APSEC '08

A Verification Framework for FBD based Software in Nuclear Power Plants

JUNBEOM YOO

KONKUK University, Korea

jbyoo@konkuk.ac.kr
http://dslab.konkuk.ac.kr

Other Authors

Sungdeok Cha

- Professor in Korea University

Enukyoung Jee

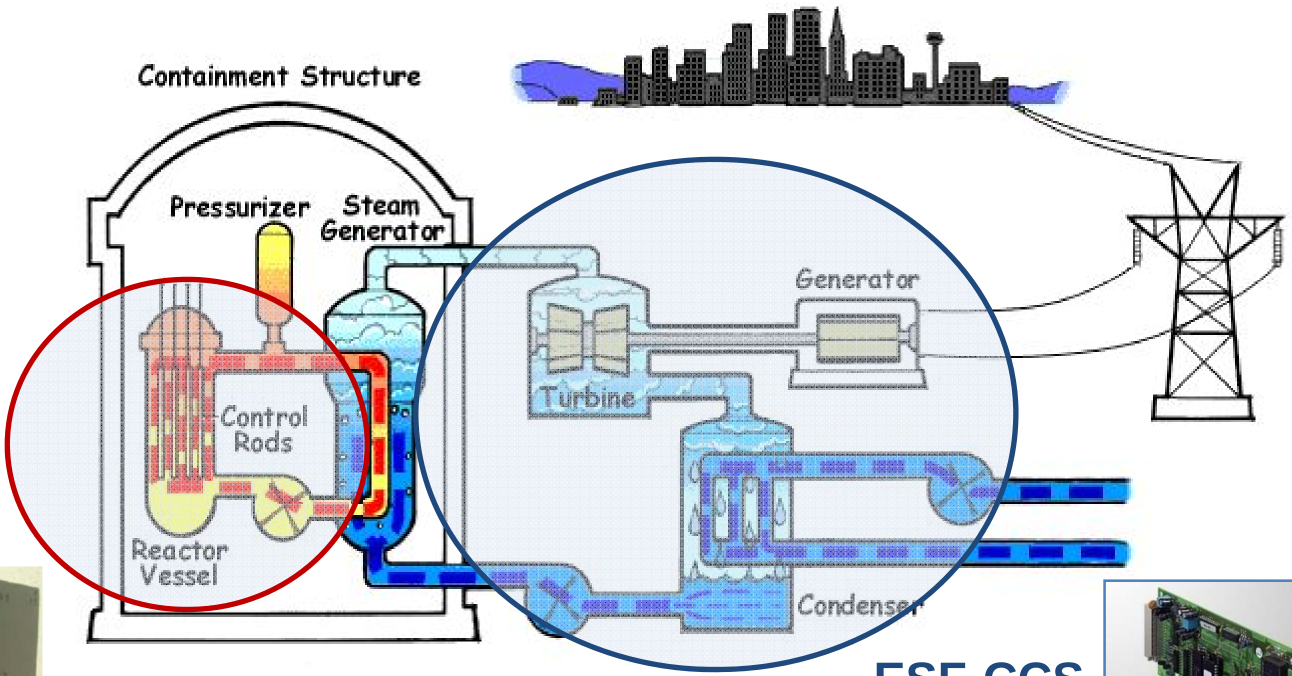
- PhD Candidate in KAIST

Contents

- Introduction
  - Safety-Critical Software in Nuclear Power Plants
  - Software Development Process
- Background
  - FBD
- Verification Framework
  - VIS Equivalence Checking
  - SMV Model Checking
  - Case Study
- Conclusion & Future Work

Introduction

- Safety-Critical Software in Nuclear Power Plants
  - RPS (Reactor Protection System)
  - ESF-CCS (Engineering Safety Features Component Control System)



RPS

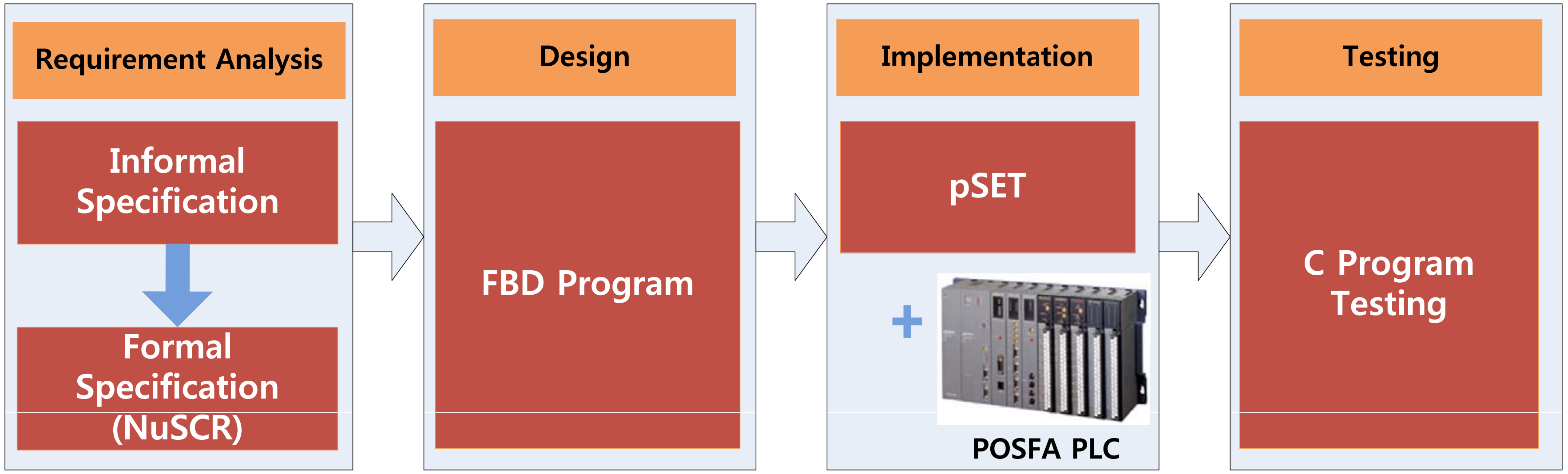
ESF-CCS

PLC

DCS

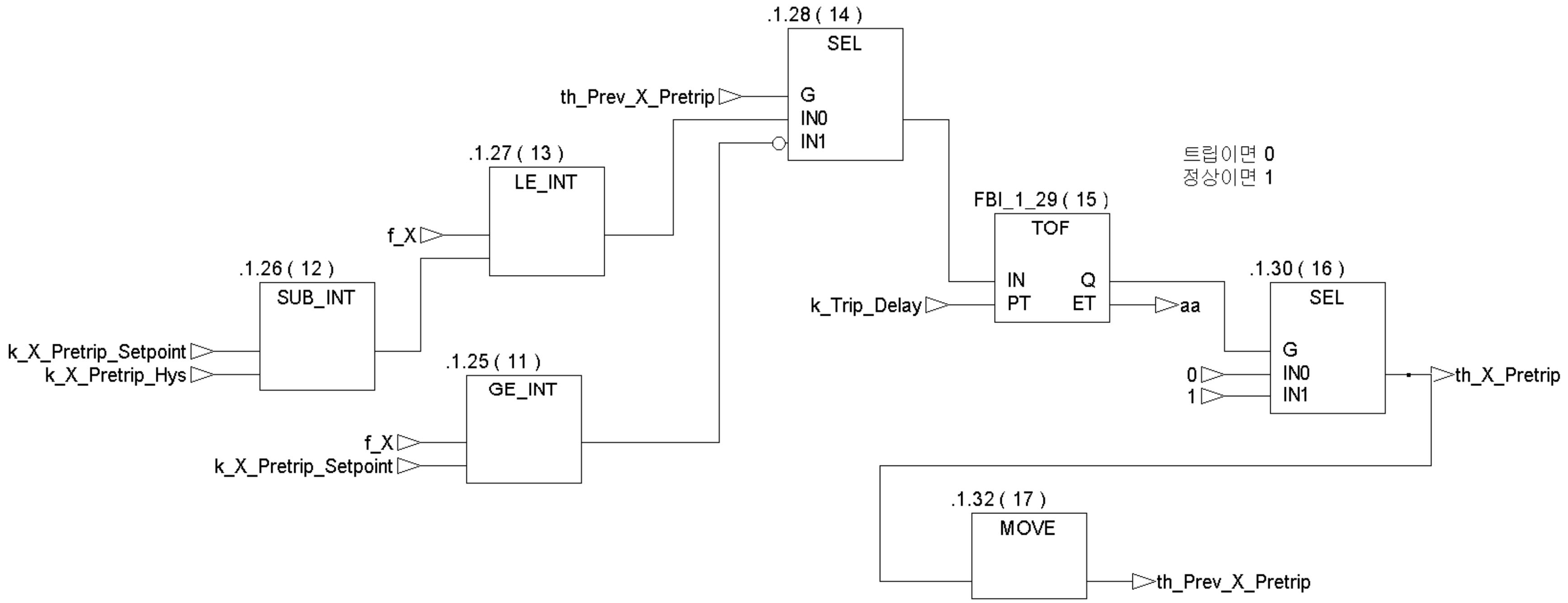
Introduction

- A Software Development Process for RPS in KNICS APR-1400 NPP
  - (http://www.knics.re.kr/english/eindex.html)



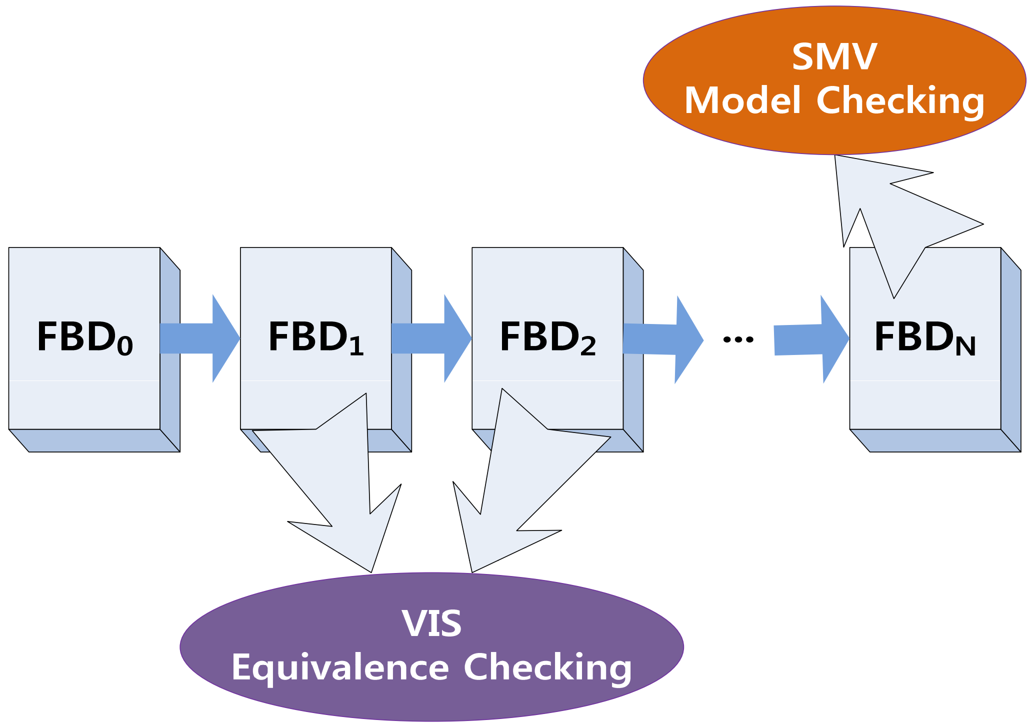
Background

- FBD (Function Block Diagram)
  - IEC 61131-3 standard declared 5 programming languages for PLC (ST, LD, IL, SFC, FBD)
  - KNICS consortium decided to use FBD to program KNICS RPS software
  - Sequential Interconnections between function blocks



Verification Framework

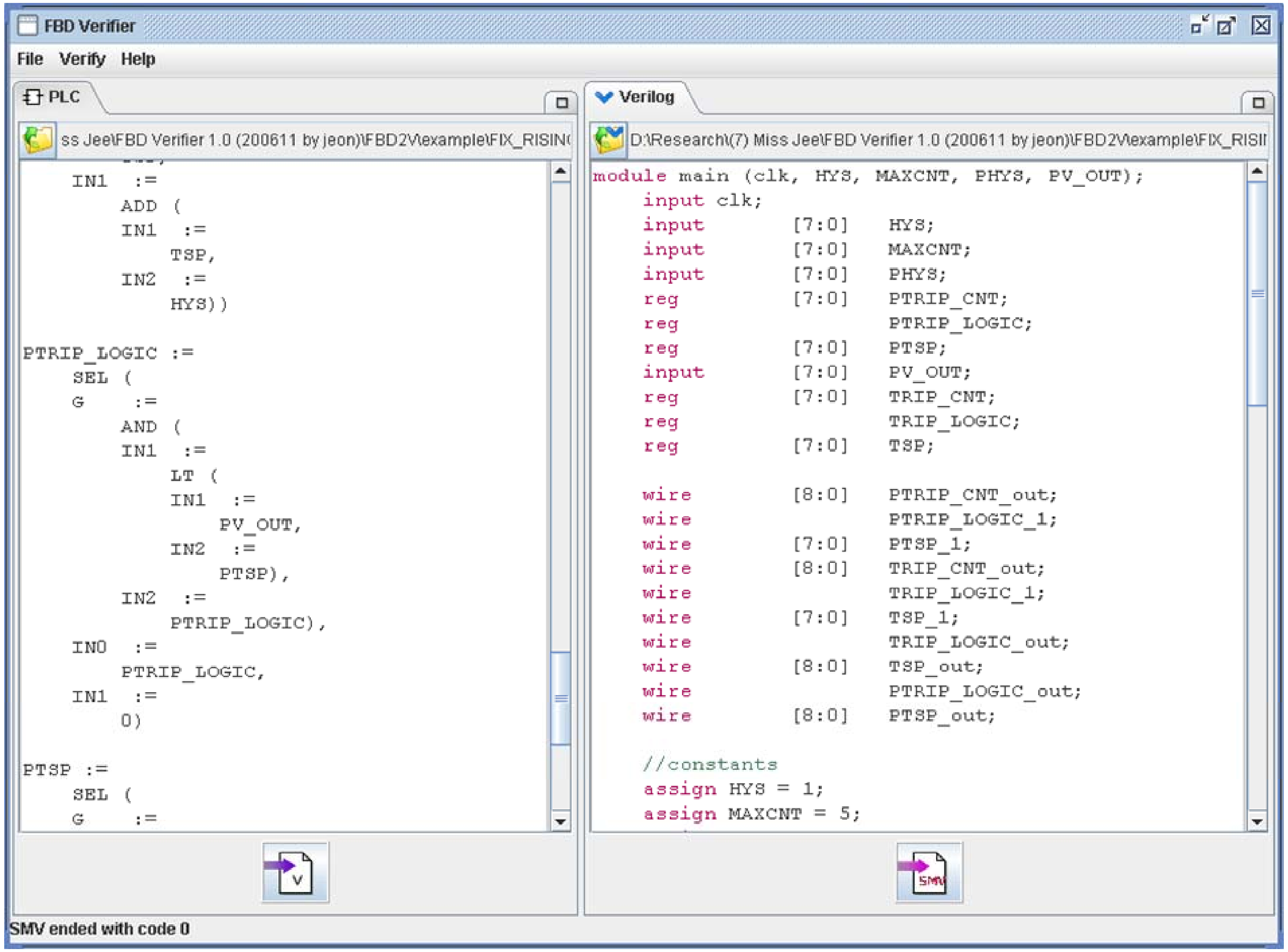
- In design phase,
- Two different formal verifications to verify FBD programs efficiently,
- Based on our experience on KNICS RPS for 7 years
  - Equivalence Checking : VIS verification system (ver.2.0)
  - Model Checking : Cadence SMV model checker



1. SMV Model Checking

- SMV Model Checking with LTL properties
  - **Cadence SMV model checker** (http://www.kenmcmil.com/smv.html)
    - An extension of SMV from CMU
    - CTL / LTL model checking
    - Two front-ends
      - SMV input programs (for CTL/LTL properties)
      - Verilog program (for LTL properties)
  - **FBD Verifier 1.0** (http://dslab.konkuk.ac.kr/Nuclear-Design/FBD_Verifier.htm)
    - Translates FBD into Verilog automatically
    - Properties are inserted into Verilog programs (through "assert" statement)
    - Executes Cadence SMV with translated Verilog program seamlessly

1. Read an FBD program in standard XML format
2. Translate the FBD into an equivalent Verilog program

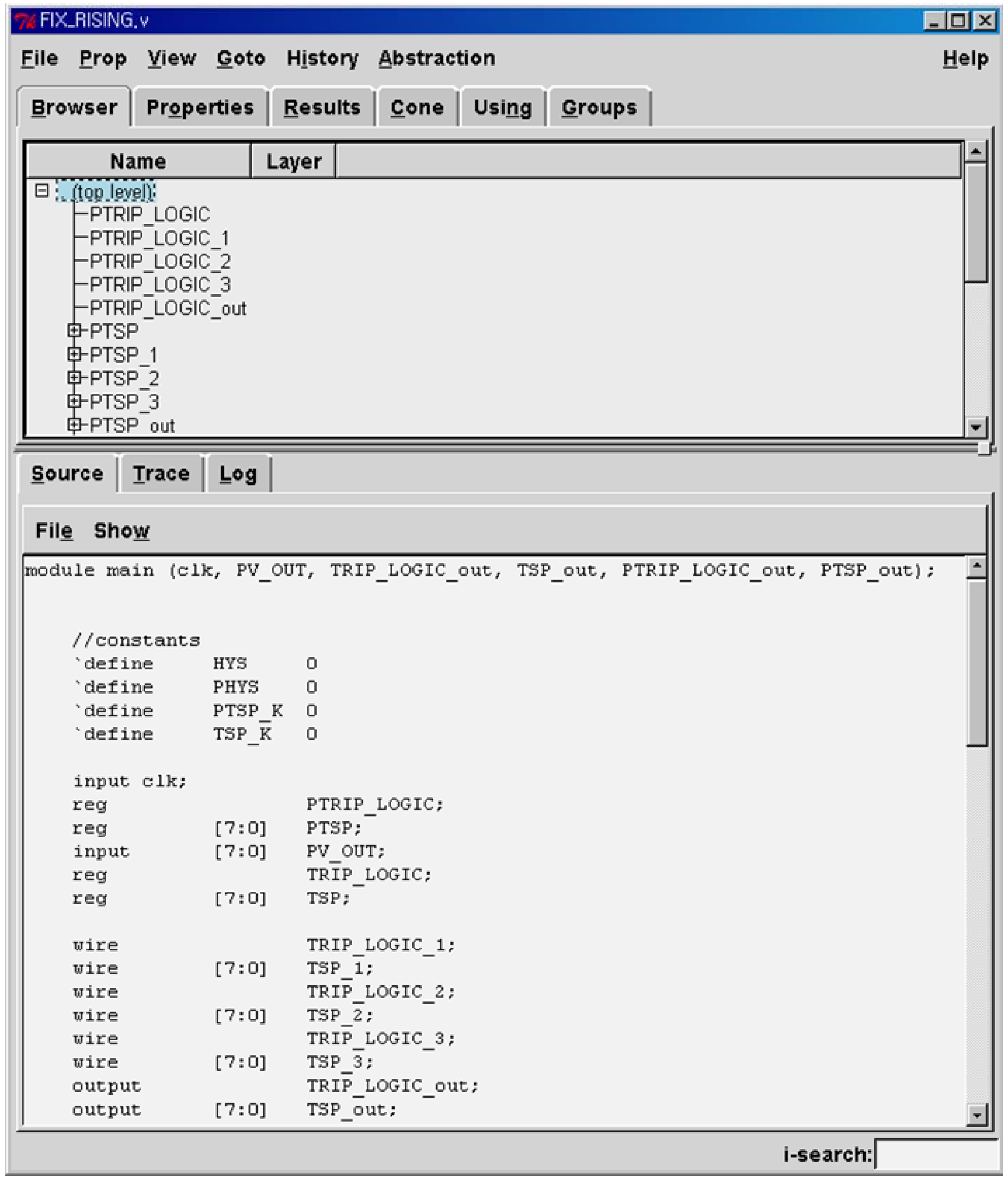

FBD Verifier
File Verify Help
PLC
Verilog
ss Jee\FBD Verifier 1.0 (200611 by jeon)\FBD2V\example\FIX_RISIN
D:\Research\(7) Miss Jee\FBD Verifier 1.0 (200611 by jeon)\FBD2V\example\FIX_RISII

```
    IN1  :=
        ADD (
        IN1  :=
            TSP,
        IN2  :=
            HYS))

PTRIP_LOGIC :=
    SEL (
    G    :=
        AND (
        IN1  :=
            LT (
            IN1  :=
                PV_OUT,
            IN2  :=
                PTSP),
        IN2  :=
            PTRIP_LOGIC),
    IN0  :=
        PTRIP_LOGIC,
    IN1  :=
        0)

PTSP :=
    SEL (
    G    :=
```

```
module main (clk, HYS, MAXCNT, PHYS, PV_OUT);
    input clk;
    input        [7:0]    HYS;
    input        [7:0]    MAXCNT;
    input        [7:0]    PHYS;
    reg          [7:0]    PTRIP_CNT;
    reg                   PTRIP_LOGIC;
    reg          [7:0]    PTSP;
    input        [7:0]    PV_OUT;
    reg          [7:0]    TRIP_CNT;
    reg                   TRIP_LOGIC;
    reg          [7:0]    TSP;

    wire         [8:0]    PTRIP_CNT_out;
    wire                  PTRIP_LOGIC_1;
    wire         [7:0]    PTSP_1;
    wire         [8:0]    TRIP_CNT_out;
    wire                  TRIP_LOGIC_1;
    wire         [7:0]    TSP_1;
    wire                  TRIP_LOGIC_out;
    wire         [8:0]    TSP_out;
    wire                  PTRIP_LOGIC_out;
    wire         [8:0]    PTSP_out;

    //constants
    assign HYS = 1;
    assign MAXCNT = 5;
```

SMV ended with code 0

3. Execute Cadence SMV model checker



```
module main (clk, PV_OUT, TRIP_LOGIC_out, TSP_out, PTRIP_LOGIC_out, PTSP_out);


    //constants
    `define     HYS     0
    `define     PHYS    0
    `define     PTSP_K  0
    `define     TSP_K   0

    input clk;
    reg                 PTRIP_LOGIC;
    reg         [7:0]   PTSP;
    input       [7:0]   PV_OUT;
    reg                 TRIP_LOGIC;
    reg         [7:0]   TSP;

    wire                TRIP_LOGIC_1;
    wire        [7:0]   TSP_1;
    wire                TRIP_LOGIC_2;
    wire        [7:0]   TSP_2;
    wire                TRIP_LOGIC_3;
    wire        [7:0]   TSP_3;
    output              TRIP_LOGIC_out;
    output      [7:0]   TSP_out;
```

Case Study

- SMV model checking for KNCIS RPS BP & CP
  - Performed to up-to-date whole KNICS-RPS-SDS231 Rev.02

System	# of pages of requirements Spec. (Natural lang.)	# of function blocks	# of variables	# of lines in Verilog model
BP	190	1,335	1,038	7,862
CP	163	1,623	820	3,085

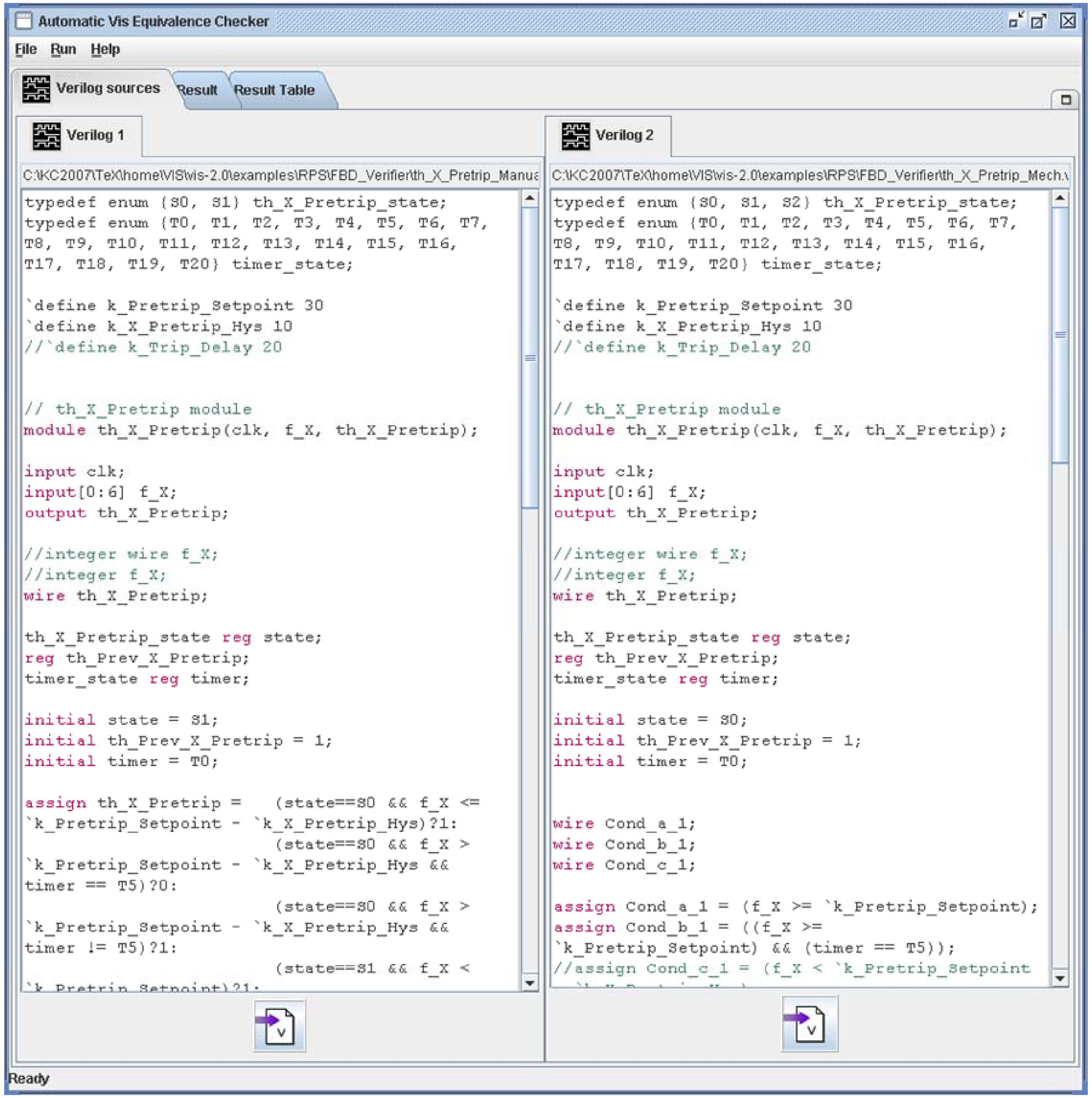
  - Found a few, not many important verification results

System		**BP**	**CP**
# of Properties		216	83
Found Errors	**Incorrect Logic**	14	6
	Omission	0	2
	Ambiguous Logic	4	0
	Incorrect FBD	13	5
	Incorrect Design	16	0
Total # of Errors		47	13
Distinct # of Errors		10	3

2. VIS Equivalence Checking

- Behavioral Equivalence Checking between two FBD programs
  - **VIS verification system 2.0** (http://embedded.eecs.berkeley.edu/research/vis//)
    - Widely used in hardware design,
      - Simulation
      - CTL model checking
      - Equivalence checking
      - Etc.
    - Reads Verilog program
    - But, no graphical interface
  - **VIS Analyzer 1.0** (http://dslab.konkuk.ac.kr/Nuclear-Design/VIS_Analyzer.htm)
    - Seamless execution of VIS verifications
      - *vl2mv* → *read_blif_mv* → *flatten_hierarchy* → *seq_verify* , *simulate*
    - Automatic reorganization of verification result through VIS simulation

1. Read two Verilog programs



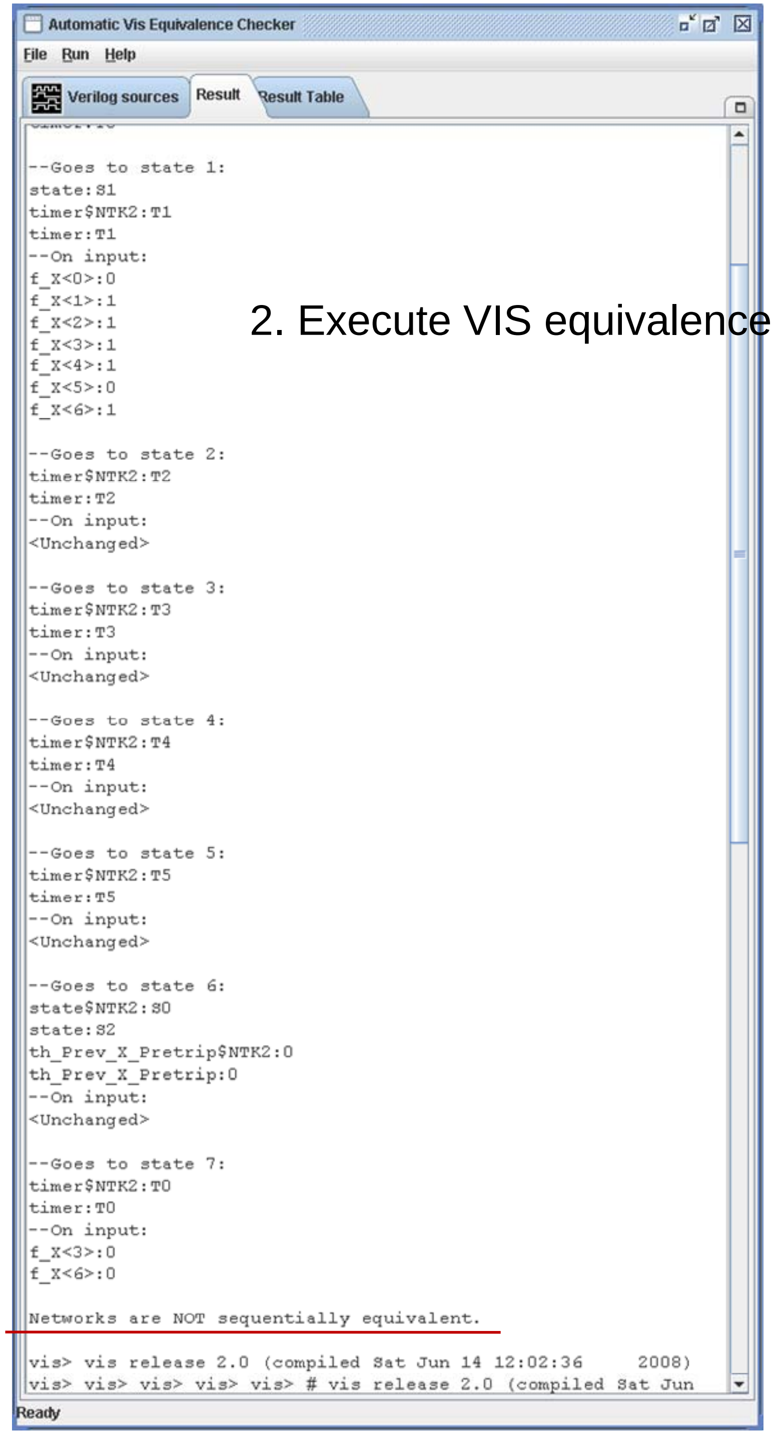
C:\KC2007\TeX\home\VIS\vis-2.0\examples\RPS\FBD_Verifier\th_X_Pretrip_Manua

```verilog
typedef enum {S0, S1} th_X_Pretrip_state;
typedef enum {T0, T1, T2, T3, T4, T5, T6, T7,
T8, T9, T10, T11, T12, T13, T14, T15, T16,
T17, T18, T19, T20} timer_state;

`define k_Pretrip_Setpoint 30
`define k_X_Pretrip_Hys 10
//`define k_Trip_Delay 20


// th_X_Pretrip module
module th_X_Pretrip(clk, f_X, th_X_Pretrip);

input clk;
input[0:6] f_X;
output th_X_Pretrip;

//integer wire f_X;
//integer f_X;
wire th_X_Pretrip;

th_X_Pretrip_state reg state;
reg th_Prev_X_Pretrip;
timer_state reg timer;

initial state = S1;
initial th_Prev_X_Pretrip = 1;
initial timer = T0;

assign th_X_Pretrip =   (state==S0 && f_X <=
`k_Pretrip_Setpoint - `k_X_Pretrip_Hys)?1:
                        (state==S0 && f_X >
`k_Pretrip_Setpoint - `k_X_Pretrip_Hys &&
timer == T5)?0:
                        (state==S0 && f_X >
`k_Pretrip_Setpoint - `k_X_Pretrip_Hys &&
timer != T5)?1:
                        (state==S1 && f_X <
`k_Pretrip_Setpoint)?1:
```

C:\KC2007\TeX\home\VIS\vis-2.0\examples\RPS\FBD_Verifier\th_X_Pretrip_Mech.\

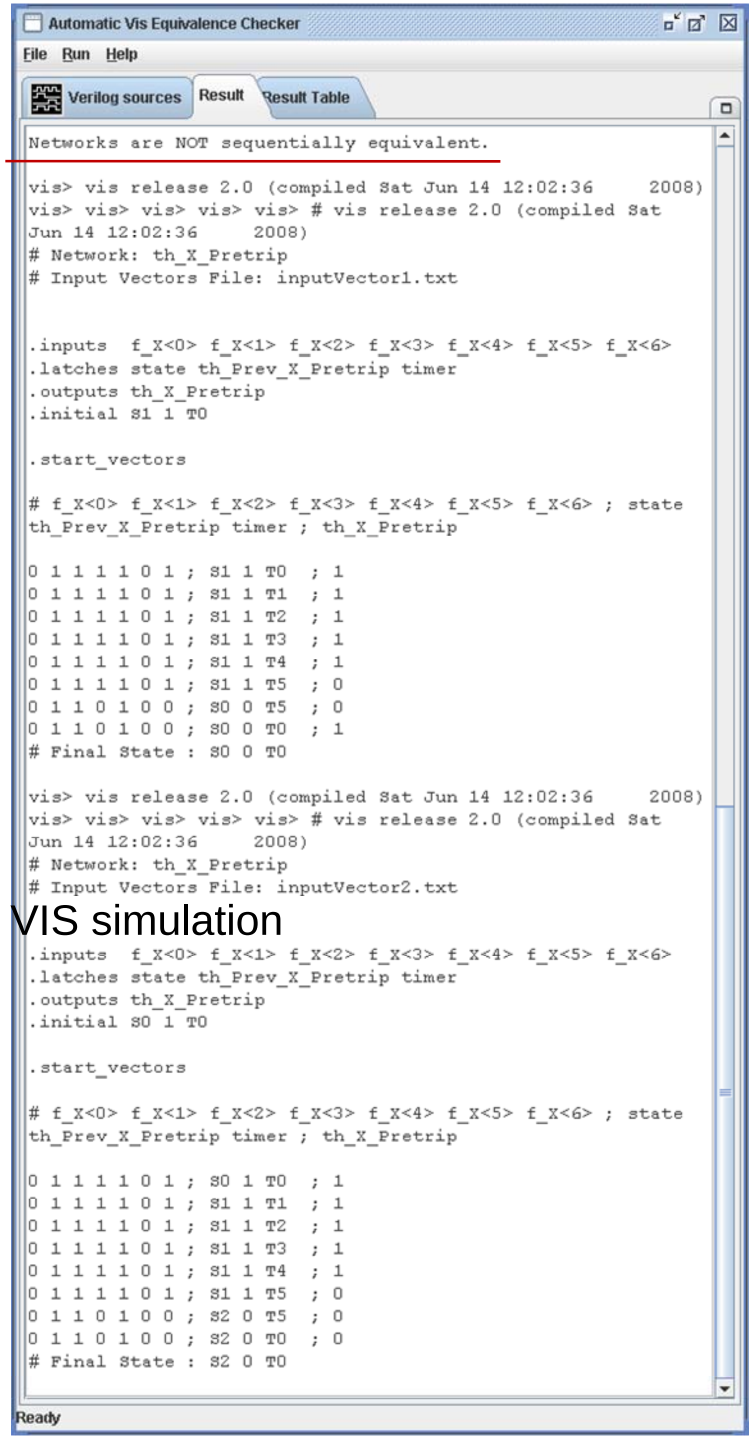
```verilog
typedef enum {S0, S1, S2} th_X_Pretrip_state;
typedef enum {T0, T1, T2, T3, T4, T5, T6, T7,
T8, T9, T10, T11, T12, T13, T14, T15, T16,
T17, T18, T19, T20} timer_state;

`define k_Pretrip_Setpoint 30
`define k_X_Pretrip_Hys 10
//`define k_Trip_Delay 20


// th_X_Pretrip module
module th_X_Pretrip(clk, f_X, th_X_Pretrip);

input clk;
input[0:6] f_X;
output th_X_Pretrip;

//integer wire f_X;
//integer f_X;
wire th_X_Pretrip;

th_X_Pretrip_state reg state;
reg th_Prev_X_Pretrip;
timer_state reg timer;

initial state = S0;
initial th_Prev_X_Pretrip = 1;
initial timer = T0;


wire Cond_a_1;
wire Cond_b_1;
wire Cond_c_1;

assign Cond_a_1 = (f_X >= `k_Pretrip_Setpoint);
assign Cond_b_1 = ((f_X >=
`k_Pretrip_Setpoint) && (timer == T5));
//assign Cond_c_1 = (f_X < `k_Pretrip_Setpoint
```

2. Execute VIS equivalence checking



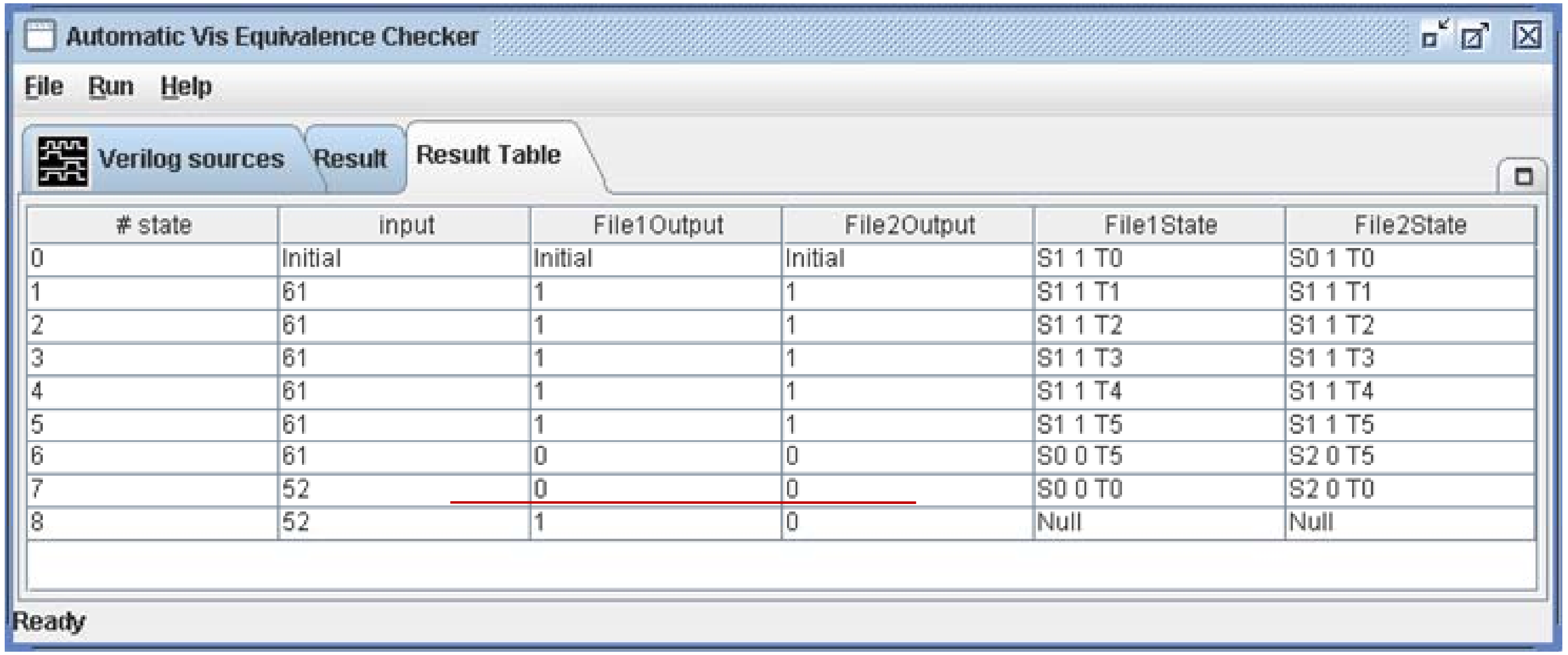
```
--Goes to state 1:
state:S1
timer$NTK2:T1
timer:T1
--On input:
f_X<0>:0
f_X<1>:1
f_X<2>:1
f_X<3>:1
f_X<4>:1
f_X<5>:0
f_X<6>:1

--Goes to state 2:
timer$NTK2:T2
timer:T2
--On input:
<Unchanged>

--Goes to state 3:
timer$NTK2:T3
timer:T3
--On input:
<Unchanged>

--Goes to state 4:
timer$NTK2:T4
timer:T4
--On input:
<Unchanged>

--Goes to state 5:
timer$NTK2:T5
timer:T5
--On input:
<Unchanged>

--Goes to state 6:
state$NTK2:S0
state:S2
th_Prev_X_Pretrip$NTK2:0
th_Prev_X_Pretrip:0
--On input:
<Unchanged>

--Goes to state 7:
timer$NTK2:T0
timer:T0
--On input:
f_X<3>:0
f_X<6>:0

Networks are NOT sequentially equivalent.

vis> vis release 2.0 (compiled Sat Jun 14 12:02:36     2008)
vis> vis> vis> vis> vis> # vis release 2.0 (compiled Sat Jun
```

3. Execute VIS simulation



```
Networks are NOT sequentially equivalent.

vis> vis release 2.0 (compiled Sat Jun 14 12:02:36     2008)
vis> vis> vis> vis> vis> # vis release 2.0 (compiled Sat
Jun 14 12:02:36     2008)
# Network: th_X_Pretrip
# Input Vectors File: inputVector1.txt


.inputs  f_X<0> f_X<1> f_X<2> f_X<3> f_X<4> f_X<5> f_X<6>
.latches state th_Prev_X_Pretrip timer
.outputs th_X_Pretrip
.initial S1 1 T0

.start_vectors

# f_X<0> f_X<1> f_X<2> f_X<3> f_X<4> f_X<5> f_X<6> ; state
th_Prev_X_Pretrip timer ; th_X_Pretrip

0 1 1 1 1 0 1 ; S1 1 T0  ; 1
0 1 1 1 1 0 1 ; S1 1 T1  ; 1
0 1 1 1 1 0 1 ; S1 1 T2  ; 1
0 1 1 1 1 0 1 ; S1 1 T3  ; 1
0 1 1 1 1 0 1 ; S1 1 T4  ; 1
0 1 1 1 1 0 1 ; S1 1 T5  ; 0
0 1 1 0 1 0 0 ; S0 0 T5  ; 0
0 1 1 0 1 0 0 ; S0 0 T0  ; 1
# Final State : S0 0 T0

vis> vis release 2.0 (compiled Sat Jun 14 12:02:36     2008)
vis> vis> vis> vis> vis> # vis release 2.0 (compiled Sat
Jun 14 12:02:36     2008)
# Network: th_X_Pretrip
# Input Vectors File: inputVector2.txt


.inputs  f_X<0> f_X<1> f_X<2> f_X<3> f_X<4> f_X<5> f_X<6>
.latches state th_Prev_X_Pretrip timer
.outputs th_X_Pretrip
.initial S0 1 T0

.start_vectors

# f_X<0> f_X<1> f_X<2> f_X<3> f_X<4> f_X<5> f_X<6> ; state
th_Prev_X_Pretrip timer ; th_X_Pretrip

0 1 1 1 1 0 1 ; S0 1 T0  ; 1
0 1 1 1 1 0 1 ; S1 1 T1  ; 1
0 1 1 1 1 0 1 ; S1 1 T2  ; 1
0 1 1 1 1 0 1 ; S1 1 T3  ; 1
0 1 1 1 1 0 1 ; S1 1 T4  ; 1
0 1 1 1 1 0 1 ; S1 1 T5  ; 0
0 1 1 0 1 0 0 ; S2 0 T5  ; 0
0 1 1 0 1 0 0 ; S2 0 T0  ; 0
# Final State : S2 0 T0
```

4. Display a full trace for counterexample



# state	input	File1Output	File2Output	File1State	File2State
0	Initial	Initial	Initial	S1 1 T0	S0 1 T0
1	61	1	1	S1 1 T1	S1 1 T1
2	61	1	1	S1 1 T2	S1 1 T2
3	61	1	1	S1 1 T3	S1 1 T3
4	61	1	1	S1 1 T4	S1 1 T4
5	61	1	1	S1 1 T5	S1 1 T5
6	61	0	0	S0 0 T5	S2 0 T5
7	52	0	0	S0 0 T0	S2 0 T0
8	52	1	0	Null	Null

Case Study

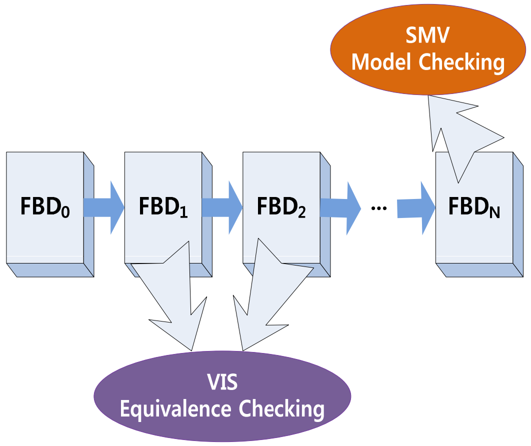
- VIS equivalence checking for KNCIS RPS BP
  - We didn’t meet the schedule, so a few official verification results are left only.
    - Requirements: KNICS-RPS-SRS101 Rev.00 (prototype)
    - Original FBD: KNICS-RPS-SDS101 Rev.00 (prototype)
    - Compared FBD: Synthesized automatically from the requirements spec.

  - Found several errors

Trip Logic	**Error Type**	**Compared FBD (Num. of Errors)**	**Original FBD (Num. of Errors)**
Fixed Set-Point Rising Trip without Operating Bypass	Syntactic Logical	0 0	0 1
Manual Reset Variable Set-Point Trip without Operating Bypass	Syntactic Logical	0 6	3 2

BP: Bistable Processor in RPS

Conclusion & Future Work



- We proposed a software verification framework
  - Target: KNICS RPS
    - HW: PLC (Programmable Logic Controller)
    - SW: FBD (Function Block Diagram)
  - Two verification techniques together
    - SMV Model Checking (Cadence SMV + FBD Verifier)
    - VIS Equivalence Checking (VIS 2.0 + VIS Analyzer)
  - They performed the formal verification of KNICS RPS sufficiently.

- We're planning
  - A combined tool-set (FBD Verifier + VIS analyzer) with enhanced GUIs
  - Enhance through applying to other systems (e.g. ESF-CCS)