솔루션 발표

김의섭

목차

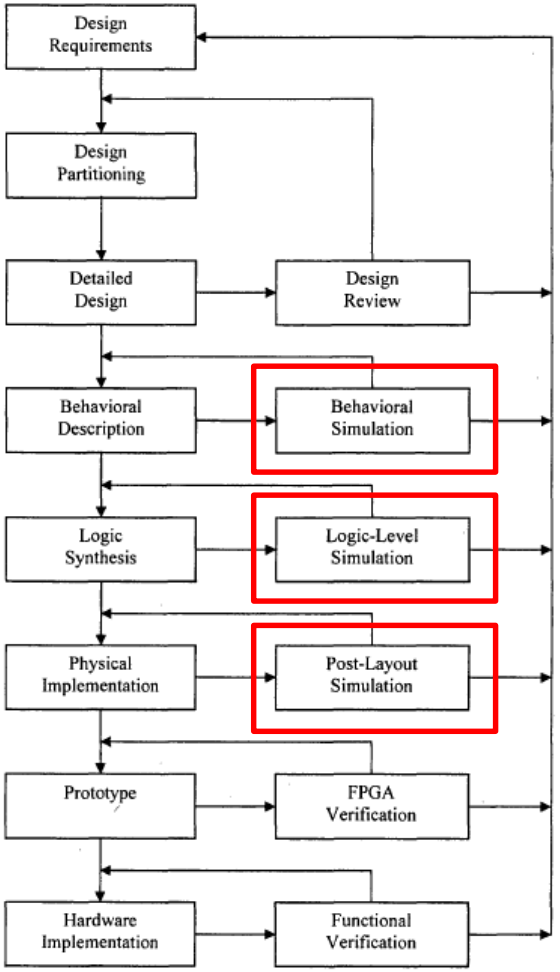
- 1. Coverage 관련
  - TestBench Generation
  - Multi-level Coverage
- 2. Stability Analysis 관련
- 3. 창의과제 논문 관련
  - A Seamless Platform Change of Digital I&Cs from PLC to FPGA: Empirical Case Study
    - An Integrated Development Environment for Developing FPGA-based I&C Software in Nuclear Power Plant

COVERAGE

NUREG/CR – 7006

- 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

DO – 254

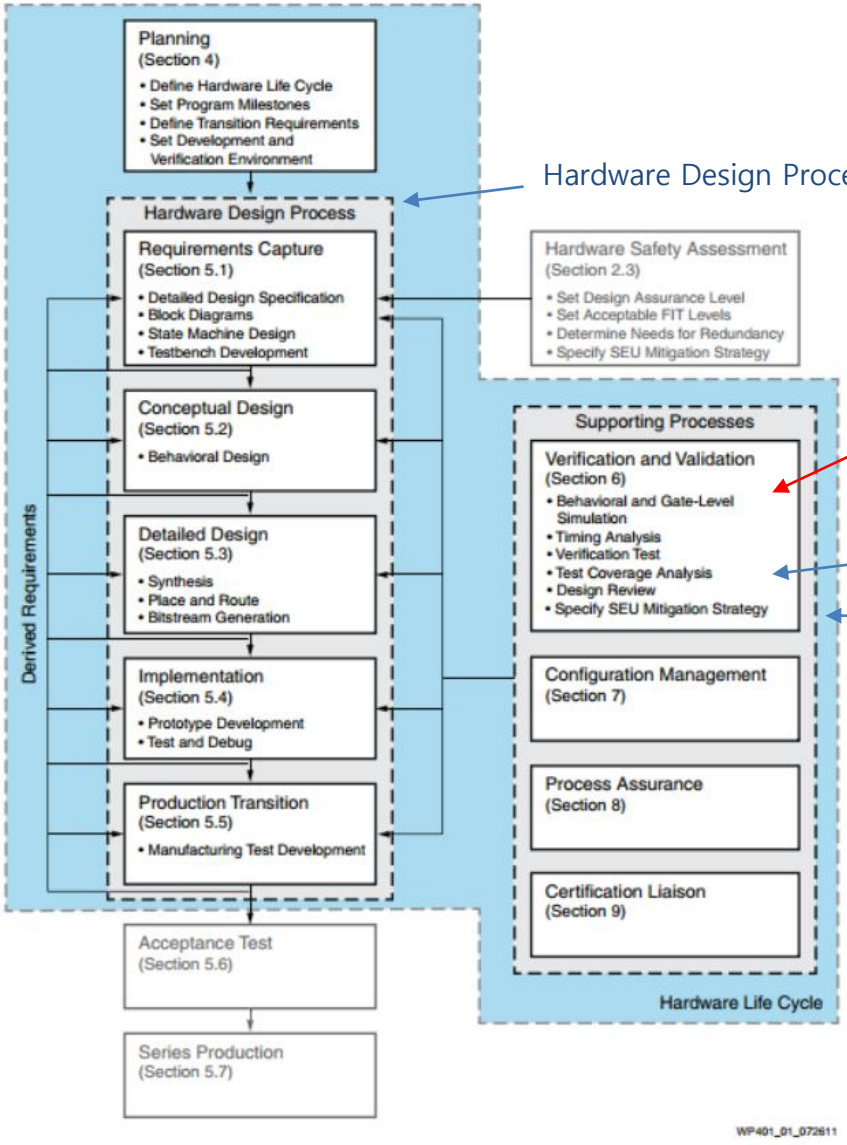


Figure 1: **DO-254 Hardware Life Cycle for Complex Electronic Hardware**

Hardware Design Process (FPGA 개발)

Simulation

100% statement coverage is required by DO-254 (synopsys)

FPGA는 single-event upsets에 민감
High-energy에 의해 silicon 내부적으로
충돌이 생길 수 있음
static memory element (configuration memory) 영향
→ FPGA's programmed logic 변경의 초래

System safety assessment (SSA) 필요 (Xilinx)

EFRI TR – 1019181

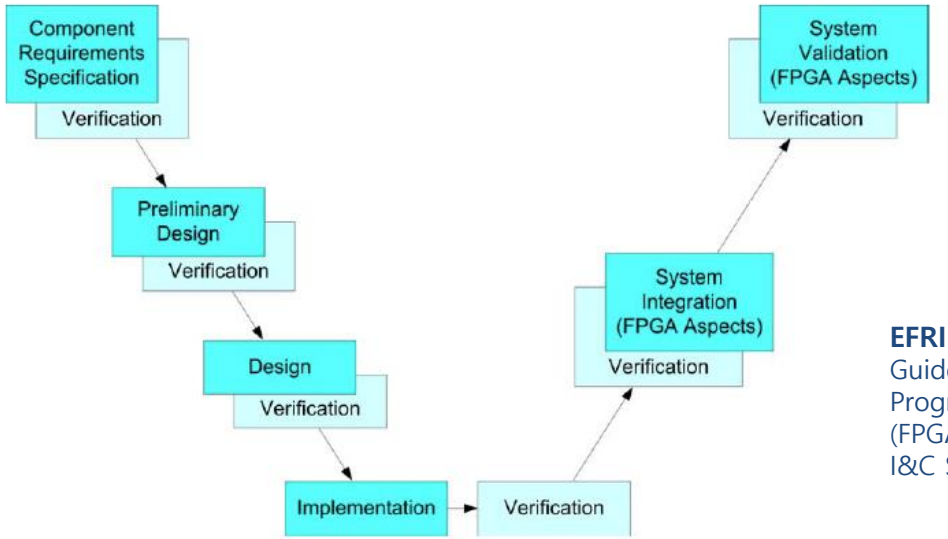


Figure 2-10
V-Shaped FPGA Programming Lifecycle

EFRI TR-1019181
Guidelines on the Use of Field Programmable Gate Arrays (FPGAs) in Nuclear Power Plant I&C Systems

Verification

Testing & Simulation
Formal Verification
Static Timing Analysis
Verification of Synthesis and Place & Route

- 100% code coverage should be sought in each of the following categories:

  Statements – every executable statement is examined for execution counts and those not having at least one execution are flagged;

  Branches – possible outcomes of IF and CASE blocks are examined for execution counts with those not having at least one execution being flagged;

  Conditions – any sub-conditions of a code branch (e.g., IF, AND, CASE, WHEN statements) are examined as to fulfillment and those not having at least one true outcome are flagged;

  Paths – any un-traversed paths are flagged;

  Triggering – checks whether a single signal or a combination of signals is responsible for trigger execution, and any candidate signals that have not had their own individual opportunity to trigger the process or statement are flagged.

- The code should be successively refined until the goal of 100% coverage of the above categories is met.

DO – 254

XILINX

Hardware Design Process

The hardware design process phase is broken into five distinct sub-processes that must be documented:

- Requirements capture: The architecture of the system (and the system-level requirements), including items such as test structures and interfaces, needs to be described and documented. During this phase, the design team must develop a block-level description of the system, including block diagrams, state diagrams, and flow charts, that are consistent with the requirements.
- Conceptual design: During this phase, hardware design can begin with HDL development. The output of these activities plus the results of preliminary design review are submitted to the DER for review.
  - Simulation, while part of verification and validation, is considered a natural part of the conceptual and detailed design processes.
- Detailed design: During this phase, the design is synthesized, and place-and-route is completed. After the confidence in the design is high, bitstreams are generated.
- Implementation: During this phase, FPGAs are programmed and prototypes are developed to allow test and debug.
- Production transition: In the last phase, the FPGA/board is prepared for release to manufacturing. After the board and FPGA are fully debugged, test engineering completes production and reliability testing, a baseline is established to ensure consistent system production, acceptance testing is defined.

현안

- FPGA 개발에 있어 Simulation은 필수

- Simulation을 수행 하는데 있어 Code Coverage를 만족할 것을 요구
  - **산업에서는 Code Coverage만 취급 (+Functional Coverage)**
    - 다양한 연구실에서 새로운 (개선된) coverage를 제시하고 있지만 사용되고 있지 않음
      - 공신력↓ (사용 이력 X, 상호 합의 X, standard or citification 에서 요구 X)
    - Do-254 – code coverage 100%
      - "The document should also spell out goals for code and assertion coverage (at minimum, 100% statement coverage is required by DO-254)."
    - 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..

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

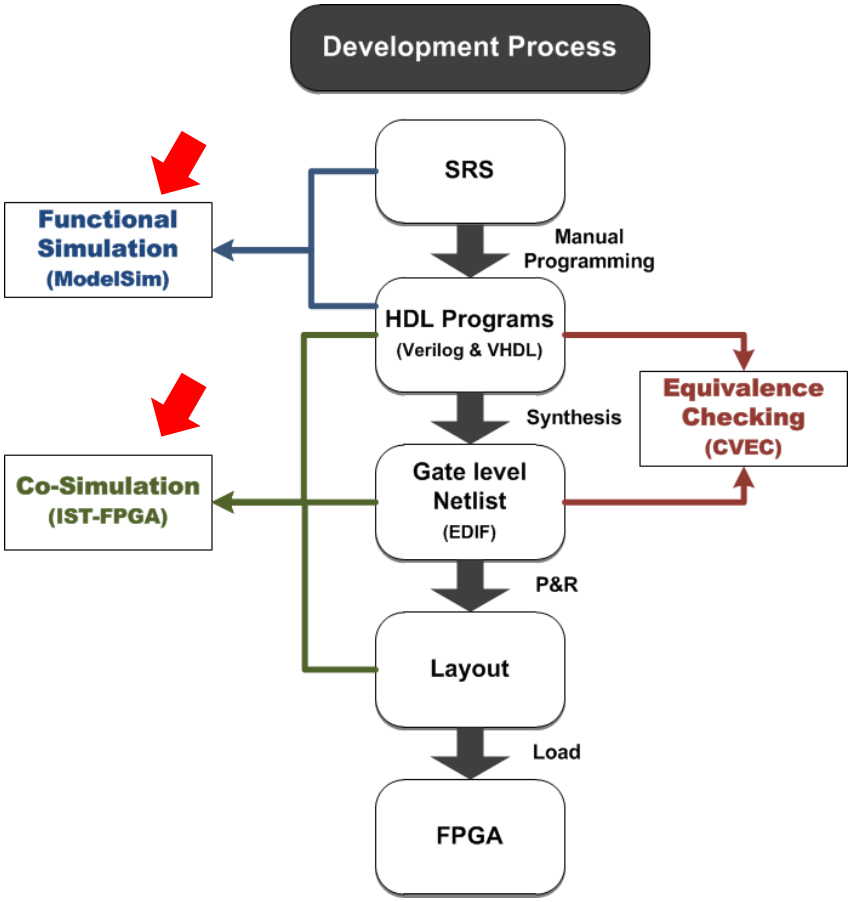
Others

- 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
  - Coverage 중 code coverage (+ fault coverage) 이외는 언급이 거의 없음
  - 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~~

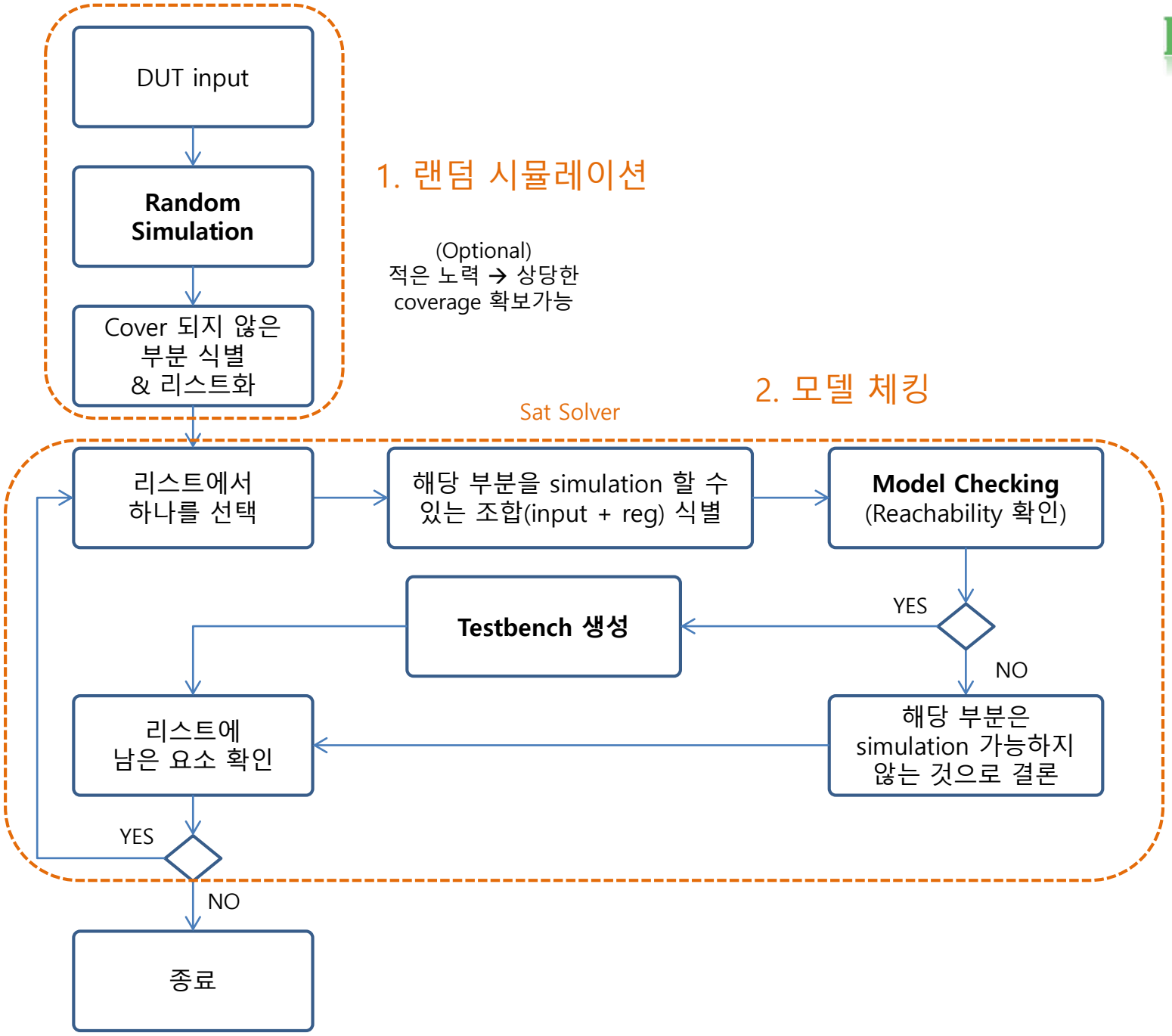
문제점 및 해결책 1

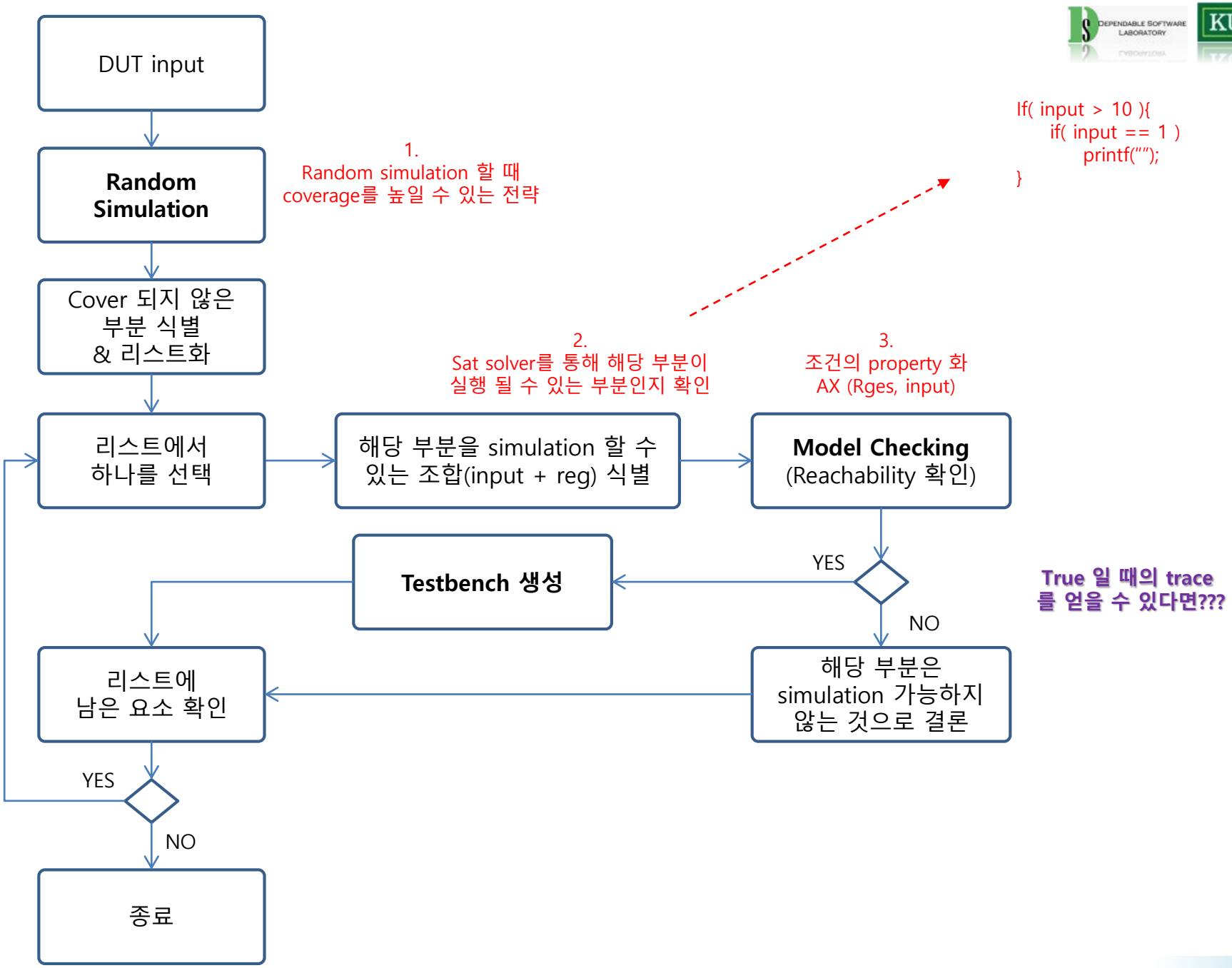
- 일반적으로 HDL 단계에서는 Code Coverage를 많이 씀
  - Statement Coverage / Branch Coverage / Toggle Coverage …
- **문제**
  - Code Coverage 를 측정하는 방법은 많이 있지만,
  - Code Coverage 를 충분히(100%) 만족하는 Testbench를 생성하는 기술은 미비 (?)
- **해결**
  - Code Coverage 100%를 만족하는 testbench 생성의 필요

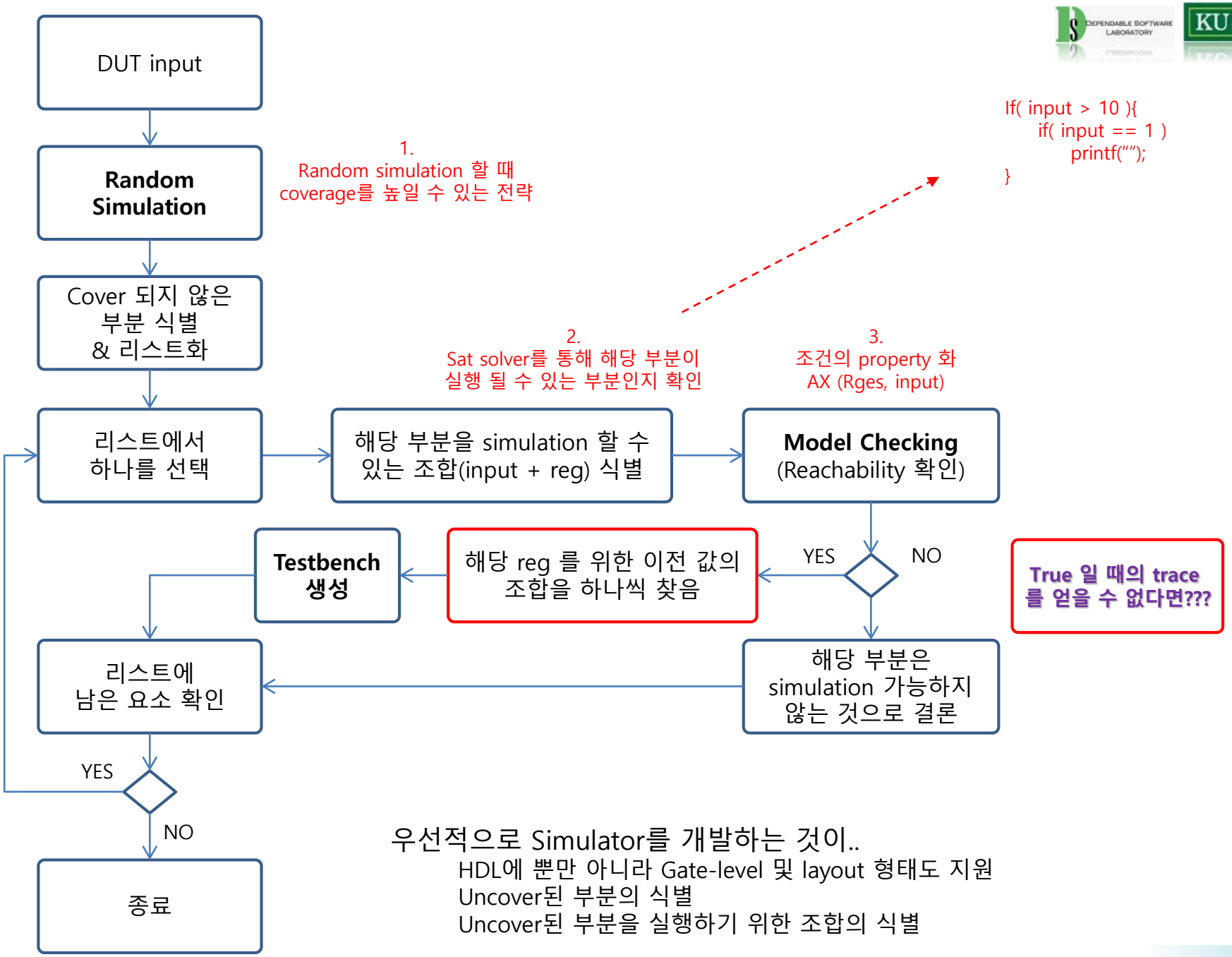


전략

- 100% Code coverage 를 만족 하는 testbench 생성 연구
  - 100% 가 어려운 이유
    - Testing의 경우 해당 부분을 수행하기 위한 input 및 reg만 조합 필요
    - Simulation의 경우 reg가 연속적
    - 따라서, 연속적인 reg의 조합을 찾아야 함 (조합의 조합의 … 을 찾아야 함)
  - Random simulation + Model checking (+Sat Solver)를 이용
    - Random simulation으로 일정부분 coverage를 확보하고,
    - 특정부분에 대해서는 model checking을 활용


DUT input
Random Simulation
Cover 되지 않은 부분 식별 & 리스트화
1. 랜덤 시뮬레이션
(Optional)
적은 노력 → 상당한 coverage 확보가능
2. 모델 체킹
Sat Solver
리스트에서 하나를 선택
해당 부분을 simulation 할 수 있는 조합(input + reg) 식별
Model Checking (Reachability 확인)
YES
Testbench 생성
NO
리스트에 남은 요소 확인
해당 부분은 simulation 가능하지 않는 것으로 결론
YES
NO
종료


DUT input
Random Simulation
Cover 되지 않은 부분 식별 & 리스트화
리스트에서 하나를 선택
1.
Random simulation 할 때 coverage를 높일 수 있는 전략
2.
Sat solver를 통해 해당 부분이 실행 될 수 있는 부분인지 확인
해당 부분을 simulation 할 수 있는 조합(input + reg) 식별
3.
조건의 property 화
AX (Rges, input)
Model Checking
(Reachability 확인)
If(input > 10){
if(input == 1)
printf("");
}
YES
Testbench 생성
NO
해당 부분은 simulation 가능하지 않는 것으로 결론
True 일 때의 trace 를 얻을 수 있다면???
리스트에 남은 요소 확인
YES
NO
종료



우선적으로 Simulator를 개발하는 것이..

HDL에 뿐만 아니라 Gate-level 및 layout 형태도 지원

Uncover된 부분의 식별

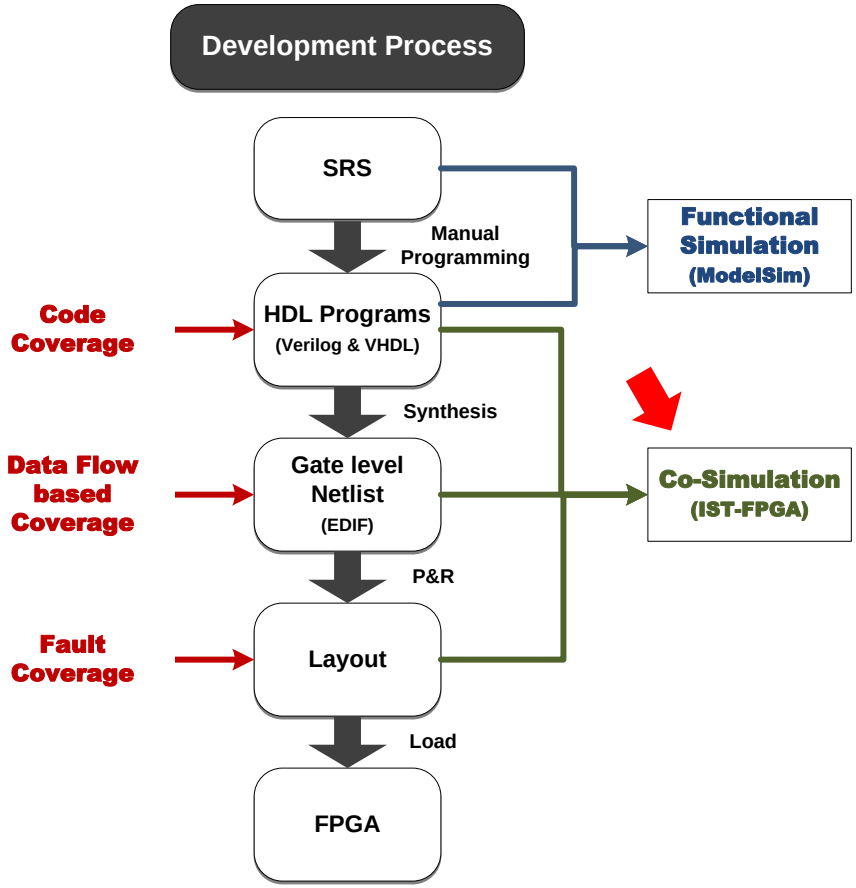
Uncover된 부분을 실행하기 위한 조합의 식별

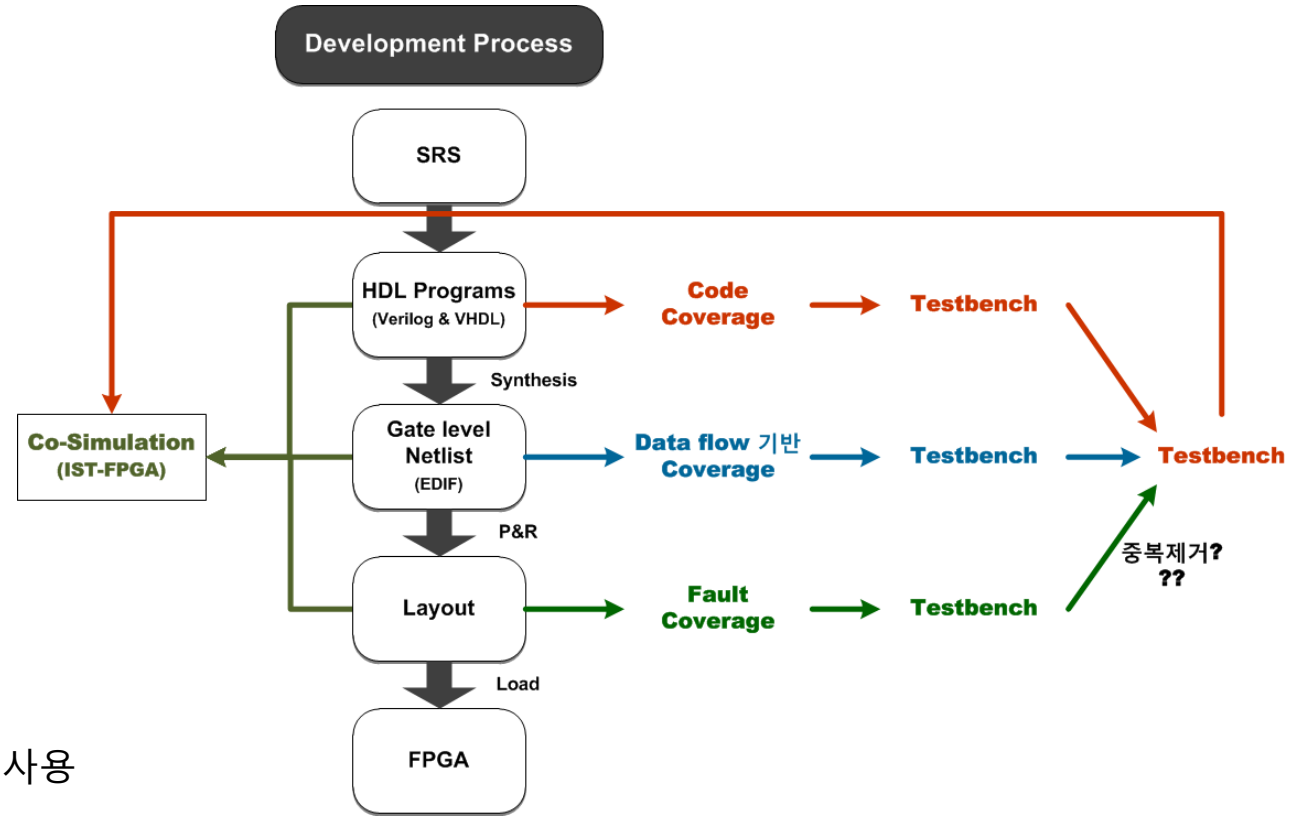
연구 진행 방향

- 시나리오 생성기 구현
  - 1. Simulator를 개발
    - Uncover된 부분의 식별
    - HDL에 뿐만 아니라 Gate-level 및 layout 형태도 지원
    - Uncover된 부분을 실행하기 위한 조합의 식별
  - 2. Coverage 확인 기능 구현
  - 3. Uncover된 부분의 식별 및 Sat변환 기능 구현
  - 4. 모델체킹 후 testbench 생성 기능 구현
  - → 100% Code coverage 를 만족 하는 testbench 생성 획득 가능

문제점 및 해결책 2

- FPGA 개발 각 단계마다 유효한 coverage가 다름
  - HDL 단계 → Code Coverage
  - Gate level/Layout 단계 → Fault Coverage
- 문제
  - 각 level 마다 유효한 Coverage 가 다른데, gate level에서 HDL 단계의 testbench를 사용 (code coverage 바탕으로 생성)
    - Ex) HDL에서의 Code Coverage가 Gate-level에서는 의미가 있을 수도 없을 수도 있다.
- 해결
  - 각 level에 적합한 coverage 를 사용하여 Testbench 생성
- 전략
  - 각 level간 coverage 의 연관성 분석
    - Code Coverage만으로 충분한지 평가
  - 각 level을 적절히 만족하는 testbench 생성
  - IST-FPGA에 의미 있게 적용

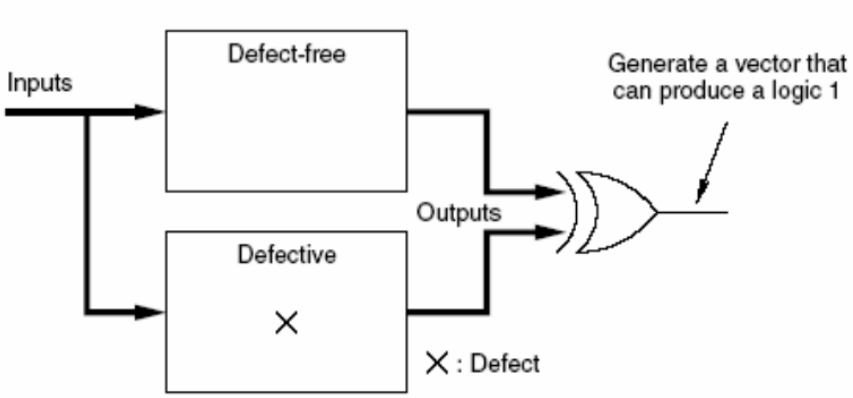




- Technique
  - HDL
    - Code Coverage 사용
  - Gate-level
    - Netlist에 data flow coverage 적용 연구 (new?)
      - 지은경 박사님 방법도 사용할 수 있으면 사용 / HW 에 특징적인 부분을 반영 (동시성 고려)
  - Layout
    - 적절한 fault model 선정 (중요) 과 Fault Coverage 사용
  - Simulator의 기능 확장
    - 각각의 coverage를 확인 가능해야 함

Fault Coverage

- 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**

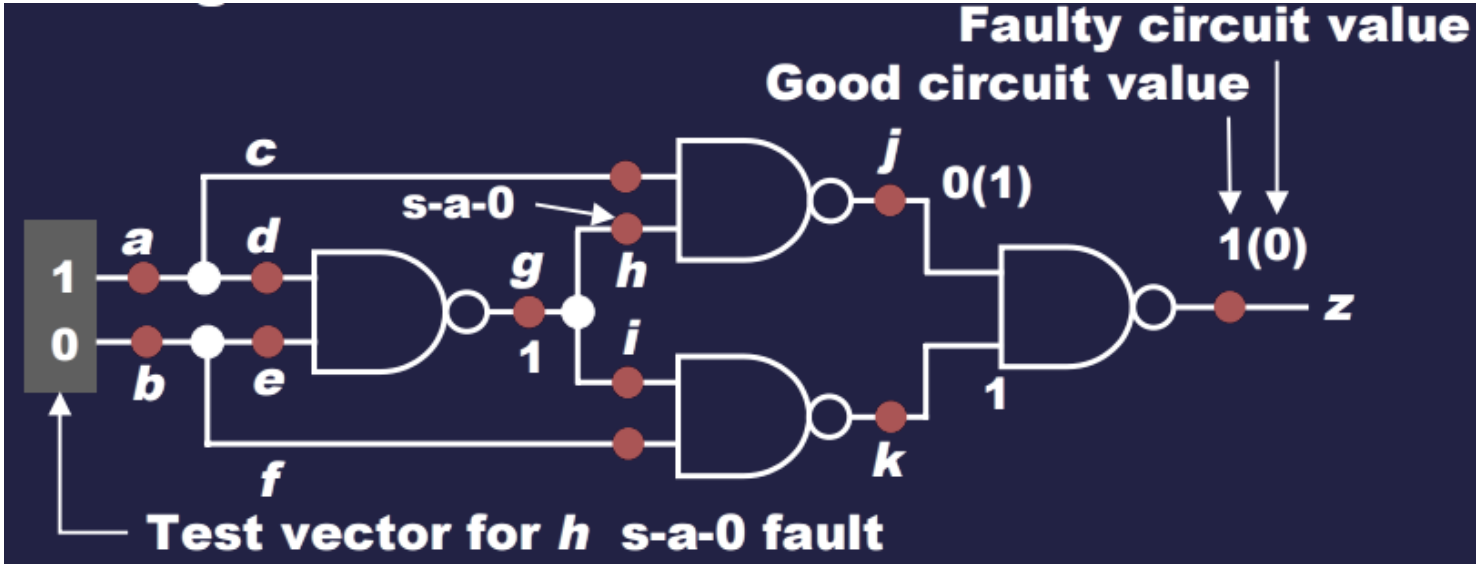
- 주어진 회로에 대하여 Fault model을 가정하여 Fault 을 검출하는 입력 패턴 생성을 자동화 시키는 과정.
  - Original design 과 fault model 을 생성하여 동일한 입력을 주어 결과값이 동일하지 않으면 fault 를 찾은 것
  - 이때 다른 결과를 내는 입력이 Test pattern

- 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



연구 진행 방향

- 1. Data Flow 기반 Coverage의 netlist 적용

- 2. 시뮬레이터 기능 확장
  - Code Coverage 를 체크하고 test case를 생성할 수 있는 기능 구현 (1)
  - Data Flow 기반 Coverage 를 체크하고 test case를 생성할 수 있는 기능 구현
  - Fault Coverage 를 체크하고 test case를 생성할 수 있는 기능 구현

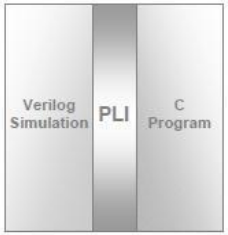
- 3. 각 레벨의 coverage간의 관계 분석
  - 100% code coverage를 만족하지만
    Fault coverage는 만족하지 못하는 경우가 있는지 확인

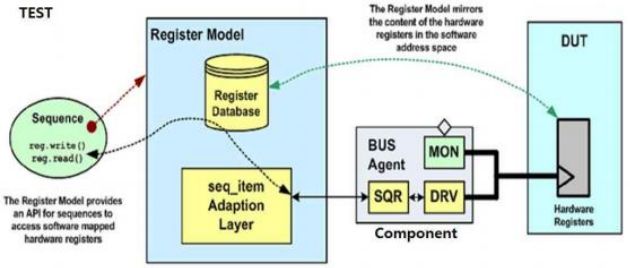
- 4. IST-FPGA에 적용

전략 – 3

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

- 다양한 방면으로 고려
  - 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





<그림 1> UVM REGISTER MODEL의 구조

STABILITY ANALYSIS

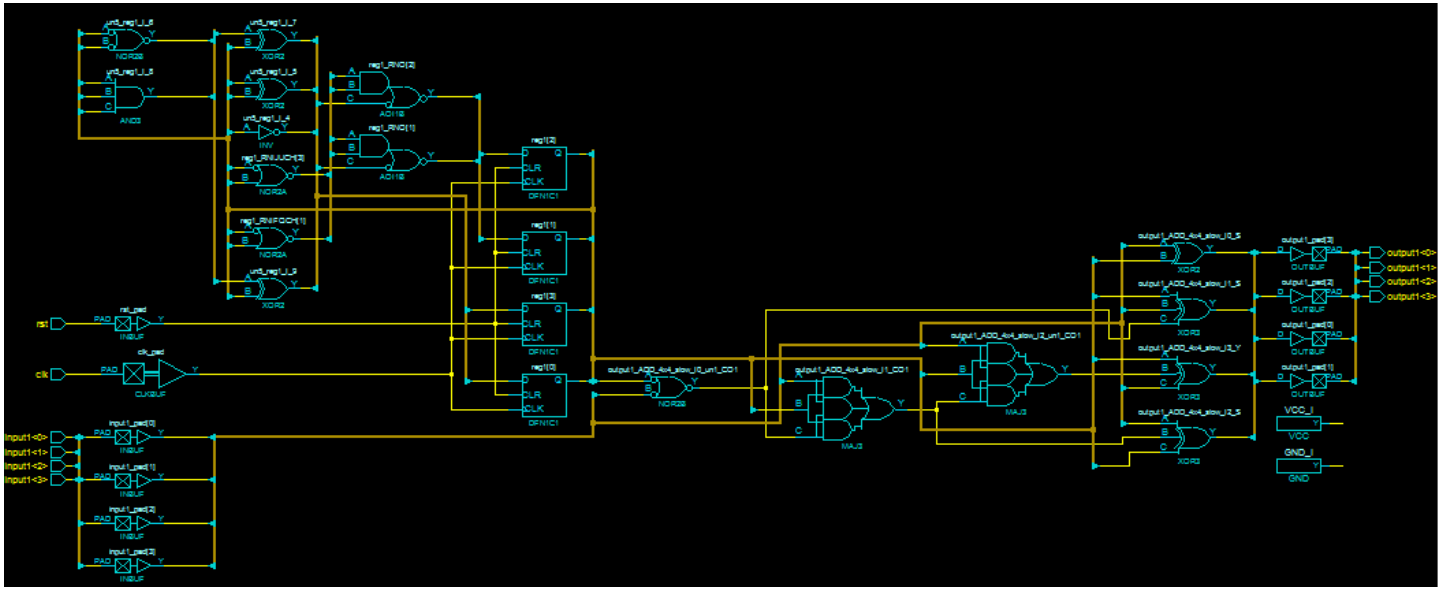
Stability

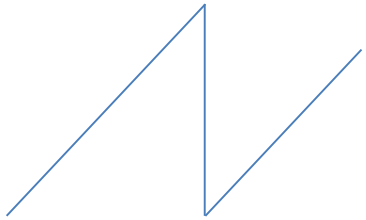
- 현황
  - 어떤 사용자는 input을 주었을 때 FPGA가 일정 주기 후 stable 해진 output 값 원함

- 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)
    - 따라서, STA나 기타 simulation으로는 해당 요소를 찾을 수 없다.
    - 개발자의 분석 노하우를 바탕으로 진행 가능? (heuristic)
    - 따라서 적절한 해결책이 필요

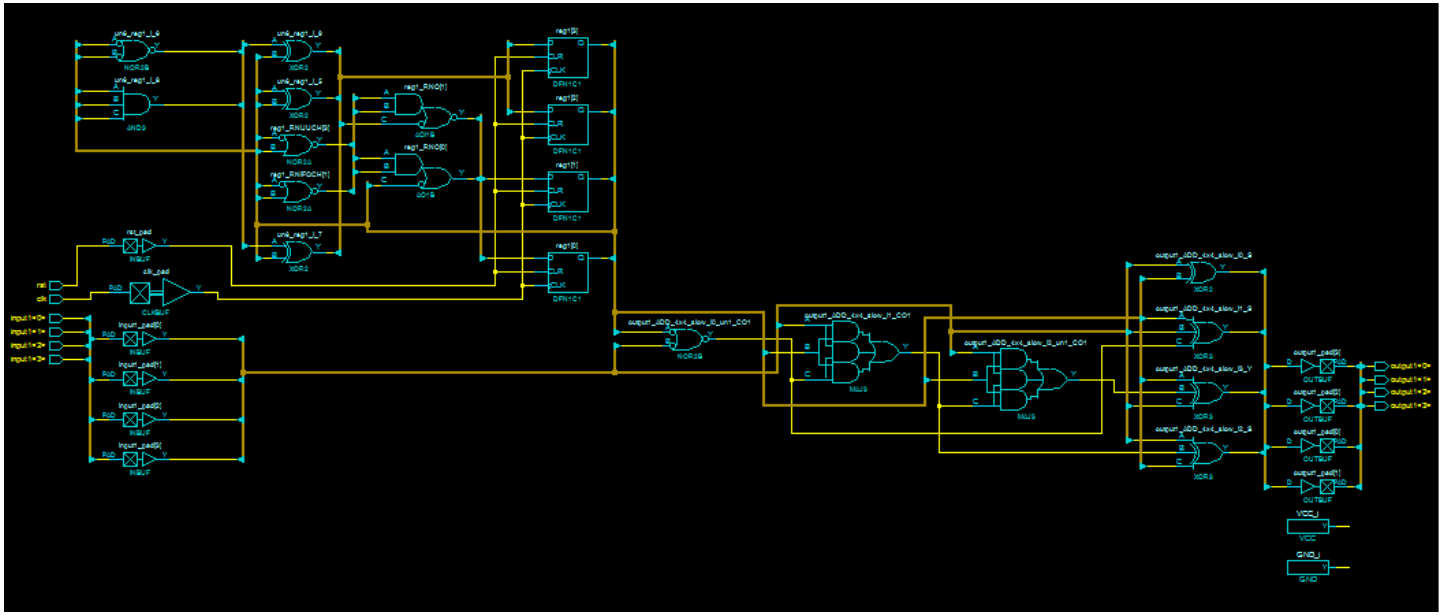
Static Timing Analysis

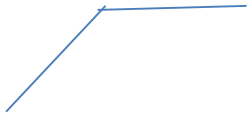
- Glitch (clk skew)
  - 글리치(Glitch)는 디지털 회로에서 발생할 수 있는 매우 짧은 기간동안 나타나고, 사라지는 전압이나, 전류의 원하지 않은 노이즈 펄스이다.
  - 이러한 노이즈 펄스의 폭은 소자의 물리적 특성과 사용 환경(온도, 습도, 진동 등)에 따라 변할 수 있다. 따라서, 글리치는 사용조건에 따라 어떤 때는 논리회로에 전혀 영향을 주지 않을 수도 있고, 어떤 때는 논리회로의 오동작을 발생시킬 수 있다.
- Metastability
  - Setup-time, hold-time
  - 클록의 상승에지 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





1 2 3 4 5 6 7 8



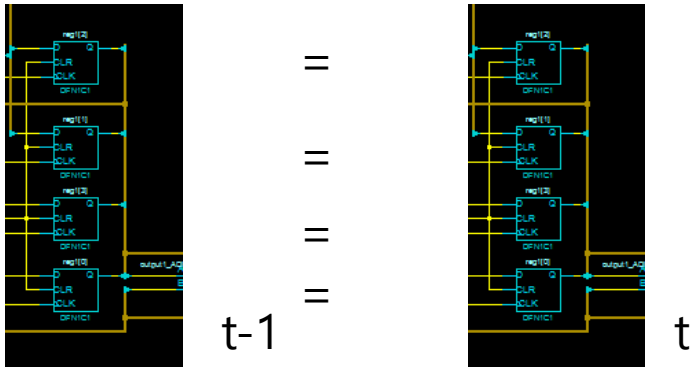


1 2 3 4 5 6 7 8

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)

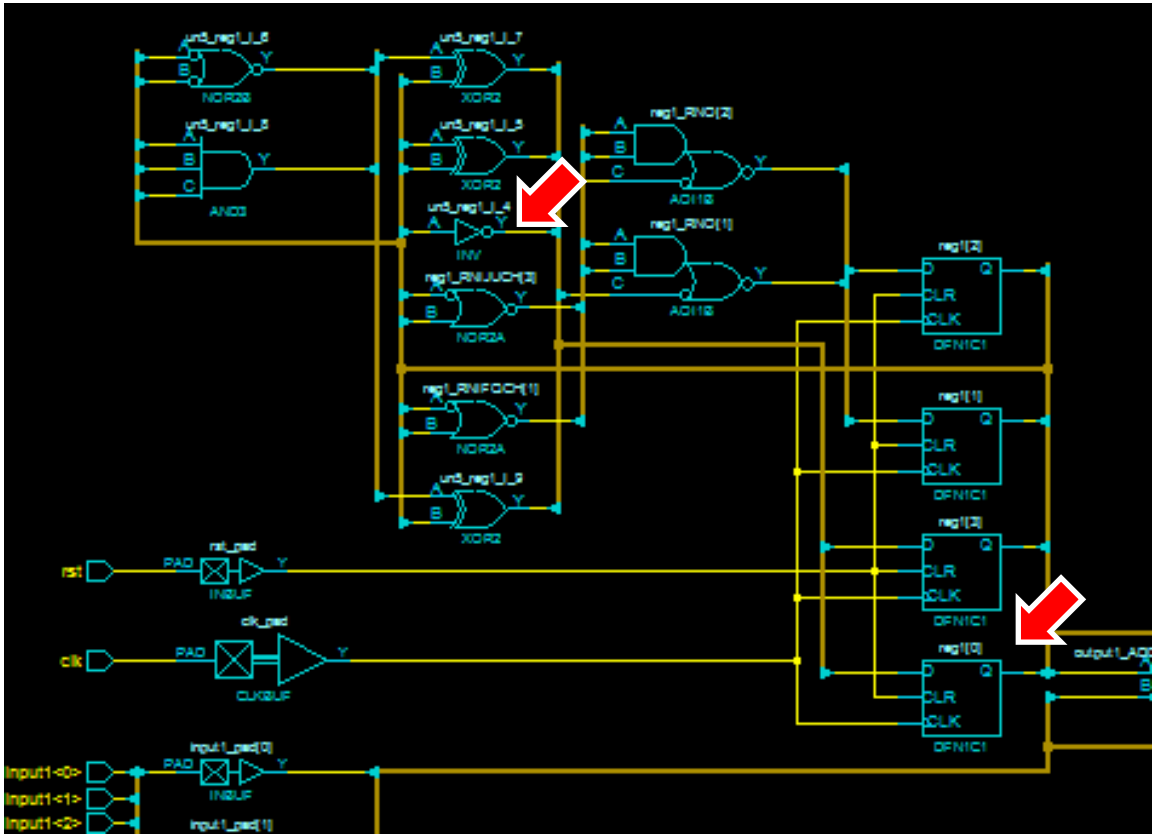


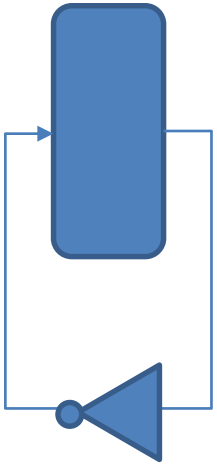
전략

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

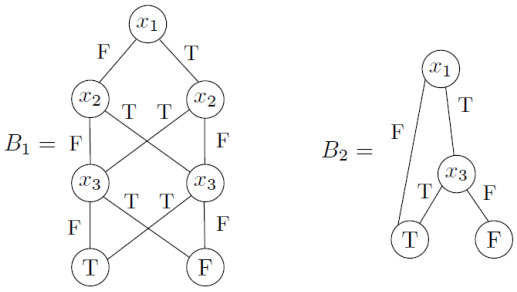
- 2. Stable 해질 가능성이 있다면, 언제 stable 해질지 계산
  - 모든 input 조합을 바탕으로 reg들의 t-1 값과 t값의 비교를 수행
    → 동일하면 stable
    → worst를 찾기 위해서는 모든 input 조합에 대해 수행해야 함

- 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

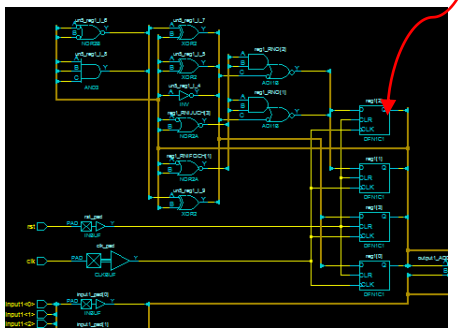




- 1. logic이 stable 해질 수 있는지 체크
  - Ex) input이 stable 하게 들어오는 logic인지 확인
  - Ex) 플리플롭의 값이 자체적으로 계속 변하게 되는 로직인지 확인
- 2. Stable 해질 가능성이 있다면, 언제 stable 해질지 계산
  - 모든 input 조합을 바탕으로 reg들의 t-1 값과 t값의 비교를 수행 → 동일하면 stable
    → worst를 찾기 위해서는 모든 input 조합에 대해 수행해야 함
    - Reg의 값은 수식으로 표현 가능 (Boolean 연산)
      - 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
- BDD는 연산이 아니라 카노 맵 형태

이용할 만한 정보 (빠른 성능 보장)
- reg는 0부터 시작
- input은 고정 (0 or 1) → 재사용

연구 진행 방향

- 도구 구현
  - 1. logic이 stable 해질 수 있는지 체크 방법
  - 2. BDD 연산을 이용하는 방법
    - 알고리즘 구현

- Stability Analysis 통해
  - FPGA가 일정 주기 후 stable 해진 output 을 출력하길 원하는 사용자에게 언제 stable 해질지
  - Worst case의 t주기를 알려 주게 될 것임

창의과제

창의과제

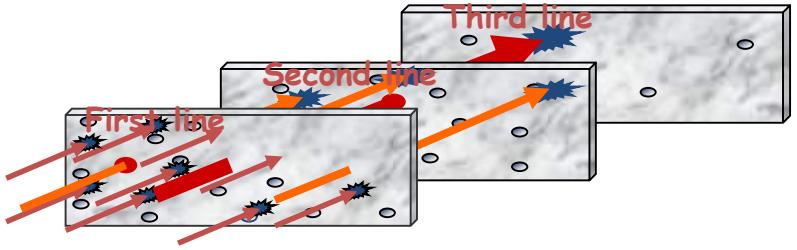
- A Seamless Platform Change of Digital I&Cs from PLC to FPGA: Empirical Case Study

목차

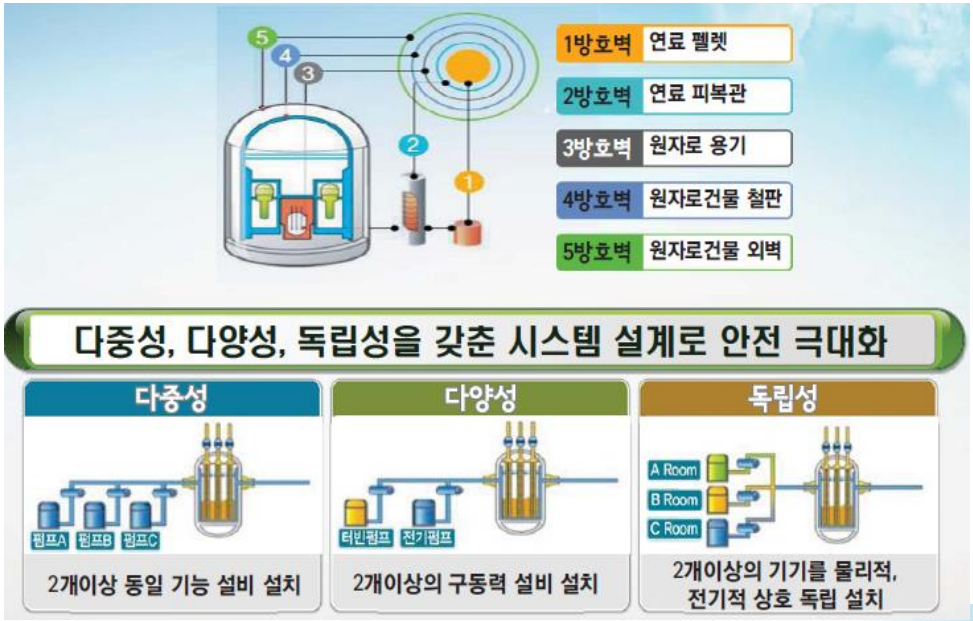
- 1. Introduction
- 2. Background
  - PLC Development Life Cycle
  - FPGA Development Life Cycle
- 3. An Integrated Development Environment
  - Overview
  - Development Tools
    - FBD Editor …
  - V&V Tools
    - IST-FPGA …
- 4. Case Study
  - From FBD (BP Logic - ref 하여 자체 제작)
    - Development
    - V&V
- 5. Further Consideration
  - COTS Synthesis tool ← COTS Dedication
  - Coverage
  - Stability
- 6. Conclusion

배경

- 원전 다양성 및 심층방어 개념
  - 기본 개념
    - (물리적 다중방벽 + 다단계 방호)
      - 사고예방 + 사고 완화



- 계측제어계통 다양성 설계 확보 필요
  - 원전 계측제어 계통 아날로그-기반 시스템에서 디지털-기반 시스템으로 변경
  - 전세계적으로 신규 인허가현안 발생
    - 사이버보안
    - CCF(공통원인고장)



연구배경

CPU-기반 제어기

FPGA-기반 제어기

Hardware Description Language
- VHDL
- Verilog

FPGA 로직 개발 공정
- 병렬처리
- 합성
- 배선 및 배치
- Gate Logic into FPGA

Graphic Language
- FBD
- LD

High Level Language
- Fortran
- C

소프트웨어 개발 공정
- 순차 처리
- 컴파일
- 메모리 다운로드

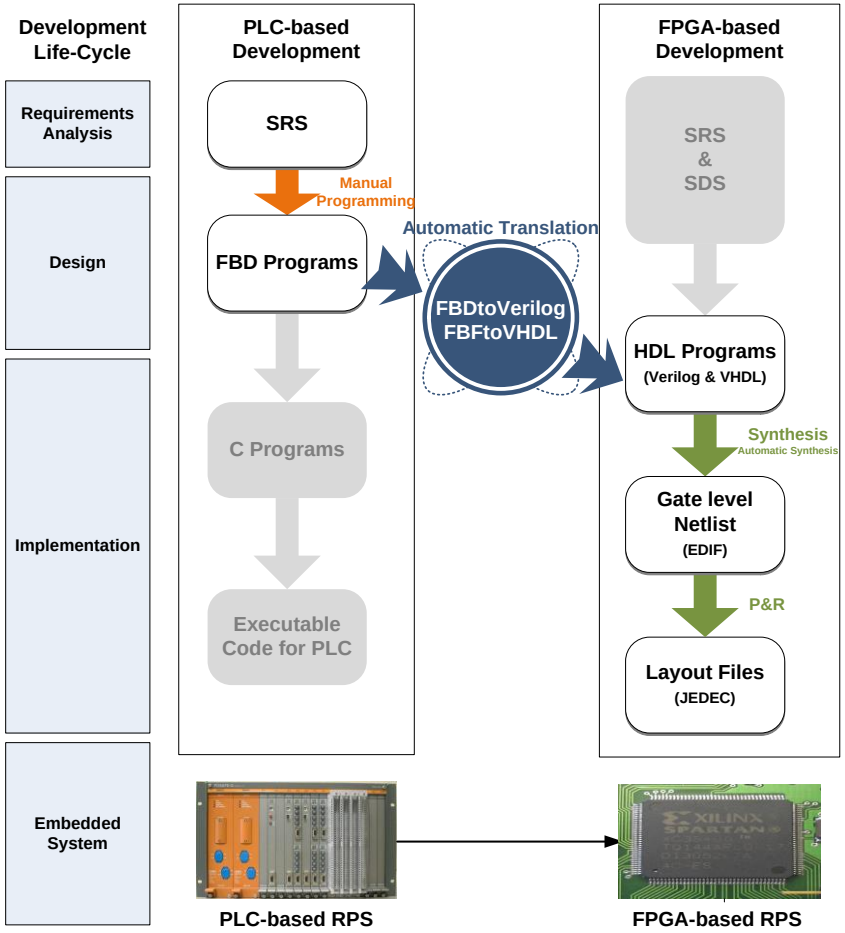
- 소프트웨어 개발 경험 , 노하우 포기
- 소프트웨어 엔지니어 전환교육 필요
- 새로운 HW 개발방법론 습득

원전 계측제어 전문가

38

기존 연구

- 기존 연구
  - FBD를 HDL로 변환을 통해 당면한 문제 해결하려는 시도
- 문제
  - 도구 미 구현
  - 검증 관련 부분의 누락
- 그래서 우리가 통합 개발 환경을 구현하게 되었다.
  - 도구 구현을 통한 사용성 확보
    - FBD Editor
  - 다양한 검증 도구 및 프로세스를 통해 최종 FPGA의 Correctness 확보
    - 검증 가능한 방법 및 도구 제시
  - 언어적 다양성 확보
    - 기존 Verilog 외 VHDL 변환 지원
  - 변환의 신뢰성 확보
    - 전문가가 작성한 Lib 사용
    - 검증 가능한 방법 및 도구 제시



- 진행 예정...