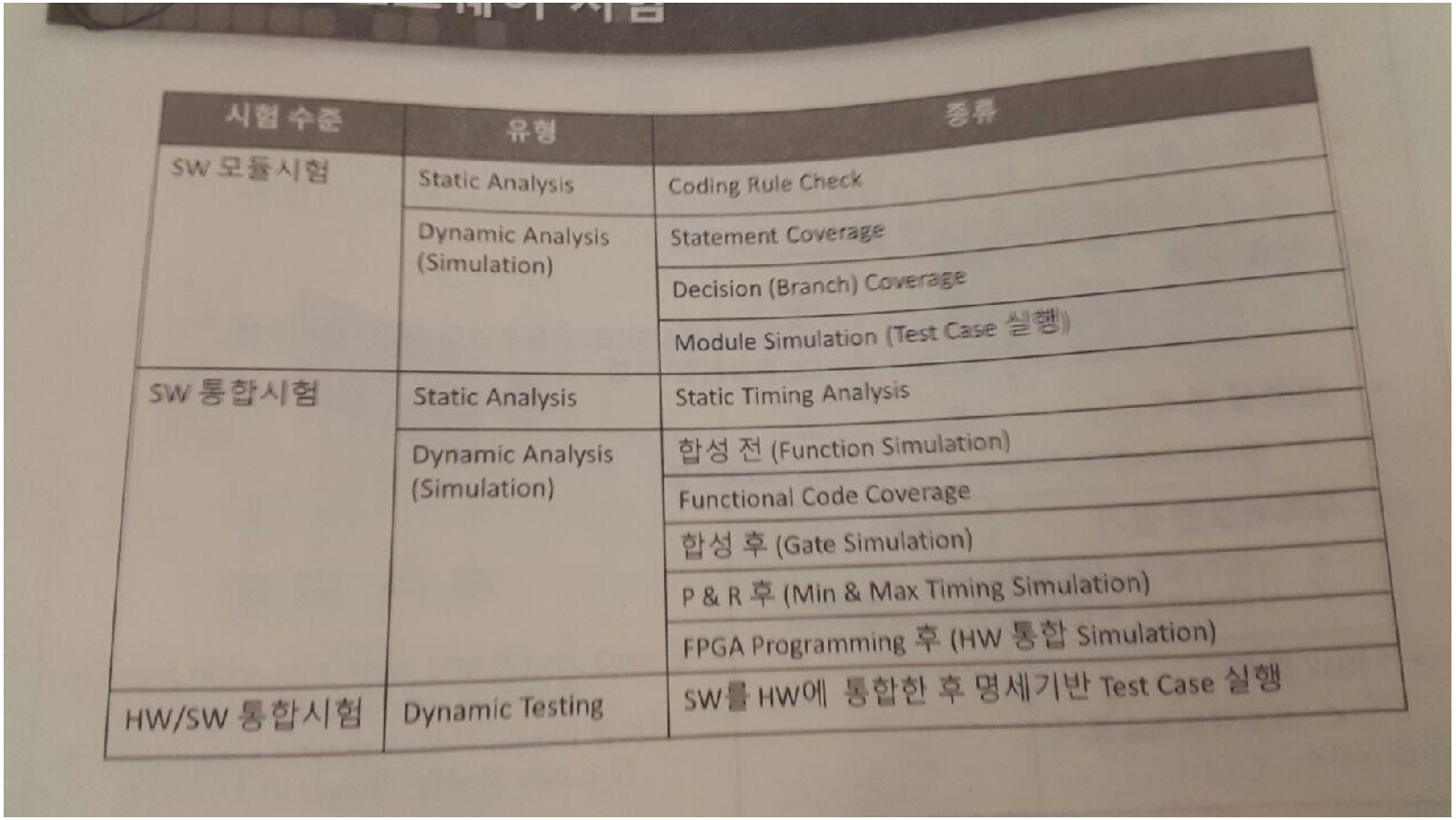
FPGA V&V

김의섭

목차

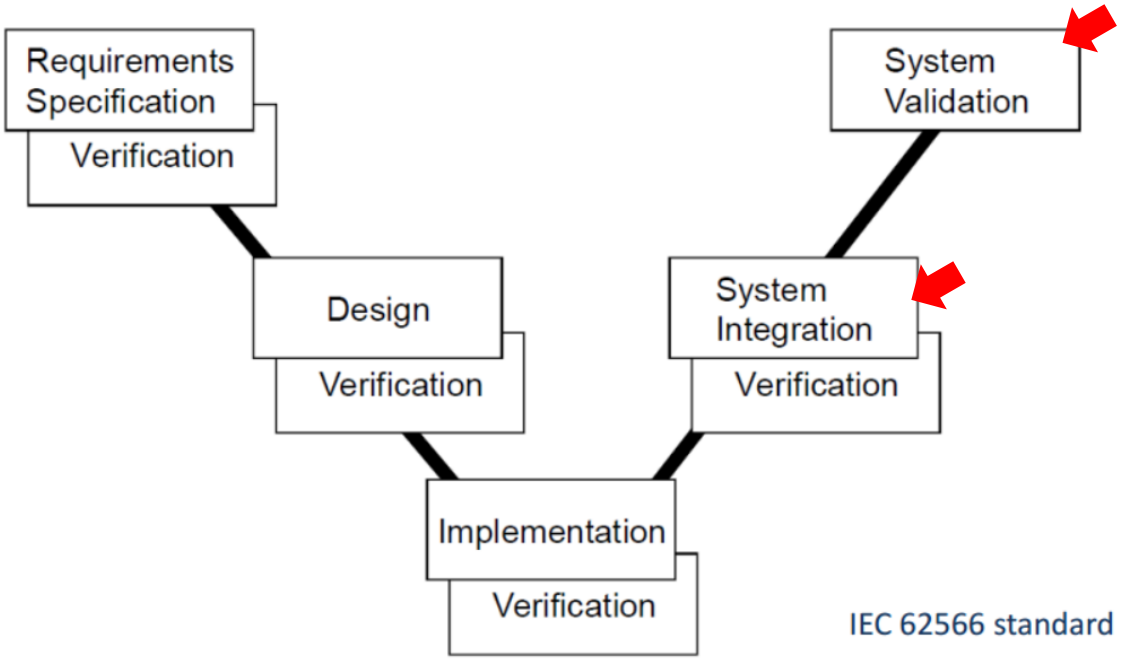
- Static Timing Analysis
  - 문제: logic 이 Stable 한 값을 출력하는지 확인 못한다.
  - 해결책: Stability Analysis

- Simulation Coverage



시험 수준	유형	종류
SW 모듈시험	Static Analysis	Coding Rule Check
	Dynamic Analysis (Simulation)	Statement Coverage
		Decision (Branch) Coverage
		Module Simulation (Test Case 실행)
SW 통합시험	Static Analysis	Static Timing Analysis
	Dynamic Analysis (Simulation)	합성 전 (Function Simulation)
		Functional Code Coverage
		합성 후 (Gate Simulation)
		P & R 후 (Min & Max Timing Simulation)
		FPGA Programming 후 (HW 통합 Simulation)
HW/SW 통합시험	Dynamic Testing	SW를 HW에 통합한 후 명세기반 Test Case 실행

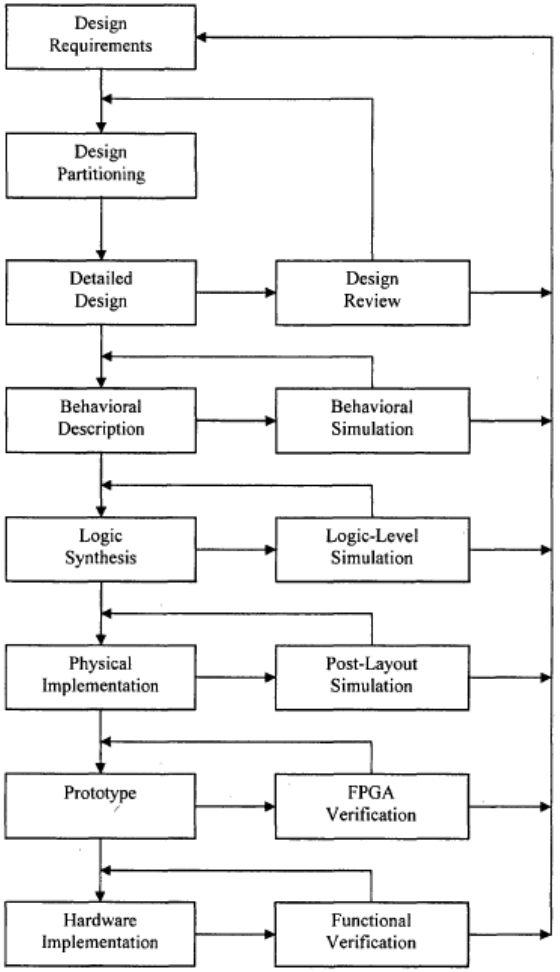
IEC 62566



IEC 62566 standard

- 1. Behavior Simulation
  - Code Coverage와 Functional Coverage 이용
- 2. Logic-level Simulation
  - (1)에서 사용한 test bench 사용 / (1)과 동일한 기능을 하는지 확인
- 3. Timing Simulation
  - (1)에서 사용한 test bench 사용 + Timing info. 포함
  - (1)과 동일한 기능을 하는지 확인
- 4. Static Timing Analysis
  - Glitch, Metastability, Worst case 고려 + user constraints 포함
- 5. FPGA Verification
  - (1)에서 사용한 test bench 사용 / 단일 FPGA를 대상으로 simulation
- 6. Functional Hardware Verification (Validation)
  - Board 위에 올려 기능 확인
  - Manually 수행

- +. On-site functional compliance with requirements
(Installation and Checkout phase)
  - IEEE Std 7-4.3.2-2010: IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations
  - System에 올려 기능 확인 다른 component와 동시에 기능 확인



NUREG/CR-7006

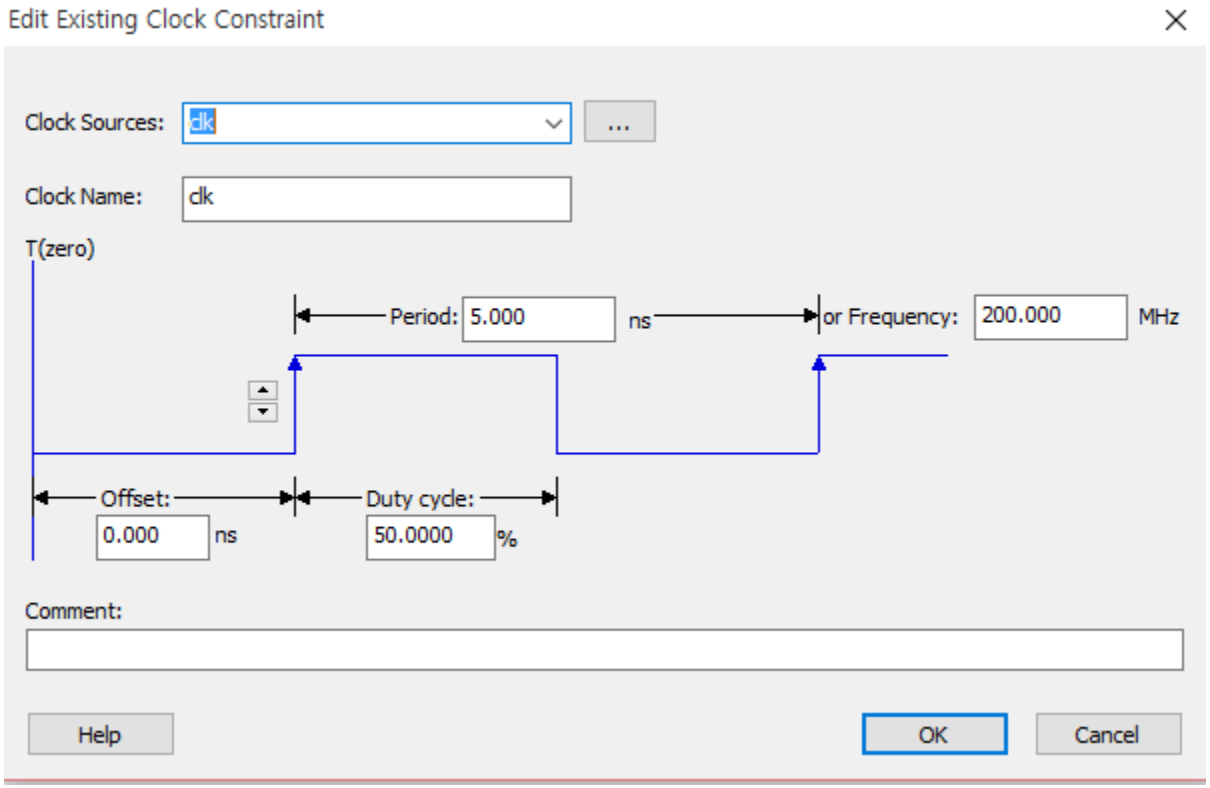
STA

Static Timing Analysis

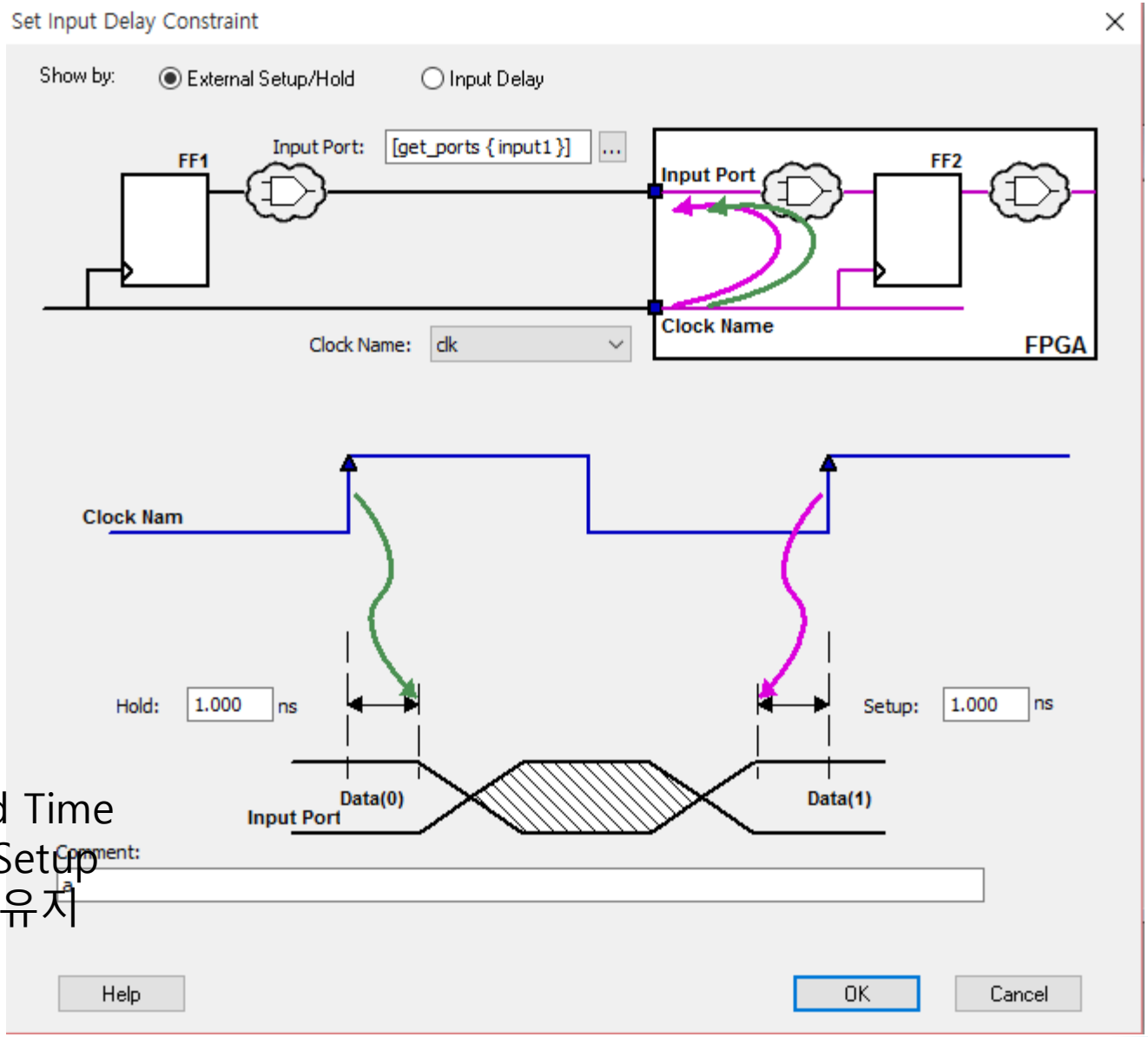
- Glitch (clk skew)
  - 글리치(Glitch)는 디지털 회로에서 발생할 수 있는 매우 짧은 기간동안 나타나고, 사라지는 전압이나, 전류의 원하지 않은 노이즈 펄스이다.
  - 이러한 노이즈 펄스의 폭은 소자의 물리적 특성과 사용 환경(온도, 습도, 진동 등)에 따라 변할 수 있다. 따라서, 글리치는 사용조건에 따라 어떤 때는 논리회로에 전혀 영향을 주지 않을 수도 있고, 어떤 때는 논리회로의 오동작을 발생시킬 수 있다.
- Metastability
  - Setup-time, hold-time
  - 클록의 상승에지 이전에 어느 정도의 시간동안 안정 - 셋업시간(tsu)
  - 클록 상승에지 이후에 어느 정도의 시간동안 안정 - 홀드시간(th)
- Worst case
  - Worst Case 경우에도 요구된 기능을 요구된 제한 시간 안에 수행할 수 있는 지를 확인
  - 요구된 기능: in/output 과 FF 사이의 combination logic

SmartTime (Libero)

- Clk 주기



SmartTime (Libero)

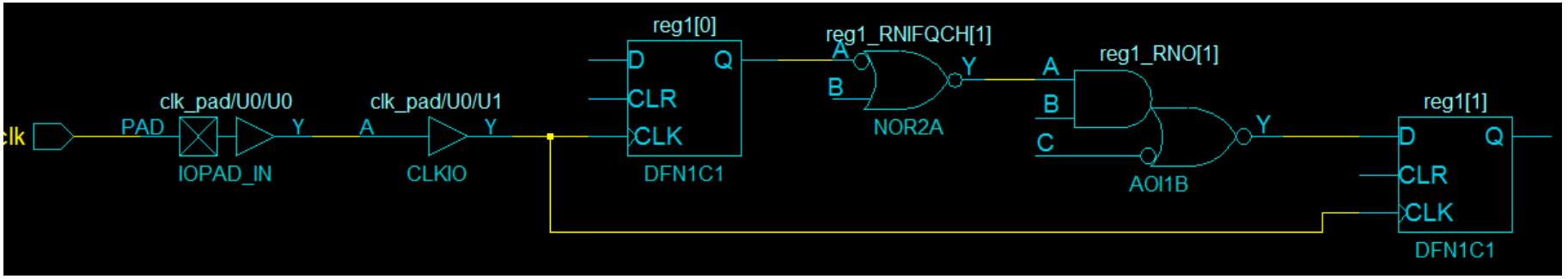


Setup & Hold Time
Data 도착 < Setup
Hold < Data 유지

	Source Pin	Sink Pin	Delay (ns)	Slack (ns)	Arrival (ns)	Required (ns)	Setup (ns)	Minimum Period (ns)	Skew (ns)
1	reg1[0]:CLK	reg1[1]:D	4.462	-0.001	7.127	7.126	0.539	5.001	0.000
2	reg1[0]:CLK	reg1[2]:D	4.462	-0.001	7.127	7.126	0.539	5.001	0.000
3	reg1[2]:CLK	reg1[1]:D	4.200	0.261	6.865	7.126	0.539	4.739	0.000
4	reg1[2]:CLK	reg1[2]:D	4.200	0.261	6.865	7.126	0.539	4.739	0.000
5	reg1[1]:CLK	reg1[2]:D	4.081	0.380	6.746	7.126	0.539	4.620	0.000
6	reg1[1]:CLK	reg1[1]:D	4.074	0.387	6.739	7.126	0.539	4.613	0.000
7	reg1[1]:CLK	reg1[3]:D	3.834	0.627	6.499	7.126	0.539	4.373	0.000
8	reg1[0]:CLK	reg1[3]:D	3.791	0.670	6.456	7.126	0.539	4.330	0.000
9	reg1[2]:CLK	reg1[3]:D	3.728	0.733	6.393	7.126	0.539	4.267	0.000
10	reg1[3]:CLK	reg1[2]:D	3.345	1.116	6.010	7.126	0.539	3.884	0.000
11	reg1[3]:CLK	reg1[1]:D	3.345	1.116	6.010	7.126	0.539	3.884	0.000
12	reg1[0]:CLK	reg1[0]:D	3.035	1.426	5.700	7.126	0.539	3.574	0.000
13	reg1[3]:CLK	reg1[3]:D	2.767	1.694	5.432	7.126	0.539	3.306	0.000

데이터가 clk보다 늦게 도착

2.127ns



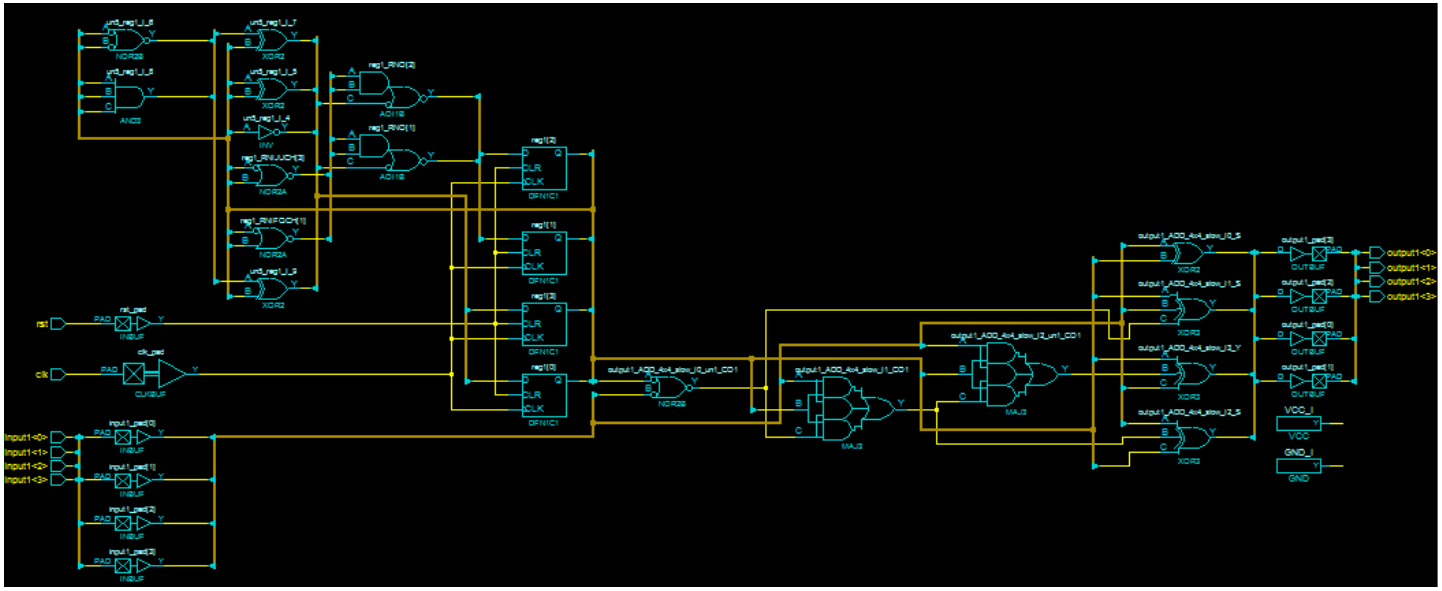
2.126ns

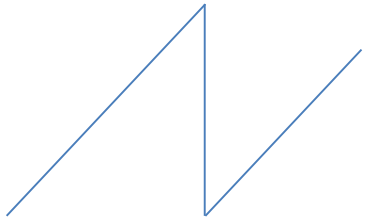
5ns+2.126ns

Stability

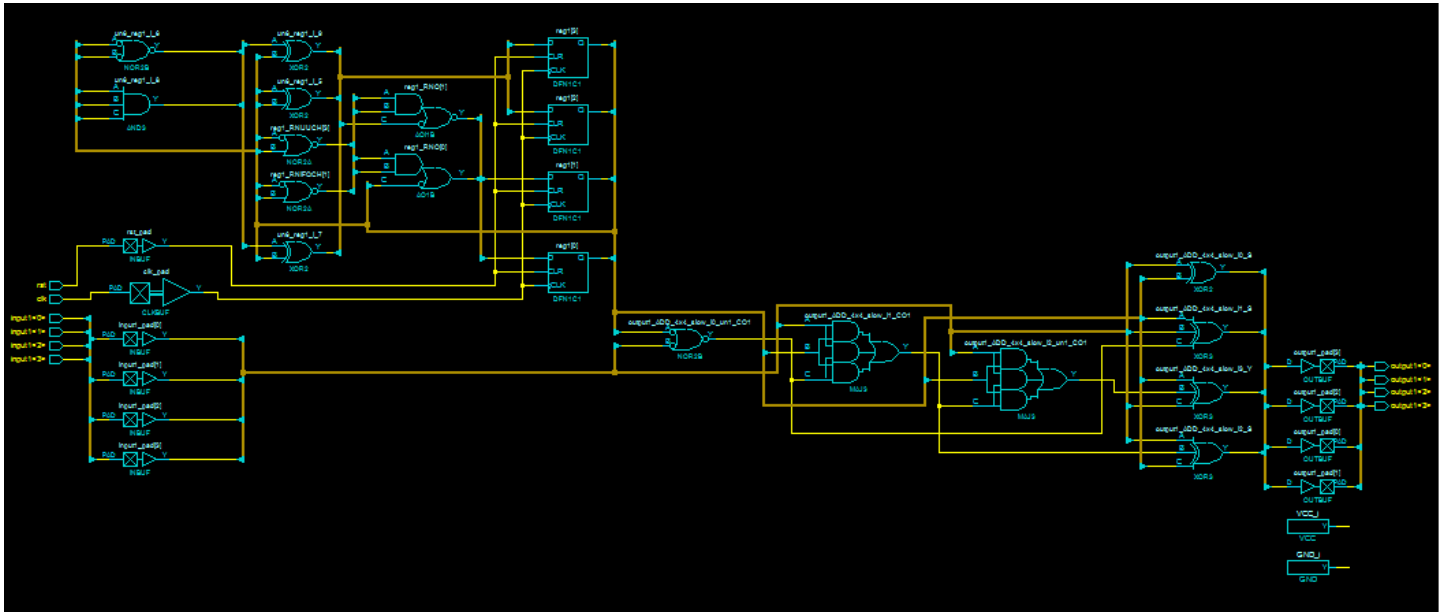
- 문제: 어느 주기 이후에 output 값이 stable 해질까?
  - FPGA 개발 시 일정 시간 후에 Output을 stable하게 유지하도록 개발할 수 있다. (아닐 수도 있다. → 기능적인 요소)
  - 하지만, 어떤 사용자는 input을 주었을 때 FPGA가 일정 주기 후 stable 해진 output 값 원하기도 함.
  - 따라서 이를 분석해줄 필요가 있음.

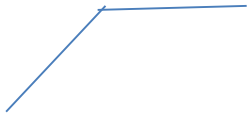
  - STA는 FPGA 개발 시 존재하는 violation을 찾아 주는 것 → 기능적인 오류 및 에러는 식별 X
  - STA로 stable 해지는 주기를 찾는 것은 불가능
  - 따라서 적절한 해결책이 필요





1 2 3 4 5 6 7 8

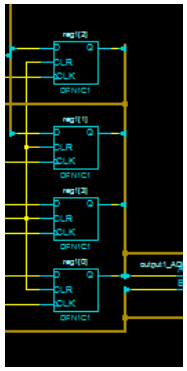




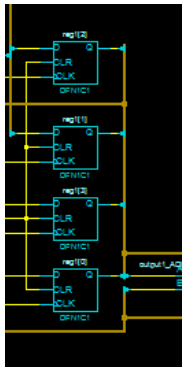
1 2 3 4 5 6 7 8

해결책: Stability Analysis

- (전제1) Input, output, reg 값이 이전 clk 값과 동일하다면 stable 해질 것임.
  - 변하는 signal이 존재하지 않다면, 앞으로도 모든 값은 계속 변하지 않을 것이다.
- (전제2)로직의 input은 stable 하게 주어진다.
  - 1. Stable해지길 원하는 개발자라면 input은 당연히 stable하게 줄 것이다.
  - 2. Reg 값이 stable해진다면 output도 당연히 stable하다.
  - 따라서, reg 값만의 비교로 분석 가능
- 언제 동일해 지는지 계산을 통해 파악 할 수 있을 것이다.
  - Simulation 을 통해 확인 가능 할 수도 있음 (하지만 performance)
  - Formal verification 개념 (모든 input에 대해 수학적으로 확인 가능)



t-1 = = = =



t

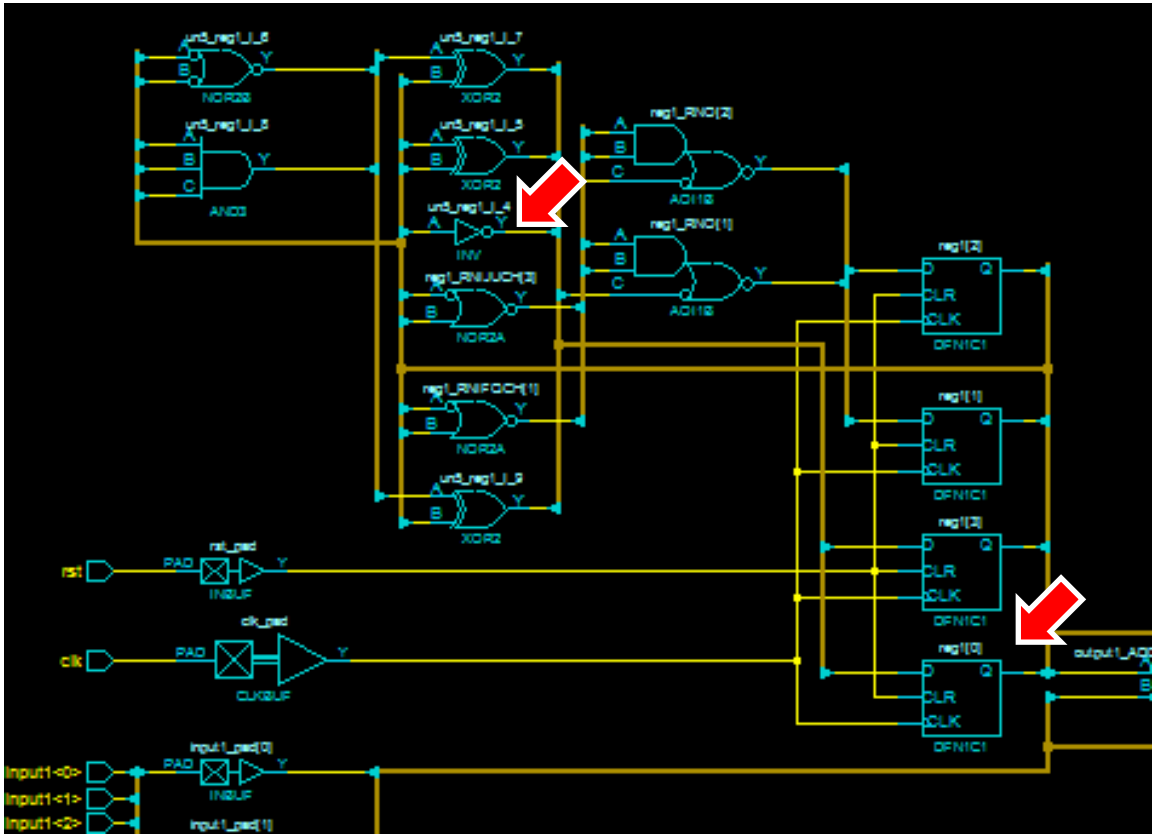
- 1. logic이 stable 해질 수 있는지 체크
  - Ex) input이 stable 하게 들어오는 logic인지 확인
  - Ex) 플리플롭의 값이 자체적으로 계속 변하게 되는 로직인지 확인

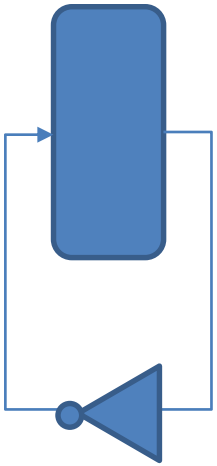
- 2. Stable 해질 가능성이 있다면, 언제 stable 해질지 계산
  - 모든 input 조합을 바탕으로 reg들의 t-1 값과 t값의 비교를 수행

    → 동일하면 stable

    → worst를 찾기 위해서는 모든 input 조합에 대해 수행해야 함

- 1. logic이 stable 해질 수 있는지 체크
  - Ex) input이 stable 하게 들어오는 logic인지 확인
  - Ex) 플리플롭의 값이 자체적으로 계속 변하게 되는 로직인지 확인
  - 다양한 경우를 생각

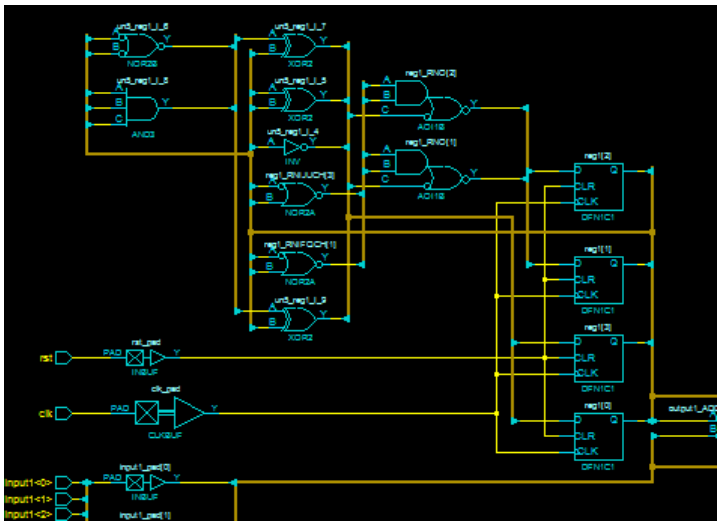




- 2. Stable 해질 가능성이 있다면, 언제 stable 해질지 계산
  - 모든 input 조합을 바탕으로 reg들의 t-1 값과 t값의 비교를 수행

    → 동일해지면 stable

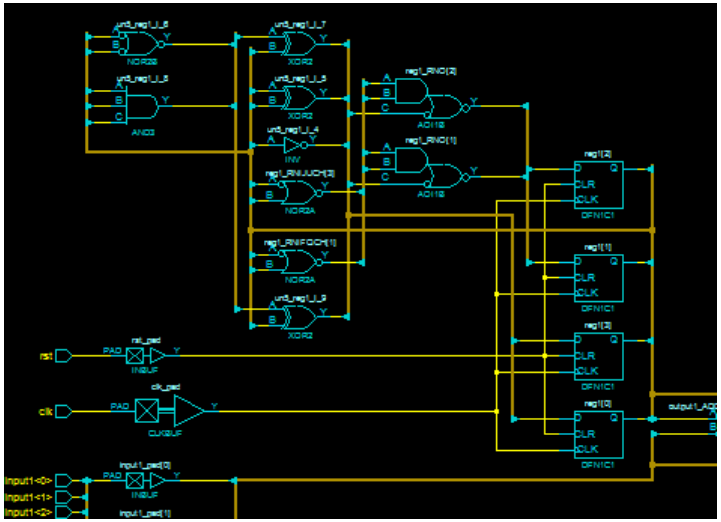
    → worst를 찾기 위해서는 모든 input 조합에 대해 수행해야 함
    - Reg의 값은 수식으로 표현 가능 (Boolean 연산)
      - Y(t+1) = x & y & ~z & t & input[0] & & input[1]
      - X(t+1) = ~x | y | z | t | & input[1]
      - Z(t+1) = x & ~y | z & t | & input[2]
      - T(t+1) = x & y | ~z | t

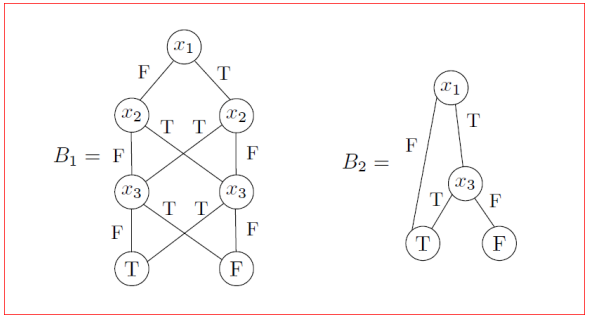


- 2. Stable 해질 가능성이 있다면, 언제 stable 해질지 계산
  - 모든 input 조합을 바탕으로 reg들의 t-1 값과 t값의 비교를 수행

    → 동일해지면 stable

    → worst를 찾기 위해서는 모든 input 조합에 대해 수행해야 함
    - Reg의 값은 수식으로 표현 가능 (Boolean 연산)
      - Y(t+1) = x & y & ~z & t & input[0] & & input[1]
      - X(t+1) = ~x | y | z | t | & input[1]
      - Z(t+1) = x & ~y | z & t | & input[2]
      - T(t+1) = x & y | ~z | t





BDD를 이용하여 계산 가능

트리만 만들어 놓으면 연산은 빠름
연산이 아니라 카노 맵 형태가 될 것임

이용할 만한 정보
+ reg는 0부터 시작
+ input은 고정 (0 or 1) → 재사용

- Stability Analysis
  - FPGA가 일정 주기 후 stable 해진 output 을 출력하길 원하는 사용자에게 언제 stable 해질지
  - Worst case의 t주기를 알려 주게 될 것임
  - 만약 수행한다면
    - logic이 stable 해질 수 있는지 체크 방법
    - BDD 연산을 이용하는 방법

COVERAGE

Classification

- Code Coverage
  - Statement Coverage
  - Block Coverage
  - Decision Coverage
  - Path Coverage
  - Expression Coverage
  - Event Coverage
- FSM Coverage
  - Conventional FSM Coverage
  - SFSM Coverage
- Other Coverage Metrics
  - Observability-Based Code Coverage
  - Toggle Coverage
  - Variable Coverage

Paper: Coverage Analysis Techniques for HDL Design Validation

- Syntactic Coverage Metrics
  - Code Coverage
    - Statement coverage
    - Branch coverage
  - Circuit Coverage
    - Latch coverage
    - Toggle coverage
- Semantic Coverage Metrics
  - FSM Coverage
    - Limited-path coverage
    - Transition coverage
  - Assertion Coverage (functional coverage)
  - Mutation Coverage

Paper: Coverage Metrics for Formal Verification

Others

- Post-silicon code coverage
- Observability-Based Code Coverage
- Observability-Enhanced Statement Coverage
- Data Flow Fault Coverage
- Dumpfile-Based Coverage
- Validation Vector Grade (VVG): A New Coverage Metric for Validation and Test

- 몇몇 문서를 확인
  - **EPRI TR-1019181**
    - Guidelines on the Use of Field Programmable Gate Arrays in **Nuclear** Power Plant I&C Systems
  - **EPRI TR-109390**
    - Design Description of a Prototype Implementation of Three **Reactor Protection System** Channel Using Field-Programmable Gate Arrays
  - **EPRI TR-1022983**
    - Recommended Approaches and Design Criteria for Application of Field Programmable Gate Arrays in **Nuclear** Power Plant Instrumentation and Control Systems
  - **NUREG/CR-7006**
    - Review Guidelines for Field-Programmable Gate Arrays in **Nuclear** Power Plant Safety Systems
  - Coverage에 관해서는 언급이 거의…
  - DO-254
    - Design Assurance Guidance For **Airborne** Electronic Hardware
    - **Code Coverage는 100% 만족을 요구?**
  - ~~IEC 62566~~
    - ~~**Nuclear** power plants - Instrumentation and control important to safety - Development of HDL-programmed integrated circuits for systems performing category A functions~~

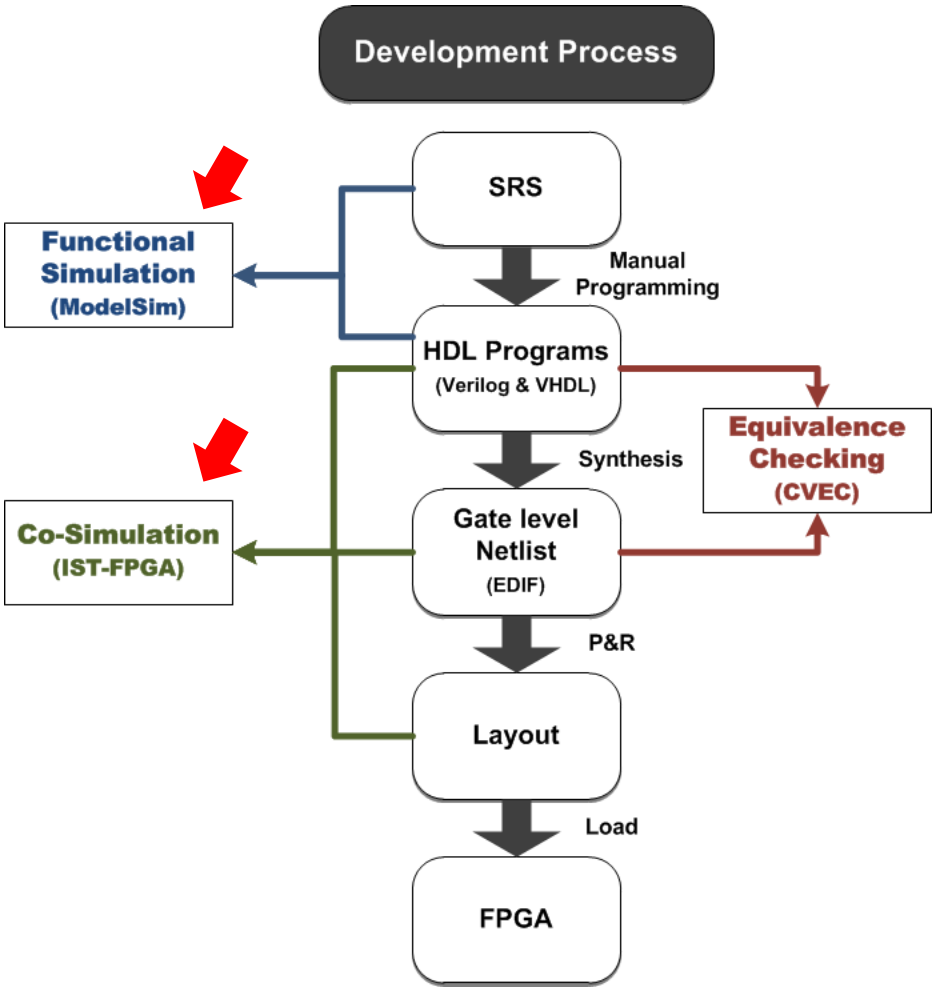
- 개인적으로 내린 결론

- Q. 산업에서는 Code Coverage만 사용? (+Functional Coverage)
  - (아마도?...)
    - Do-254 – code coverage 100%
    - Commercial Tools (ISE Simulator, Incisive Enterprise Simulator, ModelSim 등) – code coverage 제공
  - 공인된 coverage는 code coverage뿐?
  - 하지만 100% coverage를 만족하는 test case 생성 방법은?
    - 특정 coverage를 100% 만족하는 Testbench를 생성하는 방법..

- Q. 다른 Coverage의 존재?
  - 기타 연구실에서 주장하는 다양한 coverage (비주류)
  - Fault Coverage (가장 많이 언급되는 Coverage)
    - ASIC 개발 공정 중 사용되던 coverage
      - ASIC 반도체 설계 및 제조 공정 중 생기는 물리적인 결함을 검사하기 위한 coverage
    - ATPG(Automatic Test Patten Generation) 에 사용

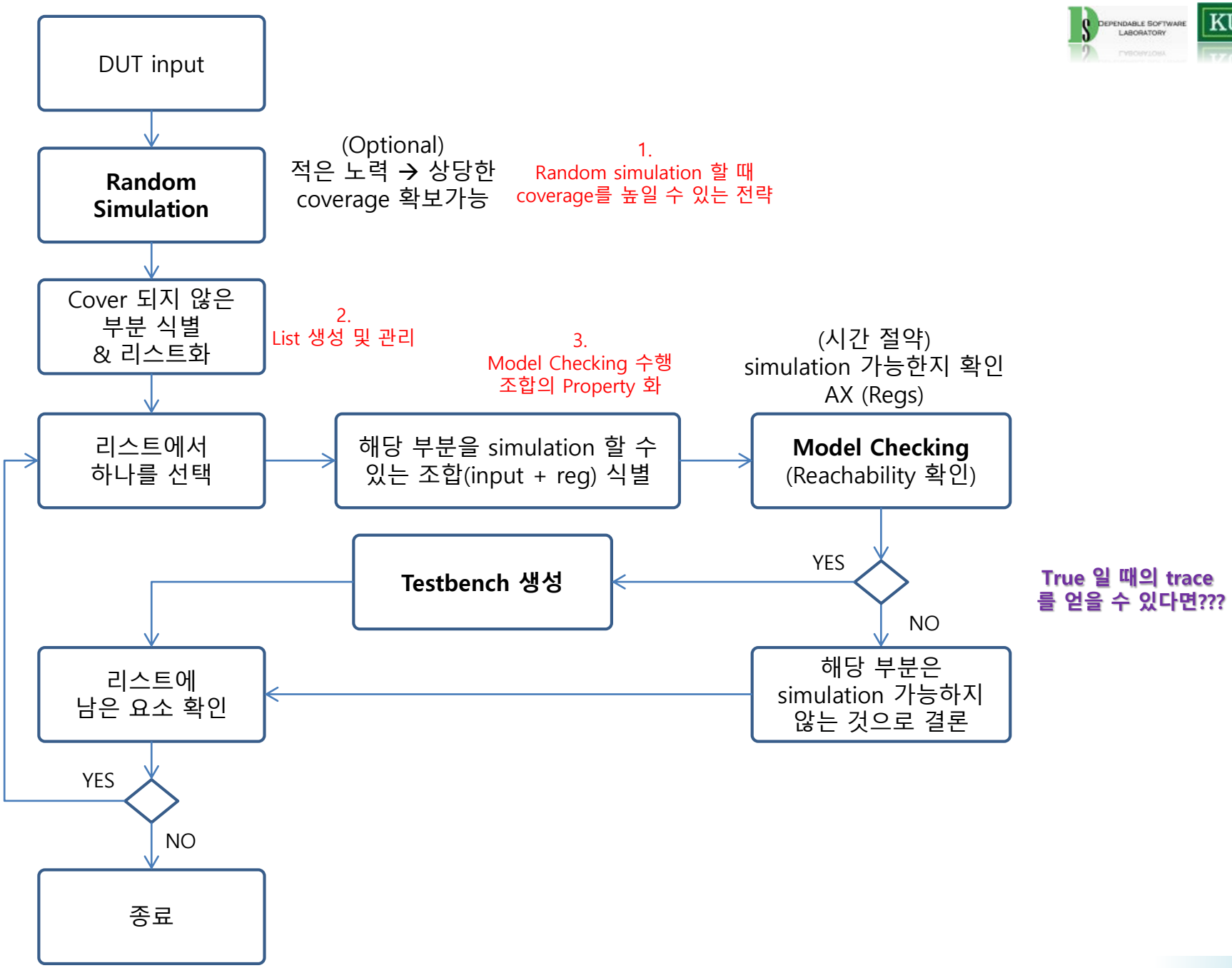
문제점 및 해결책 1

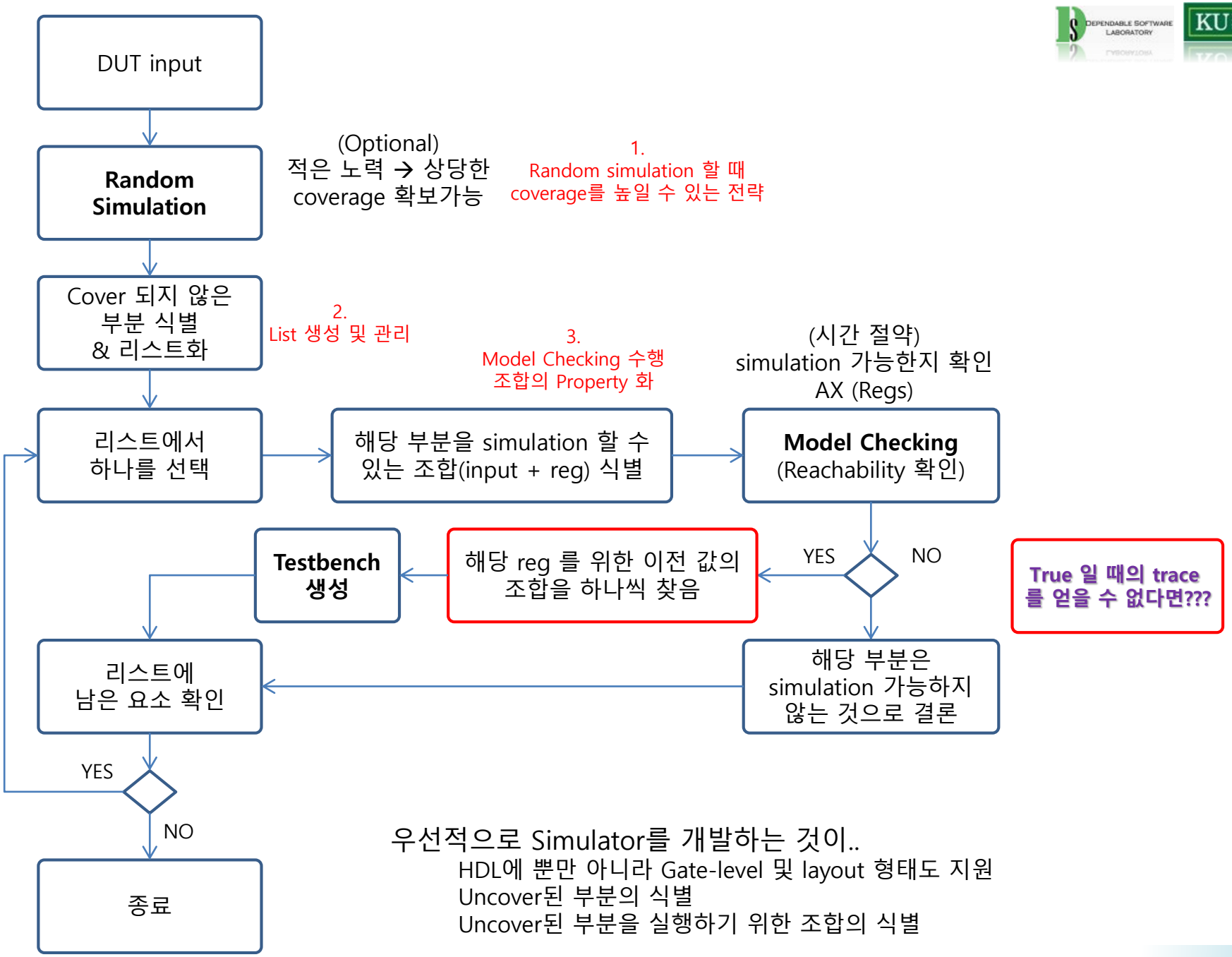
- 일반적으로 HDL 단계에서는 Code Coverage를 많이 씀
  - Statement Coverage / Branch Coverage / Toggle Coverage …
- **문제**
  - Code Coverage 를 측정하는 방법은 많이 있지만,
  - Code Coverage 를 충분히(100%) 만족하는 Testbench를 생성하는 기술은 미비 (?)
- **해결**
  - 따라서, 특정 coverage를 충분히 만족하는 Testbench 생성 방법 및 도구 구현 연구
- Contribution
  - 100% 만족 또는 그에 상응하는 testbhench 생성이 의미가 있을 것이다.



전략

- 100% Code coverage 를 만족 하는 testbench 생성 연구
  - 100% 가 어려운 이유
    - Simulation의 경우 reg가 연속적 이여서 특정 부분을 simulation 하기 위해서는 그 상황까지의 연속적인 reg의 조합을 찾아야 함 (조합의 조합을 찾아야함)
    - 반면, Testing의 경우 해당 부분을 수행하기 위한 input 및 reg의 조합 필요
  - Random simulation + Model checking 를 이용할 계획
    - Random simulation으로 일정부분 coverage를 확보하고,
    - 특정부분에 대해서는 model checking을 활용


DUT input
Random Simulation
(Optional)
적은 노력 → 상당한 coverage 확보가능
1.
Random simulation 할 때 coverage를 높일 수 있는 전략
Cover 되지 않은 부분 식별 & 리스트화
2.
List 생성 및 관리
3.
Model Checking 수행
조합의 Property 화
(시간 절약)
simulation 가능한지 확인
AX (Regs)
리스트에서 하나를 선택
해당 부분을 simulation 할 수 있는 조합(input + reg) 식별
Model Checking
(Reachability 확인)
YES
Testbench 생성
True 일 때의 trace 를 얻을 수 있다면???
NO
리스트에 남은 요소 확인
해당 부분은 simulation 가능하지 않는 것으로 결론
YES
NO
종료

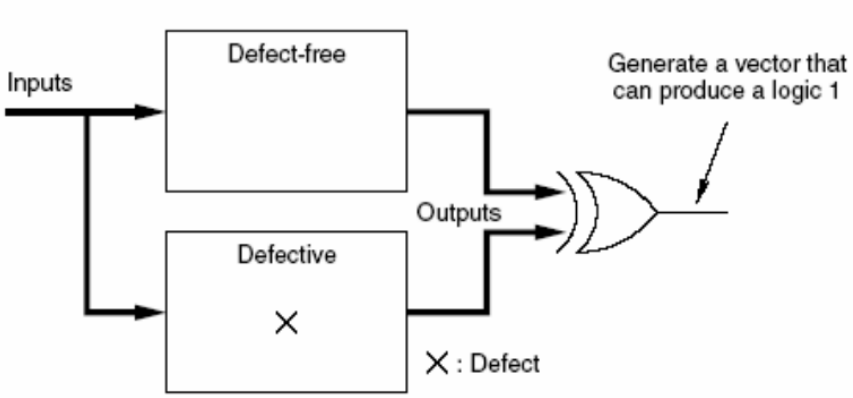


우선적으로 Simulator를 개발하는 것이..

- HDL에 뿐만 아니라 Gate-level 및 layout 형태도 지원
- Uncover된 부분의 식별
- Uncover된 부분을 실행하기 위한 조합의 식별

Fault Coverage

- Q. 다른 Coverage의 존재?
  - 기타 연구실에서 주장하는 다양한 coverage (비주류)
  - Fault Coverage (가장 많이 언급되는 Coverage)
    - ASIC 개발 공정 중 사용되던 coverage
      - ASIC 반도체 설계 및 제조 공정 중 생기는 물리적인 결함을 검사하기 위한 coverage
    - ATPG(Automatic Test Patten Generation) 에 사용
- Fault Coverage
  - 주어진 회로에 Fault 를 삽입하여 test case가 이를 수행하였는지를 확인
  - 오리지날 회로와 fault를 삽입한 회로에 동일한 test caser를 사용,
    결과가 다르다면 fault를 수행하였다고(찾았다고) 판별



Concept view of ATPG

ATPG

- ATPG (Automatic Test pattern Generation)
  - ASIC 반도체 설계와 제조 공정 과정에서 물리적인 결함 발생을 가정
  - 효과적인 검사가 이루어지지 않을 경우 출하된 chip이 불량이 확률이 높아짐
  - **효과적인 검사를 위해서는 chip의 결함을 찾을 수 있는 양질의 test pattern 이 필요**

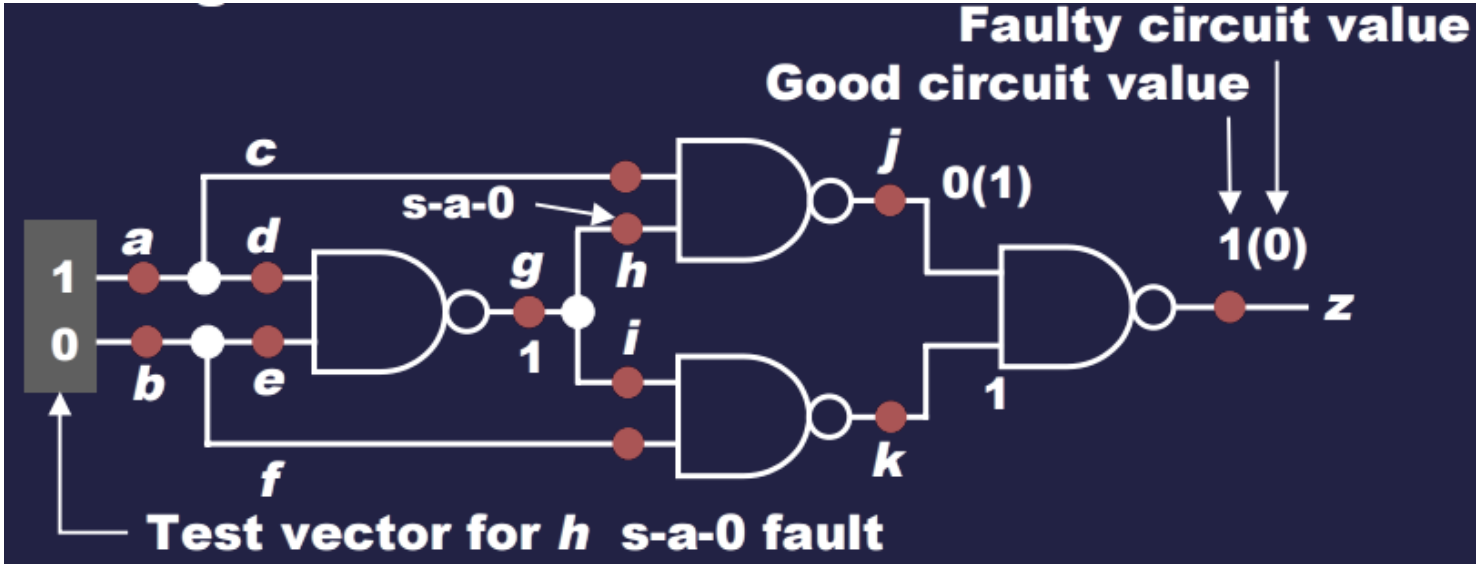
- 주어진 회로에 대하여 Fault model을 가정하여 Fault 을 검출하는 입력 패턴 생성을 자동화 시키는 과정.
  - Original design 과 fault model 을 생성하여 동일한 입력을 주어 결과값이 동일하지 않으면 fault 를 찾은 것
  - 이때 다른 결과를 내는 입력이 Test pattern

- Fault model
  - The Stuck-at fault model
  - Transistor faults
  - Bridging faults
  - Opens faults
  - Delay faults
  - Fault collapsing

고착 고장모델 (stuck-at fault model)

: 설계 오류 또는 제조 결함으로 인해 신호 선이 Vss 또는 Vdd 에 합선된 것처럼 동작하게 됨을 가정한 것

- 신호선이 Vss 와 합선되면 일정한 값 0을 가짐 → stuck-at-0(s-a-0) 고장
  신호선이 Vdd 와 연결되면 일정한 값 1을 가짐 → stuck-at-1(s-a-1) 고장
- 만약 논리게이트, 전가산기 등과 같은 논리연산장치의 입력과 출력에 사용되어진다면 고착고장모델이 가장 효율적
- 테스트에서 이러한 고장모델의 적용은 s-a-0을 위해 강제적으로 신호 선을 1의 값으로 만들고 s-a-1을 위해-신호 선을 0으로 만들며 회로의 응답이 분석되어짐

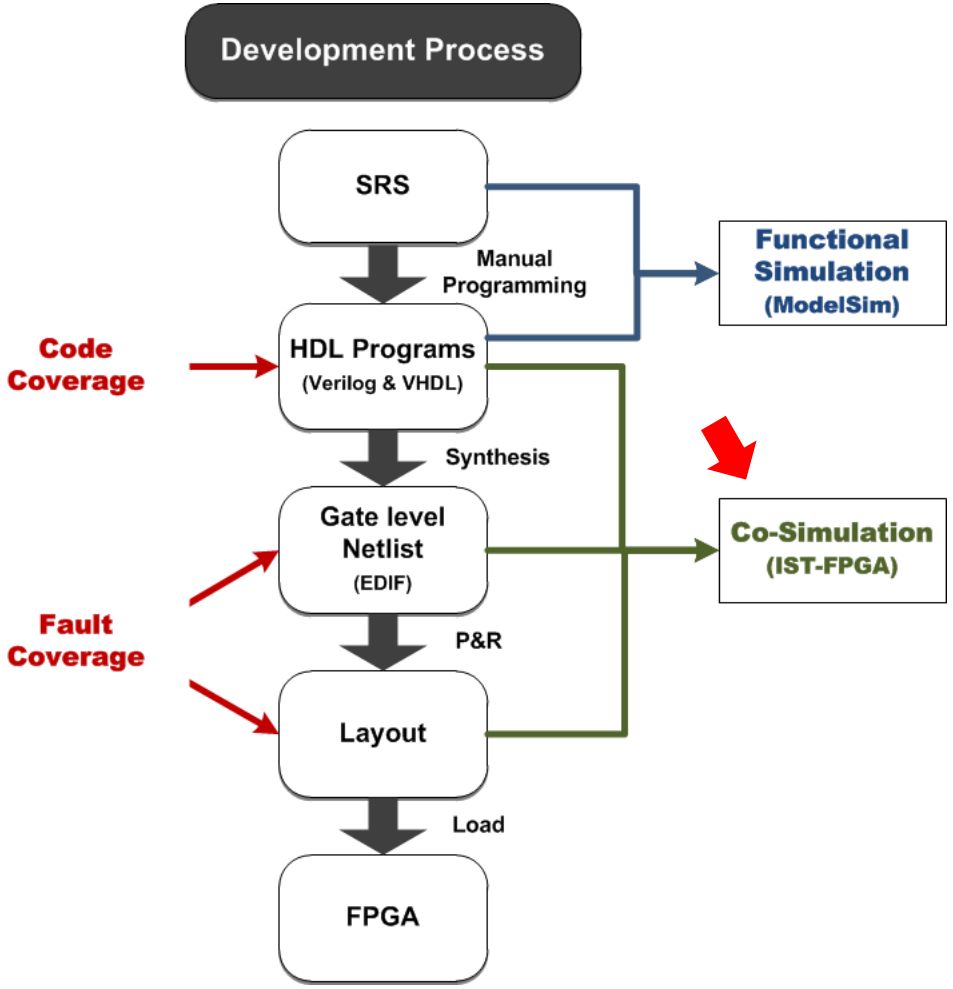


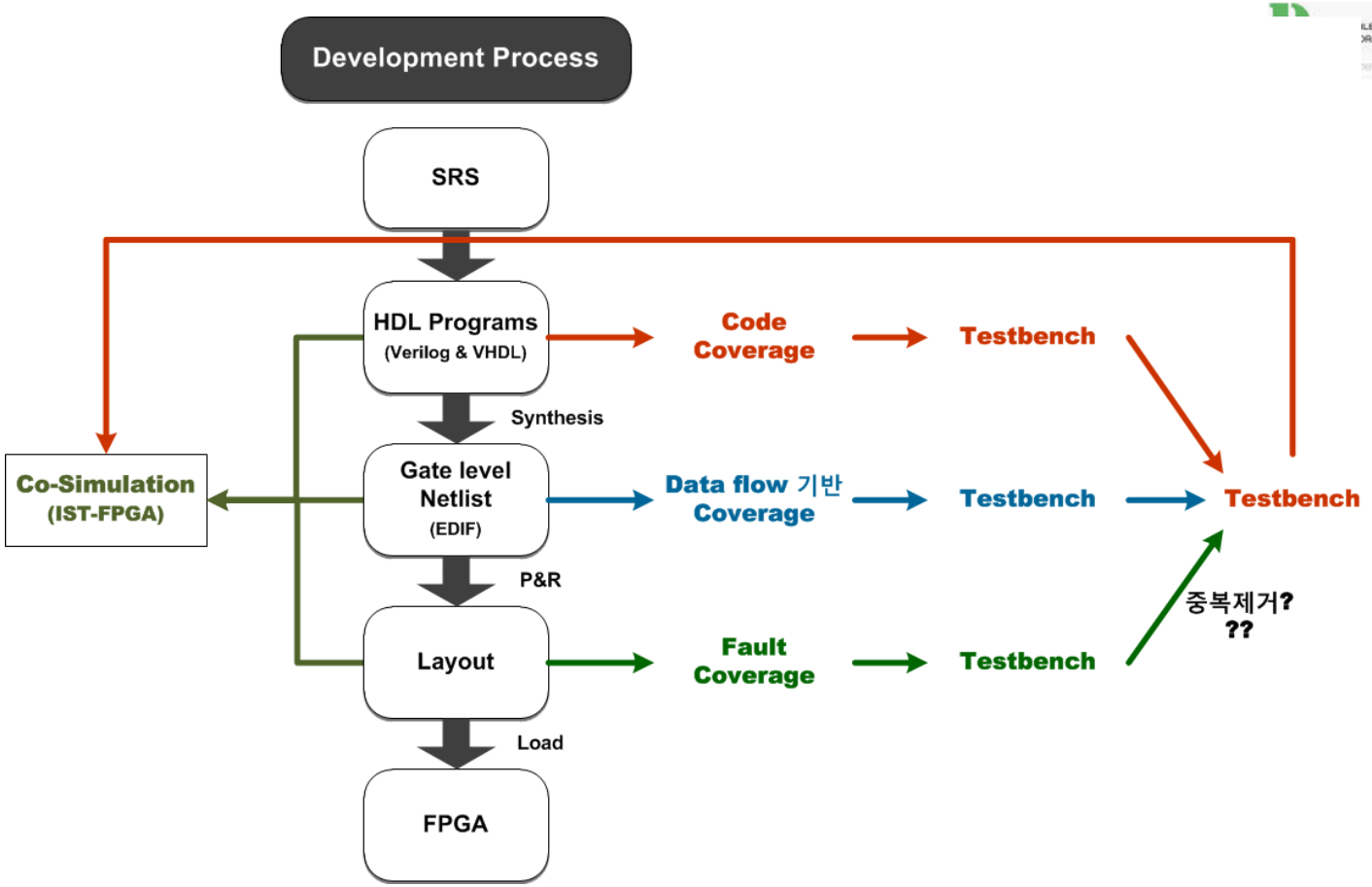
- 각 레벨에 맞는 coverage가 존재
  - **Code coverage는 RTL 단계에서 사용하기 적절**
  - Fault Coverage는 circuit 이나 layout 단계에서 사용하기 적절
  - 100% code coverage를 만족하지만
    Fault coverage는 만족하지 못하는 경우가 있는지 확인

문제점 및 해결책 2

- HDL 단계에서는 Code Coverage
  Gate level 또는 Layout 단계에서는 Fault Coverage 를 많이 씀
  - ATPG – Automatic Test Patten Generation
    - Fault model을 이용해 Fault Simulation을 수행하여 Coverage 확인

- **문제**
  - **각 level 마다 의미 있는 Coverage 가 다르다 (?)**
    - Ex) HDL에서의 Code Coverage가 Gate-level에서 는 의미가 있을 수도 없을 수도 있다.
- **해결**
  - Testbench 생성시 각 level에 적합한 coverage 를 이용할 필요가 있음
- Contribution
  - 각 level간 coverage 의 연관성을 분석 후, Code Coverage만으로 충분한지 평가
  - 각 level을 적절히 만족하는 testbench 생성
  - IST-FPGA에 의미 있게 적용 가능





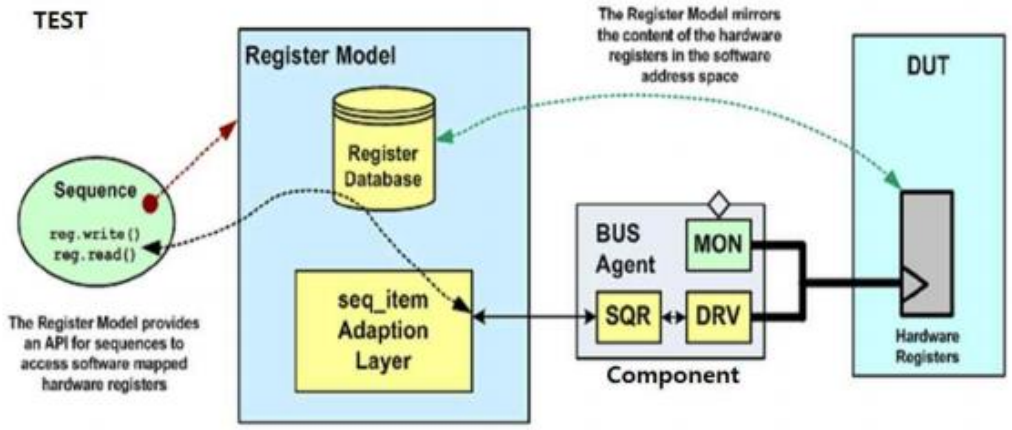
- 걸림돌
  - Gate-level
    - Netlist에 data flow coverage 적용 연구 (new?)
      - 지은경 박사님 방법도 사용할 수 있으면 사용 / HW 에 특징적인 부분을 반영 (동시성 고려)
  - Layout
    - 적절한 fault model 선정 (중요)
  - Simulator의 기능 확장
    - 각각의 coverage를 확인 가능해야 함

- STA
  - Stability Analysis

- Coverage
  - 100% Code Coverage
  - Multi level Coverage

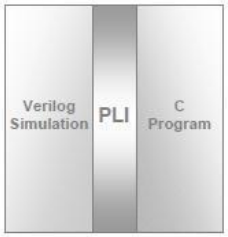
Universal Verification Methodology

- UVM register model은 소프트웨어 주소 공간에 하드웨어 레지스터를 복사, DUT의 register contents(register, memory location)을 추적, 접근하는 방법로 계층적 구조
- UVM 테스트벤치에서 현재의 DUT의 하드웨어 상황을 복제해 mirrored 된 값을 frontdoor, backdoor 방식을 통해 엑세스 하여 register model 데이터베이스에 업데이 트 한다. 이러한 register model은 DUT에 stilmulus를 직접주거나 register model 데이터베이스를 업데이트 하는 방식으로 테스트벤치 구성을 구축
  - 코딩시 UVM 라이브러리를 사용해야 함



<그림 1> UVM REGISTER MODEL의 구조

전략 – 3



- New coverage 개발
  - 1 + 2. 연구의 노하우를 바탕으로..

- 다양한 방면으로 고려
  - Other coverage
    - Observability-based statement coverage metric
      - Base는 statement coverage
      - 변수 중 하나가 correct 인지 incorrect 인지
        어떤 특정 지점에서 monitor 하거나 식별
  - PLI (Programming Language Interface) 사용 고려
    - Verilog 에서 C/C++ 과 같은 언어와의 연동을 위한 API
  - UVM (Universal Verification Methodology)
    - Functional Simulation 을 위한 기법
    - Testbench의 체계적인 생성과 usability 확보 + 체계적인 testbench 생성