VIS 정형 검증 자동화를 위한 소프트웨어 VIS Analyzer 개발

정세훈[O,a], 유준범[b], 차성덕[a]
[a] 고려대학교 컴퓨터학과, [b] 건국대학교 컴퓨터공학부
gifaranga@yahoo.co.kr, jbyoo@konkuk.ac.kr, scha@korea.ac.kr

<VIS Analyzer: Automation & Visualization of VIS>

Sehun Jeong[a,O], Junbeom Yoo[b], Sungdeok Cha[a]
[a] Dept. of CS and Engineering. Korea Univ.
[b] Div. of CS and Engineering. Konkuk Univ.

요 약

Safety-Critical System을 구성하는 소프트웨어는 높은 안전성과 품질 보장을 위하여, 정형 기법의 적용이 요구되고 있다. VIS(Verification Interacting with Synthesis)는 모델 체킹, 일치성 검증, 시뮬레이션 등 다양한 정형검증을 지원하는 널리 사용되는 도구이다. 하지만, VIS는 콘솔 기반의 UI만 제공하며, 부분적인 정보 만으로 구성된 검증 결과를 제공하므로, VIS의 시뮬레이션 기능을 사용하여 보다 정확하고 자세한 검증 내용을 확인해야 하는 문제점이 있다. 본 논문에서는 이러한 VIS의 검증 기능 사용에서 발생하는 문제점들을 해결할 수 있는 지원 도구인 VIS Analyzer를 소개한다. VIS Analyzer는 Windows 기반의 GUI를 통해 VIS 검증 과정을 자동화했으며, 시뮬레이션을 자동으로 수행함으로써 VIS 일치성 검증 결과를 정확하게 분석할 수 있도록 지원하고 있다. 제안하는 도구의 효율성을 검증하기 위해서 현재 KNICS 사업단에서 개발중인 APR-1400 원자로 보호 시스템의 원자로 보호 시스템(RPS: Reactor Protection System)을 예로 사용하였다.

1. 서 론

Safety-critical system[1]은 실행 중 오류가 발생했을 경우 큰 인적, 물질적 피해가 일어날 수 있는 시스템이다. 그러므로 이러한 시스템에서 실행되는 소프트웨어는 안전성(safety), 완전성(completeness), 무결성(integrity) 등을 검증해야 한다. 원자력 발전소 제어 시스템 등이 그 대표적인 예로 들 수 있다. 정형 기법[2]은 정형 명세와 정형 검증을 통해 소프트웨어의 안전성, 완전성, 무결성 등을 검증할 수 있어, 이러한 safety-critical system의 개발에 널리 사용된다.

VIS[4]는 정형 기법 적용에 활용할 수 있는 통합 검증 도구로서, CTL 모델 체킹, 순차적 일치성 검증(sequential equivalence checking) 기능, 조합적 일치성 검증(combinational equivalence checking) 기능, 시뮬레이션 기능을 제공한다. CTL 모델 체킹은 소프트웨어가 특정 속성을 만족하는 지 검사할 수 있다. 그리고 순차적 일치성 검증 기능과 조합적 일치성 검증 기능은 두 소프트웨어간의 행동 일치성을 각각 순차적, 조합적으로 검증할 수 있다. VIS는 위와 같은 여러 검증기법을 수행할 수 있으므로 safety-critical system의 정형 기법 적용하는 데[3] 효과적이다.

하지만 VIS 검증 도구는 일치성 검증을 수행할 때 두 가지 단점을 가진다. 첫 번째, 실행 환경이 콘솔 기반이기 때문에 모든 명령어는 사용자가 일일이 입력해줘야 하며 한 화면에 다양한 정보를 출력할 수 없다. 두 번째, VIS의 순차적 검증 결과는 전체 입력과 출력 정보가 빠져있어 불완전하며 해석하기 어렵다.

본 논문에서는 위와 같은 VIS의 단점을 보완하는 도구인 VIS Analyzer를 소개한다. VIS Analyzer는 두 가지 목적에 초점을 두고 개발되었다. 첫 번째 목적은 그래픽 사용자 인터페이스를 사용하여 VIS의 파일 입/출력과 검증 과정을 간략화 하는 것이다. 두 번째 목적은 VIS의 순차적 검증 결과를 분석하고 보완하여 가독성이 좋은 표 형태로 보여주는 것 이다. 보완을 위해 VIS의 시뮬레이션 기능을 이용한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 VIS 검증 도구를 이용한 일치성 검증 과정을 예제를 들어 설명한다. 3장에서는 VIS Analyzer의 기능을 2장과 같은 예제를 이용하여 설명한다. 2장과 3장에서 사용한 예제는 KNICS[6] APR-1400 원자로 보호 시스템을 위한 FBD(Function Block Diagram)로 작성된

소프트웨어를 Verilog로[5] 변환한 예제를 사용하였다. 마지막으로 4장에서 결론과 향후 연구 방향을 설명하고 있다.

2. VIS

2.1 순차적 검증

VIS는 Verilog 프로그램에 대한 순차적 일치성 검증 기능을 제공한다. VIS의 순차적 일치성 검증은 BLIF-MV 파일 생성 후 순차적 일치성 검증 순서로 이루어 진다. BLIF-MV파일은 VIS에서 읽을 수 있는 중간 파일(intermediate file)이며 Verilog 프로그램에서 생성된다. VIS는 이를 위해 vl2mv란 이름의 변환 프로그램을 제공한다. 순차적 일치성 검증은 VIS상에서 명령어들을 입력하여 수행된다. 명령어들은 *read_blif_mv <BLIF-MV file1> → flatten_hierarchy → seq_verify <BLIF-MV file2>* 순으로 입력된다. VIS의 검증 결과가 불일치일 경우에는 불일치를 야기하는 입력 값이 출력된다. 그러나 검증 결과가 일치일 경우에는 *"Networks are sequentially equivalent."*라는 문구를 출력한다. 그림 1은 불일치일 경우를 보여주는 예제로서, APR-1400 원자로 보호 시스템의 trip(Shutdown) logic의 일부인 *h_X_Pretrip_Manual.v*과 *h_X_Pretrip_Mech.v*의 행동 일치성을 검증 한다.

```
$ vl2mv h_X_Pretrip_Manual.v
h_X_Pretrip_Manual.v

$ vl2mv h_X_Pretrip_Mech.v
h_X_Pretrip_Mech.v

$ vis
vis release 2.0 (compiled Thu Jun 26 11:08:16     2008)
vis> read_blif_mv h_X_Pretrip_Manual.mv
vis> flatten_hierarchy
vis> seq_verify h_X_Pretrip_Mech.mv

--State 0:
AA.Prev_th_Reset_Ini$NTK2:0
AA.Prev_th_Reset_Ini:0
AA.state$NTK2:A0
AA.state:A0
AA.timer$NTK2:T5
AA.timer:T5
BB.Prev_Pk_State$NTK2:S1
BB.f_X_Prev_PTSP<0>$NTK2:0
BB.f_X_Prev_PTSP<1>$NTK2:0
BB.f_X_Prev_PTSP<2>$NTK2:1
BB.f_X_Prev_PTSP<3>$NTK2:0
BB.f_X_Prev_PTSP<4>$NTK2:0
BB.f_X_Prev_PTSP<5>$NTK2:1
BB.f_X_Prev_PTSP<6>$NTK2:1
BB.f_X_t0<0>$NTK2:1
BB.f_X_t0<0>:1
BB.f_X_t0<1>$NTK2:0
BB.f_X_t0<1>:0
BB.f_X_t0<2>$NTK2:1
BB.f_X_t0<2>:1
BB.f_X_t0<3>$NTK2:1
BB.f_X_t0<3>:1
BB.f_X_t0<4>$NTK2:1
BB.f_X_t0<4>:1
BB.f_X_t0<5>$NTK2:1
BB.f_X_t0<5>:1
BB.f_X_t0<6>$NTK2:1
BB.f_X_t0<6>:1
BB.h_Prev_X_Pretrip_Setpoint<0>:0
BB.h_Prev_X_Pretrip_Setpoint<1>:0
BB.h_Prev_X_Pretrip_Setpoint<2>:1
BB.h_Prev_X_Pretrip_Setpoint<3>:0
BB.h_Prev_X_Pretrip_Setpoint<4>:0
BB.h_Prev_X_Pretrip_Setpoint<5>:1
BB.h_Prev_X_Pretrip_Setpoint<6>:1
BB.state:S0
CC.f_X_Prev<0>$NTK2:1
CC.f_X_Prev<0>:1
CC.f_X_Prev<1>$NTK2:0
CC.f_X_Prev<1>:0
CC.f_X_Prev<2>$NTK2:1
CC.f_X_Prev<2>:1
CC.f_X_Prev<3>$NTK2:1
CC.f_X_Prev<3>:1
CC.f_X_Prev<4>$NTK2:1
CC.f_X_Prev<4>:1
CC.f_X_Prev<5>$NTK2:1
CC.f_X_Prev<5>:1
CC.f_X_Prev<6>$NTK2:1
CC.f_X_Prev<6>:1
DD.state$NTK2:S1
DD.state:S0
DD.th_Prev_X_Pretrip$NTK2:1
DD.th_Prev_X_Pretrip:1
DD.timer$NTK2:T0
DD.timer:T0

--Goes to state 1:
BB.f_X_t0<0>$NTK2:0
BB.f_X_t0<0>:0
BB.f_X_t0<1>$NTK2:1
BB.f_X_t0<1>:1
BB.f_X_t0<2>$NTK2:0
BB.f_X_t0<2>:0
CC.f_X_Prev<0>$NTK2:0
CC.f_X_Prev<0>:0
CC.f_X_Prev<1>$NTK2:1
CC.f_X_Prev<1>:1
CC.f_X_Prev<2>$NTK2:0
CC.f_X_Prev<2>:0

...

--Goes to state 11:
BB.f_X_t0<0>$NTK2:0
BB.f_X_t0<0>:0
BB.f_X_t0<1>$NTK2:0
BB.f_X_t0<1>:0
CC.f_X_Prev<0>$NTK2:0
CC.f_X_Prev<0>:0
CC.f_X_Prev<1>$NTK2:0
CC.f_X_Prev<1>:0
--On input:
<Unchanged>

Networks are NOT sequentially equivalent.
```

그림 1 <순차적 검증 작동 예>

VIS 순차적 일치성 검증은 그림 1에서 알 수 있듯이 자동으로 이루어 진다는 점과 불일치를 야기하는 입력 값을 보여준다는 점에서 소프트웨어의 변경에 따른 검증에 유용하다고 할 수 있다. 하지만 이 기능을 실제 현장에서 활용하기 위해선 아래와 같은 단점을 보완해야 한다.

① 실행 환경이 문자열 기반이라 실행이 번거롭다.
② 출력 결과가 각 단계별로 나와있어 한 눈에 들어오지 않는다.
③ 입력 값에 대한 출력 값을 보여주지 않는다.
④ 각 단계(state)별로 전체 입력, 상태 값이 나오지 않고 변경된 부분만 부분적으로 출력된다.
⑤ 마지막 단계의 입력, 상태 값을 보여주지 않는다.

2.2 시뮬레이션

VIS는 Verilog 프로그램에 대한 시뮬레이션 기능을 제공한다. 본 논문에서 VIS의 시뮬레이션 기능을 사용하는 이유는 입력 값에 대한 상태 변화와 출력 값을 완전히 알 수 있어서 이다. 이와 같은 이유로 VIS시뮬레이션 기능은 VIS 순차적 일치성 검증 결과를 분석하고 보완하는데 효과적이다. VIS의 순차적 일치성 검증 결과는 입력, 상태 값의 일부만 보여주며 출력 값을 보여주지 않는다.

VIS 시뮬레이션의 수행 순서는 BLIF-MV파일 생성→input vector파일 생성→시뮬레이션 실행이다. Input vector파일은 시뮬레이션에 사용 할 입력 값을 저장하고 있는 텍스트 파일이다. 본 논문에서는 VIS 순차적 일치성 검증에서 불일치를 야기한 입력 값을 이 파일에 저장한다. VIS 순차적 일치성 검증 결과는 불일치를 야기한 입력 값을 완전히 보여주지 않으므로 보완을 해야 한다. 보완 방식은 다음과 같다. 일치성 검증 결과에서 *"--On input:"*이하가 *"<Unchanged>"*이면 이전 상태의 입력 값을 사용한다. 그렇지 않으면 *"--On input:"*이하에 명시되어 있는 입력 값을 이전 상태의 입력 값에 적용하여 사용 한다.

시뮬레이션 기능을 사용하기 위해선 VIS상에서 *read_blif_mv <blif-mv file> → flatten_hierarchy → static_order → build_partition_mdds → simulate –i <input vector file>* 순으로 명령어를 입력한다. 시뮬레이션 결과는 Verilog 프로그램의 모든 입력, 상태, 출력 변수 이름과, input vector의 입력 값에 대한 상태, 출력 값을 포함한다. 그림 2는 예제 Verilog 프로그램과, 그림 1에서 추출한 입력 값을 가진 input vector파일을 이용하여 시뮬레이션을 실행 한 결과이다. 예제 Verilog 프로그램은 APR-1400 원자로 보호 시스템의 trip logic의 일부인 *h_X_Pretrip_Manual.v*를 사용하였다.

```
vis release 2.0 (compiled Thu Jun 26 11:08:16     2008)
vis> read_blif_mv h_X_Pretrip_Manual.mv
vis> flatten_hierarchy
vis> static_order
vis> build_partition_mdds
vis> simulate -i inputVector1.txt
# vis release 2.0 (compiled Thu Jun 26 11:08:16     2008)
# Network: main
# Input Vectors File: inputVector1.txt


.inputs  f_X_Raw<0> f_X_Raw<1> f_X_Raw<2> f_X_Raw<3> f_X_Raw<4>
f_X_Raw<5> f_X_Raw<6> temp
.latches AA.Prev_th_Reset_Ini AA.state AA.timer BB.Prev_Pk_State
BB.f_X_Prev_PTSP<0> BB.f_X_Prev_PTSP<1> BB.f_X_Prev_PTSP<2>
BB.f_X_Prev_PTSP<3> BB.f_X_Prev_PTSP<4> BB.f_X_Prev_PTSP<5>
BB.f_X_Prev_PTSP<6> BB.f_X_t0<0> BB.f_X_t0<1> BB.f_X_t0<2>
BB.f_X_t0<3> BB.f_X_t0<4> BB.f_X_t0<5> BB.f_X_t0<6> CC.f_X_Prev<0>
CC.f_X_Prev<1> CC.f_X_Prev<2> CC.f_X_Prev<3> CC.f_X_Prev<4>
CC.f_X_Prev<5> CC.f_X_Prev<6> DD.state DD.th_Prev_X_Pretrip DD.timer
.outputs th_X_Pretrip
.initial 0 A0 T5 S1 0 0 1 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 S1 1 T0

.start_vectors

# f_X_Raw<0> f_X_Raw<1> f_X_Raw<2> f_X_Raw<3> f_X_Raw<4>
f_X_Raw<5> f_X_Raw<6> temp ; AA.Prev_th_Reset_Ini AA.state AA.timer
BB.Prev_Pk_State BB.f_X_Prev_PTSP<0> BB.f_X_Prev_PTSP<1>
BB.f_X_Prev_PTSP<2> BB.f_X_Prev_PTSP<3> BB.f_X_Prev_PTSP<4>
BB.f_X_Prev_PTSP<5> BB.f_X_Prev_PTSP<6> BB.f_X_t0<0> BB.f_X_t0<1>
BB.f_X_t0<2> BB.f_X_t0<3> BB.f_X_t0<4> BB.f_X_t0<5> BB.f_X_t0<6>
CC.f_X_Prev<0> CC.f_X_Prev<1> CC.f_X_Prev<2> CC.f_X_Prev<3>
CC.f_X_Prev<4> CC.f_X_Prev<5> CC.f_X_Prev<6> DD.state
DD.th_Prev_X_Pretrip DD.timer ; th_X_Pretrip

1 0 1 1 1 1 0 0 ; 0 A0 T5  S1 0 0 1 0 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 S1 1 T0  ; 1
1 0 1 1 1 1 0 1 ; 0 A0 T5  S1 0 0 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 1 1 S1 1 T1  ; 1
1 0 1 1 1 1 0 1 ; 1 A1 T0  S0 1 1 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 S1 1 T2  ; 1
1 0 1 1 1 1 0 1 ; 0 A0 T1  S4 1 1 0 0 0 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 S1 1 T3  ; 1
1 0 1 1 1 1 0 1 ; 0 A0 T2  S4 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 1 1 1 S1 1 T4  ; 1
1 0 1 1 1 1 0 1 ; 0 A0 T3  S4 1 1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 S1 1 T5  ; 0
1 0 1 1 1 1 0 1 ; 0 A0 T4  S4 1 1 0 0 0 1 1 1 1 0 1 0 1 1 1 1 0 1 0 1 1 S0 0 T5  ; 0
1 0 1 1 1 1 0 0 ; 0 A0 T5  S4 1 1 0 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 1 S0 0 T5  ; 0
1 0 1 1 1 1 0 0 ; 0 A0 T5  S4 1 1 0 0 0 1 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 S0 0 T5  ; 0
1 0 1 1 1 1 0 0 ; 0 A0 T5  S4 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 1 S0 0 T0  ; 0
1 0 1 1 1 1 0 0 ; 0 A0 T5  S4 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 S0 0 T0  ; 0
1 0 1 1 1 1 0 0 ; 0 A0 T5  S4 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 0 1 1 1 0 1 S0 0 T0  ; 1
# Final State : 0 A0 T5  S4 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 0 1 1 1 0 1 S0 0 T0
```

그림 2 <시뮬레이션 결과>

시뮬레이션의 결과는 Verilog 프로그램의 초기 상태와 입력 값, 상태, 출력 값의 변화를 완전히 보여준다. 그러므로 Verilog 프로그램의 행동을 관찰하는데 유용하다. 하지만 다음과 같은 단점을 가지고 있다.

① 시뮬레이션을 수행하는데 입력해야 하는 명령어들이 많다.

② 모든 입력 값을 별도의 input vector파일을 만들어 사용해야 한다

3. VIS Analyzer

3 장에서는 본 논문에서 제안하는 VIS Analyzer의 기능을 설명한다. 그리고 그 기능의 효율성을 검증하기 위해서 2장에서 사용한 예제를 사용한다.

3.1 VIS Analyzer의 기능

VIS Analyzer는 Windows 기반의 UI를 통해 VIS 순차적 일치성 검증을 자동화 하고 검증 결과를 시각화 한다. 2장에서 확인할 수 있는 바와 같이, VIS 순차적 일치성 검증을 위해서는 BLIF-MV파일을 수동으로 생성해야 하고 *read_blif_mv, flatten_hierarchy* 등의 명령어를 일일이 입력해야 한다. 하지만 VIS Analyzer는 순차적 일치성 검증 과정이 자동으로 실행되며 그에 필요한 파일 또한 자동으로 생성된다. 그리고 VIS Analyzer는 VIS 시뮬레이션을 자동으로 실행하여 VIS 순차적 일치성 검증 결과를 보완한다. 보완된 VIS 순차적 일치성 검증 결과는 표로 시각화되어 출력된다. 2장의 VIS 순차적 일치성 검증 및 시뮬레이션 과정과 VIS Analyzer의 검증 과정을 비교하면 그림 3과 같다.

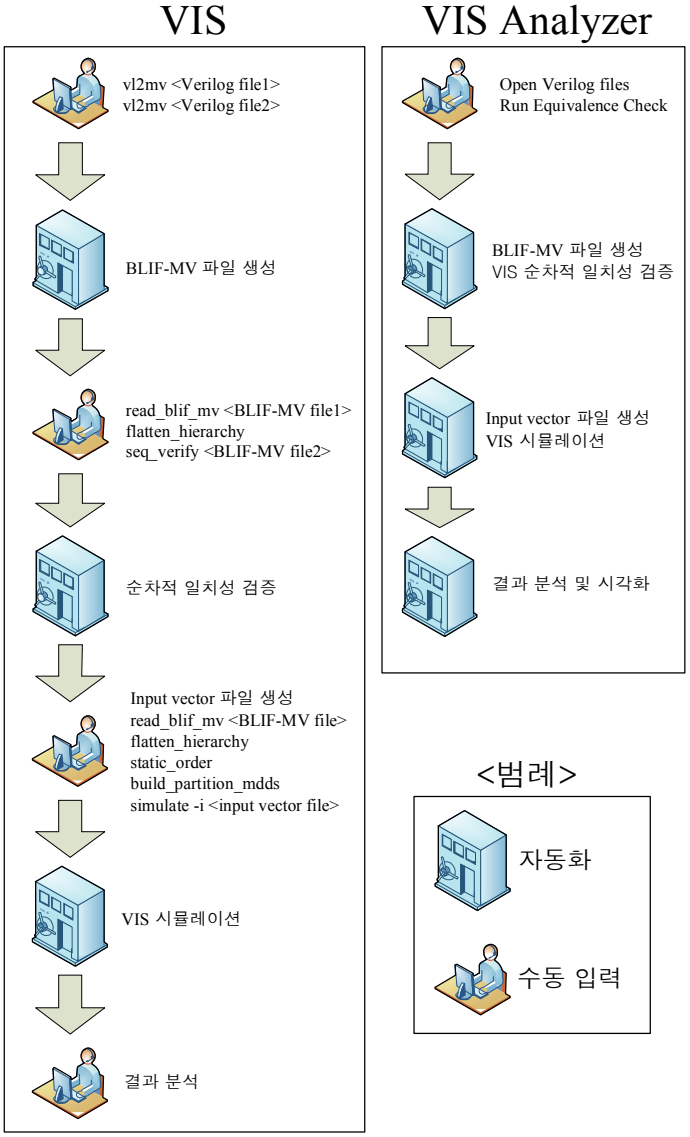

그림 3 <검증 과정 비교>

3.2 VIS Analyzer를 이용한 검증

VIS Analyzer를 이용한 검증 과정은 File메뉴의 *Configuration*(최초 한번) → *File*메뉴의 *Open verilog files* → *Run*메뉴의 *Run VIS equivalence check* 이다. *Configuration*메뉴에서는 VIS와 vl2mv, cygwin의 경로설

정을 한다. 이 경로는 실행되는 컴퓨터 환경마다 다를 수 있으므로 최초 한번 설정해줘야 한다.

그림 4는 예제 Verilog 프로그램을 불러온 후 *Run VIS equivalence check*를 실행하는 화면이다. *Run VIS equivalence check*는 순차적 일치성 검증과 VIS 시뮬레이션을 자동으로 수행한다. 순차적 일치성 검증에 필요한 BLIF-MV파일은 자동으로 생성되며 수행에 필요한 명령어들도 자동으로 입력된다. 검증 수행 시간은 Verilog 프로그램의 복잡도에 따라 차이가 날 수 있다.

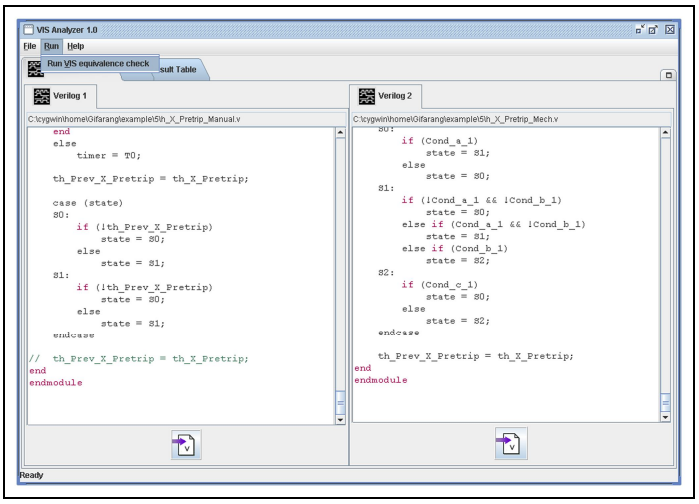

그림 4 <검증 수행>

순차적 일치성 검증 결과는 2.1장에서 밝힌 바와 같이 자세한 정보를 가지고 있지 않다. 그러므로 검증에 쓰인 두 Verilog 프로그램이 어떤 상태변화를 거쳐 불일치를 야기하였는지 관찰하기 어렵다. 그러므로 VIS Analyzer는 순차적 검증 결과로 나온 불일치를 야기하는 입력 값을 시뮬레이션 할 수 있도록 VIS 시뮬레이션 기능을 자동으로 수행한다. 순차적 일치성 검증 결과를 시뮬레이션 한 결과는 두 Verilog 프로그램 간에 불일치를 야기한 입력 값에 대한 출력 값과 상태 변화를 자세히 보여준다. 그러므로 이 정보는 순차적 일치성 검증이 보여주는 정보보다 완전하며 유용하다. 시뮬레이션에 필요한 input vector파일은 순차적 일치성 검증 결과를 시뮬레이션 할 수 있게끔 자동으로 생성된다. 그리고 VIS 시뮬레이션에 필요한 명령어들은 자동으로 입력된다.

최종 검증 결과는 표의 형태로 VIS 시뮬레이션의 결과와 순차적 일치성 검증 결과를 재구성하여 출력된다. 그림 5은 최종 검증 결과 표가 *Result Table*탭에 출력되는 모습을 보여준다. VIS Analyzer는 또한 전체적인 검증 과정을 관찰할 수 있도록 로그를 제공한다. 로그는 순차적 일치성 검증과 시뮬레이트의 실행 결과를 그대로 출력하는 것 이다. 그림 6는 *Result*탭에 출력된 로그를 보여준다.

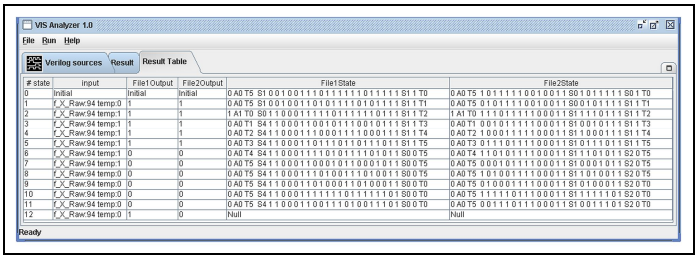

그림 5 <검증 결과>

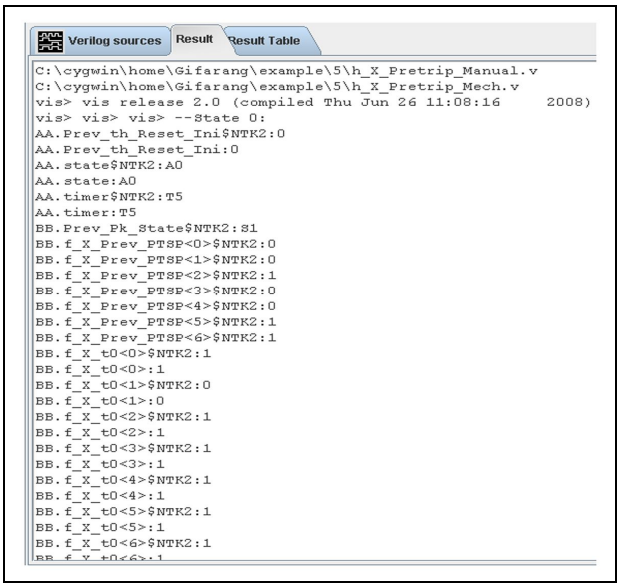

그림 6 <검증 로그>

그림 6에서 확인할 수 있는 바와 같이 불일치를 야기하는 입력 값이 존재할 경우 그 값과 그에 따른 상태, 출력 값 들이 표로 정리되어 출력된다. 입력 값은 10진수 표현으로 변환하여 표시하고 입력 변수 이름을 같이 출력하여 가독성이 높다.

VIS Analyzer는 다음과 같은 장점이 있다.

① GUI를 사용하여 검증 과정이 간편하다.

② VIS 시뮬레이션 기능을 활용해 VIS의 순차적 일치성 검증 결과를 분석하고 더 완전하게 보완한다.

③ 결과를 표의 형태로 보여주어 가독성이 높다.

4. Conclusion & Future Work

안전이 최우선시 되는 프로그램을 개발할 때 정형 기법을 도입하는 것은 이상적이다. VIS는 정형 기법에 활용될 수 있는 도구로서 두 Verilog 프로그램 간의 순차적 일치성을 정형적으로 검증할 수 있는 기능을 제공한다. 하지만 VIS는 실행 환경과 검증 결과가 가지고 있는 문제점 때문에 실제 활용엔 어려움이 많다. VIS Analyzer는 VIS가 가지는 어려움들을 VIS 시뮬레이션 기능의 활용과 GUI 및 표를 이용한 출력으로 극복하고자 하였고 예제를 통한 실험에서 효율성을 검증하였다. 그러므로 VIS Analyzer는 VIS를 활용한 정형 검증에 효과적이다. 앞으로의 연구 과제는 다음과 같다.

① VIS가 제공하는 CTL모델 체킹을 효율적으로 실행하고, 결과를 시각화 할 수 있는 기능을 구현한다.
② 조합적 Verilog프로그램에 대한 일치성 검증을 효율적으로 실행하고, 결과를 분석해 시각화 하는 기능을 구현한다.
③ VIS 순차적 일치성 검증 결과에 보여지지 않는 마지막 상태의 입력 값을 알아내어 검증 결과를 완전히 하는 기능을 구현한다.

참고 문헌

[1] N.G. Leveson. *SAFEWARE, System Safety and Computers.* Addison-Wesley, 1995.
[2] Doron A. Peled. *SOFTWARE RELIABILITY METHODS.* Spinger, 2001.
[3] Junbeom Yoo, Sungdeok Cha, Eunkyoung Jee. *"Verification of PLC programs written in FBD with VIS"*, accepted to *Nuclear Engineering and Technology,* 2008.
[4] VIS (Verification Interacting with Synthesis), *http://embedded.eecs.berkeley.edu/research/vis/.*
[5] D. E. Thomas and P. R. Moorby. *The Verilog Hardware Description Language.* Kluwer Academic Publishers, 1991.
[6] KNICS, *http://www.knics.re.kr/english/eindex.html.*