

# ModelSim 튜토리얼

## 시작하기

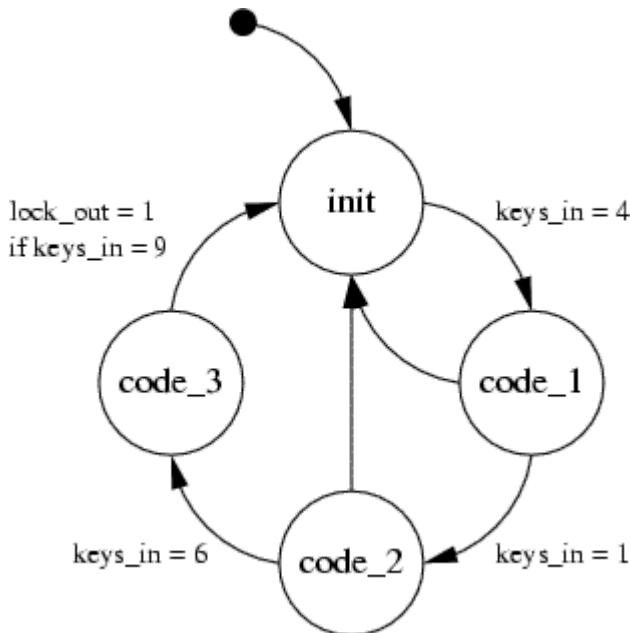
- [소개](#)
- [1. 초기화](#)
  - [a\) 리눅스](#)
  - [b\) 솔라리스](#)
  - [1.1 라이브러리 생성](#)
- [2. 디자인 컴파일](#)
- [3. 시뮬레이션](#)
  - [3.1 설계로드](#)
  - [3.2 시뮬레이션 준비](#)
  - [3.3 시뮬레이션 실행](#)
  - [3.4 디자인 디버깅](#)
- [4. 요약](#)

## 소개

ModelSim은 Mentor Graphics의 사용이 간편하면서도 다양한 VHDL / (System) Verilog / SystemC 시뮬레이터입니다. 동작, 레지스터 전송 레벨 및 게이트 레벨 모델링을 지원합니다. ModelSim은 퍼베이시브 컴퓨팅 부서 ( 예 : Linux, Solaris 및 Windows) 및 기타 여러 플랫폼에서 사용되는 모든 플랫폼을 지원합니다 . Linux 및 Solaris 플랫폼에서 ModelSim은 부서의 컴퓨터에 사전 설치되어 있습니다 ( [Linux](#) / [Mustatikli](#) 참조 ). 그러나 Windows 사용자는 스스로 설치해야합니다.

이 학습서는 ModelSim 시뮬레이터에 대한 경험이 없는 사용자를 위한 것입니다. ModelSim 시뮬레이터를 설정하고, ModelSim SE로 설계 및 시뮬레이션 기초를 컴파일하는 방법에 대한 기본 흐름을 소개합니다. 이 자습서에 사용 된 예는 VHDL로 작성된 작은 디자인이며 이 자습서에서는 가장 기본적인 명령만 다룹니다. 이 학습서는 Linux에서 ModelSim SE 버전 6.1b를 사용하여 작성되었습니다.

이 학습서에 사용 된 예는 키패드에서 4 자리 PIN ( 4169 ) 코드를 입력하여 잠금을 해제 할 수 있는 전자 잠금을 설명하는 간단한 디자인입니다. 잠금 장치가 올바른 입력 순서를 감지하면 도어 잠금을 해제하기 위한 신호로 한 클럭주기 동안 출력을 높게 설정합니다. 아래 그림은 디자인의 상태 머신을 보여줍니다. 디자인에는 실제 의미는 없지만 ModelSim에서 디버그 메소드를 시연하는 데 사용되는 하나의 더미 변수 ( `count_v` )가 포함됩니다.



잠금 상태 머신 다이어그램.

재설정되면 상태 머신은 상태 'init'에서 시작합니다. 상태는 keys\_in 버스에서 4 번을 읽고 상태 'code\_1'로 이동할 때까지 기다립니다 . 그리고, 다음의 두 숫자 경우 keys\_in 버스 1 및 6의 상태 '를 통해 상태 머신이 진행되어 code\_2 상태'에서 'code\_3'; 그렇지 않으면 상태 'init'로 돌아갑니다 . 상태 'code\_3'에서 상태 머신은 항상 'init' 상태로 돌아갑니다 . keys\_in 의 코드가 버스는 9이고, 상태 머신은 또한 한 클럭주기 동안 출력력을 높게 설정합니다.

## 1. 초기화 (Linux / Solaris 만 해당)

이 섹션에서는 디자인을 위한 작업 디렉토리와 디자인이 컴파일 된 디자인 라이브러리를 만듭니다. 그러나 ModelSim을 사용하려면 환경을 설정해야합니다. 여기에는 몇 가지 변수, 파일 및 디렉토리 설정이 포함되며 아래에 제공된 스크립트를 소싱하여 수행 할 수 있습니다. (파일 이름의 자동 완성이 반드시 디렉토리 / share에서 작동하지는 않습니다)

Windows 사용자는 이 단계를 건너 뛰고 [라이브러리 작성](#) 섹션으로 바로 이동할 수 있습니다 . Windows에서 ModelSim을 사용하기 전에 실행할 스크립트가 없습니다.

### a) 리눅스

Linux에서 쉘 창에 다음을 작성하여 환경을 설정하십시오. 파일 경로 완성이 반드시 / share에서 작동하는 것은 아닙니다 (이유는 약간 불분명합니다).

```
$ 소스 /share/tktprog/mentor/modeltech-6.3d/modeltech.sh
```

### b) 솔라리스

[먼저이](#) 링크 ([Mustatikli](#))를 확인 하여 Solaris에서 현재 사용 가능한 최신 버전의 ModelSim을 찾으십시오. 선택한 페이지에 제공된 명령을 사용하여 소스를 제공하십시오. 특정 버전을 사용해야 할 이유가 없는 한 사용 가능한 최신 버전을 선택하십시오.

```
$ 소스 /opt/mentor/modeltech-6.3a/modeltech.sh
```

지정된 스크립트를 소싱 한 후에는 소스 스크립트를 읽고 환경이 올바르게 설정되었음을 나타내는 다음과 유사한 메시지가 표시됩니다.

```
#####
##### ModelSim SE 버전 6.3a #####
#####
```

## 1.1 라이브러리 생성

먼저, 모든 프로젝트 (프로젝트 파일 및 디렉토리 등)를 다른 프로젝트 및 파일과 깔끔하게 정리하고 분리하려면 프로젝트 디렉토리가 없는 경우 프로젝트 디렉토리를 작성하십시오.

```
$ mkdir ex1_tutorial  
$ cd ex1_tutorial
```

그런 다음 새로 작성된 프로젝트 디렉토리에서 설계 라이브러리를 작성하고 맵핑하십시오. 설계 라이브러리는 ModelSim이 컴파일 된 설계 단위를 저장하는 라이브러리입니다. ModelSim이 설계 라이브러리를 찾을 수 있도록 맵핑이 필요합니다. 기본 디자인 라이브러리는 VHDL 파일에서 작업이라고 합니다. 기호 이름은 기본적으로 디렉토리 ./work에 매핑됩니다. 여기서는 이를 my\_lib 디렉토리에 맵핑 합니다.

```
$ vlib my_lib # 자신 만의 디자인 라이브러리 만들기 ...  
$ vmap work $ PWD / my_lib # ...  
/work/tktprog/mentor/modeltech-6.1b/linux/.../modelsim 복사하기 modelsim.ini의 ini 모델 im.ini  
수정
```

보시다시피 ModelSim은 my\_lib 디렉토리를 현재 디렉토리에 만들고 \_info 파일을 만듭니다. 이 파일을 직접 편집해서는 안됩니다. ModelSim은 모든 설계 단위와 설계 상태를 추적하기 위해 사용합니다. 시뮬레이터에 대한 디자인 라이브러리의 목차처럼 생각할 수 있습니다.

맵핑에서 ModelSim은 현재 디렉토리 (이 경우 ex1\_tutorial)에 modelsim.ini라는 파일을 복사하고 라이브러리 섹션을 수정합니다. ModelSim이 호출되면 파일을 읽고 해당 맵핑을 사용하여 설계 라이브러리를 찾습니다. 텍스트 편집기에서 현재 라이브러리 맵핑을 열거나 인수없이 vmap 명령을 호출하여 modelsim.ini의 내용을 확인할 수 있습니다.

```
$ vmap  
읽기 modelsim.ini  
"work"는 my_lib 디렉토리에 매핑됩니다.  
/share/tktprog/mentor/modeltech-6.1b/linux/.../modelsim.ini  
"std" 읽기는 /share/tktprog/mentor/modeltech-6.1b/linux/.../std 디렉토리에 매핑됩니다.  
"ieee"는 /share/tktprog/mentor/modeltech-6.1b/linux/.../ieee 디렉토리에 매핑됩니다.  
...
```

modelsim.ini에 올바르게 맵핑 되고 ModelSim에서 찾을 수 있는 한 파일 시스템의 원하는 위치에 원하는 수의 디자인 라이브러리가 있을 수 있습니다. 예를 들어 할당량을 저장하기 위해 / tmp / \$ USER /에 배치 할 수 있습니다. 소스 코드를 tmp에 저장하지 마십시오.

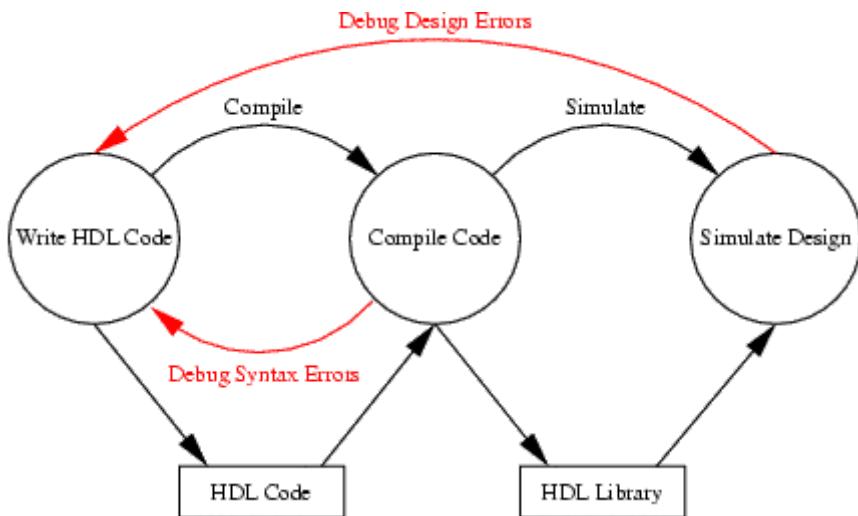
마찬가지로 소스 파일을 여러 위치에서 동일한 디자인 라이브러리로 컴파일 할 수 있습니다.

## 2. 디자인 컴파일

이제 환경을 설정하고 프로젝트 디렉토리 ex1\_tutorial 과 디자인 라이브러리 my\_lib를 만들었습니다. 시뮬레이터는 VHDL (또는 Verilog) 소스 코드 자체를 읽을 수 없으므로 다음 단계는 디자인을 디자인 라이브러리로 컴파일하는 것입니다. VHDL은 VHDL 파일이 컴파일되는 순서에 따라 엄격

합니다. 즉, 다른 파일 (예 : 패키지)에서 참조하는 파일은 파일을 참조하는 파일 (예 : 해당 패키지를 사용하는 파일)보다 먼저 컴파일되어야합니다.

컴파일하는 동안 모든 구문 버그가 지적됩니다. 성공적인 컴파일 후 실제 시뮬레이션 / 디버깅이 시작될 수 있습니다. 딜레이 사이클이 아래 그림과 같이 편집-컴파일-시뮬레이션-편집 ... 과 같은 것을 보게 될 것 입니다.



일반적인 설계주기. HDL 파일이 작성된 후에는 컴파일되고 구문 오류가 수정됩니다. 그런 다음 컴파일 된 코드가 시뮬레이션되고 기능 오류가 디버깅됩니다.

먼저 프로젝트 디렉토리 (`ex1_tutorial/`)에 소스 파일을 위한 별도의 디렉토리를 만들고

`mkdir vhd`

둘째, 다음 VHDL 파일을 다운로드하십시오 ( 오른쪽 마우스 버튼-> 다른 이름으로 저장 또는 인수로 아래 링크와 함께 \$ wget 사용 )

- lock\_pkg.vhd
  - lock.vhd

이제 소스를 컴파일 할 차례입니다. 합성 할 수 없는 구조를 식별 하려면 명령 행 옵션 '`-check_synthesis`'를 사용하십시오 . 있습니다 `-check_synthesis`의 옵션이 몇 가지 가장 일반적인 합성 규칙에 대한 최소한의 준수를 확인합니다 (그러나 반드시 모든).

```
$ VCOM -check_synthesis VHD / lock_pkg.vhd  
$ VCOM -check_synthesis VHD / lock.vhd  
또는  
$ VCOM -check_synthesis VHD / lock_pkg.vhd의 VHD / lock.vhd
```

위에 표시된 것처럼 하나의 명령으로 소스 파일을 하나씩 또는 모두 컴파일 할 수 있습니다. 어쨌든 올바른 순서로 컴파일해야합니다. 파일 lock.vhd는 패키지 파일을 사용하므로 마지막에 컴파일해야합니다.

파일이 많은 프로젝트에서 작업하는 경우 모든 파일의 컴파일 관리가 너무 번거로워집니다 (즉, 하나의 패키지 파일을 편집하면 다른 파일의 재 컴파일이 필요할 수 있으며 다른 파일의 재 컴파일이 다시 필요함) 등 ). UNIX에서는 이러한 프로세스를 `make`를 사용하여 자동화 할 수 있습니다 . 파일과 파일 업데이트 방법 사이의 관계를 설명하는 `Makefile` 이라는 파일이 필요 합니다. 이러한 파일이 있으면 `make`를 호출 하여 필요한 설계 단위 ( 즉, 수정 한 설계 단위 및 재 컴파일 된 설계 단위에 종속된 것) 만 다시 컴파일 할 수 있습니다..

운좋게도, 당신은 makefile을 작성하는 것에 대해 아무것도 모른다. ModelSim 은 이를 위한 도구를 제공합니다 : vmake . vmake 는 명령 줄에 지정된 디자인 라이브러리에 대한 make 파일을 자동으로 만듭니다. 참고 vmake가 해당 라이브러리를 위한 메이크 파일을 생성하기 위해 설계 라이브러리에서 발견 된 정보를 사용하여 (즉, 당신이 호출하기 전에 위의 그림과 같이 한 번 수동으로 모든 파일을 컴파일해야합니다 vmake를 메이크 파일을 만듭니다).

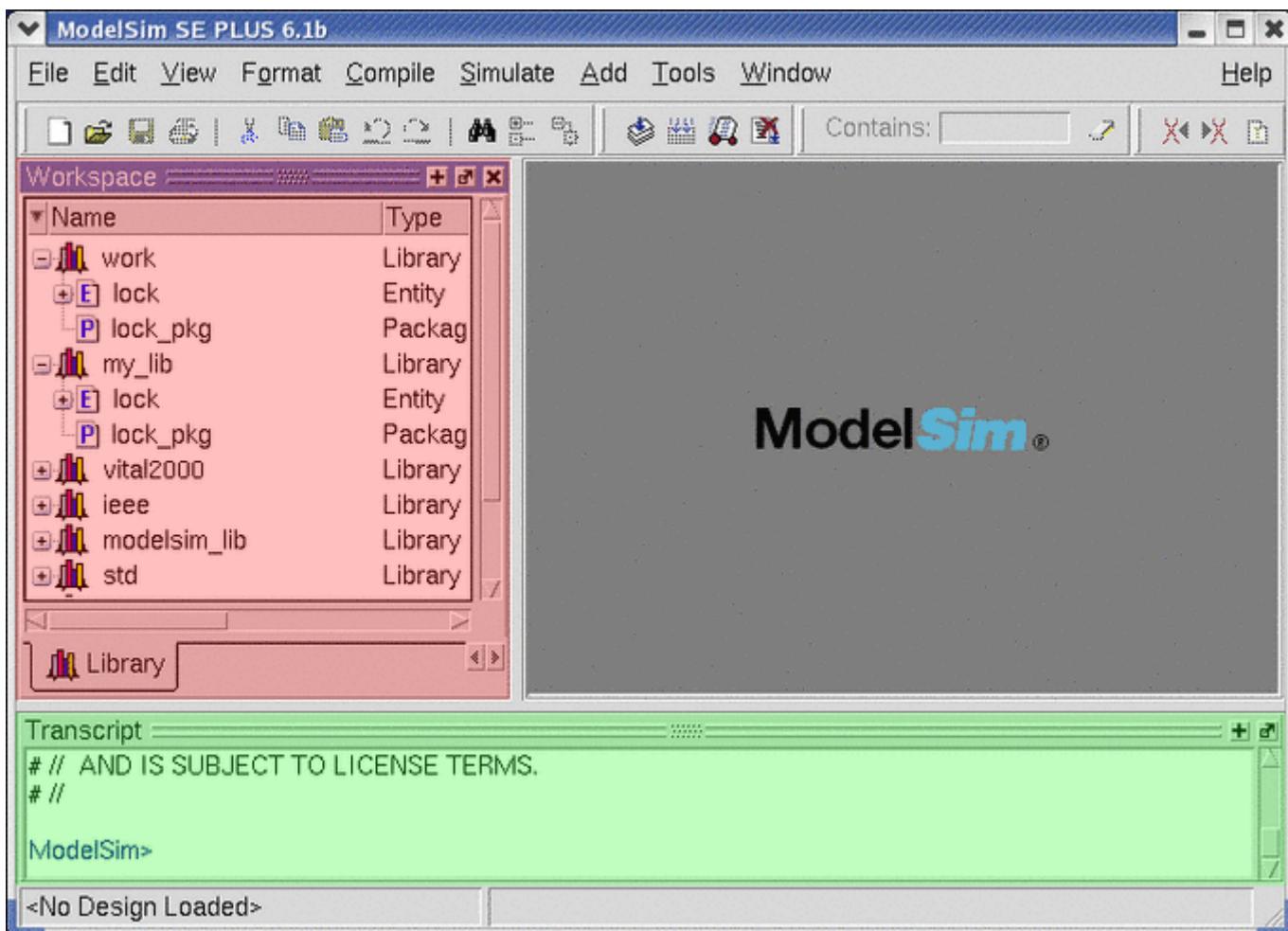
```
$ vmake my_lib> 메이크 파일
```

이제 make 만 호출하면 디자인 라이브러리를 최신 상태로 유지할 수 있습니다 . 디자인에 새 파일을 추가해야하는 경우 파일을 디자인 라이브러리로 컴파일 한 다음 make 파일을 재생성하여 수행 할 수 있습니다. 마찬가지로 vdel 명령을 사용하여 파일을 제거하고 (디자인 라이브러리에서 파일을 제거) make 파일을 다시 생성 할 수 있습니다. **노트!** 알 수없는 이유로 Vmake 명령이 Windows에서 작동하지 않는 것 같습니다.

### 3. 시뮬레이션

이제 디자인을 시뮬레이션 할 차례입니다. ModelSim 시뮬레이터는 vsim 명령으로 호출됩니다 . 기본 창은 아래 그림과 같습니다. 처음 시작할 때 시작 창이있을 수도 있습니다.

```
$ vsim &
```



기본 시뮬레이터 창. 빨간색은 작업 영역 창과 녹색 대화 창을 나타냅니다.

위의 그림 은 작업 공간 창에서 열린 라이브러리 작업 및 my\_lib 라이브러리를 보여줍니다 . 라이브러리 때문에 my\_lib이 라이브러리 (기본값으로 매핑 된 일 ) 동안 my\_lib이 생성, 이 두 라이브러리는 하나의 동일한 라이브러리는 사실이다.

대화 창에는 시뮬레이터 ( 예 : 시뮬레이터에서 발생한 오류 또는 설계 / 테스트 벤치에서 인쇄 한 메시지)와 디자이너 ( ModelSim> 프롬프트에 입력 한 명령 ) 사이의 메시지가 표시됩니다. 성적 창에서 메시지는라는 파일에 기록 된 성적 증명서 .

다른 창에는 파형을 볼 수있는 Wave 창, Dataflow 창, 소스 창 등이 있습니다. 필요한 경우 메뉴 표시줄 ( 보기 )에서 찾을 수 있습니다 .

## 3.1 설계로드

가장 먼저 할 일은 설계를 시뮬레이터에로드하는 것입니다. ModelSim이 호출 된 후 일반적으로 명령을 실행하는 두 가지 또는 세 가지 방법이 있습니다. 명령은 항상 스크립트 창에서 ModelSim 프롬프트에 쓸 수 있습니다. 일부 명령은 도구 모음 단추 또는 메뉴 모음 대화 상자에서도 찾을 수 있습니다.

예를 들어, 라이브러리 my\_lib 에서 디자인 ' lock ' 을로드하려면 다음을 수행 할 수 있습니다.

- 작업 공간 분할 창에서 my\_lib 라이브러리를 열고 lock 엔티티를 두 번 클릭하십시오.
- 메뉴 바에서 시뮬레이션-시뮬레이션 시작 ... 을 선택한 다음 위와 유사하게 시뮬레이션하려는 라이브러리와 엔티티를 선택하십시오.
- ModelSim 프롬프트에서 vsim my\_lib.lock 을 작성하십시오 (또는 명령 행에서 해당 명령으로 시뮬레이터를 시작하십시오).

이 학습서에서는 대부분 스크립트 파일에 기록 될 때 스크립트 창에 명령을 작성합니다 . 시뮬레이션을 마치면 스크립트 파일을 사용하여 후속 시뮬레이션에서 재사용 할 수있는 매크로 파일을 만들 수 있습니다.

ModelSim> vsim my\_lib.lock

명령 줄 인수를 제공 할 수도 있지만이 자습서에서는 필요하지 않습니다. 그러나 최신 Modelsim 버전은 때때로 너무 탐욕스럽게 최적화되므로 모든 신호를 볼 필요는 없습니다. 이 경우 최적화를 비활성화하십시오. vsim -novopt my\_lib.lock

## 3.2 시뮬레이션 준비

시뮬레이터에 설계를로드 한 후 시뮬레이션을 시작하기 전에 설계에 대한 시뮬레이션 / 디버그 환경을 설정해야합니다. 즉, 중요하다고 생각되는 디버그 창을 열고 해당 디버그 창에서 추적하려는 신호 및 변수 등 을 선택해야합니다.

디버그 창을 처음 시작할 때 Wave, Source, Locals (변수) 및 Objects (신호) 창을 선택하는 것이 좋습니다. 풀다운 메뉴 보기 또는 명령 줄에서 열 수 있습니다.

웨이브 - 파형 참조

사물 - 신호와 그 값을보십시오

출처 - 코드를 한 줄씩 추적

현지인 - 변수 및 해당 값보기

VSIM는> 뷰 객체

VSIM> 뷰 주민을

VSIM> 소스보기

VSIM> 별도 창으로보기 표면파 -undock 번호 분리 웨이브 (해제)

다음으로, Wave 창에서 추적하려는 신호를 시뮬레이터에 알려야합니다. 신호가 많으면 관심있는 신호만 선택하는 것이 좋습니다. 그러나이 경우 모든 신호를 추적합니다.

VSIM> add wave \* # Wave 창의 Objects 창에 있는 모든 신호를 표시합니다

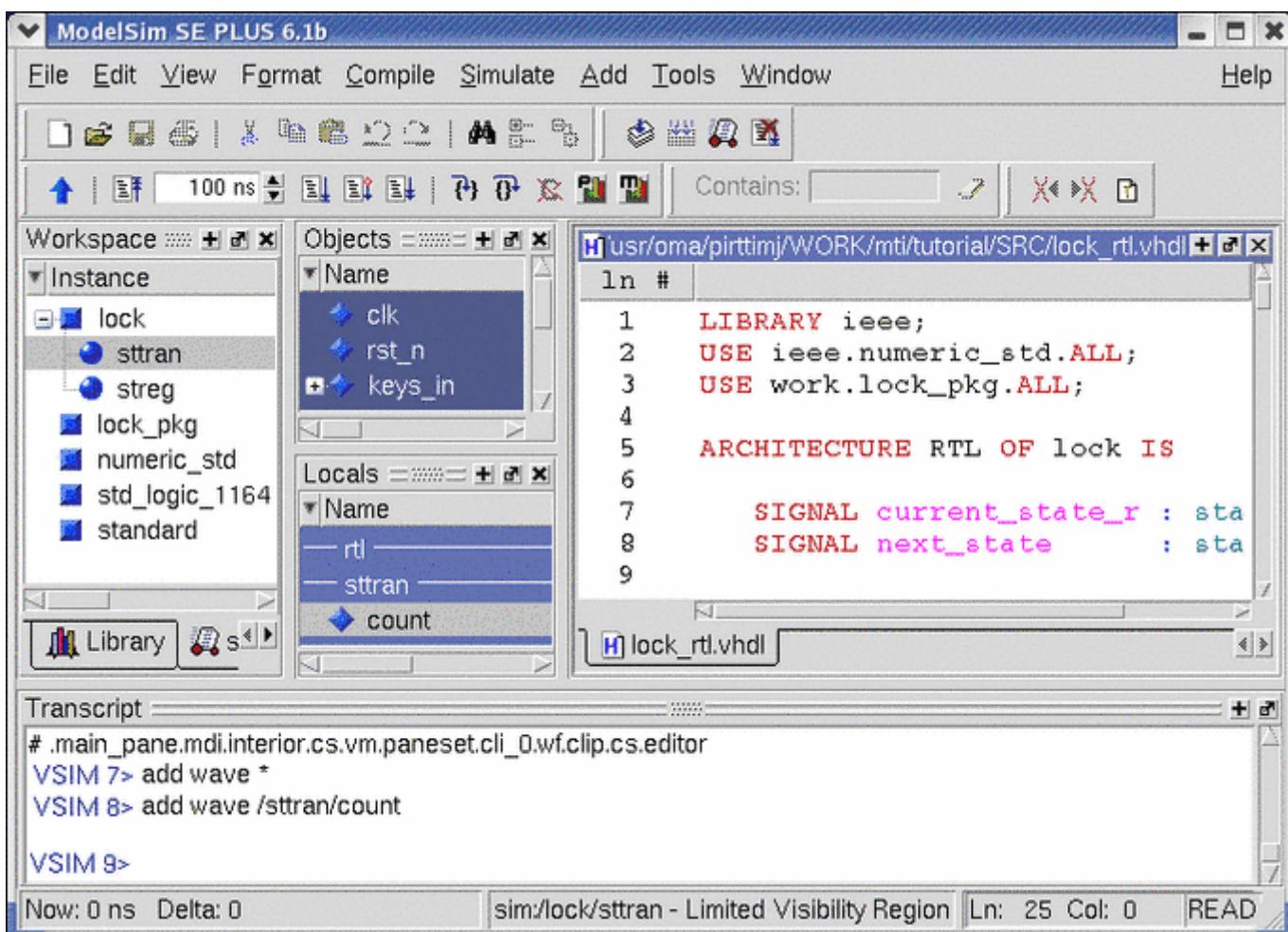
Wave 창에 신호를 추가 할 수도 있습니다.

- 객체 패널 ( Shift + 왼쪽 마우스 )에서 신호를 선택하고 선택을 웨이브 패널로 드래그 앤 드롭 합니다 .
- Objects 창에서 마우스 오른쪽 버튼을 클릭하고 Add to Wave를 선택 합니다.
- 웨이브에 추가 대화 상자는 메뉴 막대 ( 추가 )에서도 찾을 수 있습니다 .

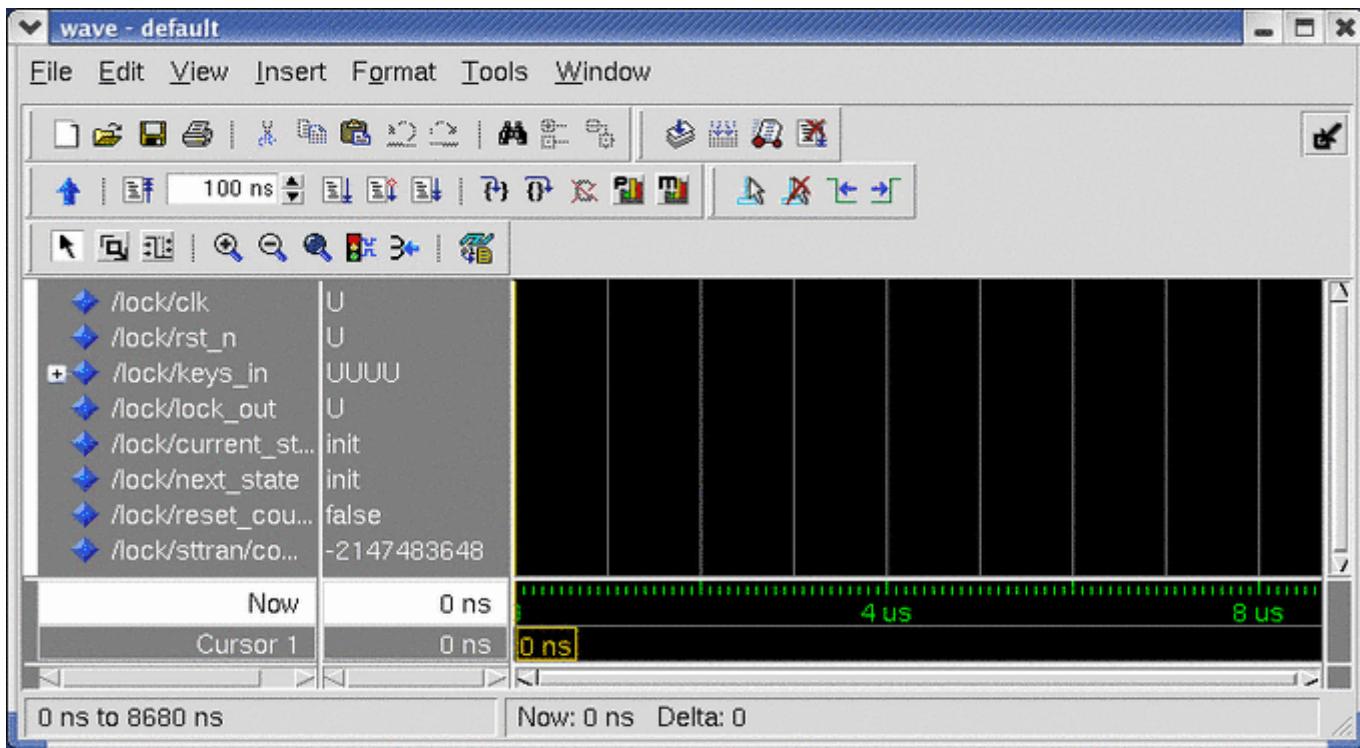
Wave 창에서 추적 할 변수 ( count\_v )도 있습니다.

VSIM> 웨이브 추가 / sttran / count\_v

이제 모든 것이 아래와 같이 보일 것입니다 :



Workspace, 오브젝트 (신호), 로컬 (변수) 및 소스 분할 창을 표시하는 ModelSim 기본 창 . 이 디자인에는 두 개의 프로세스 ( sttran 및 streg )가 있으며 작업 공간 분할 창에 표시됩니다 . 지역 분할 창은 현재 작업 공간 분할 창 ( sttran )에서 선택된 프로세스의 변수 ( count\_v )를 표시합니다 .



웨이브 창. 신호 및 변수에 대해 표시된 값은 여전히 초기화되지 않았습니다.

File-> Save ...를 선택하여 웨이브 윈도우의 현재 신호 구성을 저장하는 것이 매우 유리합니다 .

시뮬레이션을 시작하기 전에해야 할 일이 하나 더 있습니다. 디자인에 대한 입력 자극을 생성해야 합니다. 설계에는 비동기식, 능동형 로우 리셋 (rst\_n), 시스템 클록 (clk) 및 키패드의 버스 (keys\_in) 등 세 가지 입력이 있습니다 . 시뮬레이션 시작을 위해 시스템 클록 (50MHz 및 50/50 비율), 리셋 (45ns 동안 활성화) 및 입력 버스를 0 (0000)으로 설정 한 입력 자극을 생성합니다.

```
VSIM> force -deposit / rst_n 00, 1 {45 ns}
VSIM> force -deposit / clk 100, 0 {10 ns} -repeat 20 VSIM
> force -deposit / keys_in 0000 0
```

마지막 라인은이 세 라인을 이해하기가 가장 쉬운데 , 0 ns에서 keys\_in bus의 값을 0000으로 설정하기 만하면 됩니다. 첫 번째 라인은 약간 더 복잡하지만은 리셋 신호 생성 0 시간에 0을 NS 및 높은 이동 45 NS . 두 번째 라인은 시스템 클럭을 생성합니다. 먼저, 클록은 0 ns ( 1 0 )로 높게 설정 되고 10 ns 이후에는 낮아집니다 ( 0 {10 ns}). 이것은 클럭 사이클의 전반부입니다. 클럭 사이클의 후반 동안 클럭은 다시 변경 될 때까지 낮게 유지됩니다. 명령이 20 ns 마다 반복 / 시작되기 때문에 (-repeat 20 ),이 명령은 클럭 사이클의 전반에 대해 높고 후반에 대해 낮고주기가 20 ns 인 클럭 신호를 생성합니다..

참고 : 기본 시간 단위 (해상도)는 modelsim.ini에 정의되어 있으며 ns 입니다. 따라서 시간 단위를 모두 생략하고 다음과 같이 두 개의 첫 번째 명령을 작성할 수 있습니다 (-repeat 도 -r 로 쓸 수 있음 ).

```
VSIM> 강제 -deposit / rst_n 0 0, 1 45
VSIM> 강제 -deposit / clk 1 0, 0 10 -r 20
```

이제 디자인의 첫 번째 시뮬레이션을 수행 할 준비가되었습니다.

참고 : 실제 상황에서는 클럭 신호가 테스트 벤치에서 더 쉽게 할당되고 손으로 지정되지 않습니다. 그러나 경우에 따라 이러한 명령이 유용합니다.

### 3.3 시뮬레이션 실행

모든 입력은 초기 값으로 설정되었으며 모든 것이 시뮬레이션 준비가되었습니다. 시뮬레이션을 실행하려면 실행 버튼을 누릅니다. Run 버튼은 ModelSim Main 및 Wave 창과 메뉴 표시 줄 (Simulate-> Run-> Run 100ns)에서 찾을 수 있습니다.



ModelSim Main / Wave Window Toolbar의 Run 버튼.

항상 그렇듯이 ModelSim 프롬프트에서 동일한 명령을 작성할 수도 있습니다.

VSIM> 실행 <시간>

이제 수백 나노초 동안 시뮬레이션을 실행하십시오. 이전 단계에서 설정 한 초기 값을 시작할 때 각 신호가 어떻게 걸리는지, 그리고 몇 번의 클럭주기 후에 리셋이 어떻게 비활성화되는지 확인해야 합니다. 다음으로 올바른 PIN 코드 (4169)를 입력하여 디자인이 어떻게 반응하는지 확인합니다.

경고와 같은 일부 디자인은 0 ns에서 막대한 추악한 경고를 생성 합니다. 산술 피연산자에 'U' | 'X' | 'W' | 'Z' | '-'가 있으면 결과는 ' X '(들). 정의되지 않은 'U'와 같이 입력이 실수가 아닌 숫자 변환 함수로 인해 더 중요한 경고를 찾기가 더 어려워집니다. 다음과 같이 시뮬레이션을 시작하여 제거 할 수 있습니다.

```
vsim mylib.lock
set StdArithNoWarnings 1
실행
0ns 설정 StdArithNoWarnings 0
실행 100
```

경고가 다시 활성화됩니다. 이러한 경고를 영구적으로 비활성화하기 위해 `modelsim.ini` 를 편집 할 수도 있지만 일반적으로 0ns 이후에 경고가 표시되는 경우를 참조하십시오.

### 3.3.1 입력 생성

키 누름을 시뮬레이션하려면 각 클럭 사이클에서 `keys_in` 입력을 변경해야 합니다.

```
VSIM> force -deposit / keys_in 2 # 0100 # keys_in bus를 4로 설정 ... VSIM
> run 20 # ... 한 클럭주기 동안 실행
VSIM> force -deposit / keys_in 2 # 0001 # 1 VSIM
> run 20
VSIM> force -deposit / keys_in 2 # 0110 # 6 VSIM
> 실행 20
VSIM> force -deposit / keys_in 2 # 1001 # 9
VSIM> run 20 # lock_out은 이제 활성화되어야 합니다
VSIM> force -deposit / keys_in 2 # 0000 # 0 ("버스 비우기")
VSIM> 실행 100
```

위와 같이 입력을 입력하는 것은 물론 지루한 작업입니다. 먼저, 각 사이클에 대해 '`force -deposit ...`' 를 쓴 다음 시뮬레이션을 한 클록 사이클 앞으로 실행해야합니다. 둘째, 입력을 이진 형식으로 입력해야합니다. 약간의 비트로는 가능하지만 32 비트 이상에서는 그리 실용적이지 않습니다.

물론 위의 명령을 작성하는 대안이 있고 더 빠른 방법이 있습니다.

```
VSIM> force -deposit / keys_in 10 # 4, 10 # 1 20, 10 # 6 40, 10 # 9 60, 10 # 0 80
VSIM> 실행 200
```

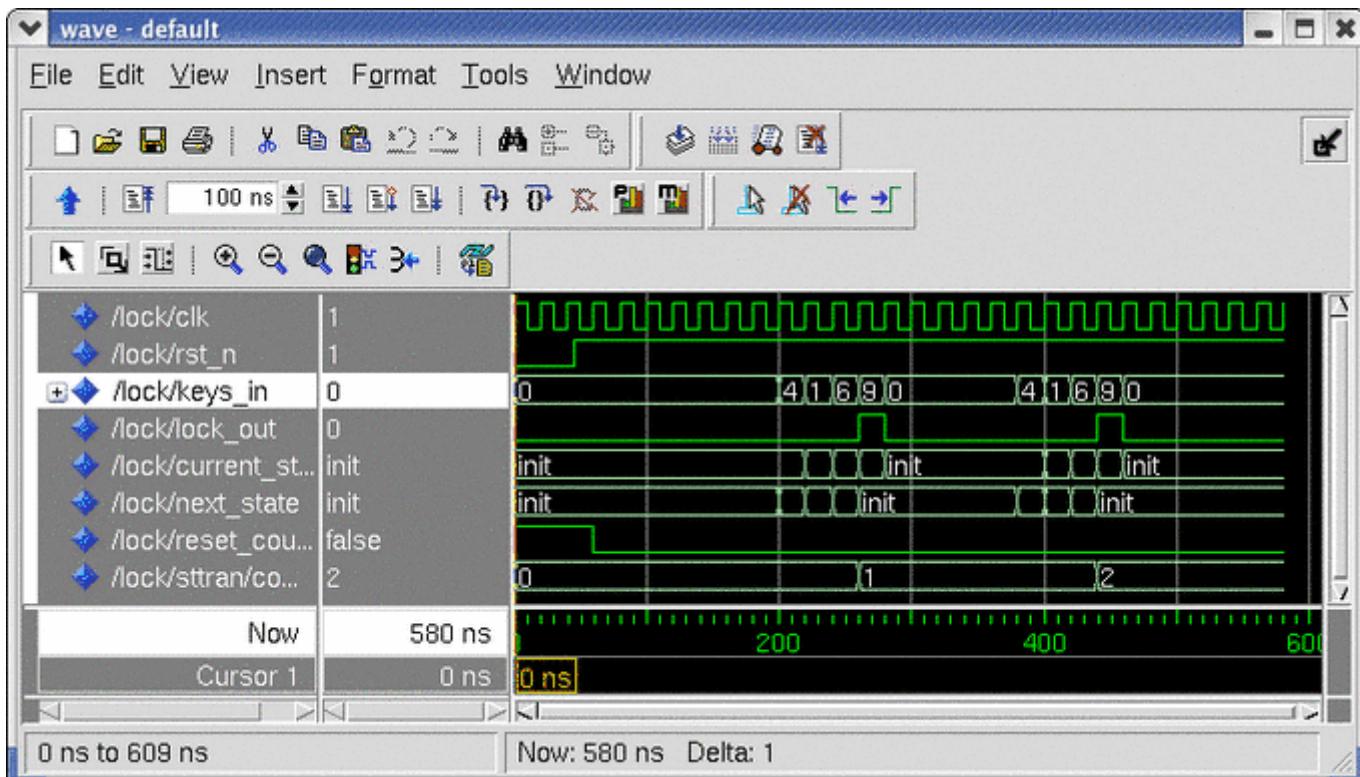
위의 줄은 간단히 말해 : `keys_in` 버스를 지금 순간부터 시작하여 원하는 값 4 (`10 # 4`)로 변경하십시오 . 그런 다음 20ns 후 입력을 1 (`10 # 1 20`), 40ns 후 6 (`10 # 6 40`), 60ns 후 9 (`10 # 9 60`),

마지막으로 80ns 후 0 ( 10 # 0 80 ). 모든 것이 한 줄로 이루어지며 입력을 십진수 형식으로 입력하는 방법도 보여줍니다. 사용 가능한 다른 근사값은 2 진 (2 #), 8 진 (8 #) 및 16 진 (16 #)입니다.

명확성을 위해 때때로 파형 창에 표시된 일부 신호의 기수를 변경하는 것이 좋습니다. 당신이 당신의 디자인에 넓은 버스가 있는 경우 예를 들어, 데이터가 표시되면 그 버스에 값을 확인하기 쉬울 것입니다 예 16 진수 형식이 아닌 바이너리 형식으로. 신호의 기수 변경은 Wave 창에서 신호를 마우스 오른쪽 버튼으로 클릭하고 대화 상자에서 기수를 선택한 다음 기수를 선택하여 수행 할 수 있습니다 . 이 경우 올바른 기수가 부호가 없습니다.

```
VSIM> 속성 파-기수 부호 없음 / lock / keys_in
```

이제 Waveform 창은 다음과 같아야합니다.



첫 번째 시뮬레이션 실행 결과. keys\_in 버스의 기수 가 서명되지 않은 것으로 변경되었습니다 .

디자인이 제대로 작동하는 것 같지만 지금까지 입력 시퀀스는 하나만 테스트했습니다. 이제 몇 가지 다른 입력 시퀀스를 시도하여 디자인이 어떻게 처리되는지 확인합니다.

먼저 랜덤 시퀀스 ( 25416918 )에 포함 된 올바른 시퀀스를 포함하는 랜덤 입력 시퀀스를 시도합니다 .

```
VSIM> force -deposit /keys_in 10#2, 10#5 20, 10#4 40, 10#1 60, 10#6 80, 10#9 100, 10#1 120, 10#8  
140, 10#0 160  
VSIM> run 300
```

보시다시피, 올바른 코드를 포함하는 긴 임의의 입력 시퀀스가 있으면 코드가 승인되고 도어 잠금이 해제됩니다. 이것이 원하지 않는 잠금 기능인 경우 시스템을 다시 설계해야합니다.

다음으로 입력 시퀀스로 부분적으로 올바른 코드 ( 4119 )를 시도합니다 .

```
VSIM> force -deposit / keys_in 10 # 4, 10 # 1 20, 10 # 9 60, 10 # 0 80  
VSIM> 실행 200
```

입력 시퀀스는 분명히 잘못된 코드이지만 허용됩니다. 무엇이 잘못되었는지 알아 보려면 디자인을 디버깅해야합니다.

### 3.3.2 매크로 (.do) 파일 생성

그러나 코드 디버깅을 시작하기 전에 지금까지 수행 한 작업에서 매크로 파일을 만듭니다. 매크로 ( 할) 파일은 ModelSim과 시뮬레이터 및 시뮬레이션을 제어하기 위한 Tcl 명령이 포함 된 파일입니다. 매크로 파일은 시뮬레이터 설정 (디버그 창 열기) 또는 시뮬레이션 (신호 초기화) 등과 같은 반복적인 작업을 줄이는 데 도움이되는 유용한 파일입니다. 하나의 명령으로 전체 시뮬레이션을 실행할 수도 있습니다. 시뮬레이션을 마치면 사용한 명령으로 매크로 파일을 만든 다음 매크로 파일을 호출하여 동일한 시뮬레이션을 실행할 수 있습니다. 디자인을 항상 수정하고 이전과 동일한 데 이터로 다시 시뮬레이션해야하므로 매우 유용합니다.

앞에서 말했듯이 ModelSim 프롬프트에서 작성한 모든 명령 (및 메뉴 / 도구 모음 단추를 통해 선택된 일부 명령)은 파일 스크립트에 기록되었습니다 . 그러나 파일을 사용하기 전에 파일을 약간 수정합니다. 예를 들어 일부 명령을 작성할 때 오타가 있었을 수 있으며 ModelSim이 작성한 주석도 제거되어 제거 될 수 있습니다.

따라서 좋아하는 텍스트 편집기를 사용하여 transcript라는 파일을 편집하십시오 .

- 파일에서 주석을 제거
- 라인 추가 ' 웨이브 추가 \* '라인 추가 ' 웨이브 추가 \*
- 오타 수정 (있는 경우)
- 마지막 실행 명령을 제거

예를 들어, 새 매크로 파일의 이름을 lock.do로 바꾸십시오 (또는 새 스크립트 파일로 덮어 씁니다 ). 새 매크로 파일은이 [lock.do](#) 예제와 유사해야합니다 .

그렇지 않으면 시뮬레이션을 다시 시작할 때마다 설계의 신호가 파형 창에 다시 추가되므로 위의 라인 삭제 파형 \*이 필요합니다. 매크로 파일을 startup.do와 lock.do의 두 부분으로 나눌 수 있으며 시뮬레이터 시작시 수행 할 수있는 모든 작업 (창 열기 및 파형 창에서 신호 / 변수 추가)을 첫 번째 파일에 넣을 수 있습니다. 두 번째로 나머지는 모두 그런 다음 줄을 주석 해제하십시오 . modelsim.ini 파일에서 Startup = startup.do를 시작하고 시뮬레이터를 종료했다가 다시 시작하십시오. 시뮬레이터는이 디렉토리에서 호출 될 때마다 지금은 실행됩니다 startup.do를 파일을 작성하고 명령을 실행하십시오.

이제 매크로 파일을 호출하여 지금까지 수행 한 전체 시뮬레이션을 실행할 수 있습니다.

```
VSIM > restart -f VSIM> do lock.do
```

그리고 불완전한 입력 시퀀스 이후 마지막 실행 명령을 제거 했으므로 이제 오류 디버깅을 위해 시뮬레이션이 설정됩니다.

## 3.4 디자인 디버깅

이 디자인의 오류는 코드를 읽는 것만으로 쉽게 찾을 수 있지만 ModelSim이 디자인 디버깅을 위해 어떤 종류의 메소드를 제공하는지 살펴 보겠습니다. 이러한 방법에는 신호 및 변수 값 검사 및 설정, 중단 점 설정 및 라인 별 코드 실행, 시뮬레이션 상태를 복원하기위한 체크 포인트 설정 등이 포함됩니다.

오류를 디버깅하기 전에 이를 감지하고 반복해야 합니다. 일반적으로 자극 생성과 반응 확인을 모두 자동화해야 합니다.

### 3.4.1 웨이브 원도우에서 신호 검사

먼저 시뮬레이션 중에 신호 및 변수 값을 검사하는 방법을 살펴 보겠습니다.

신호를 끌어서 놓거나 복사 / 잘라내 기하여 붙여 넣을 수 있습니다. 예를 들어 클릭 신호는 일반적으로 몇 번 복사됩니다. 순서와 적절한 기수는 가독성과 디버깅 가능성에 큰 영향을 미칩니다. 신호 활동은 "왼쪽 위에서 오른쪽 아래로 흐릅니다". 신호 목록에 **분배기 텍스트**를

추가하여 목적에 따라 신호를 그룹화 할 수 있습니다 . 신호를 선택한 다음 마우스 오른쪽 버튼을 클릭하거나 메뉴에서 Insert-> Divider를 선택하십시오 . 분배기는 많은 수의 신호를 처리하는 데 도움이됩니다.

"in", "out", "full"(전체 실행을 하나의보기로 표시) 및 "활성 커서에서"의 4 가지 확대 / 축소 옵션이 있습니다. 마지막은 활성 커서를 창의 중간에 유지합니다.

웨이브 창에서 신호를 선택하면 이 버튼으로 예지를 검색합니다 (떨림 및 상승) 

오른쪽 마우스 버튼-> 강제로 디버그 목적으로 선택된 신호를 새로운 값으로 강제 할 수 있습니다. 특정 값을 검색 할 수도 있습니다. 이것은 특히 멀티 비트 신호 (= 버스)에 적합합니다.

메뉴에서 편집-> 신호 검색-> 신호 값 검색 ...을 선택하십시오.

첫 글자는 기수를 나타냅니다. B = 이진수, X = 16 진수이며 같은 따옴표 안에 표시됩니다 (예 : x "3")

### 3.4.2 명령을 사용하여 신호 및 변수 검사 및 설정

신호 및 변수를 항상 파형 창에 넣을 수 있지만 더 큰 설계의 경우 한 번에 표시하기에 충분한 양 이상의 데이터가 있을 수 있습니다. 검사 명령 코드에서 두 신호 및 변수의 값을 검사 할 수 있습니다 :

```
VSIM> curr_state_r  
# init VSIM 검사  
> / lock / sttran / count_v  
# 3 검사
```

그 신호 참고 curr\_state\_r이 현재 컨텍스트에 지정됩니다 ( SIM : / 잠금 )하지만 변수 count\_v하는 것은 아니다 (이 컨텍스트에 정의되어 시뮬레이션 : / 잠금 / sttran , 즉 프로세스 내부 sttran 의 아키텍처의 잠금 ). 따라서 curr\_state\_r은 경로를 필요로하지 않지만, count\_v는 그것에 경로를 지정해야합니다.

시간 순간을 지정하고 기수를 설정하여 객체의 과거 값을 검사 할 수도 있습니다 (현재 컨텍스트가 `sim : / lock` 이므로 현재 컨텍스트를 기준으로 `count_v` 경로를 지정할 수도 있습니다 ).

지금까지 명령 력으로 신호를 설정하는 방법을 배웠다 . 그러나 변수를 새로운 값으로 강요 할 수는 없습니다. 변수 값을 변경하려면 `change` 명령을 사용해야합니다 .

```
VSIM> sttran / count_v 12 변경
```

### 3.4.3 중단 점 사용

중단 점은 오류가 있는 위치 (기능, 절차, 프로세스 등)를 대략 알고 있거나 일부 코드 섹션의 작동 방식을 알고 싶을 때 유용합니다. 관심 있는 코드 섹션의 시작 부분에 중단 점을 설정하고 시뮬레이션을 실행하십시오. 중단 점에 도달하면 코드를 한 줄씩 단계별로 디버깅 할 수 있습니다.



ModelSim Main / Wave Window Toolbar의 단계 버튼. 빨간색은 Step 버튼과 녹색 Step Over 버튼을 나타냅니다. Step 명령은 코드의 현재 줄을 실행하고 다음 실행 줄로 이동합니다. Step Over 명령은 비슷하게 작동하지만 함수 및 프로시저 호출을 표시하지 않습니다.

먼저 중단 점을 활성화해야 합니다. 소스 창에서 행 번호를 마우스 오른쪽 단추로 클릭하고 대화 상자에서 중단 점 사용을 선택하여 중단 점을 활성화 할 수 있습니다. 줄 번호 옆에 있는 빨간색 점은 활성화 된 중단 점을 나타냅니다. 실행 가능한 행만 중단 점으로 설정할 수 있습니다.

상태 머신 다이어그램 과 코드를 비교하십시오. 파일 lock.vhd의 상태 code\_2에서 중단 점을 설정하고 시뮬레이션을 실행하십시오. 예를 들어, 64 행은 약 1시에 도달해야 합니다. 820ns

```
VSIM> bp lock.vhd 64 VSIM
> 실행
```

이제 들어오는 keys\_in 버스와 예상되는 third\_c 상수 값을 모두 검사하십시오.

```
VSIM> -decimal keys_in
# 1
검사 VSIM> lock_pkg / third_c
# 6 검사
```

이제 한 단계 앞으로 계속하십시오.

```
VSIM> 단계
```

무엇이 잘못되었는지 파악하고 코드를 수정하십시오.

수정 된 코드를 컴파일하십시오. 이 튜토리얼의 시작 부분에서 디자인을 위한 Makefile 을 만들었으므로, (트랜스 스크립트 창 또는 터미널 창에서) make 를 호출 하면 됩니다.

```
VSIM> make
또는
$ make
```

이제 수정 된 코드의 기능을 확인해야 합니다. 시뮬레이션을 다시 시작하십시오 ( restart -f ). 다시 시작하면 ModelSim이 수정 된 코드를 자동으로 로드합니다. 원하는 경우 중단 점을 비활성화하고 ( disablebp ) 매크로를 실행하십시오 ( do lock.do ).

```
VSIM > restart -f VSIM> disablebp VSIM
> do lock.do VSIM
> 실행 200
```

예상 한대로 디자인은 해당 입력 시퀀스에서도 올바르게 작동합니다.

### 3.4.4 시간 측정

웨이브 뷰어에서 최상위 메뉴 추가-> 커서에서 선택하십시오. 이제 두 개의 수직선 (커서)이 표시됩니다. 당신은 쉽게 그들을 드래그하여 바닥에 시간 간격을 볼 수 있습니다. 실행 시간 (또는 신호주파수)을 측정해야 할 때 매우 편리합니다. 시간을 클럭주기의 지속 시간으로 나누면 클럭주기 수를 도출 할 수 있습니다.

### 3.4.5자가 점검 테스트 벤치 사용

마지막 단계로 자동화 된 테스트 [벤치 인 tb\\_lock.vhd](#)를 컴파일 하고 실행하십시오.

```
$ vcom vhd / tb_lock.vhd
$ vsim tb_lock & VSIM
> 10ms 실행
```

수정 된 버전이 모든 테스트를 통과하는 동안 원래 lock.vhd를 시뮬레이션하면 오류 메시지가 표시됩니다. tb 자체가 완료되면 testbech가 시뮬레이션을 중단하며 가장 쉬운 방법은 의도적으로 어설션 오류를 발생시키는 것입니다. 따라서 "실패"를 무시할 수 있습니다.

의도적으로 잠금에 대한 버그를 작성하고 다시 컴파일 한 후 테스트 벤치가 오류를 감지하는지 확인하십시오. 이것을 몇 번 반복하십시오.

## 4. 요약

이것으로 ModelSim 튜토리얼을 마칩니다. 이것은 ModelSim 사용을 시작하는 방법에 대한 간단한 소개 일 뿐이지 만 여전히 배울 것이 많습니다. 예를 들어, 명령은 종종 여기에 표시된 것보다 더 다양합니다. 중단 점은 예를 들어 조건부 일 수 있습니다 ( 즉, 중단 점에 중단 점이 있는지 여부를 판별하는 조건이 포함될 수 있음). 중단 점은 중단 점에 도달했을 때 실행될 명령을 지정할 수도 있습니다. ModelSim Command Reference에서 모든 명령, 옵션 및 제한 사항을 확인하십시오.

이 문서에서 다루지 않은 기능도 있습니다. 여기에는 코드 적용 범위 ( 시뮬레이션 실행 중 명령문 / 분기 / 조건 등 이 실행 되었는지 여부 ), 프로파일 러 (시뮬레이션 실행 중 CPU / 메모리 사용량 결정 ) 등을 측정하는 기능이 포함 됩니다. 통계의 첫 번째 유형은 매우 유용 할 수 있습니다 예를 들어, 테스트 벤치의 선 (善)을 측정하고 후자는 시뮬레이션 실행 시간 / 메모리 사용을 개선하는 데 사용할 수 있습니다. 이러한 기능과 다른 기능에 대해 배우려면 ModelSim 사용 설명서를 읽으십시오.

이 튜토리얼에서 사용 된 기본 ModelSim 명령에 대한 요약입니다.

설계 라이브러리 작성 및 설계 컴파일 명령 쉘 또는 대화 창에서 프롬프트 할 수 있습니다.

vlib	- 새로운 디자인 라이브러리 만들기
vmap	- 논리적 라이브러리 이름과 디렉토리 경로 간의 매핑 정의
vdir	- 설계 라이브러리의 내용을 나열합니다.
vdel	- 지정된 라이브러리에서 설계 단위 삭제
vmake	- 지정된 설계 라이브러리에 대한 Makefile 작성
vcom	- VHDL 소스 코드를 지정된 설계 라이브러리로 컴파일
vsim	- VSIM 시뮬레이터를 호출하십시오.
verror	전 - 경고 / 오류 메시지에 대한 정보를 인쇄합니다. / 는 4 자리 msg 번호입니다.

## 시뮬레이션 명령.

- 전망 - 창 (웨이브, 목록, 소스 등)을 만듭니다.
- 웨이브를 추가 - Wave 창에 객체 추가
- 힘 - VHDL 신호에 자극을 가한다
- 변화 - VHDL 변수, 상수 또는 일반 값 변경
- 운영 - 시뮬레이션 실행
- 재시작 - 시뮬레이션을 다시로드하고 다시 시작하십시오. 옵션 -f는 확인이 필요합니다
- 해야 할 것 - 매크로 (do) 파일 실행

## 디버그 명령.

- 검사하다 - 신호 또는 변수의 값을 검사
- bp - 중단 점 설정
- disablebp - 중단 점 비활성화
- enablebp - 중단 점 사용
- 검문소 - 시뮬레이터 상태 저장
- 복원 - 저장된 시뮬레이터 상태 복원