KU KONKUK UNIVERSITY

Simulation-based Testing

김재엽

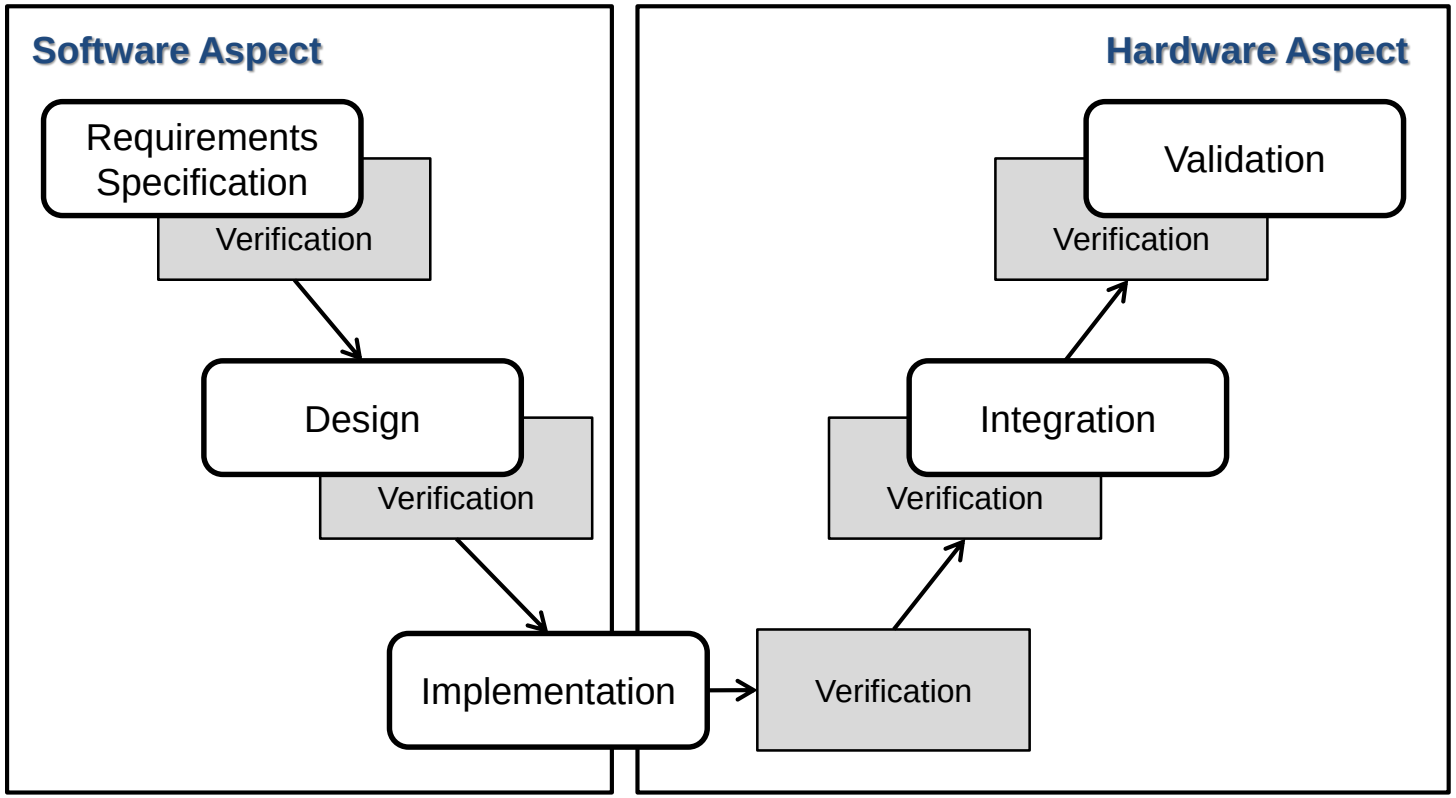
목차

- Simulation-based Testing
- Coverage

Simulation-based Testing

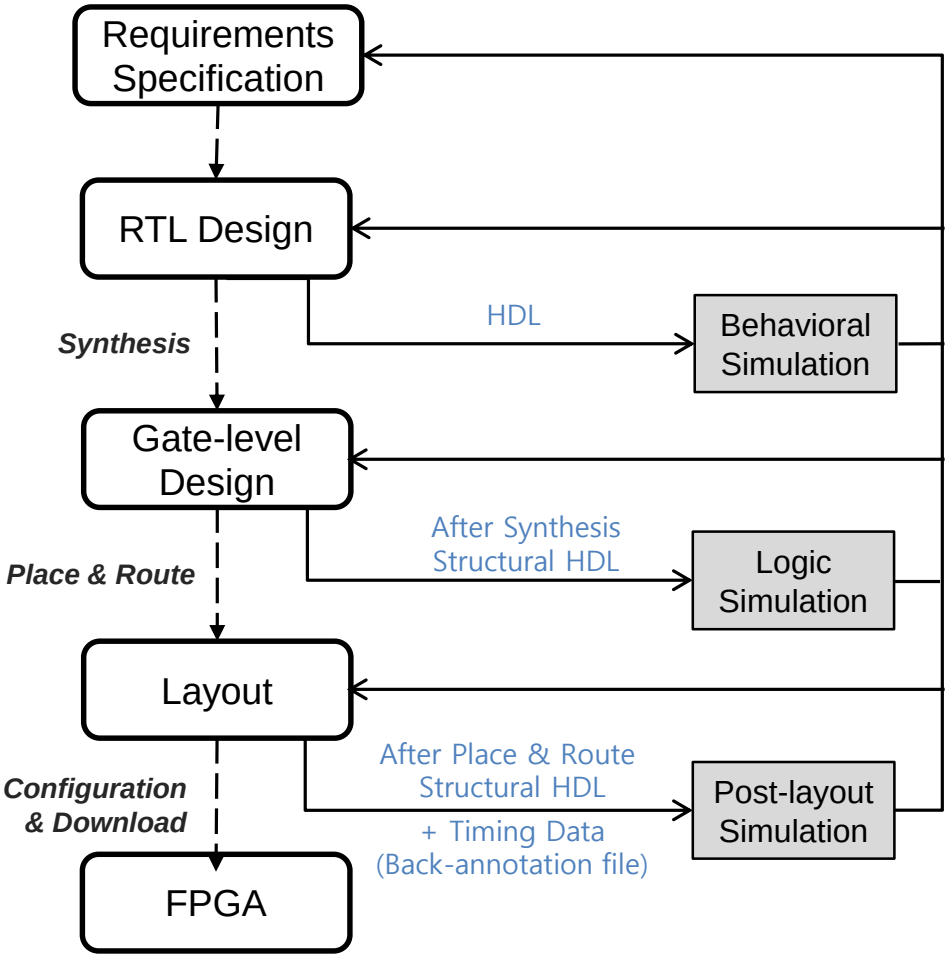
FPGA Development Process

- FPGA Development Process의 Software와 Hardware적 측면



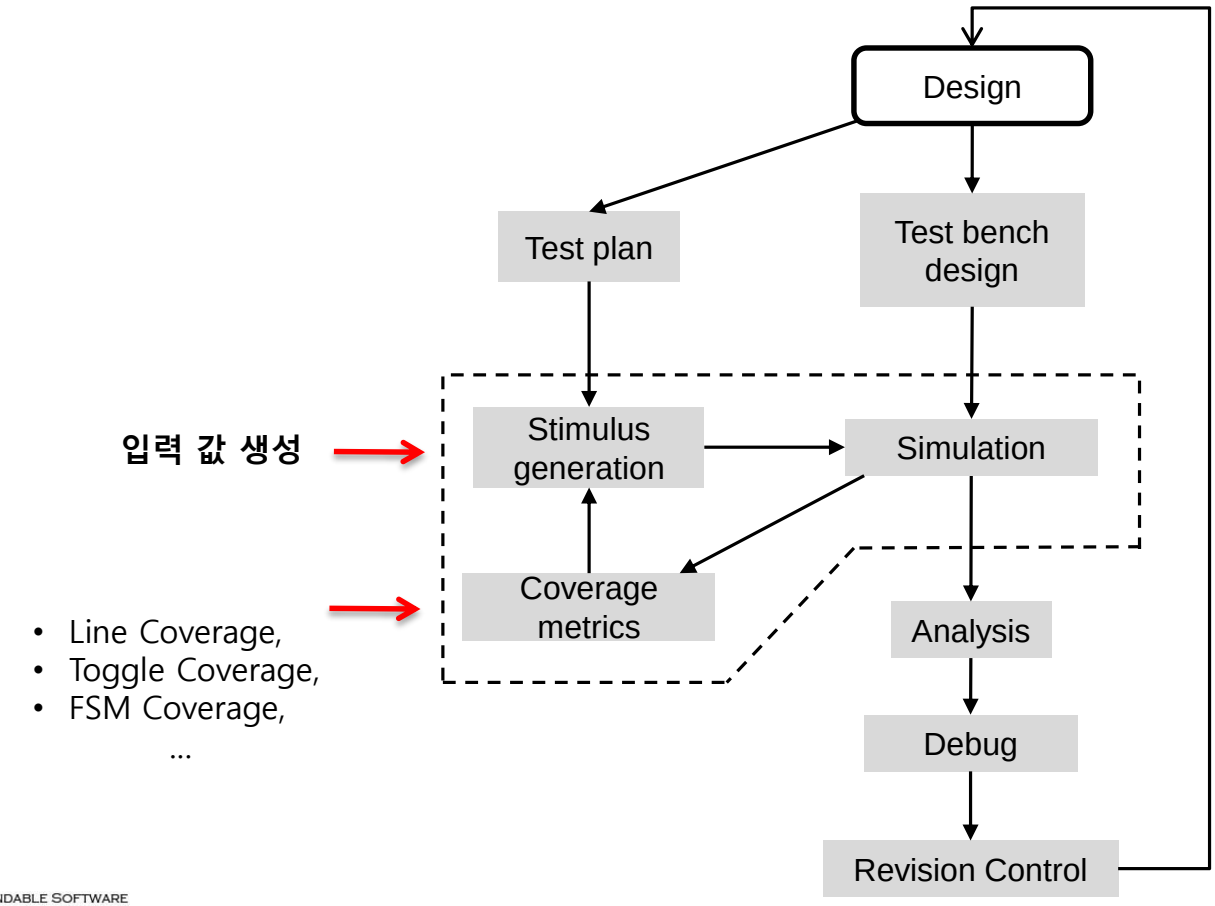
Software Aspect

- Software적인 측면의 Detail
- 설계를 검증하기 위해 시뮬레이션을 통한 검증을 사용



Flow of simulation based verification

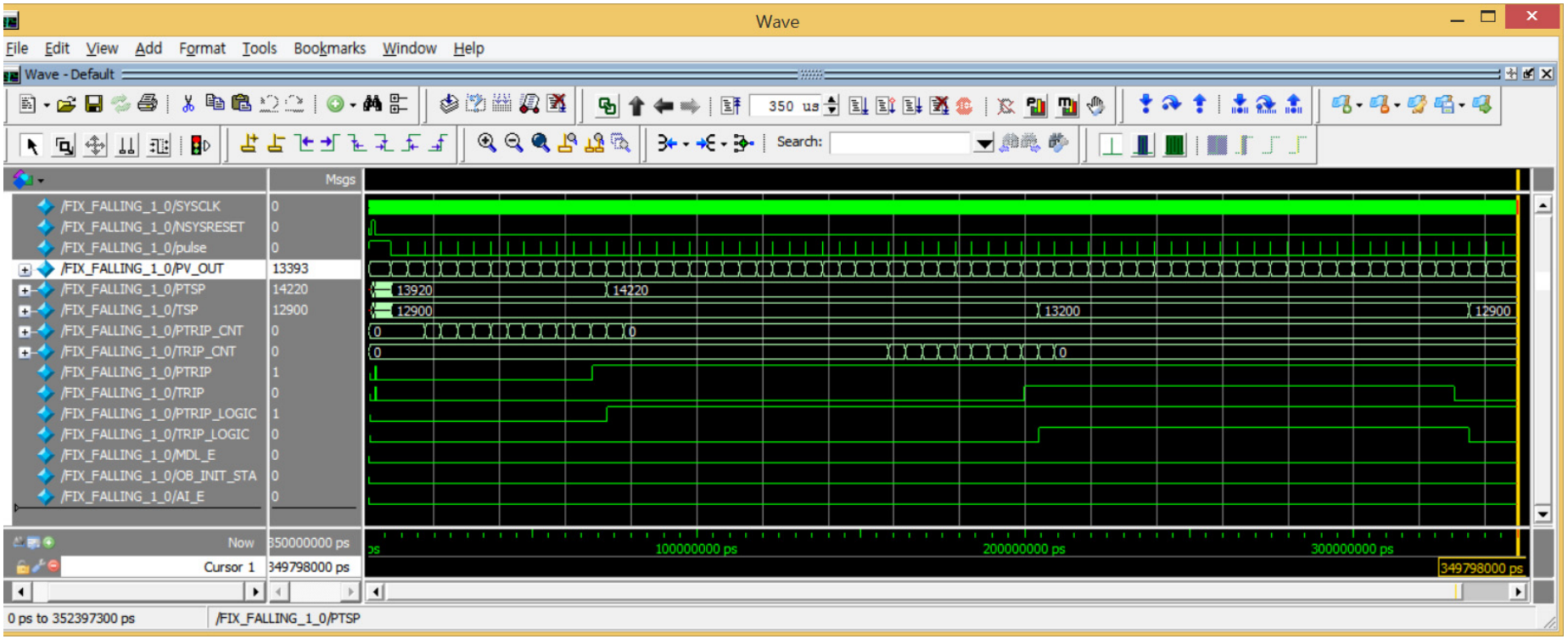
- 기본적인 시뮬레이션 기반 검증의 흐름
- 특정 커버리지가 만족될 때까지 점선 부분이 반복적으로 수행됨



Behavioral Simulation

- RTL Design을 이용해 수행하는 시뮬레이션
  - Verilog나 VHDL로 작성된 설계를 사용
- 설계의의 Syntax와 Functionality를 검증하기 위해 수행
- 시뮬레이션 기반 검증 중 가장 빠름
  - Design cycle에서 가장 조기에 가장 적은 cost로 error를 발견/수정 가능

Simulation using ModelSim



Logic Simulation

- Synthesis를 수행 한 후에 실시되는 Functional Simulation
  - Structural HDL을 사용하여 시뮬레이션 수행
- RTL design이 synthesis tool을 통해 맞게 합성되었는지 확인
- Behavioral Simulation 보다 느리지만 더 자세한 정보를 제공
  - Netlist는 Behavioral construct(HDL)가 아닌 Structural description(gate-level)이기 때문에 실제 HW와 가까운 simulation을 수행
- Netlist(EDIF)에는 Timing 정보가 포함되지 않기 때문에 Timing simulation이라고 보는 것은 불가

  ✓ Synthesis 과정에서 수행되는 Optimization 때문에 관찰하고자 하는 Signal이 Netlist에서 나타나지 않는 경우
    - Optimization을 막기 위한 attribute를 사용하여 해결

Testbench?

- Testbench 란…
  - 설계한 logic을 simulation할 때 simulation을 원활하게 하기 위해서 작성하는 별도의 code

- Logic Simulation에서 사용되는 Testbench를 Behavioral Simulation에서 사용된 Testbench 그대로 사용한다?
  - Synthesis Tool은 synthesis의 결과로 **EDIF** type과 **Structural HDL** type의 Netlist를 만듦
  - 입출력 포트에 대한 정보는 변하기 않기 때문에 사용하는 것이 가능

Testbench (Example)

Simulation의 대상이 되는 설계 포트 정보

Scenario

```
entity FIX_RISING_1_0 is
generic (
     INT_HI : integer := 8388607;
     INT_LO : integer := -8388608
);end FIX_RISING_1_0;

architecture behavioral of FIX_RISING_1_0 is

     constant SYSCLK_PERIOD : time := 100 ns;

     signal SYSCLK : std_logic := '0';
     signal NSYSRESET : std_logic := '0';
     signal pulse : std_logic := '0';
     signal PV_OUT : integer range INT_LO to INT_HI;
     signal MDL_E : std_logic;
     signal AI_E : std_logic;
     signal OB_INIT_STA : std_logic;

     component FIX_RISING
         port(
         clk : in std_logic;
         rst : in std_logic;
         pulse : in std_logic;
         PV_OUT : in integer range INT_LO to INT_HI;
         MDL_E : in std_logic;
         AI_E : in std_logic;
         OB_INIT_STA : in std_logic;
         TSP : buffer integer range INT_LO to INT_HI;
         PTSP : buffer integer range INT_LO to INT_HI;
         TRIP_LOGIC : buffer std_logic;
         PTRIP_LOGIC : buffer std_logic;
         TRIP_CNT : buffer integer range INT_LO to INT_HI;
         PTRIP_CNT : buffer integer range INT_LO to INT_HI;
         TRIP : out std_logic;
         PTRIP : out std_logic
         );
     end component;
```

```
begin
     process(SYSCLK)
         begin
             SYSCLK <= not(SYSCLK) after (SYSCLK_PERIOD / 2.0);
     end process;

     process
         begin
     --init
             PV_OUT <= 1000;
             MDL_E <= '0';
             AI_E <= '0';
             OB_INIT_STA <= '0';
             NSYSRESET <= '1';
             wait for (SYSCLK_PERIOD * 10);
             NSYSRESET <= '0';

             wait for (SYSCLK_PERIOD * 50); pulse <= '1';   PV_OUT <= 0;
                MDL_E <= '0';   AI_E <= '0';    OB_INIT_STA <= '1018';
                 wait for (SYSCLK_PERIOD / 2.0);    pulse <= '0';
             wait for (SYSCLK_PERIOD * 50); pulse <= '1';   PV_OUT <= 0;
                MDL_E <= '0';   AI_E <= '0';    OB_INIT_STA <= '1212';
                 wait for (SYSCLK_PERIOD / 2.0);    pulse <= '0';
             wait for (SYSCLK_PERIOD * 50); pulse <= '1';   PV_OUT <= 0;
                MDL_E <= '0';   AI_E <= '0';    OB_INIT_STA <= '1146';
                 wait for (SYSCLK_PERIOD / 2.0);    pulse <= '0';
```

Structural Verilog/VHDL

- Verilog와 VHDL 모두 Structural한 묘사 가능
- Verilog와 VHDL의 IEEE 표준에 Structural Description에 대한 항목이 포함되어 있음
- 오른쪽 예제의 경우 Behavior과 Structural design이 같은 동작을 수행
- 일반적인 Structural design은 변환하는 도구에 따라 묘사하는 방식이 다름

Behavior

```
module full_addr (A, B, Cin, S, Cout);
  input     A, B, Cin;
  output    S, Cout;

  assign {Cout, S} = A + B + Cin;
endmodule
```

Structural

```
module adder4 (A, B, Cin, S, Cout);
  input  [3:0] A, B;
  input        Cin;
  output [3:0] S;
  output       Cout;
  wire         C1, C2, C3;

  full_addr fa0 (A[0], B[0], Cin, S[0], C1);
  full_addr fa1 (A[1], B[1], C1,  S[1], C2);
  full_addr fa2 (A[2], B[2], C2,  S[2], C3);
  full_addr fa3 (A[3], B[3], C3,  S[3], Cout);
endmodule
```

Logic Simulation Test

- Synthesis tool 또는 Simulation tool에 따라 결과가 다르게 나오는지 확인하는 작업을 수행
  - Synthesis tool: Synplify Pro, Precision RTL, XST
  - Simulation tool: ModelSim, ISIM
- Verilog로 작성된 동일한 설계를 각각의 Synthesis tool을 사용해 합성하고, 각각의 Simulation tool을 사용해 시뮬레이션을 수행
  - Synthesis의 결과물로 Structural Verilog를 생성하게 하고 Code 비교
    - 자세한 코드는 다르지만, 유사한 구조를 보임
  - 3개의 Structural Verilog를 2개의 Simulation tool을 이용해 시뮬레이션
    - 확인된 부분: ModelSim에서 Synplify Pro와 Precision RTL을 대상으로 시뮬레이션을 수행하고 결과를 비교하였을 때 같은 출력을 보임을 확인
    - XST와 ISIM은 Xilinx ISE 환경에 포함된 도구로, 추후 실험을 수행할 예정
      - 과거의 수행 사례 다수 확인 (library를 추가하여 수행해야 함)
- Synthesis tool에 의한 동작은 변하지 않으며(제대로 수행된 경우), Simulation tool은 Synthesis tool에서 지원하는 도구에 한하여 시뮬레이션이 가능 (library, 협력관계 등으로 인한 제약사항이 발생)

Post-layout Simulation

- Place&Route 수행 후에 실시되는 Timing Simulation
  - Place&Route의 결과가 적용된 Structural HDL과
    Cell 내부 delay와 Interconnection delay의 정보가 포함된 파일을 사용

- Design에 Timing 정보가 추가될 때 맞게 동작하는지 확인
  - Delay가 삽입되기 때문에 동일한 출력을 생성하는지 확인 필요

- 가장 느리지만 HW와 가장 가까운 정보를 제공
  - Timing 정보(cell 내부, cell 사이의 delay)까지 고려한 simulation

Structural Verilog?

- Synthesis 후와 Place&Route 후의 Structural Verilog가 다른가?
  - 소스코드를 확인한 결과 → 다름
  - "P&R후의 Structural Verilog는 P&N의 결과가 적용되어있다" (Microsemi Libero SoC의 User manual 中)
  - 실제 코드에서 Synthesis 한 결과는 다수의 module로 나눠져 있는 반면 P&R 후의 결과물에는 하나의 module로 묘사됨
  - 추가적인 확인 필요

Simulation Tools

- Commercial non-free simulators (Big 3)

Simulator Name	Author/Company	Languages
ISE Simulator	Xilinx	VHDL-93, V2001
ModelSim and Questa	MentorGraphics	VHDL-1987 ~ -2008, V2001 SV2005, SV2009, SV2012
VCS	Synopsys	VHDL-2002, V2001, SV2005

* V=Verilog, SV=System Verilog

- Free and open-source simulator

Simulator Name	Author/Company	Languages
Icarus Verilog	Stephen Wiliams	V1995, V2001,
Verilator	Veripool	Synthesizable V2001, synthesizable V2005, synthesizable SV2009
LIFTING	A. Bosio, G. Di Natale (LIRMM)	V1995

SystemVerilog?

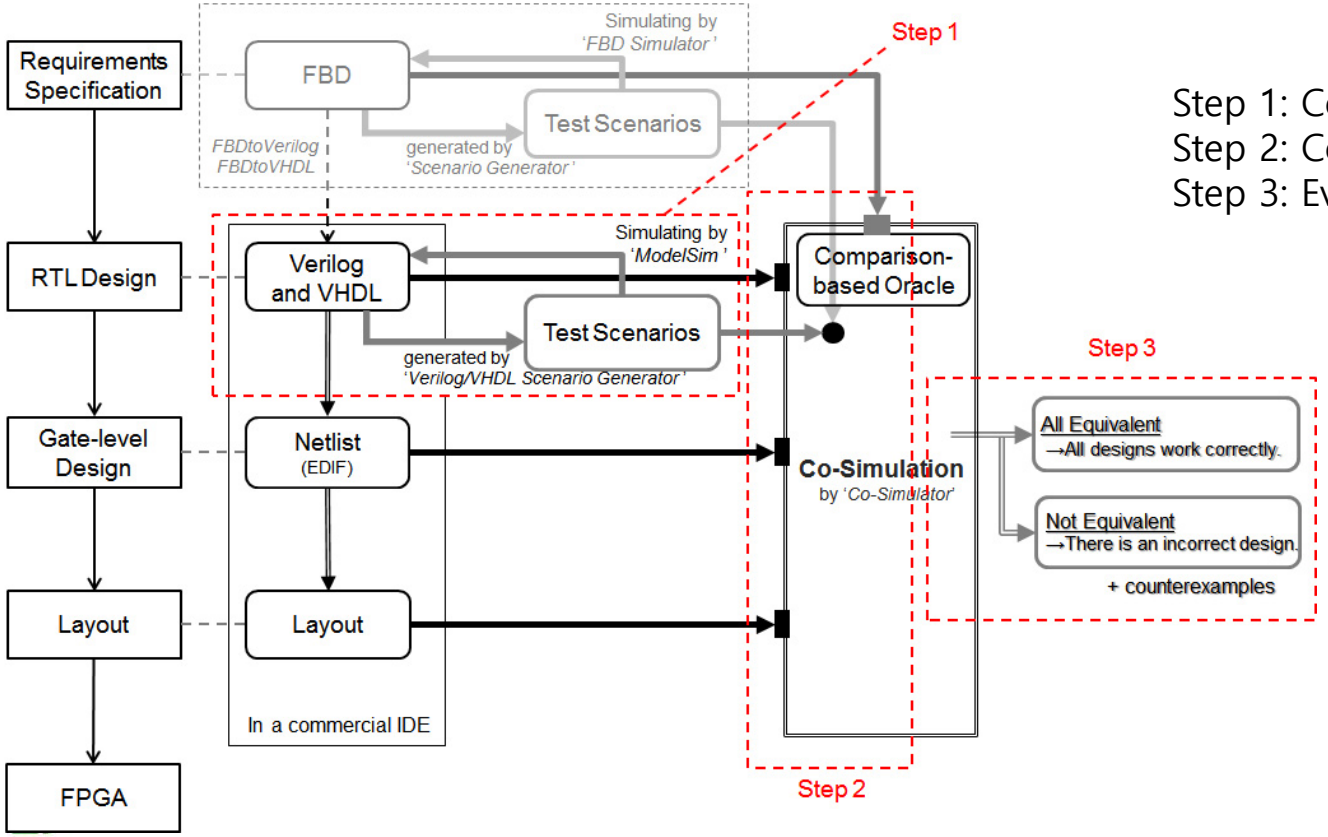
- IEEE 표준에 정의된 SystemVerilog는…
  Verilog-2005를 기반으로 만들어진 통합 언어로
  설계, 검증, 명세를 위한 언어로 사용된다.
  (“기존 Verilog + 검증용(혹은 모델링용) 확장 " 이라 불림)

- SystemVerilog는 Verilog-2005 + SuperLog +Vera C + C++ +VHDL + OVA + PSL… 등 다양한 언어를 더해서 만듦
- HDVL(Hardware Description & Verification Language)이라는 새로운 언어를 만들어냄
  - 하나의 언어로 설계와 검증을 모두 잡는 것이 목적인 언어를 창조

- 2002년 6월 등장하여 EDA tool에 적용되기 시작하여
  현재에는 SV를 지원하는 도구들이 다수 존재

Issue

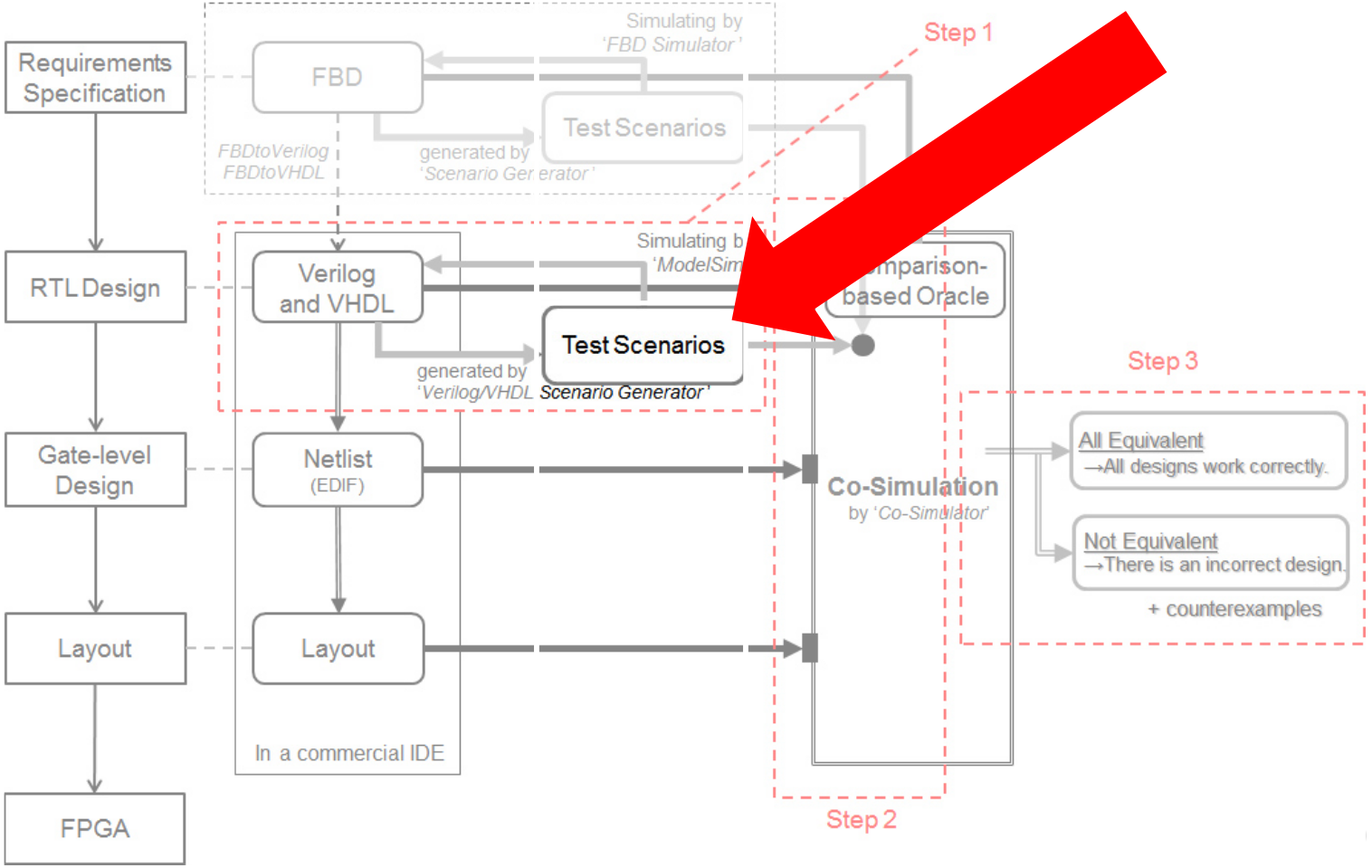
- 단계별 Simulation based Verification을 위해 소비되는 Cost가 높음
  - **IST-FPGA(Integrated Simulation based Testing Framework for FPGA)**
    - **3단계 절차를 통해 FPGA 설계의 Correctness를 통합적으로 확인하여 검증 비용과 검증 시간의 절감을 위해 제안한 통합 프레임워크**



Step 1: Comparison-based Oracle 생성
Step 2: Co-Simulation 수행
Step 3: Evaluation

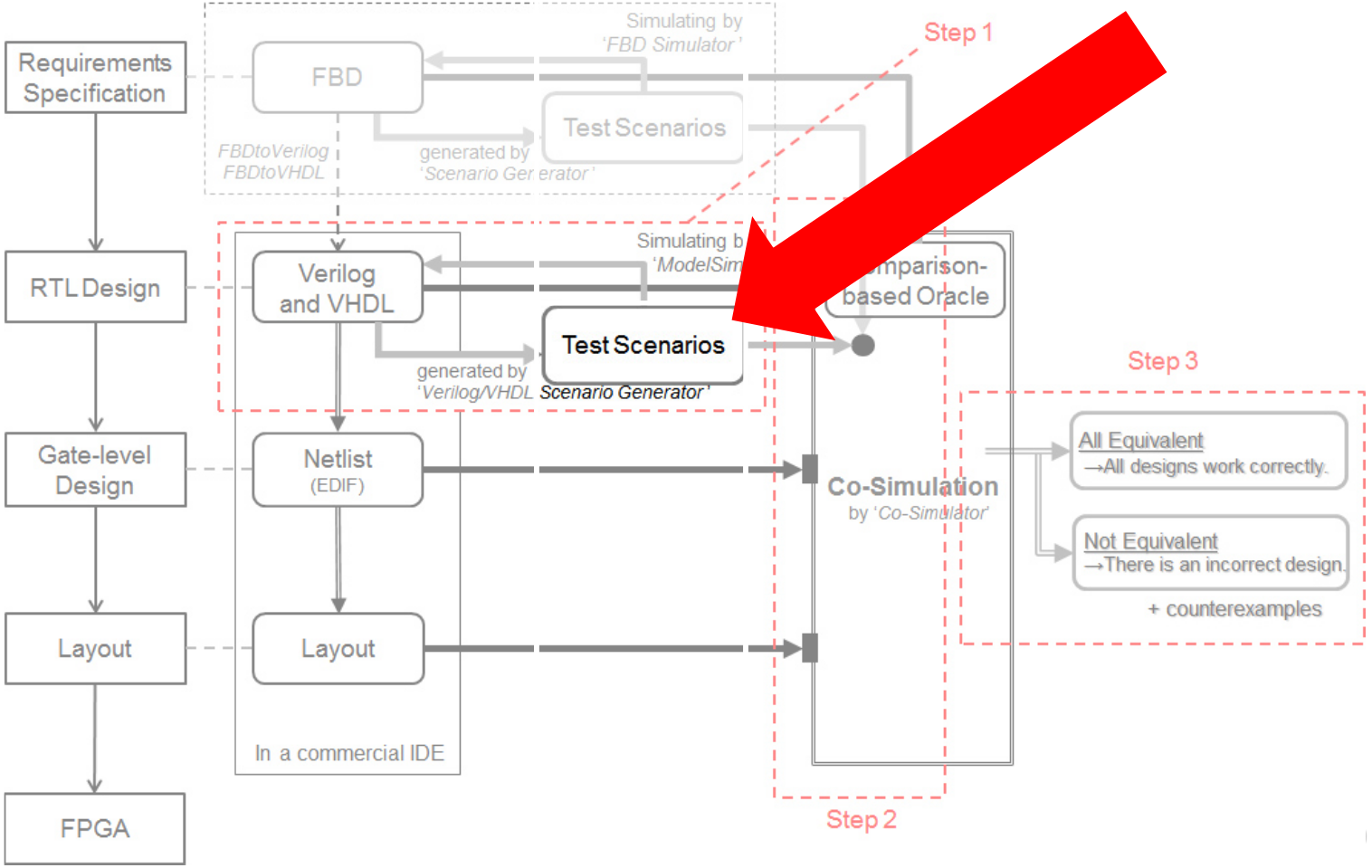
Issue (2)

- Testbench의 manual한 생성
  - **Verilog/VHDL Scenario Generator를 통한 Testbench 자동 생성**



Issue (3)

- Testbench(stimulus)의 Quality…?



Coverage

Coverage란?

- Coverage를 측정하여..
  - 빈틈 없는 Test suite을 만들기 위한 지표로 사용
  - Testing에서 더 신경써야 하는 부분을 알려주는 지표로 사용

- What can code coverage do?
  - 다음과 같은 물음에 대답하는 것이 가능
    1. "How much logic in the design is actually being exercised?"
       (설계의 얼마나 많은 부분이 실제로 실행되었는가?)
    2. "Does my test suite cover all of the design?"
       (Test suite이 모든 설계를 커버하였는가? (다수의 설계가 함께 수행 될 경우))
    3. "Am I done writing tests for the logic?"
       (로직을 위한 테스팅을 수행하였는가? → 제대로된 테스팅을 수행하였는가?)
    4. "Do I have any redundant diagnostics in my test suite?"
       (test suite에서 중복되는 부분은 없는가? → 효율적이고 제대로 된 테스팅을 수행?)

➡ Coverage를 통해 Scenario의 Quality를 높이는 것이 목적!

Coverage Tools

- Simulator에서 Coverage analysis를 지원

Simulator Name	Author/Company	Type
ModelSim and Questa	MentorGraphics	Commercial
MPSIM	Axiom → MentorGraphics	Commercial
VCS	Synopsys	Commercial
Active-HDL	Aldec	Commercial
Covered	Trevor Williams	Open source

많은 도구들이 대형 회사에 흡수되거나 사라짐
무료 도구 중에선 Covered가 가장 완성도 높고 공개되어 있음

Coverage Metrics

- Covered에서 제공되는 Coverage Metrics
  - Line Coverage
  - Toggle Coverage
  - Memory Coverage
  - Combinational Logic Coverage
  - Finite State Machine (FSM) Coverage
  - Assertion Coverage

(http://covered.sourceforge.net/user/chapter.metrics.html#section.metrics.line)

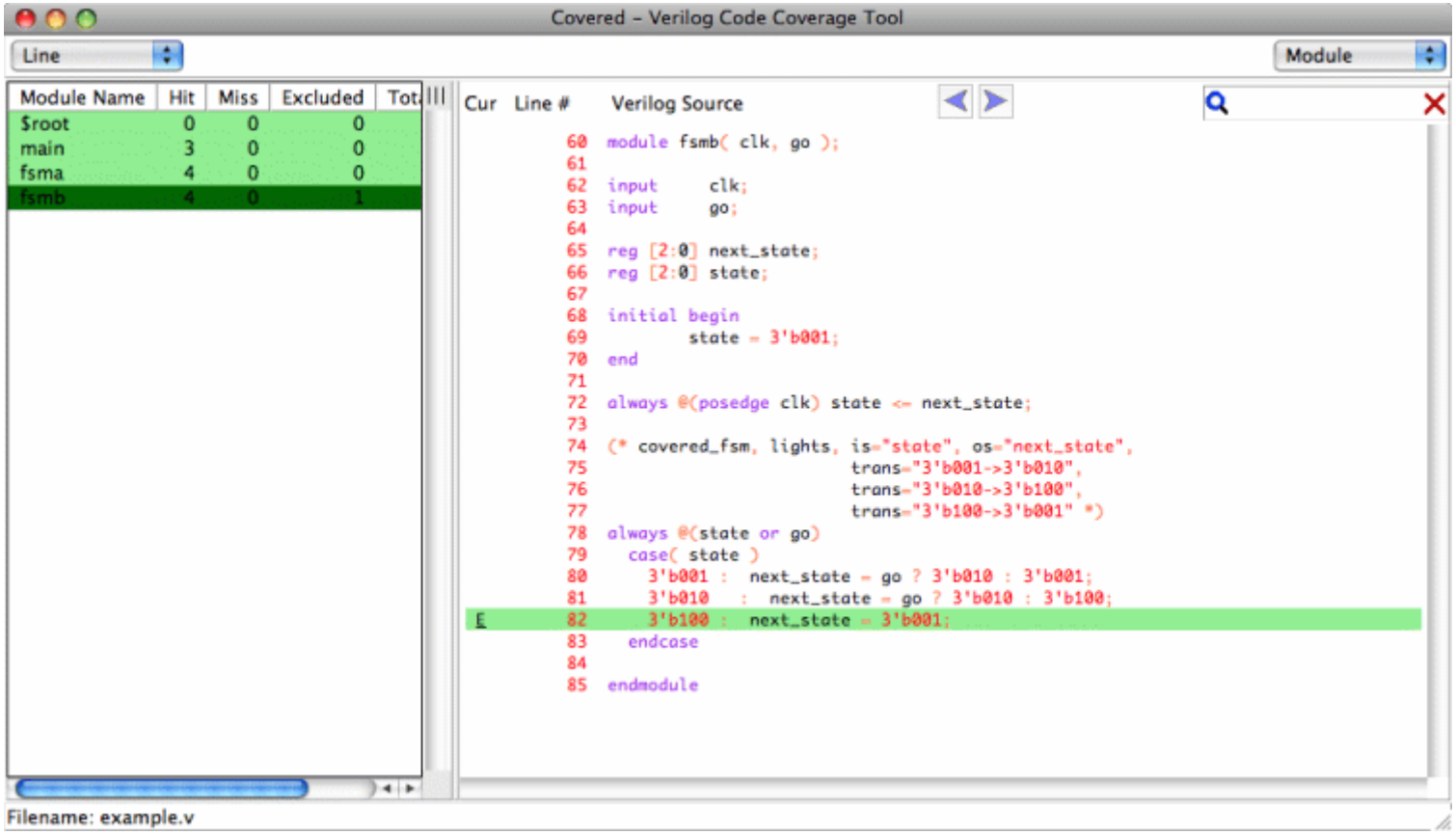
Line Coverage

- Design 내에서 얼마나 많은 line이 simulation 동안 수행되는가?

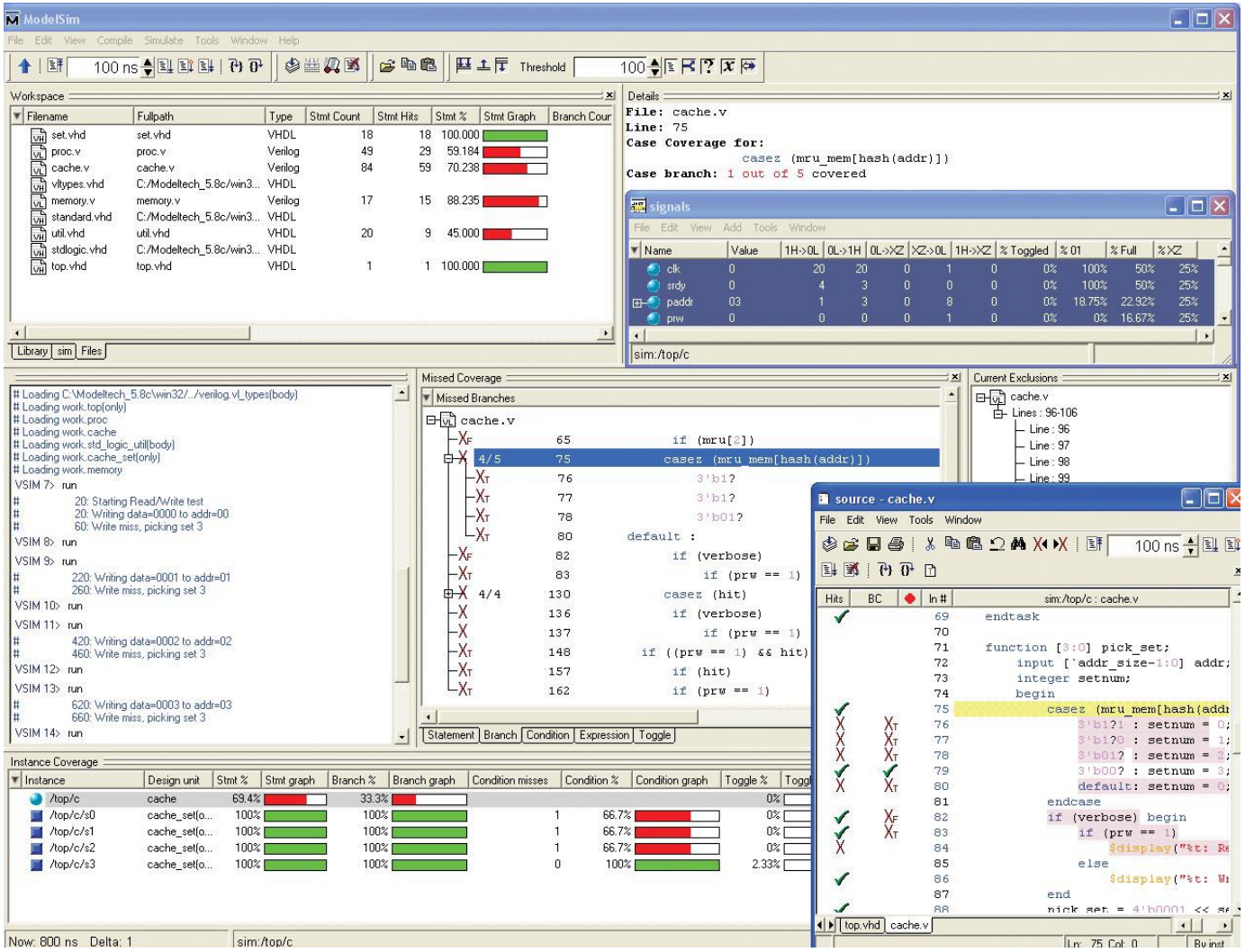
```
module test;

  reg a, b, c;

  initial begin
    a = 0;    // Line 1
    b = 1;    // Line 2
    if( a )
      c = b;  // Line 3
    else
      c = ~b; // Line 4
  end

endmodule
```

예제의 경우 line 1, 2가 수행되고
조건문에 따라 line 4가 수행되기 때문에
→ Line Coverage는 75%

Line Coverage Report in Coverd



Coverage Report in ModelSim



Toggle Coverage

- wire/register의 값이 simulation 동안 [0→1] & [1→0]으로 변화하는가?

```
module test;

  reg [2:0] a;

  initial begin
    a = 3'b0;
    #10;
    a = 3'b110;
    #10;
    a = 3'b010;
    #10;
  end

endmodule
```

예제의 경우 reg a의 값이 다음과 같이 변화

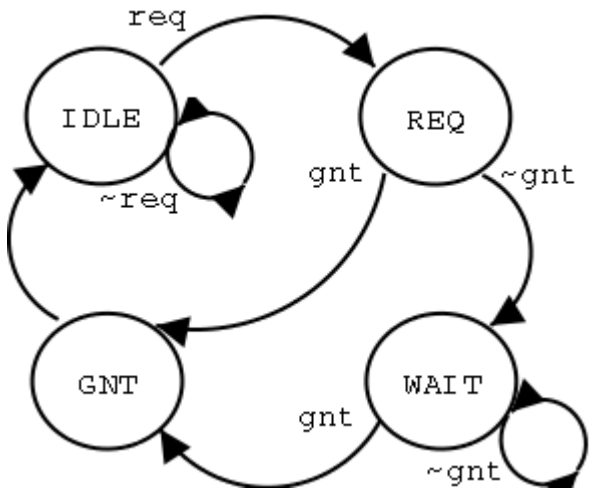
a = 000
a = 110
a = 010

Bit 0 : 변화X = 0%
Bit 1 : 0→1 변화, 1→0 변화X = 50%
Bit 2 : 0→1 변화, 1→0 변화 = 100%
→ Toggle Coverage는 50%

Finite State Machine Coverage

- FSM의 state와 transition을 얼마나 수행하는가?

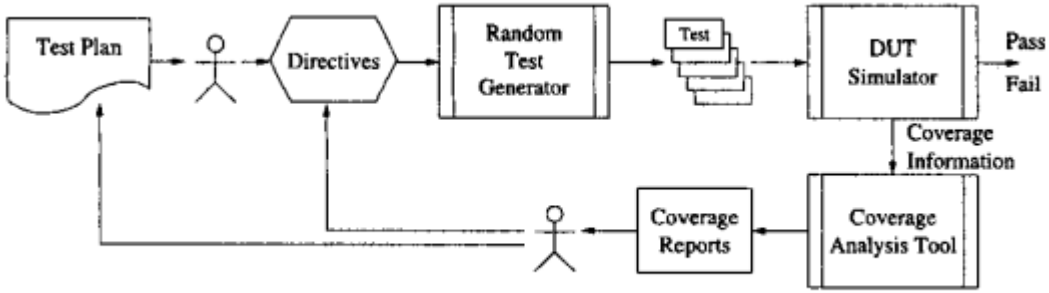
```
module test( clock );

  input clock;

  reg [1:0] state;
  reg req, gnt;

  parameter IDLE 2'b00;
  parameter REQ  2'b01;
  parameter WAIT 2'b10;
  parameter GNT  2'b11;

  initial begin
    req = 1'b0;
    gnt = 1'b0;
    repeat(4) @(posedge clock);
    req <= 1'b1;
    @(posedge clock);
    req <= 1'b0;
    gnt <= 1'b1;
    @(posedge clock);
    gnt <= 1'b0;
    repeat(2) @(posedge clock);
  end

  always @(posedge clock)
    case( state )
      IDLE : state <= req ? REQ : IDLE;
      REQ  : state <= gnt ? GNT : WAIT;
      WAIT : state <= gnt ? GNT : WAIT;
      GNT  : state <= IDLE;
      default : state <= 2'bx;
  endcase

endmodule
```



예제의 경우
→ FSM Coverage (State) 는 75%
→ FSM Coverage (Transition) 은 57%

Coverage를 이용한 사례

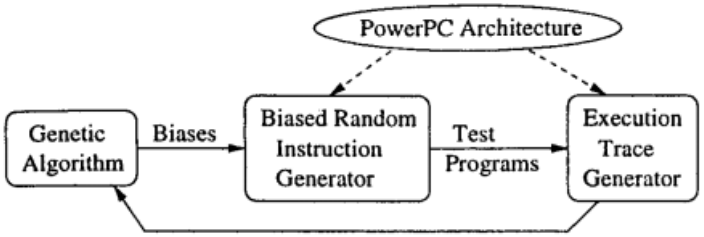
- Coverage Directed test Generation (CDG)



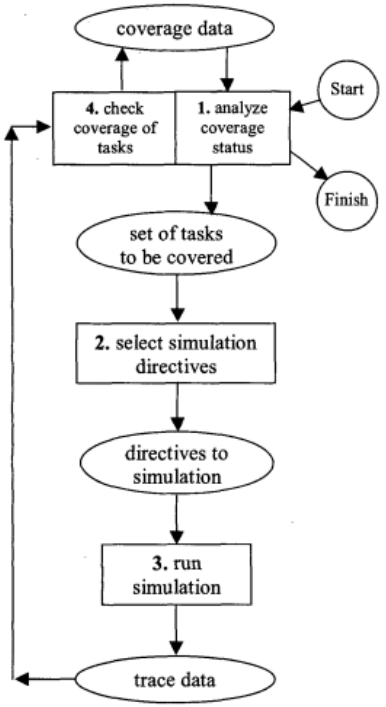
  - 크게 2가지 CDG가 존재
    - Feedback-based CDG
      - Coverage를 측정하고 이를 이용하여 알고리즘을 통해 시나리오를 개선
    - CDG by construction
      - 별도의 모델링을 수행하고 모델 각각에 알고리즘을 적용하여 시나리오를 생성

Feedback-based CDG 적용 사례

- **"A genetic approach to automatic bias generation for biased random instruction generation"**
  - Genetic algorithm이 test case를 선택/수정 하여 coverage를 증가시킴



- **"Cost evaluation of coverage directed test generation for the IBM mainframe"**
  - Coverage 분석 결과가 generation rule을 수행시켜 직접적으로 test를 변화시킴



CDG by construction 적용 사례

- **"Functional Coverage Driven Test Generation for Validation of Pipelined Processors"**
  - Pipeline Processor를 FSM으로 모델링하고 Functional Coverage를 사용하여 Test generation을 수행
  - Test generation은 정의한 fault model 별로 알고리즘을 만들어 수행

Algorithm 1: *Test Generation for Register Read/Write*
Input: Graph model of the architecture G.
Output: Test programs for detecting faults in register read/write.
begin /*** $TestProgramList = \{\}$ ***/
  for each register *reg* in architecture G
    $value_{reg}$ = GenerateUniqueValue(*reg*);
    writeInst = an instruction that writes $value_{reg}$ in register *reg*.
    $testprog_{reg}$ = createTestProgram(*writeInst*)
    $TestProgramList = TestProgramList \cup testprog_{reg}$;
  endfor
  return *TestProgramList*.
end

Table 2. Quality of the Proposed Functional Fault Model

Fault Models	Test Programs	HDL Code Coverage
Register Read/Write	130	85%
Operation Execution	182	91%
Execution Path	320	86%
Pipeline Execution	626	100%

Issue

- 현존하는 Coverage가 Safety Critical System을 위한 Scenario 생성에 적합한가?
  - Data flow coverage
  - 현재 사용되는 Coverage들은 structural coverage에 속함
  - FBD를 위해 만든 Coverage를 적용할 수 있는가?

- Verilog/VHDL Scenario Generator에 어떻게 CDG를 적용할 것인가?
  - 사례에 대한 추가적인 조사 필요

감사합니다