Development of Safety-critical Software Exhaustive Testing Framework for Nuclear Power Plant Digital I&C System

Committee Members:

Prof. Kang, Hyun Gook

Prof. Danon, Yaron

Prof. Liu, Emily

Prof. Milanova, Ana L.

Sang Hun Lee

Ph.D. Defense

10/30/2019 10:00 a.m.

JEC 3117

Contents

- Introduction
  - Research Background / Literature Review
  - Research Objective / Scope
- Methodology
  - Simulation-based Test Environment for NPP Safety Software
  - Exhaustive Test Case Generation Framework for NPP Safety Software
- Application
  - Lamp On / Off Software
  - Reactor Protection System Trip Logic Software
- Conclusion
  - New Academic/Technical Contributions
  - Thesis Summary
- Appendix I. ~ XI.

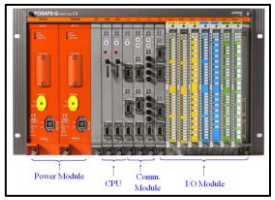
Digital I&C Systems in NPP

Traditional analog-based MCR of NPP

Advanced MCR of APR-1400 NPP

Features
- Control Panel
- Indicator Light
- Hard-wired Recorder

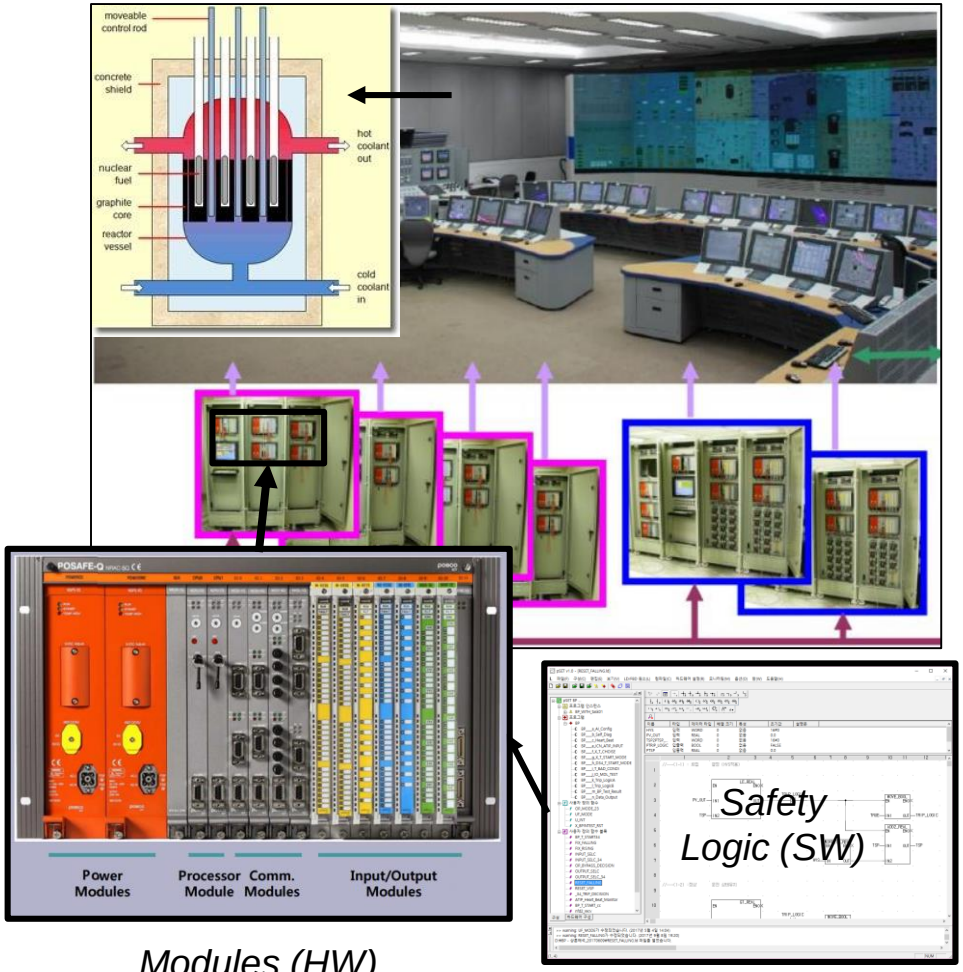
Platform
- Switches (Rotary/Push-button)
- Analog-circuit-based Controller



Features
- Large Display Panel
- Safety Console
- Workstation/Soft Control

Platform
- Distributed Control System
- Programmable Logic Controller

Software is important for NPP safety



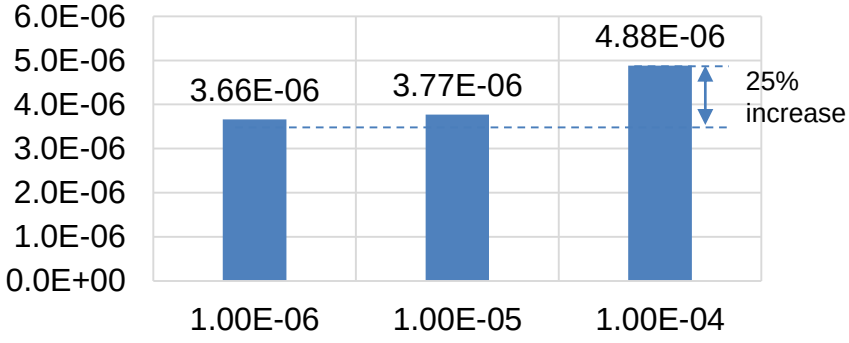
Modules (HW)

PLC used in NPP safety I&C systems

Digital system failure events reported in LERs

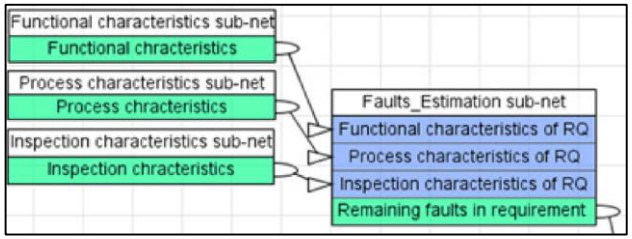
Category	Number of Event
Software error	30 (39.97%)
HMI error	25 (31.65%)
EM interference	15 (18.99%)
Random HW failure	9 (11.39%)
Total	79

RPS unavailability for various software failure probability



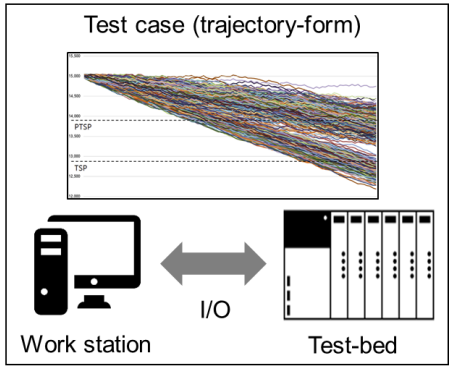
Literature Review

- **Model-based approach**
  - Software Reliability Growth Model
  - Bayesian Belief Network
    - Limitation: - Lack of software failure data
      - Model/parameter uncertainty



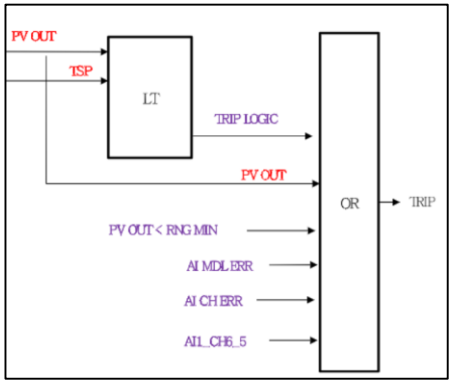
Example of Bayesian belief network

- **Test-based approach**
  - Black-box testing
    - Test cases sampled from SW operational profile
    - Limitation: - Hard to handle SW internal variables
      - Long testing time per test case



Example of black-box testing

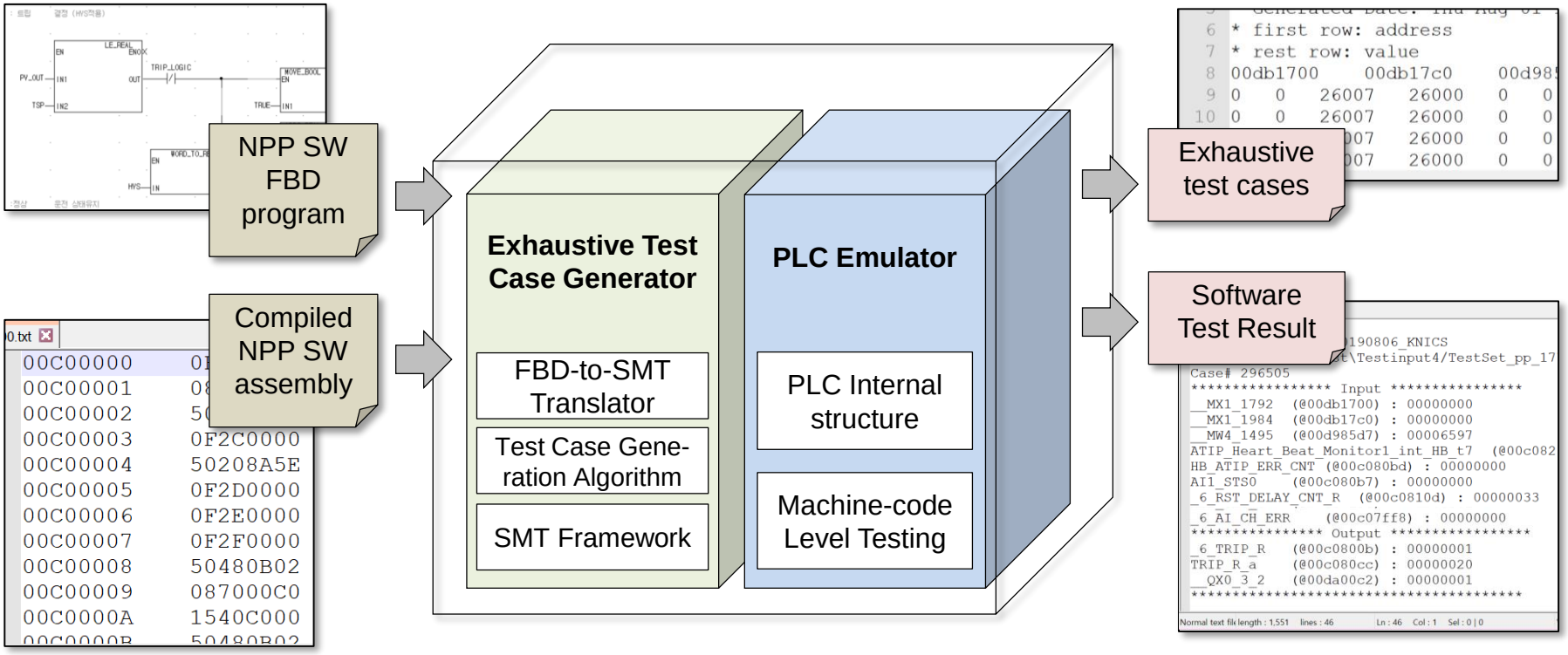
  - White-box testing
    - Test cases derived based on SW internal structure
    - Limitation: - Complex analysis on SW logic
      - Necessity to investigate full range of SW input/internal variables makes it hard to test using hardware test-beds



Example of software logic analysis

Research Objective

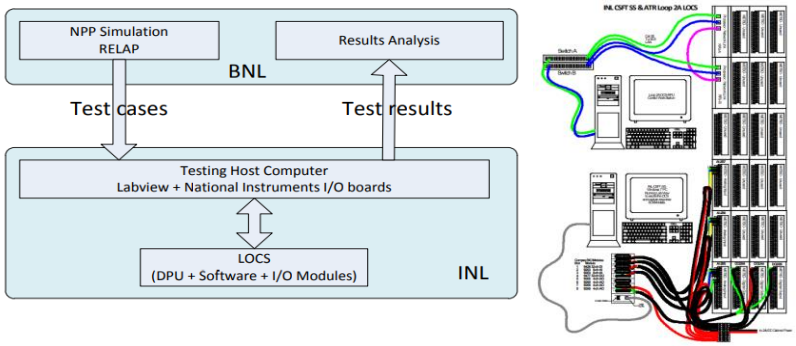
- Integrated Exhaustive Testing Framework for NPP Safety Software
  - Emulator for PLC Software Testing
  - SMT-based Exhaustive Test Case Generation



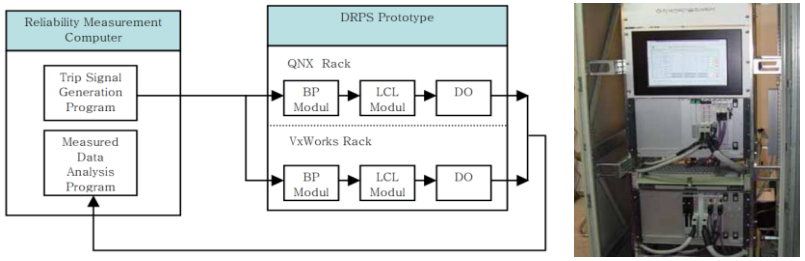
Block Diagram of NPP Software Exhaustive Testing Framework

Need for Effective Software Test-bed

Time per test case : ~ few minutes - hours (Trajectory-based)



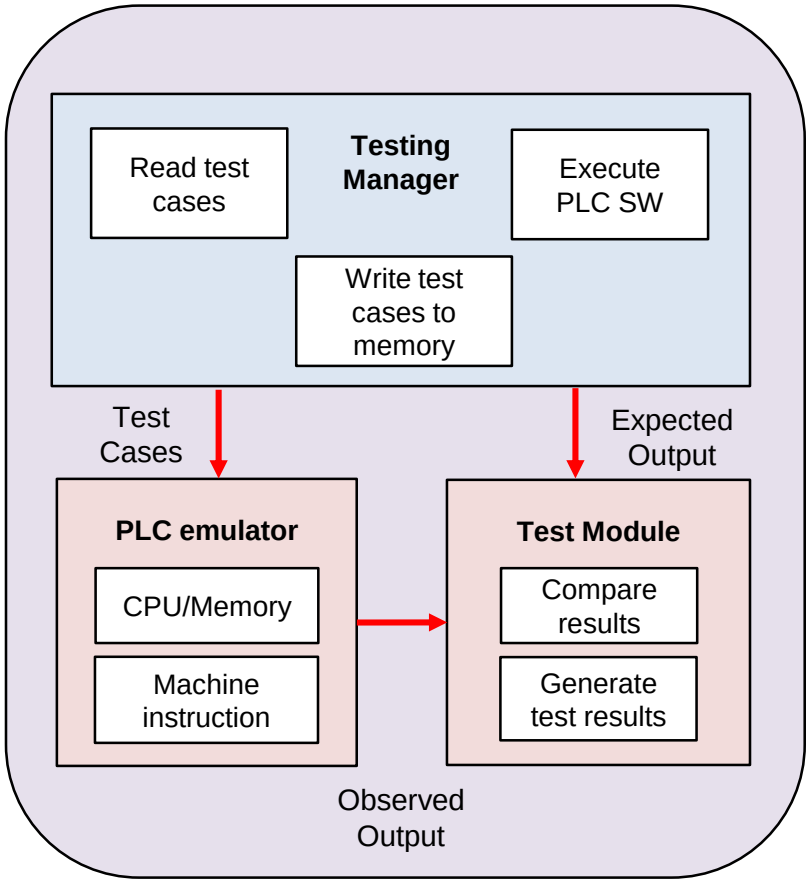
Test configuration of loop-operating control system (LOCS)
(10000 tests → no software failure, 45 delayed/ 16 early trip cases)



Test configuration of application software of DRPS Prototype
(7000 tests for SGL-low trip software → no software failure)

Hardware-based software test environment

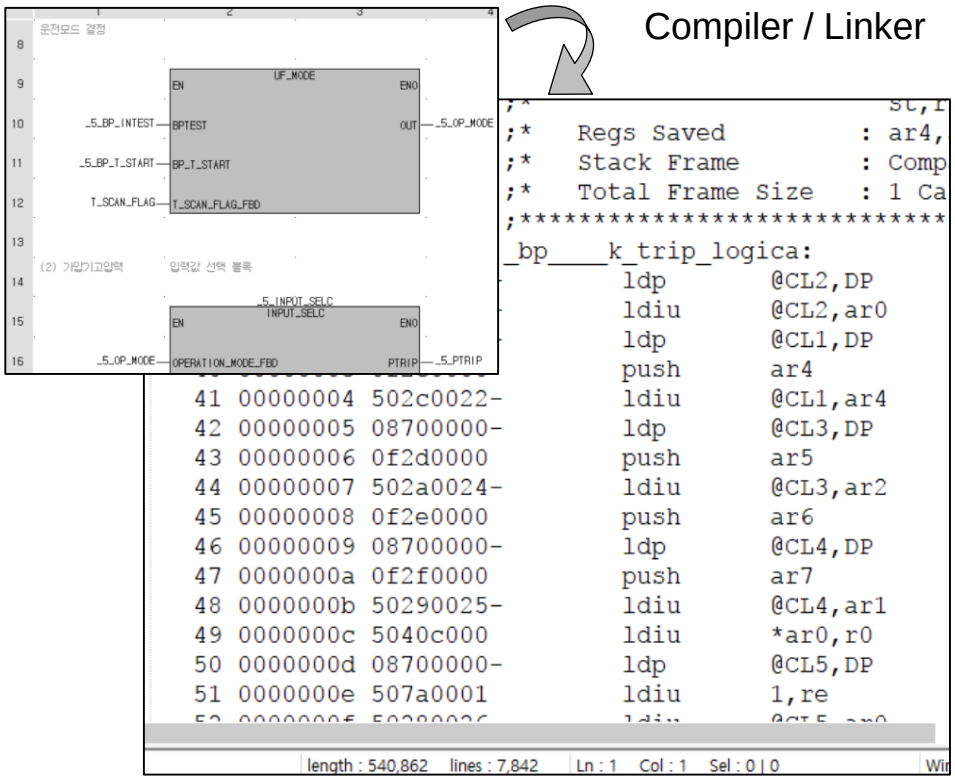
Time per test case : ~ few milliseconds (Input-space-based)



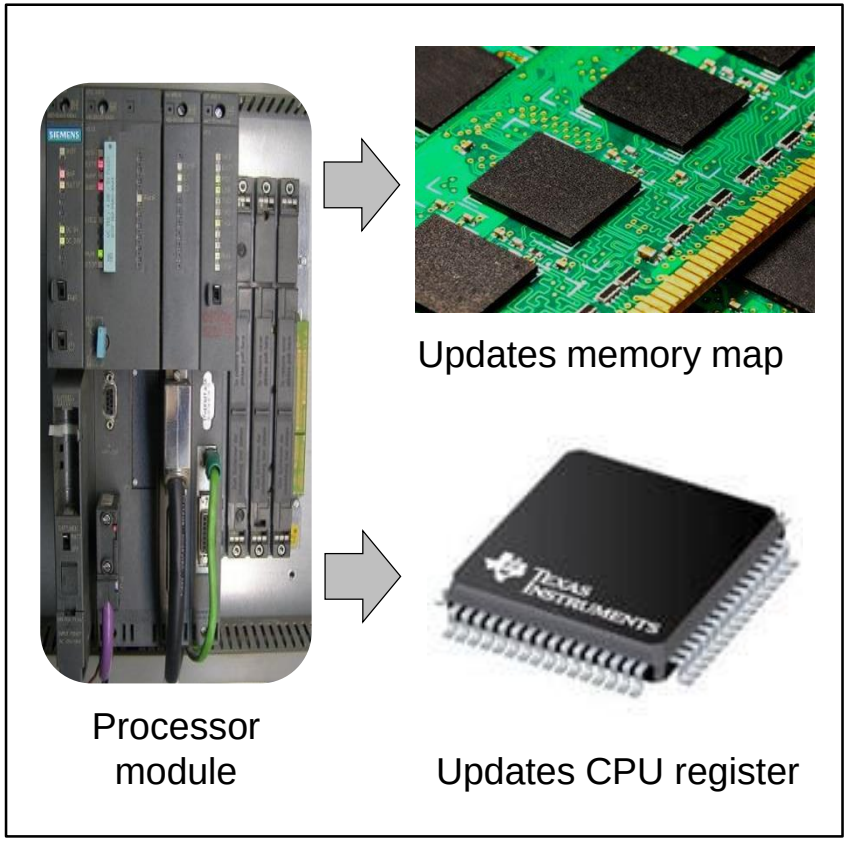
Simulation-based software test environment

Basic Operation of NPP Safety PLC

- At each line of SW program, the processor module updates the PLC status (e.g., CPU register, memory map) based on the execution result.



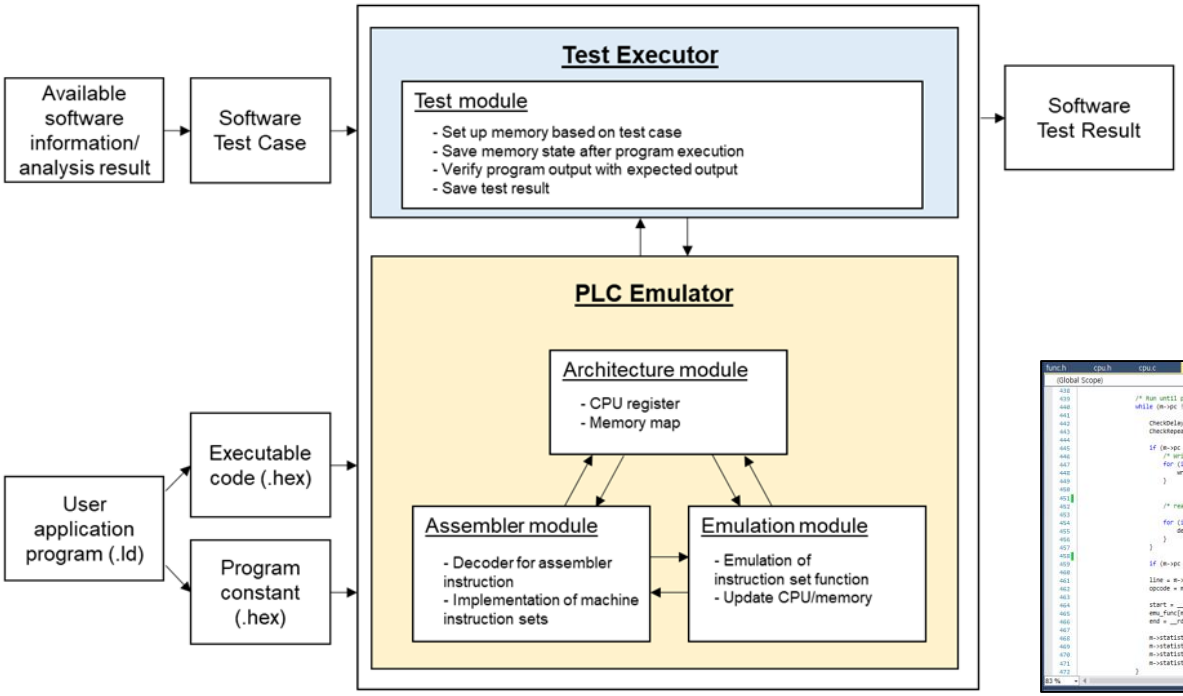
Compiled machine code of PLC program



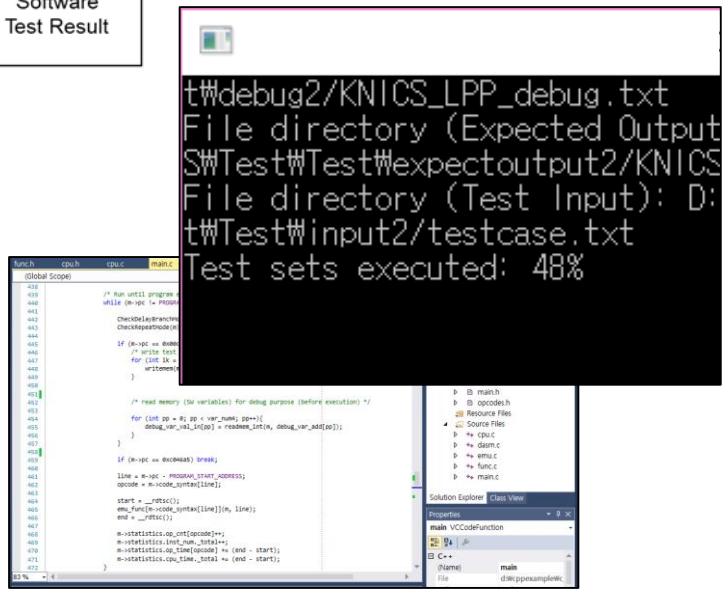
Basic PLC operation

Simulation-based SW Test-bed*

- **Architecture module** : Emulation of CPU registers, memory map
- **Assembler module** : Decoding compiled machine codes of SW program
- **Emulation module** : Emulation of PLC instruction set operation
- **Test module** : Implementation of test procedure



Screenshot of test-bed execution

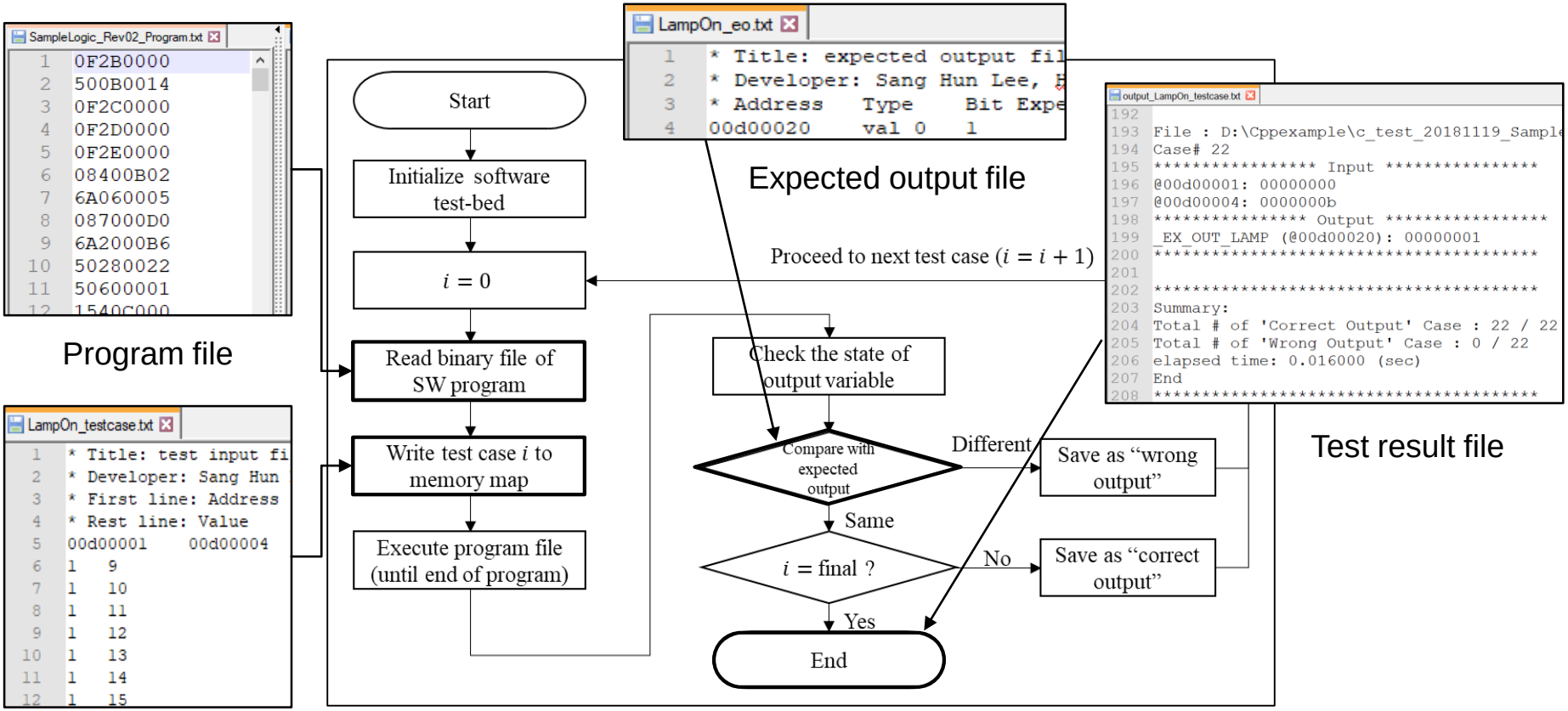


An overview of the simulation-based test-bed for safety-critical PLC software testing

* S. H. Lee, S. J. Lee, J. Park, E. C. Lee, and H. G. Kang, “Development of simulation-based testing environment for safety-critical software,” *Nuclear Engineering and Technology*, vol. 50, no. 4, pp. 570-581, 2018.

Testing Procedure for PLC SW

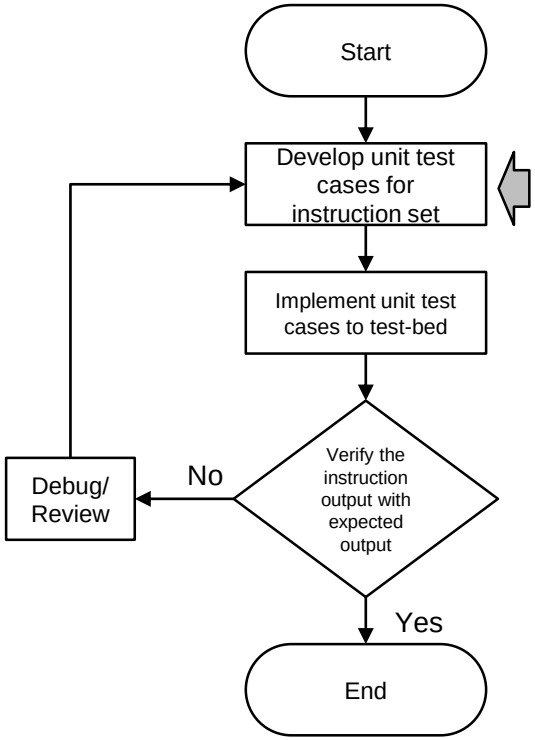
- 1) Loading program/constant files of PLC software
- 2) Read test case file and write cases to memory maps
- 3) Check whether correct output was generated after program execution



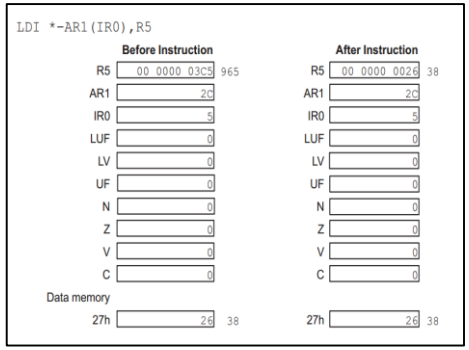
Software testing procedure using test-bed with derived test cases

Verification of Test-bed

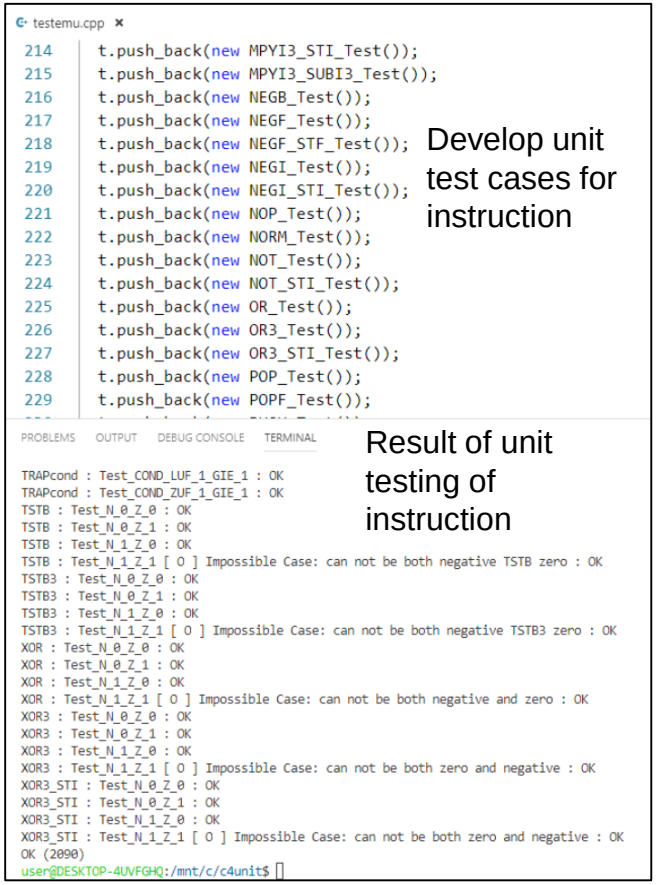
- **Unit testing** : verify the correctness of emulated instruction set
- **Functional testing** : verify the overall functionality of test-bed with benchmark software



Process of the test-bed verification using unit test cases



Example of unit test case developed for LDI (load integer) instruction

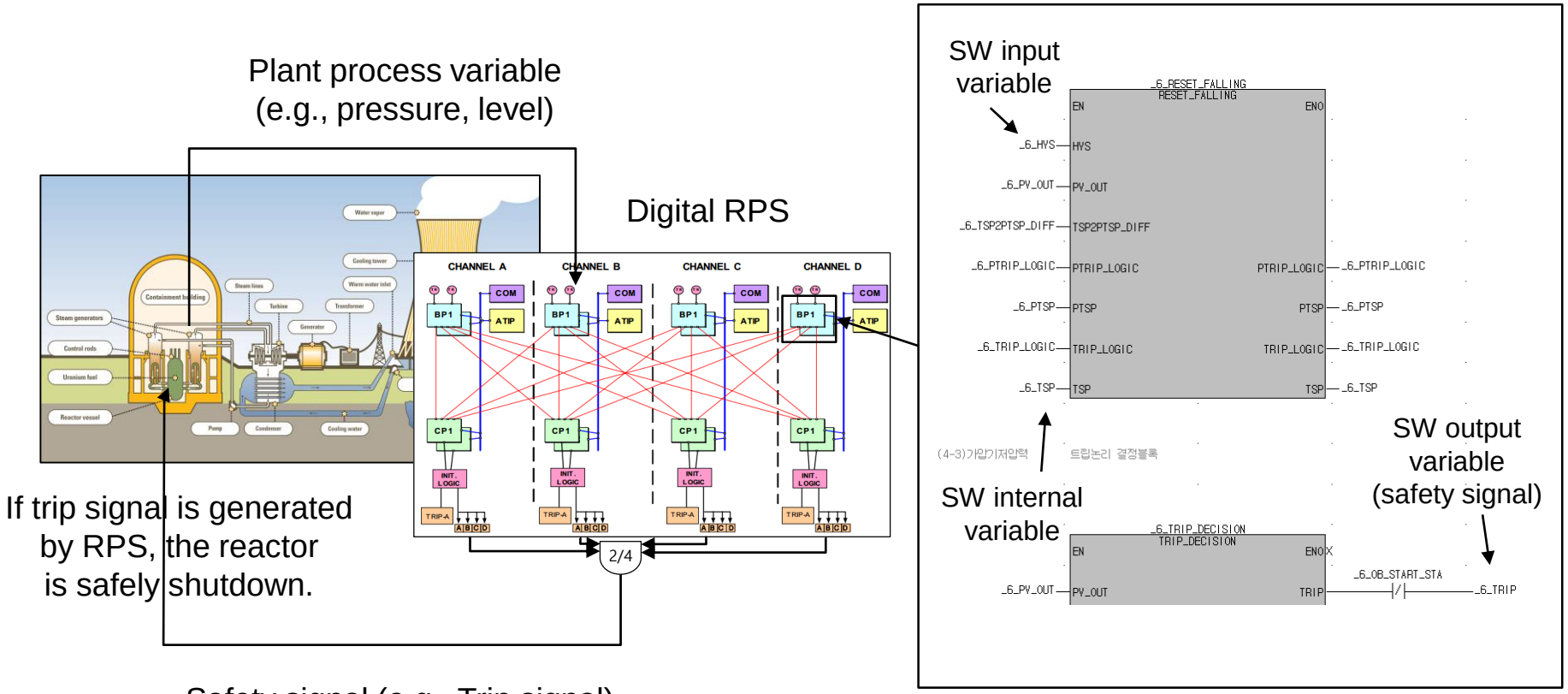


Result of instruction set testing for test-bed

Definition of Exhaustive Test Case

- **Software failure**: fail to generate safety signal when demand comes.
  - Demand: a plant condition that requires actuation of safety systems

- **Exhaustive test case** : all possible sets of the SW input/internal variables that generates safety output (e.g. trip signal).



Theory of Test Case Generation

- **Generating Test Cases → Satisfiability Problem**
  - Satisfiability : A formula is *satisfiable* if there is a model that makes formula *true*.

- Examples of Satisfiability Problems
  - Boolean Satisfiability Problem (SAT) : Propositional formula
  - Satisfiability Modulo theory (SMT) : First order logic formula

$$f_1 = x_1 \wedge x_2 \qquad f_2 = x_1 \wedge \neg x_1$$

$$f_3 = (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

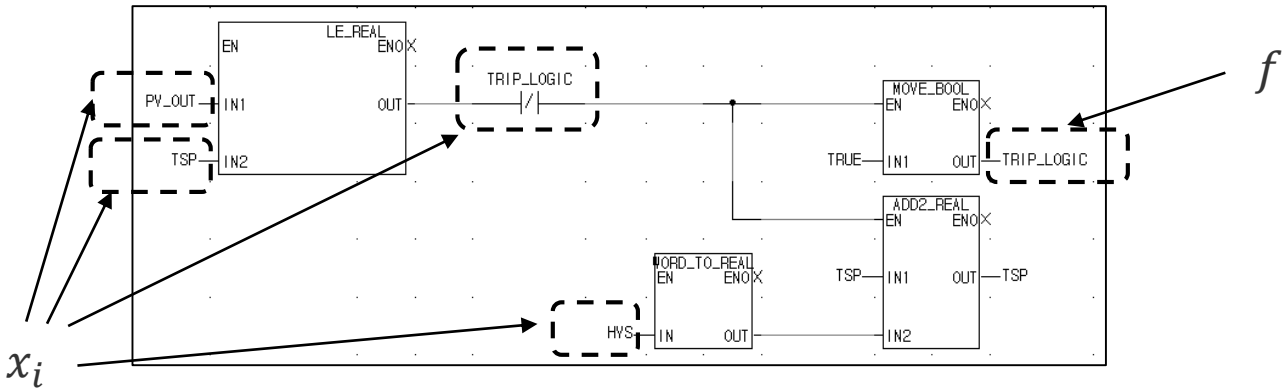
$$f_4 = (x_1 + 5 = x_2) \wedge (x_2 \leq x_3) \wedge (x_3 \leq 8)$$

  - Is the formula (f_1, f_2, f_3, f_4) *satisfiable*?
  - If the formula is *satisfiable*, do we have x_i's that make formula true?

Formulation of Test Case Generation Problem

- **<u>Objective</u> :** Derive all possible software input/internal variables (x_i's) such that the software output ($f(x_i)$) becomes *true*.

$$\underset{x_i \in S}{\boldsymbol{arg}}\ \boldsymbol{f(x_i) = true}$$

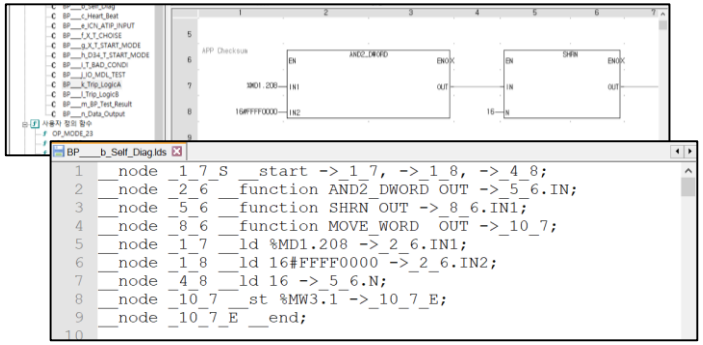


An example FBD program for calculating *TRIP_LOGIC*

- **<u>Formula</u>** (f) : Software output
- **<u>Literals</u>** (x_i) : Software input / internal variables
- **<u>Space</u>** (S) : Domain of all possible software input / internal variables that may encounter during software operation

Exhaustive Test Case Generation Framework

- 1) Translating the SW program to semantically-equivalent SMT formula
- 2) Generating exhaustive test cases by finding the models for the literals of the formula by checking the satisfiability given desired software output.



NPP safety software in FBD language

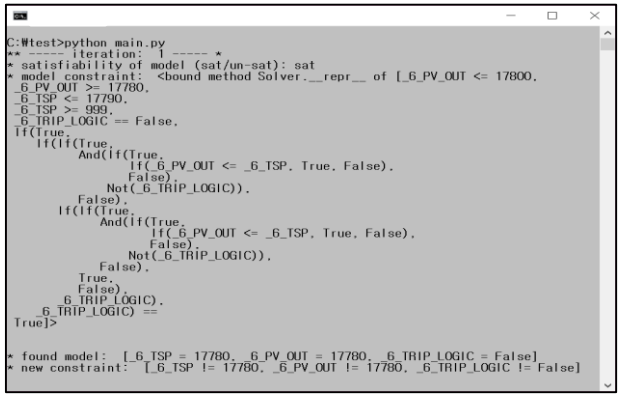
```
# rung 0 #

AND2_DWORD0_in_IN1 = __MD1_208__in

AND2_DWORD0_in_IN2 = BitVecVal(int("FFFF0000",
 16), 32)

AND2_DWORD0_in_EN = True
AND2_DWORD0_out_ENO, AND2_DWORD0_out_OUT =
AND2_DWORD(AND2_DWORD0_in_EN,
AND2_DWORD0_in_IN1, AND2_DWORD0_in_IN2,
AND2_DWORD0_out_ENO, AND2_DWORD0_out_OUT)
```

Translated SMT formula of software



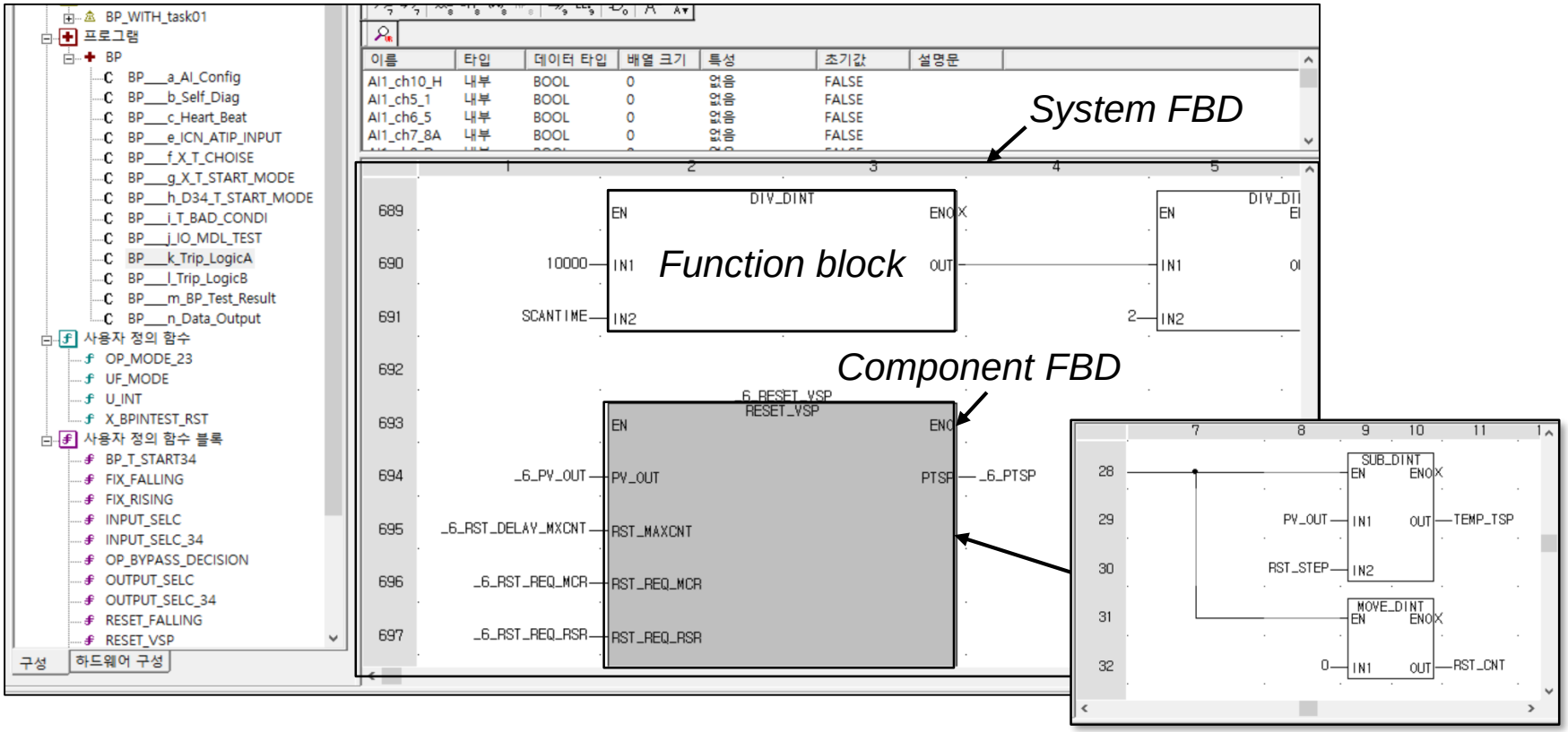
Exhaustive test case generation algorithm

1	_6_TRIP_LOGIC	False	_6_TSP	17780	_6_PV_OUT	17780
2	_6_TRIP_LOGIC	False	_6_TSP	17781	_6_PV_OUT	17781
3	_6_TRIP_LOGIC	False	_6_TSP	17782	_6_PV_OUT	17782
4	_6_TRIP_LOGIC	False	_6_TSP	17783	_6_PV_OUT	17783
5	_6_TRIP_LOGIC	False	_6_TSP	17784	_6_PV_OUT	17784
6	_6_TRIP_LOGIC	False	_6_TSP	17785	_6_PV_OUT	17785
7	_6_TRIP_LOGIC	False	_6_TSP	17786	_6_PV_OUT	17786
8	_6_TRIP_LOGIC	False	_6_TSP	17787	_6_PV_OUT	17787
9	_6_TRIP_LOGIC	False	_6_TSP	17788	_6_PV_OUT	17788
10	_6_TRIP_LOGIC	False	_6_TSP	17789	_6_PV_OUT	17789
11	_6_TRIP_LOGIC	False	_6_TSP	17790	_6_PV_OUT	17790
12	_6_TRIP_LOGIC	False	_6_TSP	17789	_6_PV_OUT	17782
13	_6_TRIP_LOGIC	False	_6_TSP	17789	_6_PV_OUT	17785
14	_6_TRIP_LOGIC	False	_6_TSP	17788	_6_PV_OUT	17785
15	_6_TRIP_LOGIC	False	_6_TSP	17787	_6_PV_OUT	17785
16	_6_TRIP_LOGIC	False	_6_TSP	17787	_6_PV_OUT	17786
17	_6_TRIP_LOGIC	False	_6_TSP	17785	_6_PV_OUT	17784
18	_6_TRIP_LOGIC	False	_6_TSP	17783	_6_PV_OUT	17780
19	_6_TRIP_LOGIC	False	_6_TSP	17783	_6_PV_OUT	17781

Exhaustive test cases for safety output

NPP safety software - FBD program

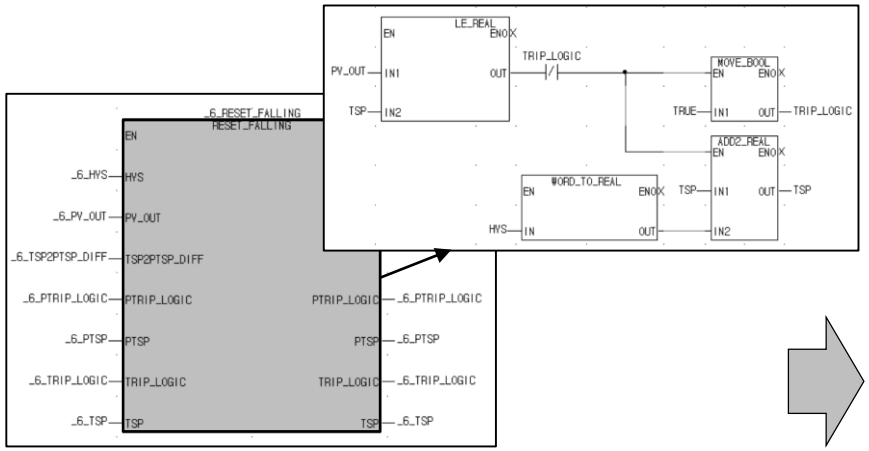
- **Function Block** : a basic unit having input/internal/output variables
- **Component FBD** : composed of FBs interconnected according to execution order
- **System FBD** : composed of component FBDs (a whole software)



An example of PLC software

Translation of FBD programs

- FBD-to-SMT translation rules are developed based on the formal definition.
- The translation generates equivalent SMT formulas for all components.



RESET_FALLING.lds

```
__node _1_3  __ld PV_OUT -> _2_2.IN1;
__node _1_4  __ld TSP -> _2_2.IN2;
__node _5_3  __ldc TRIP_LOGIC -> _8_3;
__node _9_4  __ld __TRUE -> _10_3.IN1;
__node _12_4  __st TRIP_LOGIC -> _12_4_E;
__node _9_6  __ld TSP -> _10_5.IN1;
__node _12_6  __st TSP -> _12_4_E;
__node _6_7  __ld HYS -> _7_6.IN;
__node _12_4_E  __end;
```

Source code of program

1) Definition of elements

RESET_FALLING.py

```
def RESET_FALLING(EN, in_HYS, in_PV_OUT, in_TSP2PTSP_DIFF, in_PTRIP_LOG
    ## ----- function operation ----- ##
    # rung 0 #
    LE_REAL0_IN1 = in_PV_OUT
    LE_REAL0_IN2 = in_TSP
    LE_REAL0_EN = True
    ENO__1__RESET_FALLING, OUT__1__RESET_FALLING = LE_REAL(LE_REAL0_EN,
    LE_REAL0_OUT = OUT__1__RESET_FALLING
    temp1 = And(LE_REAL0_OUT, Not(in_TRIP_LOGIC))
    MOVE_BOOL1_EN = temp1
    ADD2_REAL2_EN = temp1
    WORD_TO_REAL3_IN = in_HYS
    WORD_TO_REAL3_EN = True
    ENO__4__RESET_FALLING, OUT__4__RESET_FALLING = WORD_TO_REAL(WORD_TO
    WORD_TO_REAL3_OUT = OUT__4__RESET_FALLING
    ADD2_REAL2_IN2 = WORD_TO_REAL3_OUT
    return ENO1, PTRIP_LOGIC__5, PTSP__2, TRIP_LOGIC__5, TSP__3
```

2) Definition of operation

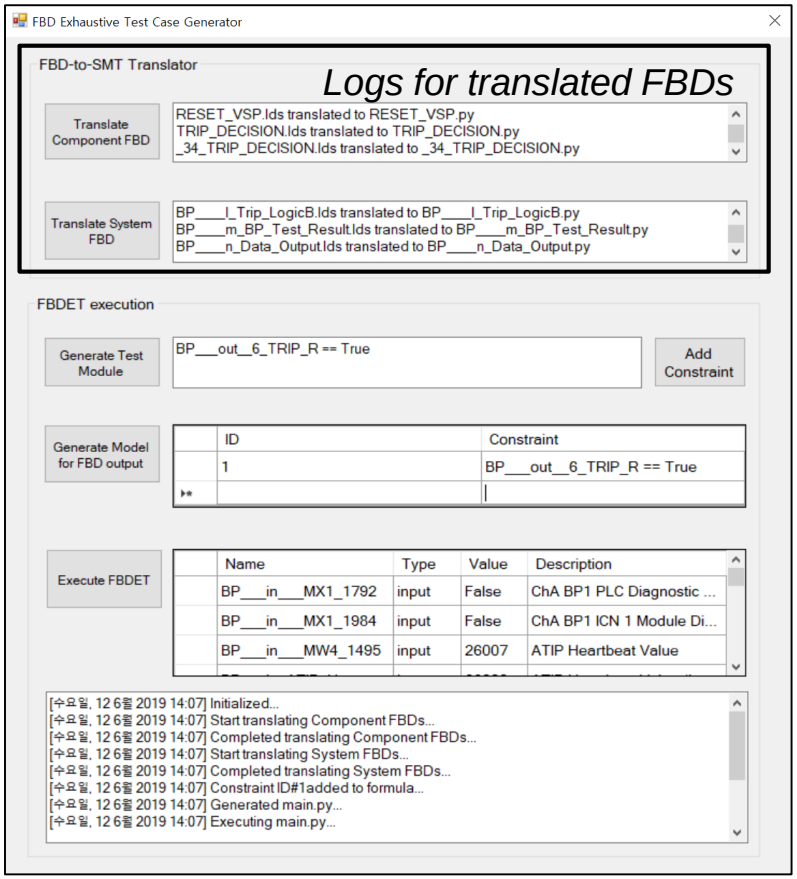
3) Return output

Translated SMT formula for program

An example of component FBDs translation

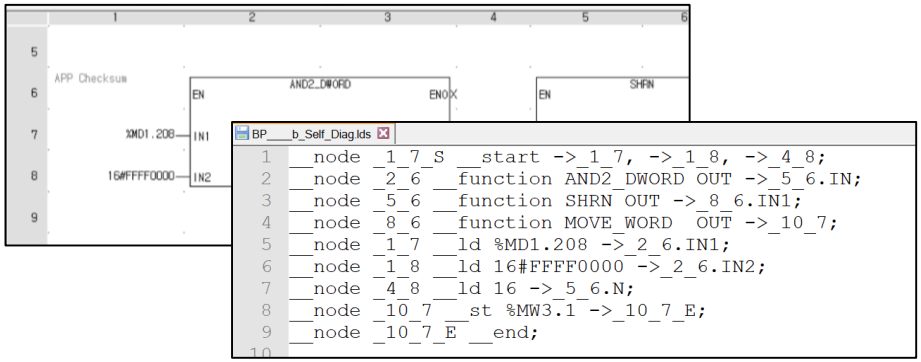
Translation of FBD programs

- Exhaustive test case generator automatically translates the source code to SMT formula based on the developed translation rules.



Screenshot of exhaustive test case generator

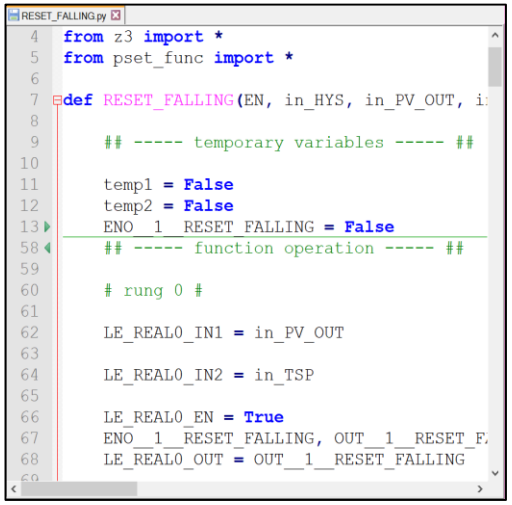
Source code of PLC program

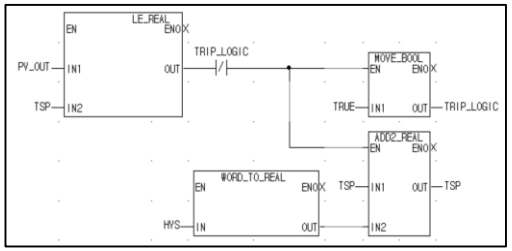


Translated SMT formula for the example program

Single Test Case Generation

Translated FBD program





1) Define Program under Test

1-1. Define variables

1-2. Define program

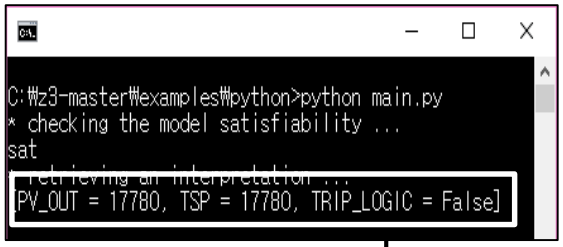
2) Define Test Requirements

2-1. Define desired software output (e.g., *TRIP_LOGIC* = true)

3) Solve for Test Requirements

3-1. Check satisfiability (→ satisfiable/unsatisfiable)

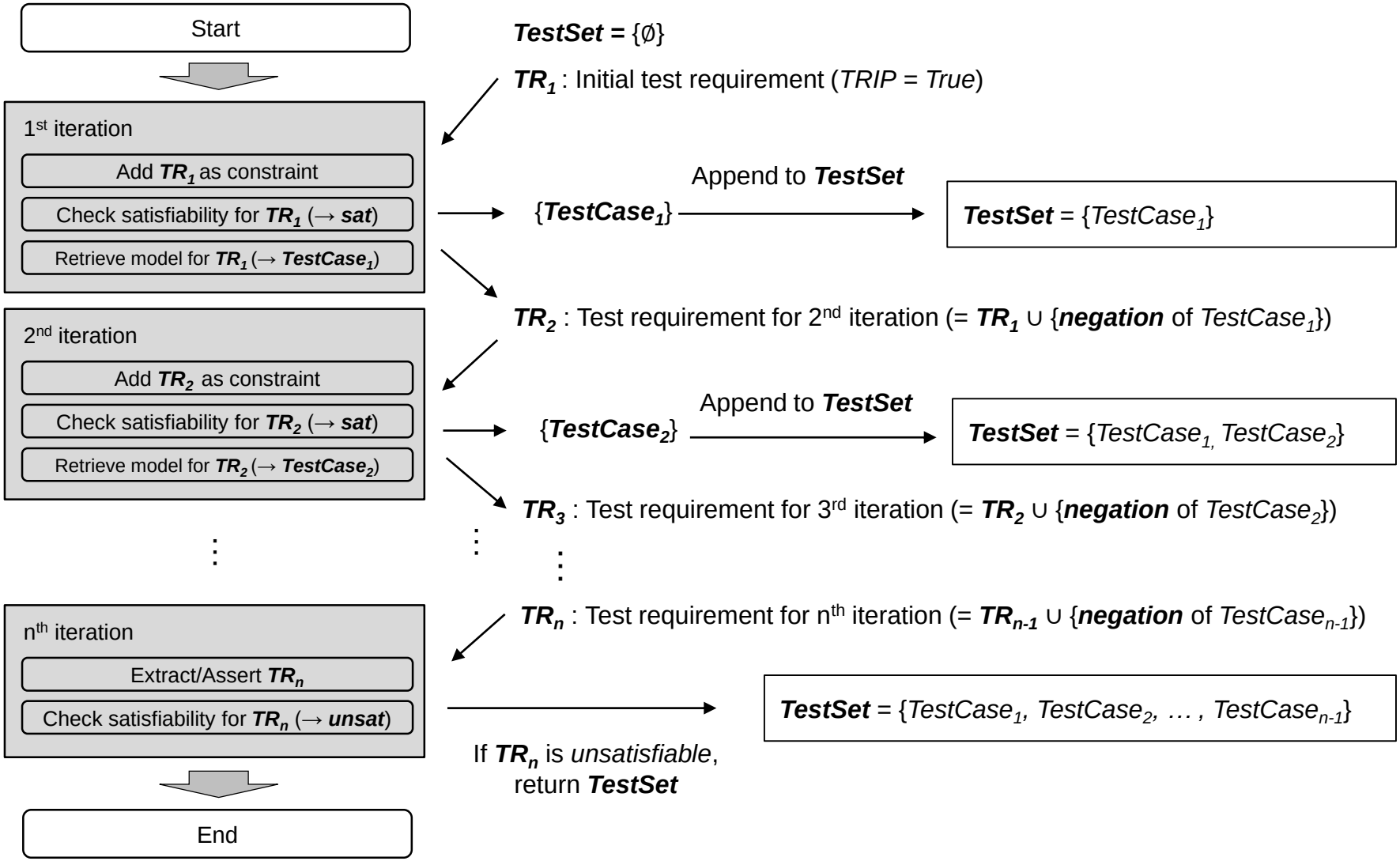
3-2. Retrieve model for the program variables (→ test case)



Generated test case

Process of single test case generation for PLC program

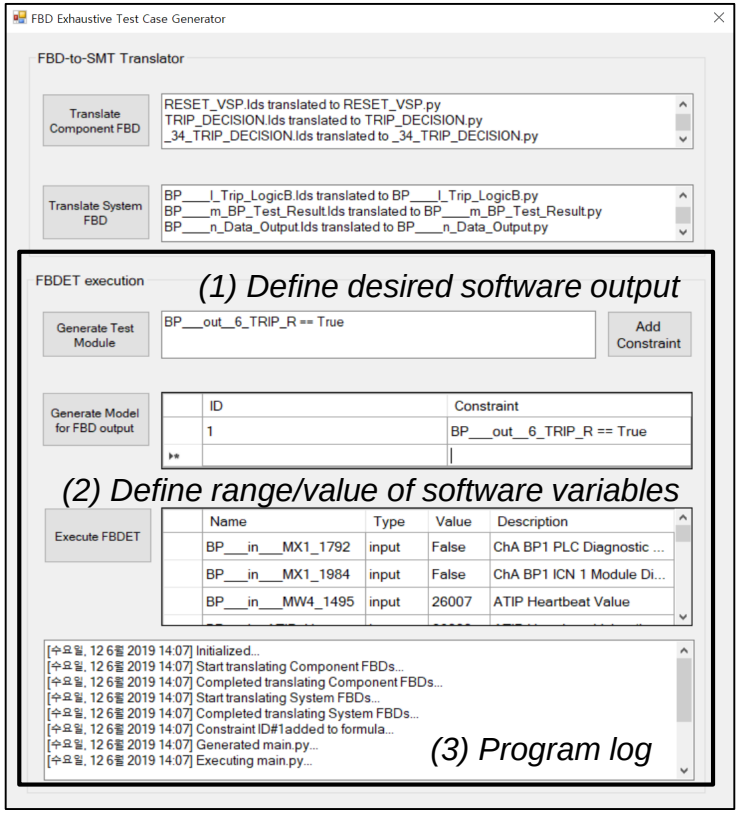
Exhaustive Test Case Generation Algorithm



Process of test case generation for FBD program

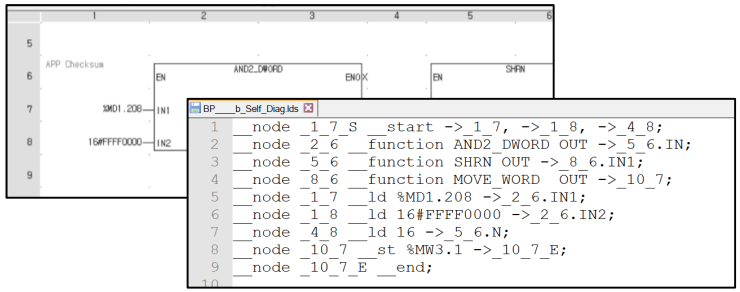
Exhaustive Test Case Generation Algorithm**

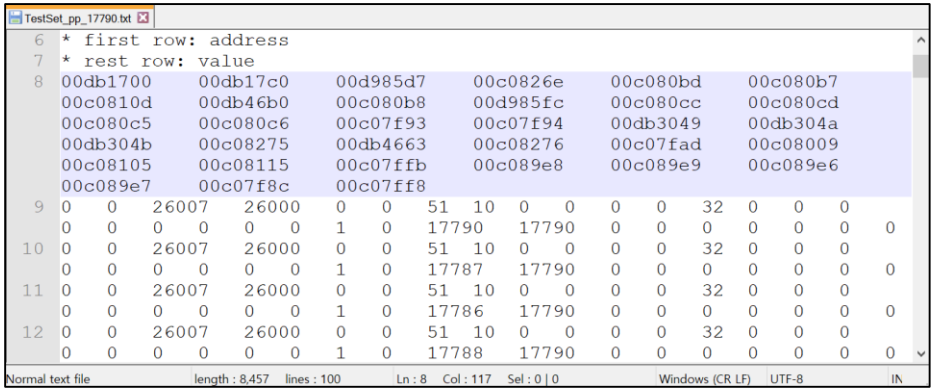
- The algorithm is implemented to exhaustive test case generator.
- It automatically generates exhaustive test cases for PLC program given the defined software output.



Screenshot of exhaustive test case generator

Source code of program



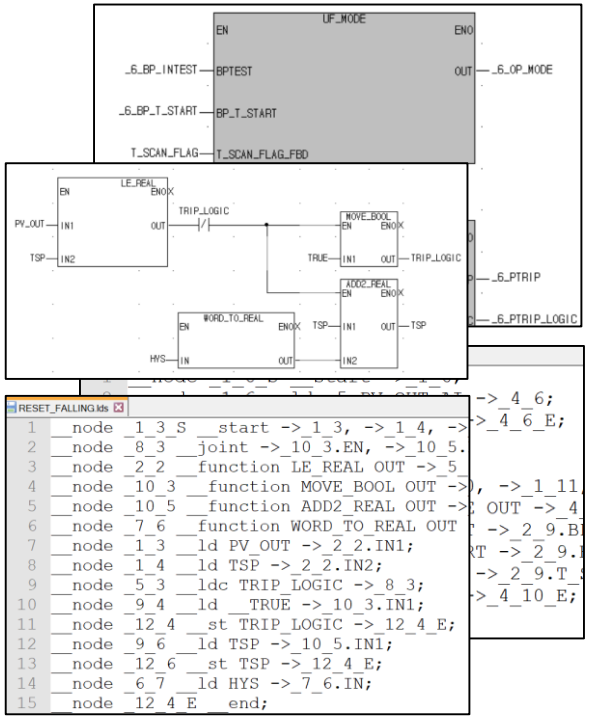


Generated test case file for Rx. trip signal output

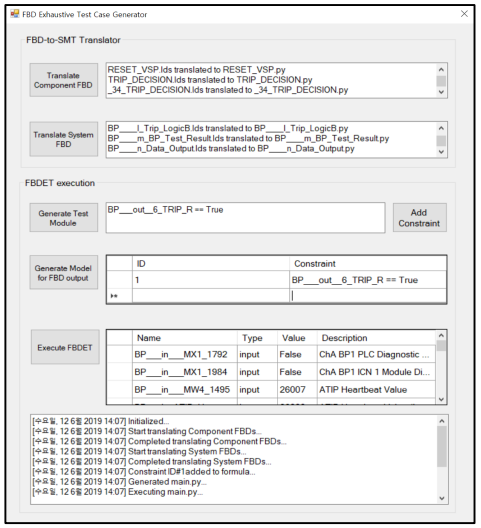
** S. H. Lee, S. J. Lee, S. M. Shin, E.-C. Lee, H. G. Kang, "Exhaustive Testing of Safety-critical Software for Reactor Protection System," *Reliability Engineering & System Safety*, vol. 193, 106667, Jan, 2020.

Important Properties of the Framework

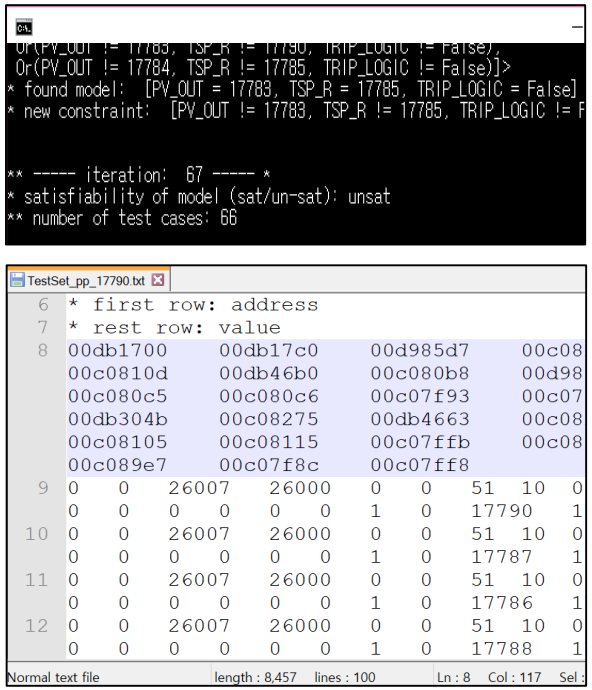
- The algorithm is general to any NPP safety software.
- Each test case is unique to each other (***exclusive***)
- Generated test cases represent all possible combination (***exhaustive***) of software input/internal variables that makes software output to be true.



PLC program



FBD-to-SMT translation
+ Exhaustive test case
generation algorithm

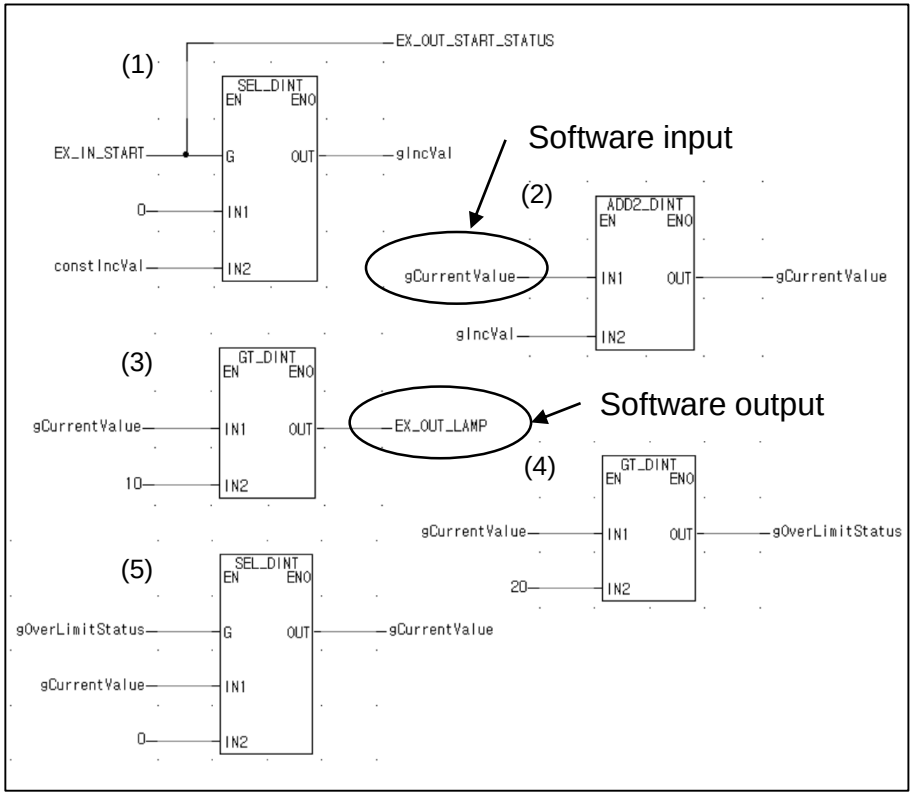


Generated test cases

Exhaustive test case generation framework for NPP safety software

Application - Lamp On/Off Software

- Software output (*EX_OUT_LAMP*) becomes *true* when software input (*gCurrentValue*) is higher than the setpoint (= 10).



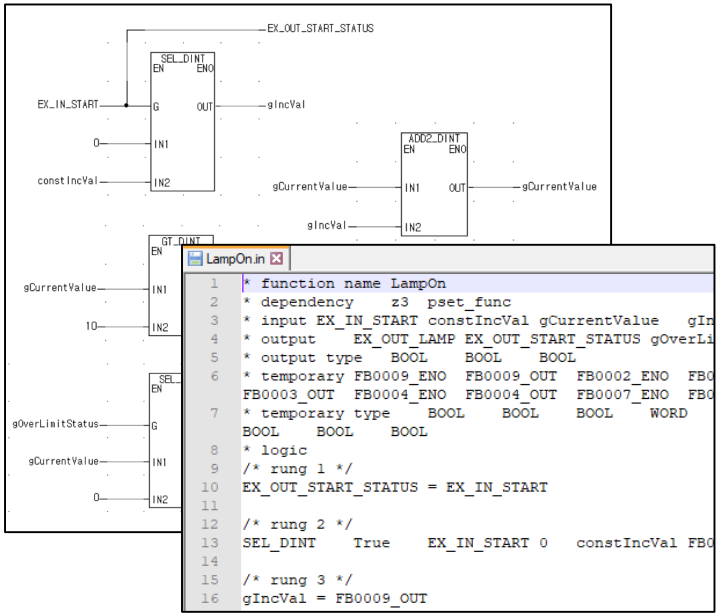
A FBD program for Lamp On/Off function
(written and compiled with pSET)

Input/Internal/Output variables
of Lamp On/Off program

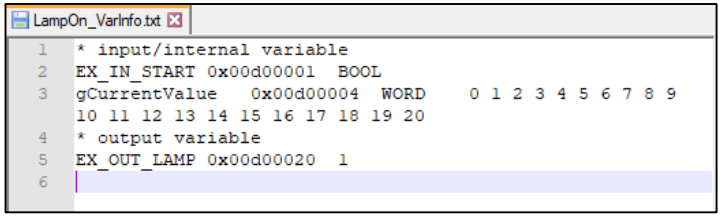
Name	Port	Type	Description
EX_IN_START	Input	BOOL	start signal
gCurrentValue	Input	DINT	increased by constIncVal (=2) when EX_IN_START is on, and set to 0 when the value exceeds 20
constIncVal	Internal	DINT	constant increment value (= 2)
gIncVal	Internal	DINT	gIncVal set to constIncVal when EX_INT_START is set as true; else 0
EX_OUT_LAMP	Output	BOOL	Lamp on when gCurrentValue exceeds 10
EX_OUT_ START_STATUS	Output	BOOL	EX_IN_START status
gOverLimitStatus	Output	BOOL	set to true when gCurrentValue exceeds 20

Test Case for ‘Lamp On’ Scenario

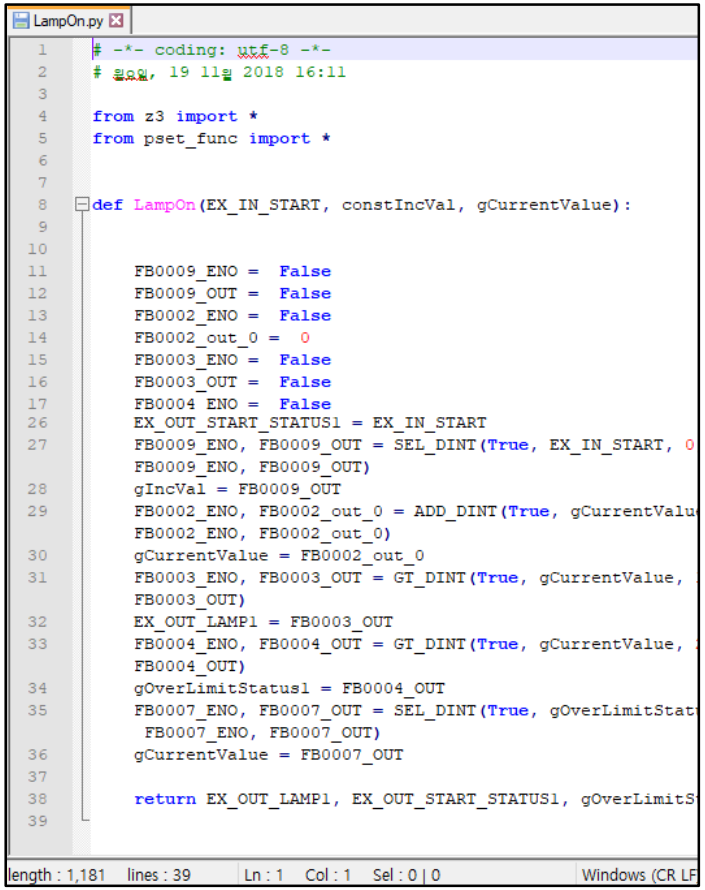
- SW output : *EX_OUT_LAMP* (= true)
- SW input/internal variables: *gCurrentValue*, *EX_IN_START*



Source code (from developer)



Variable information



Translated SMT formula

Test Case for ‘Lamp On’ Scenario

- SW output : *EX_OUT_LAMP (*= true)
- SW input/internal variables: *gCurrentValue*, *EX_IN_START*

```
** ----- iteration:  22 ----- *
* satisfiability of model (sat/un-sat): sat
* model constraint:  <bound method Solver.__repr__ of [gCurrentV
alue >= 0,
 gCurrentValue <= 20,
 If(True,
    If(If(True,
          gCurrentValue + If(True, If(EX_IN_START, 2, 0), 0),
          0) >
       10,
       True,
       False),
    False) ==
 True,
 Or(EX_IN_START != True, gCurrentValue != 9),
 Or(EX_IN_START != False, gCurrentValue != 11),
 Or(EX_IN_START != False, gCurrentValue != 12),
 Or(EX_IN_START != False, gCurrentValue != 13),
 Or(EX_IN_START != False, gCurrentValue != 14),
 Or(EX_IN_START != False, gCurrentValue != 15),
 Or(EX_IN_START != False, gCurrentValue != 16),
 Or(EX_IN_START != False, gCurrentValue != 17),
 Or(EX_IN_START != False, gCurrentValue != 18),
 Or(EX_IN_START != False, gCurrentValue != 19),
 Or(EX_IN_START != False, gCurrentValue != 20),
 Or(EX_IN_START != True, gCurrentValue != 19),
 Or(EX_IN_START != True, gCurrentValue != 18),
 Or(EX_IN_START != True, gCurrentValue != 17),
 Or(EX_IN_START != True, gCurrentValue != 16),
 Or(EX_IN_START != True, gCurrentValue != 15),
 Or(EX_IN_START != True, gCurrentValue != 14),
 Or(EX_IN_START != True, gCurrentValue != 13),
 Or(EX_IN_START != True, gCurrentValue != 12),
 Or(EX_IN_START != True, gCurrentValue != 11),
 Or(EX_IN_START != True, gCurrentValue != 10)]>
* found model:  [EX_IN_START = True, gCurrentValue = 20]
* new constraint:  [EX_IN_START != True, gCurrentValue != 20]


** ----- iteration:  23 ----- *
* satisfiability of model (sat/un-sat): unsat
** number of test cases: 22

D:₩Cppexample₩₩WindowsFormsApplication1_test_20181119₩₩WindowsForm
sApplication1_test₩bin₩Debug₩resultfiles>₩
```

Screenshot of the exhaustive test case generation algorithm for Lamp On/Off program

LampOn_testcase.txt

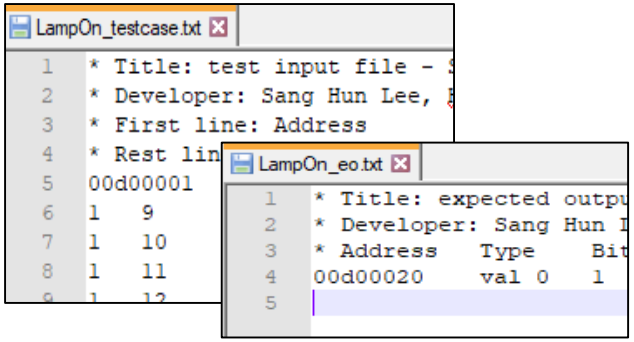
```
* Title: test input file
* Developer: Sang Hun Lee
* First line: Address
* Rest line: Value
00d00001    00d00004
1   9
1   10
1   11
1   12
1   13
1   14
1   15
1   16
1   17
1   18
1   19
1   20
0   16
0   17
0   18
0   19
0   20
0   15
0   14
0   13
0   12
0   11
```

Ln : 28 Col : 1 Sel Unix (LF) UT

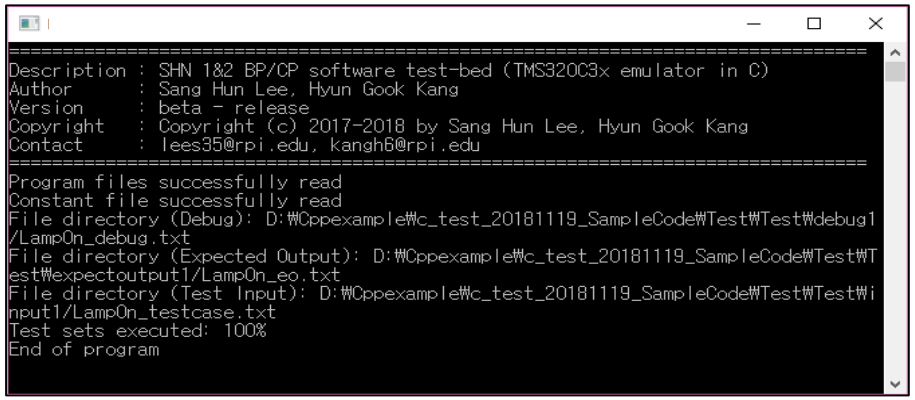
Exhaustive test cases generated for ‘Lamp On’ scenario (22 cases)

Testing of Lamp On/Off Software

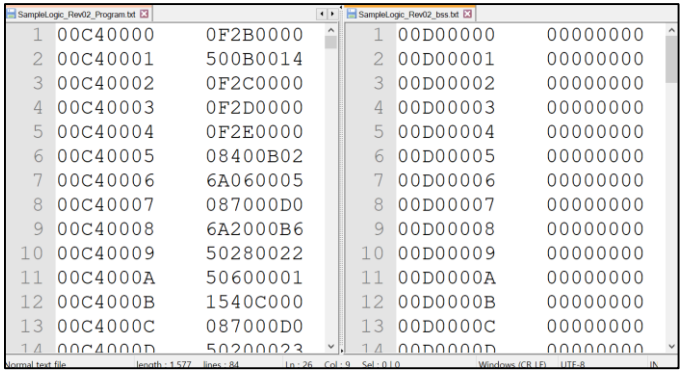
- The test result implies software is error-free for ‘Lamp On’ scenario.
- The proposed framework can be applied for complex NPP safety logic in a similar manner.



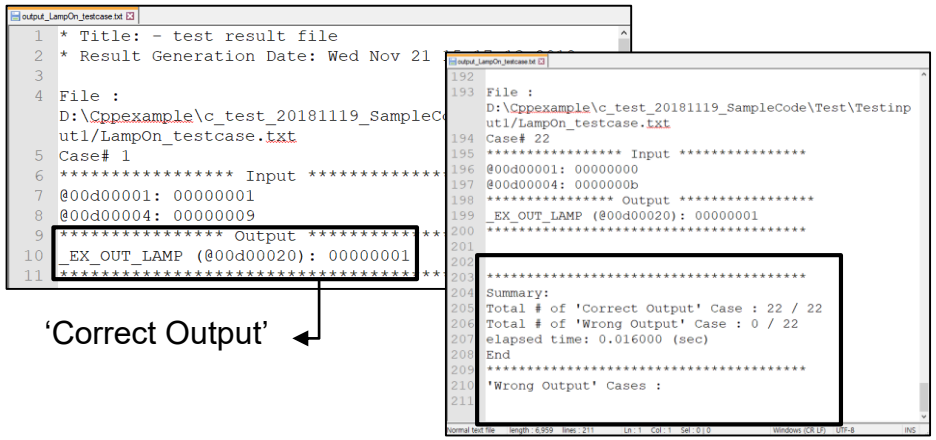
Test case & Expected output files



Software test-bed execution



Compiled machine code of
Lamp On/Off program (from developer)

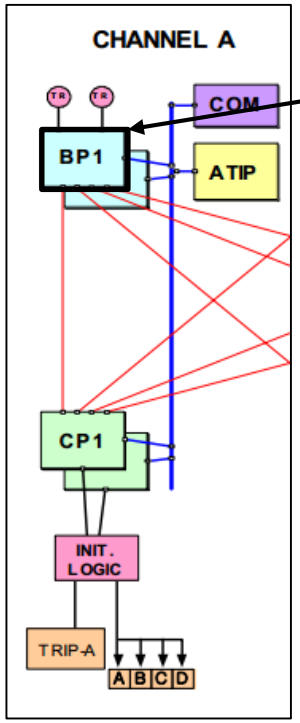


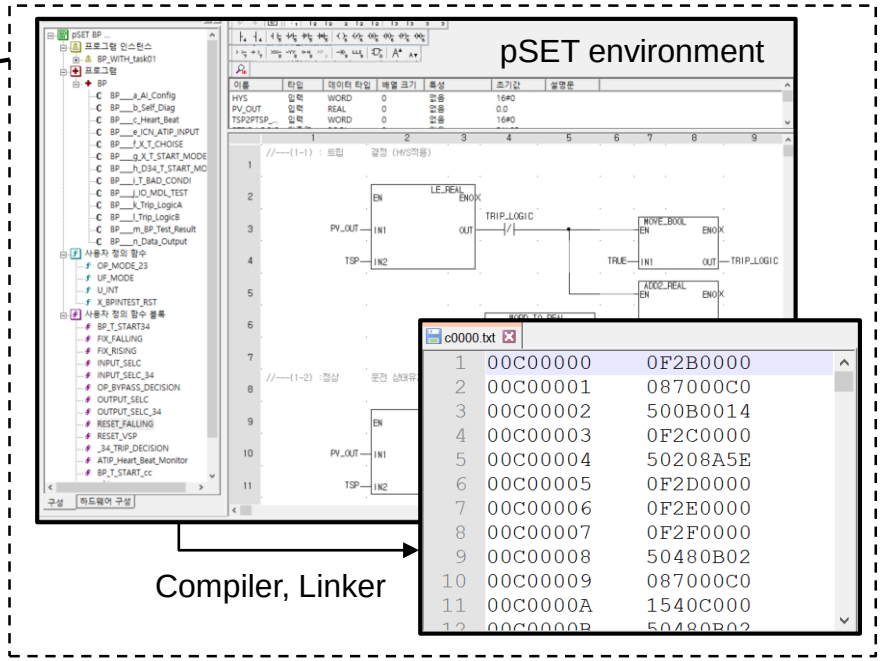
‘Correct Output’

Test Result file

Application - RPS Trip Logic Software

- IDiPS-RPS was developed in Korea to satisfy the design requirements of APR-1400 which has full-digital I&C systems.
- BP software has 15 trip logic modules which generate Rx. trip signal when the process parameter deviates from permissible operating condition.





Architecture of one channel IDiPS-RPS and prototype

Source code and compiled machine code of BP trip logic software

Test Case for Target Trip Logic

- Target Trip Logic: Pressurizer-pressure low trip logic
- Target Test Scope: Trip-initiation condition (SW output: '*_6_TRIP_R = true*')
- Target Scenario: All possible accidents

pset_func.py

```python
# -*- coding: utf-8 -*-

from z3 import *

def LE_REAL(EN, IN1, IN2, ENO, OUT):
    BD = If(IN1 <= IN2, True, False)
    ENO2 = If(EN, True, False)
    OUT2 = If(EN, BD, OUT)
    return ENO2, OUT2

def ADD_DINT(EN, IN1, IN2, ENO, OUT):
    BD = IN1 + IN2
    ENO2 = If(EN, True, False)
    OUT2 = If(EN, BD, OUT)
    return ENO2, OUT2
```

(a) Function Blocks

RESET_FALLING.py

```python
from z3 import *
from pset_func import *

def RESET_FALLING(EN, in_HYS, in_PV_OUT
    ## ----- temporary variables -----
    temp1 = False
    temp2 = False

    ## ----- function operation ----- #
    # rung 0 #
    LE_REAL0_IN1 = in_PV_OUT
    LE_REAL0_IN2 = in_TSP
    LE_REAL0_EN = True
    ENO__1__RESET_FALLING, OUT__1__RESE
    LE_REAL0_OUT = OUT__1__RESET_FALLIN
```

(b) Component FBDs

BP____k_Trip_LogicA.py

```python
from RESET_VSP import *
from RESET_FALLING import *
from OP_BYPASS_DECISION import *

def BP____k_Trip_LogicA(BP____k_Trip_Logic


    ## ----- temporary variables ----- ##

    temp1 = False

    RESET_FALLING50_out_ENO, RESET_FALLING
    _6_PTRIP_LOGIC__2 = If(RESET_FALLING50
    _6_PTSP__3 = If(RESET_FALLING50_out_EN
    _6_TRIP_LOGIC__2 = If(RESET_FALLING50_
```

(c) System FBD

Generated files from BP trip logic SW source code

Information on the translated FBDs

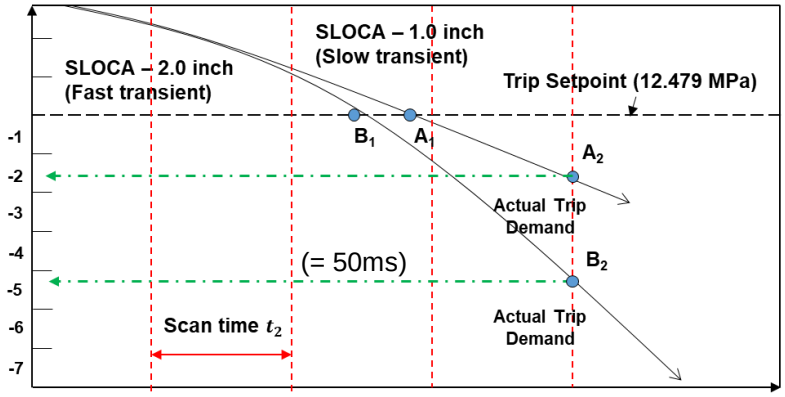
Category	Quantity
Function Blocks (FBs)	242
Component FBDs	122
System FBD	1
System input/internal variables	32 (/ 612)
System output variable	1
Temporary variables used*	2,661

List of software input/internal variables that contributes to trip signal generation

Variables	Description
_6_TSP_R	PZR-pressure-low trip setpoint
_6_PV_OUT_AI	PZR-pressure-low process parameter
AI_2_MDL_ERR	PLC analog input module error signal
_6_RST_REQ_MCR_DI	Operator trip setpoint reset request (from MCR)
⋮	⋮

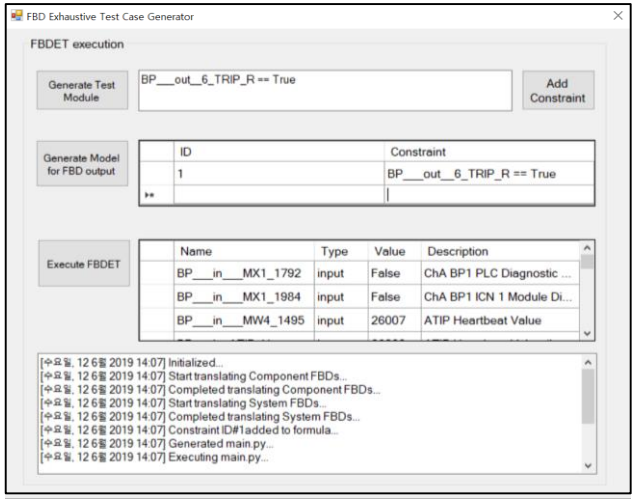
Test Case for Target Trip Logic

- The possible states/range of software input/internal variables are identified based on the available information (e.g. plant simulation, SRS/SDS).

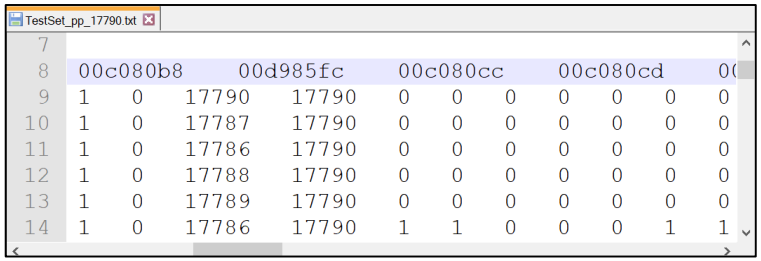


ID	Category	Hole diameter (inch)	D_{max} (count)
1	SLOCA	0.5 (0.0001267 m²)	1
2	SLOCA	1 (0.0005067 m²)	2
3	SLOCA	1.5 (0.001140 m²)	3
4	SLOCA	2.0 (0.002026 m²)	5
5	SGTR	0.75 x 2 (0.00057 m²)	1

Obtained profile of PZR pressure from the thermo-hydralic analysis of plant accidents



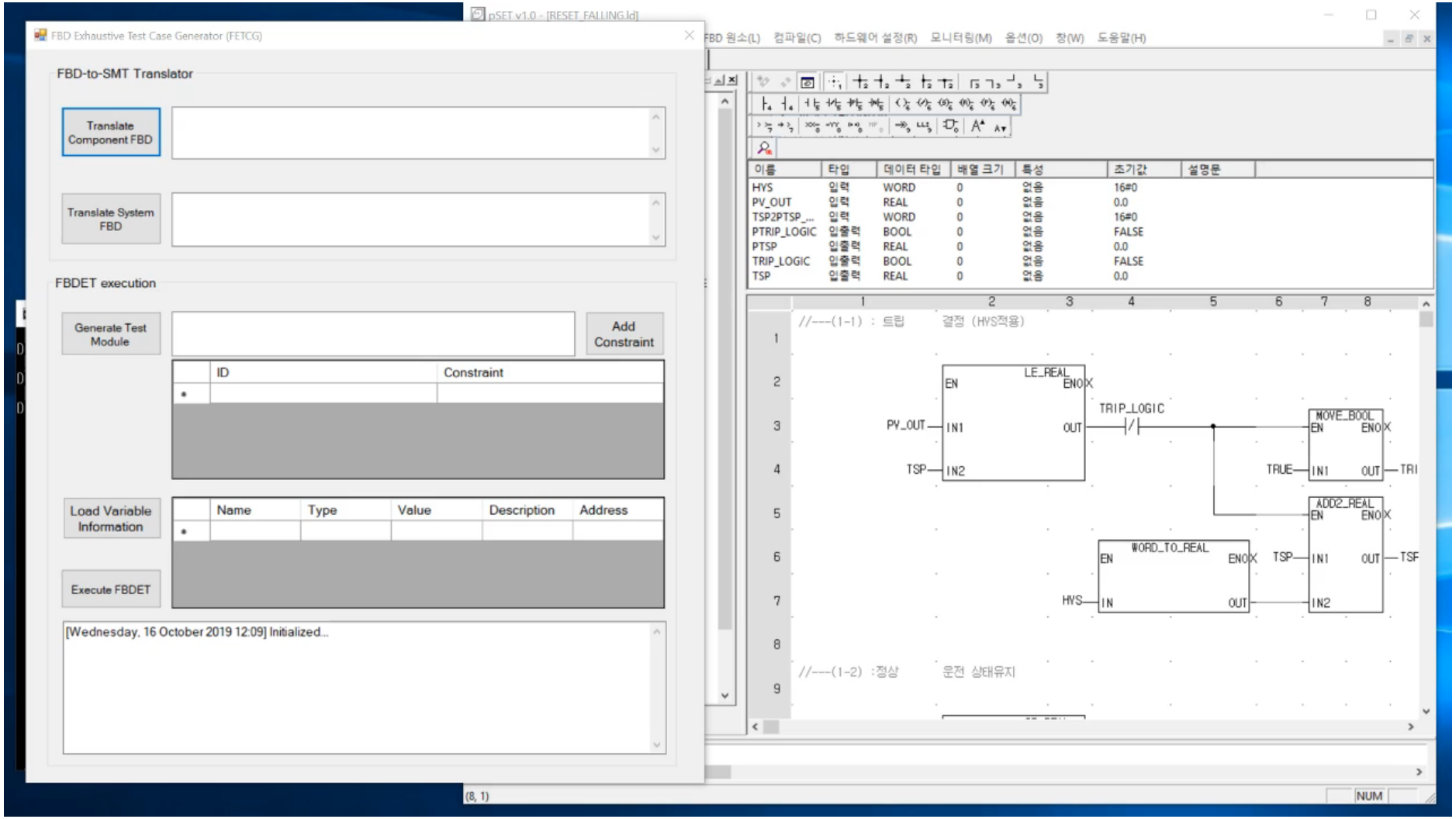
Definition of software variables' states



A part of generated exhaustive test cases

* D_{max} : maximum i[th] digital value below trip setpoint given POSAFE-Q ADC resolution

Demonstration of Test Case Generator



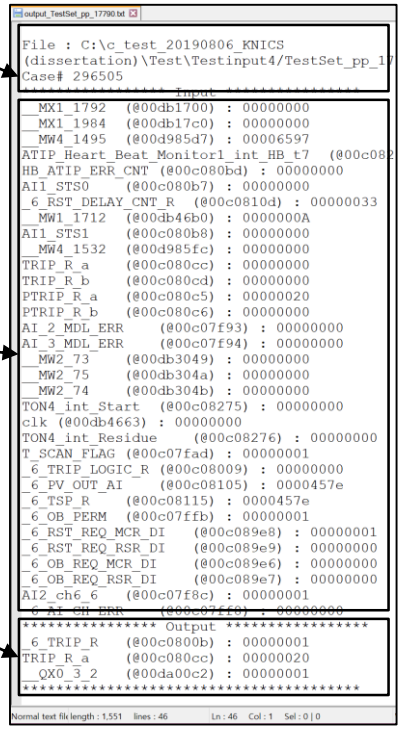
Testing of BP Trip Logic Software

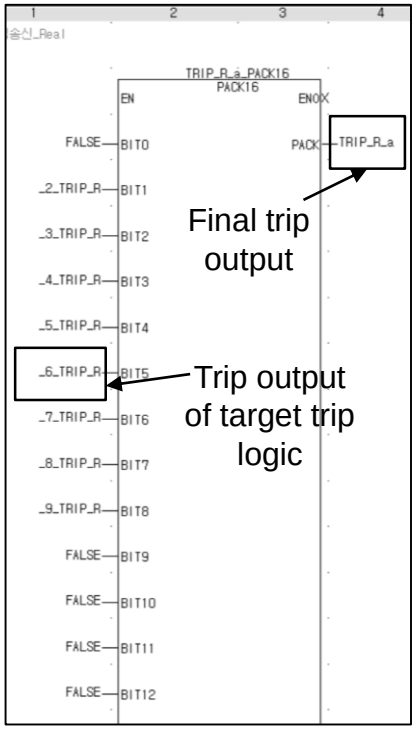
- A total of 4,206,164,480 test cases were generated for the target BP trip logic and tested in the developed test-bed.
- As a result, all test cases generated the trip output after BP software is executed. (~ *6.205 msec/case*)

ID of test case

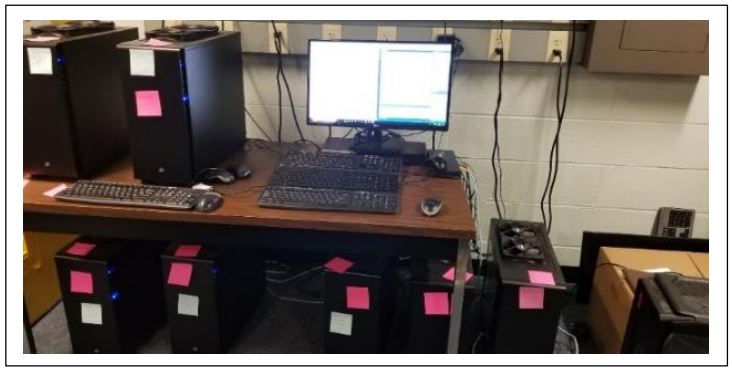
Status of input/ internal variable before test execution in test-bed

Status of output variable after test execution by test-bed

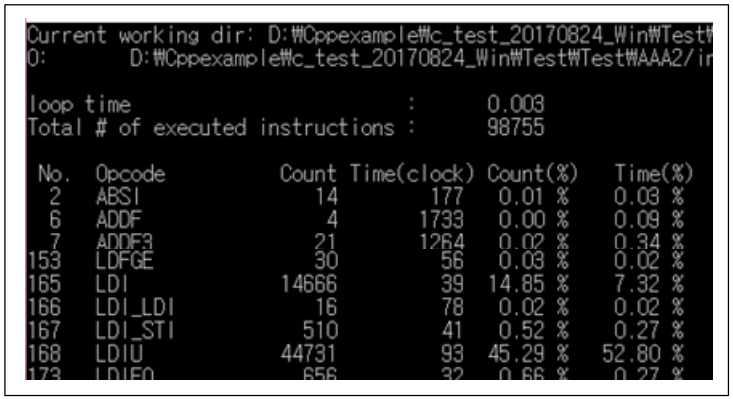




A part of test result for BP target trip logic



Test Setup (3.60GHz 640 CPUs)



```
Current working dir: D:₩Cppexample₩c_test_20170824_Win₩Test₩
O:       D:₩Cppexample₩c_test_20170824_Win₩Test₩Test₩AAA2/in

loop time                          :      0.003
Total # of executed instructions :      98755

 No.  Opcode          Count Time(clock) Count(%)   Time(%)
  2   ABSI               14        177   0.01 %    0.03 %
  6   ADDF                4       1733   0.00 %    0.09 %
  7   ADDF3              21       1264   0.02 %    0.34 %
153   LDFGE              30         56   0.03 %    0.02 %
165   LDI             14666         39  14.85 %    7.32 %
166   LDI_LDI            16         78   0.02 %    0.02 %
167   LDI_STI           510         41   0.52 %    0.27 %
168   LDIU            44731         93  45.29 %   52.80 %
173   LDIFO             656         32   0.66 %    0.27 %
```

Statistics on BP software execution in test-bed

Demonstration of Test-bed

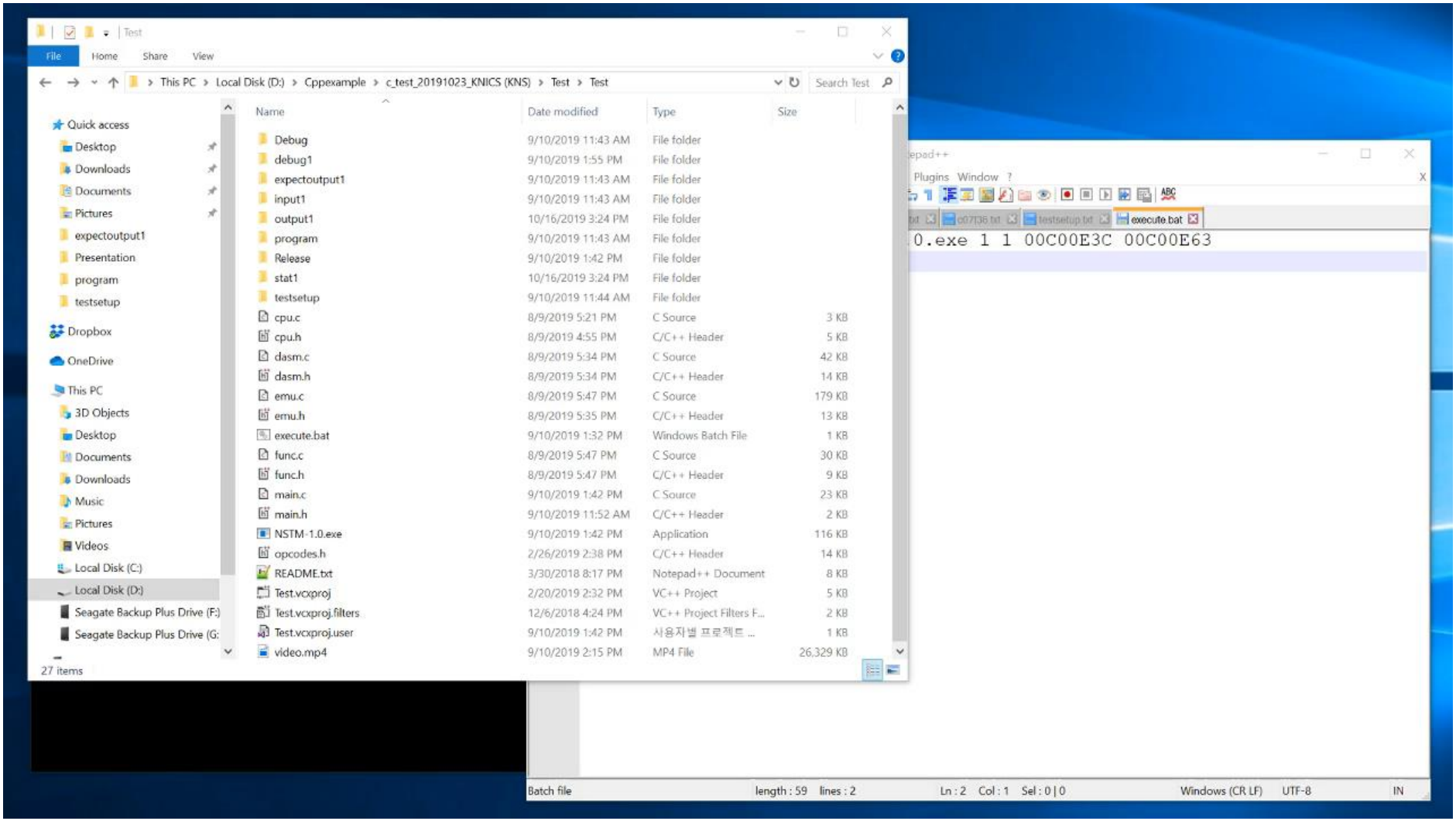
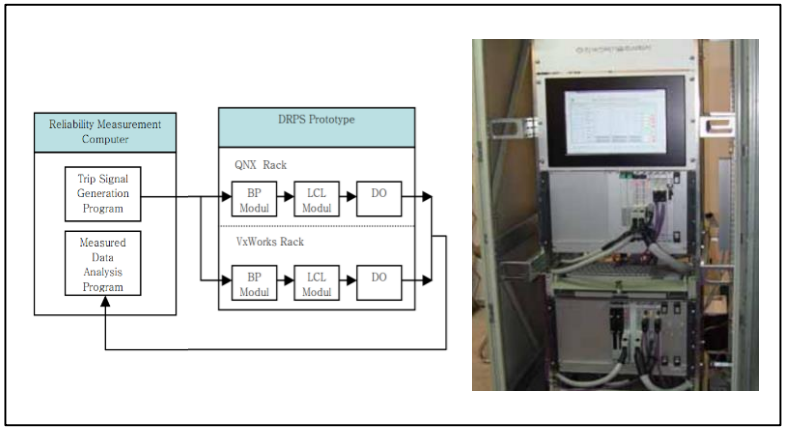


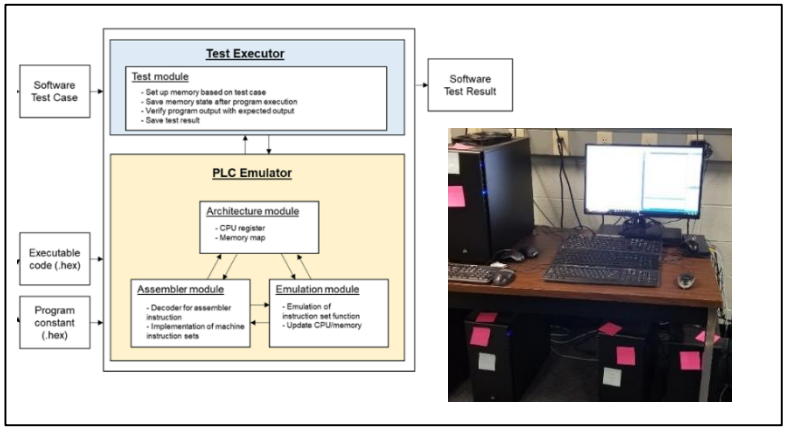
Figure of Merit : Enabling Exhaustive Testing

- Trajectory-based approach **(~ *hours/case*)**
  - 1) Tests the software installed in prototype test environment
  - 2) Interrupts software run and check software output for each test input
  - 3) Often not successful in explaining anomalies in test results

- Input-space-based approach (~ ***msec/case***)
  - 1) Directly tests the software in machine language level
  - 2) Automates the software testing process (feed input ~ analyze output)
  - 3) Able to track test results by checking CPU register/memory states



Trajectory-based approach



Input-space-based approach

Figure of Merit: Test Case Generation

- Nested-loop method **(~ *58.5 msec/case*)**
  - Step1: run formula with given software input/internal states
  - Step2: check whether software satisfies test requirement & save result

- SMT-based method **(~ *19.9 msec/case*)**
  - Step1: Check satisfiability of a formula given test requirement
  - Step2: retrieve and save model of a formula
  - Step3: add new constraint to the formula that negates last found models

Target Logic	Proposed : Step 1 / Step 2 / Step 3 Conventional : Step 1 / Step 2				(ms/case)	N
PZR_PR_LO_TRIP	4.895	0.158	14.855	19.907	58.508	4,206,164,480
	57.971			0.537		

➔ Same number of exhaustive test cases derived, but the proposed method showed 2.94 times faster performance compared to nested-loop method.

New Academic/Technical Contributions

Integrated Software Exhaustive Testing Framework for NPP Safety SW

(Test Case Generation)
How can we generate exhaustive test cases for software?

(Test environment)
How can we conduct exhaustive testing for software?

Challenge

- Need to develop a method that can ensure exhaustive test cases based on concrete theory and can be applied to any safety SW.

Solution

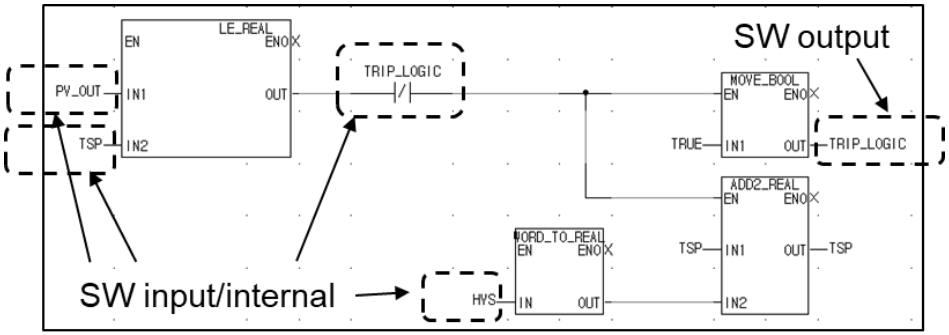
- Formulate the problem to finding all possible solutions for FBD program satisfying the test requirements for NPP SW safety function.

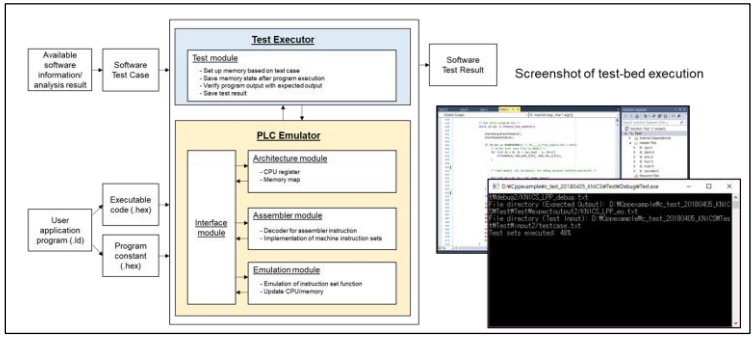
Challenge

- Existing trajectory-based approach utilizing hardware test-beds is not applicable.

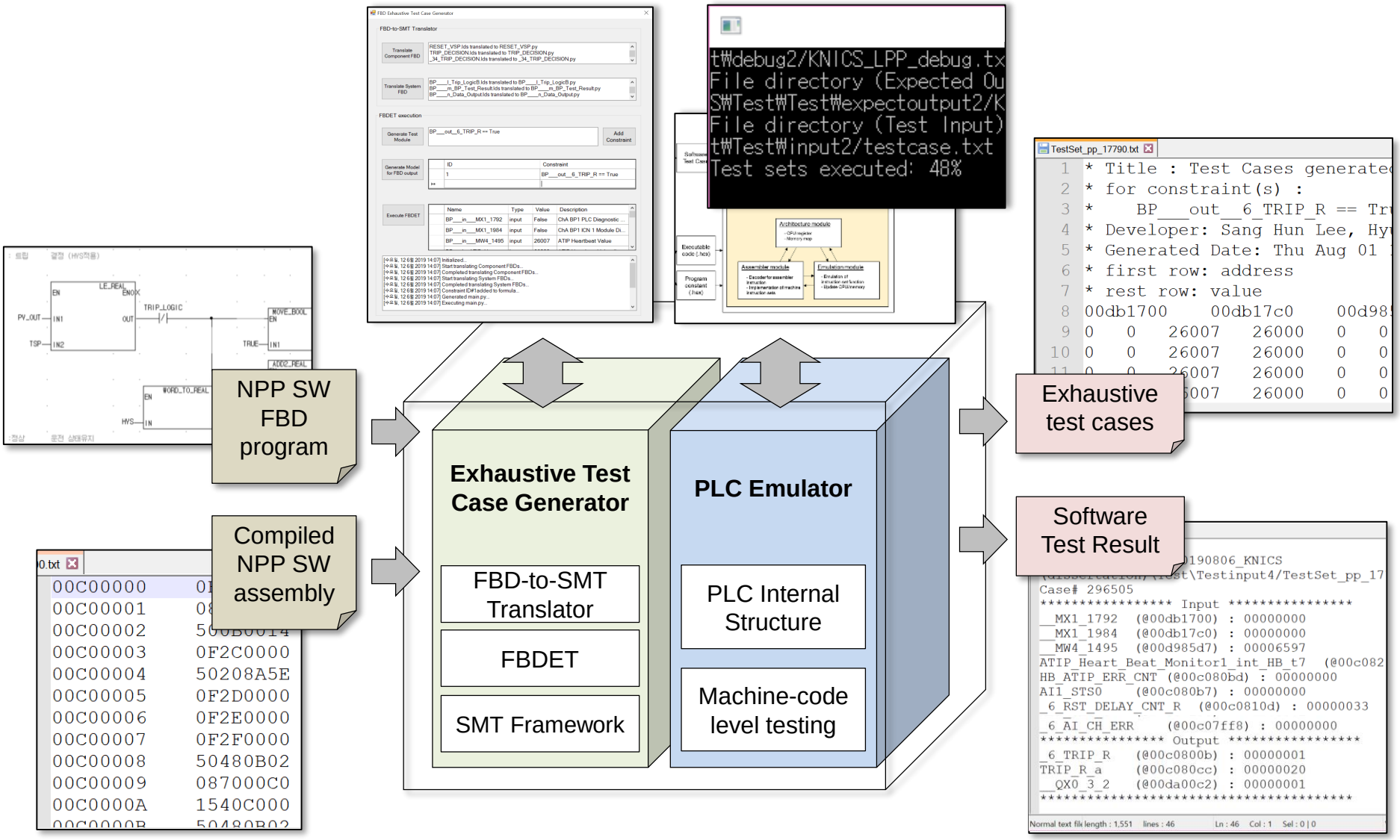
Solution

- Develop a simulation-based software test-bed which emulates PLC operation and tests the NPP SW in machine language level.





Thesis Summary



Exhaustive Testing Framework for NPP Safety Software

Publications

- **Published Journal Papers**

1. **S. H. Lee**, S. J. Lee, S. M. Shin, E.-C. Lee, H. G. Kang, "Exhaustive Testing of Safety-critical Software for Reactor Protection System," *Reliability Engineering & System Safety*, vol. 193, 106667, Jan, 2020.
2. **S. H. Lee**, S. J. Lee, S. R. Koo, T.-L. Chu, A. Varuttamaseni, M. Yue, M. Li, J. Cho, H. G. Kang, "Optimization of Software Development Life Cycle Quality for NPP Safety System based on a Risk-cost Model," *Annals of Nuclear Energy*, vol. 135, 106961, Jan, 2020.
3. S. J. Lee, **S. H. Lee**, T.-L. Chu, A. Varuttamaseni, M. Yue, M. Li, J. Cho, H. G. Kang, "Bayesian Belief Network Model Quantification using Distribution-based Node Probability and Experienced Data Updates for Software Reliability Assessment," *IEEE Access*, vol. 6, pp. 64556-64568, Oct. 2018.
4. H. G. Kang, **S. H. Lee**, S. J. Lee, T.-L. Chu, A. Varuttamaseni, M. Yue, H. S. Eom, J. Cho, M. Li, "Development of a Bayesian belief network model for software reliability quantification of digital protection systems in nuclear power plants," *Annals of Nuclear Energy*, vol. 120, pp. 62–73, Oct. 2018.
5. **S. H. Lee**, S. J. Lee, J. Park, E. Lee, H. G. Kang, “Development of Simulation-based Testing Environment for Safety-critical Software,” *Nuclear Engineering and Technology*, vol. 50, Issue 4, pp. 570-581, May 2018.
6. **S. H. Lee**, K. S. Son, W. Jung, and H. G. Kang, “Risk Assessment of Safety Data Link and Network Communication in Digital Safety Feature Control System of Nuclear Power Plant,” *Annals of Nuclear Energy*, vol. 108, pp. 394–405, Oct. 2017.
7. **S. H. Lee** and H. G. Kang, “Integrated Framework for the External Cost Assessment of Nuclear Power Plant Accident considering Risk Aversion: The Korean Case,” *Energy Policy*, vol. 92, pp. 111–123, May 2016.
8. M. Yun, **S. H. Lee**, and H. G. Kang, “Analysis of the Relationship between Risk Perception and Willingness to Pay for Nuclear Power Plant Risk Reduction,” *Science and Technology of Nuclear Installations*, vol. 2016, pp. 1–9, Mar. 2016.
9. **S. H. Lee**, H. E. Kim, K. S. Son, S. M. Shin, S. J. Lee, and H. G. Kang, “Reliability Modeling of Safety-critical Network Communication in a Digitalized Nuclear Power Plant,” *Reliability Engineering & System Safety*, vol. 144, pp. 285–295, Dec. 2015.
10. **S. H. Lee** and H. G. Kang, “Integrated Societal Risk Assessment Framework for Nuclear Power and Renewable Energy Sources,” *Nuclear Engineering and Technology*, vol. 47, no. 4, pp. 461–471, Jun. 2015.

Publications

- **Peer-reviewed Conference Papers**

1. **S. H. Lee**, S. J. Lee, S. M. Shin, E. Lee, H. G. Kang, "Exhaustive Testing Framework for Nuclear Safety-Critical Software," *The 30th European Safety and Reliability Conference and the 15th Probabilistic Safety Assessment and Management Conference (ESREL 2020/PSAM 15)*, Venice, Italy, June 21-26, 2020. (Submitted)
2. **S. H. Lee**, S. J. Lee, S. M. Shin, E. Lee, H. G. Kang, "Automated Exhaustive Test Case Generation for Safety-critical Software," *2019 ANS Winter Meeting and Nuclear Technology Expo (ANS Winter 2019)*, Washington DC, November 17-21, 2019.
3. **S. H. Lee**, S. J. Lee, S. M. Shin, E. Lee, H. G. Kang, "Development of Simulation-based Testing Environment for Safety-critical Software in Digitalized Nuclear Power Plant," *11th Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies, Orlando (NPIC&HMIT 2019)*, FL, February 9-14, 2019.
4. **S. H. Lee**, S. J. Lee, S. M. Shin, E. Lee, H. G. Kang, "Development of Software Test-based Reliability Assessment Method for Nuclear Power Plant Safety-critical Software," *The 14th Probabilistic Safety Assessment and Management (PSAM 14)*, September 2018, Los Angeles, CA, September 16-21, 2018.
5. **S. H. Lee**, S. J. Lee, S. M. Shin, E. Lee, H. G. Kang, "Development of Software Testing Environment for Safety-critical Software Reliability Quantification," *2018 ISA Power Industry Division Symposium (POWID 2018)*, Knoxville, TN, June 26-28, 2018.
6. **S. H. Lee**, S. J. Lee, S. M. Shin, E. Lee, H. G. Kang, "Development of Simulation-based Test-bed for Nuclear Power Plant Safety-critical Software," *ANS Annual Meeting 2018 (ANS 2018)*, Philadelphia, PA, June 17-21, 2018.
7. **S. H. Lee**, S. J. Lee, J. Park, E. Lee, H. G. Kang, "Development of Software Testing Environment for Safety-critical Software Reliability Quantification," *2017 International Symposium on Future I&C for Nuclear Power Plants (ISOFIC 2017)*, Gyeongju, Korea, November 26-30, 2017.
8. M. Li, H. G. Kang, **S. H. Lee**, S. J. Lee, T. Chu, A. Varuttamaseni, M. Yue, J. Cho, "Treatment of Expert Opinion Diversity in Bayesian Belief Network Model for Nuclear Digital I&C Safety Software Reliability Assessment," *2017 International Topical Meeting on Probabilistic Safety Assessment and Analysis (PSA 2017)*, Pittsberg, PA, USA, September 24-28, 2017.
9. **S. H. Lee**, H. G. Kang, T. Chu, A. Varuttamaseni, M. Yue, J. Cho, M. Li, S. J. Lee, "Analysis on the Important Indicators in Bayesian Belief Network Model for RPS Software Reliability Assessment," *2017 International Topical Meeting on Probabilistic Safety Assessment and Analysis (PSA 2017)*, Pittsberg, PA, USA, September 24-28, 2017.

Publications

- **Peer-reviewed Conference Papers**

10. S. J. Lee, **S. H. Lee**, H. G. Kang, T. Chu, A. Varuttamaseni, M. Yue, H. S. Eom, J. Cho, M. Li, "Application of a Bayesian Belief Network Model to Reliability Assessment of Nuclear Safety-Related Software," *The 10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC & HMIT 2017)*, San Francisco, CA, USA, June 11-15, 2017.
11. H. G. Kang, **S. H. Lee**, S. J. Lee, T. Chu, A. Varuttamaseni, M. Yue, S. Yang, H. S. Eom, J. Cho, M. Li, "Development of a Bayesian Belief Network Model for the Software Reliability Assessment of Nuclear Digital I&C Safety Systems," *The 10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC & HMIT 2017)*, San Francisco, CA, USA, June 11-15, 2017.
12. **S. H. Lee**, H. G. Kang, W. Jung, K. S. Son, "Risk Assessment of Safety-Critical Data Communication in Digital Safety Feature Control System," *The 10th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC & HMIT 2017)*, San Francisco, CA, USA, June 11-15, 2017. [pdf]
13. **S. H. Lee**, H. G. Kang, "Reliability Assessment of Safety-Critical Network Communication in a Digitalized Nuclear Power Plant," *ANS Annual Meeting 2016 (ANS 2016)*, Vol. 114, New Orleans, LA, USA, June 12-16, 2016.
14. S. M. Shin, S. J. Lee, **S. H. Lee**, H. G. Kang, "Quantitative risk assessment of digitalized safety systems," *Korean Nuclear Society Spring Meeting 2016 (KNS spring 2016)*, Jeju, Korea, May 11-13, 2016.
15. **S. H. Lee**, H. G. Kang, "External Cost Assessment of Nuclear Power Plant Accident considering Public Risk Aversion Behavior: the Korean Case," *Korean Nuclear Society Spring Meeting 2016 (KNS spring 2016)*, Jeju, Korea, May 11-13, 2016.
16. **S. H. Lee**, H. G. Kang, "Fault-Tree Modeling of Safety-Critical Network Communication in a Digitalized Nuclear Power Plant," *Korean Nuclear Society Autumn Meeting 2015 (KNS autumn 2015)*, Gyeongju, Korea, October 29-30, 2015.
17. M. Yun, **S. H. Lee**, H. G. Kang, "Analysis of the relationship between risk perception and willingness to pay about nuclear power plant," *International Symposium on Socially and Technically Symbiotic Systems/International Symposium on Symbiotic Nuclear Power Systems (STSS/ISSNP 2015)*, Kyoto, Japan, August 25-28, 2015.

Publications

- **Peer-reviewed Conference Papers**

18. **S. H. Lee**, H. G. Kang, "Measuring Risk Aversion for Nuclear Power Plant Accident: Results of Contingent Valuation Survey in Korea," *Korean Nuclear Society Spring Meeting 2015 (KNS spring 2015)*, Jeju, Korea, May 7-8, 2015.
19. S. M. Shin, **S. H. Lee**, H. G. Kang, H. S. Son, "Test Based Reliability Quantification Method for a Safety Critical Software using Finite Test Sets," *The 9th International Topical Meeting on Nuclear Plant Instrumentation, Control & Human–Machine Interface Technologies (NPIC&HMIT 2015)*, Charlotte, N.C., USA, February 23-26, 2015.
20. **S. H. Lee**, H. G. Kang, "Comparative Health Risk Assessment of CdTe Solar PV System and Nuclear Power Plant," *International Symposium on Future I&C for Nuclear Power Plants/International Symposium on Symbiotic Nuclear Power Systems 2014 (ISOFIC/ISSNP 2014)*, Jeju, Korea, August 24-28, 2014.
21. **S. H. Lee**, H. G. Kang, "Preliminary Assessment of Population Dose during Nuclear Power Plant Normal Operation," *International Youth Nuclear Congress 2014 (IYNC 2014)*, Burgos, Spain, July 6-12, 2014.

Thank you for your attention

Q&A