IEE5650 VLSI Testing
Podem Tutorial

Hao-Yu Yang

Feb. 23, 2010

Introduction

- Podem
  - A ATPG tool for stuck-at/transition fault model
  - Support logic simulation and fault simulation function
- We use Podem in this course
  - To implement various testing algorithms for assignments
- Requirements
  - Basic data structure concepts
    - Class, member, member function
  - C/C++ programming ability

Environment

- Under linux-like system
  - g++ 3.x
  - make 3.8+
  - flex 1.875+
  - bison 2.5.4+
- Under Windows system
  - Dev-c, Visual C++, etc
- Useful website for learning linux
  - 鳥哥的 **Linux** 私房菜
    - http://linux.vbird.org/

Setup

- Download the tarball “podem.tgz” from the course website
- Decompress the tarball
  - $ tar zxvf podem.tgz
- Check file list
  - $ ls podem/

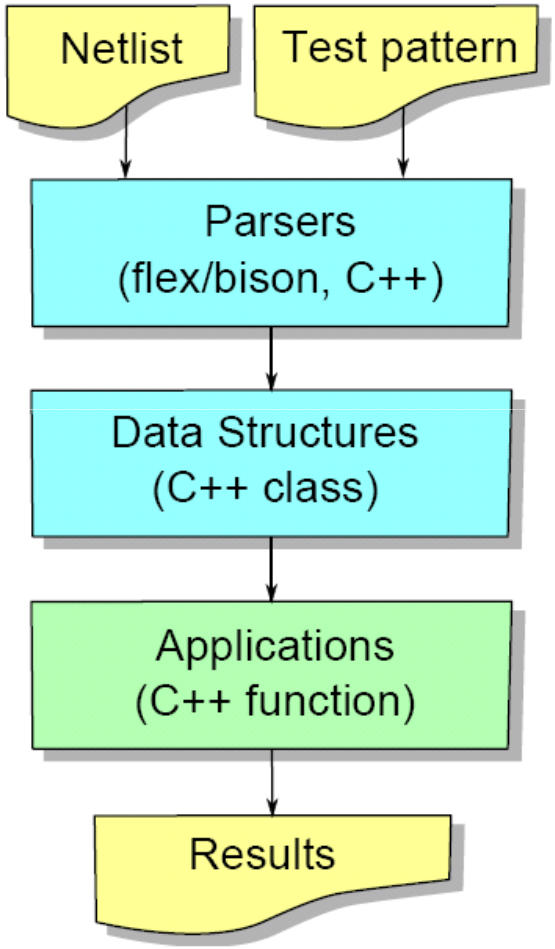
Setup

- Change directory
  - $ cd podem
- Compile the source code with Makefile
  - $ make
- An execution file “atpg” will be generated
  - $ ls atpg
- Clean compiled results (generally unnecessary)
  - $ make clean

How to Use Podem

- Show usage
  - $ ./atpg –help
  - usage: atpg [options] input_circuit_file
  - -help (print this help summary)
  - -logicsim (run logic simulation)
  - -plogicsim (run parallel logic simulation)
  - -fsim (run stuck-at fault simulation)
  - -stfsim (run single pattern single transition-fault simulation)
  - -transition (run transition-fault ATPG)
  - -input <$val> (set the input pattern file)
  - -output <$val> (set the output pattern file)
  - -bt [$val] (set the backtrack limit)
- Please refer "README.pdf" in directory for detailed functions and examples

Podem Scheme



- Input files
  - Netlist and Test pattern
- Parser
  - To translate files as data structures
- Data structures
  - CIRCUIT, GATE, and etc
- **Applications**
  - Logic/fault simulator and ATPG
  - Do homeworks in this layer
- Results
  - Fault coverage and test patterns
  - Performance information

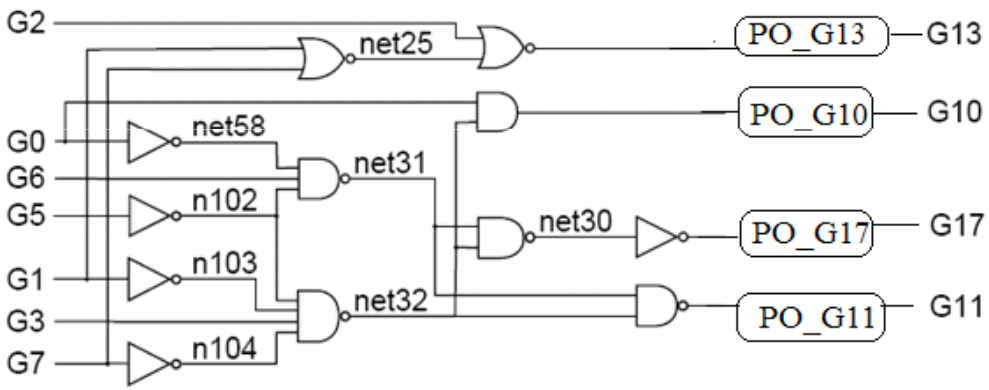
Arguments

- Setup (the SetupOption function in main.cc)
  - option.enroll(Name, InputValue, Description, 0);
  - InputValue
    - GetLongOpt::NoValue
    - GetLongOpt::MandatoryValue
    - GetLongOpt::OptionalValue
  - Example
    - option.enroll("logicsim", GetLongOpt::NoValue, "run logic simulation", 0);
    - option.enroll("input", GetLongOpt::MandatoryValue, "set the input pattern file", 0);

Arguments

- Usage
  - In code
    - option.retrieve(Name_of_agrument)
    - Example
      - if(option.retrieve("logicsim"))
  - In program run
    - ./atpg -logicsim [circuit_name]

Netlist Format




```
# s27.bench
INPUT(G0)
INPUT(G1)
INPUT(G2)
INPUT(G3)
<skip>
OUTPUT(G17)
OUTPUT(G10)
OUTPUT(G11)
<skip>
G17 = NOT(net30)
G10 = AND(net32, G0)
G11 = NAND(net32, net31)
<skip>
net25 = NOR(G7, G1)
net30 = NAND(net31, net32)
```

Test Pattern Format

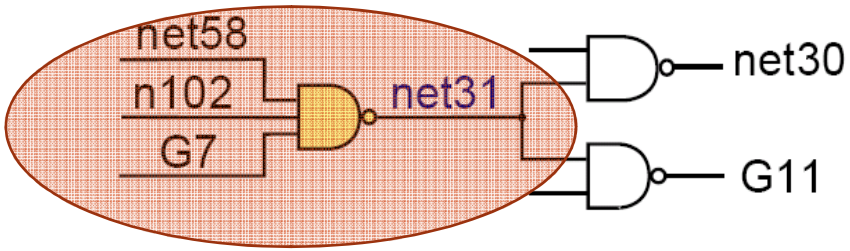
- Describe primary input names in the first line
- Describe primary input values in other lines
- Example: 7 test patterns for s27.bench
  - PI G0 PI G1 PI G2 PI G3 PI G5 PI G6 PI G7
  - 1100011
  - 0110010
  - 1001011
  - 1001010
  - 1101010
  - 0011111
  - 1011100

Data Structures - GATE

- Data members used in this assignment
  - Name – gate name
  - Value – S0, S1, X, D, B, ILLIGAL
  - Function – G_PI, G_PO, G_PPI, G_PPO, G_NOT, G_AND, G_NAND, G_OR, G_NOR, G_DFF, G_BUF
  - Input_list/Output_list – fanin/fanout list
- Data members used in later assignments
  - Value, Flag, Fault status, ATPG, status, etc

Data Structures Example

- Example: net31
  - Name = net31
  - Function – G_NAND
  - Input_list = {net58, n102, G7}
  - Output_list = {net30, G11}



Benches

- Circuits
  - | iscas85
    - cXXX.bench is the combinational ISCAS85 circuits
  - | iscas89_seq
    - sXXX_seq.bench is the original ISCAS89 circuits
  - | iscas89_opt
    - sXXX_opt.bench is the combinational parts of sXXX_seq.bench optimized with the Synopsys DC
  - | iscas89_com
    - sXXX_com.bench is the same as sXXX_opt.bench (We do not need these circuits at all.)