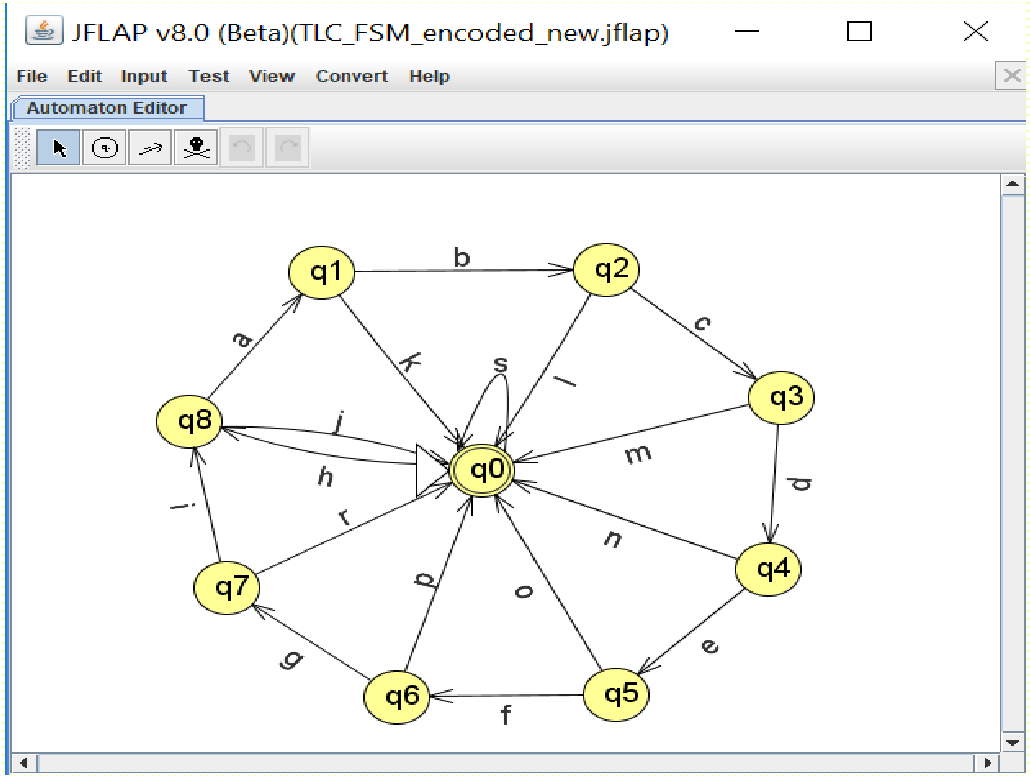
Encoding

a: grrr 0 / yrrr; b: yrrr 0 / rgrr; c: rgrr 0 / ryrr; d: ryrr 0 / rrgr; e:rrgr 0 / rryr; f: rryr 0 / rrrg; g: rrrg 0 / rrry; h: xxxx 0 / grrr; i: rrry 0 / grrr; j: xxxxx - grrr 0 / rrrr;...
where a transition is a combination of input and outputs of the circuit

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

b) Test Preparation (Cont’d)

Converting the FSA Model of Traffic Light Controller into a Regular Expression (RegEx)


JFLAP v8.0 (Beta)(TLC_FSM_encoded_new.jflap)
File Edit Input Test View Convert Help
Automaton Editor
q0
q1
q2
q3
q4
q5
q6
q7
q8
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
r
s

Encoding

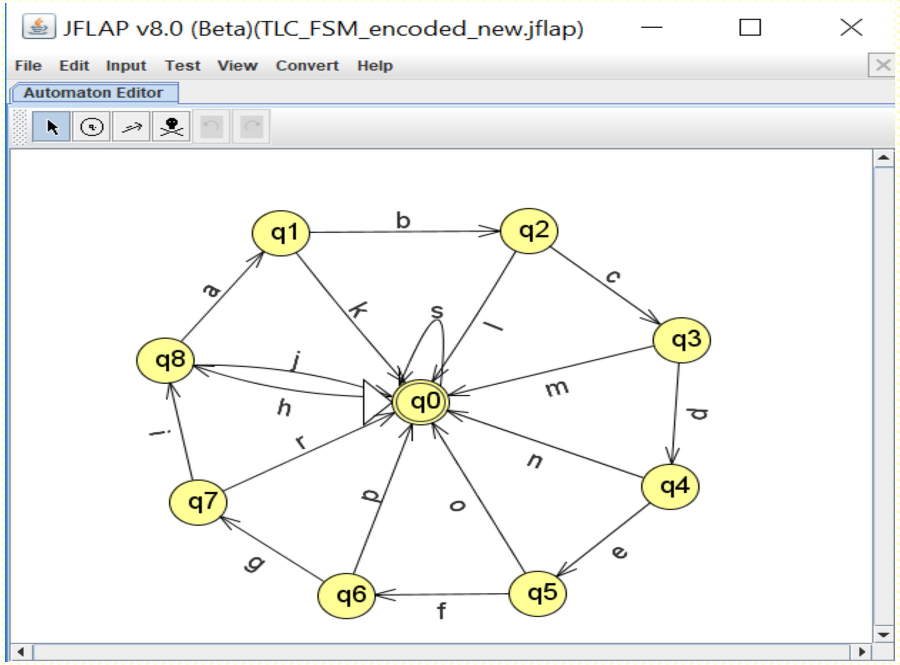
a: grrr 0 / yrrr; b: yrrr 0 / rgrr; c: rgrr 0 / ryrr; d: ryrr 0 / rrgr; e:rrgr 0 / rryr; f: rryr 0 / rrrg; g: rrrg 0 / rrry; h: xxxx 0 / grrr; i: rrry 0 / grrr; j: xxxxx - grrr 0 / rrrr;...
where a transition is a combination of input and outputs of the circuit

[(h(abcdefgi)*(abcdefgr+abcdefp+abcdeo+abcdn+abcm+abl+ak+j)+s)*]

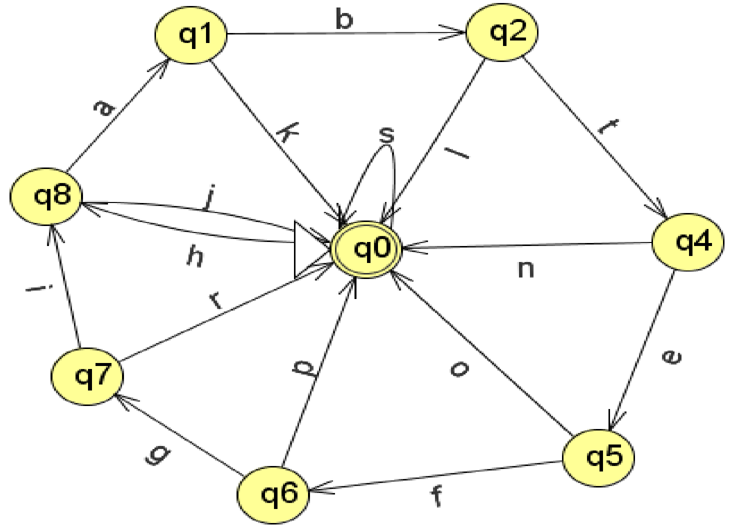
1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

b) Test Preparation (Cont'd)

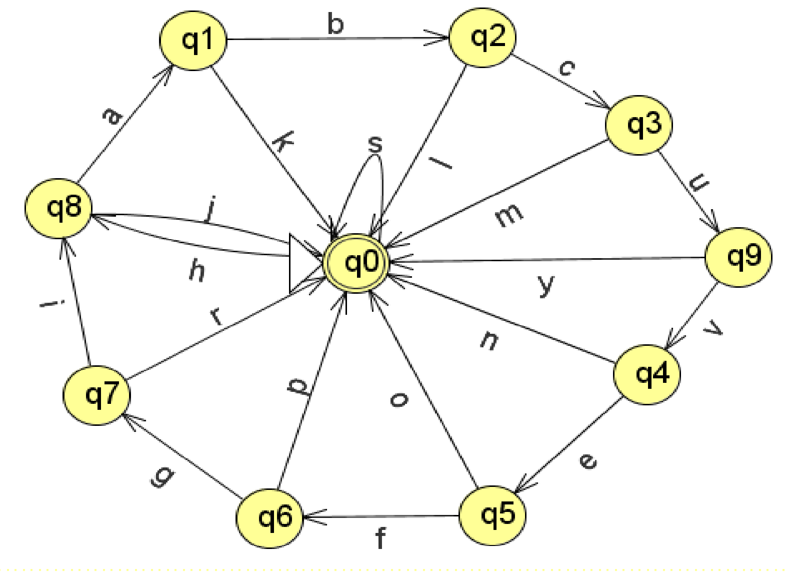
- As an example, FSM based faults, missing state and transition, extra state and transition, are considered.



Original Model



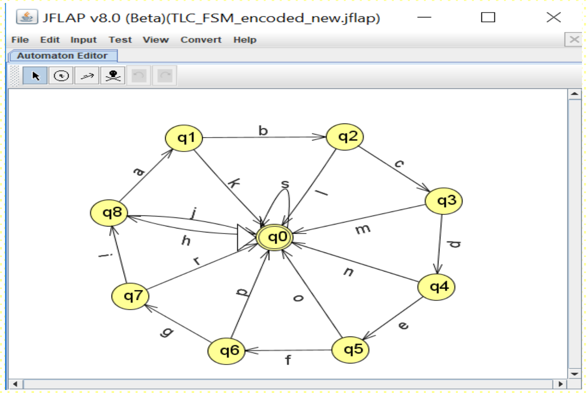
Missing State and Transition Fault



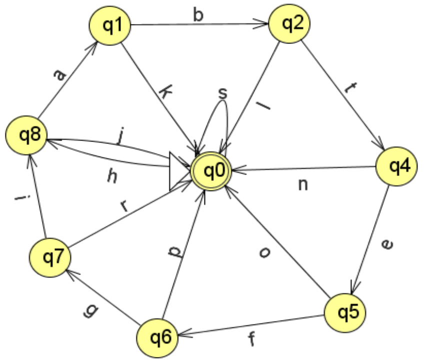
Extra State and Transition Fault

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

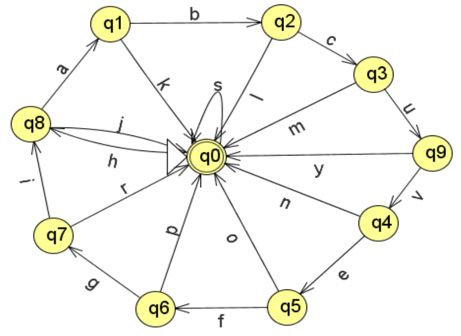
b) Test Preparation (Cont'd)


JFLAP v8.0 (Beta)(TLC_FSM_encoded_new.jflap)
File Edit Input Test View Convert Help
Automaton Editor
q0
q1
q2
q3
q4
q5
q6
q7
q8
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
r
s

Original Model



Missing State and Transition Fault



Extra State and Transition Fault

[(habcu(vefgiabcu)*(vefgi(abcm+abl+ak+j)+vefgr+vefp+veo+vn+y)+h(abcm+abl+ak+j)+s)*]

[(h(abtefgi)*(abtefgr+abtefp+abteo+abtn+abl+ak+j)+s)*]

[(h(abcdefgi)*(abcdefgr+abcdefp+abcdeo+abcdn+abcm+abl+ak+j)+s)*]

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

c) Test Generation

Original Model

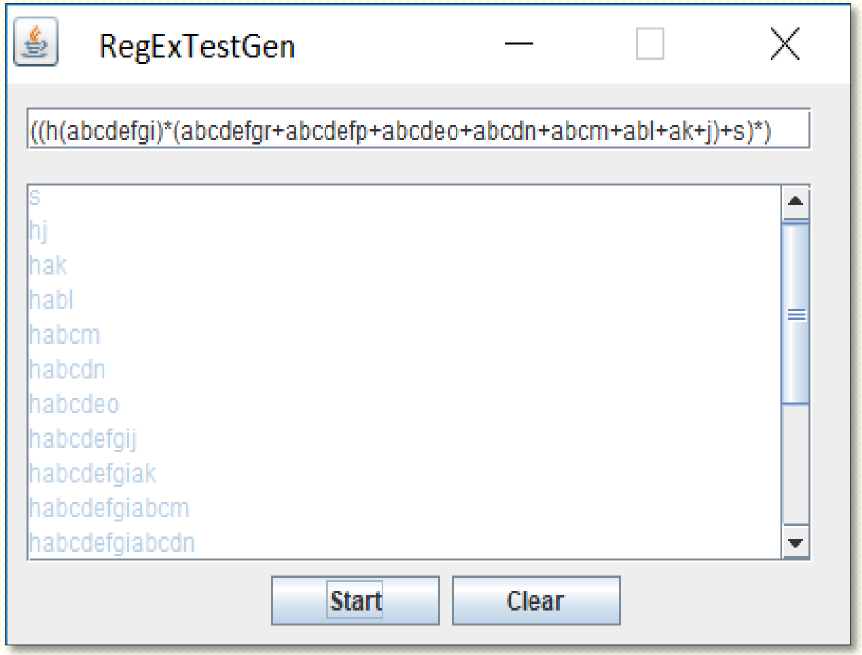
[(h(abcdefgi)*(abcdefgr+abcdefp+abcdeo+abcdn+abcm+abl+ak+j)+s)*]

Missing State and Transition Fault

[(h(abtefgi)*(abtefgr+abtefp+abteo+abtn+abl+ak+j)+s)*]

Extra State and Transition Fault

[(habcu(vefgiabcu)*(vefgi(abcm+abl+ak+j)+vefgr+vefp+veo+vn+y)+h(abcm+abl+ak+j)+s)*]



((h(abcdefgi)*(abcdefgr+abcdefp+abcdeo+abcdn+abcm+abl+ak+j)+s)*)

```
s
hj
hak
habl
habcm
habcdn
habcdeo
habcdefgij
habcdefgiak
habcdefgiabcm
habcdefgiabcdn
```

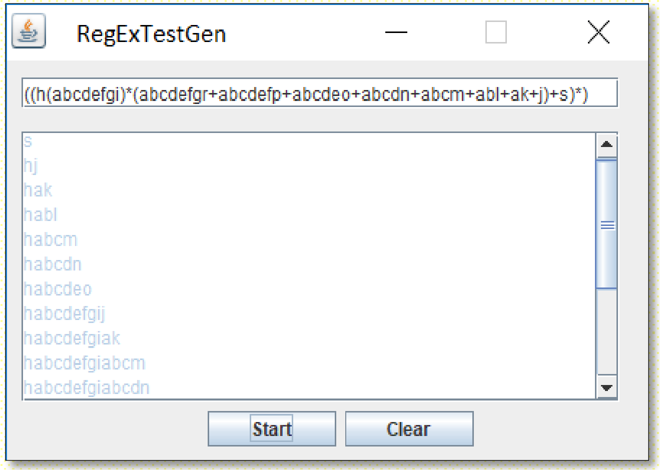
Regular Expression Based Test Generation Tool

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

d) Test Execution

Four cases are to carry out.

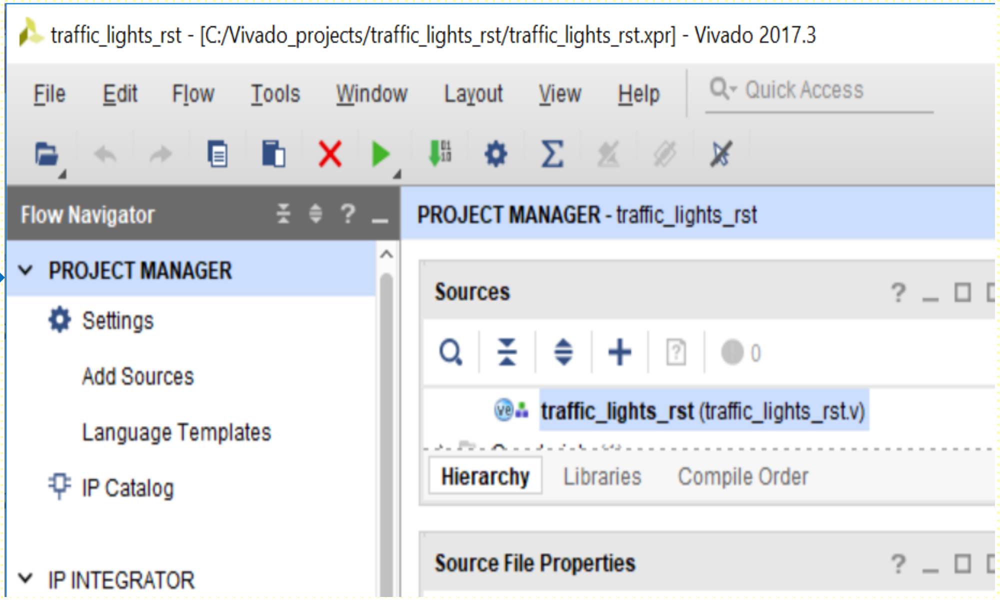
- (Positive Testing) Applying test sequences generated from **original model** to **original SUT**
- (Positive Testing) Applying test sequences generated from **original model** to **mutant SUTs**
- (Negative Testing) Appling test sequences generated from **mutant models** to **original SUT**
- (Negative Testing) Applying test sequences generated from **mutant models** to **mutant SUTs**



```
((h(abcdefgi)*(abcdefgr+abcdefp+abcdeo+abcdn+abcm+abl+ak+j)+s)*)
```

```
s
hj
hak
habl
habcm
habcdn
habcdeo
habcdefgij
habcdefgiak
habcdefgiabcm
habcdefgiabcdn
```

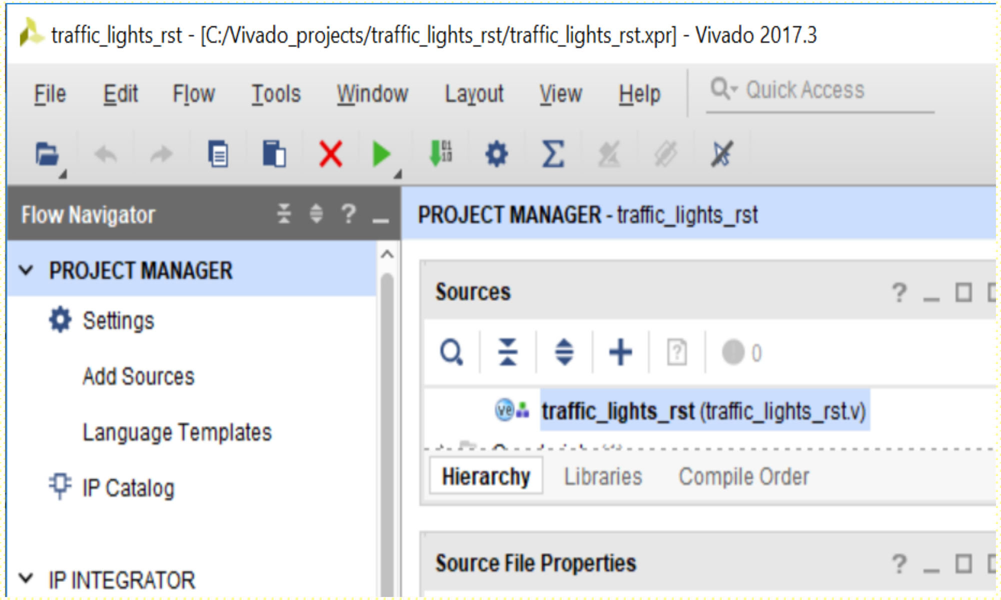
Regular Expression Based Test Generation Tool

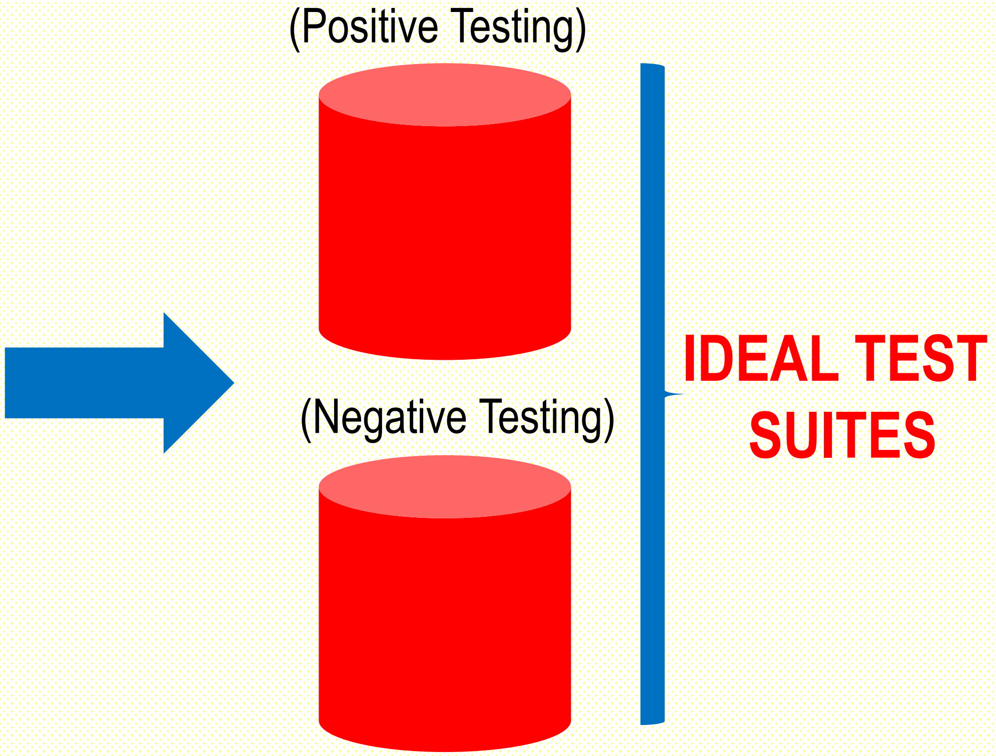

traffic_lights_rst - [C:/Vivado_projects/traffic_lights_rst/traffic_lights_rst.xpr] - Vivado 2017.3

Vivado 2017.4 design suite

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

e) Test Selection





1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

- ❖ Ideal Test Suites are constructed for positive and negative testing based defined selection criteria
- ❖ Ideal Test Suites are used to test presence or absence of faults
- ❖ Proposed method detects modeled faults, if not it also guaranties absence of faults based on Ideal Testing
- ❖ In example, corresponding test suites for modeled faults are constructed succesfully.

Testing	Fault-free SUT	Mutant SUT
(i)Positive	Test passed!	-
(iii)Negative	Test failed!	-
(ii)Positive	-	Test failed!
(iv)Negative	-	Test passed!

Table: Criteria for Test Selection

Achieving the Ideal Testing on HDL Programs

4. Results

The advantages of the proposed approach are:

- Ideal testing guaranties coverage of modeled faults in positive and negative test generation based on mutation testing. It also guaranties that whether fault(s) are **present** or **absent**.
- Ideal testing proposes construction of **reliable** and **valid** test selection criteria in terms of defined requirements
- It offers **higher-level** test generation based on RE which provides higher **abstraction, compactness and conformity** compared to traditional test generation algorithms

Further perspectives

❖ Comparing effectiveness of selected model for mutation testing

❖ Coverage of behavioral level faults

❖ Scalability of proposed approach

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion

5. Conclusion

The contributions of the proposed approach are listed:

- Achieving the Ideal Testing for testing HDL program to target FSM faults
- Combining the Holistic and Mutation testing to test HDL programs
- Test sequence generation based on Regular Expression
- Evaluating the proposed framework with the traffic light controller example.

Achieving the Ideal Testing on HDL Programs

1. Introduction
2. Proposed Approach: Extending Ideal Testing
3. A Demonstrating Example
4. Results and Discussion
5. Conclusion