JK ED&C

ModelSim/Questa Tutorial

Software Version 10.4

서울시 영등포구 양평동3가 이앤씨 드림타워 1113호 www.ednc.com Tel : 02-2069-0099

Table of Contents

Chapter 1
Introduction

아시다시피 ModelSim 과 Questa 는 FPGA/ASIC을 개발하는데 있어서 꼭 있어야 하는 필수 HDL Simulator 입니다. 사용법도 상당히 쉽게 되어 있어서 Tool 에 대한 사전 지식이 없어도 편리하고, 익숙하게 사용할 수 있도록 많은 문서들을 같이 제공하고 있습니다.

해당 문서들은 Tool 이 설치되면 자동으로 같이 설치가 되어 영문 PDF, HTML 형태로 제공이 되고 있고, Windows 환경에서 혹은 Linux, Solaris 의 CDE, KDE, GNOME 환경에서 보실 수 있습니다.

각 문서들은 아래의 표를 통해서 보실 수 있습니다.

문서 종류	파일 포맷	보는 방법 (Tool 동작시)
Installation & Licensing Guide	PDF	**Help > PDF Bookcase**
	HTML and PDF	**Help > InfoHub**
Quick Guide (command and feature quick-reference)	PDF	**Help > PDF Bookcase** and **Help > InfoHub**
Tutorial	PDF	**Help > PDF Bookcase**
	HTML and PDF	**Help > InfoHub**
User's Manual	PDF	**Help > PDF Bookcase**
	HTML and PDF	**Help > InfoHub**
Command Reference Manual	PDF	**Help > PDF Bookcase**
	PDF	**Help > InfoHub**
Graphical User Interface (GUI) Reference Manual	PDF	**Help > PDF Bookcase**
	PDF	**Help > InfoHub**
Foreign Language Interface Manual	PDF	**Help > PDF Bookcase**
	HTML	**Help > InfoHub**
OVL Checkers Manager User's Guide	PDF	**Help > PDF Bookcase**
	HTML	**Help > InfoHub**
Power Aware Simulation User's Manual	PDF	**Help > PDF Bookcase**
	HTML	**Help > InfoHub**

Command Help	ASCII	type **help [command name]** at the prompt in the Transcript pane
Error message help	ASCII	type **verror <msgNum>** at the Transcript or shell prompt
Tcl Man Pages (Tcl manual)	HTML	select **Help** > **Tcl Man Pages**, or find *contents.htm* in *\modeltech\docs\tcl_help_html*
Technotes	HTML	available from the support site

Chapter 2
Conceptual Overview

ModelSim/Questa 는 VHDL, Verilog, SystemVerilog, SystemC 그리고 mixed-language 로 설계된 Design 을 Simulation 하는 제품입니다.

이번 챕터에서는 ModelSim/Questa 를 통해 Simulation 을 진행하는데 있어 다음 5가지 방법에 간략하게 알아보겠습니다.

- Design Optimizations – 최근에 발표되는 ModelSim/Questa 의 Default Mode 입니다.
- Basic Simulation flow - 챕터 3 에서 자세히 살펴보겠습니다.
- Project flow – 챕터 4 에서 자세히 살펴보겠습니다.
- Multiple library flow – 챕터 5 에서 자세히 살펴보겠습니다.
- Debugging tools – 챕처 6 이후부터 기능을 하나씩 살펴보겠습니다.

Design Optimizations

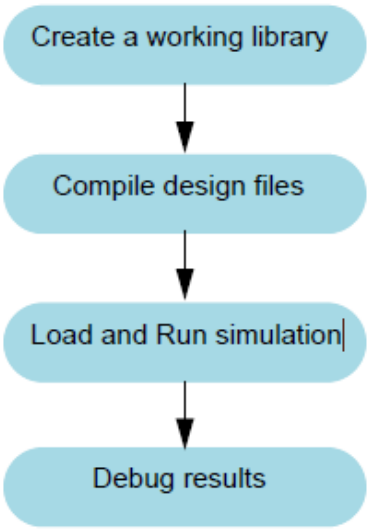
Basic simulation flow 에 대해 알아보기 전에, Design Optimization 기능에 대해 확실하게 개념을 잡는 것이 중요합니다. ModelSim/Questa 는 기본적으로 Optimization Mode 가 Default Mode 로 되어 있습니다. Optimization Mode 로 동작을 하면 Simulation Performance 는 약 10 배 정도의 이득을 볼 수 있습니다. 대신에 Simulation 결과를 Wave window 를 통해 보는데 있어서는 제약이 따르게 됩니다. 물론 유저 분이 중요하다고 생각되는 특정한 모듈이나 signals 에 대해서는 옵션을 통해 제약을 해제할 수도 있습니다.

따라서, 최선의 방법은 유저 분이 Design 상의 정보 중 어떤 것을 Optimization 을 하고 어떤 것을 Optimization 을 하지 않을 것인지 결정을 하여 Design 을 최적화 하는 것입니다. ModelSim/Questa 에서는 위에서 언급한 내용처럼 디폴트 모드가 Optimization 모드로 되어 있고, vopt command 를 사용할 수도 있습니다. vopt command 사용법은 Modelsim/Questa 의 User Manual 중에 Optimization Design with vopt 챕터를 보시면 자세히 나와 있습니다.

Basic Simulation Flow

유저 분이 Design 을 ModelSim/Questa 틀 통해 Simulation 을 하기 위한 가장 기본적인 것들을 이번엔 살펴볼 것입니다. ModelSim/Questa 의 Basic Simulation Flow 는 아래의 그림을 보시면 됩니다.

Figure 2-1. Basic Simulation Flow - Overview Lab



- Creating a working library

  ModelSim/Questa 에서 Simulation을 진행하기 위해서는 모든 Design 이 Compile 된 Library 가 있어야 합니다. 일반적으로 새로운 Simulation 을 하기 위해 Tool 을 구동했으면 제일 먼저 work 라는 이름으로 불리는 Working library 를 만드는 단계입니다.

- Compiling design files

  Working library 를 만들었으면 이제는 Library 에 Design file 을 Compile 해야 합니다. Compile 된 라이브러리는 다른 Platform 에서 진행 할 경우 다시 Re-Compile 할 필요 없이 바로 사용이 가능합니다.

- Load and run simulation

  모든 Design 이 Compile 이 되었으면, Simulator 에 top-level module 을 invoke 합니다. Design 이 성공적으로 load 가 되면, Run command 를 이용하여 simulation 을 진행합니다.

- Debug

  Simulation 진행으로 얻어진 결과가 완벽하지 않다면, ModelSim/Questa 의 debug 기능을 이용하여 debugging 을 합니다.

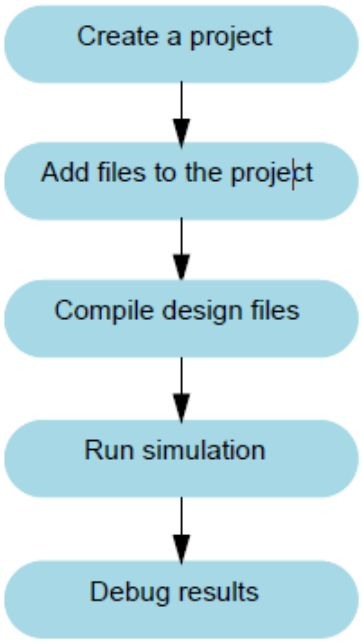
Project Flow

Project Flow 는 HDL design 을 Simulation 하는 방법 중에 한가지 입니다. Project Flow 를 이용하면 매우 편리하게 Design file 을 관리하고 Simulation 결과를 지정할 수 있습니다. 하지만,

ModelSim/Questa 를 사용하시면서 꼭 Project Flow 를 사용할 필요는 없습니다. Basic Flow 와 Project Flow 중에 유저 분이 편한 대로 진행하시면 됩니다.

아래의 그림은 Project Flow 방식의 진행 순서를 보여주고 있습니다.

Figure 2-2. Project Flow



위의 Flow 처럼 Project Flow 를 통한 Simulation 진행 순서는 Basic Simulation 과 상당히 유사합니다. 하지만 다른 점도 있습니다.

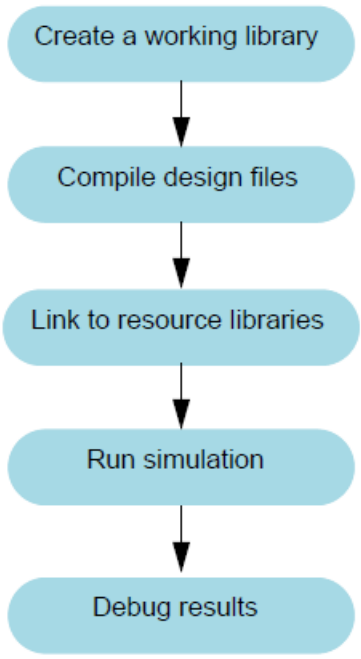
- 유저 분이 직접 Working library 는 만들 필요가 없습니다. Project 를 만들 때 자동으로 Library 가 만들어 집니다.
- 작업하고 있는 Project 는 지속됩니다. – ModeSim/Questa 를 종료 후 다시 실행할 때 최근에 작업했던 Project 가 자동으로 열립니다.

Multiple Library Flow

ModelSim/Questa 에서 Library 를 사용하는 방법에는 2가지 방법이 있습니다. 첫 번째는 Design 이 컴파일 된 로컬 library 를 사용하는 방법과 두 번째로 Resource library 를 사용하는 방법이 있습니다. Working library 를 이용할 때 만약 Design 의 변경이 있다면 Recompile 을 해주어야 합니다. Resource library 는 유저가 생성 할 수 있으며, 디자인을 위해 부품 역할을 할 수 있도록 실리콘 공급 업체에 의해서 제공받을 수도 있습니다

아래의 그림은 Multiple Library Flow 를 보여주고 있습니다.

Figure 2-3. Multiple Library Flow



Project Flow 를 통해 진행을 할 때도 Link to resource libraries 를 할 수 있습니다. Project Flow 를 통해 진행하고 있다면 맨 위의 Create a working library 단계를 create the project 와 add the testbench to the project 의 2 스텝 단계로 변경하여 진행하시면 됩니다.

Debugging Tools

ModelSim/Questa 에는 Design 을 분석하고 디버깅할 수 있는 많은 기능들을 가지고 있습니다.

- Using projects
- Working with multiple libraries
- Simulating with SystemC
- Setting breakpoints and stepping through the source code
- Viewing waveforms and measuring time
- Exploring the "physical" connectivity of your design
- Viewing and initializing memories
- Creating stimulus with Waveform Editor

- Analyzing simulation performance
- Testing code coverage
- Comparing waveform
- Debugging with PSL assertions
- Using SystemVerilog assertion and cover directives
- Using the SystemVerilog DPI
- Automating simulation

Chapter 3
Basic Simulation

이번 챕터에서는 ModelSim./Questa 를 통해 Basic Simulation 하기 위해 아래의 단계별로 순차적으로 진행을 할 것입니다.

1. Create the working library
2. Compile the Design Units
3. Optimize the Design
4. Load the Design
5. Run the Simulation

이번 챕터부터는 예제 파일을 가지고 작업을 진행할 것입니다. 이번 챕터의 예제 파일은 8bit binary up counter 이며 해당 파일은 ModelSim/Questa 를 설치된 경로 안에 있습니다.

Verilog – <*install_dir*>*/examples/tutorials/verilog/basicSimulation/counter.v* and t*counter.v*
VHDL – <*install_dir*>*/examples/tutorials/vhdl/basicSimulation/counter.vhd* and *tcounter.vhd*

예제 파일들은 Verilog 와 VHDL 둘 다 지원을 하고 있습니다. 가지고 계신 ModelSim/Questa 의 License 에 따라 해당하는 언어 예제를 선택하시면 됩니다.

Create the Working Design Library

Design 을 Simulation 을 하기 전에 먼저 Library 를 만들어 Compile 을 해야 합니다.

1. 새로운 디렉토리를 만들어 예제 파일을 그 디렉토리에 Copy 합니다.
   A. ModeSim/Questa 를 실행합니다.

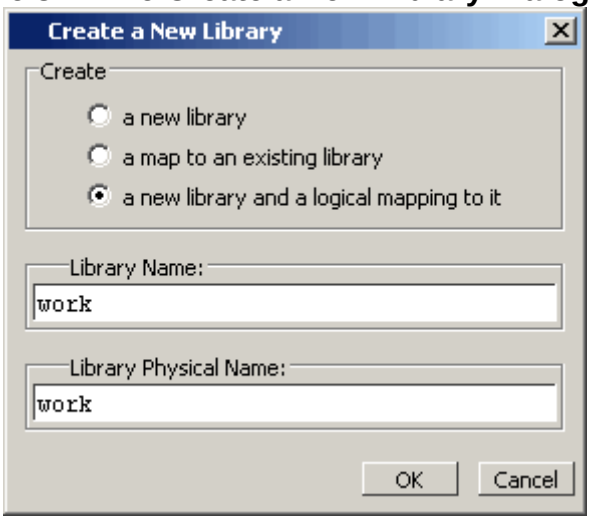
      단축 아이콘을 클릭하거나 명령프롬프트에 vsim command 를 입력하면 ModelSim/Questa 의 GUI 가 올라옵니다.

      GUI 가 올라오면 File>Change Directory 를 통해 1번에서 만든 디렉토리로 이동합니다.

2. Working library 를 만듭니다.

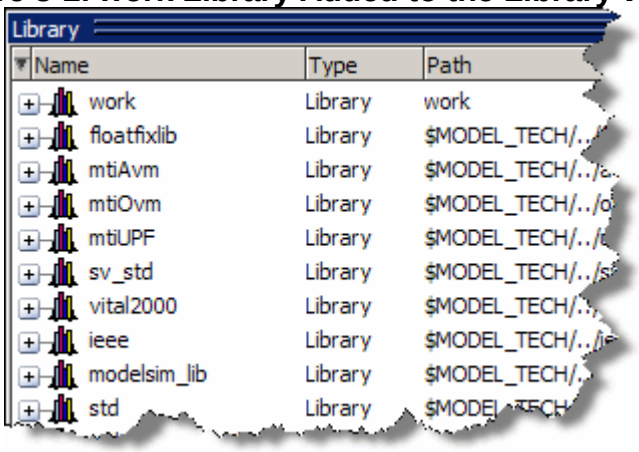
A. File>New>Library 를 선택합니다

Figure 3-1. The Create a New Library Dialog Box



B. 보여지는 Dialog 창에서 Library name 과 Physical name(실제로 만들어지는 디렉토리)에 work 라고 입력합니다
입력이 끝났으면 OK 버튼을 클릭합니다.

Figure 3-2. work Library Added to the Library Window



OK 버튼을 클릭하면 Library 윈도우에 library 가 만들어 진 것을 확인 할 수 있습니다.

위에서의 작업을 command 로 작업을 하기 위해선 아래의 command 를 입력해 주면 됩니다.

vib work
vmap work work

Compile the Design Units

Work library 가 만들어 졌으면, 이제 Compile 을 할 차례입니다.

1. Copy 를 한 예제 파일들을 Compile 을 할 것 입니다.

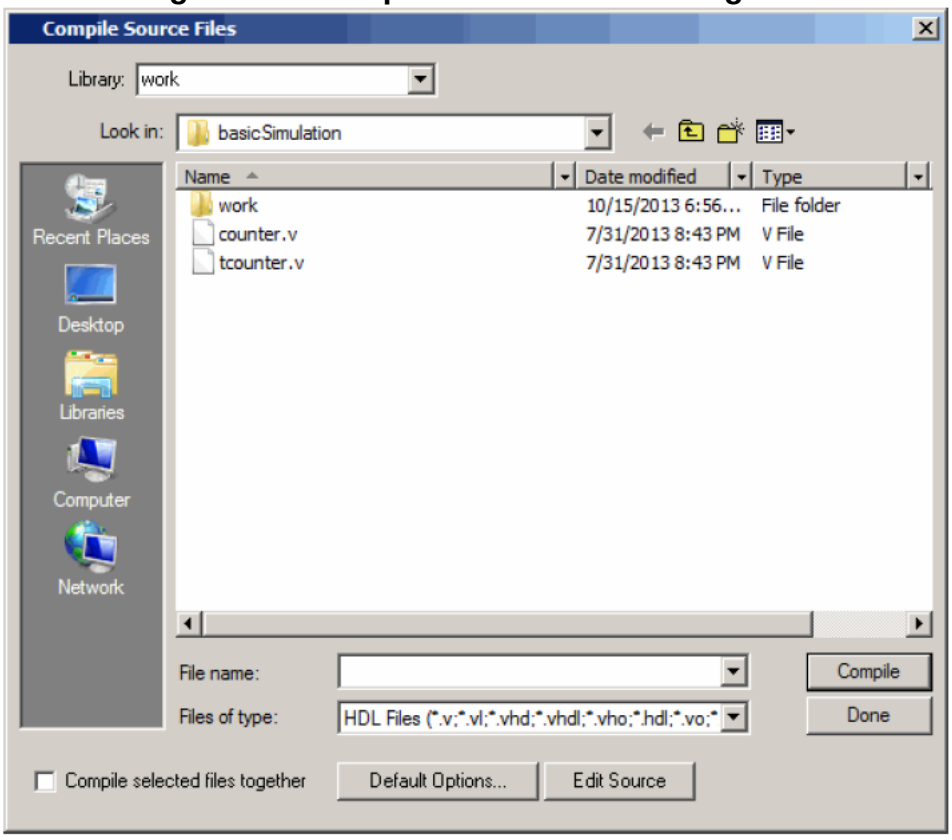
   A. ModelSim/Questa 의 메뉴 중에 Compile>Compile 을 선택합니다.

      만약에 해당하는 메뉴가 보여지지 않으면 Project Flow 상태일 수도 있습니다. Project Flow 상태이면 File>Close 를 선택해 주시면 Project 가 종료되며 원하시는 메뉴를 선택할 수 있습니다.

   B. Compile 창이 열리면 Copy 한 2개의 파일 – counter.v tcounter.v 파일을 선택하고 Compile 버튼을 클릭합니다. 그러면 Work library 안에 해당 파일이 compile 이 됩니다.

   C. 두 개의 파일을 Compile 을 했으면 Done 버튼을 클릭합니다.

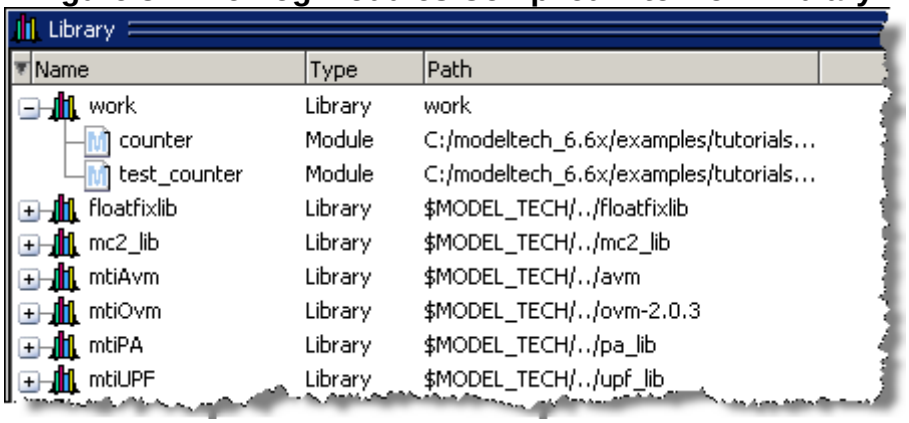
Figure 3-3. Compile Source Files Dialog Box



2. Compile 한 결과를 확인할 것입니다.

   A. Library 윈도우를 보면 Work library 옆에 "+" 아이콘이 보이실 겁니다. "+" 를 클릭하면 2개의 Design unit 이 보일 것입니다.

Figure 3-4. Verilog Modules Compiled into work Library



Optimize the Design

1. vopt command 를 이용하여 Full visibility 가 가능한 Optimize design 을 만들 것입니다.

   A. ModelSim/Questa 의 GUI 에서 하단부에 있는 Transcript 윈도우에 아래의 command 를 입력합니다.

      vopt +acc test_counter –o testcounter_opt

      +acc 는 visibility 관련 옵션이며, -o 는 결과 파일 이름을 지정하는 옵션입니다.

주의 : vopt command 를 사용할 때 같은 이름으로는 결과 파일이 만들어 지지 않습니다.

Load the Design

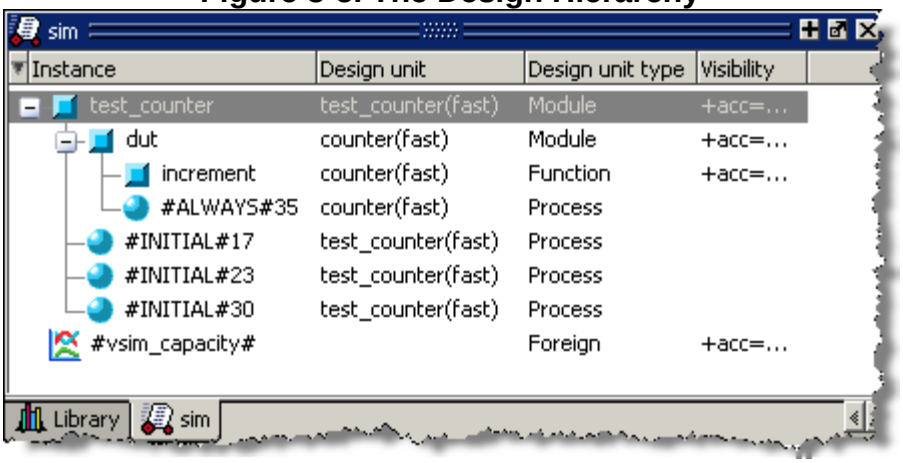
1. 이제는 test_counter module 을 Simulator 에 Loading 할 것입니다.

   A. Design loading 은 vsim command 를 통해 이루어 집니다. 전 단계에서 만들어 놓은 Optimize design 을 사용하기 위해 아래의 command 를 입력합니다.

      vsim testcounter_opt

      Design 이 Load 가 되면 Structure 윈도우에 해당하는 Design 정보를 볼 수 있습니다.

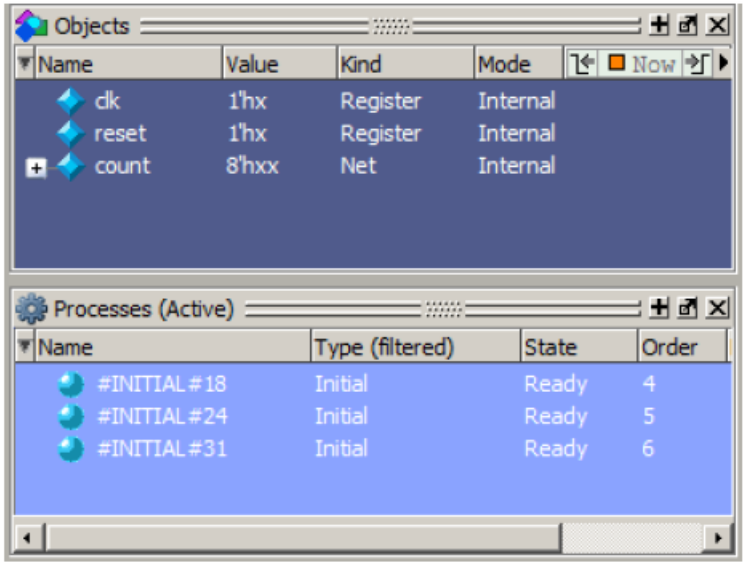
Figure 3-5. The Design Hierarchy



추가적으로 Objects 와 Process 윈도우가 열리게 됩니다. Object 윈도우에서는 Structure 윈도우에서 선택한 Module 이 가지고 있는 Data 에 대한 정보를 볼 수 있습니다. 확인 할 수 있는 Data 는 signals, net, register, constants, variable 과 디자인상에서 선언된 process, generics, parameters 와 SystemC module 의 Data variable 입니다.

Process 윈도우에서는 HDL, SystemC process 정보를 4가지 모드 -Active, In region, Design, Hierarchical-로 보여줍니다.

Figure 3-6. The Object Window and Processes Window



Run the Simulation

이제 Simulation 을 진행하기 위한 준비단계가 끝났습니다. 이제 Wave 윈도우에 signal 을 추가하고 Simulation 을 진행할 것입니다.

1. Wave 윈도우 열기

   A. **view wave** 라는 command 를 입력합니다.

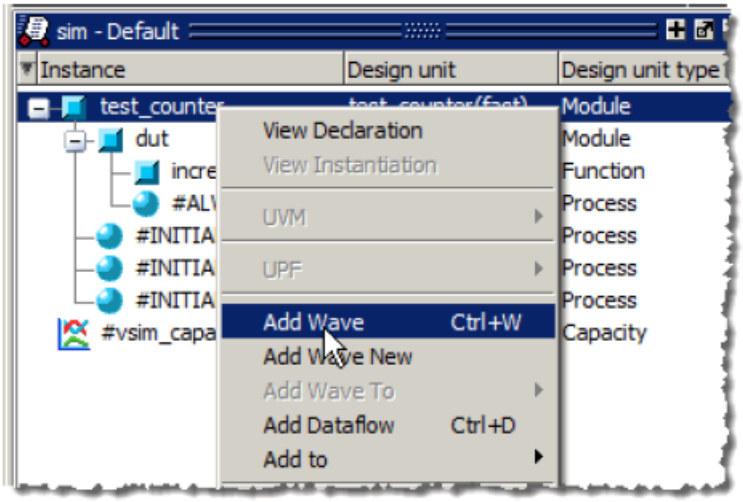
      command 를 입력하면 화면 오른쪽에 Wave 윈도우가 나타납니다.
      물론 GUI 를 통해서도 Wave 윈도우를 열 수 있습니다. View>Wave 를 선택하시면 마찬가지고 화면 오른쪽에 윈도우가 나타납니다.

2. Wave 윈도우에 Signal 추가하기

   A. Structure(sim) 윈도우에서 test_counter 모듈을 선택하고 마우스 오른쪽 버튼을 클릭하면 팝업 메뉴가 열립니다.

   B. 팝업메뉴에서 Add>To Wave>All items in region 을 선택하면 Design 의 모든 signal 이 Wave 윈도우에 추가 됩니다.

Figure 3-7. Using the Popup Menu to Add Signals to Wave Window



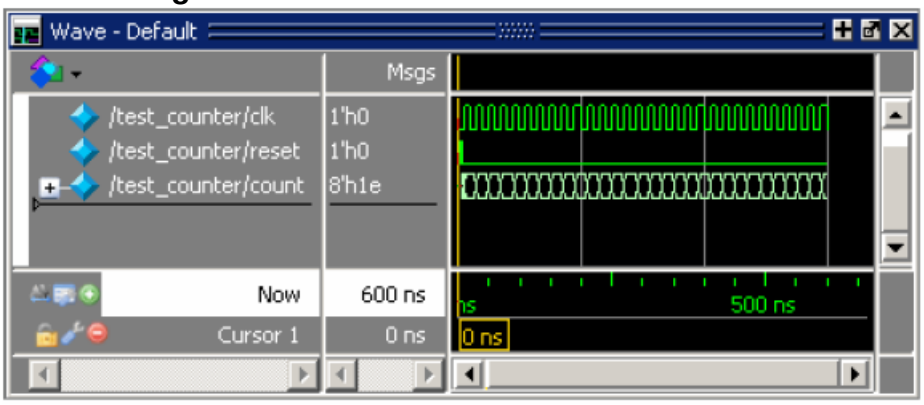
3. Simulation 시작하기

   A. Run 아이콘을 클릭 합니다. 

      Run 아이콘을 클릭하면 100ns 만큼 Simulation 이 진행되며, 그 결과가 Wave 윈도우에 표시됩니다.

   B. Transcript 윈도우에 500 이라고 입력하면, Simulation 은 기존의 100ns 에서 500ns 만큼 추가적으로 진행되어 총 600ns 까지의 Simulation 결과는 Wave 윈도우에서 확인할 수 있습니다.

Figure 3-8. Waves Drawn in Wave Window



C. 메인 윈도우나 Wave 윈도우에서 Run-All 아이콘을 클릭합니다. 

그러면, Simulation 은 Design 상에 Break –Verilog 의 $stop–부분까지 혹은 외부에서 강제로 Break 를 걸 때까지 Simulation 이 진행됩니다.

위의 아이콘이 Break 아이콘 입니다.

Set Breakpoints and Step through the Source

이제는 Debugging 을 하기 위해 Breakpoint 를 Source code 안에 삽입을 할 것입니다. Break point 를 설정하고 Simulation 을 진행하면 Simulation 은 해당 break point 에서 멈춥니다. Break point 는 Source code 중에 실행이 되는 Line 에 설정 할 수 있고, 설정 가능한 Line 은 Source 윈도우상에 빨간색 Line number 로 표시됩니다.

1. Source 윈도우에서 counter.v 코드 열기

   A. 메인 메뉴에서 View>Files 를 선택해 Files 윈도우를 엽니다.

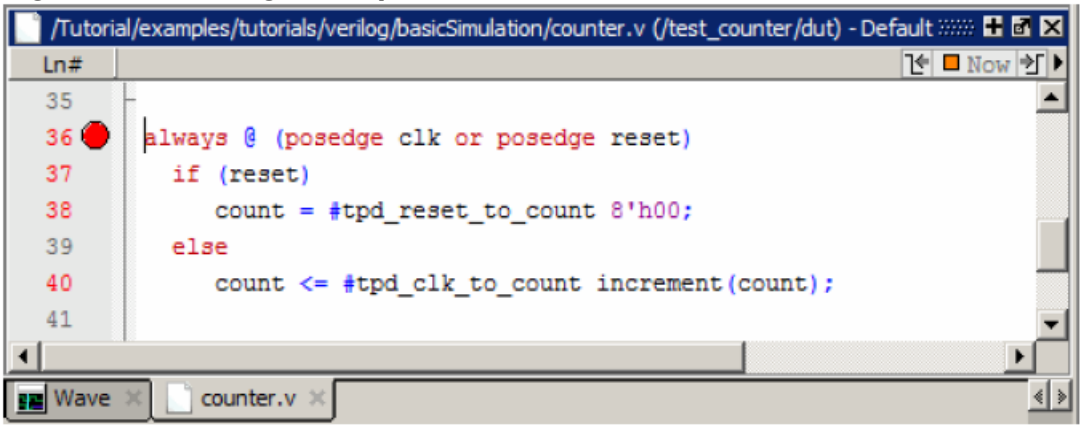
   B. Files 윈도우에서는 현재 진행중인 Simulation 에 대한 정보가 나오며 "+" 를 클릭하면 File 리스트를 볼 수 있습니다.

   C. File 리스트 중에서 counter.v 를 더블 클릭하면 Source 윈도우에 해당 파일이 열립니다.

2. Break point 설정하기

   A. Source 윈도우에서 열린 counter.v 파일의 36번 Line 까지 마우스 스크롤을 이용하여 이동합니다. 윈도우의 36번 라인에 Line 숫자 부분을 클릭하면 Line 옆에 빨간색 공이 생기며 Break point 가 설정됩니다.

Figure 3-9. Setting Breakpoint in Source Window



3. Break point 삭제 및 활성화, 비활성화 시키기

   A. break point 가 만들어진 Line 에서 해당 Line 의 숫자 부분을 다시 클릭합니다. 빨간색 공이 검정색 공이 되며 비활성화 됩니다.

   B. 다시 해당 Line 을 클릭하면 빨간 색 공이 되며 활성화 됩니다.

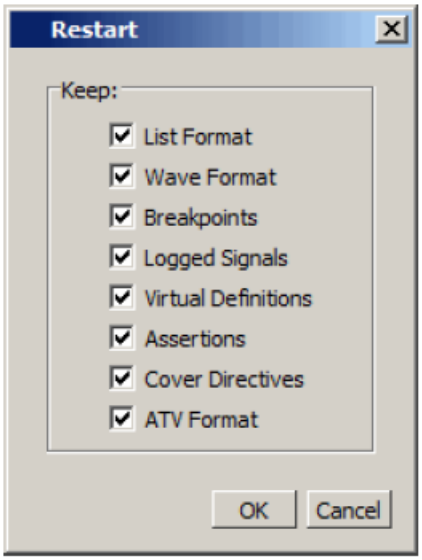
   C. Break point 를 삭제하기 위해서는 해당 Break point 에서 마우스 오른쪽 버튼을 클릭하고 Remove Breakpoint 36 클릭 하시면 해당 Break point 가 삭제됩니다.

   D. 다시 36 번 라인에 Break point 를 만들고 활성화 시킵니다.

4. Simulation 다시 시작하기

   A. 현재 진행하고 있는 Simulation 을 Reload 하기 위해서는 Restart 아이콘을 클릭하면 됩니다.
   Restart 아이콘을 클릭하면 아래와 같은 창이 나타납니다.

Figure 3-10. Setting Restart Functions

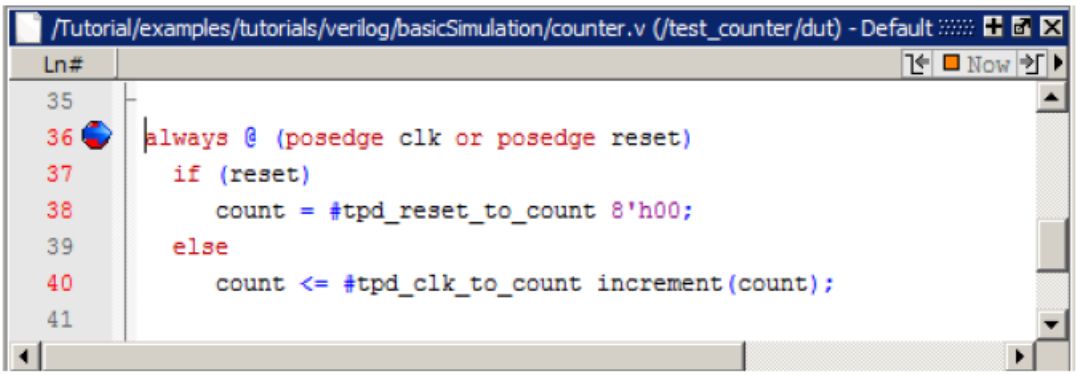


B. 전체를 체크하고 Restart 버튼을 클릭합니다.

C. Run-All 아이콘을 클릭 합니다.

이제 Simulation 은 위에서 설정한 36번 Line 에 도달할 때까지 Simulation 이 진행됩니다. Simulation 이 해당 Break point 에 도달하면 Simulation 은 멈추고 해당 파일이 Source 윈도우에 열리며, 파란색 화살표 마크가 해당 Line 에 표시됩니다.

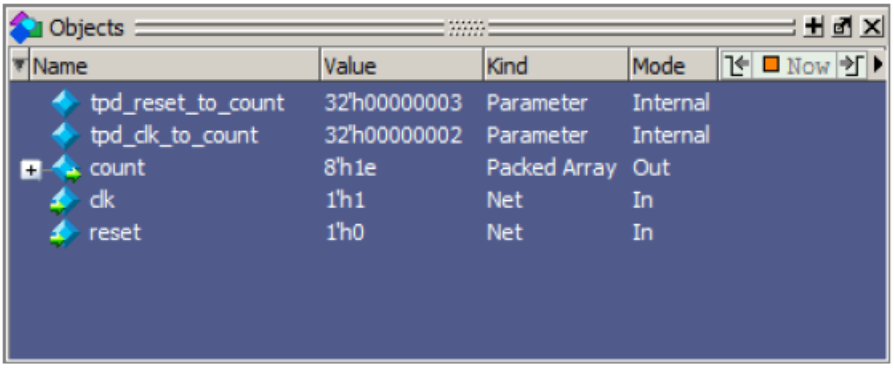
Figure 3-11. Blue Arrow Indicates Where Simulation Stopped.



일반적으로 Break point 에 의해 Simulation 이 멈추면 유저들은 더 많은 현 상태에서의 더 많은 정보를 얻기를 원할 것 입니다. 이제 그 정보를 얻을 수 있는 여러 옵션에 대해 살펴보겠습니다.

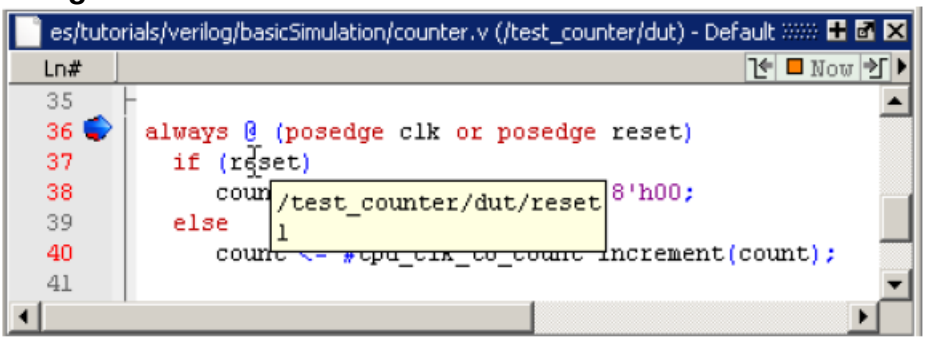
우선 Object 윈도우를 통해 signal value 를 확인하겠습니다.

Figure 3-12. Values Shown in Objects Window



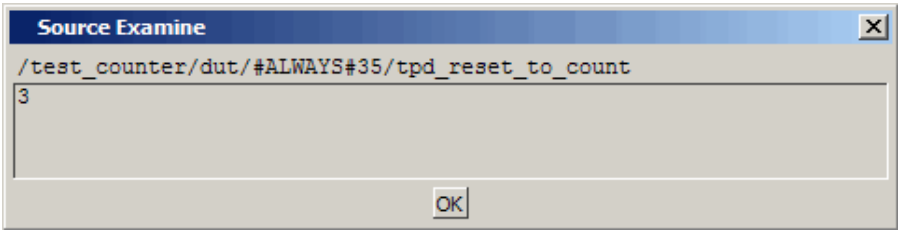
Source 윈도우에서 보여지는 Design 코드상의 highlight 처리된 signal 혹은 parameter, variable 위에 마우스 포인트를 놓으면 Wave Window 에서 cursor 가 선택된 부분의 변수 값이 노란 박스로 표시됩니다.

Figure 3-13. Hover Mouse Over Variable to Show Value



또한 Design 코드상의 highlight 처리된 signal 혹은 parameter, variable 위에 마우스 포인트를 놓고 마우스 오른쪽 버튼을 클릭하여 Examine 을 선택하면 현재의 Value 가 창에 표시가 됩니다.

Figure 3-14. Parameter Name and Value in Source Examine Window



examine 라는 command 를 입력을 해서 Value 를 확인할 수 있습니다.

EX) examine count

5. Step command 를 활용하기
   A. 메인 메뉴의 아이콘 중 step 아이콘을 클릭합니다.

      Step 아이콘은 debugger 기능입니다.
      Step 아이콘을 클릭하거나 step command 를 입력하면 현재 멈추어져 있는 Simulation 이 Code 한 줄 한 줄씩 실행이 되어 갑니다.

6. Simulation 종료하기
   A. Simulate>End Simulation 을 선택하면 현재 진행중인 Simulation 이 종료 됩니다.

Chapter 4
Projects

이번 챕터에서는 3장에서 진행했던 Simulation 방법을 Project Flow 방식으로 진행 할 것입니다. Project Flow 를 통해 작업을 하면 .mpf 파일이 만들어지고 Project 는 아래와 같은 정보로 구성 됩니다.

- HDL Source 파일을 저장하거나 Reference 경로
- Source 파일 외에 다른 문서(READMEs, Documentation)
- Local library
- Reference global library

이번 챕터부터는 예제 파일을 가지고 작업을 진행할 것입니다. 이번 챕터의 예제 파일은 8bit binary up counter 이며 해당 파일은 ModelSim/Questa 를 설치된 경로 안에 있습니다.

Verilog – <*install_dir*>*/examples/tutorials/verilog/projects/counter.v* and t*counter.v*

VHDL – <*install_dir*>*/examples/tutorials/vhdl/ projects /counter.vhd* and *tcounter.vhd*

예제 파일들은 Verilog 와 VHDL 둘 다 지원을 하고 있습니다. 가지고 계신 ModelSim/Questa 의 License 에 따라 해당하는 언어 예제를 선택하시면 됩니다.

Create a New Project

1. 새로운 디렉토리를 만들고 예제 파일은 해당 디렉토리로 Copy 하기

   A. ModeSim/Questa 를 실행합니다

      단축 아이콘을 클릭하거나 vsim command 를 입력하면 ModelSim/Questa 의 GUI 가 올라옵니다.

      만약 지난 챕터를 이어서 바로 시작하시는 경우엔 ModelSim/Questa 가 구동 되어 있으니 다시 실행할 필요가 없습니다.

      File>Change Directory 에서 위에서 만든 디렉토리로 지정합니다.

2. Project 만들기

A. 메인 메뉴에서 File>New>Project 를 선택합니다.

   메뉴를 선택하면 Create Project 창이 열립니다. 이 창에서 유저 분은 Project name, Project Location-directory, default library name, Modelsim/Questa 설정 파일인 .ini 파일 등을 지정할 수 있습니다.

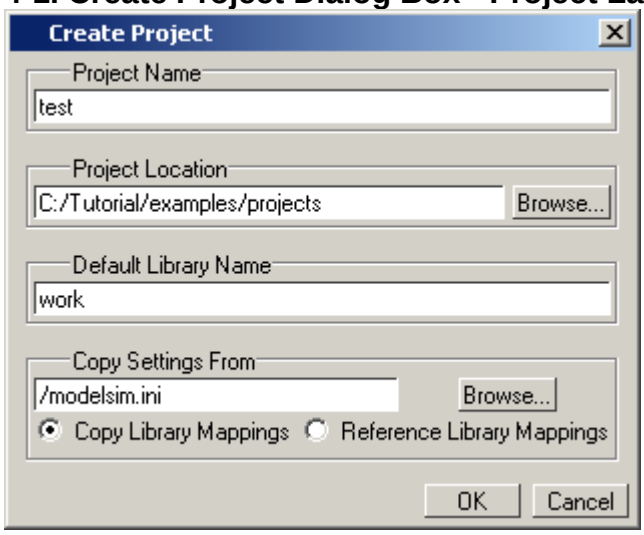
B. Project name 을 test 로 입력합니다.

C. Browse 버튼을 클릭해서 Project 를 저장할 디렉토리를 지정합니다.

D. Default library 이름을 work 로 입력합니다.

E. OK 를 클릭합니다.

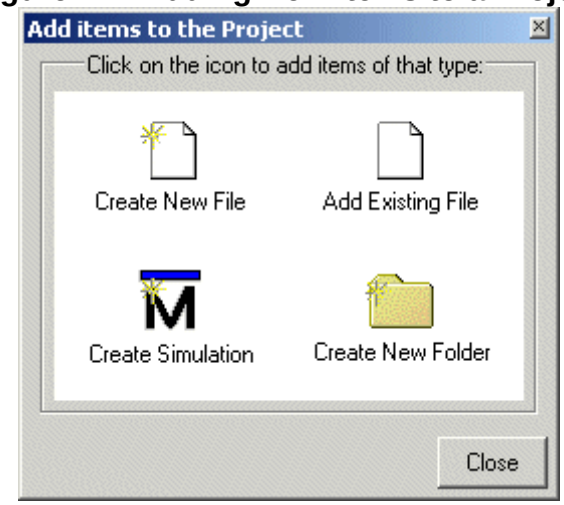
Figure 4-1. Create Project Dialog Box - Project Lab



Add Objects to the Project

앞선 순서에서 OK 버튼을 클릭하면서 Project 를 위한 기본적인 설정은 끝났습니다. 그리고 이제 ModelSim/Questa 의 GUI 상에 Add items to the Project 라는 창이 나타납니다.

Figure 4-2. Adding New Items to a Project



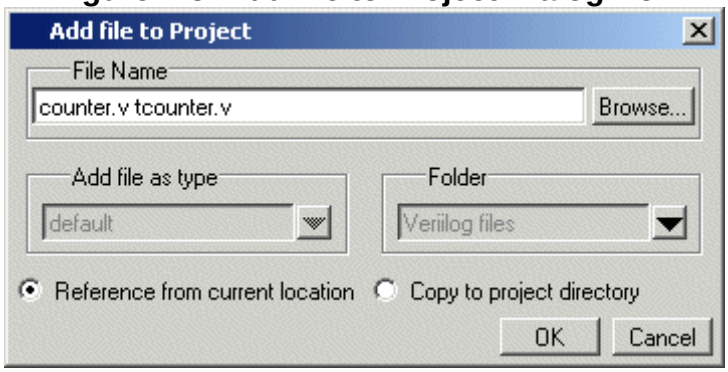
이 창을 통해 유저 분은 Project 에 해당하는 작업을 할 수 있습니다.

새로운 파일(Create New File) 을 만들거나, 기존의 파일(Add Existing File) 을 추가하거나, Simulation Configuration 파일(Create Simulation) 을 만들거나, Project 내에 디렉토리(Create New Folder) 를 만들 수 있습니다.

1. 2개의 파일을 추가하기
   A. Add Existing File 을 선택합니다.

      파일을 추가할 때에는 Project 가 위치한 디렉토리에 Copy 를 할 수도 있고, Project 내에 위치 경로만 지정하는 Reference 방법 2가지가 있습니다.

Figure 4-3. Add file to Project Dialog Box



   B. Browse 버튼을 클릭하여 Copy 한 2개의 파일을 선택합니다.

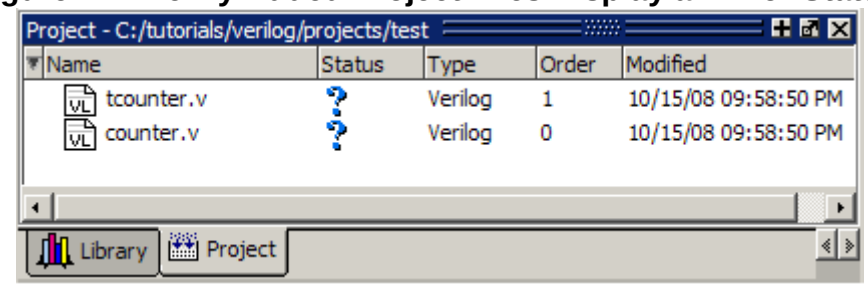
   C. 파일을 선택하고 Reference form current location 을 선택하고 OK 를 클릭합니다.

   D. Add items to the Project 창이 보이면 Close 를 통해 닫습니다.

      이제 유저 분은 Project 에 2개의 파일을 추가하였습니다. 그리고 그 결과는 Project 탭

을 통해서 확인할 수 있습니다.

Figure 4-4. Newly Added Project Files Display a '?' for Status



Changing Compile Order (VHDL)

Verilog Design 을 진행하시는 경우 이 단계를 스킵 하셔도 됩니다.

ModelSim/Questa 에서는 VHDL Design 을 vsim command 를 이용하여 loading 할 때 자동적으로 Design 정보를 모아서 Loading 이 됩니다. 하지만 VHDL Design 같은 경우에 Compile 을 할 때 순서에 맞추어 Compile 하는 것이 매우 중요합니다. (많은 파일을 가지고 Simulation 을 진행하는 경우 Order 맞추어 Compile 하는 것도 큰 작업이 됩니다.) 하지만, Project Flow 방식으로 Simulation 을 진행할 경우엔 아래의 순서대로 하시면 ModelSim/Questa 에서 자동적으로 Order 에 맞추어 Compile 이 진행됩니다.

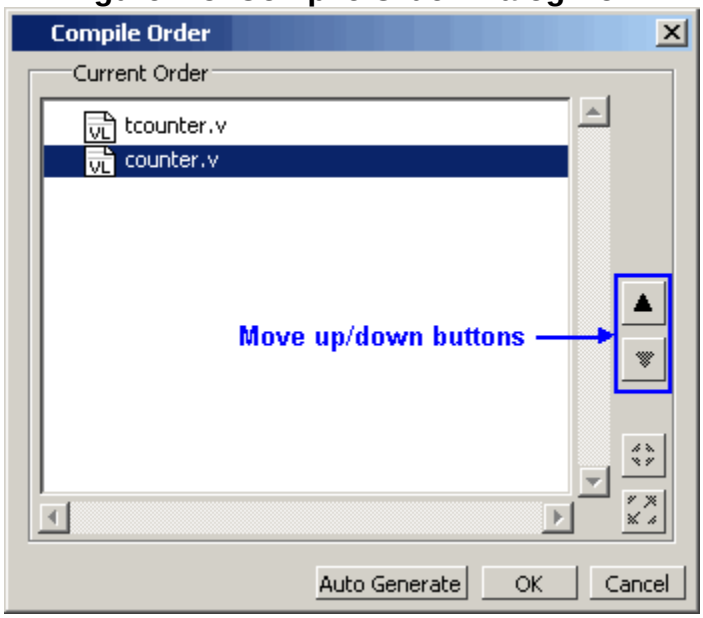
1. Compile order 변경하기

   A. Compile>Compile order 를 선택합니다.

   B. Compile Order 창이 열리면 Auto Generate 를 클릭합니다.

      Modelsim/Questa 에서 2개의 이상의 VHDL 파일을 Auto generate 로 Compile 을 하게 되면 우선 하나의 파일을 Compile 을 하고 나서 다른 파일을 Compile 을 하게 됩니다. 만약 Compile 을 실패하면, 해당 파일은 순서를 맨 마지막으로 이동시키고 나서 다른 파일을 Compile 하는 방법으로 Auto Generate 를 하게 됩니다.
      유저 분이 직접 Compile order 를 지정할 수도 있습니다. 아래의 그림에서처럼 파일을 선택하고 화살표를 클릭해서 순서를 바꿀 수 있습니다.

Figure 4-5. Compile Order Dialog Box



C. 순서를 counter, tcounter 순서로 지정하고 OK 버튼을 클릭합니다.

Compile the Design

1. File Compile 하기

   A. Project 윈도우에서 마우스 오른쪽 버튼을 클릭하고 Compile>Compile All 을 클릭합니다.

   Compile 이 아무런 문제가 없이 진행되면 Project 창에 Status 의 "?" 가 녹색 체크마크로 변합니다. 만약 Compile 이 실패하면 빨간색 X 마크가 되고 Transcript 윈도우에 Compile 실패 메시지가 나타납니다.

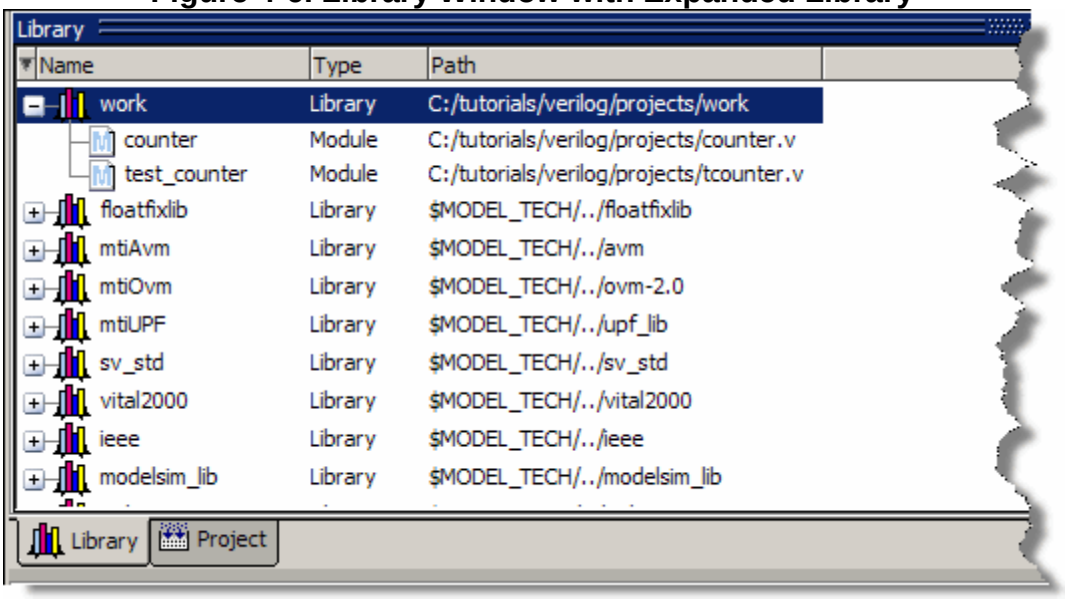
2. Design Unit 보기

   A. Library 탭을 클릭합니다.

   B. Work library 에서 "+" 아이콘을 클릭하여 확대합니다.

   확대하여 보면 Compile 된 2개의 Design unit 을 확인할 수 있습니다. 그리고 unit 에 해당하는 Source file 의 위치정보가 같이 보여집니다.

Figure 4-6. Library Window with Expanded Library



Optimize for Design Visibility

1. vopt command 를 이용하여 Full visibility 를 갖는 Library 로 Optimize 하기.

   A. Transcript 윈도우에 아래의 command 를 입력합니다.

   vopt +acc test_counter -o testcounter_opt

   +acc 는 visibility 관련 옵션이며, -o 는 결과 파일 이름을 지정하는 옵션입니다.

   주의 : vopt command 를 사용할 때 같은 이름으로는 결과 파일이 만들어 지지 않습니다.

Load the Design

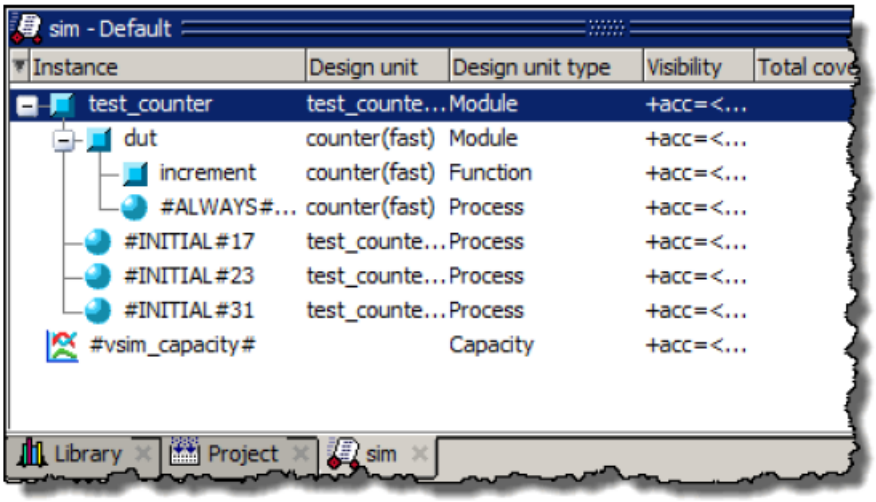
1. test_counter Load 하기.

   A. 위에서 만든 Optimize design 을 vsim command 를 이용하여 load 합니다.

   vsim testcounter_opt

   그러면, 아래의 그림과 같이 해당 Simulation 에 대한 정보를 볼 수 있습니다.

Figure 4-7. Structure(sim) window for a Loaded Design



위의 그림을 보시면 현재 진행하고 있는 Simulation 은 sim 탭에, 나타나고 Project 탭에서는 Project 에 관한 정보를 볼 수 있습니다.

2. Simulation 종료하기

   A. Simulate>End Simulation 을 선택하고 Yes 를 클릭합니다.

Organizing Projects with Folders

만약 유저 분이 현재 진행하는 Project 에 포함되는 파일이 많은 경우엔 Project 안에 Folder 를 만들어서 계층구조로 File 을 정리, 관리할 수 있습니다. 이렇게 만들어진 Folder 실제적으로 만들어지는 디렉토리가 아니라 Project 안에서만 보여지는 Folder 입니다.

Adding Folders

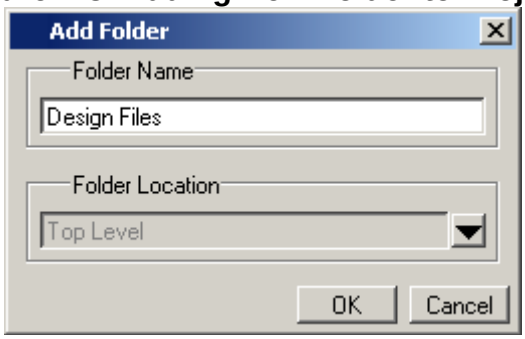
위에서 말한 Folder 추가를 해보겠습니다.

1. 새 폴더 만들기

   A. Project 창에서 마우스 오른쪽 버튼을 클릭하여 Add to Project>Folder 를 선택합니다.

   B. Folder name 부분에 Design Files 라고 입력 합니다.

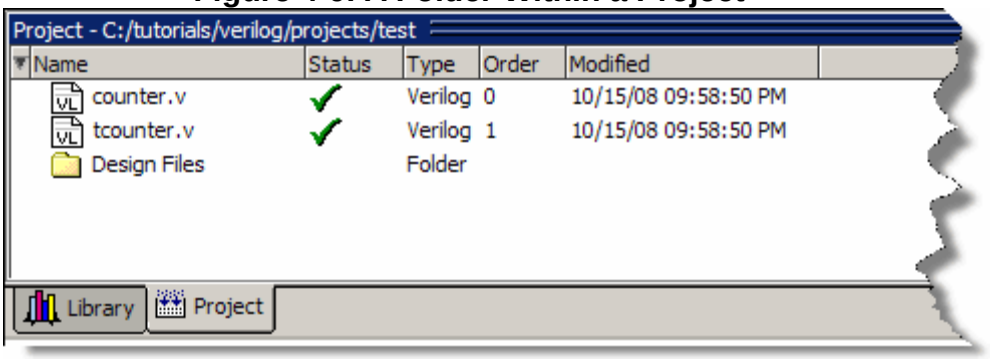
Figure 4-8. Adding New Folder to Project



C. 입력이 다 끝났으면 OK 를 클릭합니다.

Project 창에 Design Files 라는 이름의 Folder 가 나타납니다.

Figure 4-9. A Folder Within a Project

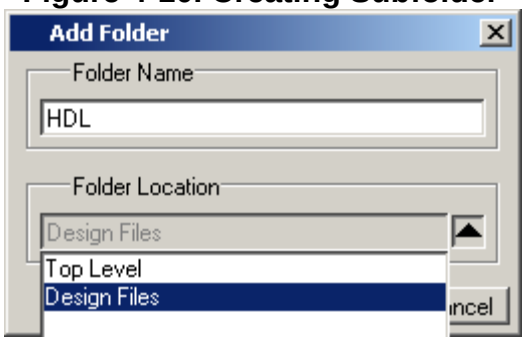


2. Sub-Folder 만들기

A. Project 창에서 마우스 오른쪽 버튼을 클릭하여 Add to Project>Folder 를 선택합니다.

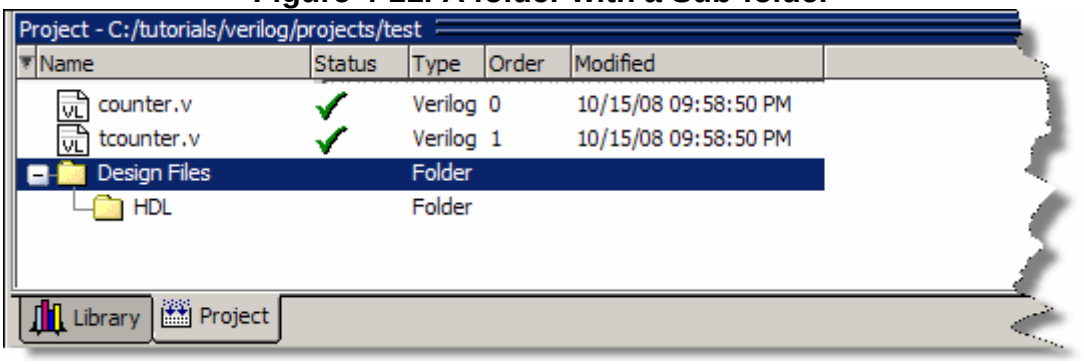
B. Folder name 에 HDL 을 입력합니다.

Figure 4-10. Creating Subfolder



C. Folder Location 부분에 화살표를 눌러 Design Files 를 선택하고 OK 를 클릭합니다. Design Files 의 "+" 아이콘을 클릭하면 Design Files 밑에 HDL Folder 가 만들어진 것을 보실 수 있습니다.

Figure 4-11. A folder with a Sub-folder



Moving Files to Folders

유저 분은 위에서 Folder 를 만들었습니다. 그럼 이제 실제로 Folder 에 파일들을 이동 해보겠습니다.

1. HDL 폴더로 파일 이동시키기

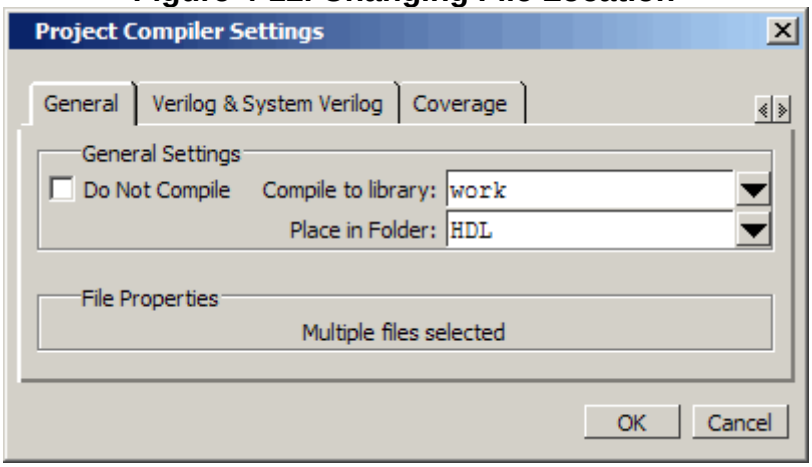
   A. Project 윈도우에서 2개의 파일을 전부 선택합니다.

   B. 파일을 선택했으면 마우스 오른쪽 버튼을 클릭하고 Properties 클릭합니다.

   C. Place in Folder 를 클릭하고 HDL 폴더를 선택합니다.

   D. 선택이 끝났으면 OK 버튼을 클릭합니다.

Figure 4-12. Changing File Location



"+" 아이콘을 클릭하시면 Design File 밑에 HDL 폴더가 있고 HDL 폴더 안에 Design 파일들이 추가된 것을 볼 수 있습니다.

Project 윈도우 상에 Status 가 "?" 로 변경이 되어 있을 겁니다. 이것은 파일이 이동이 되었기 때문에 현재 상태를 알지 못하기 때문에 "?" 로 표시되는 것입니다.

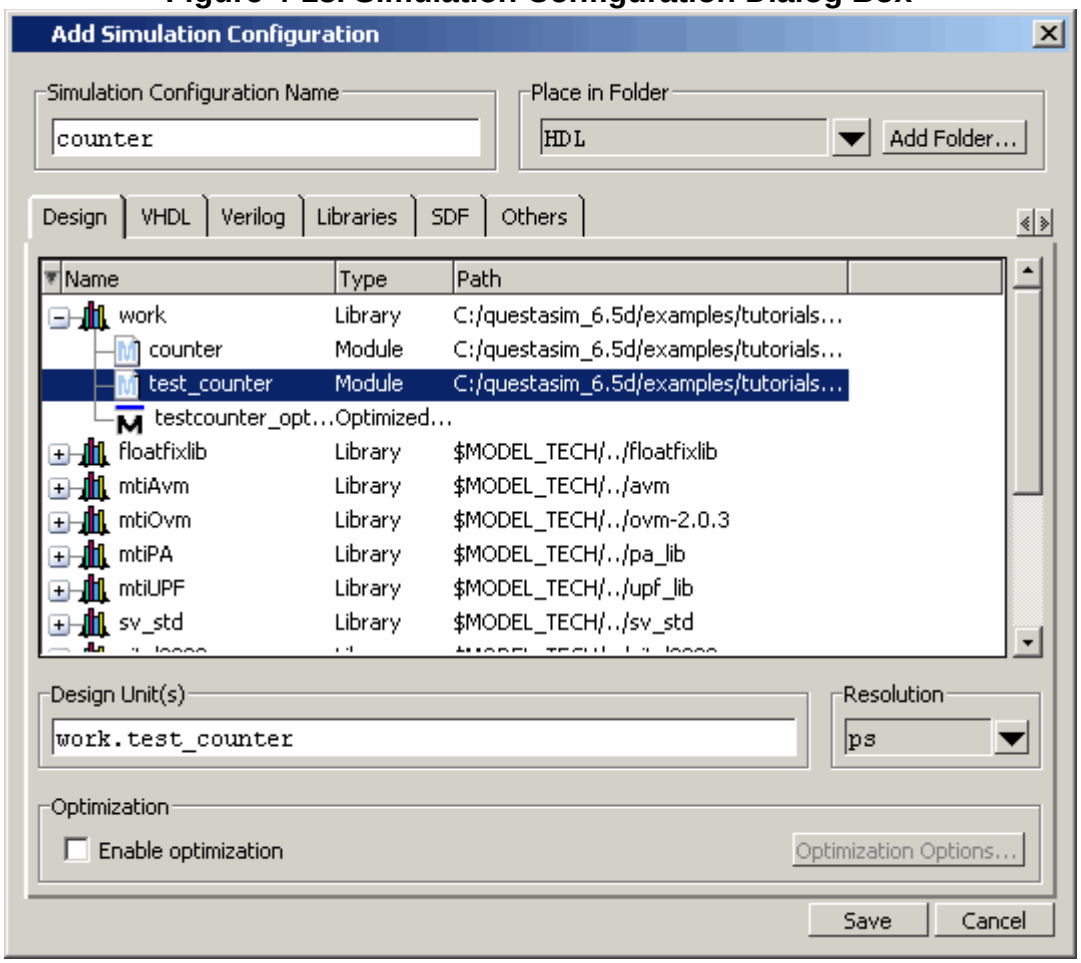
Using Simulation Configurations

Simulation configuration 은 Simulation 을 진행하는 필요한 option 들이 모여 있습니다. 예를 들어 어떤 Design 을 Simulation 을 하는데 resolution time, Hazard option 등을 체크하여 Simulation 을 진행하였다면, 여러분은 Resolution time 과 Hazard 2개의 Simulation Configuration 을 이용하여 Simulation 을 진행한 것입니다. Project Flow 방식으로 진행하면 이런 Simulation configuration 을 따로 저장하여 더블 클릭 하는 것만으로 자동적으로 해당 Option 에 맞추어 Simulation 을 진행할 수 있습니다.

1. 새로운 Simulation Configuration 만들기

   A. Project 윈도우에서 마우스 오른쪽 버튼을 눌러 나오는 팝업 메뉴에서 Add to Project>Simulation configuration 을 선택합니다.

Figure 4-13. Simulation Configuration Dialog Box



B. Simulation Configuration Name 부분에 counter 이라고 입력합니다.

C. Place in Folder 부분을 클릭하여 HDL 로 선택합니다.

D. Design 탭에서 Work 앞의 "+" 아이콘을 클릭하여 test_conter 를 선택합니다.

E. Resolution time 을 ps 로 설정합니다.

F. Enable optimization 부분의 체크를 해제합니다.

G. Verilog 를 사용하는 유저는 Verilog 탭을 선택하여 Enable hazard checking 을 합니다.

VHDL 유저는 진행을 안 하셔도 됩니다.

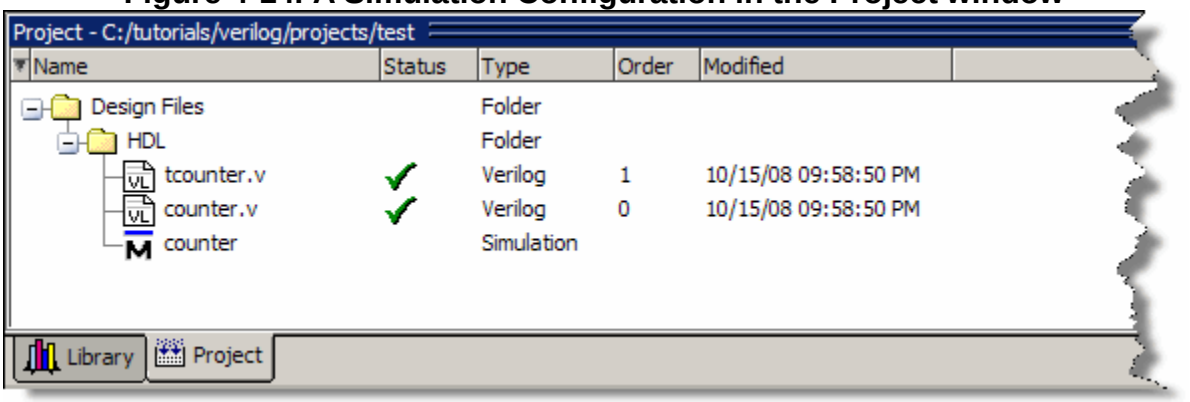
H. Save 버튼을 클릭합니다.
그러면 Project 윈도우 HDL 폴더 밑에 counter 라는 이름의 Simulation Configuration 이 만들어 집니다.

I. 이제 Status 가 "?" 로 표시된 파일들을 Compile 하겠습니다.

Project 창에서 2개의 파일을 전부 선택합니다.

J. Compile>Compile All 을 선택하면 해당 파일이 Compile 이 되고 아래와 같이 Status 가 녹색 체크로 변하게 됩니다.

Figure 4-14. A Simulation Configuration in the Project window

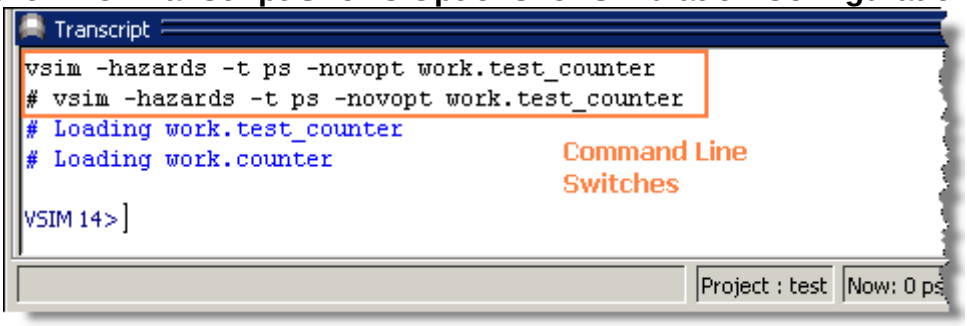


2. Simulation Configuration Load 하기

A. 위의 단계에서 만들어 놓은 Simulation Configuration 인 counter 를 더블 클릭합니다.

위의 Simulation Configuration 을 실행하면 Transcript 윈도우에는 해당하는 Option 들이 command 로 변경되어 보여집니다.

Figure 4-15. Transcript Shows Options for Simulation Configurations



3. Simulation 종료하기
   A. Simulate>End Simulation 을 선택하면 현재 진행중인 Simulation 이 종료 됩니다.
   B. Project 윈도우에서 마우스 오른쪽 버튼을 클릭하여 Close Project 를 선택하여 Project 를 종료합니다.

Chapter 5
Working With Multiple Libraries

이번 챕터에서는 Multiple library 를 이용하여 Simulation 하는 방법을 연습해 볼 것입니다. 여러분들은 Design 을 Simulation 하기 위해 특정 third-party 의 IP, 혹은 다른 Library 등과 연계하여 Simulation 을 할 수도 있습니다.

그런 경우에 대비하여 Resource library 를 만들고 자신의 Design 와 연동하여 Simulation 을 하는 것에 대해 살펴볼 것입니다.

이 챕터에서 예제 파일을 가지고 작업을 진행할 것입니다. 챕터의 예제 파일은 8bit binary up counter 이며 해당 파일은 ModelSim/Questa 를 설치된 경로 안에 있습니다.

Verilog – <*install_dir*>*/examples/tutorials/verilog/libraries/counter.v* and t*counter.v*
VHDL – <*install_dir*>*/examples/tutorials/vhdl/ libraries /counter.vhd* and *tcounter.vhd*

예제 파일들은 Verilog 와 VHDL 둘 다 지원을 하고 있습니다. 가지고 계신 ModelSim/Questa 의 License 에 따라 해당하는 언어 예제를 선택하시면 됩니다.

Creating the Resource Library

Resource library 를 만들기 전에 여러분은 ModelSim/Questa 가 설치된 디렉토리에 있는 modelsim.ini 파일을 읽기 전용으로 만들어 주어야 합니다. 읽기 전용으로 변경하셨으면 이제 아래의 절차에 따라 진행해 주시기 바랍니다.

1. Resource library 를 위한 디렉토리 만들기

   resource library 라는 이름으로 새로운 디렉토리를 만들고 <*install_dir*>*/examples/tutorials/verilog/libraries* 디렉토리에서 counter 파일을 copy 합니다.

2. Test bench 를 위한 디렉토리 만들기

   이제는 Test bench 를 위한 디렉토리를 만들 것입니다. testbench 라는 디렉토리를 만들고 <*install_dir*>*/examples/tutorials/verilog/libraries* 디렉토리에서 tcounter 파일을 copy 합니다.

   이제 유저 분은 이번 챕터에서 2개의 디렉토리를 만들었습니다.

3. 이제 ModelSim/Questa 를 실행하고 Change Directory 를 resource library 로 지정합니다.

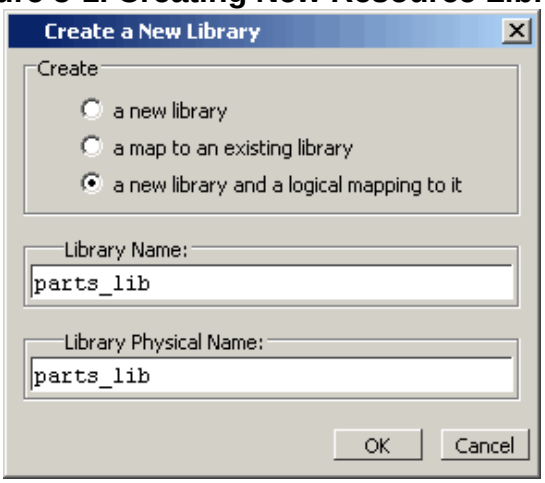
   A. 만약 지난 챕터를 이어서 바로 시작하시는 경우엔 ModelSim/Questa 가 구동 되어 있으니 다시 실행할 필요가 없습니다.

      단축 아이콘을 클릭하거나 vsim command 를 입력하면 ModelSim/Questa 의 GUI 가 올라옵니다.

   B. File>Change Directory 에서 위에서 만든 디렉토리로 지정합니다.

4. Resource library 만들기
   A. File>New>Library 를 선택합니다.
   B. Library name 부분에 parst_lib 을 입력합니다.

Figure 5-1. Creating New Resource Library



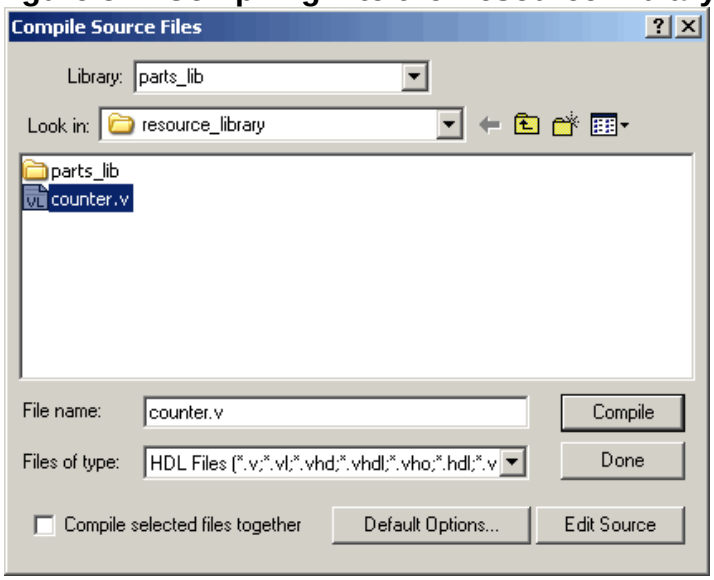
Library name 부분을 입력하면 자동적으로 Library Physical Name 부분도 채워집니다. 위와 같이 입력이 되었으면 OK 를 클릭합니다.

5. Resource Library 에 counter 파일 Compile 하기

   A. 메인 메뉴의 Compile 아이콘을 클릭합니다.

   B. Library 부분의 화살표를 클릭하여 parts_lib 을 선택합니다.

Figure 5-2. Compiling into the Resource Library



C. counter.v 파일을 선택하고 더블 클릭을 하거나 Compile 버튼을 클릭합니다.

D. Done 버튼을 클릭합니다.

6. 디렉토리를 test bench 로 변경하기
   A. File>Change Directory 를 선택하고 test bench 디렉토리를 선택합니다.

Creating the Project

Counter 의 test bench 를 위한 Project 만들기

1. 프로젝트 만들기

   A. File>New>Project 를 선택합니다.
   B. Project name 부분을 counter 로 선택합니다.

   C. Project Location 혹은 Default library name 부분은 변경하지 않습니다.

   D. OK 를 선택합니다.

2. Project 에 test bench 추가하기

   A. Add Existing File 을 선택합니다.
   B. Browse 버튼을 클릭하고 tcounter.v 파일을 추가합니다.

3. Compile 하기
   A. tcounter.v 파일을 선택하고 마우스 오른쪽 버튼을 클릭한 후 Compile>Compile

Selected 를 선택합니다.

Loading Without Linking Libraries

지금 과정에서는 이전에서 만들어 놓은 parts_lib library 를 link 를 걸어서 Simulation 을 진행해 볼 것입니다. 이 과정은 Verilog 과 VHDL 이 진행과정이 서로 다릅니다.

Verilog

Optimize the Verilog Design for Debug Visibility

1. vopt 를 이용하여 full visibility 를 가지는 optimize design 만들기

   A. Transcript 윈도우에서 다음과 같이 command 를 입력합니다.

   vopt +acc test_counter –o testcounter_opt

   B. quit –sim 을 입력합니다.

VHDL

Optimize the VHDL Design for Debug Visibility

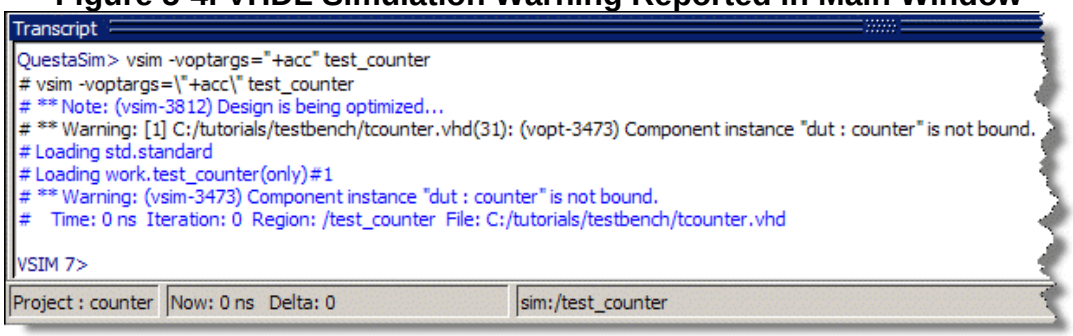
1. vopt 를 이용하여 full visibility 를 가지는 optimize design 만들기

   A. Transcript 윈도우에서 다음과 같이 command 를 입력합니다.

   vopt +acc test_counter –o testcounter_opt

   위와 같이 command 를 입력하면 아래와 같은 3473 Warning 메시지를 보게 될 것입니다. 이 메시지는 dut counter 에 대한 정보를 읽어오지 못해 발생하는 메시지 입니다.

Figure 5-4. VHDL Simulation Warning Reported in Main Window



   B. verror 3473 이라고 입력합니다.

verror 은 warning, error 메시지에 대해 좀더 자세한 정보를 보기 원할 때 사용되는 command 입니다.

C. quit –sim 을 입력합니다.

Linking to the Resource Library

유저 분이 특정한 Library 를 Link 를 걸어서 Simulation 을 진행하려는 경우엔 search library 라는 옵션을 사용하여 Simulator 를 동작시켜야 합니다.

1. Simulation 을 진행하면서 특정한 library 찾기

A. 메인 메뉴에서 Simulate 아이콘을 클릭하거나 Simulate>Simulate 를 클릭합니다.

B. work library 에서 "+" 아이콘을 클릭하여 test_counter 를 선택합니다.

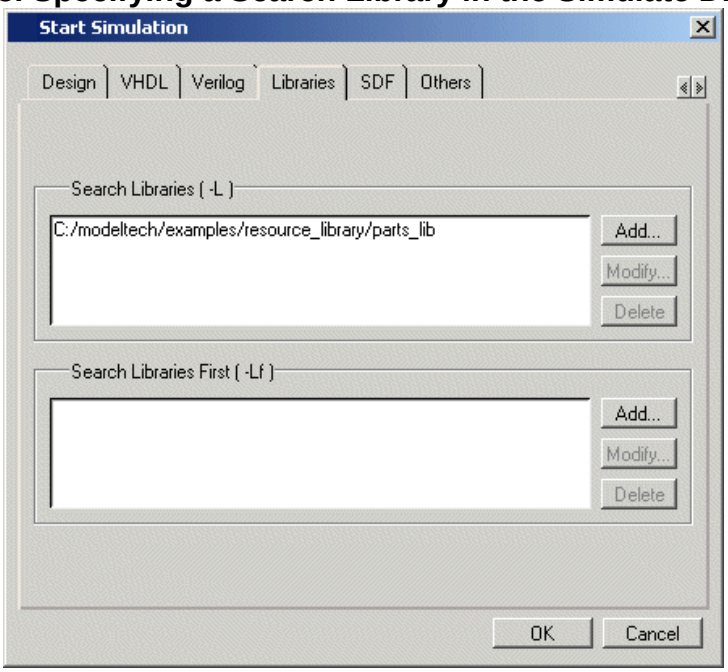
C. Enable optimization 체크 박스 의 체크를 해제합니다.

D. Library 탭을 선택합니다.

E. Search Library 에서 Add 버튼을 클릭하여 resource library 에 parts_lib 을 선택하고 OK 를 클릭합니다.

아래의 그림처럼 Library 가 추가된 것을 확인할 수 있습니다.

Figure 5-5. Specifying a Search Library in the Simulate Dialog Box



F. OK 버튼을 클릭합니다.

Permanently Mapping VHDL Resource Libraries

위에서와 같이 VHDL Design 을 Simulation 을 진행하려고 할 때마다 Library 를 추가해서 작업을 하면 유저 분은 상당히 많은 불편을 감수하면서 Project 마다 Link 를 걸어주어야 합니다. 하지만 ModelSim/Questa 환경 설정 파일인 modelsim.ini 파일을 수정하시면 그런 불편 없이 사용이 가능합니다.

1. ModelSim/Questa 가 설치된 디렉토리로 이동합니다.
2. 만약을 대비해 반드시 Backup 파일을 만들어 둡니다.
3. Modelsim.ini 파일의 읽기전용을 해제합니다.
4. 파일을 워드 패드를 이용해 Open 합니다.

   [Library] 부분에 위에서 만들었던 parts_lib 을 등록합니다.

   Ex) parts_lib = c:/libraries/parts_lib
5. 파일을 저장합니다.
6. 저장한 파일을 읽기 전용파일로 변경합니다.
7. Simulation 종료하기

   A. Simulate>End Simulation 을 선택하면 현재 진행중인 Simulation 이 종료 됩니다.

Chapter 6
Simulating SystemC Designs

ModelSim/Questa 에서는 SystemC 로 설계된 Design 역시 Simulation 을 진행할 수 있습니다. HDL Design 과 마찬가지로 Simulation 과 Debugging 환경을 제공합니다.

이번 챕터에서는 먼저 SystemC 로 만들어진 아주 간단한 Basic 이라는 Design 을 Simulation 할 것입니다. 그 다음엔 HDL 과 SystemC 로 구현되어 있는 ring buffer 를 가지고 Simulation 을 진행할 것입니다. 예제 파일의 위치는 다음과 같습니다.

SystemC – *<install_dir>/examples/systemc/sc_basic*

SystemC/Verilog – *<install_dir>/examples/systemc/sc_vlog*

SystemC/VHDL – *<install_dir>/examples/systemc/sc_vhdl*

Setting up the Environment

SystemC 를 사용하기 위해서는 SystemC 관련 License 가 있어야 합니다. 만약 SystemC 관련 License 가 없으시다면 이번 챕터는 진행할 수 없습니다. 또한 SystemC 를 사용하기 위해서는 다음 표에서 보여지는 Platform 별 C Compiler 가 필요합니다.

Table 6-1. Supported Platforms for SystemC

Platform/OS	Supported compiler versions	32-bit support	64-bit support
Intel and AMD x86-based architectures (32- and 64-bit) SUSE Linux Enterprise Server 9.0, 9.1, 10, 11 Red Hat Enterprise Linux 3, 4, 5	gcc 4.0.2, gcc 4.1.2, gcc 4.3.3 VCO is linux (32-bit binary) VCO is linux_x86_64 (64-bit binary)	yes	yes
Solaris 8, 9, and 10	gcc 4.1.2	yes	no
Solaris 10 on x86	gcc 4.1.2	yes	yes
Windows XP[1], Vista and 7	Minimalist GNU for Windows (MinGW) gcc 4.2.1	yes	no

1. SystemC supported on this platform with gcc-4.2.1-mingw32vc9.

보다 자세한 정보는 User Manual 의 SystemC 관련 부분을 살펴보시기 바랍니다.

Preparing an OSCI SystemC Design

Open SystemC Initiative – OSCI 와 호환되는 SystemC Design 을 Simulation 을 하기 위해서는 아래의 사항을 체크해 주셔야 합니다.

- sc_main() 으로 선언되어 있는 함수를 sc_module() 로 변경합니다
- sc_initialize() 는 삭제합니다.
- 최상위 module 은 sc_module_export 를 사용합니다.

1. 디렉토리 만들고 예제 파일 복사하기

   새로운 디렉토리를 만들고 *<install_dir>/examples/systemc/sc_basic* 디렉토리에서 파일을 새로운 디렉토리로 파일을 복사합니다.

2. 이제 ModelSim/Questa 를 실행하고 Change Directory 를 resource library 로 지정합니다.

   A. 만약 지난 챕터를 이어서 바로 시작하시는 경우엔 ModelSim/Questa 가 구동 되어있으니 다시 실행할 필요가 없습니다.

      단축 아이콘을 클릭하거나 vsim command 를 입력하면 ModelSim/Questa 의 GUI 가 올라옵니다

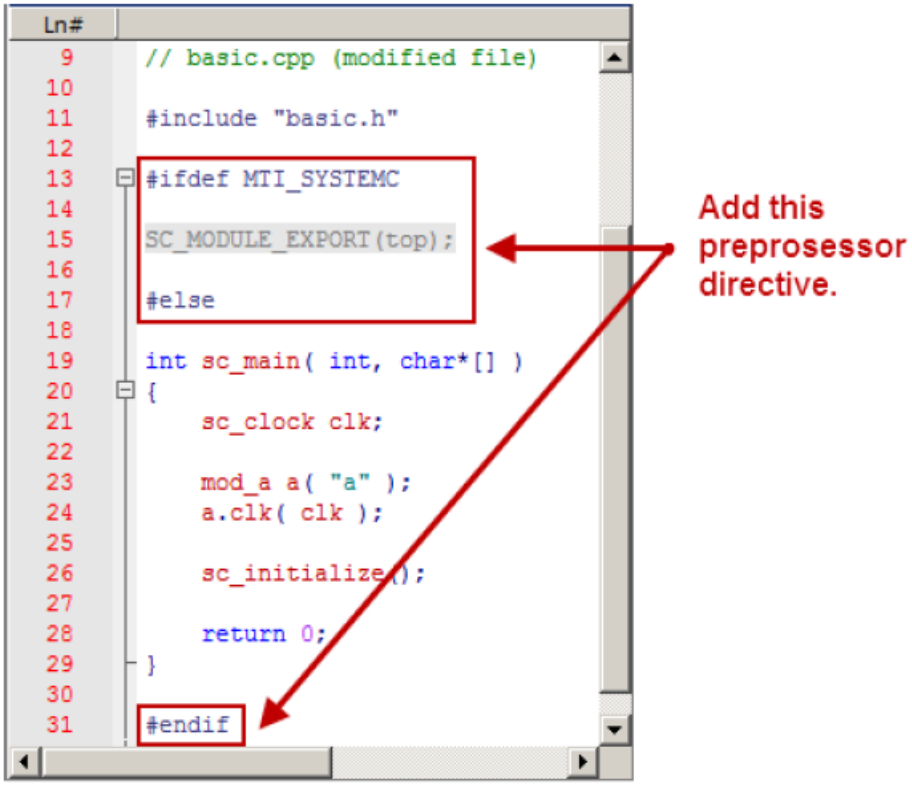
   B. File>Change Directory 에서 위에서 만든 디렉토리로 지정합니다.

3. 메인 메뉴의 File>Open 를 선택하여 ModelSim/Questa 에서 제공하는 Text editor 를 이용해 basic_orig.cpp 파일을 엽니다.

   A. 파일은 읽기 전용일 것입니다. Source 윈도우에서 마우스 오른쪽 버튼을 클릭하여 읽기전용을 해제합니다.

   B. 아래의 그림처럼 파일을 수정합니다.

Figure 6-1. The SystemC File After Modifications



C. 파일을 저장합니다.

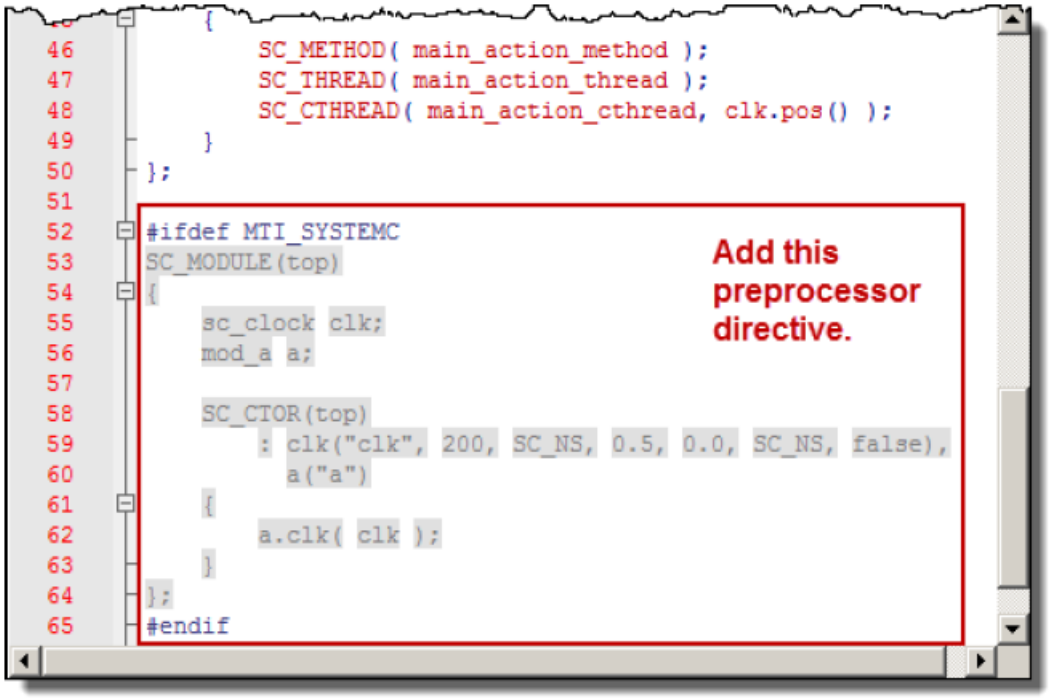
수정한 파일은 sc_basic/gold 디렉토리에 있으니 복사해서 사용해도 됩니다.

1. basic_orig.h 파일 수정하기

A. 파일은 읽기 전용일 것입니다. Source 윈도우에서 마우스 오른쪽 버튼을 클릭하여 읽기 전용을 해제합니다.

B. 그리고 아래의 그림처럼 수정합니다.

Figure 6-2. Editing the SystemC Header File.



C. basic.h 로 파일을 저장합니다.
이 파일 역시 /sc_basic/gold 에 있는 파일을 카피해서 사용해도 됩니다.

이제 Simulation 을 하기 위한 파일 수정이 끝났습니다.

Compiling a SystemC-only Design

파일 수정이 끝났으면 이제는 파일들을 Compile 해야 합니다. ModelSim/Questa 에서는 sccom command 를 이용하여 compile 할 수 있습니다.

1. Work library 만들기

   A. Transcript 윈도우에 vlib work 라고 입력합니다.

2. SystemC 파일 Compile 하고 Link 하기

   A. 계속해서 아래의 command 를 입력합니다.

   sscom –g basic.cpp

   B. 컴파일이 끝났으면 SystemC object 와 link 를 걸어야 합니다. 아래의 command 를 입력합니다.

sccom – link

이제 Compile 과 link 가 성공적으로 마무리 되었습니다. 이제 다음 단계에서는 SystemC 와 HDL 이 Mix 되어 있는 Design 에 대해 진행을 할 것입니다.

Mixed SystemC and HDL Example

이제 여러분은 HDL Instant 를 가지고 있는 SystemC Design 을 가지고 작업을 진행할 것입니다,

1. 새로운 디렉토리 만들고 파일 복사하기

   새로운 디렉토리를 만들고 *<install_dir>/examples/systemc/sc_vlog* 디렉토리에서 파일을 새로운 디렉토리로 파일을 복사합니다.

   VHDL 을 사용하시는 분은 *install_dir>/examples/systemc/sc_vhdl* 의 파일을 새로운 디렉토리로 카피합니다.

2. 이제 ModelSim/Questa 를 실행하고 Change Directory 를 resource library 로 지정합니다.

   A. 만약 지난 챕터를 이어서 바로 시작하시는 경우엔 ModelSim/Questa 가 구동 되어 있으니 다시 실행할 필요가 없습니다.

      단축 아이콘을 클릭하거나 vsim command 를 입력하면 ModelSim/Questa 의 GUI 가 올라옵니다

   B. File>Change Directory 를 통해 위에서 만든 디렉토리로 지정합니다.

3. Work library 만들기

   A. Transcript 윈도우에 vlib work 를 입력합니다.

4. Compile 하기

   A. Verilog

      Transcript 윈도우에 vlog *.v 를 입력합니다.

   B. VHDL

      Transcript 윈도우에 vcom -93 *.vhd 를 입력합니다.

5. Verilog ringbuf 모듈을 위해 외부 모듈 선언하기

A. Verilog

Transcript 윈도우에 **scgenmod –map "scalar=bool" ringbuf>ringbuf..h** 를 입력합니다. –map "scalar=bool" 옵션은 외부 포트를 내부 논리값으로 사용하기 위해 사용됩니다.

B. VHDL

scgenmod ringbuf<ringbuf.h 를 입력합니다.

이제 ringbuf.h 파일을 열어보면 다음과 같습니다.

Figure 6-3. The ringbuf.h File.

C:/examples/systemc/sc_vlog/ringbuf.h - Default

```
Ln#
 1  #ifndef _SCGENMOD_ringbuf_
 2  #define _SCGENMOD_ringbuf_
 3
 4  #include "systemc.h"
 5
 6  class ringbuf : public sc_foreign_module
 7  {
 8  public:
 9      sc_in<bool> clock;
10      sc_in<bool> reset;
11      sc_in<bool> txda;
12      sc_out<bool> rxda;
13      sc_out<bool> txc;
14      sc_out<bool> outstrobe;
15
16
17      ringbuf(sc_module_name nm, const char* hdl_name,
18         int num_generics, const char** generic_list)
19       : sc_foreign_module(nm),
20         clock("clock"),
21         reset("reset"),
22         txda("txda"),
23         rxda("rxda"),
24         txc("txc"),
25         outstrobe("outstrobe")
26      {
27          elaborate_foreign_module(hdl_name, num_generics, generic_list);
28      }
29      ~ringbuf()
30      {}
31
32  };
33
34  #endif
```

Test_ringbuf.cpp 파일을 보면 아래와 같이 test_ringbuf.h 파일을 포함하고 있는 것을 볼 수 있습니다.

Figure 6-4. The test_ringbuf.cpp File

6. SystemC 파일 Compile 하고 Link 걸기

   A. Transcript 윈도우에 **sccom –g test_ringbuf.cpp** 를 입력합니다.

   B. Transcript 윈도우에 **sccom –link** 를 입력합니다.

7. Full visibility optimize design 만들기

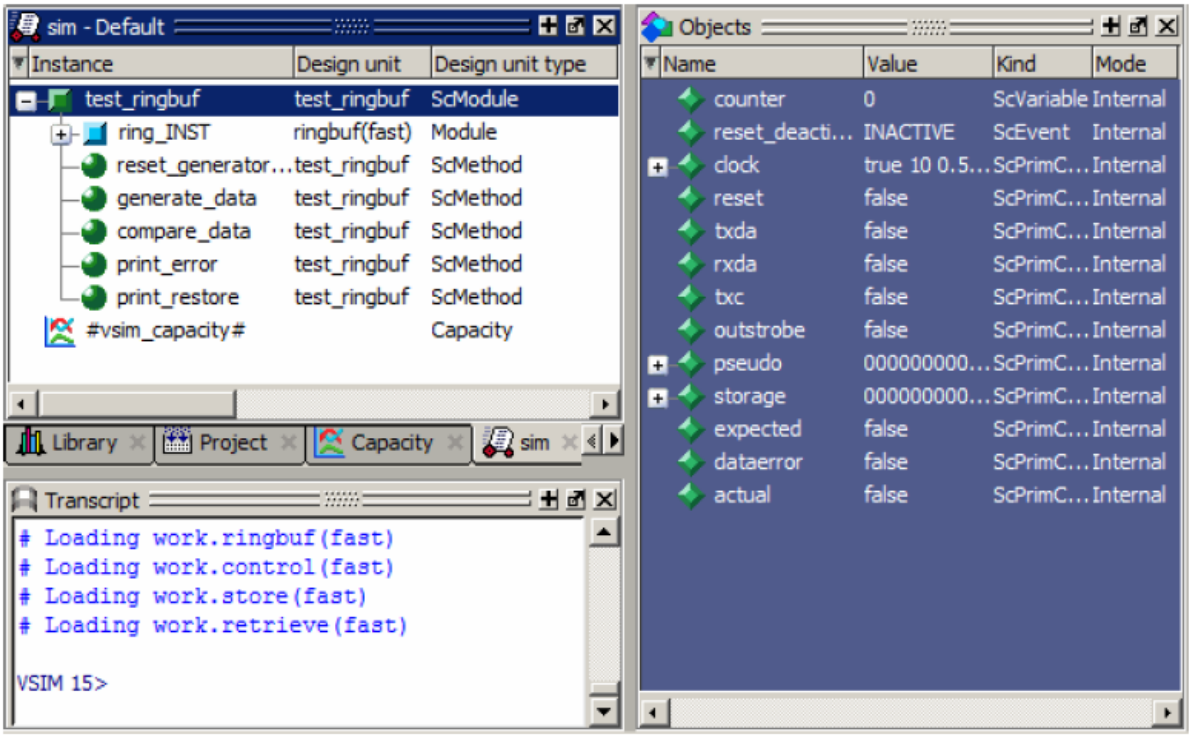
   A. Transcript 윈도우에 **vopt +acc test_ringbuf –o test_ringbuf_opt** 를 입력합니다,

8. 디자인 Load 하기

   A. Transcript 윈도우에 **vsim test_ringbuf_opt** 를 입력합니다.

      디자인이 로드가 되면서 Process 윈도우나 Objects 윈도우가 열릴 것 입니다. 해당 윈도우는 메인 메뉴의 View 메뉴를 통해서 새로운 윈도우를 열거나 닫을 수 있습니다.

Figure 6-5. The test_ringbuf Design



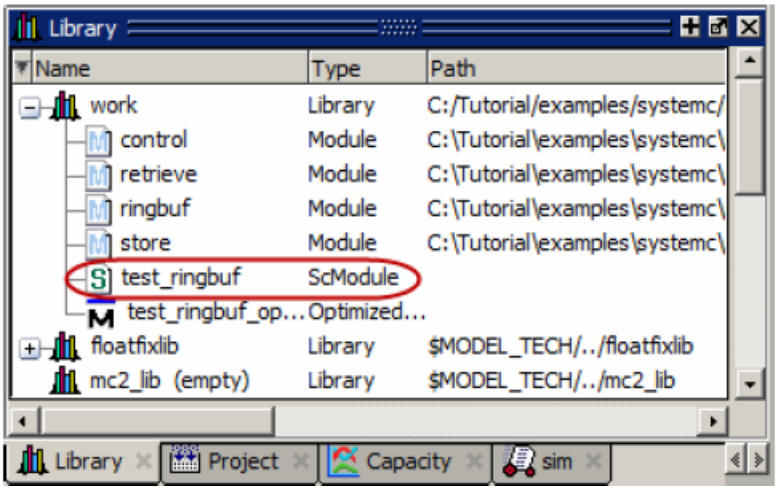
Viewing SystemC Objects in the GUI

SystemC 의 Design Object 들은 ModelSim/Questa 의 GUI 상에 녹색 아이콘으로 표시됩니다. 그리고 Object 의 종류에 따라 해당 아이콘이 사각형, 원 혹은 다이아몬드 형태로 표시됩니다.

1. Library 윈도우에서 Objects 보기

   A. Library 윈도우에서 work library 앞의 "+" 아이콘을 클릭하여 펼칩니다. SystemC Object 는 녹색 S 마크로 표현됩니다.

Figure 6-6. SystemC Objects in the work Library



2. Wave 윈도우에 Object 추가하기
   A. Structure 윈도우(sim) 에서 test_ringbuf 를 선택하고 마우스 오른쪽 버튼을 클릭하여 Add>To wave>All items in region 을 선택합니다.

Setting Breakpoints and Stepping in the Source Window

Source 윈도우를 통해 HDL 에서 breakpoint 를 설정한 것과 마찬가지로 SystemC 파일에서도 Breakpoint 를 설정할 수 있습니다. ModelSim/Questa 의 C debug 기능을 통해 작업이 이루어 집니다. C Debug 에 대해서는 유저 매뉴얼에서 C Debug 부분을 살펴보시면 더욱 자세히 알 수 있습니다.

1. C Debug 를 이용하여 breakpoint 를 사용하기 위해서는 ModelSim/Questa 의 설정을 변경해 주셔야 합니다.

   A. Tools>C Debug>Allow lib step 혹은 Enable auto step 을 선택합니다.

2. Breakpoint 설정하기

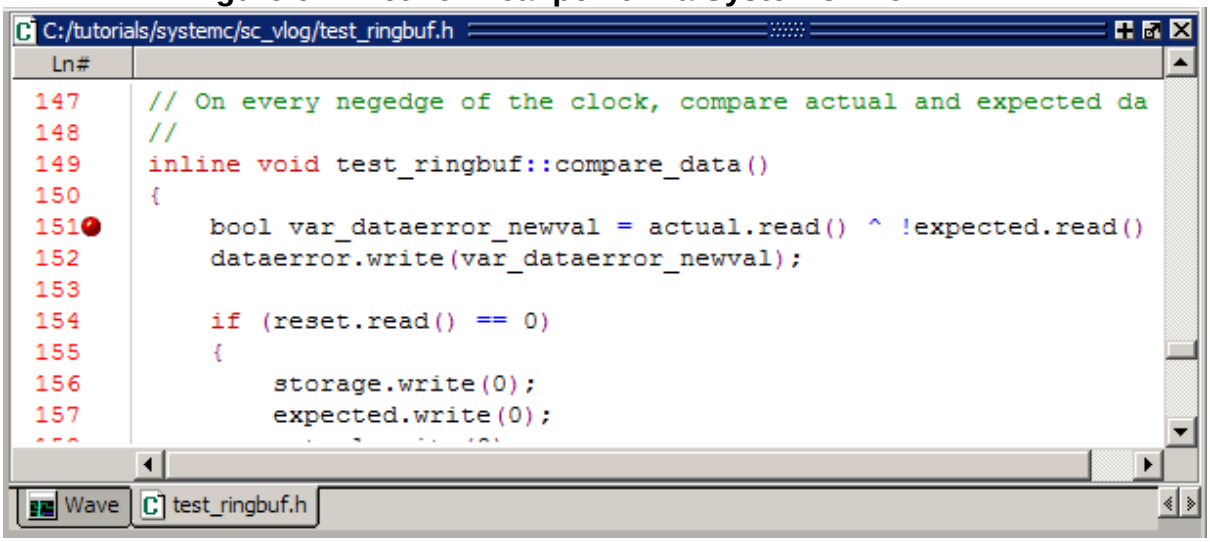
   A. Structure (sim) 탭에서 test_ringbuf 를 선택하여 Source 윈도우에 Open 합니다.

   B. Source 윈도우에서 VHDL, Verilog 상관 없이 150번 line 으로 이동합니다.

   C. 아래의 이미지에서 보이는 것처럼 해당 line 에 breakpoint 를 설정합니다.

      Verilog : bool var_dataerror_newval = actual.read()...
      VHDL : sc_logic var dataerror newval = actual.read ...

Figure 6-7. Active Breakpoint in a SystemC File



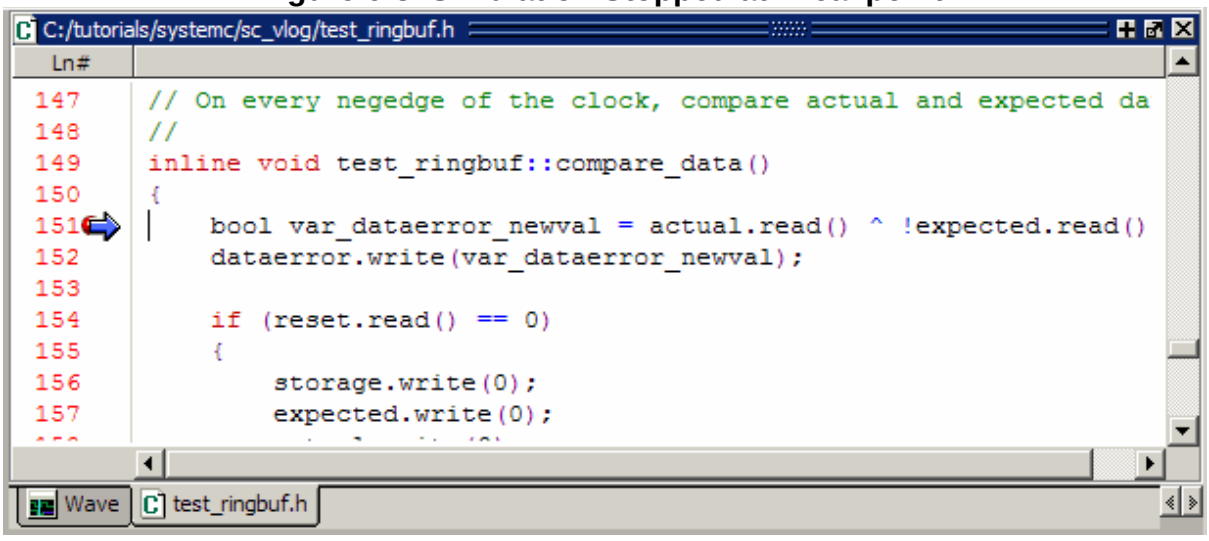
3. Simulation 동작 하기

A. run 500 을 입력합니다.

Simulation 이 진행을 하시면, 기존의 breakpoint 설정하셨던 거와 마찬가지고 해당 Source 윈도우에서 해당 line 에 화살표가 표시되면서 Simulation 이 멈추어지며 아래의 메시지가 Transcript 윈도우에 표시됩니다.

```
# C breakpoint c.1
# test_ringbuf::compare_data (this=0x27c4d08) at test_ringbuf.h:151
```

Figure 6-8. Simulation Stopped at Breakpoint



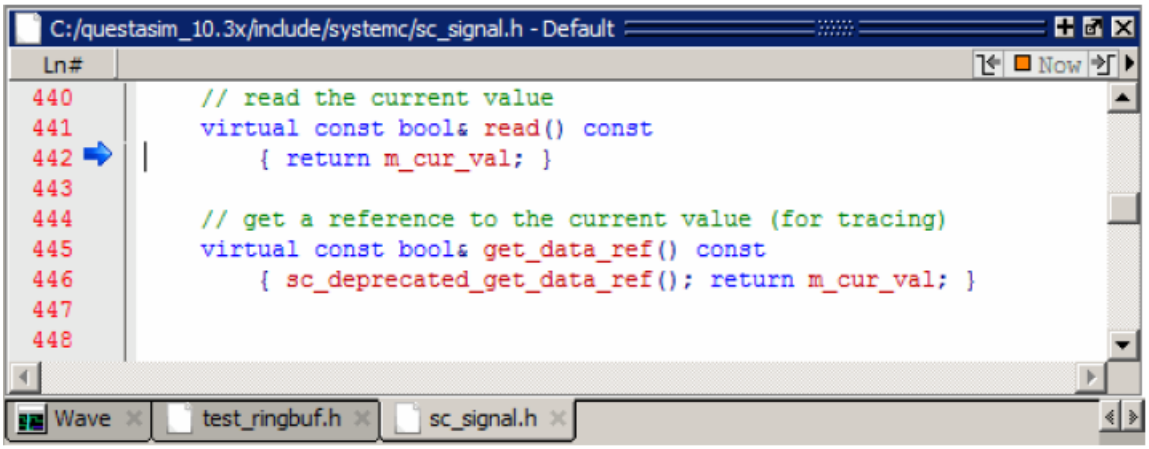
B. 메인 메뉴에서 step 아이콘을 클릭합니다. 

해당 아이콘을 클릭하면 다음 Statement 도 이동을 합니다.

다음 Statement 는 파일과는 상관없이 Simulation 을 진행되어 가는 동안에 발생하는

Function 에 따로 동작하게 됩니다. 아래의 이미지처럼 sc_signal.h 파일의 440 번 line 으로 이동됩니다.

Figure 6-9. Stepping into a Separate File



C. Continue run 아이콘을 클릭하면 다시 simulation 이 진행되며  breakpoint 에서 동작이 멈추어 집니다.

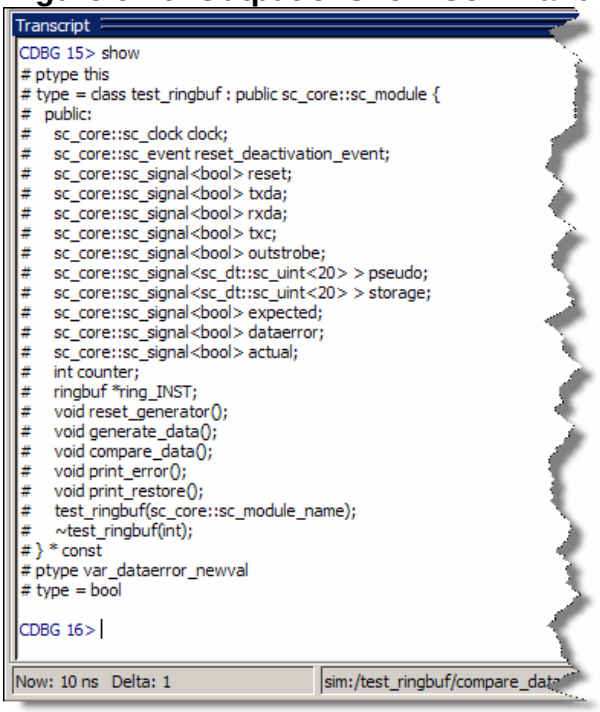
Examining SystemC Objects and Variables

SystemC 의 Object 의 value 는 Object 윈도우에서 볼 수 있습니다. 그리고 examine 라는 command 를 입력해서도 볼 수 있습니다.

1. sc_signal 의 type 과 value 보기

   A. CDBG 프롬프트 상태에서 show 를 입력하면, Transcript 윈도우에 Design 의 Object 들의 Type 과 Value 가 보여집니다.

Figure 6-10. Output of show Command



B. examine dataerror 를 입력합니다.

True 라는 메시지가 출력됩니다.

2. SystemC variable Value 보기

A. 위에서와 마찬가지로 examine counter 를 입력합니다.

"32'hFFFFFFFF" 라는 value 가 메시지로 표시됩니다.

Removing a Breakpoint

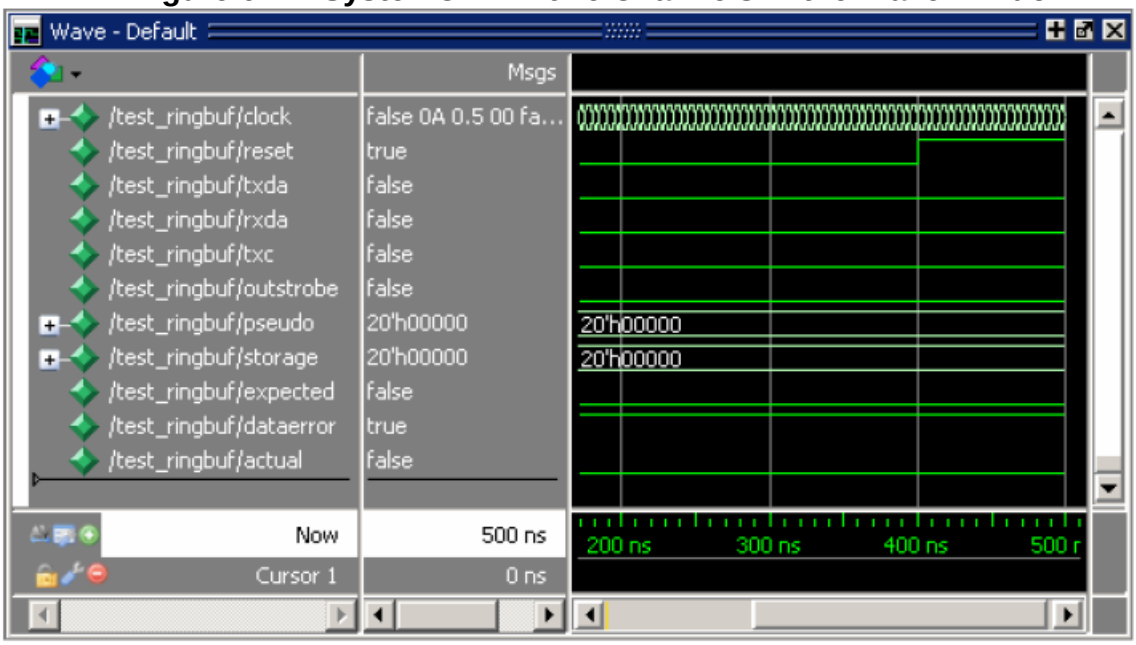
1. Source 윈도우로 들어가 열려있는 test_ringbuf.h 파일에 설정된 breakpoint 를 제거하기

A. 440 line 에서 마우스 오른쪽 버튼을 클릭하여 remove breakpoint 를 설정합니다.

2. Simulation 구동하기

A. run 500ns 를 입력하고 wave 윈도우에서 파형을 살펴봅니다.

Figure 6-11. SystemC Primitive Channels in the Wave Window



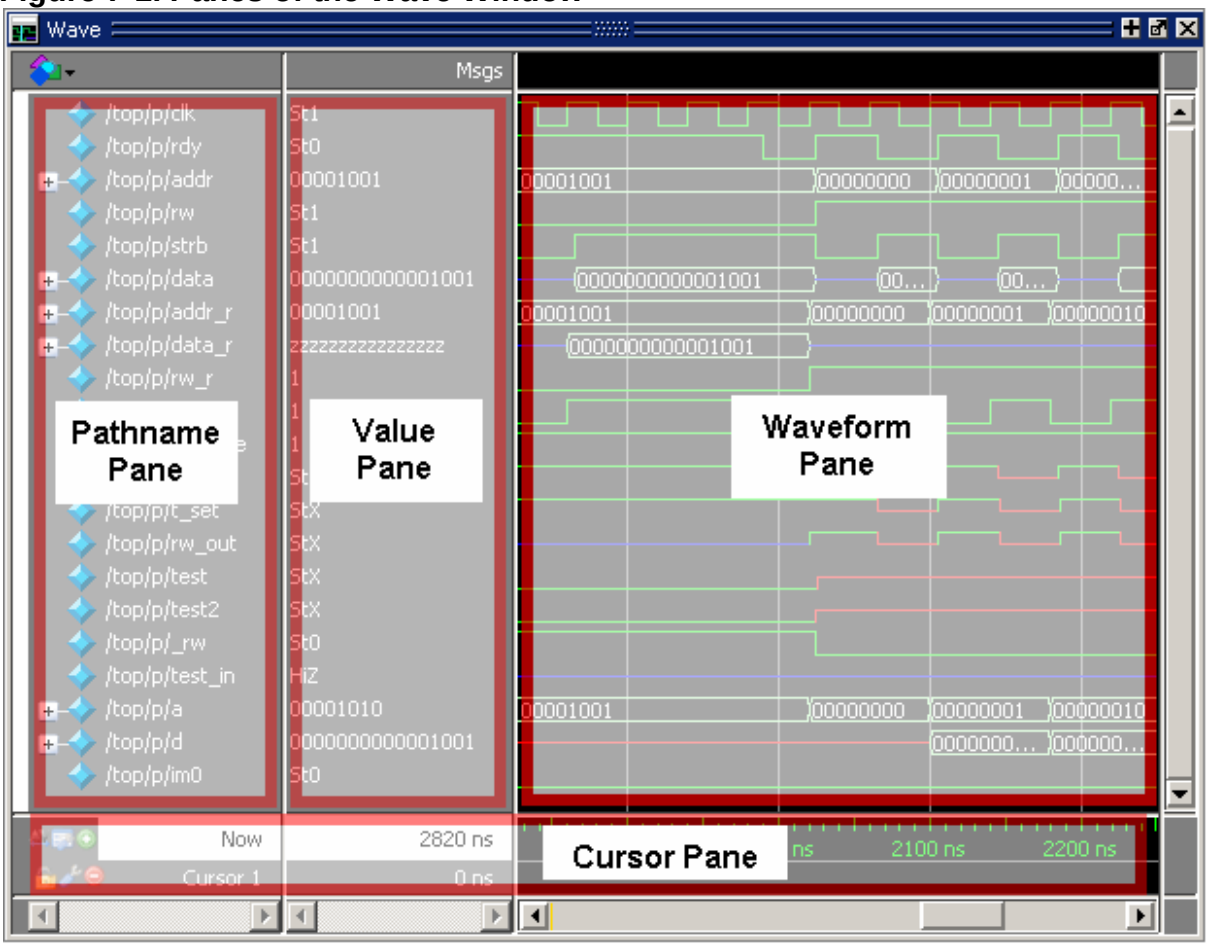
3. C debugger 종료하기

   A. Tool>C Debug > Quit C Debug 을 선택하면 현재 진행중인 C debugger 가 종료 됩니다.

Chapter 7
Analyzing Waveforms

Simulation 결과는 Wave 윈도우에서 확인 할 수 있습니다. Wave 윈도우는 3개의 작은 창으로 구성되어 있습니다. Signal 이름을 보는 Pathname pane, Value 를 확인하는 Value pane, 마지막으로 파형이 보여지는 Waveform pane 입니다. 이 3개의 Pane 은 유저가 원하는 사이즈로 resize 가 가능합니다.

Figure 7-1. Panes of the Wave Window



Loading a Design

이번에 사용할 Design 을 기존 Basic Simulation 에서 사용했던 Design 을 가지고 진행을 할 것입니다.

1. 이전 Chapter 를 진행하시고 ModelSim/Questa 를 종료를 안 했다면, 이어서 하면 되고, 만약 종료하셨으면, Tool 를 새로 동작시킵니다.

A. 단축 아이콘을 더블 클릭하거나, vsim 을 입력합니다.

2. 디자인 load 하기

   A. File>Change Directory 에서 기존 Basic Simulation 에서 사용했던 디렉토리를 지정합니다.

   B. 디렉토리를 지정하면 이전에 챕터를 진행하면서 만들어 놓았던 work library 가 올라와 있는 것을 볼 수 있습니다.

   C. 기존 챕터를 진행을 하며 만들어 놓았던 optimize design 을 load 합니다.

      vsim testcounter_opt

Add Objects to the Wave Window

ModelSim/Questa 에서는 Wave 윈도우로 Object 를 추가하는 여러 가지 방법들이 있습니다. 이번에는 그 방법들을 하나 하나씩 연습해 보겠습니다.

1. Object 윈도우로부터 Object 추가하기

   A. View>Object 를 선택하여 Object 윈도우를 엽니다.

   B. Object 윈도우에서 한 Object 를 선택하고, 마우스 오른쪽을 클릭하여 Add>To Wave>Signals in Region 를 선택합니다.

      Wave 윈도우가 열리면서 Wave 윈도우에 선택한 Signal 과 같은 region 에 있는 signal 들이 보여집니다.

2. Wave 윈도우 외부로 독립시키기

   일반적으로 ModelSim/Questa 에서 Wave 윈도우는 메인 윈도우 안에서 하나의 창으로 열립니다. 하지만 외부로 창을 독립시켜 단독 윈도우로 사용하는 것도 가능합니다. 기본설정 역시 변경하는 것이 가능합니다. 기본 설정 변경은 유저 매뉴얼에서 Simulator GUI Preference 를 참조하시면 됩니다.

   A. Wave 윈도우 오른쪽 상단의 아이콘 중에 undock 아이콘을 클릭합니다. 

      아이콘을 클릭하면 Wave 윈도우는 독립된 창으로 열립니다.

3. Drag and Drop 을 통해서 Object 추가하기

   ModelSim/Questa 의 여러 윈도우-Structure, Objects ad Locals-에서 Drag and Drop 을 통해서 Signal 을 추가할 수 있습니다.

A. Wave 윈도우에서 Eidt>Select All 을 선택하고 Edit>Delete 를 선택하여 모든 signal 을 삭제합니다.

B. Work space 윈도우의 Structure(sim) 탭에서 instance 를 선택하고 drag and drop 으로 wave 윈도우에 추가합니다.

C. Wave 윈도우에서 Eidt>Select All 을 선택하고 Edit>Delete 를 선택하여 모든 signal 을 삭제합니다.

4. add wave command 사용하여 추가하기

A. Transcript 윈도우에 add wave * 를 입력합니다.

B. Run 500ns 를 입력하고 wave 윈도우에서 파형을 봅니다.

Zooming the Waveform Display

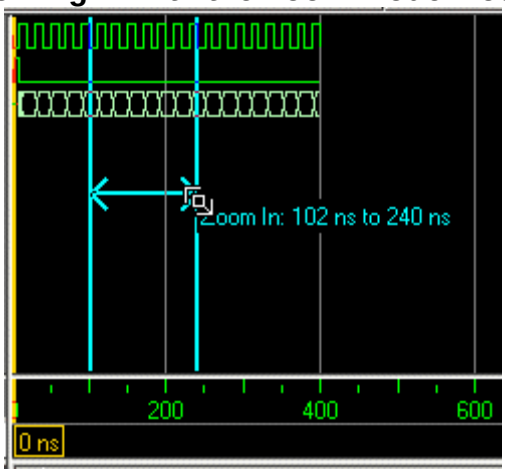
Wave 윈도우의 파형을 줌-인/줌-아웃 하는 여러 가지 방법에 대해 알아보겠습니다.

1. zoom 하는 여러 가지 방법 알아보기

A. wave 윈도우에서 zoom 아이콘을 클릭합니다. 

B. wave 윈도우의 파형 부분에서 마우스를 클릭하고 오른쪽 아래방향으로 드래그 합니다. 그러면 아래의 이미지처럼 확대가 됩니다.

Figure 7-2. Zooming in with the Zoom Mode Mouse Pointer



C. View>Zoom>Zoom Last 를 클릭합니다.

그러면 화면은 이전의 zoom 전의 화면으로 전환 됩니다.

D. Zoom In 아이콘을 클릭합니다. 

E. View>Zoom>Zoom Full 을 클릭합니다.

Using Cursors in the Wave Window

유저 분은 Wave window 에서 Cursor 를 이용하여 mark 할 수 있습니다. Wave Window 를 처음 열면 Cursor 는 기본적으로 time 0 에 위치하고 있습니다. 그 뒤에 Wave Window 에서 마우스 클릭하면 해당 커서가 클릭한 위치로 이동을 하게 됩니다.

유저 분은 또한,

Cursor 를 여러 개 추가할 수 있습니다.

Cursor 가 움직이지 않게 Lock 을 하거나 지울 수도 있습니다.

Cursor 간의 interval 간격을 볼 수 있습니다.

Cursor 를 이용하여 Wave 윈도우의 다음 Transition 을 찾을 수 있습니다.

Working with a Single Cursor

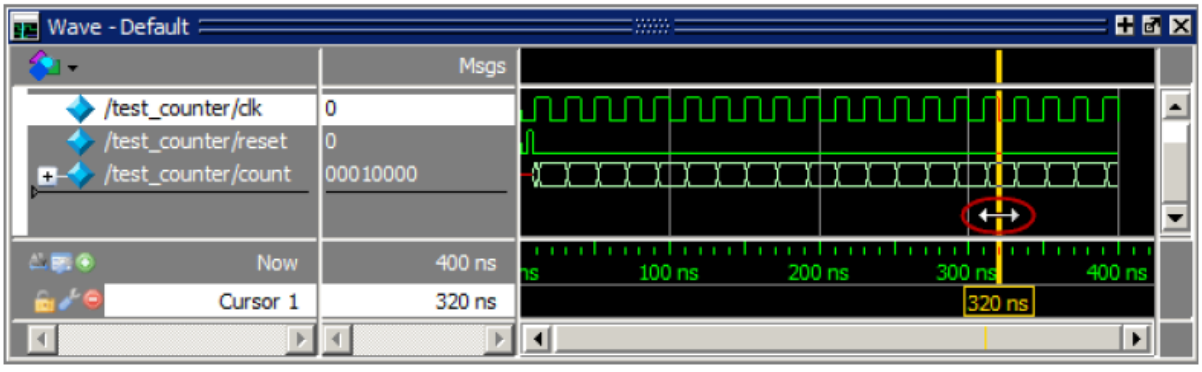
1. 클릭과 드래그를 통해 Cursor 이동하기

   A. Wave Window 의 toolbar 에서 Select Mode icon 을 클릭합니다. 

   B. Wave Window 의 파형이 그려지는 부분 아무 곳이나 클릭합니다.

      클릭한 위치에 Cursor 가 나타납니다.

Figure 7-3. Working with a Single Cursor in the Wave Window



   C. Cursor 드래그 하면서 Value 확인하기

      Cursor 가 움직이면 Value 창에서 값이 변경되는 것을 볼 수 있습니다.

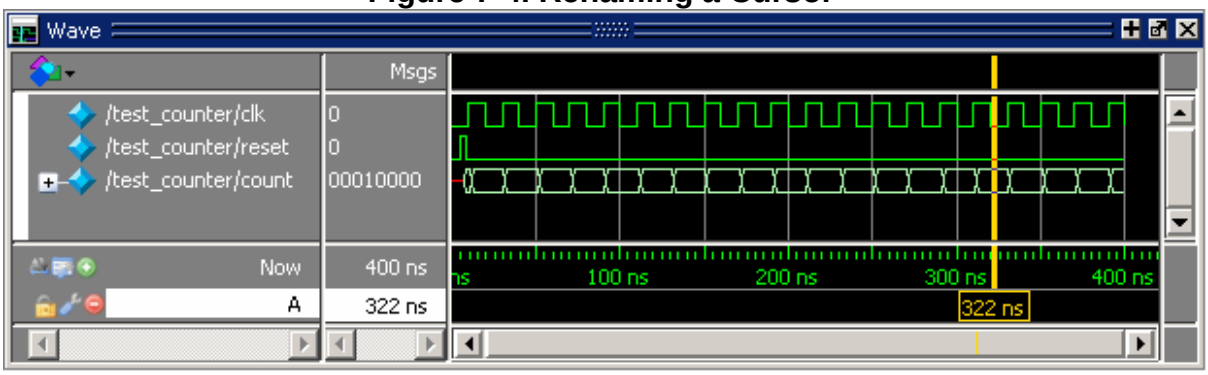
2. Cursor 의 이름 바꾸기

A. Wave 윈도우의 Cursor 창에서 "Cursor 1" 마우스 오른쪽 버튼을 클릭합니다.

B. Text 를 delete 키를 이용하여 삭제하고 A 를 입력합니다.

Cursor 의 이름이 A 로 변경된 것을 확인 할 수 있습니다.

Figure 7-4. Renaming a Cursor



3. Cursor 점프 이동 시키기

A. Pathname 창에서 counter 를 클릭합니다.

B. Wave 윈도우의 메뉴 중에서 아래의 아이콘을 클릭합니다. 

선택한 Signal 의 next transition 으로 Cursor 가 점프하여 이동합니다.

C. Wave 윈도우의 메뉴 중에서 아래의 아이콘을 클릭합니다. 

이전의 transition 으로 Cursor 가 점프하여 이동합니다.

Working with Multiple Cursors

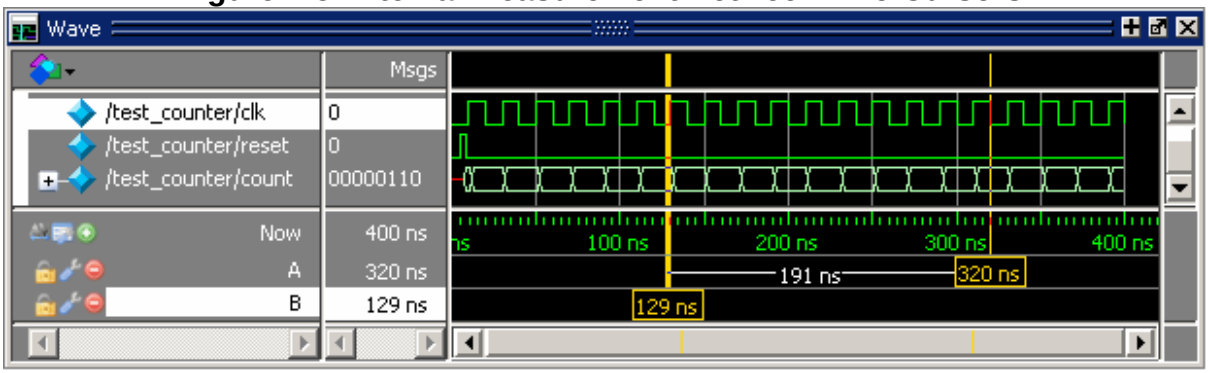
1. Cursor 추가하기

A. Cursor 를 추가하기 위해 아이콘을 클릭합니다. 

B. 마우스 오른쪽 버튼을 이용하여 새로 추가된 Cursor 의 이름을 B 로 수정합니다.

C. Cursor B 를 이동하면 자동적으로 A 와 거리가 측정됩니다.

Figure 7-5. Interval Measurement Between Two Cursors



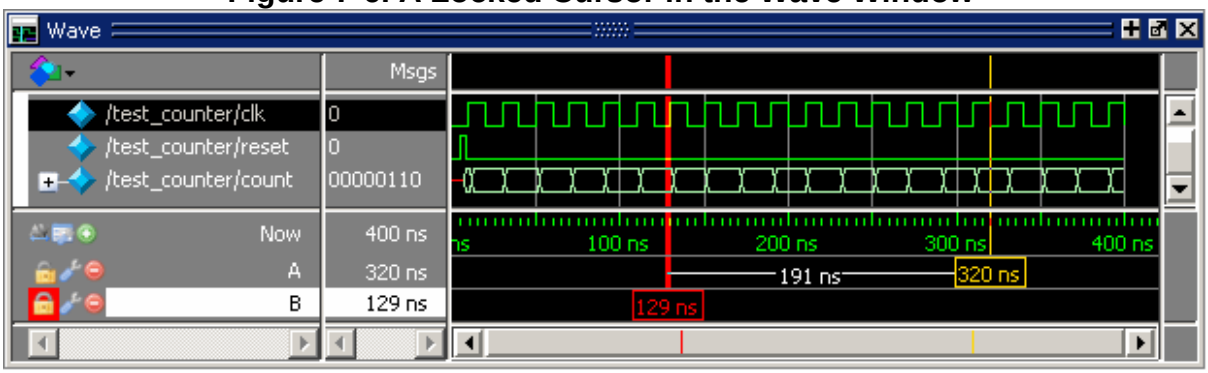
2. Cursor B 고정하기

   A. Cursor B 를 56ns 근처로 이동시킵니다.

   B. 마우스 오른쪽 버튼을 클릭하여 나온 Pop up 메뉴에서 Lock B 를 클릭합니다.

   아래의 그림처럼 Cursor 가 빨간색으로 변경되며 움직여지지 않습니다.

Figure 7-6. A Locked Cursor in the Wave Window



3. Cursor B 삭제하기

   A. Cursor B 에서 마우스 오른쪽 버튼을 클릭하여 Delete B 를 선택합니다.

Saving and Reusing the Window Format

유저 분이 Wave 윈도우를 닫아버리시면 지금까지 Wave 윈도우에서 했던 작업 (Signal 추가, Cursor 세팅) 같은 작업을 Wave 윈도우를 열 때 마다 다시 작업을 해야 합니다. 그러나, Save format command 를 이용하여 do file 로 format 을 저장하면 이러한 불편 없이 해당 format 을 이용하는 것이 가능합니다.

1. Format file 저장하기

   A. Wave 윈도우에서 File>Save format 을 선택합니다.

B. 파일 이름을 Wave.do 파일로 수정하여 저장합니다.

C. Wave 윈도우를 종료합니다.

2. Format 파일 불러오기

A. View>Wave 를 선택합니다.

B. Wave 윈도우를 Undock 한 후에 Wave 윈도우에서 File>Load 를 선택합니다.

C. 위에서 저장했던 wave.do 파일을 불러옵니다.

D. 저장했을 당시의 Signal 들이 자동으로 추가됩니다.

E. Wave 윈도우를 종료합니다.

F. Simulate>End Simulation 을 선택하여 현재 진행중인 Simulation 을 종료합니다.

Chapter 8
Creating Stimulus With Waveform Editor

Wave 윈도우에서 waveform Editor 를 이용하여 Simulation Stimulus 를 만들어서 Simulation 을 진행할 수 있습니다. 원하는 Simulation time 만큼 Stimulus 를 만들어서 Simulation 을 진행하고 해당 Stimulus 를 저장하여 추후에 사용 할 수도 있습니다.

이번 챕터에서는 이전에 사용했던 간단한 8 bit counter 응 이용하여 Simulation 을 진행할 것입니다. 예제 파일의 위치는 다음과 같습니다.

Verilog – *<install_dir>/examples/tutorials/verilog/basicSimulation*

VHDL – *<install_dir>/examples/tutorials/vhdl/basicSimulation*

이번 챕터에서 진행되는 Waveform Editor 는 유저 매뉴얼의 Generating Stimulus with Waveform Editor 나 Wave Window 를 보시면 좀 더 자세히 알 수 있습니다.

Compile and Load the Design

1. 새로운 디렉토리를 만들고 tutorial file 복사하기

   이번 챕터를 진행하기 위한 directory 를 만듭니다.

   그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. ModelSim/Questa 를 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 ModelSim/Questa 를 구동합니다.

   B. File>change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. Working library 만들고 compile 하기

   A. vlib work 를 입력합니다.

   B. 해당 파일들을 compile 합니다.

      Verilog

      vlog counter.v

VHDL

vcom counter.vhd

4. Design Load 하기

   A. vsim –novopt counter 를 입력합니다.

5. Wave 윈도우 열기

   A. View>Wave 를 선택합니다.

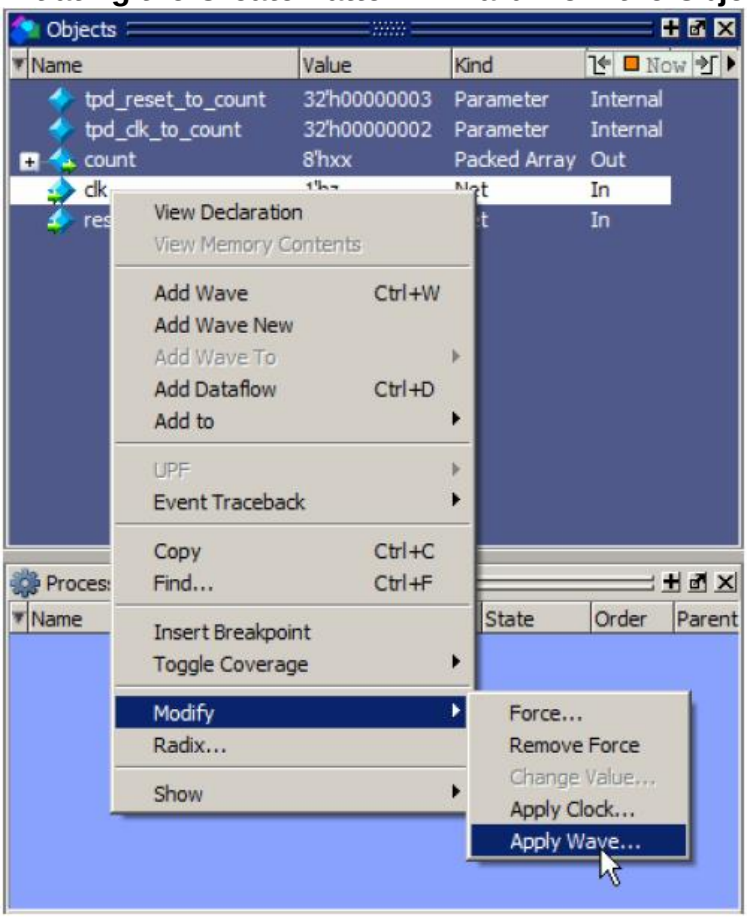
Create Graphical Stimulus with a Wizard

Waveform Editor 에는 Patten Wizard 가 포함되어 있습니다.

1. Patten Wizard 를 이용하여 Clock 만들기

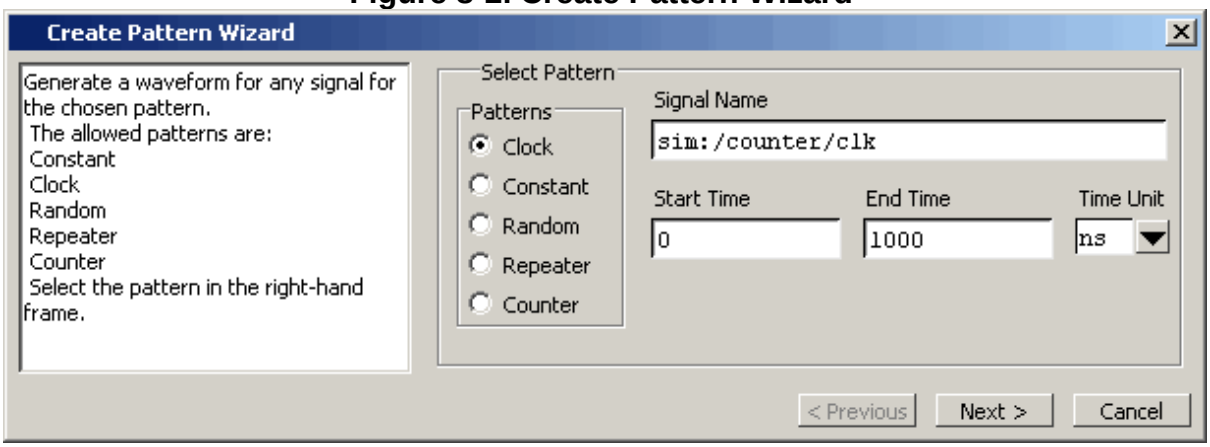
   A. Object 윈도우에서 clk 를 선택하고 마우스 오른쪽 버튼을 클릭하여 Create Wave 를 선택합니다.

Figure 8-1. Initiating the Create Pattern Wizard from the Objects Window



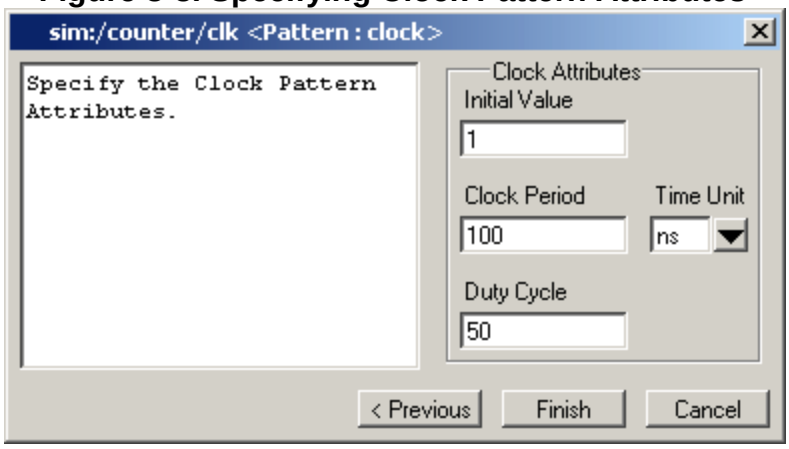
B. Patten Wizard 창이 열리면 Clock 를 선택하고 아래의 그림과 같이 설정합니다.

Figure 8-2. Create Pattern Wizard



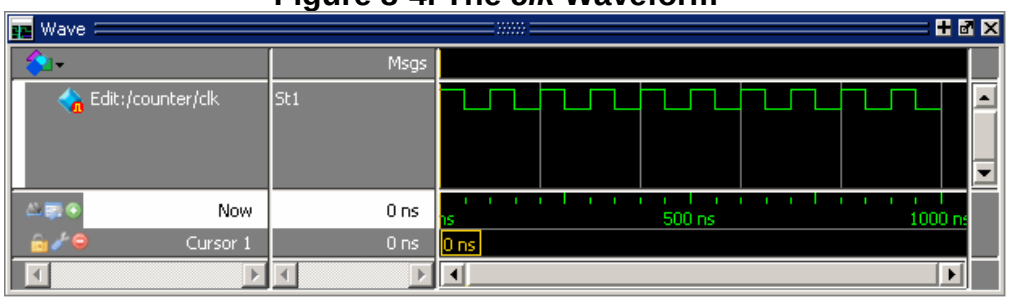
C. 두 번째 창이 열리면 다시 한번 아래와 같이 설정합니다.

Figure 8-3. Specifying Clock Pattern Attributes



위와 같이 설정을 하고 Finish 버튼을 클릭하면 아래와 같이 Clock Patten 이 만들어 집니다.

Figure 8-4. The *clk* Waveform



2. Wizard 를 이용하여 또 다른 Signal 만들기

A. Object 윈도우에서 reset 를 선택하고 마우스 오른쪽 버튼을 클릭하고 Create Wave

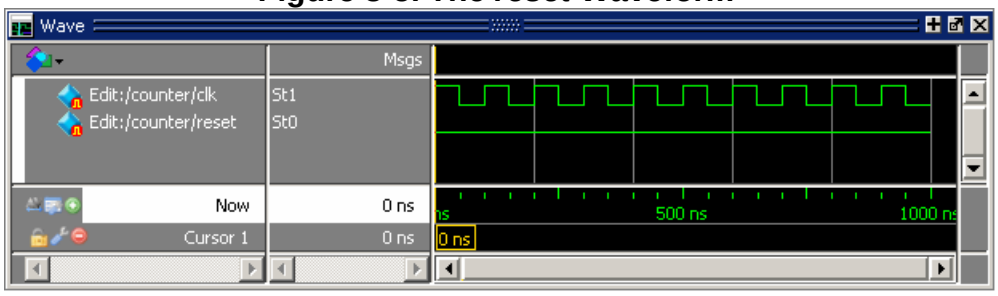
를 선택합니다.

B. Patten 타입을 Constant 를 선택하시고 Next 를 선택합니다.

C. Value 에 0 를 입력하고 Finish 를 클릭합니다.

두 번째로 만들어진 파형이 Wave 윈도우에 보여집니다.

Figure 8-5. The *reset* Waveform



Edit Waveforms in the Wave Window

Waveform Editor 는 Patten Wizard 뿐만 아니라 editing 할 수 있는 여러 Command 를 제공하고 있습니다. (invert, mirror, stretch edge, cut, etc.)

1. reset 에 Pulse 삽입하기

   A. Edit Mode 를 활성화하기 위해 아래의 아이콘을 클릭합니다. 

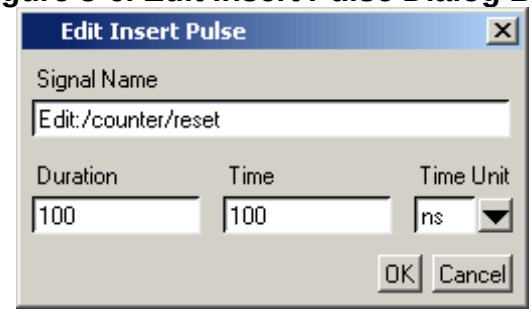
   B. Pathnames 에서 reset 를 선택합니다.

   C. Insert Pulse 아이콘을 클릭합니다. 

   혹은, Wave 윈도우에서 reset 를 선택하고 마우스 오른쪽을 클릭하여 Wave>Wave Editor>Insert Pulse 를 클릭합니다.

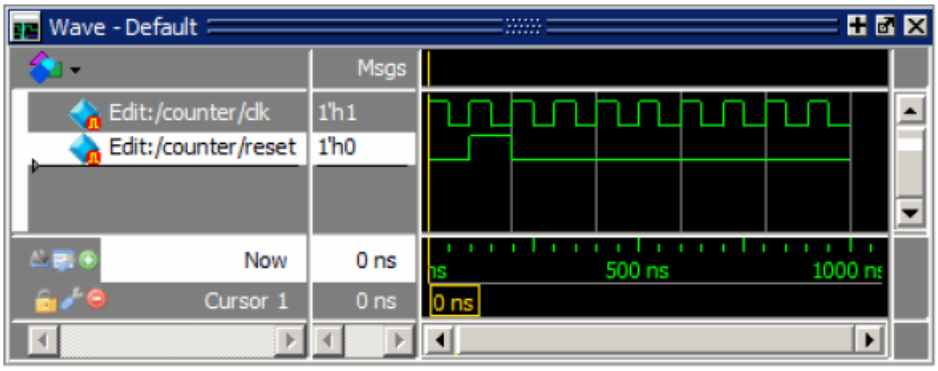
   D. Insert Pulse 를 선택하고 아래의 창과 같이 입력합니다.

Figure 8-6. Edit Insert Pulse Dialog Box



OK 를 클릭하면 reset 의 100ns 에서 200ns 사이에 High 신호가 나타납니다.

Figure 8-7. Signal *reset* with an Inserted Pulse

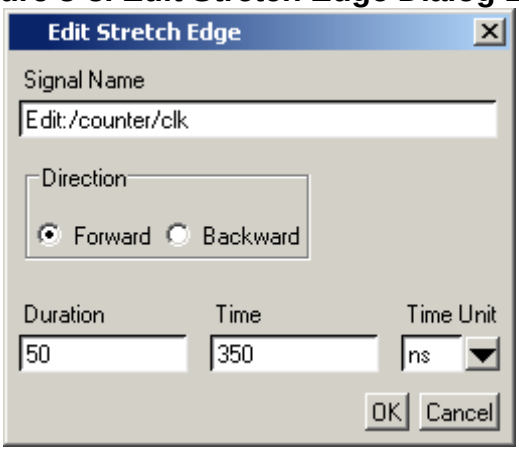


2. clk 의 edge 늘리기

   A. Wave 윈도우에서 clk 를 클릭하고 마우스 오른쪽 버튼을 클릭하여 Wave Editor>Stretch Edge 를 클릭합니다.

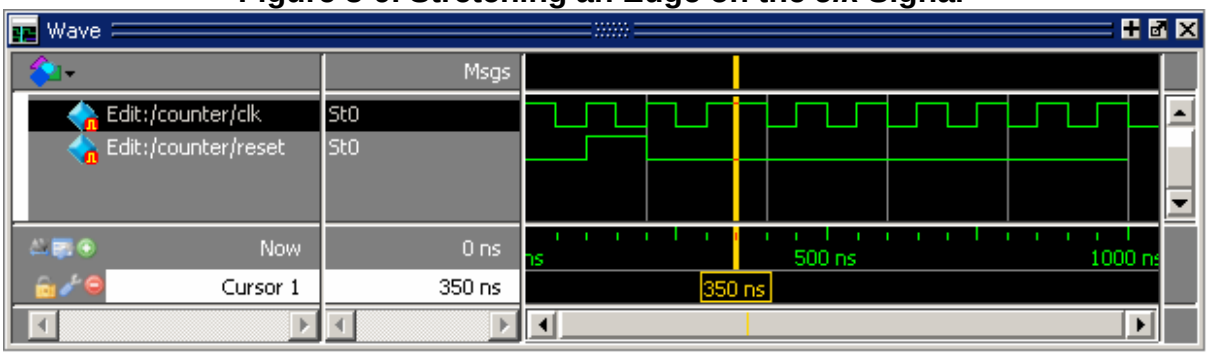
   B. 창이 열리면 Duration 에 50, Time 에 350 를 입력하고 OK 를 클릭합니다.

Figure 8-8. Edit Stretch Edge Dialog Box



OK 를 클릭하여 300~400ns 사이에 high 가 유지되는 파형으로 변경됩니다.

Figure 8-9. Stretching an Edge on the *clk* Signal



3. Edge 지우기

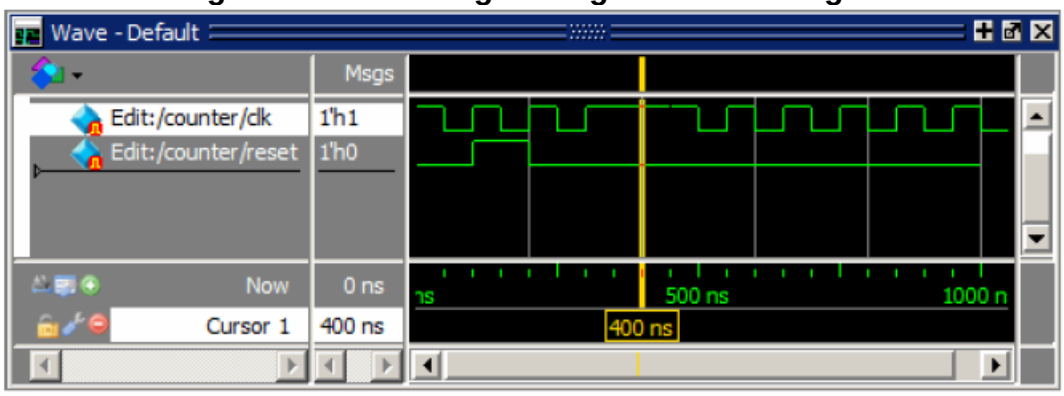
A. Cursor 를 400ns 근처에 놓고 clk 를 클릭합니다.

B. Delete Edge 아이콘을 클릭합니다. 

아이콘을 클릭하면 창이 열립니다.

창이 열리면 Time 에는 400 ns 를 입력하고 OK 버튼을 클릭합니다.

Figure 8-10. Deleting an Edge on the *clk* Signal



clk 는 400~500ns 사이에 있던 edge 가 사라지고 위와 같이 파형이 나타납니다.

4. Undo 와 Redo 를 이용하여 edit 하기

A. Undo 아이콘을 클릭합니다. 

Undo 아이콘을 클릭하여 Undo Count 를 물어보는 창이 나타납니다. 1 를 입력하고 OK 를 클릭하면 Delete Edge 하기전의 파형으로 돌아갑니다.

B. clk 을 다시 선택하고 Redo 아이콘을 클릭합니다. 

Redo 아이콘을 클릭하면 마찬가지고 Redo Count 를 물어보는 창이 나타나고, 1 를 입력하면 Delete Edge 가 진행된 형태로 파형이 나타납니다.

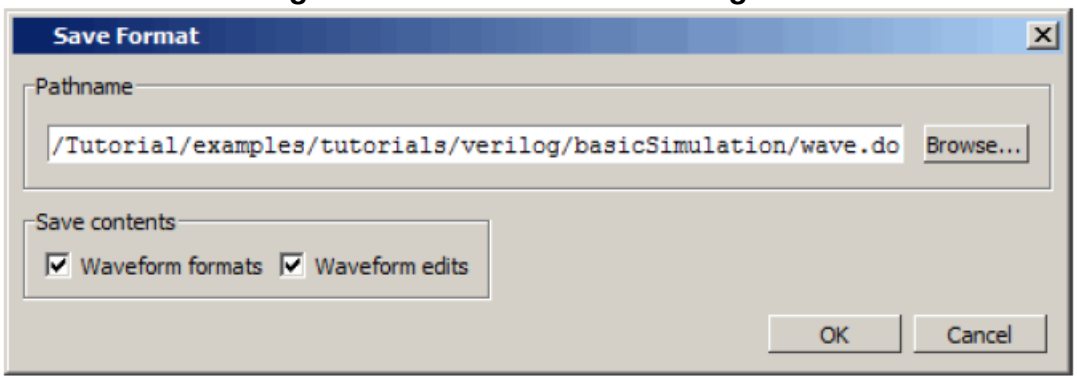
Save and Reuse the Wave Commands

ModelSim/Questa 에서 작업한 Waveform Stimulus 를 특정한 Format 의 파일로 저장할 수 있습니다. 또한 Load 를 해서 반복적으로 사용할 수도 있습니다.

1. Format File 저장하기

A. Wave command 를 저장하기 위해 File>Save Format 를 선택합니다.

Figure 8-11. Save Format Dialog Box



기본적으로 File 이름은 wave.do 로 저장됩니다.

B. Yes 를 클릭합니다.

C. File name 필드에 waveedit.do 를 입력하고 Save 를 클릭합니다.

이제 Waveform 은 waveedit.do 파일로 저장이 됩니다.

2. Simulation 종료하고 다시 Design 로드하기

A. Simulate>End Simulation 을 선택하여 Simulation 을 종료합니다.

B. 다시 Design 을 로드하기 위해 아래의 Command 를 입력합니다.

vsim –novopt counter

3. Format File 열기

A. View>Wave 를 클릭하여 Wave 윈도우를 엽니다.

B. File>Load 를 선택하고 앞에서 저장했던 waveedit.do 파일을 클릭합니다.

이제 앞서서 저장했던 Wave 파형이 Wave 윈도우에 나타난 것을 확인 할 수 있습니다.

Exporting the Created Waveforms

앞에서는 do 파일 format 을 저장하는 방법을 살펴보았으나 여기서는 4개의 파일 Format 중에 원하는 형태로 저장하는 방법을 살펴보겠습니다.

1. Created Waveform 을 HDL Test bench 파일 Format 으로 저장하기

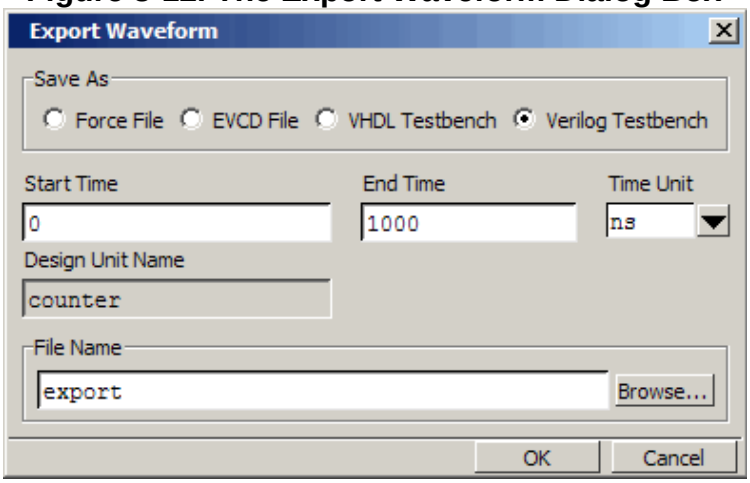
A. File>Export>Waveform 을 선택합니다.

B. Save as 부분을 VHDL Test bench 혹은 Verilog Test bench 를 선택합니다.

C. End Time 부분은 1000 을 입력합니다.

D. File Name 에는 export 를 입력하고 OK 버튼을 클릭합니다.

Figure 8-12. The Export Waveform Dialog Box



ModelSim/Questa 는 현재 디렉토리에 export.vhd 혹은 export.v 라는 이름으로 test bench 파일을 저장합니다.

2. VCD Format 을 저장하기

A. File>Export>Waveform 를 선택합니다.

B. EVCD File 을 선택합니다.

C. End Time 에 1000 을 입력하고 OK 버튼을 클릭합니다.

ModelSim/Questa 은 현재 디렉토리에 export.vcd 파일을 저장합니다.

Run the Simulation

Created Waveform 을 시작하여 원하는 Signal 로의 Edit 작업이 끝났으면, 이제 Simulation 을 진행하겠습니다.

1. Design Signal 추가하기

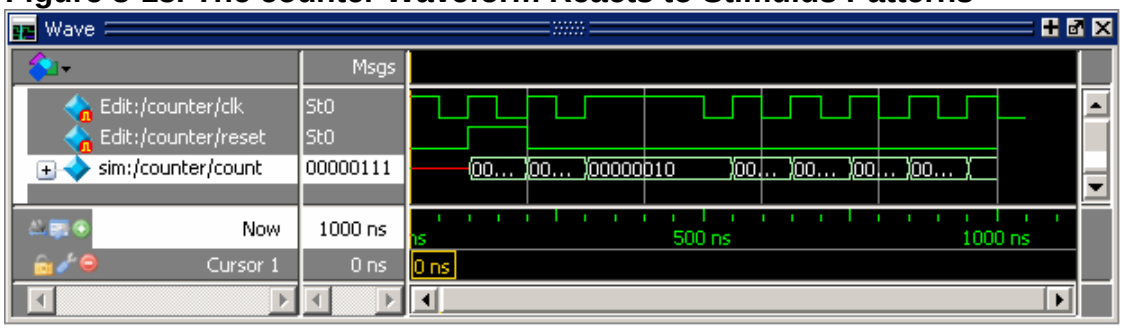
A. Object 윈도우에서 count 를 선택하고 마우스 오른쪽 버튼을 클릭하여 Add>To Wave>Selected Signals 를 선택합니다.

2. Simulation 을 동작하기

A. ModelSim/Questa 의 Transcript 윈도우에 run 1000 을 입력합니다.

Simulation 은 1000 ns 까지 동작하고 그 파형은 아래와 같습니다.

Figure 8-13. The counter Waveform Reacts to Stimulus Patterns



count 를 살펴보면 300 ~ 500 ns 사이엔 clk 이 동작하지 않았기에 count 가 늘어나지 않는 것을 볼 수 있습니다.

3. Simulation 종료하기

A. Simulate>End Simulation 을 선택하여 Simulation 을 종료합니다.

Simulating with the Test Bench File

이제부터는 앞에서 저장을 했던 Test bench Format 파일을 가지고 Simulation 을 진행해보도록 하겠습니다.

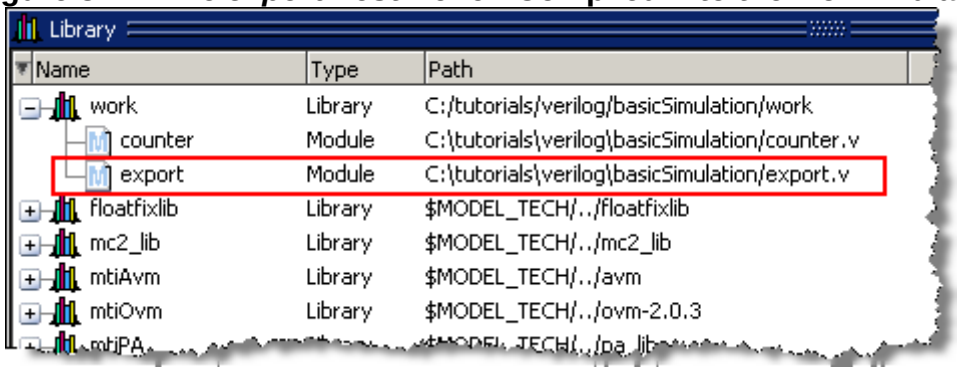
1. Test bench 파일 Compile 하고 Load 하기

A. 앞에서 저장했던 Test bench File 을 Compile 합니다.

Verilog 의 경우 vlog export.v VHDL 의 경우 export.vhd 를 입력해 주시면 Compile 됩니다.

Compile 이 완료되면 Work Library 에 export 모듈의 Library 가 보여집니다.

Figure 8-14. The *export* Test Bench Compiled into the work Library



B. Simulation 을 하기 위해 Transcript 윈도우에 아래의 command 를 입력합니다.

vsim –voptargs="+acc" export

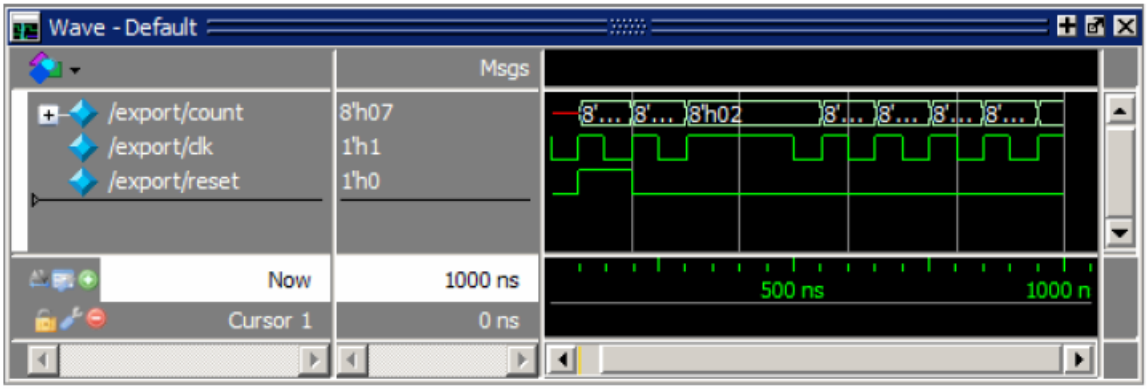
2. Signal 추가하고 Simulation 진행하기

A. add wave * 를 입력합니다.

B. run 1000 을 입력합니다.

Simulation 결과가 아래의 그림과 같이 보여집니다.

Figure 8-15. Waves from Newly Created Test Bench



3. Simulation 종료하기

A. Simulation 을 종료하기 위해 Transcript 윈도우에 quit –sim 이라고 입력합니다.

Importing an EVCD File

이제부터는 앞에서 저장을 했던 EVCD Format 파일을 가지고 Simulation 을 진행해보도록 하겠습니다.

1. Counter Design 로드하고 Signal 추가하기

A. Transcript 윈도우에 아래의 command 를 입력합니다.

vsim –voptargs="+acc+ counter

B. Object 윈도우에서 count 를 선택하고 마우스 오른쪽 버튼을 클릭하여 Add>To Wave>Selected Signals 를 선택합니다.

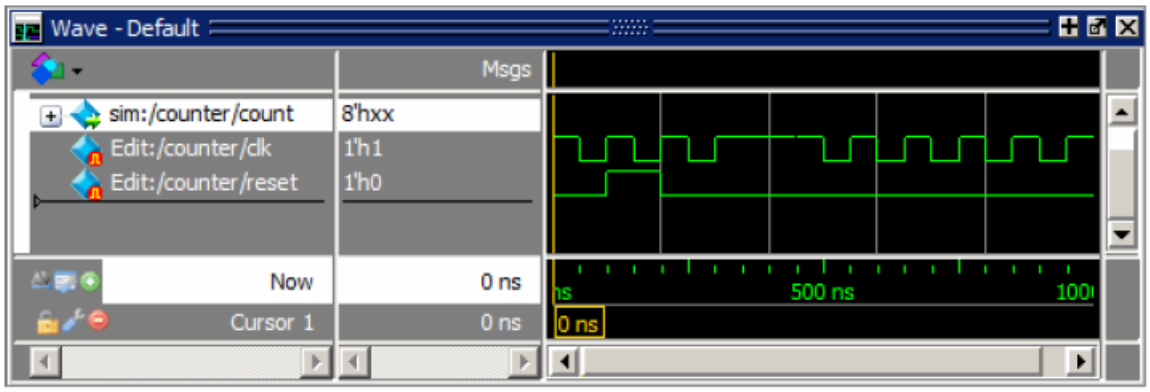
2. VCD File 불러오기

A. Wave 윈도우에서 File>Import EVCD 를 선택합니다.

B. export.vcd 파일을 더블 클릭합니다.

Created wave 작업을 한 파형이 아래와 같이 보여집니다.

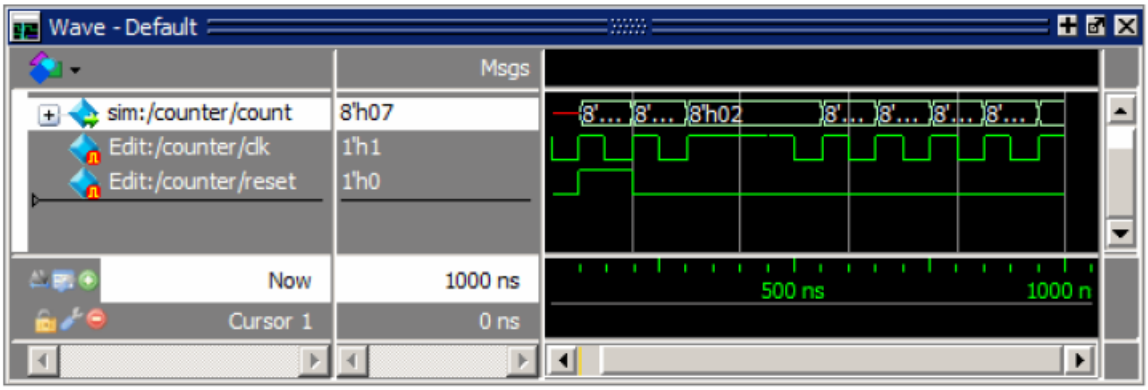
Figure 8-16. EVCD File Loaded in Wave Window



C. Run-All 아이콘을 클릭합니다. 

아이콘을 클릭하면 evcd 파일에서 만들었던 1000 ns 까지 Simulation 이 진행됩니다.

Figure 8-17. Simulation results with EVCD File



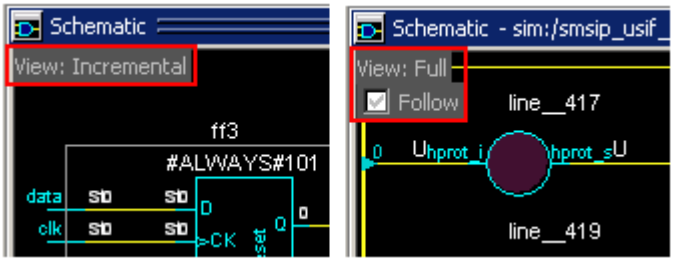
Chapter 9
Debugging With The Schematic Window

Questa 의 Schematic 윈도우에서는 Design 에서의 예상치 못한 Output 의 원인을 추적하여 식별 할 수 있는 물리적인 연결을 살펴볼 수 있는 윈도우 입니다. 이 윈도우에서는 VHDL Architectures 와 Verilog Modules 의 process, signal, nets, registers 를 살펴 볼 수 있습니다.

Schematic 윈도우는 2가지의 Design View (Full View, Incremental View) 를 제공하고 있습니다. Full View 는 Design 의 전체 Hierarchy 구조를 볼 수 있습니다. Incremental View 는 Design 의 구조를 점진적으로 살펴 볼 수 있습니다. Incremental View 에서는 디자인의 의도를 쉽게 이해 할 수 있게 하고, 디자인의 RTL 부분을 논리 게이트 형태로 표시합니다.

Schematic 윈도우의 왼쪽 상단 부분을 클릭하여 View 모드를 변경할 수 있습니다.

Figure 9-1. Schematic View Indicator



incremental View 은 Design Debugging 에 이상적입니다. Incremental View 를 이용하면 언제 어디서 Signal 의 값이 변경되었는지 Design 의 내부를 관찰할 수 있습니다.

이번 챕터에서는 Cache Module 과 Primary Memory 를 검증하는 Design 을 예제로 사용합니다. 예제 파일에 대한 경로는 다음과 같습니다

Verilog – *<install_dir>/examples/tutorials/verilog/schematic*

VHDL – *<install_dir>/examples/tutorials/vhdl/schematic*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Schematic Window 를 보시면 좀 더 자세히 알 수 있습니다.

Design Files for this Lesson

이번 챕터에서는 Compile 과 Design Load 를 Do 파일을 가지고 진행할 것입니다.

1. 새로운 디렉토리를 만들고 tutorial file 복사하기

이번 챕터를 진행하기 위한 directory 를 만듭니다.

그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. ModelSim/Questa 을 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 Questa 을 구동합니다.

   B. File>change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. WildcardFilter 설정 변경하기

   A. Transcript 윈도우에 아래와 같이 command 를 입력합니다.

      set WildcardFilter "Variable Constant Generic Parameter SpecParam Memory Assertion Endpoint ImmediateAssert"

      위의 command 를 입력하면 앞으로 "CellInternal" 이 제거되어 debugging 하는데 있어서 모든 signal 을 볼 수 있도록 변경이 됩니다.

4. Do 파일 실행하기

   A. do run.do 을 입력합니다.

      run.do 파일은 아래의 작업들을 진행하게 됩니다.

      - Work library 만들기 - vlib work
      - Design File Compile 하기 - vlog or vcom
      - Design Optimized 하기 - vopt +acc top -debugdb -o top_opt
      - Simulator 에 Design 로드하기 - vsim -debugdb top_opt
      - Wave 윈도우에 Signal 추가하기 - add wave /top/p/*
      - Design 상의 모든 Signal 을 log 에 저장하기 - log -r /*
      - Simulation 진행하기 – run –all

   B. Radix 를 Symbolic 으로 변경하기
      - Transcript 윈도우에 radix –symbolic 을 입력합니다.

Exploring Connectivity

여러분은 Schematic 윈도우를 통해서 Module 과 Architecture 의 특정 signal, net, register, process 를 관찰할 수 있습니다.

1. Schematic 윈도우 열기

   A. 메뉴에서 View>Schematic 를 선택하거나 Transcript 윈도우에 **view schematic** 를 입력합니다.

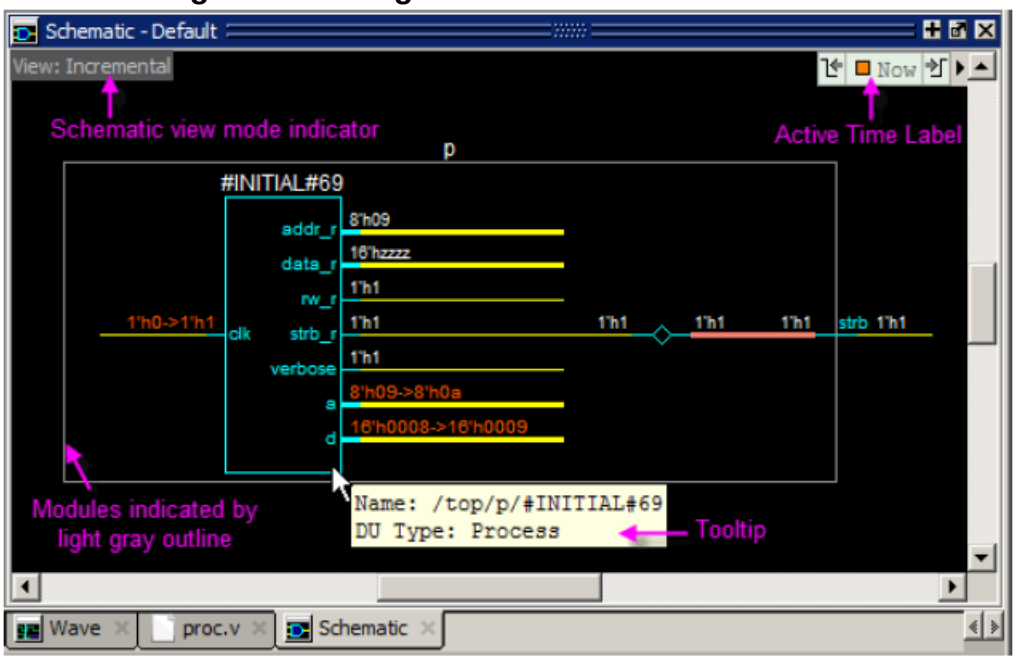
      Schematic 윈도우는 Incremental View 형태로 열립니다.

2. Schematic 윈도우에 Signal 추가하기

   A. Structure(sim) 탭에서 p instance 를 선택합니다

   B. Object 윈도우에서 strb 를 선택하여 Dataflow 윈도우로 드래그 합니다.

Figure 9-2. A Signal in the Schematic Window

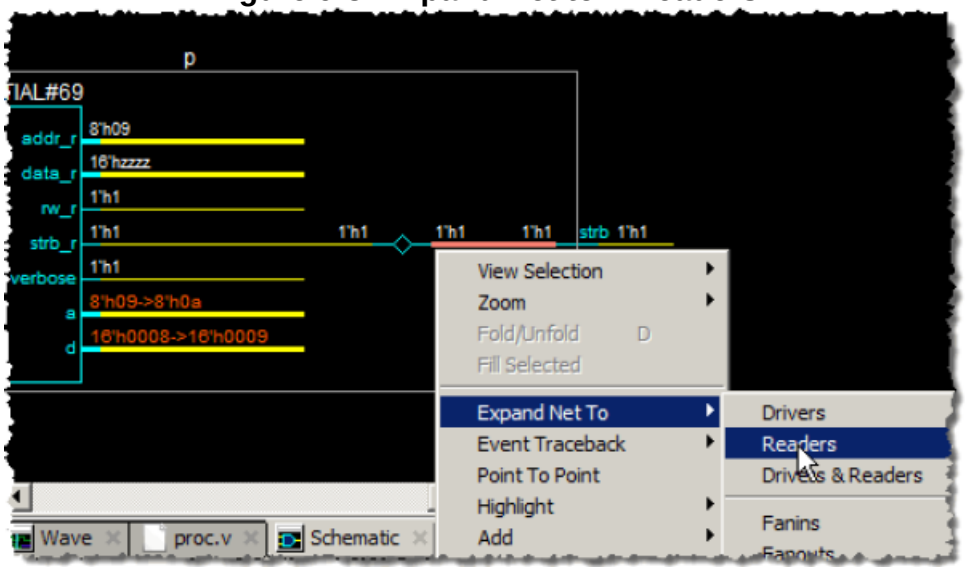


오렌지색으로 강조된 strb 신호를 보여줍니다. 위의 그림과 같이 마우스 커서를 올려 놓으면 tooltip 을 통해 간략한 정보가 보여집니다.

3. Module p 의 strb signal 을 추적하기

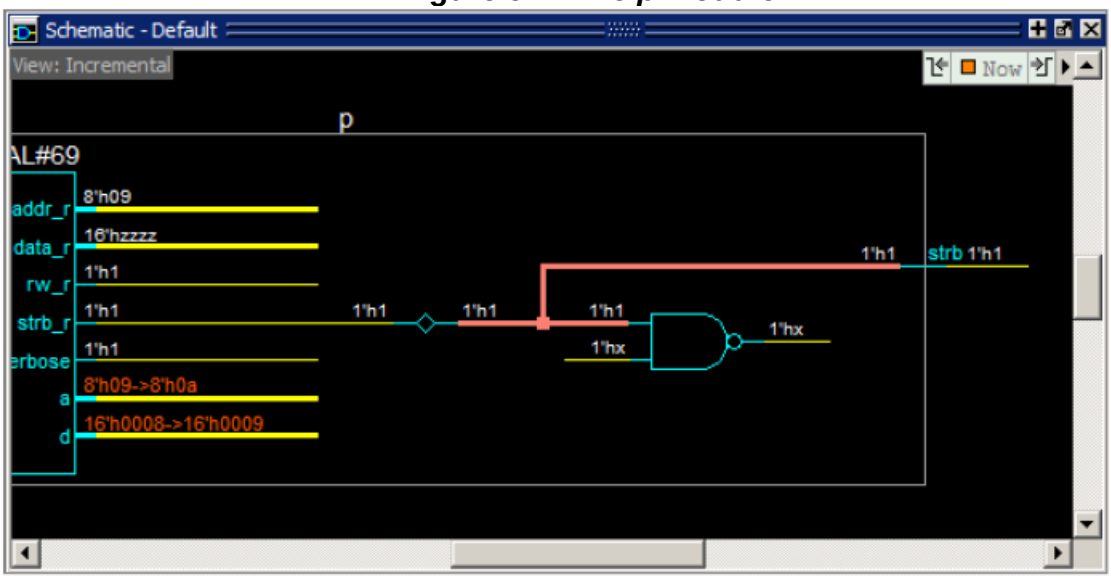
   A. strb signal 을 선택한 후 오른쪽 마우스를 클릭하여 Expand Net to > Readers 를 선택합니다.

Figure 9-3. Expand Net to > Readers



아래의 그림과 같이 확인할 수 있습니다.

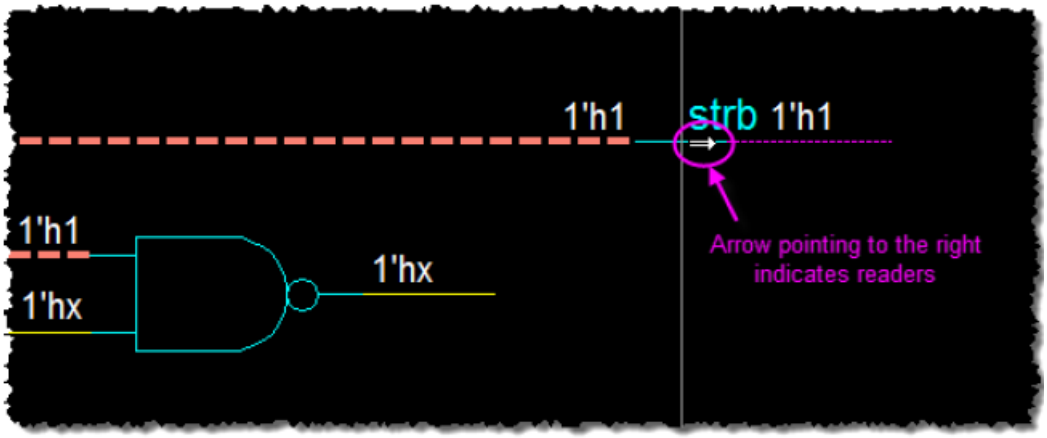
Figure 9-4. The *p* Module



위의 그림에서 보면 P Module 에서 나가는 strb 신호를 확인 볼 수 있습니다.
해당 신호에 마우스를 대면 화살표 아이콘이 나타나게 됩니다.

B. 아래의 그림과 같이 화살표가 나오면 더블클릭을 합니다..

Figure 9-5. Signal Values Overlapped



그러면 아래의 그림과 같이 strb 와 연결된 부분을 확인 할 수 있습니다.

Figure 9-6. Expanding the View to Display Readers of *strb* Signal

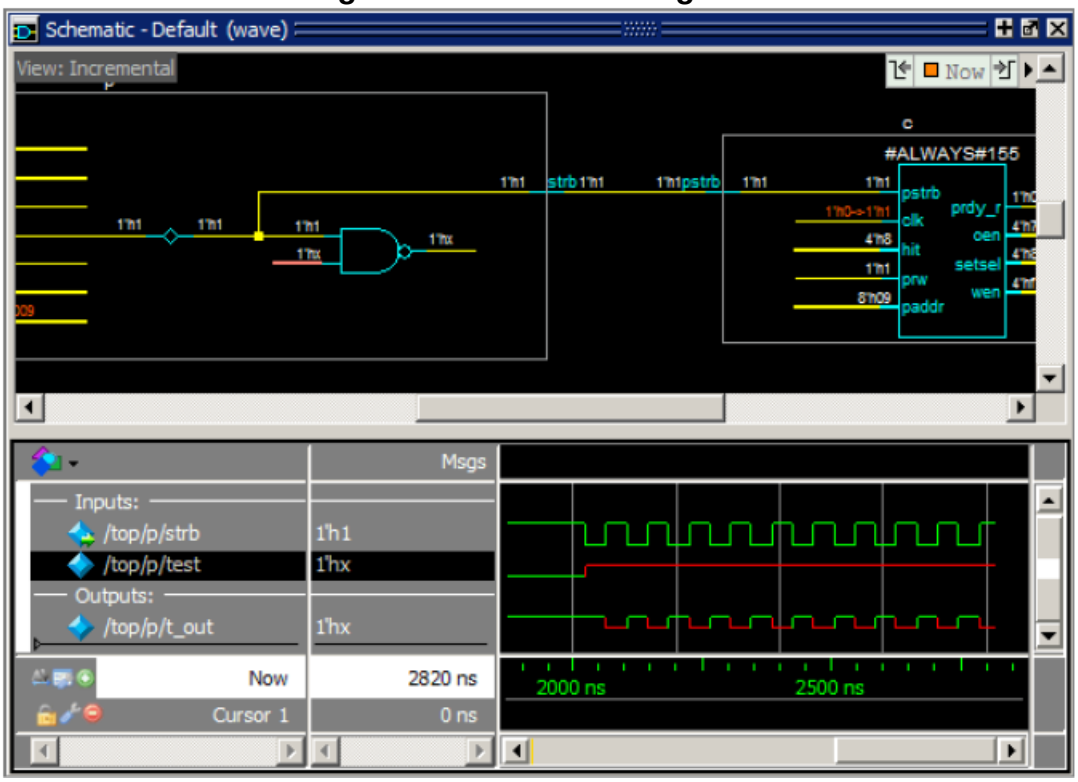
4. #NAND#50 process 에서의 test 추적하기

   A. Schematic 윈도우에서 embedded window 를 불러오기 위해 Show Wave 버튼  을 클릭합니다.

   B. Schematic 윈도우에서 #NAND#50 게이트를 선택하면 Wave Viewer 에 자동으로 Signal 들이 추가됩니다.

   C. Wave Viewer 에서 test 를 선택하면 자동적으로 Schematic 윈도우에 test 가 하이라이트 됩니다.

Figure 9-7. Select *test* signal

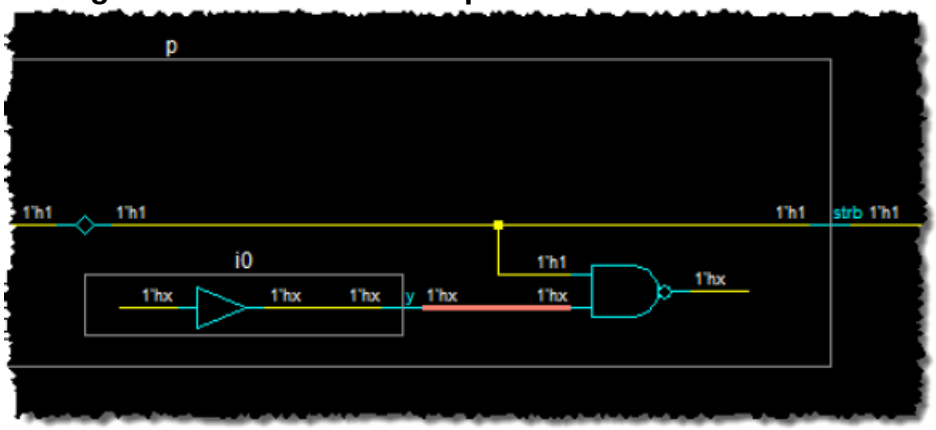


Schematic 윈도우의 Schematic 부분과 Wave 부분은 서로 연동되어 유기적으로 동작이 됩니다.

D. Schematic 윈도우에서 test 를 선택하면 Wave 부분에서 자동으로 선택 됩니다.

E. Expand net to all drivers 버튼  을 클릭하면, test signal 은 i0 NAND 게이트로 확장 되는 것을 알 수 있습니다.

Figure 9-8. The test Net Expanded to Show All Drivers

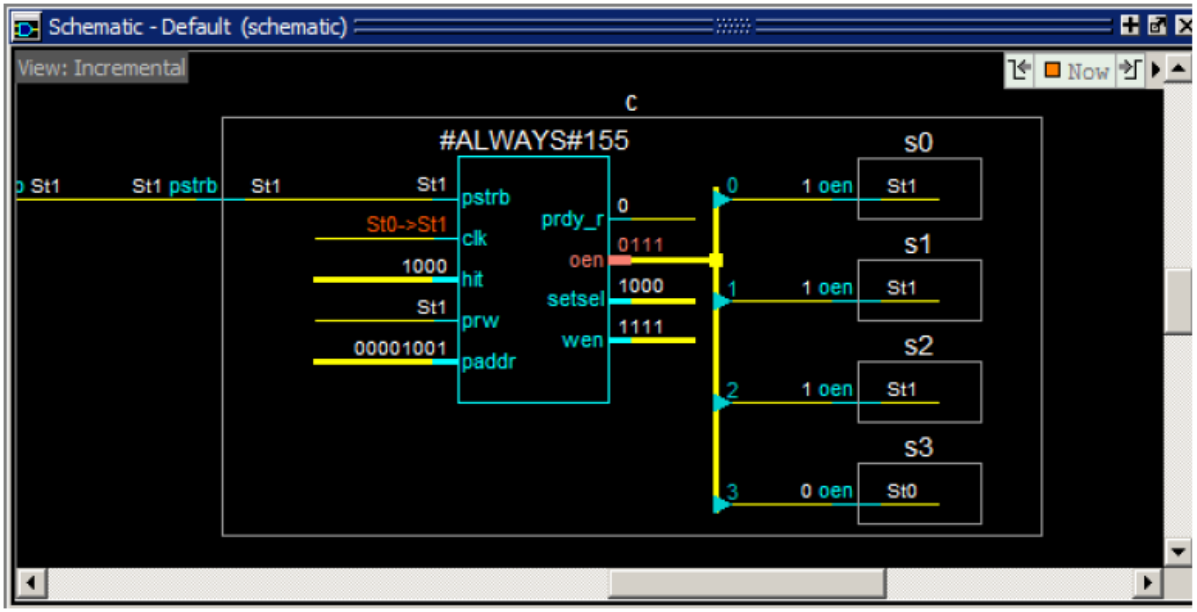


5. #ALWYAS#155 Process 의 oen signal 추적하기

   A. oen Signal 핀을 선택합니다.

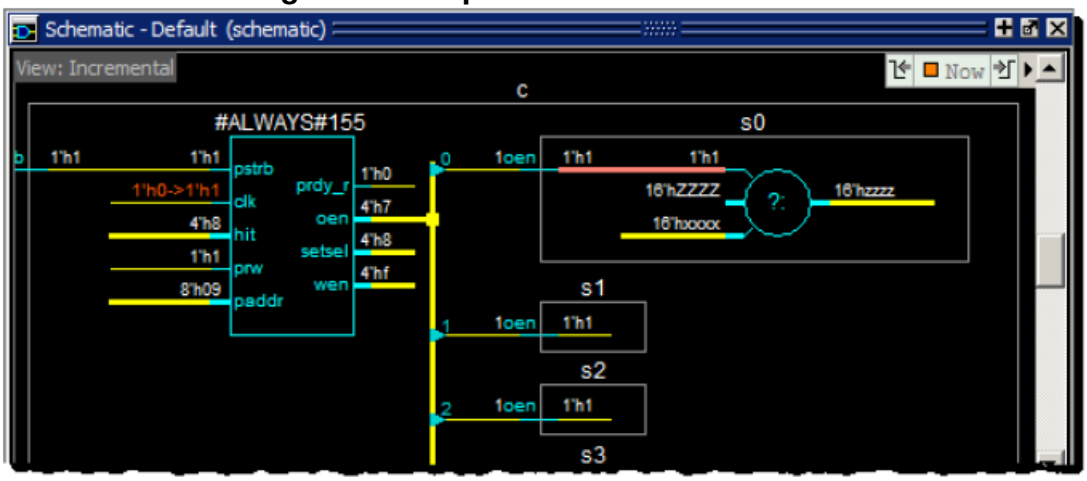
   B. 마우스 오른쪽 버튼을 클릭하여 팝업 메뉴에서 Expand Net to > Reader 를 선택하면 아래와 같이 확장된 결과가 보여집니다.

Figure 9-9. Signal *oen* Expanded to Readers



더욱 확장을 원하면 확장을 원하는 Signal 핀을 선택하고 더블 클릭을 하거나 마우스 오른쪽 버튼을 이용합니다.
s0 를 더블 클릭하면 내부를 확인 할 수 있습니다.

Figure 9-10. Sprout oen in the s0 Instance



아래의 그림을 보게 되면 신호 값이 Overlap 되어 쉽게 구별 되지 않습니다.

이런 경우 Regenerate button  을 클릭하면 정리되어 값이 잘 보여지게 됩니다.

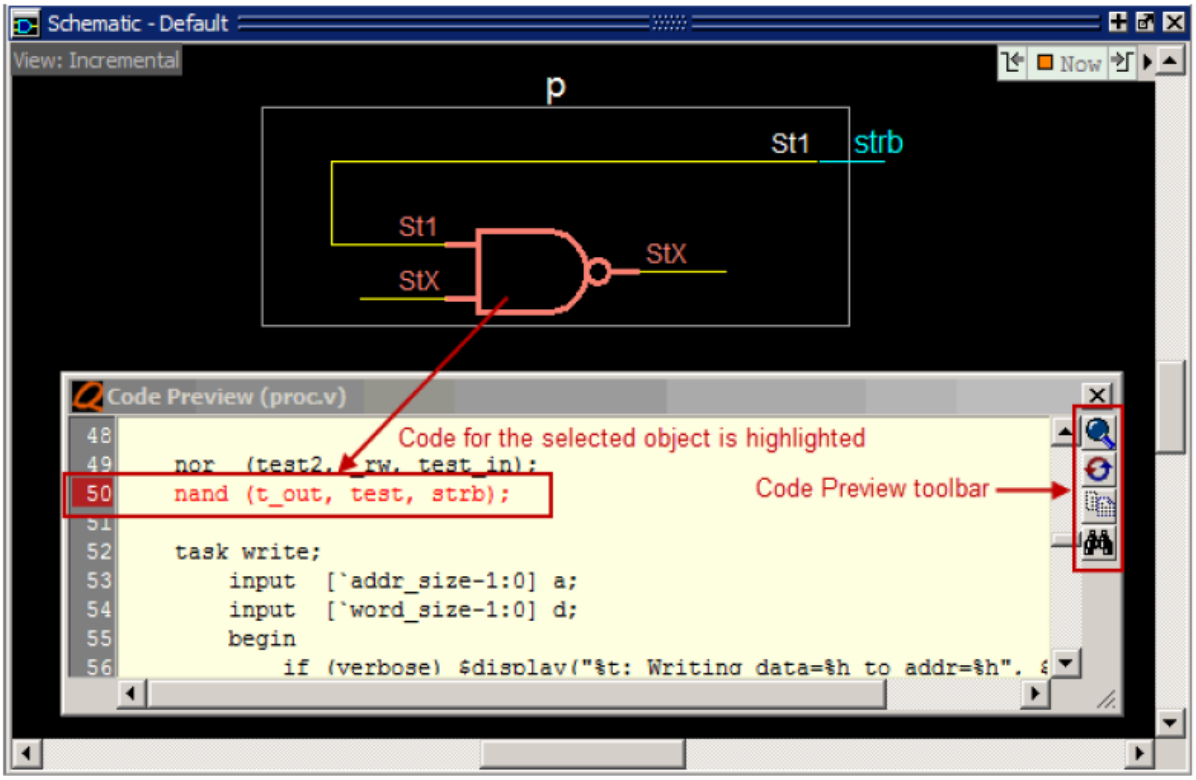
관측이 다 끝났으면, Delete All 버튼  을 클릭하면 Schematic 윈도우를 Clear 할 수 있습니다.

Viewing Source Code from the Schematic

Schematic 윈도우를 통해 디자인 객체의 소스 코드 미리 보기를 표시할 수 있습니다

1. Schematic 윈도우에 Signal 추가하기
   A. Structure (sim) 윈도우에서 p instance 를 선택합니다.
   B. Schematic 윈도우로 드래그 앤 드랍을 합니다.
   C. Schematic 윈도우에서 #NAND#50 을 선택하고 더블 클릭합니다.

Figure 9-13. Code Preview Window



Code Preview 창이 열리며 #NAND#50 에 해당하는 Source Code 가 하이라이트 처리 되어서 보여집니다.

1. Source editor 에 source code 가 보여집니다.

2. Code Preview toolbar 에 있는 search 버튼을 이용하여 특정한 code 를 검색해서 볼 수도 있습니다.

D. Code Preview toolbar 를 이용하여 다양한 작업을 할 수 있습니다.

원하는 작업을 전부 마친 후에는 **Delete All** 버튼을 클릭하면 Schematic Window 에서 진행했던 작업을 지울 수 있습니다.

Unfolding and Folding Instances

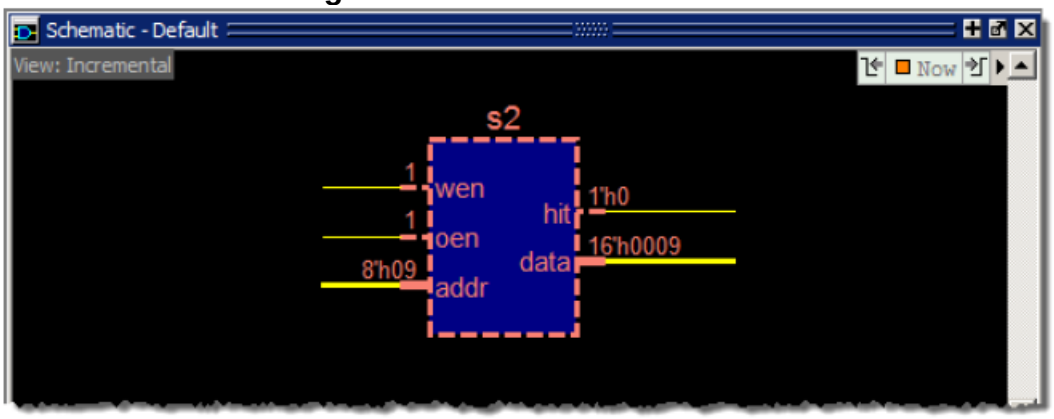
복잡한 Instance 구조를 가지고 있는 Design 인 경우 Incremental View 에서는 Schematic 윈도우의 사이즈에 맞추어 instance 가 folded 되어 보여지게 됩니다.

1. Incremental View 에서 Folded instance 보기

A. Structure 윈도우의 C module hierarchy 를 펼칩니다.

B. 펼쳐진 module 중에 s2 module 을 선택하여 Schematic 윈도우를 드래그 앤 드롭을 합니다.

Figure 9-14. Folded Instance



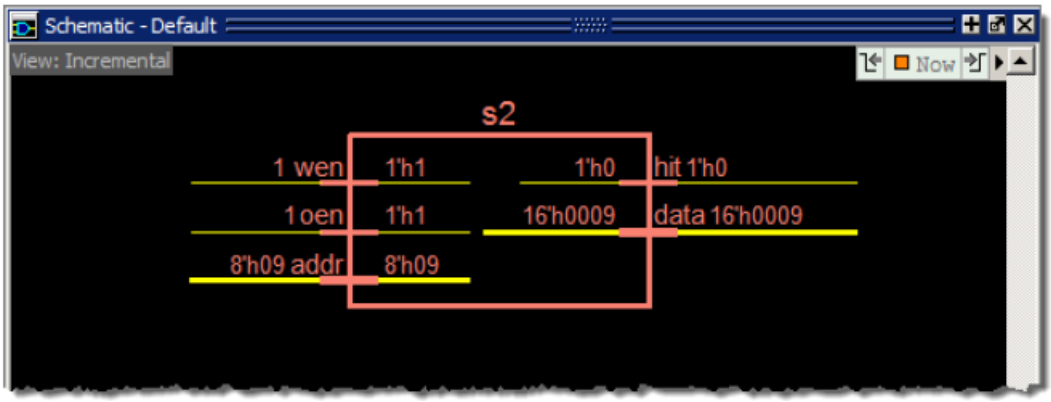
Fold 된 s2 module 이 빨간색 점선에 파란색 블록으로 보여집니다. 마우스를 s2 블록에 올려 놓으면 text top up 에 FOLDED 라는 메시지를 볼 수 있습니다.

2. Fold 된 Instance 펼치기

A. 위의 팝업 메뉴가 나온 상태에서 마우스 오른쪽을 클릭합니다.

B. 마우스 오른쪽 버튼을 클릭하면 나오는 팝업 메뉴에서 Fold/Unfold to unfold the instance 는 선택하면, 아래와 같이 이미지가 편하게 됩니다.

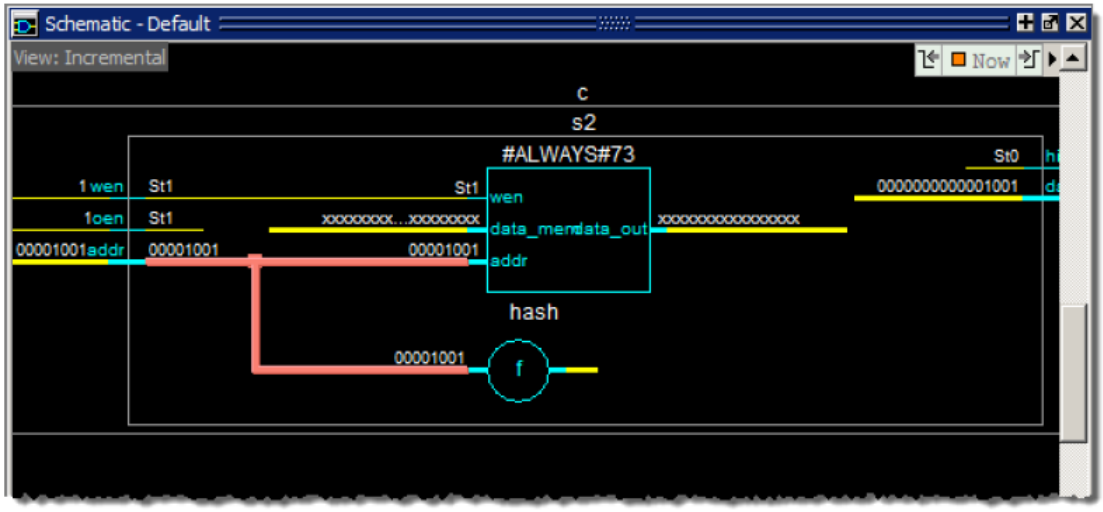
Figure 9-15. Unfolded Instance



3. s2 Module 의 내부 살펴보기

A. addr pin 을 s2 Module 내부 쪽에서 선택하고 더블 클릭을 하면 아래의 이미지처럼 addr pin 과 연결된 gate 와 내부 Instance 들이 보여지게 됩니다.

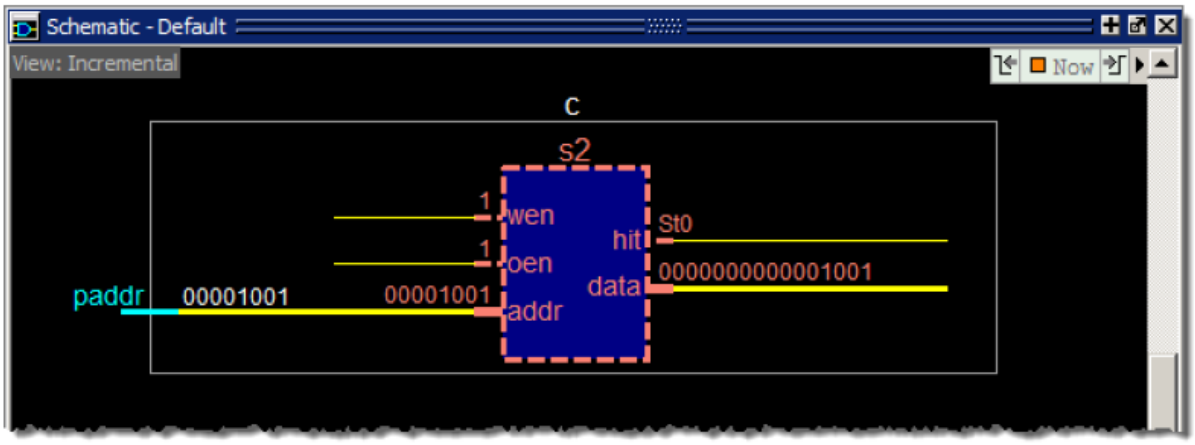
Figure 9-16. Contents of Unfolded Instance s2



4. s2 Module Fold 하기

   A. s2 Module 하이라이트 처리된 외곽라인에서 마우스 오른쪽 버튼을 클릭하여 Fold/Unfold 를 선택합니다.

Figure 9-17. Instance s2 Refolded

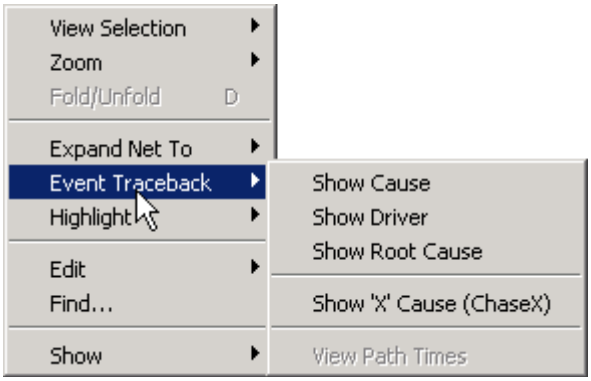


다른 fold module 인 s0, s1 s3 도 마찬가지로 확인해 봅니다. 확인이 끝났으면 **Delete All** 버튼을 클릭하여 Schematic 윈도우를 Clear 합니다.

Tracing Events

Schematic 윈도우는 Event 원인을 추적할 수 있는 기능을 가지고 있습니다. 이 Event Traceback 기능은 Incremental View 모드에서 마우스 오른쪽 버튼을 클릭하여 나오는 팝업 메뉴에서 사용할 수 있습니다.

Figure 9-18. Event Traceback Menu Options



Event Tracing 기능은 현재 설정 되어 있는 Active time 에서 시작 됩니다.

- Wave 윈도우에서 선택한 Cursor 의 Time
- Schematic 윈도우 embedded wave viewer 에서 선택한 Cursor 의 Time
- Schematic 윈도우의 Active Time

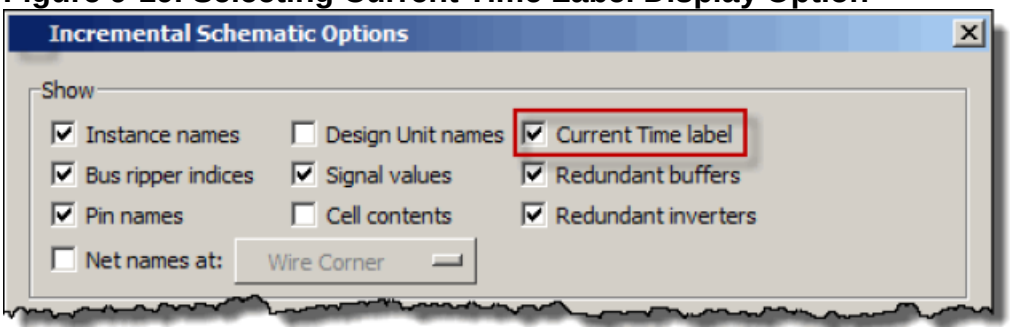
보통은 Wave 윈도우 혹은 Schematic 윈도우의 embedded wave viewer 에서 cursor 로 지정된 Active time 을 사용하지만, Schematic 윈도우에 Active time label 을 설정 할 수 있습니다.

1. Incremental View 에 Active time 표시하기

   A. Schematic 윈도우가 활성화 되어 있는 상태에서, **Schematic > Preferences** 를 선택하여 옵션 윈도우를 엽니다.

   B. 옵션 윈도우의 옵션 중에서 Active Time label 을 선택하고 OK 버튼을 클릭합니다.

Figure 9-19. Selecting Current Time Label Display Option



아래의 이미지처럼 Schematic 윈도우의 오른쪽 상단에 Active time 이 표시됩니다.

Figure 9-20. CurrentTime Label in the Incremental View

2. Schematic 윈도우에 object 추가하기

   A. Structure (sim) 윈도우에서 p 를 선택합니다.

   B. p 를 선택한 후 Objects 윈도우에서 t_out signal 을 schematic 윈도우로 드래그 합니다.

3. Schematic 윈도우의 Wave viewer 열기

   A. toolbar 에 있는 Show Wave 버튼을 클릭합니다.

4. Schematic 윈도우의 Wave viewer 윈도우를 통해 process signals 보기

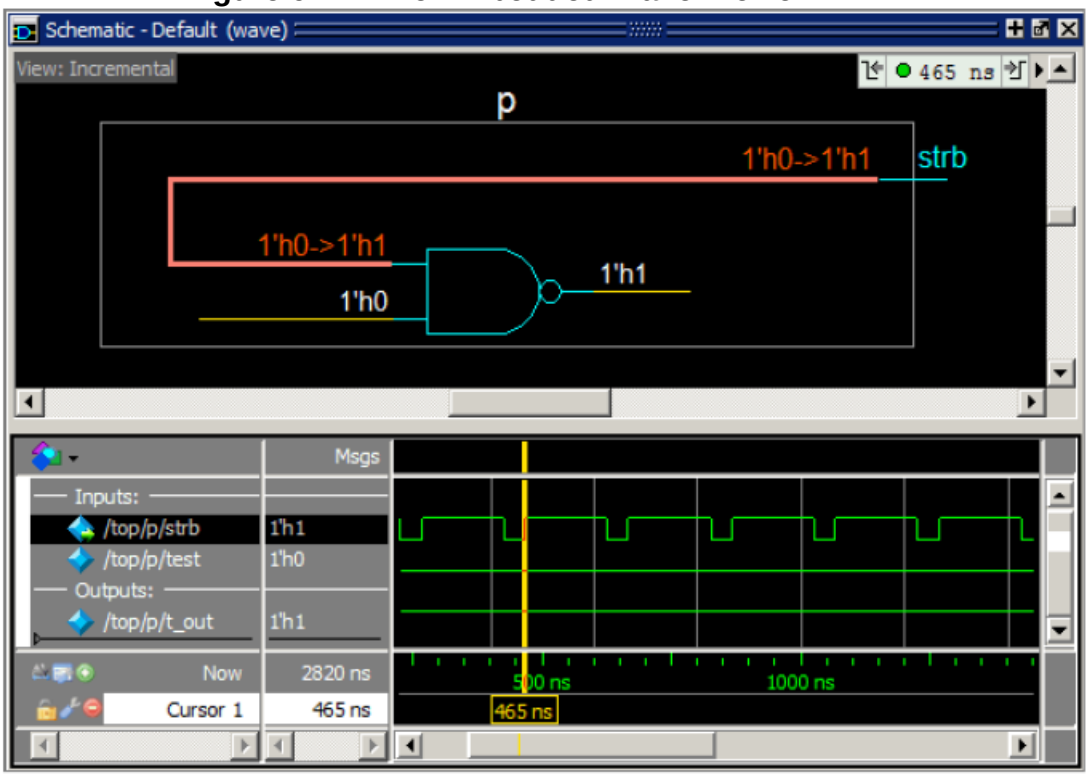
   A. Schematic 윈도우에서 #NAND#50 gate (VHDL Design 에서는 Line_71)를 선택합니다. Wave Viewer 에서는 선택한 gate 의 inputs 과 outputs 이 정렬되어 보여지게 됩니다.

5. Wave viewer 에서 cursor 움직이기

   A. Wave viewer 에서 strb signal 을 선택하고 cursor 를 465ns 로 움직입니다.

      Schematic 윈도우에서는 아래의 그림처럼 strb 가 highlight 처리가 됩니다.

Figure 9-21. The Embedded Wave Viewer



Schematic 윈도우의 오른쪽 상단 부에 cursor 로 인해 활성화된 465ns 라는 시간이 표시됩니다.

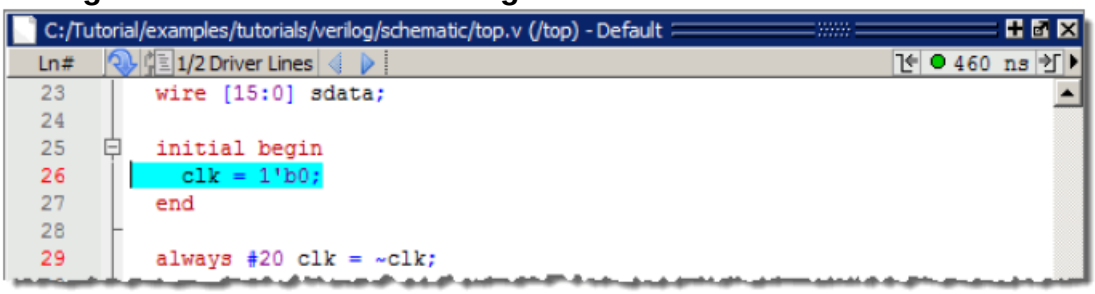
6. Event 추적하기

A. Wave viewer 윈도우의 파형 부분에서 마우스 오른쪽 버튼을 클릭하여 pop-up 메뉴를 Open 합니다.

B. pop-up 메뉴에서 Event Traceback > Show Cause 선택합니다.

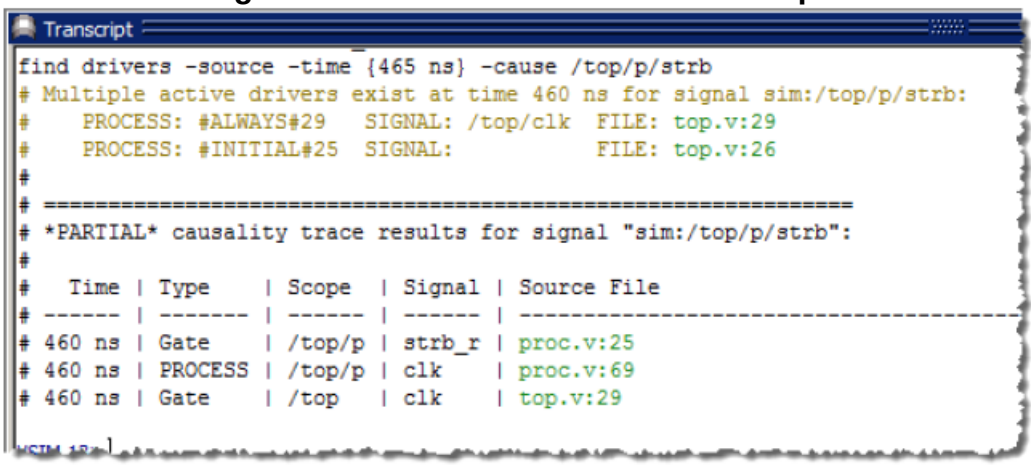
아래의 그림처럼 Source 윈도우가 열리면서 해당 code 가 highlight 처리됩니다.

Figure 9-22. Immediate Driving Process in the Source Window



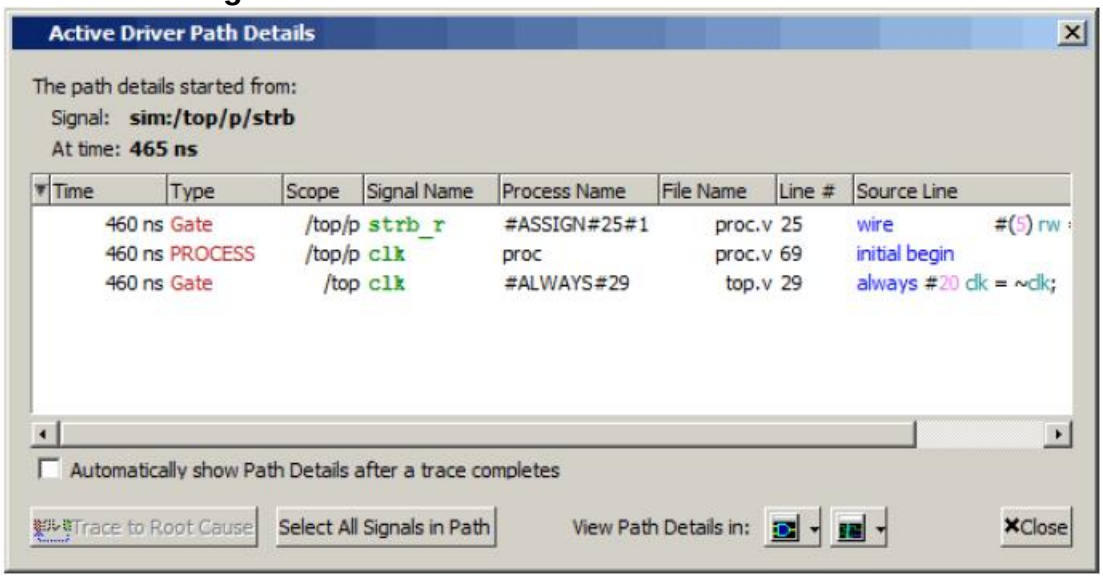
또한 아래 그림과 같이 Transcript Window 에 결과가 나타납니다.

Figure 9-23. Result of Trace in Transcript



Path Details 를 보기 위해서는  아이콘을 클릭합니다. 그러면 Active Driver Path Details 윈도우가 아래와 같이 나타납니다. Active Driver Path Details 윈도우에서는 선택한 event 의 sequential process 에 대한 정보를 보여줍니다.

Figure 9-24. Active Driver Path Details Window

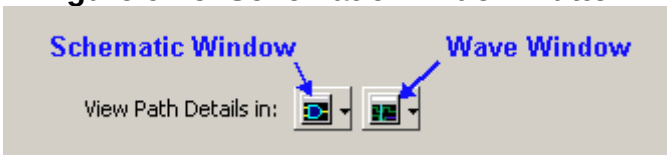


7. Schematic 윈도우에서 #ASSIGN#25#1 process 에서 strb_r 의 path detail 살펴보기

   A. Active Driver Path Details 윈도우에서 strb_r 을 선택합니다.

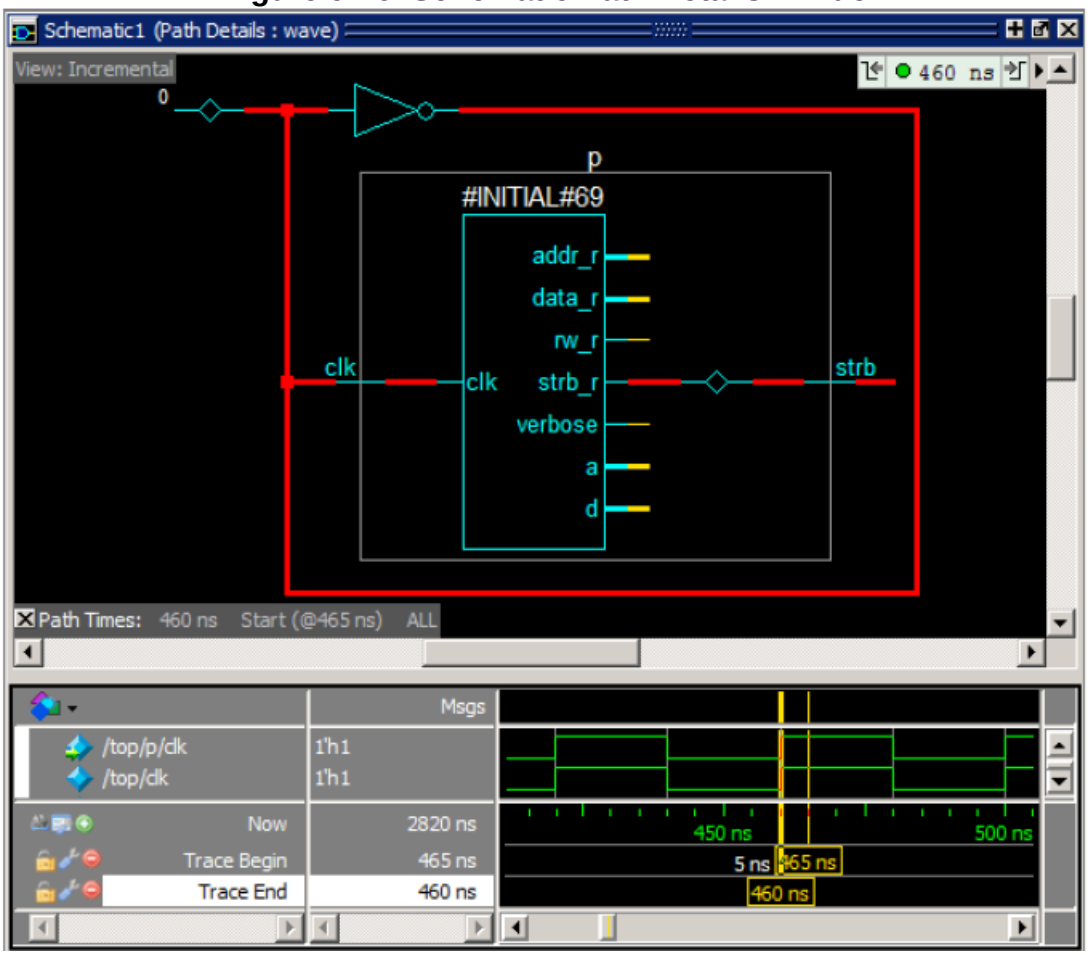
   B. Active Driver Path Details 윈도우 하단에서 Schematic Windows 버튼을 클릭합니다.

Figure 9-25. Schematic Window Button



이제 선택했던 signal 에 대해 Detail 한 Path 정보다 dedicate 된 Schematic 윈도우에 아래와 같이 보여집니다.

Figure 9-26. Schematic Path Details Window



Wave viewer 에서는 Trace start 부터 end 부분까지 보여지게 됩니다.

8. 추가적으로 계속 진행하기 전에 Schematic 윈도우 clear 하기

   A. Active Driver Path Detail 윈도우를 닫습니다.

   B. Schematic Path Detail 윈도우를 닫습니다.

   C. Original Schematic 윈도우를 Schematic 탭 메뉴를 이용하여 선택합니다.

   D. Schematic Viewer 윈도우에서 **Delete All** 아이콘을 클릭합니다.

   E. Schematic Viewer 윈도우의 Wave view 를 닫기 위해 **Show Wave** 아이콘을 클릭합니다.

Chapter 10
Debugging With The Dataflow Window

Dataflow 윈도우는 여러분이 Simulation 을 진행하려는 Design 의 Signal 의 Physical 연결 상태를 한눈에 보여주고 Output 에서 Input 까지 trace 를 가능하게 해주는 기능을 제공하고 있습니다.

이번 챕터에서는 몇몇 memory 를 가지고 있는 cache 모듈을 test bench 를 가지고 검증을 하면서 Dataflow 윈도우를 살펴볼 것입니다. 예제 파일의 위치는 다음과 같습니다.

Verilog – *<install_dir>/examples/tutorials/verilog/dataflow*

VHDL – *<install_dir>/examples/tutorials/vhdl/dataflow*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Debugging with the Dataflow Window 와 Dataflow Window 을 보시면 좀 더 자세히 알 수 있습니다.

Compile and Load the Design

이번 챕터에서는 Compile 과 Design Load 를 Do 파일을 가지고 할 것입니다.

1. 새로운 디렉토리를 만들고 tutorial file 복사하기

   이번 챕터를 진행하기 위한 directory 를 만듭니다.

   그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. ModelSim/Questa 을 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 ModelSim/Questa 을 구동합니다.

   B. File>change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. WildcardFilter 설정 변경하기

   A. Transcript 윈도우에 아래와 같이 command 를 입력합니다.

      set WildcardFilter "Variable Constant Generic Parameter SpecParam Memory Assertion Endpoint ImmediateAssert"

      위의 command 를 입력하면 앞으로 "CellInternal" 이 제거되어 debugging 하는데 있어서 모든 signal 을 볼 수 있도록 변경이 됩니다.

4. Do 파일 실행하기
   A. do run.do 을 입력합니다.

      run.do 파일은 아래의 작업들을 진행하게 됩니다.

      i. Work library 만들기

      ii. Design File Compile 하기

      iii. Design Optimized 하기

      iv. Simulator 에 Design 로드하기

      v. Wave 윈도우에 Signal 추가하기

      vi. Design 상의 모든 Signal log 하기

      vii. Simulation 진행하기

Exploring Connectivity

Dataflow 윈도우의 가장 큰 장점은 Design 의 Physical 연결 상태를 보여준다는 것 입니다. 이제 Dataflow 윈도우의 기능들에 대해서 살펴보겠습니다.

1. Dataflow 윈도우 열기

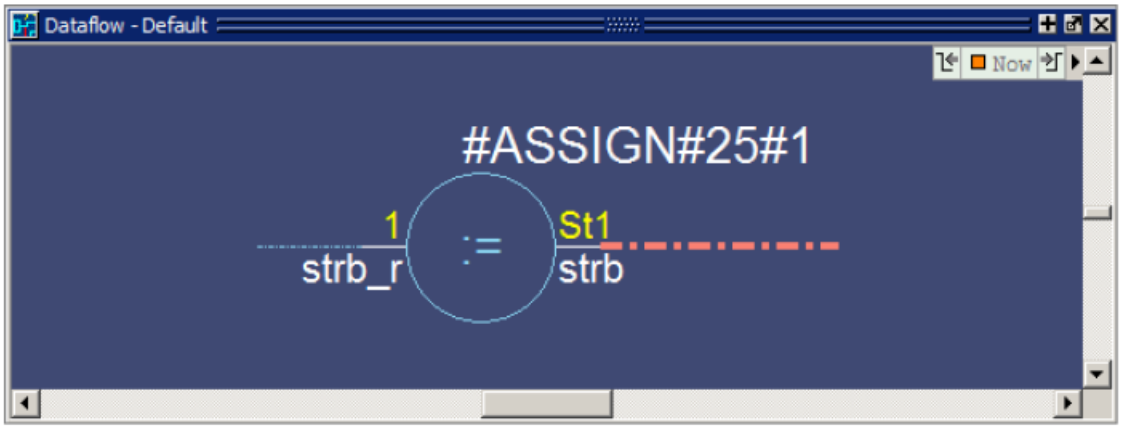
   A. View>dataflow 를 선택하거나 view dataflow command 를 입력하여 dataflow 윈도우를 엽니다.

2. Dataflow 윈도우에 signal 추가하기

   A. Structure(sim) 탭에서 p instance 를 선택합니다.

   B. Object 윈도우에서 strb 를 선택하여 Dataflow 윈도우로 드래그 합니다.

Figure 10-1. A Signal in the Dataflow Window

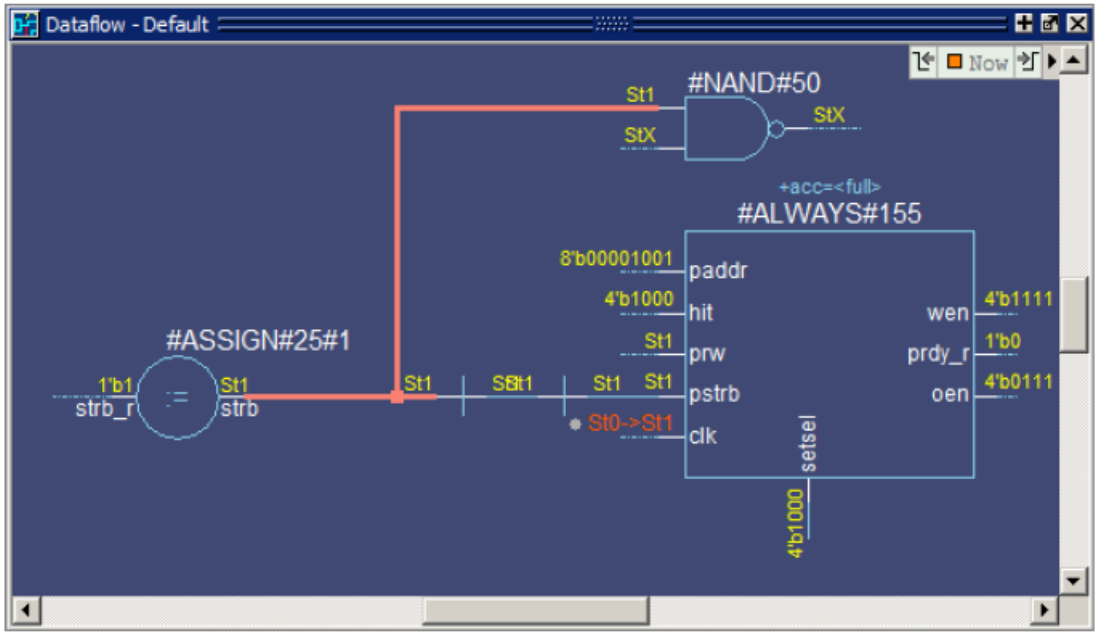


3. Design 관찰하기

A. 위의 화면에서 red 로 하이라이트 처리된 부분을 더블클릭합니다.

아래의 그림처럼 strb 에 연결된 process 들이 확장되어 보여집니다.

Figure 10-2. Expanding the View to Display Connected Processes

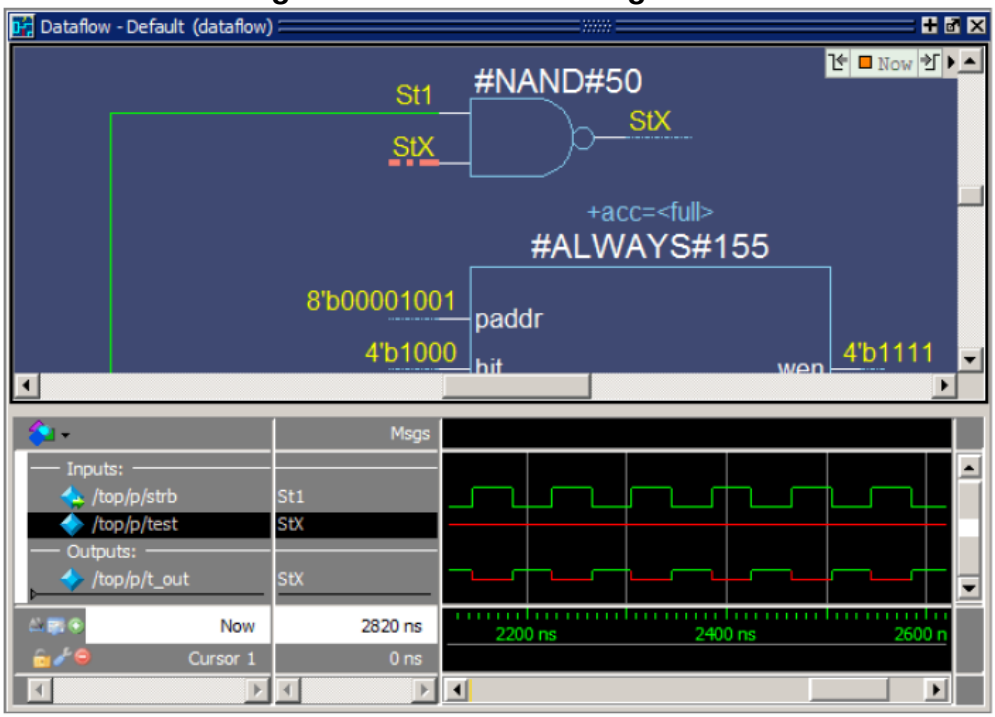


B. #NAND#50 process 의 test signal 의 drive signal 찾기

i. 메인 메뉴에서 Show Wave  아이콘을 클릭하면 Dataflow 윈도우에 Wave 윈도우가 추가되는 것을 볼 수 있습니다.

ii. #NAND#50 process 를 클릭하면 추가된 Wave 윈도우에 #NAND#50 에 해당하는 Input, Output 들이 정렬되어 파형을 보여줍니다.

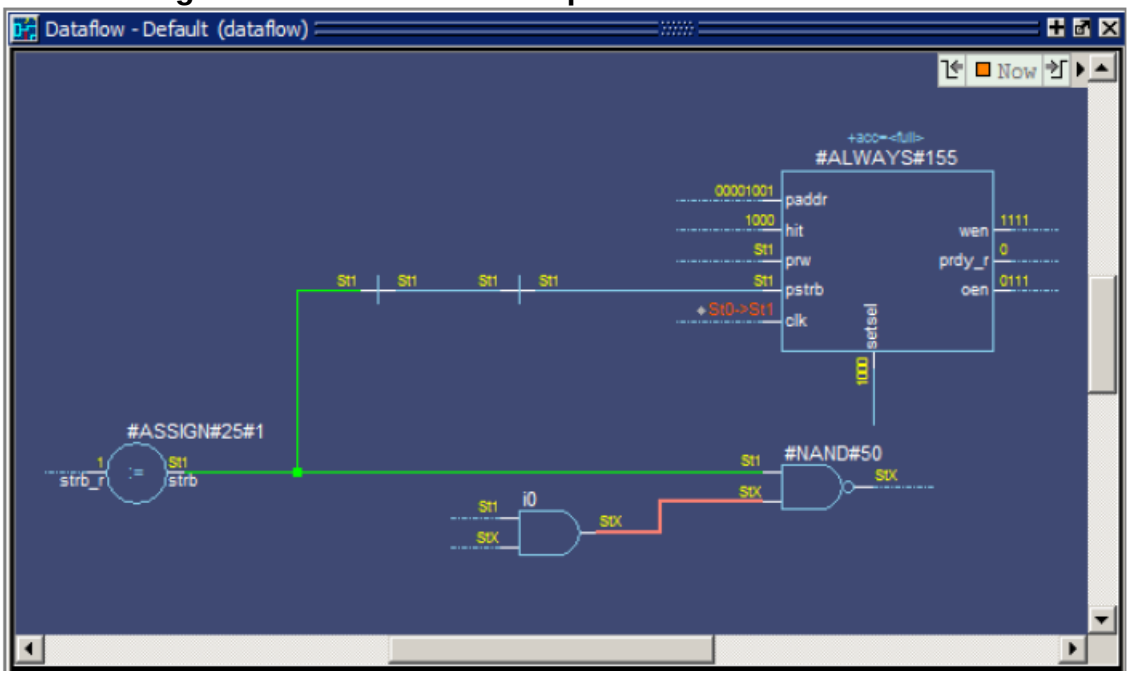
Figure 10-3. Select *test* signal



iii. Wave 에서 test 를 선택하면 Dataflow 윈도우에 자동적으로 해당 signal 이 빨간색으로 하이라이트 처리되어 보여집니다.

iv. 하이라이트 처리된 부분을 선택하고 더블 클릭을 하면 해당하는 signal 여 연결된 모든 driver 들이 펼쳐집니다.

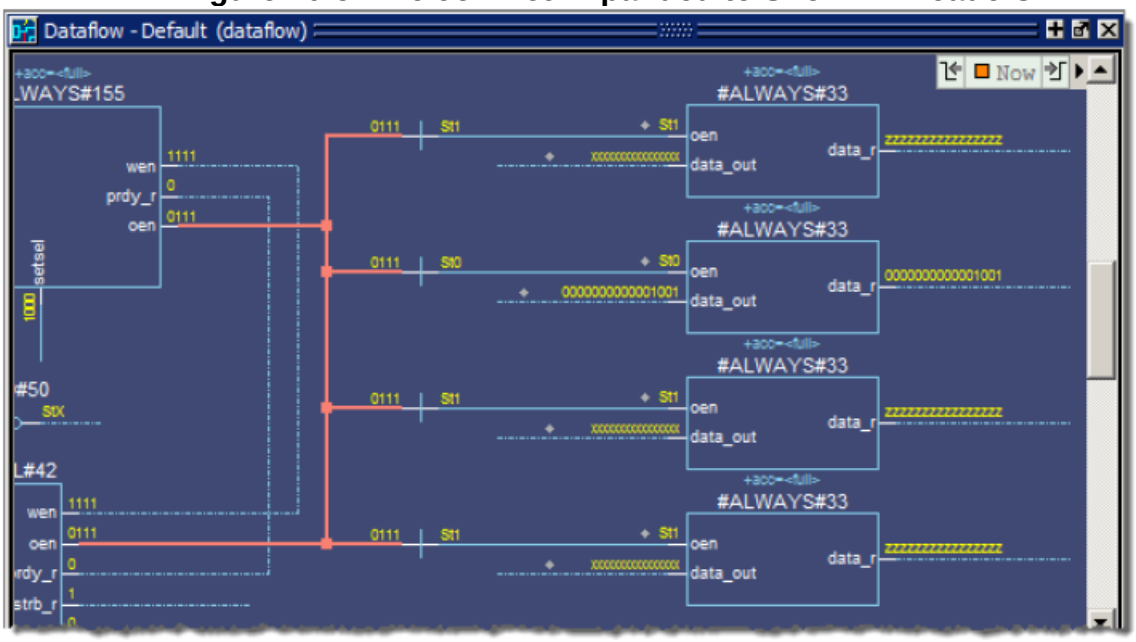
Figure 10-4. The *test* Net Expanded to Show All Drivers



v. #ALWAYS#155 process 의 oen signal 을 선택하고 Expand net to all readers

icon 을 클릭하면 마찬가지로 모든 driver 들이 펼쳐집니다.

Figure 10-5. The *oen* Net Expanded to Show All Readers



Tracing Events

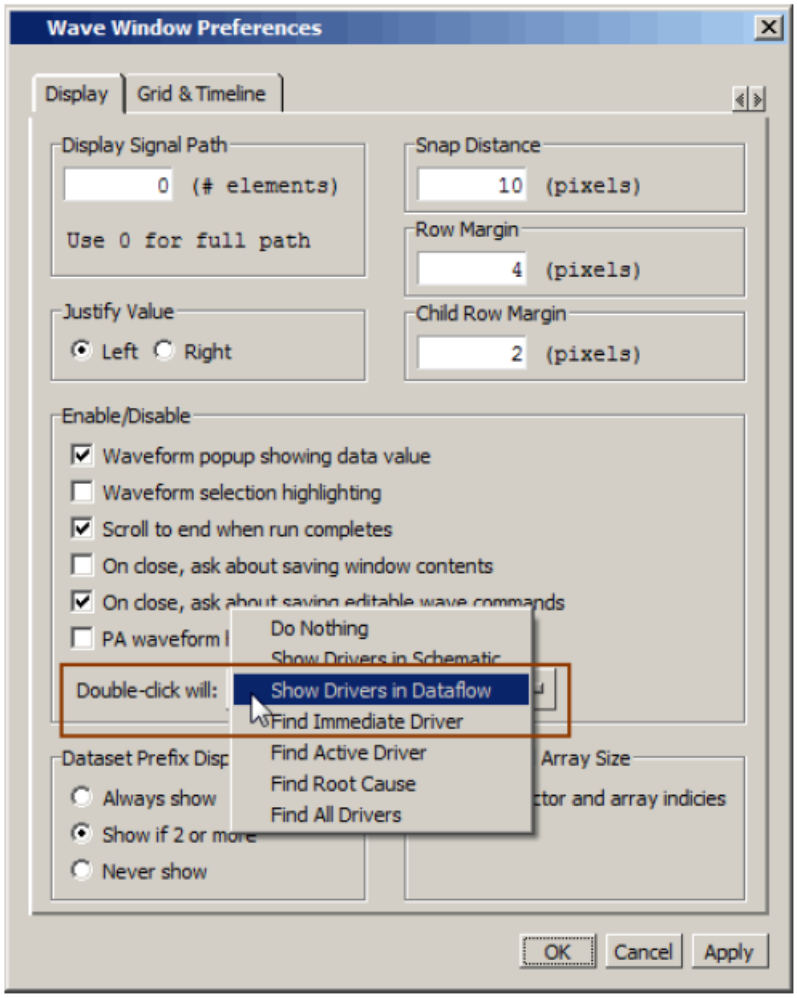
Dataflow 윈도우를 통해 작업하면 생기는 또 다른 이점은 Display 되는 Signal 의 Value 를 바로 확인하는 것이 가능한 것 입니다. Wave 윈도우에서 Cursor 를 움직이면 Dataflow 윈도우의 signal 들 역시 바로 바로 Value 가 변화는 것을 확인 할 수 있습니다.

1. Dataflow 윈도우에서 더블 클릭 마우스 설정 변경하기

   A. Wave 윈도우에서 Wave>Wave Preferences 를 선택하면 해당 창이 열립니다.

   B. Double-click will : 메뉴에서 Show Drivers in Dataflow Window 를 선택합니다.

Figure 10-6. Wave Window Preferences Dialog Box



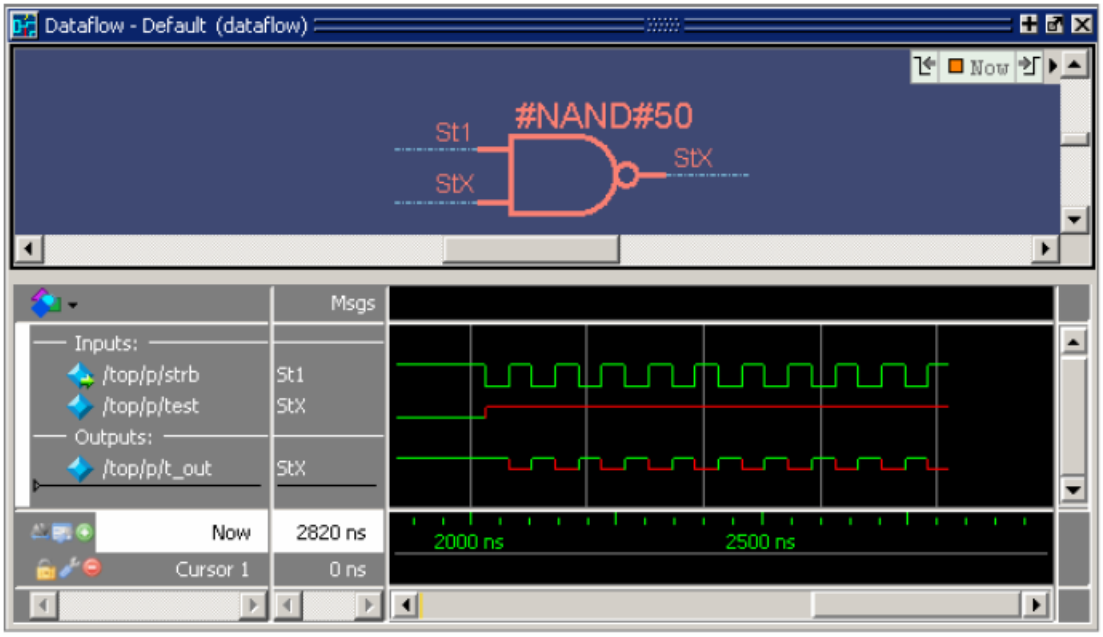
2. Dataflow 윈도우로 Object 추가하기

   A. Structure(sim) 윈도우에서 p instance 를 선택합니다.

   B. Object 윈도우에서 t_out 를 선택하여 Dataflow 윈도우로 드래그 합니다.

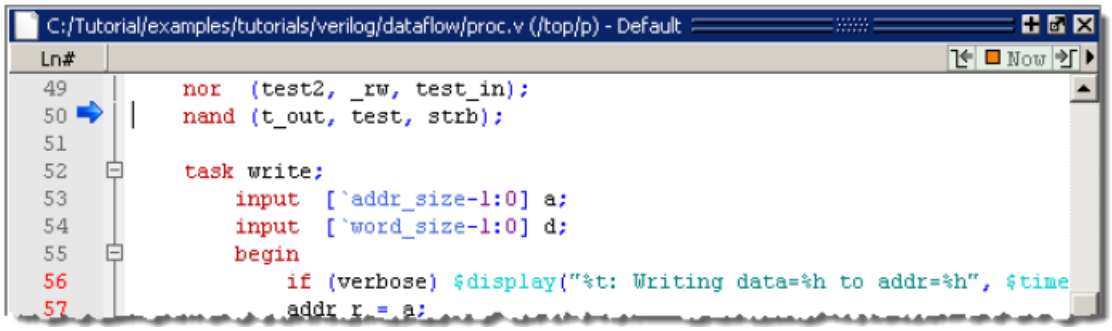
   C. Show wave 아이콘  을 클릭하면 Dataflow 윈도우에 Wave 윈도우가 추가됩니다.

Figure 10-7. The Embedded Wave Viewer



3. #NAND#50 의 Input 추적하기
   A. Dataflow 윈도우에서 #NAND#50 process 를 더블 클릭하면 Source 윈도우에서는 해당 Source 가 아래와 같이 보여집니다.

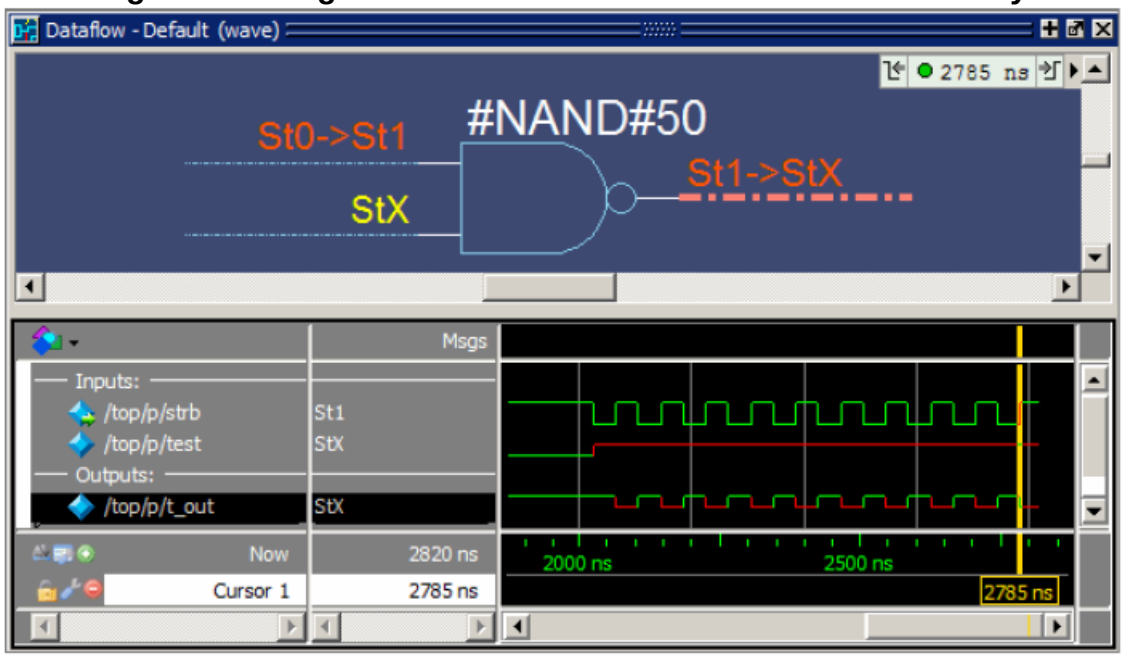
Figure 10-8. Source Code for the NAND Gate



   B. Dataflow 윈도우로 돌아가면 #NADN#50 의 모든 Input, Output 이 보여집니다.

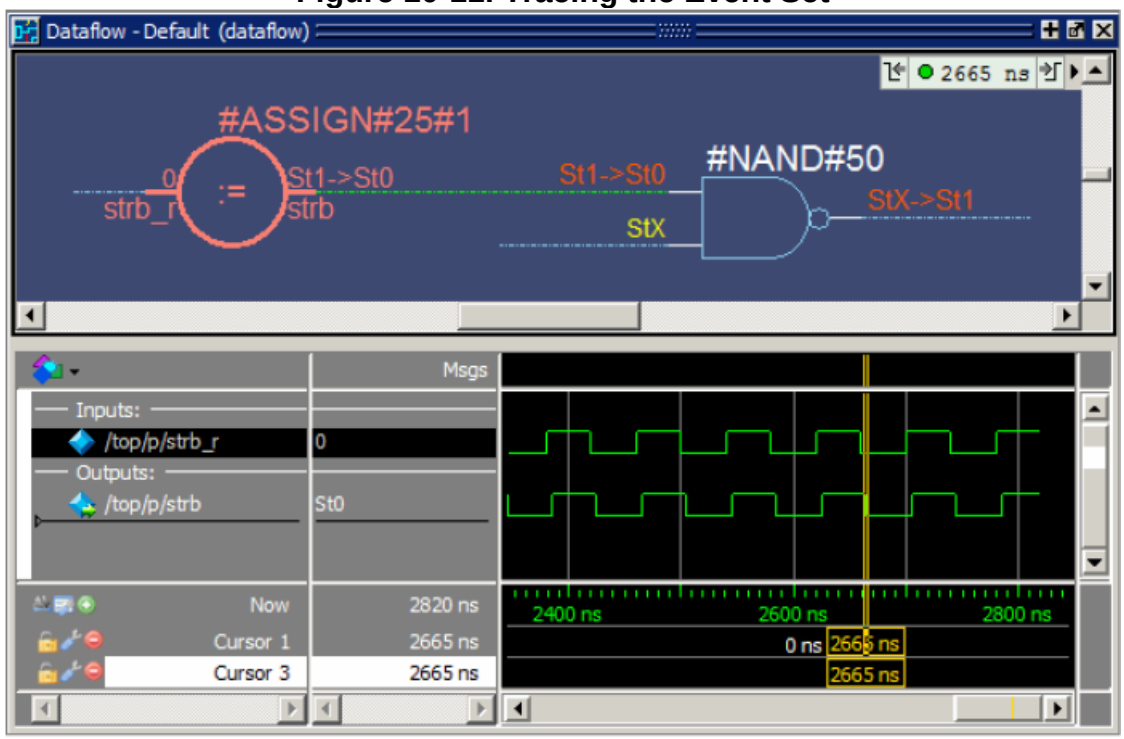
   C. Wave 윈도우에는 t_out 의 마지막 transition 에 Cursor 가 표시됩니다.

Figure 10-9. Signals Added to the Wave Viewer Automatically



D. Tools>Trace>Trace next event 를 선택하면 Cursor 가 다음 event 로 이동합니다.

Figure 10-11. Tracing the Event Set



이 방식으로 디자인의 어느 부분에서 이벤트가 발생하고 있는지 계속해서 추적이 가능합니다.

Tracing an X (Unknown)

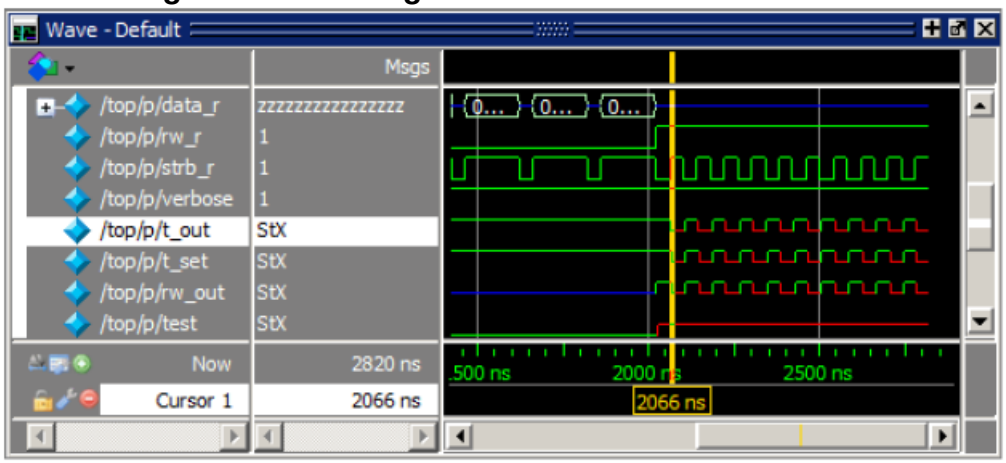
Dataflow Window 는 디자인에서 알 수 없는 값(X) 를 쉽게 추적할 수 있습니다. Wave Window 를 확인하여 X 가 나왔을 때 문제의 원인을 추적하기 위해 Dataflow Window 를 사용할 수 있도록 Dataflow Window 가 동적으로 Wave Window 에 링크되어 있습니다. Dataflow 에서 디자인을 클릭하면 Wave Window 에 해당하는 signal이 자동으로 나타나게 됩니다.

1. Wave Window 와 Dataflow Window 에서 t_out 보기

   A. Wave Window 에서 /top/p/t_out 을 찾습니다.

      t_out Signal 은 2066 ns 부터 제대로 된 값과 알 수 없는 값이 전환하고 있습니다. 빨간색 파형은 알 수 없는 값을 나타냅니다.

Figure 10-12. A Signal with Unknown Values

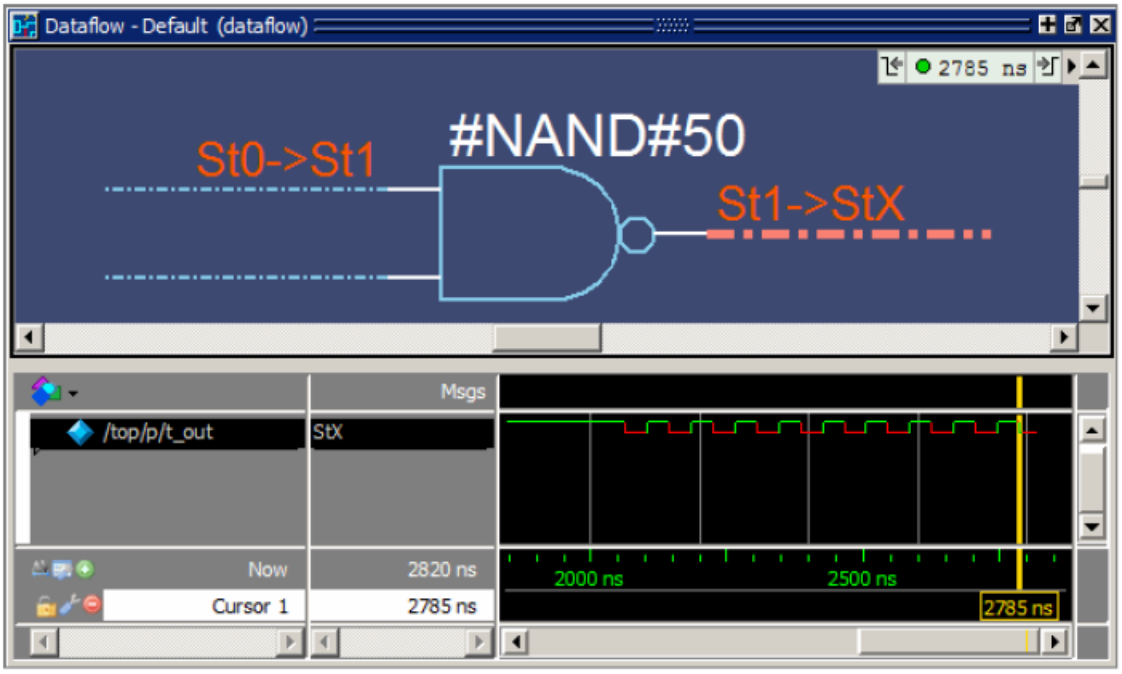


   B. 2786 ns 에서 t_out signal 을 더블 클릭합니다.

      Source code view 가 열리면서 t_out 신호를 나타납니다.

      Wave Window 에서 waveform 을 더블 클릭하면, Dataflow 가 자동으로 열리면서 t_out 에 관련된 process 를 표시합니다.

Figure 10-13. Dataflow Window with Wave Viewer

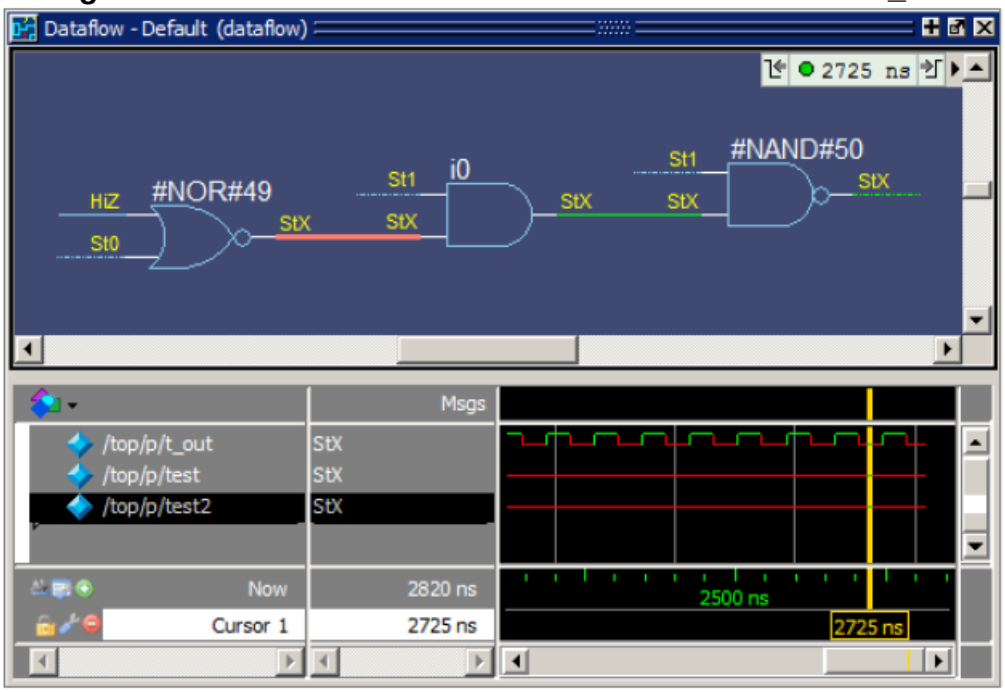


C. 커서의 위치를 보면 2785 ns 에 t_out signal 값을 알 수 없습니다.

2. Unknown 추적

A. 메뉴에서 Tools>Trace>ChaseX 를 선택합니다.
디자인에서 어느 부분부터 알 수 없는 값이 나왔는지 추적하며 확장됩니다. 확인해보면 #NOR#49 에서 알 수 없는 값이 나오는 것을 Debugging 할 수 있습니다.

Figure 10-14. ChaseX Identifies Cause of Unknown on t_out



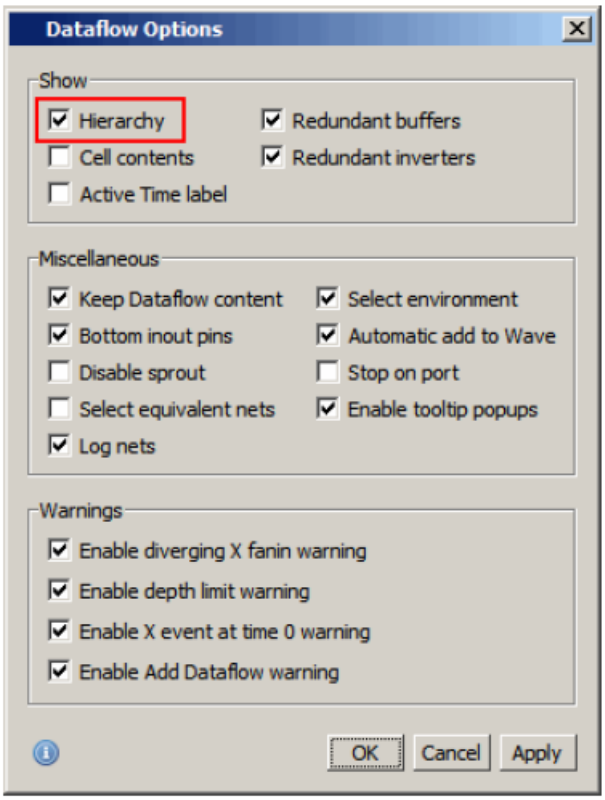
Displaying Hierarchy in the Dataflow Window

Dataflow 윈도우에서 Design 의 Signal 연결 상태를 Design Hierarchy 를 보면서 관찰을 할 수 있습니다.

1. Display option 변경하기

   A. Dataflow>Dataflow preferences>Options 를 선택합니다.

Figure 10-15. Dataflow Options Dialog Box

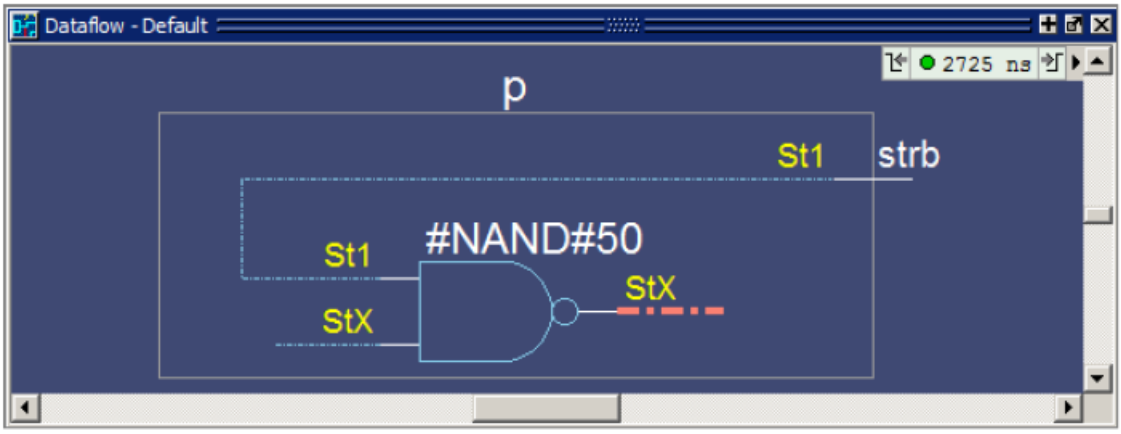


   B. 창이 나타나면 Show 부분에서 Hierarchy 를 선택합니다.

2. t_out signal Dataflow 에 추가하기

   A. Transcript 윈도우에 add dataflow /top/p/t_out 를 입력합니다.
   이번에는 Hierarchy 구조와 함께 Design 정보가 보여집니다.

Figure 10-16. Displaying Hierarchy in the Dataflow Window



Chapter 11
Viewing And Initializing Memories

이번 챕터에서는 Design 상의 Memory 정보를 보는 방법과 초기화 하는 방법에 대해서 살펴볼 것입니다. ModelSim/Questa 에서 살펴볼 수 있는 Memory 의 종류는 아래와 같습니다.

- reg, wire. And std_logic arrays
- Integer arrays
- Single dimensional arrays of VHDL enumerated types other than std_logic

이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

Verilog – *<install_dir>/examples/tutorials/verilog/memory*

VHDL – *<install_dir>/examples/tutorials/vhdl/memory*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Memory List Window 을 보시면 더욱 자세히 알 수 있습니다.

Compile and Load the Design

1. 새로운 디렉토리를 만들고 tutorial file 복사하기

   이번 챕터를 진행하기 위한 directory 를 만듭니다.

   그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. ModelSim/Questa 를 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 ModelSim/Questa 를 구동합니다.

   B. File>change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. Library 만들고 Compile 하기

   A. Verilog 의 경우
      i. vlib work

vlog *.v 를 입력합니다.

B. VHDL 의 경우

i. vlib work

vcom -93 sp_syn_ram.vhd dp_syn_ram.vhd ram_tb.vhd 를 입력합니다.

4. Design 최적화 하기

A. 아래의 Command 를 입력합니다.

i. vopt +acc ram_tb –o ram_tb_opt 를 입력합니다.

최적화된 ram_tb_opt 라는 최적화된 library 가 만들어 집니다.

5. Design 로드하기

A. 아래의 Command 를 입력합니다.

i. vsim ram_tb_opt

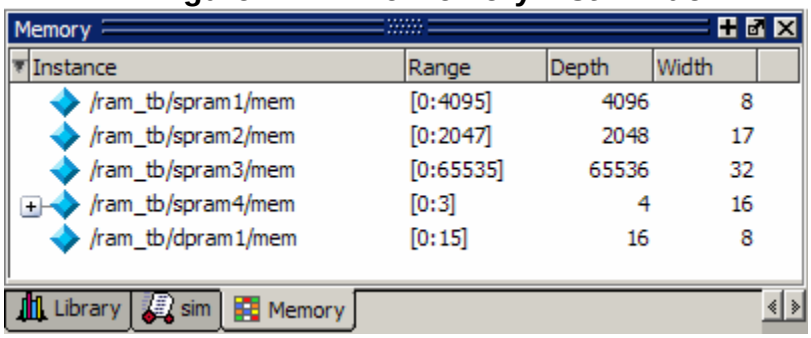
View a Memory and its Contents

Memory 윈도우에서는 모든 Memory 리스트를 볼 수 있고, 각 instance 별로 range, depth, width 정보는 물론, 해당하는 instance 를 더블 클릭하면 해당 Memory data 를 보여줍니다.

1. Memory 윈도우 열고 Data 보기

A. View>Memory List 를 선택하면 아래와 같이 Memory 윈도우가 열립니다.

Figure 11-1. The Memory List Window

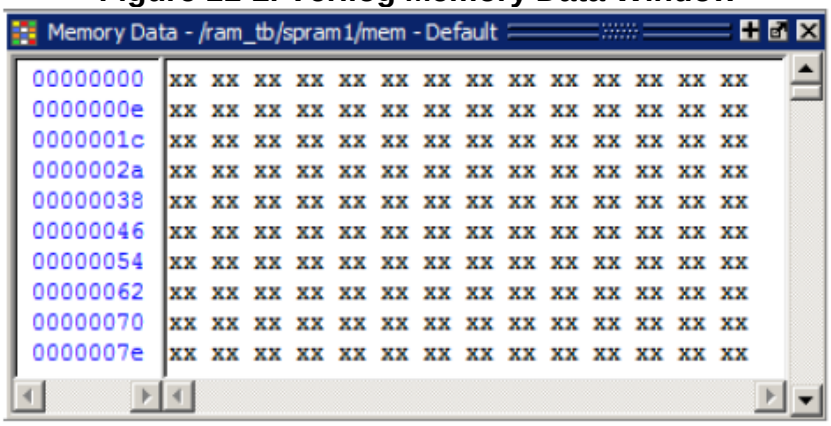


B. /ram_tb/spram1/mem 을 더블 클릭합니다.

Memory data 윈도우에 mem 의 data 들이 보여집니다. Address 와 data 로 구별되

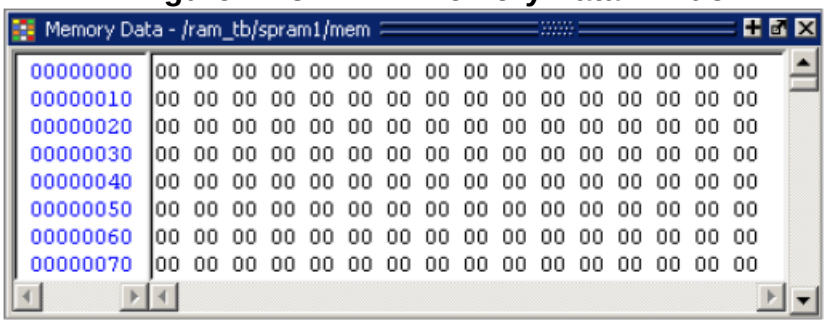
어서 나타나며, mem 의 경우엔 data 가 알 수 없는 값인 X 로 나타나는 것을 볼 수 있습니다.

Figure 11-2. Verilog Memory Data Window



만약 VHDL Design 을 진행 중이라면 해당 결과는 아래와 같이 나타납니다.

Figure 11-3. VHDL Memory Data Window



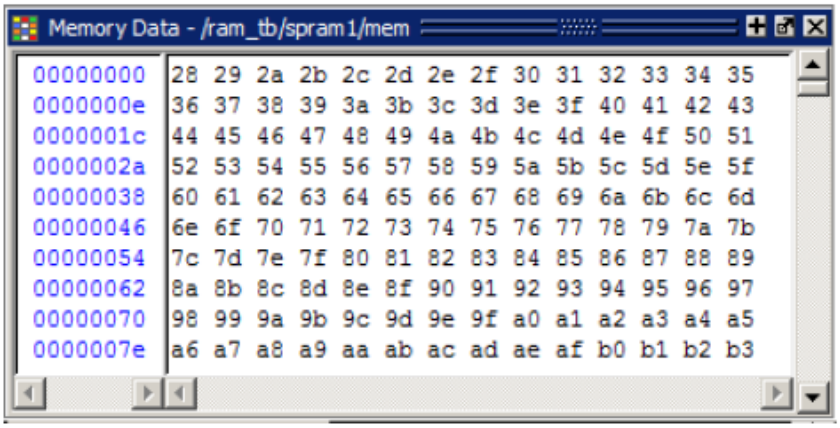
2. Simulation 진행하기

   A. 메인 윈도우에서 run –all 아이콘을 클릭합니다. 

   Simulation 은 Source code 에서 break 가 걸리 때까지 Simulation 이 진행됩니다.

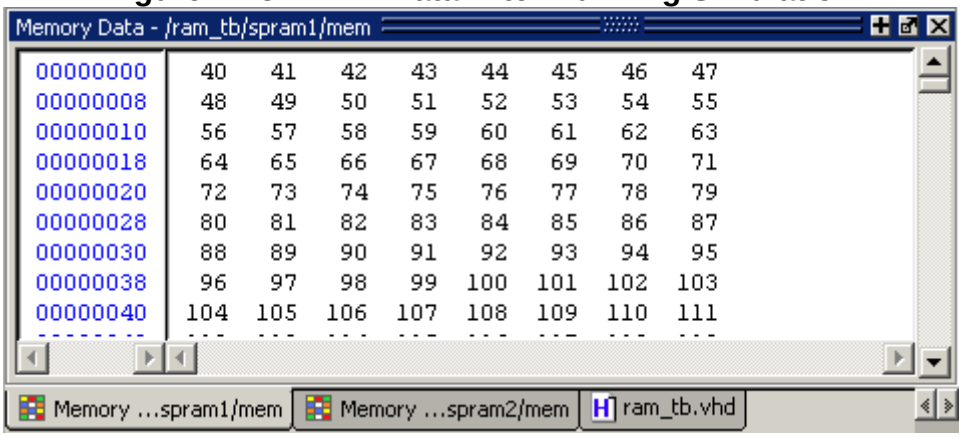
   B. Simulation 이 진행된 후에 /ram_tb/spram1/mem 를 더블 클릭하면 이번에는 아래와 같이 mem 에 data 들이 들어가 있는 것을 확인 할 수 있습니다.

Figure 11-4. Verilog Data After Running Simulation



VHDL Design 경우엔 아래와 같이 나타납니다.

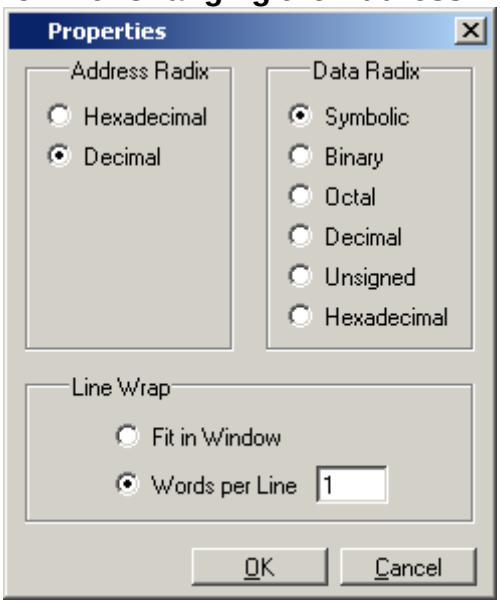
Figure 11-5. VHDL Data After Running Simulation



3. 이번에는 /ram_tb/spram1/mem 의 address radix 와 같은 것을 변경해보기

   A. Memory 윈도우에서 마우스 오른쪽을 클릭하여 Properties 를 클릭합니다. 창이 아래와 같이 나타납니다.

Figure 11-6. Changing the Address Radix



B. Address Radix 를 Decimal 로 체크를 하고, Words per Line 을 1로 저장하고 OK 버튼을 클릭합니다.

클릭하면 VHDL 과 Verilog 에 따라 아래와 같이 결과가 보여집니다.

Figure 11-7. New Address Radix and Line Length (Verilog)

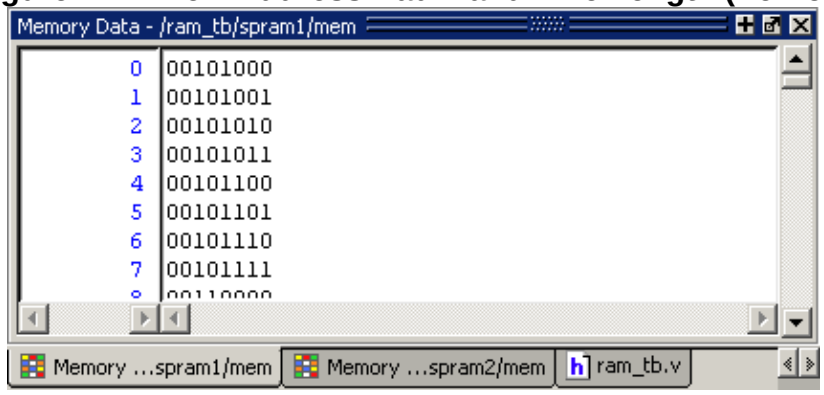
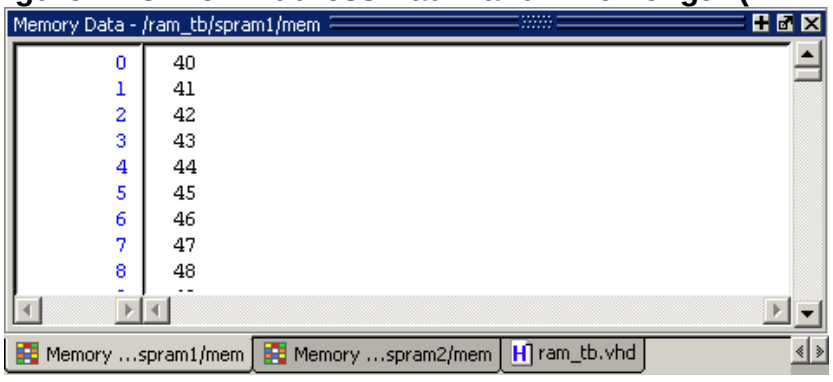


Figure 11-8. New Address Radix and Line Length (VHDL)



Navigate Within the Memory

Memory 윈도우를 통해 Memory 내용을 보는 것뿐만 아니라 특정 데이터의 패턴이나 주소 값을 이용해 원하는 부분으로 이동을 할 수도 있습니다.

1. 특정 주소로 이동하기

   A. 마우스 오른쪽 버튼을 클릭하여 Goto 선택합니다.

   B. 아래의 그림과 같이 30 을 입력합니다.

Figure 11-9. Goto Dialog Box

   C. OK 버튼을 클릭합니다.

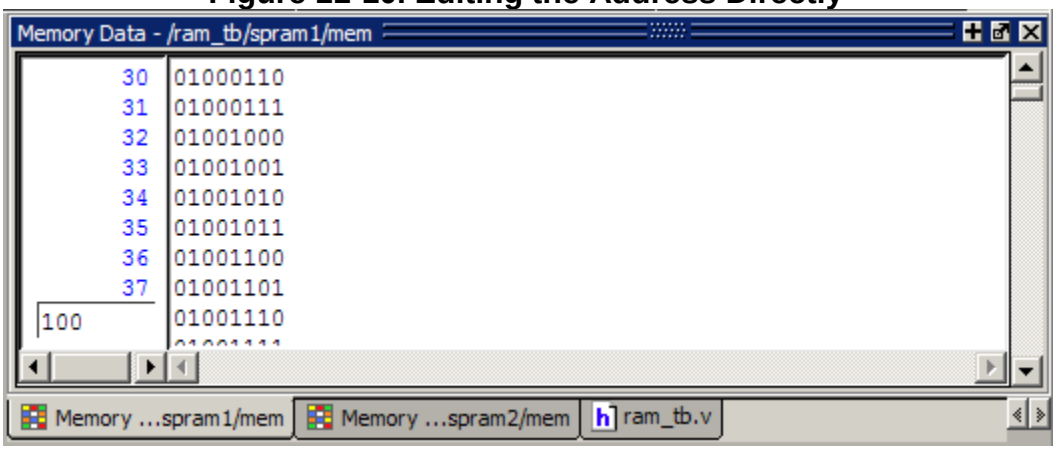
   D. 30번지 부분이 Memory 윈도우의 가장 위가 되게 표시됩니다.

2. 주소를 바로 수정하기

   A. 38번에 해당하는 주소 부분을 더블 클릭합니다.

   B. 100 을 입력합니다.

Figure 11-10. Editing the Address Directly



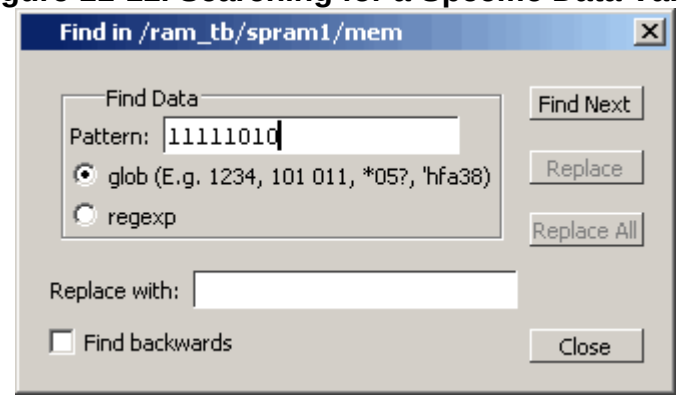
   C. 그러면 100 번지에 해당하는 주소로 이동을 합니다.

3. 데이터 패턴을 이용하여 찾기

A. 마우스 오른쪽 버튼을 클릭하여 Find 를 선택합니다.

B. 아래의 창이 열리면 Verilog 의 경우 11111010 을, VHDL 의 경우엔 250 을 입력하고 Find Next 를 클릭합니다.

Figure 11-11. Searching for a Specific Data Value



C. 해당하는 데이터로 움직이게 되고 계속 Find Next 를 클릭하면 그 다음 값으로 이동이 됩니다.

D. Close 를 클릭하면 창이 닫힙니다.

Export Memory Data to a File

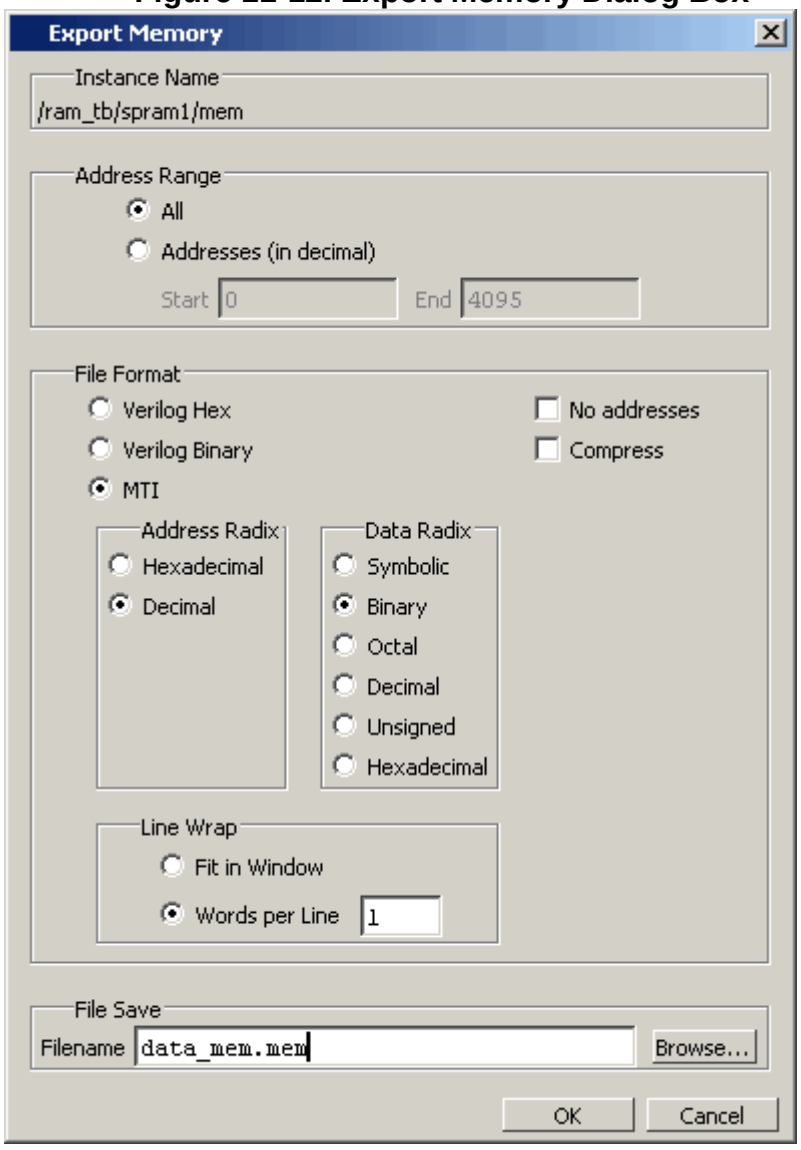
Simulation 중에 Memory 의 내용을 파일로 저장할 수 있습니다.

1. Memory 내용 저장하기

   A. /ram_tb/spram1/mem 을 선택하여 Memory 창에 보이게 합니다.

   B. File>Export>Memory data 를 클릭합니다.

Figure 11-12. Export Memory Dialog Box



C. Address radix 를 Decimal 를 선택합니다.

D. Data radix 는 Binary 를 선택합니다.

E. Words per Line 에는 1를 선택합니다.

F. Filename 에는 data_mem.mem 를 입력하고 OK 버튼을 클릭합니다.

그러면 현재 디렉토리에 해당 파일이 저장이 됩니다.

2. relocatable memory pattern file 로 저장하기

A. Memory Data Window 에서 /ram_tb/spram2/mem 을 선택합니다.

B. 오른쪽 마우스를 누른 후 Properties 를 선택하여 memory contents 팝업 메뉴를 엽니다.

C. Address Radix 를 Decimal 이나 Binary 로 선택을 한 후 close 를 누릅니다.

D. File>Export>Memory Data 를 선택합니다.

E. 주소 범위를 0 ~ 250 까지 지정하고 종료합니다.

F. 재배치를 하기 위해 File 포맷을 MTI 나 No addresses 로 생성 할 수 있으며, 다른 메모리에 사용할 수 있는 패턴을 만들 수 있습니다.

G. Address Radix 를 Decimal 로 선택하고, Data Radix 를 Binary 로 선택합니다.

H. Words per Line 을 1로 설정합니다.

I. 파일 이름을 reloc.mem 으로 한 후 메모리 내용을 저장합니다.

Initialize a Memory

ModelSim/Questa 에서는 메모리를 초기화하는 세 가지 방법(내보낸 메모리 파일, 채우기 패턴, 또는 두 가지 모두에서부터)이 있습니다.

먼저 파일을 통해 초기화 하는 방법을 살펴보겠습니다.

1. /ram_tb/spram3/mem 을 선택합니다.

   A. Memory 탭에서 /ram_tb/spram3/mem instance 를 더블 클릭합니다.

      Memory 윈도우에 /ram_tb/spram3/mem 이 보여지게 됩니다.

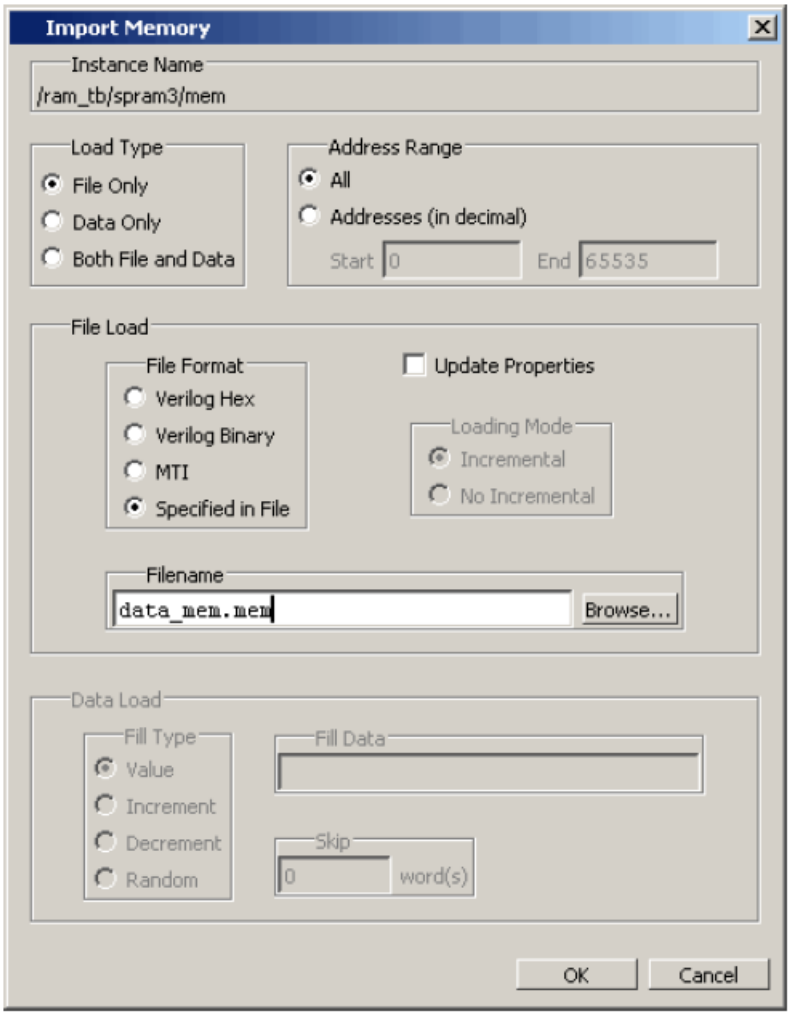
   B. Memory 윈도우에서 마우스 오른쪽 버튼을 클릭하여 Properties 를 선택합니다.

   C. Address Radix 를 Decimal 로 Data Radix 를 Binary 로 변경합니다. Words per Line 은 1로 변경합니다.

2. spram3 를 초기화

   A. data 아무 곳에서 마우스 오른쪽 버튼을 클릭하여 Import Data Patterns 을 선택합니다.

Figure 11-13. Import Memory Dialog Box



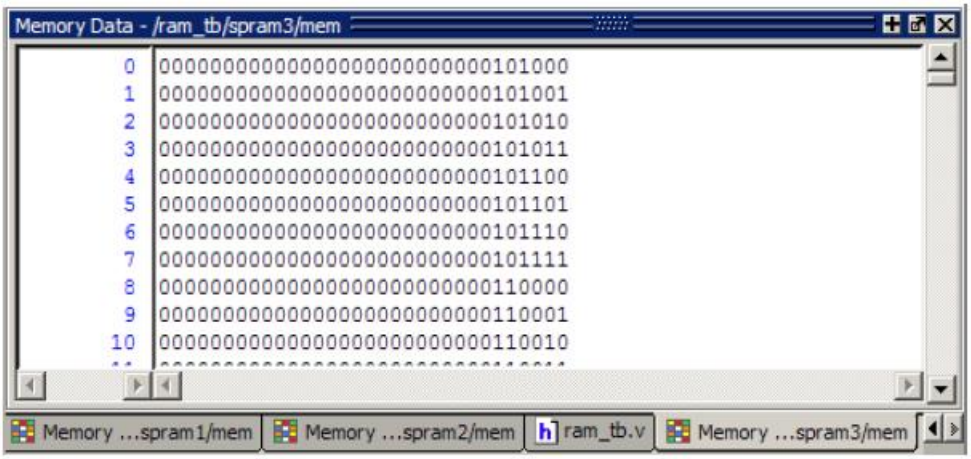
Load Type 은 File Only 가 기본설정 입니다.

B. 파일 이름을 data_mem.mem 으로 입력합니다.

C. OK 를 선택합니다.

data_mem.mem 에 /ram_tb/spram3/mem 의 주소가 업데이트 됩니다.

Figure 11-14. Initialized Memory from File and Fill Pattern



3. relocatable memory pattern(reloc.mem) 가져오기

   A. spram3 의 data 아무 곳에서 오른쪽 마우스를 누른 후 Import Data patterns 을 선택합니다.

   B. Load Type 을 both File and Data 로 변경합니다.

   C. 주소 범위를 0 ~ 300 으로 변경합니다.

   D. MTI File 포맷인 reloc.mem 을 불러 옵니다.

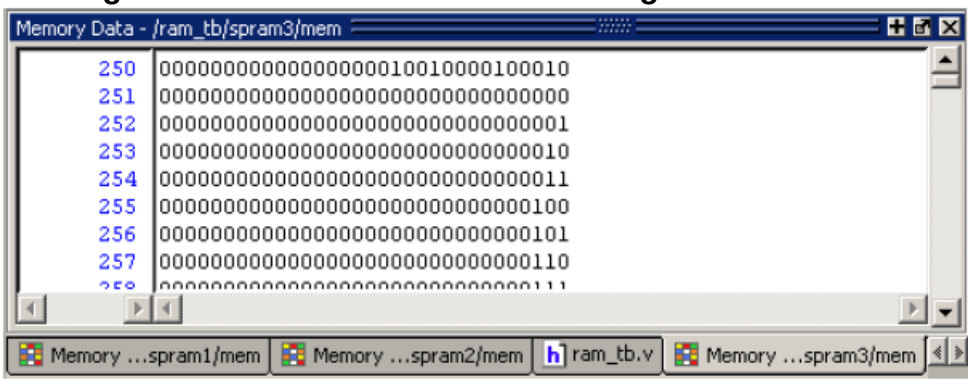
   E. Data Load 는 Increment 로 변경합니다.

   F. Fill Data field 의 incrementing data 를 0 으로 설정합니다.

   G. OK 를 선택합니다.

   H. 251 번 라인부터 incrementing data 를 확인 할 수 있습니다.

Figure 11-15. Data Increments Starting at Address 251



Memory Data Window 창 하나를 선택하여 오른쪽 마우스를 누른 후 Close All 을

선택하여 창을 닫습니다.

Interactive Debugging Commands

Memory Data Window 는 다양한 디버깅을 목적으로 사용 할 수 있습니다.

1. Memory Instance 를 열어 display characteristics 변경

   A. Memory List Window에서 /ram_tb/dpram1/mem 을 선택합니다.

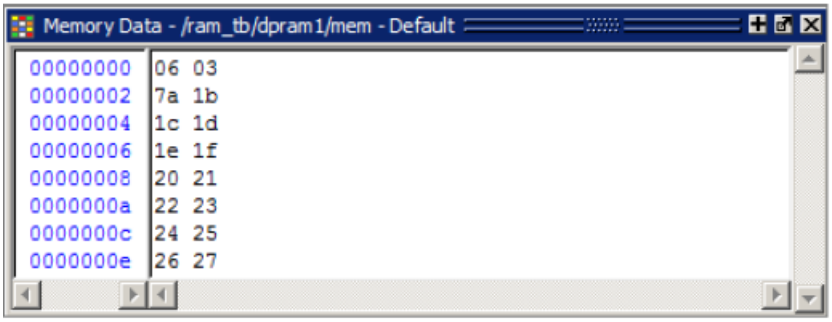
   B. 오른쪽 마우스를 눌러 Properties 를 선택합니다.

   C. Address 와 Data 의 radix 를 Hexadecimal 로 변경합니다.

   D. Words per line 을 2로 수정합니다.

   E. OK 버튼을 클릭하며 Memory 정보가 아래와 같이 보여집니다.

Figure 11-16. Original Memory Content

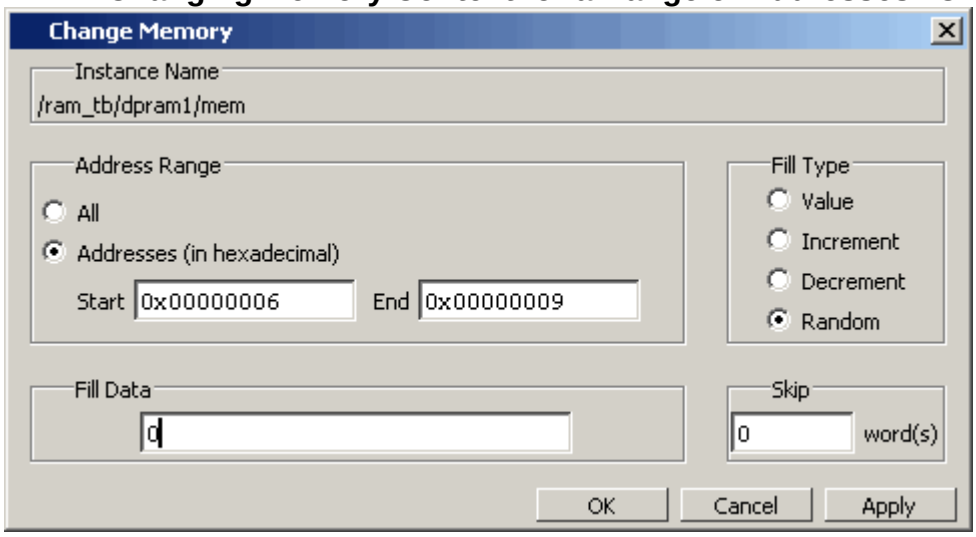


2. Fill 패턴을 이용하여 특정 Memory Address 범위 Data 초기화 하기

   A. /ram_tb/dpram1/mem 을 선택하여 Memory 윈도우에 보여지게 합니다.

   B. 오른쪽 버튼을 클릭하여 Change 버튼을 클릭하면 아래의 창이 열립니다.

Figure 11-17. Changing Memory Content for a Range of AddressesOK**



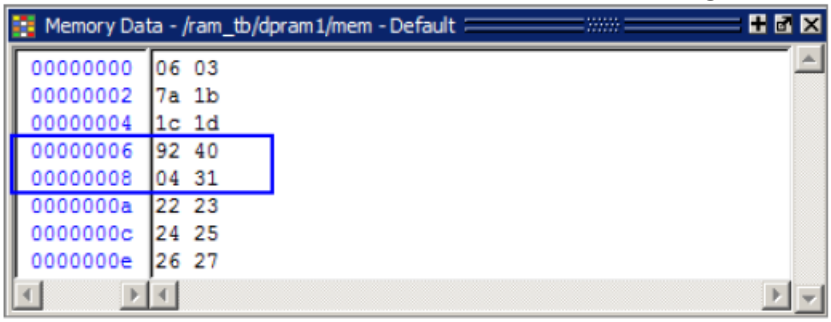
C. Address 를 선택하고 위의 그림과 같이 값을 입력합니다.

D. Fill Type 을 Random 을 선택합니다.

E. Fill Data 에 0 을 입력하고 OK 버튼을 클릭합니다.

아래의 창과 같이 Data 가 채워지는 것을 확인 할 수 있습니다.

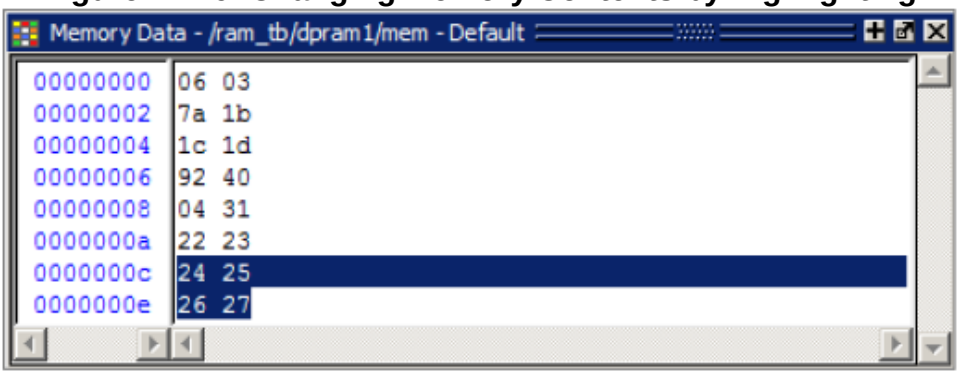
Figure 11-18. Random Content Generated for a Range of Addresses



4. 원하는 Address 의 Data 를 변경하는 거도 가능합니다.

A. 위의 그림을 보면 0x0000000c 부터 0x0000000e 까지의 데이터를 바꾸어 볼 것입니다.

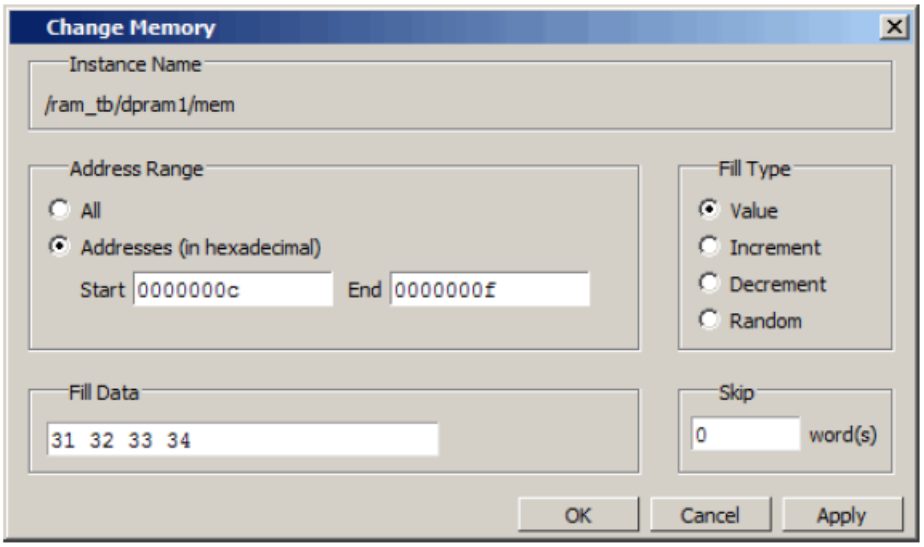
Figure 11-19. Changing Memory Contents by Highlighting



B. 마우스 오른쪽 버튼을 클릭하여 Change 를 클릭합니다.

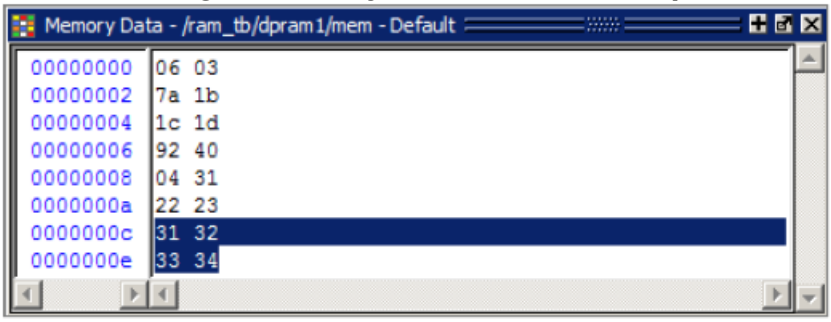
C. 창이 열리면 아래와 같이 입력을 합니다.

Figure 11-20. Entering Data to ChangeOK**



Address 부분에 000000c 와 0000000f 를 입력하고 Fill Type 엔 Value 를 체크하고 Fill Data 에는 값을 변경할 31 32 33 34을 입력하고 OK 버튼을 클릭합니다.

Figure 11-21. Changed Memory Contents for the Specified Addresses



D. 위와 같이 값이 변경되어짐을 확인 할 수 있습니다.

Chapter 12
Analyzing Performance With The Profiler

이번 챕터에서 사용할 Profiler 기능은 시뮬레이션을 진행하는 동안 Design 코드의 각 부분뿐만 아니라 메모리의 양을 각각의 Function 및 Instances 별 비율을 보여줍니다. 이 정보를 사용하면 시뮬레이션 동안 발생하는 병목 현상을 식별하고 Design 코드를 최적화하여 시뮬레이션 시간을 줄일 수 있습니다.

이번 챕터에서는 Profiler 기능을 사용하는 방법과 주요 정보보기 명령을 사용하여 병목 현상을 식별하는 방법을 설명합니다.

이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

Verilog – *<install_dir>/examples/tutorials/verilog/profiler*
VHDL – *<install_dir>/examples/tutorials/vhdl/ profiler*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Profiling Performance and Memory Use 와 Tcl and Macros 를 보시면 더욱 자세히 알 수 있습니다.

Compile and Load the Design

1. 새로운 디렉토리를 만들고 tutorial file 복사하기

   이번 챕터를 진행하기 위한 directory 를 만듭니다.

   그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. ModelSim/Questa 를 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 ModelSim/Questa 를 구동합니다.

   B. File>change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. Library 만들고 Compile 하기

   A. Verilog 의 경우
      i. vlib work

         vlog *.v 를 입력합니다.

B. VHDL 의 경우

i. vlib work

vcom -93 sm.vhd sm_seq.vhd sm_sram.vhd test_sm.vhd 를 입력합니다.

4. Design 최적화 하기

A. 아래의 Command 를 입력합니다.

i. vopt +acc test_sm –o test_sm_opt 를 입력합니다.

최적화된 test_sm_opt 라는 최적화된 library 가 만들어 집니다.

5. Design 로드하기

A. 아래의 Command 를 입력합니다.

i. vsim test_sm_opt

Run the Simulation

1. Profiler 기능 활성화 하기

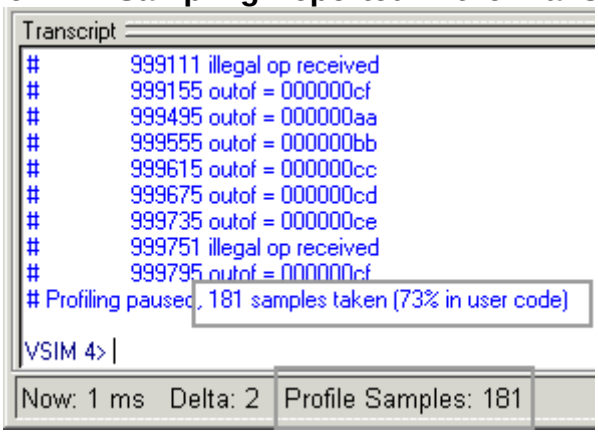
A. Tools>Profile>Performance 를 선택하거나 Performance Profiling 아이콘을 클릭합니다. 

2. Simulation 진행하기

A. Transcript 윈도우에 run 1ms 를 입력합니다.

Simulation 이 진행이 되면서 Transcript 윈도우에 아래와 같은 메시지들이 보여집니다.

Figure 12-1. Sampling Reported in the Transcript



View Performance Data in Profile Windows

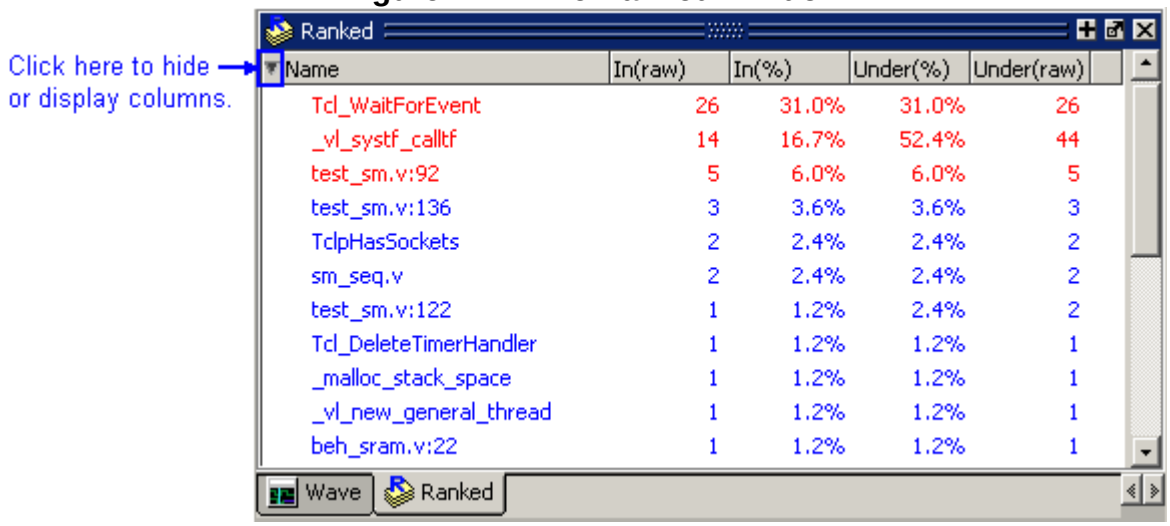
Profile data 는 Ranked, Call Tree, Structural, Design Unit 이라는 4개의 윈도우를 통해 확인 할 수 있습니다. 추가적으로 Profile detail 이라는 창을 제공하고 있습니다.

1. Profile Data 확인하기

   A. View>Profiling>Ranked Profile 를 선택합니다.

   Ranked 윈도우는 각 Function, Instance 별 Profile 결과를 아래의 그림과 같이 보여줍니다.

Figure 12-2. The Ranked Window



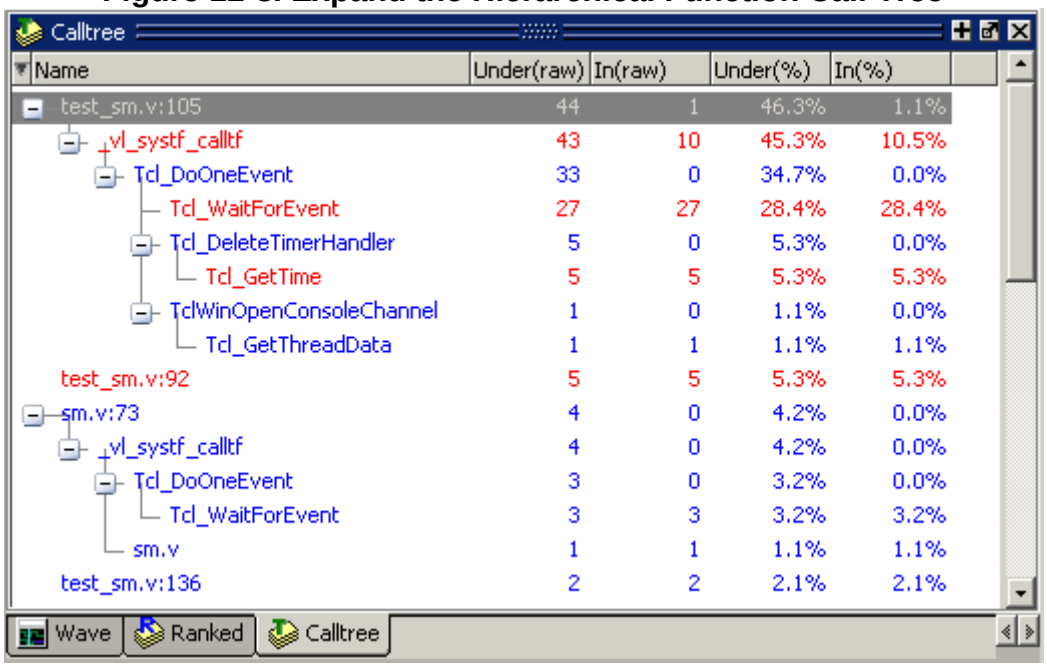
결과를 Column 을 클릭함으로써 내림차순이나 오름차순으로 정렬을 할 수가 있고, 5% 미안의 Instance 들은 푸른색으로 표시가 됩니다.

2. Hierarchical, function-call Profile Data 확인하기

A. View>Profiling>Call Tree Profile 를 선택합니다.

B. Call tree 윈도우에서 마우스 오른쪽 버튼을 클릭하여 Expand All 을 선택합니다. Function Call 의 Hierarchy 정보가 Under% Column 에 따라 보여집니다.

Figure 12-3. Expand the Hierarchical Function Call Tree



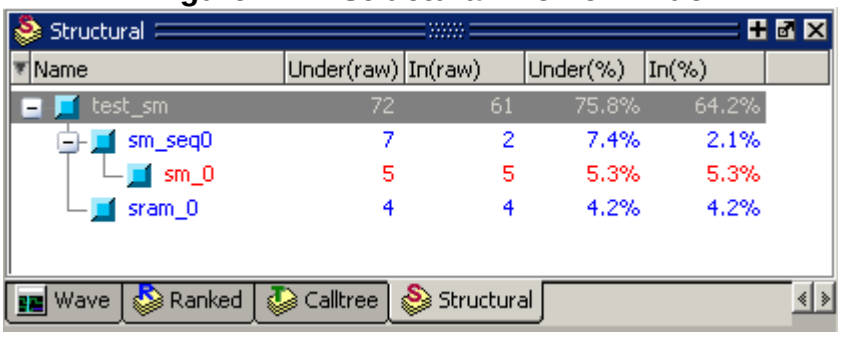
3. Hierarchy format 에 맞추어 Instance Profile Data 확인하기

A. View>Profiling>Structure Profile 을 선택합니다.

B. Structure Profile 윈도우에서 특정 Instance 를 선택하고 마우스 오른쪽 버튼을 클릭하여 Expand All 을 선택합니다.

Structure 윈도우에는 Performance sample 이 Category 형태로 분류되어 아래의 그림처럼 보여집니다.

Figure 12-4. Structural Profile Window

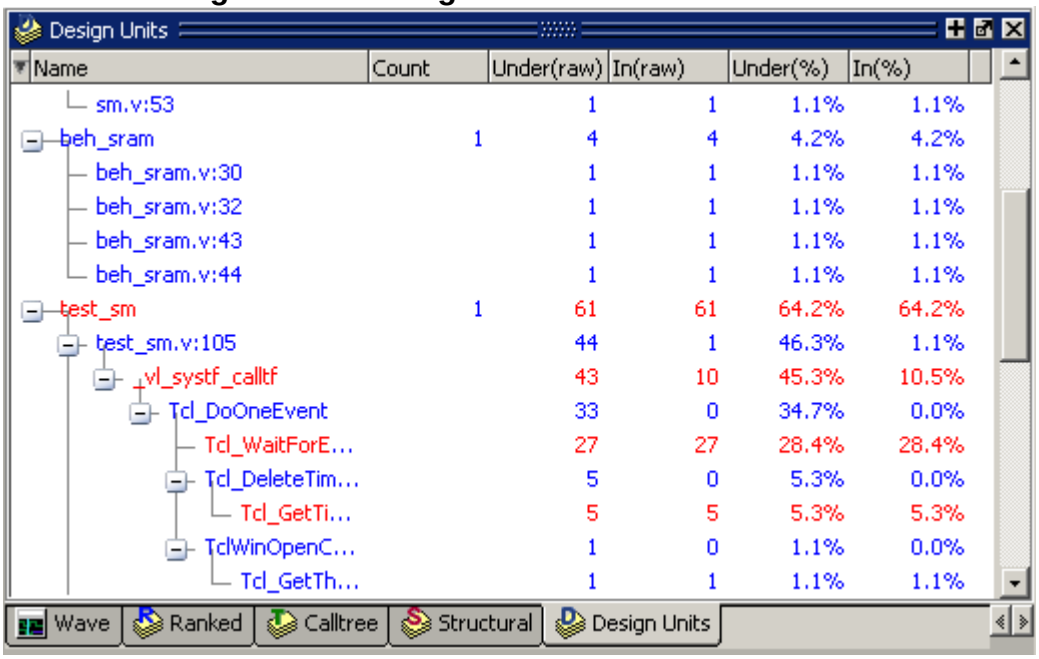


4. Design Unit 에 의한 Profile Data 보기

A. View>Profiling>Design Unit Profile 을 선택합니다.

Structure 윈도우와 비슷해 보이지만, Design Unit 에 따라 Hierarchy 정보를 가지고 있는 형태로 보여집니다.

Figure 12-5. Design Unit Performance Profile

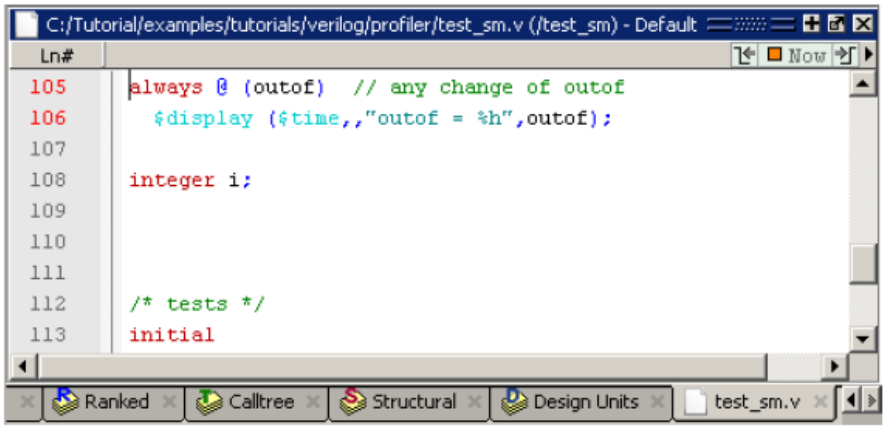


Profile 윈도우들은 Source 윈도우들과 Cross Proving 기능을 제공하고 있습니다. Profile 윈도우에서 Function, Instance 를 더블 클릭하는 것만으로 그 부분에 해당하는 Source 윈도우로 바로 이동이 됩니다.

Veilog – Design Units Profile 윈도우에서 test_sm.v:105 를 선택하고 더블 클릭합니다.
VHDL - Design Units Profile 윈도우에서 test_sm.vhd:201 를 선택하고 더블 클릭합니다.

더블 클릭을 하면 아래의 그림처럼 Source 윈도우가 보여집니다.

Figure 12-6. Source Window Shows Line from Profile Data



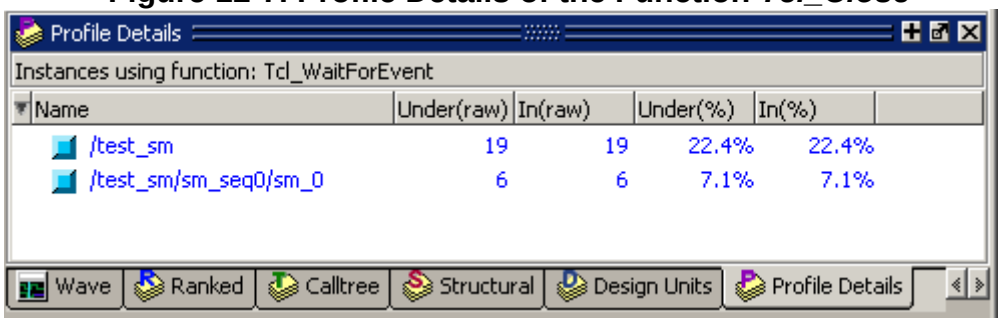
View Profile Details

Profile 정보에 대해 보다 자세한 정보를 Details 윈도우에서 볼 수 있습니다. Ranked 혹은 Call Tree 윈도우에서 특정 Function 클릭하거나 마우스 오른쪽 버튼을 클릭하여 Function Usage 를 선택합니다. Function Usage 를 선택하면 선택한 instance 의 자세한 정보가 Profile Details 윈도우에 보여집니다.

1. Call Tree 윈도우에서 Details 윈도우 열기

   A. Tcl_WaitForEvent Function 을 선택하고 마우스 오른쪽 버튼을 눌러 Function Usage 를 선택합니다. Tcl_WaitForEvent Function 을 사용한 모든 Instance 가 Profile Details 윈도우에 나타납니다.

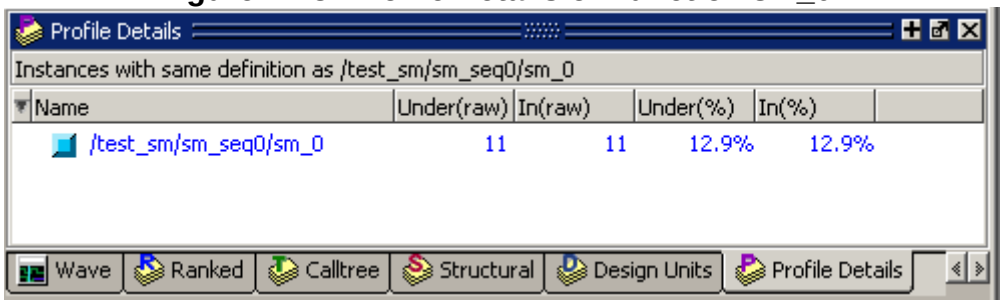
Figure 12-7. Profile Details of the Function *Tcl_Close*



2. Structure 윈도우에서 Details 윈도우 열기

   A. Structure 윈도우에서 test_sm 을 선택하고 Expand All 을 선택합니다. 위에서와 마찬가지로 자세한 정보가 보여집니다.

Figure 12-8. Profile Details of Function *sm_0*



Filtering the Data

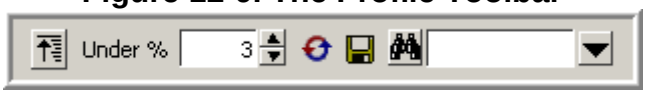
마지막 단계로, Toolbar 의 Profilter 를 사용하면 전체 Simulation 중에서 3% 미만을 차지하는 task 정보는 Filtering 할 수 있습니다.

1. 3% 미만의 Line 찾기

A. Call Tree 윈도우를 엽니다.

B. Toolbar 의 field 의 under 부분을 3 으로 입력합니다.

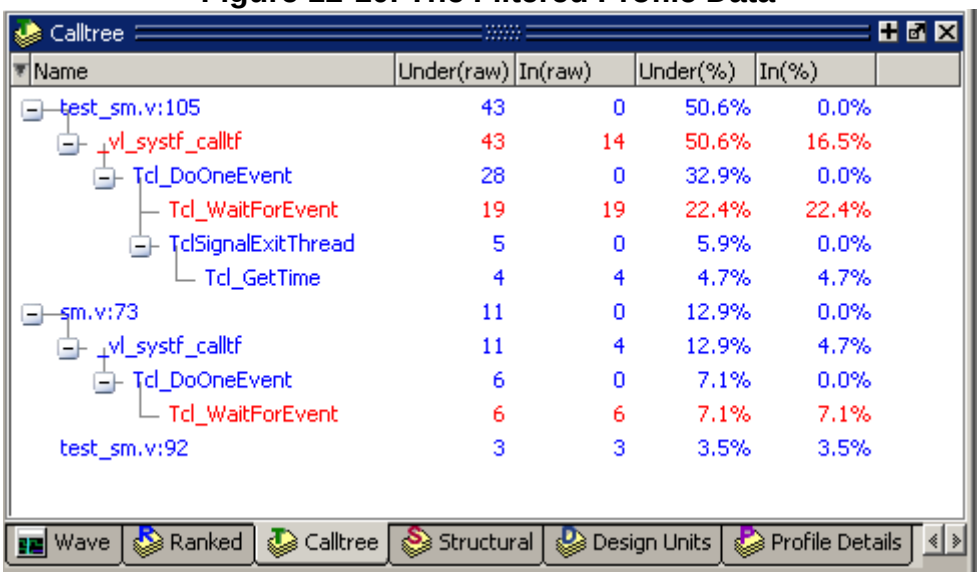
Figure 12-9. The Profile Toolbar



C. Refresh Profile Data 버턴을 클릭합니다. 

아래와 같이 3% 이상을 가지고 있는 Line 들이 아래와 같이 보여집니다.

Figure 12-10. The Filtered Profile Data



Creating a Performance Profile Report

1. Profile Data 중 Call Tree 리포트 만들기

   A. Tools>Profile>Profile Report 를 선택합니다.

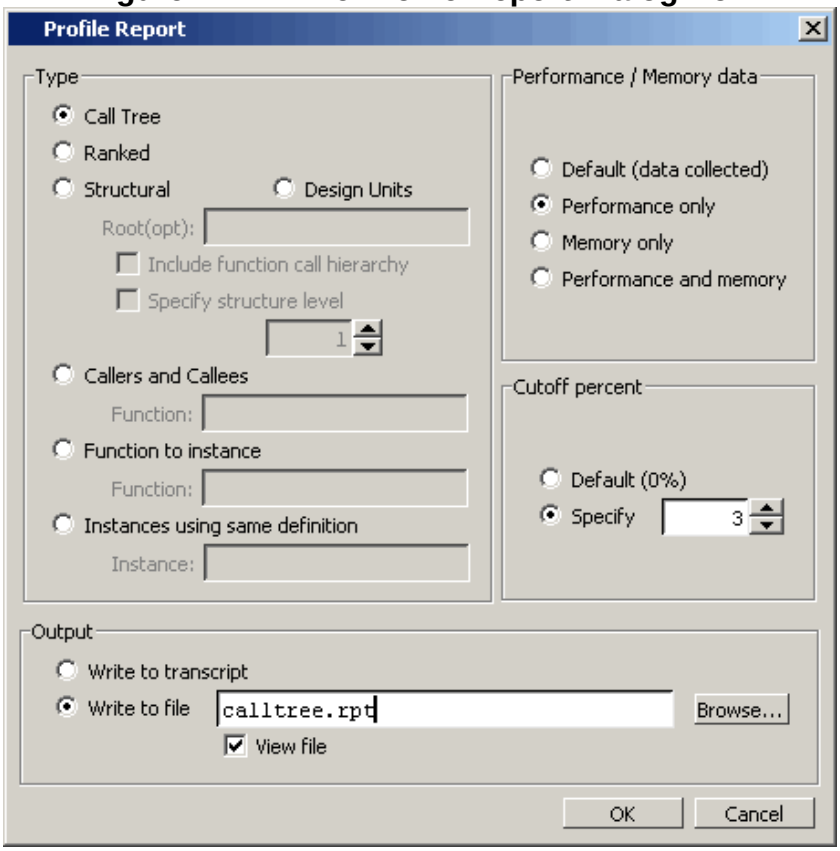
   B. Profile Report 창이 열리면 Type 에 Call Tree 를 선택합니다.

   Performance / Memory data 블록엔 Performance only 를 선택합니다.

   CutOff percent 에는 3 을 입력합니다.

   Write to file 부분엔 calltree.rpt 를 입력합니다.

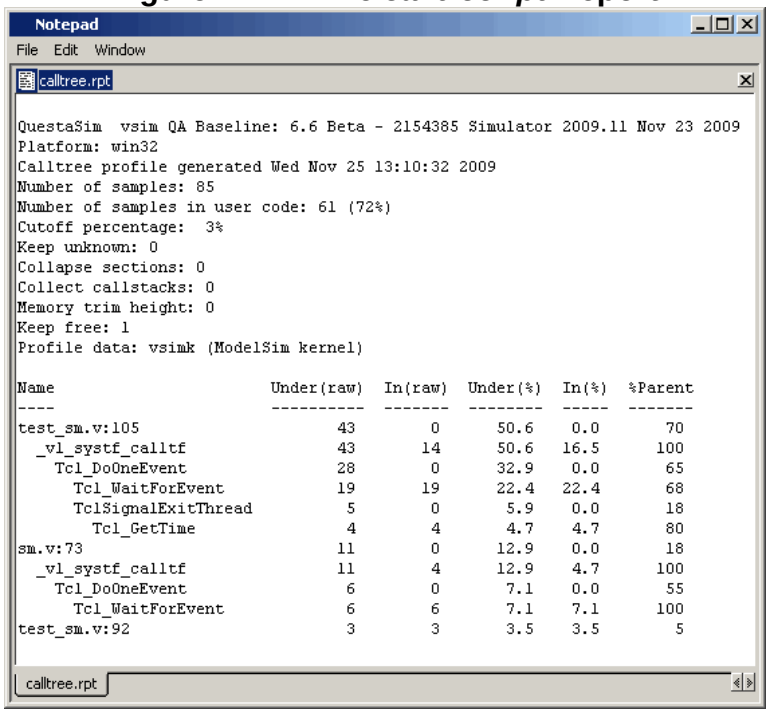
Figure 12-11. The Profile Report Dialog Box



위와 같이 입력이 끝났으면 OK 를 클릭합니다.

C. 자동적으로 저장했던 calltree.rpt 파일이 아래의 그림같이 열립니다.

Figure 12-12. The *calltree.rpt* Report



Chapter 13
Simulating With Code Coverage

ModelSim/Questa 에서 제공하는 Code Coverage 는 Simulation 이 진행되는 동안에 executable statements, branches, conditions, expressions 와 같은 Code Coverage 정보를 GUI 와 Report 형식으로 유저에게 제공을 합니다.

이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

Verilog – *<install_dir>/examples/tutorials/verilog/coverage*
VHDL – *<install_dir>/examples/tutorials/vhdl/ coverage*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Code Coverage 를 보시면 더욱 자세한 정보를 알 수 있습니다.

Compile the Design

1. 새로운 디렉토리를 만들고 tutorial file 복사하기

   이번 챕터를 진행하기 위한 directory 를 만듭니다.

   그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. ModelSim/Questa 를 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 ModelSim/Questa 를 구동합니다.

   B. File>change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. Library 만들고 Compile 하기

   A. Verilog 의 경우

      i. vlib work

         vlog *.v 를 입력합니다.

   B. VHDL 의 경우

      i. vlib work

vcom -93 *.vhd 를 입력합니다.

4. Coverage 정보를 포함하는 최적화 Design 만들기

   A. 아래의 Command 를 입력합니다.

      i. vopt +cover=bcsxf test_sm –o test_sm_opt 를 입력합니다.

         최적화된 test_sm_opt 라는 최적화된 library 가 만들어 집니다.

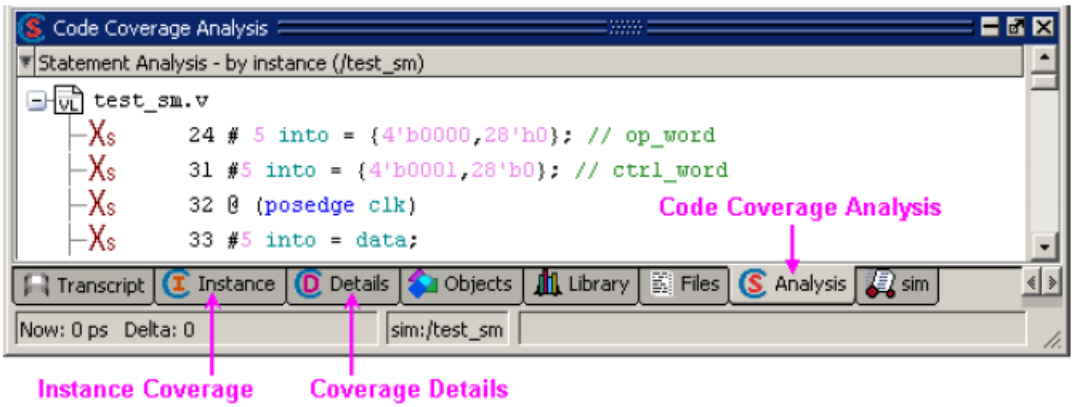
Load and Run the Design

1. Design 로드하기

   A. Transcript 윈도우에 vsim –coverage test_sm_opt 를 입력합니다.

      여러 개의 coverage 윈도우들이 함께 열리는데 그 중에 Coverage 라는 윈도우가 아래와 같이 열립니다.

Figure 13-1. Code Coverage Windows



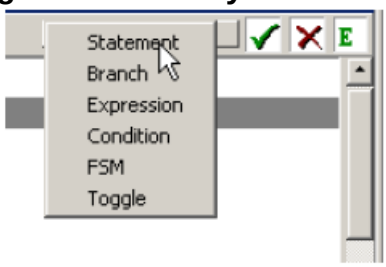
Code Coverage 를 수행하게 되면 statement, branch, condition, expression, FSM, toggle 과 같은 분석을 진행하게 되며, 분석에 따라 Code 왼쪽에 실행이 되었는지 안되었는지 Code Coverage 아이콘을 통해 확인 할 수 있습니다.

Table 13-1. Code Coverage Icons

Icon	Description/Indication
✓	All statements, branches, conditions, or expressions on a particular line have been executed
X	Multiple kinds of coverage on the line were not executed
X_T	True branch not executed (BC column)
X_F	False branch not executed (BC column)
X_C	Condition not executed (Hits column)
X_E	Expression not executed (Hits column)
X_B	Branch not executed (Hits column)
X_S	Statement not executed (Hits column)
E	Indicates a line of code to which active coverage exclusions have been applied. Every item on the line is excluded; none are hit.
E_W	Some excluded items are hit
$E_✓$	Some items are excluded, and all items not excluded are hit
$E_✗$	Some items are excluded, and some items not excluded have missing coverage
E_A	Auto exclusions have been applied to this line. Hover the cursor over the E_A and a tool tip balloon appears with the reason for exclusion.

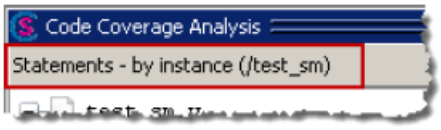
분석 툴바를 통해 선택할 수 있습니다.

Figure 13-2. Analysis Toolbar



Code Coverage Analysis window 의 제목 표시줄에서 어떤 항목을 분석하는지 확인할 수 있습니다.

Figure 13-3. Title Bar Displays Current Analysis

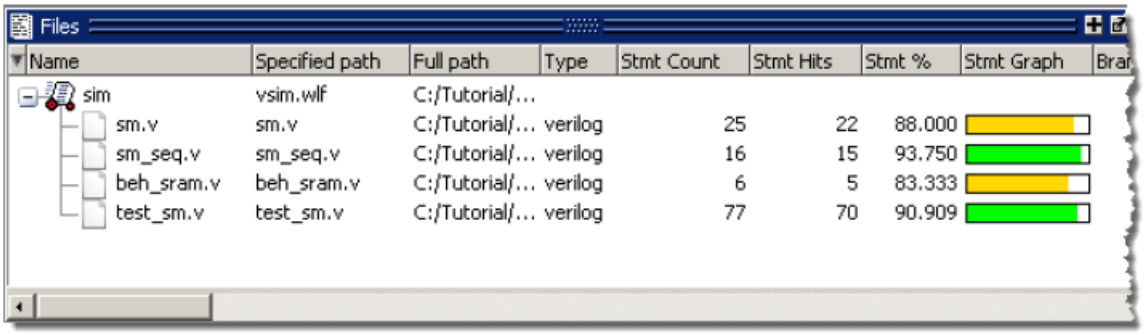


2. Simulation 진행하기

A. Transcript 윈도우에 run 1ms 를 입력합니다.

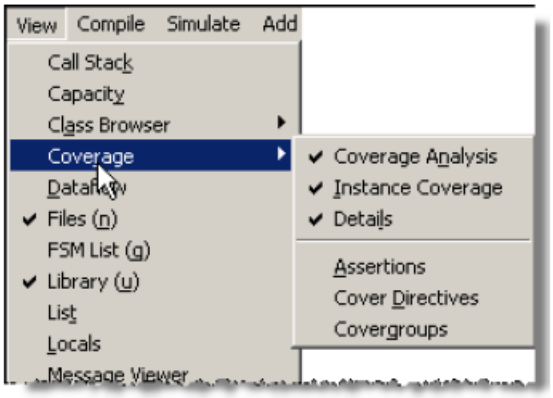
Simulation 이 진행되면 Code Coverage 관련된 윈도우에는 해당하는 정보들이 보여지게 됩니다. File 탭을 살펴보면 아래의 그림처럼 해당 파일 별 Coverage 결과가 나타남을 알 수 있습니다.

Figure 13-4. Code Coverage Columns in the Structure (sim) Window



View>Coverage 에서 Coverage 관련 윈도우들을 열 수 있습니다.

Figure 13-5. Coverage Menu



Viewing Coverage Data

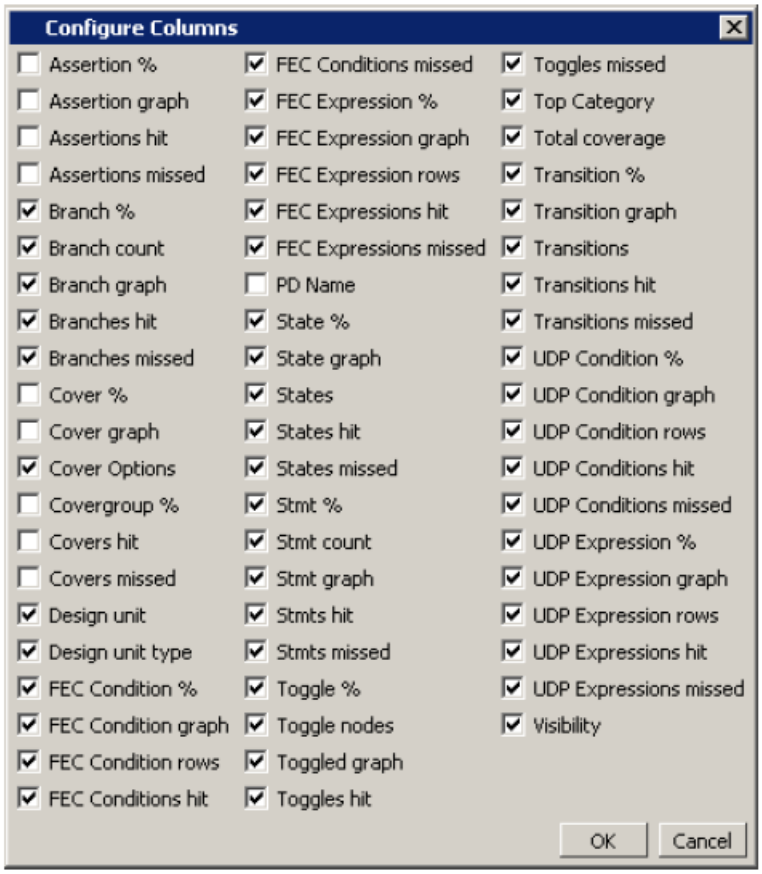
위에서 살펴본 Structure 탭에서 살펴보던 Coverage Data 를 Coverage 관련 다른 윈도우에서도 살펴 볼 수가 있습니다.

1. Structure 윈도우 살펴보기

   A. Structure(sim) 윈도우를 선택하면 위에서 보았던 Instance 별로 Coverage 정보를 볼 수가 있습니다.

   B. File 탭을 선택하면 이번에는 Design 을 구성하고 있는 File 별 Coverage 정보를 볼 수가 있습니다. 또한, Column 부분에서 마우스 오른쪽 버튼을 클릭하여 원하는 정보만 Display 할 수도 있습니다.

Figure 13-6. Right-click a Column Heading to Show Column List

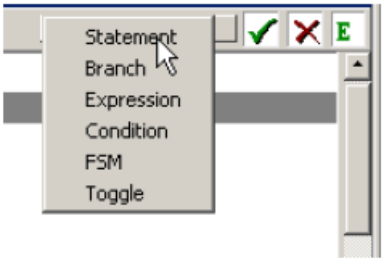


해당 Object 가 Check 가 되어 있으면 보여지고 Check 를 제거하면 Display 되지 않습니다.

2. Analysis 윈도우 살펴보기

   A. 상단에 Analysis Type 을 선택하여 원하는 Code Coverage 분석을 볼 수 있습니다.

Figure 13-7. Select Statement Analysis



   B. File 탭에서 다른 파일을 선택하여 원하는 타입으로 Code Coverage 분석 결과를 확인 할 수 있습니다.

   C. Source Window 에서 해당 줄을 표시하려면 더블 클릭합니다.

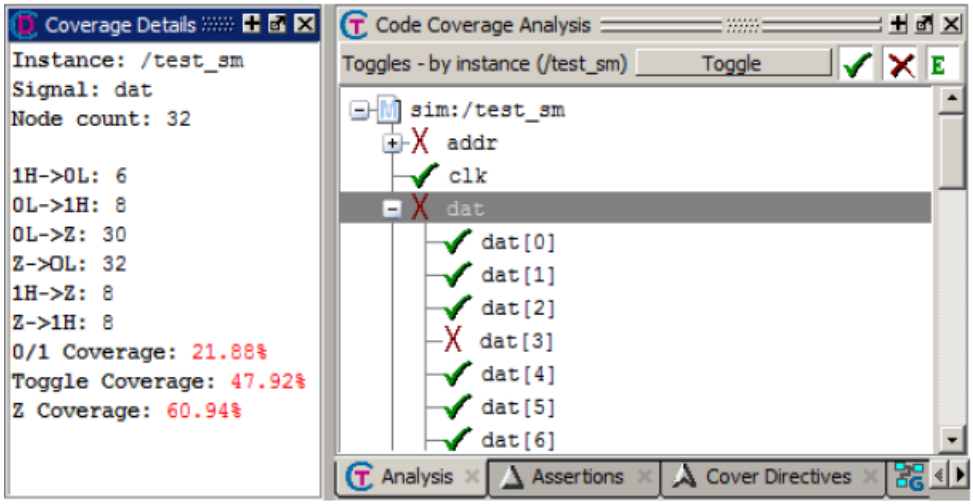
3. Coverage Details 윈도우 살펴보기.

A. 먼저 Analysis Window 에서 Coverage Type 을 Toggle 로 변경합니다.

B. View>Coverage>Details 를 선택합니다.

C. Object 를 선택하면 Details Window 를 통해 분석을 할 수 있습니다.

Figure 13-8. Coverage Details Window Undocked

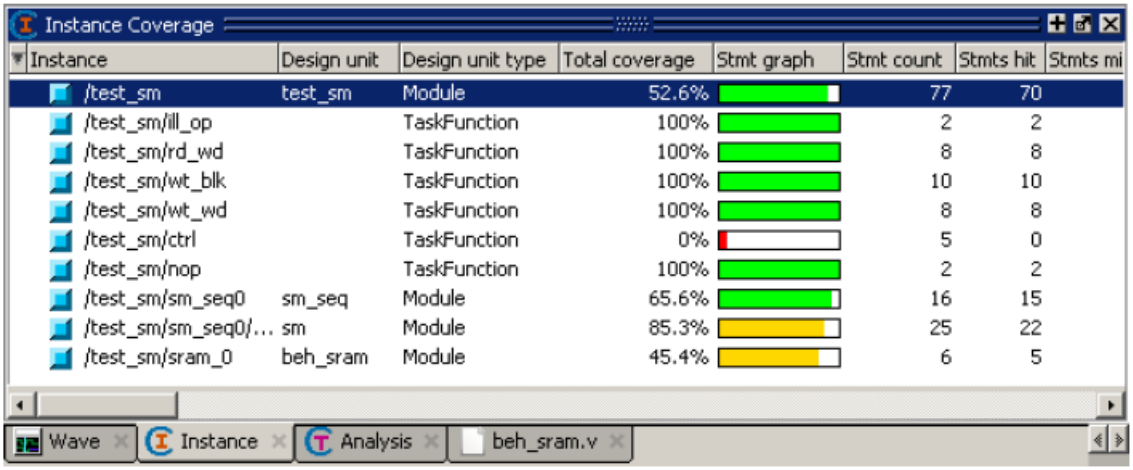


4. Instance Coverage 윈도우 살펴보기

A. Instance Coverage 윈도우를 살펴봅니다. View>Coverage>Instance Coverage 를 선택합니다.

Instance 윈도우에서는 Instance 들이 Non-hierarchical 로 보여지며 Instance 를 선택하고 더블 클릭을 하면 해당 Source 내용이 Source 윈도우를 통해 보여집니다.

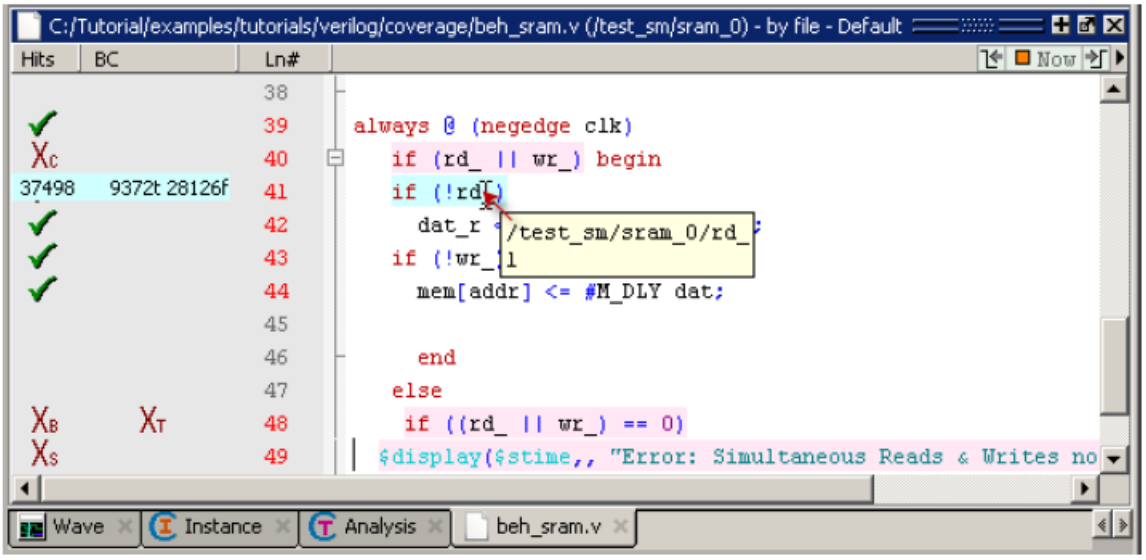
Figure 13-9. Instance Coverage Window



Coverage Statistics in the Source Window

1. Source 윈도우를 통해 beh_sram 에 대해 Coverage 정보 살펴보기
   A. File 탭에서 beh_sram.v 파일을 더블 클릭을 하면 해당 파일이 Source 윈도우에 나타납니다.
   B. 해당 파일을 아래의 그림처럼 스크롤 합니다.

Figure 13-10. Coverage Statistics in the Source Window



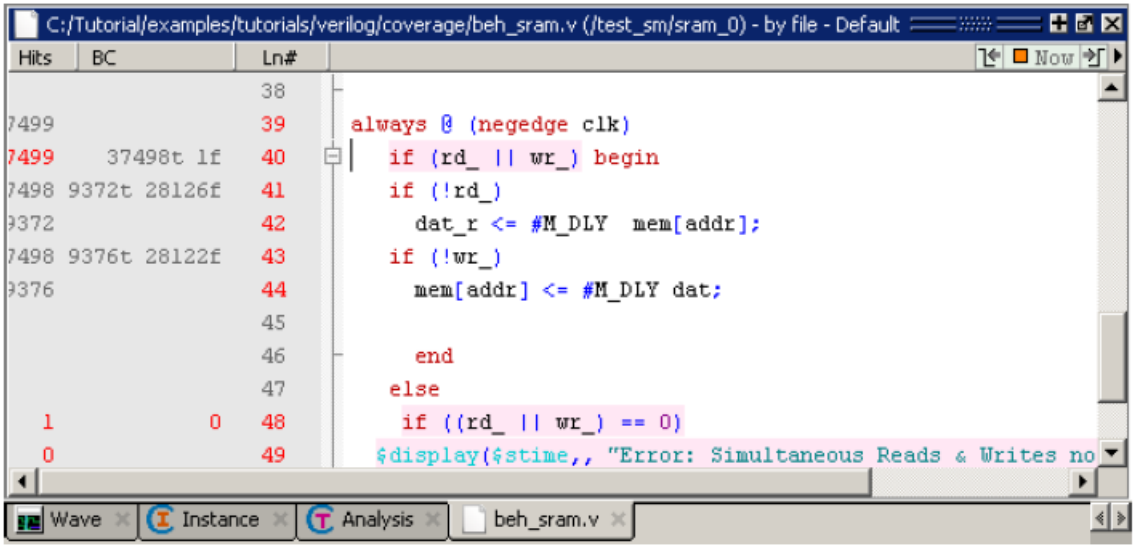
Source 윈도우를 살펴보면 Line 과 Source 내용 외에 Hits 와 BC Column 이 보입니다. 그 내용의 의미를 아래와 같습니다.

Table 13-2. Coverage Icons in the Source Window

Icon	Description
녹색 체크 마크	한번 이상 실행되어진 라인
빨간 X	한번도 실행되지 않은 라인
녹색 E	Coverage 체크를 하지 않는 라인
빨간 Xt 혹은 Xf	True 혹은 False 로 실행되지 않은 Branch 구문

C. Tools>Code coverage>Show coverage numbers 를 선택합니다.
그러면, Source 윈도우에서 해당하는 Coverage 들의 실행된 숫자가 표시됩니다.

Figure 13-11. Coverage Numbers Shown by Hovering the Mouse Pointer



Tools>Code coverage>Show coverage numbers 를 다시 선택함으로써 정보를 안보이게 할 수도 있습니다.

Toggle Statistics in the Objects Window

Toggle Coverage 는 logic node 들의 transition 을 Check 합니다. 이번 Chapter 를 진행하면서 앞서서 vlog, vcom command 로 compile 을 진행할 때, –cover x 옵션을 살펴보았을 것입니다. Toggle Coverage 에 대해서는 User Manual 일 Toggle Coverage 부분을 보시면 더욱 자세히 알 수 있습니다.

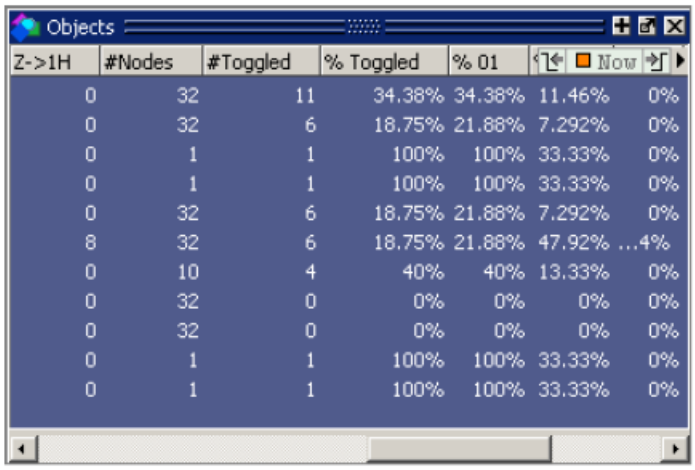
1. Objects 윈도우에서 toggle coverage data 보기

   A. Structure (sim) 탭에서 test_sm 을 선택합니다.

   B. 만약 Objects 윈도우가 열려있지 않으면 메뉴에서 **View > Objects** 를 선택하면 됩니다.

      Toggle Coverage SET 을 다 살펴보기 위해서는 column 에서 마우스 오른쪽 버튼을 클릭하여 **Show All Columns** 를 선택하면 됩니다.

Figure 13-12. Toggle Coverage in the Objects Window

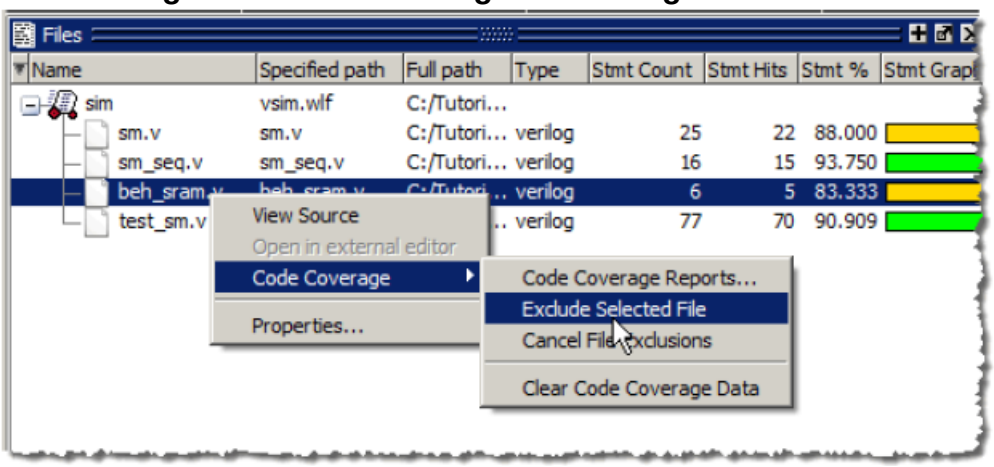


Excluding Lines and Files from Coverage Statistics

Coverage 통계에서 exclude Line 혹은 File 을 지정 할 수 있습니다. GUI 를 이용하거나 TCL file 을 이용하여 설정이 가능합니다. 보다 자세한 내용은 User's Manual 의 Coverage Exclusions 부분을 살펴 보시면 보다 자세한 내용을 보 실수 있습니다.

1. Code Coverage Analysis 윈도우에서 Statement line 제외하기
   A. Statement Analysis view 에서 line 을 선택하고 pop-up 메뉴에서 **Exclude Selected** 를 선택합니다.
2. Exclude statement 취소하기
   A. 마우스 오른쪽 버튼을 클릭하여 **Cancel Selected Exclusions** 를 선택합니다.
3. File 단위로 Exclude 하기
   A. Files 윈도우에서 sm.v(VHDL 인 경우 sm.vhd) 파일을 선택합니다.
   B. 마우스 오른쪽 버튼을 클릭하여 **Code Coverage > Exclude Selected File** 을 선택합니다.

Figure 13-13. Excluding a File Using GUI Menus



Exclude 를 취소 할 경우 마찬가지로 오른쪽 마우스 버튼을 클릭하여 **Code Coverage > Cancel File Exclusions** 를 선택합니다.

Creating Code Coverage Reports

Coverage 결과는 GUI 혹은 command 를 통해서 TEXT 혹은 HTML 형태로 Report 할 수 있습니다.

아래의 방법 중에 하나를 선택해서 진행하면 GUI 메뉴를 통해서 Textual coverage report 를 만들 수 있습니다.

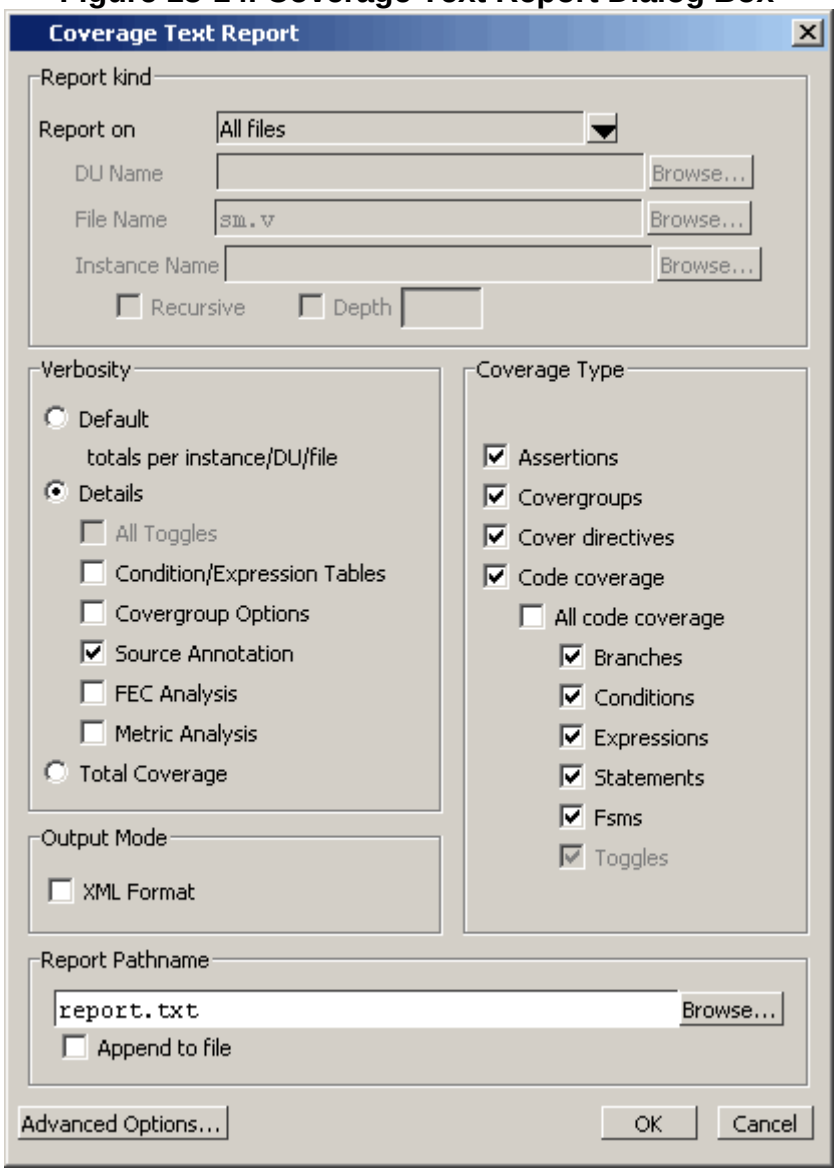
- 메뉴 바에서 Tools > Coverage Report > Text 를 선택합니다.
- sim 혹은 Files 윈도우에서 마우스 오른쪽 버튼을 클릭하면 나오는 pop-up 메뉴에서 Code Coverage > Code Coverage Report 를 선택합니다.
- Instance Coverage 윈도우에서 마우스 오른쪽 버튼을 클릭하고, pop-up 메뉴에서 Instance Coverage > Code coverage report 를 선택합니다.

이와 같은 방법으로 진행하면 아래와 같이 Coverage Text Report Dialog 창이 나타납니다. 이 윈도우를 통해 아래와 같은 요소들을 선택할 수 있습니다.

- All files
- All instances
- All design units
- Specified design unit
- Specified instance

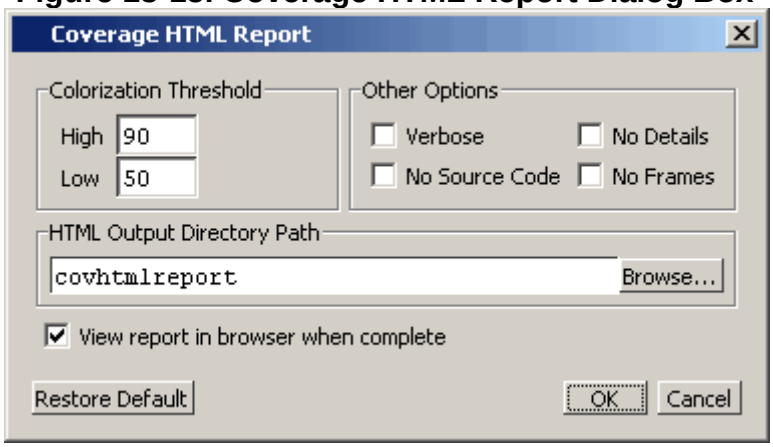
- Specified source file

Figure 13-14. Coverage Text Report Dialog Box



원하는 요소들을 선택하고 OK 버튼을 클릭하면 report.txt 파일이 만들어 집니다. HTML 형태의 Report 를 원하면 메인 메뉴에서 **Tools > Coverage Report > HTML** 를 선택하면 아래와 같은 창이 열립니다.

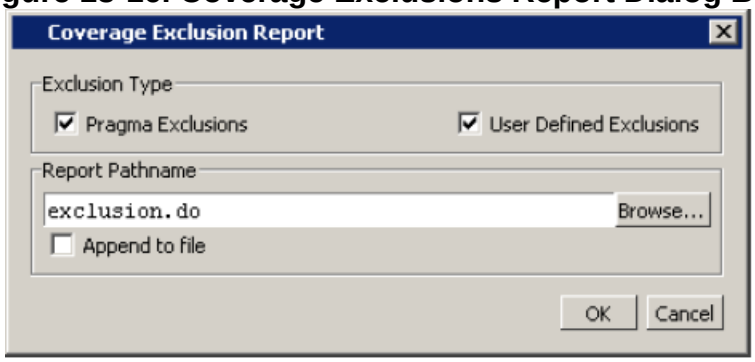
Figure 13-15. Coverage HTML Report Dialog Box



일반적으로는 coverage report 라는 command 를 이용하여 text 형태의 report 를 만들 수 있고, –html 이라는 옵션을 추가적으로 사용하면 html 형태의 report 파일을 만들 수 있습니다.

coverage exclusions report 를 생성 하기 위해서는 **Tools>Coverage Report>Exclusions** 를 통해 생성이 가능합니다.

Figure 13-16. Coverage Exclusions Report Dialog Box



Chapter 14
Debugging With PSL Assertions

Design 을 Verification 하는데 있어서 Assertions 을 이용하면 Verification 의 생산성에 많은 도움이 됩니다. Questa 는 Dynamic simulation verification 을 위해 Property Specification Language (PSL) 을 지원하고 있습니다. PSL assertions 을 이용하여 Simulation 을 진행하면 debugging 을 하는데 많은 장점이 있습니다.

이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

Verilog – *<install_dir>/examples/tutorials/psl/verilog/modeling/dram_controller*
VHDL – *<install_dir>/examples/tutorials/pslhdl/ modeling/dram_controller*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Verification With Assertions and Cover Directives를 보시면 더욱 자세한 정보를 알 수 있습니다.

Run the Design without PSL Assertions

1. 새로운 디렉토리를 만들고 tutorial file 복사하기

   이번 챕터를 진행하기 위한 directory 를 만듭니다.

   그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. Questa 을 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 Questa 를 구동합니다.

   B. File>change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. Do 파일 실행하기

   A. do run_nopls.do 라는 command 를 입력합니다.
   실행시킨 DO 파일은 아래와 같은 작업을 진행합니다.

      i. Working library 생성

      ii. Designs, Assertions files 의 compile

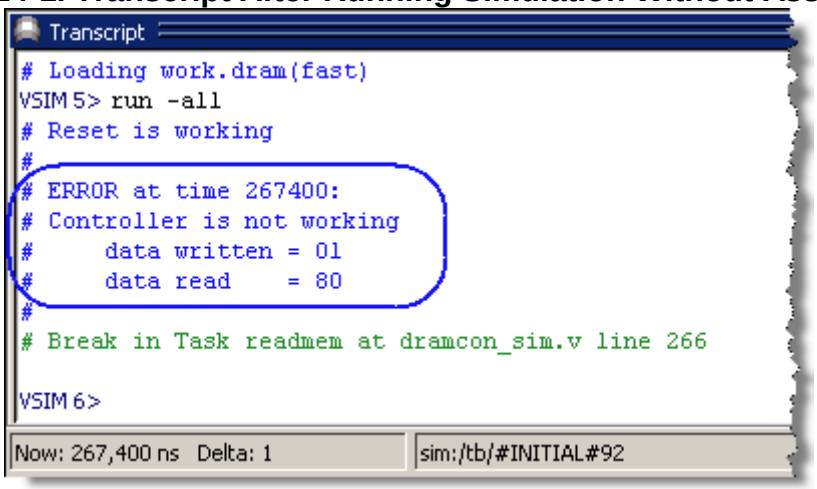
      iii. Design 최적화

      iv. Design load

v. Simulation 진행

실행 시킨 run_nopls.do 파일을 열어서 내용을 확인하면, vsim command 부분에 –voptargs="+acc" 와 –opt 라는 옵션이 사용되고 있는 것을 볼 수 있다. 이 내용은 design 을 최적화하는데 있어서 visibility 를 최대한 확보를 하라는 내용이며, -nopsl 옵션은 PSL assertions 을 무시하고 진행하는 의미입니다.

Figure 14-1. Transcript After Running Simulation Without Assertions



Simulation 이 진행되다가 위와 같은 error 메시지가 출력되면 Simulation 이 중단됩니다.

Error 를 debugging 하기 위해서는 우선 Simulation 결과인 Waveform 을 확인하고, Memory contents 를 살펴보면서 Error 를 확인해야 할 것입니다.

Debugging 시간을 단축하기 위해서는 뭔가 error 를 쉽게 찾기 위한 방법이 필요 할 것입니다.

4. Simulation 을 종료합니다.
   A. quit-sim command 를 입력하여 Simulation 을 종료합니다.

Using Assertions to Speed Debugging

Assertion 을 통한 빠른 디버깅을 위해서는 assertion failure tracking 디자인을 다시 불러와야 합니다.

1. 디자인 Load 하기

   A. Command 에 **vsim-msgmode both –asserdebug dram_opt** 를 입력합니다.

- msgmode both 는 transcript 와 WLF 파일 모두에 메시지를 출력합니다.

"dram_opt" 는 시뮬레이션을 실행 할 때 사용 할 최적화된 파일의 이름입니다.

- assertdebug 옵션은 assertion 을 실패 할 경우 디버깅을 위한 옵션입니다

2. WildcardFilter 설정 변경하기

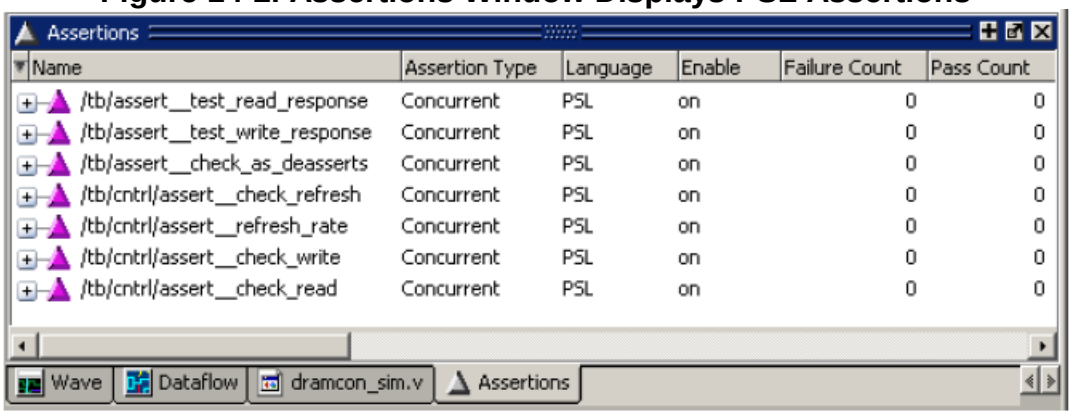
   Command 에 **set WildcardFilter "Variable Constant Generic Parameter SpecParam Memory Endpoint CellInternal ImmediateAssert"** 를 입력합니다.

   이 명령어를 사용하면 WildcardFilter 의 기본 목록인 "Assertion", "Cover", "ScVariable" 을 제거합니다. 기본적으로 디버그 환경인 시뮬레이터의 log에 Assertion, Cover Directives, SystemC Variables 가 기록됩니다.

3. Assertions window 를 통해 모든 PSL assertion 보기

   A. Command 에 view assertion 을 입력합니다. 그럼 Assertion Window 가 나타나며, 모든 PSL assertion 이 보여집니다.

Figure 14-2. Assertions Window Displays PSL Assertions

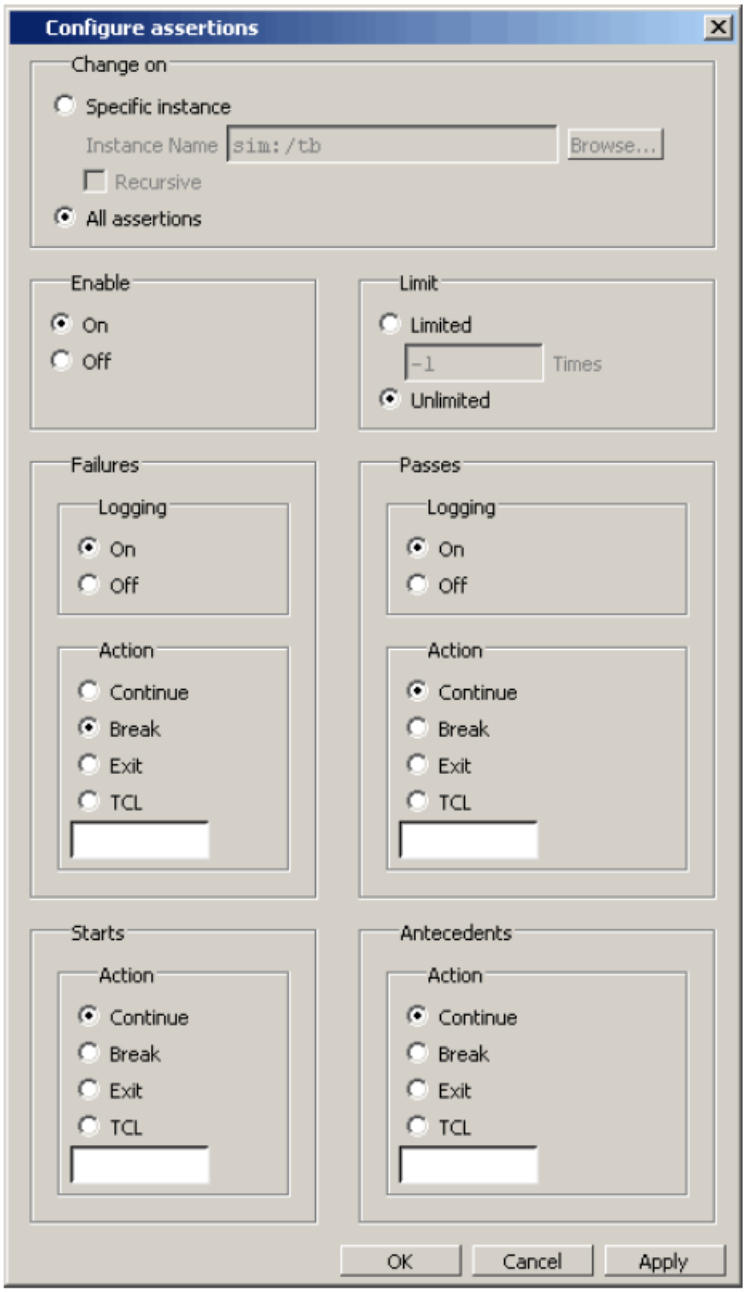


4. 모든 assertion 에서 Break 와 Failures 를 설정하기.

   A. View>Coverage>Assertion 을 통해 Assertion Window 를 활성화합니다.

   B. Assertions>Configure 를 선택합니다.

Figure 14-3. Configure Assertions Dialog Box



C. Change on 항목에서 All assertions 을 선택합니다.

D. Enable 항목에서 On 을 선택합니다.

E. Action 항목에서 Break 를 선택합니다.

F. Passes 의 Logging 항목에서 On 을 선택합니다.

G. OK 버튼을 누릅니다.

Command 에 아래의 명령어를 입력합니다.

assertion action -cond fail -exec break -r *
assertion pass -log on -r *

5. 모든 Assertion signal 을 Wave Window 에 추가.

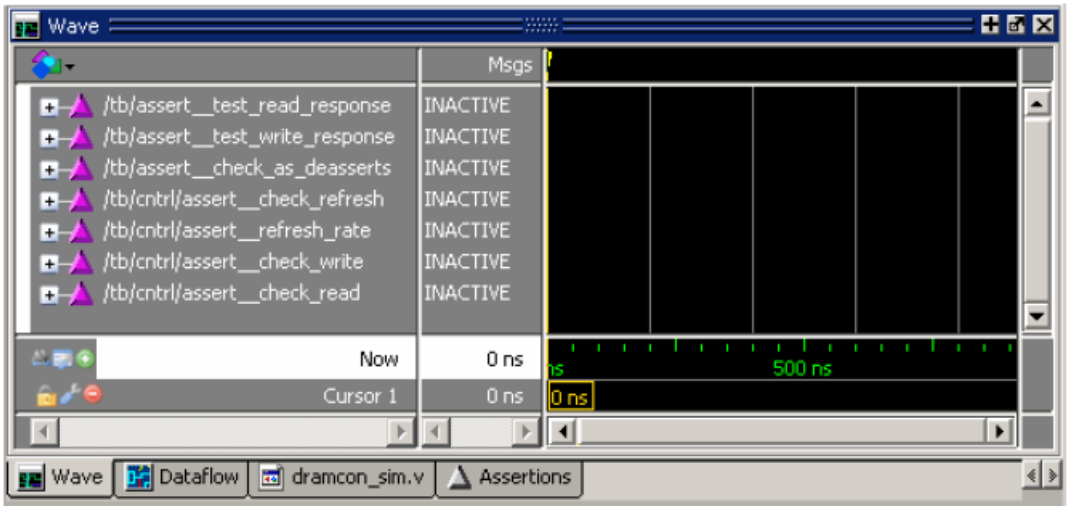
   A. Assertion Window에 있는 모든 assertions 를 선택합니다.

   B. 팝업 메뉴를 열려면 선택한 assertions 를 오른쪽 마우스로 클릭합니다.

   C. Add Wave>Selected objects 를 선택합니다.

      Wave Window 에 보라색 삼각형 아이콘과 함께 Assertion Signal 이 나타납니다.

Figure 14-4. Assertions in the Wave Window

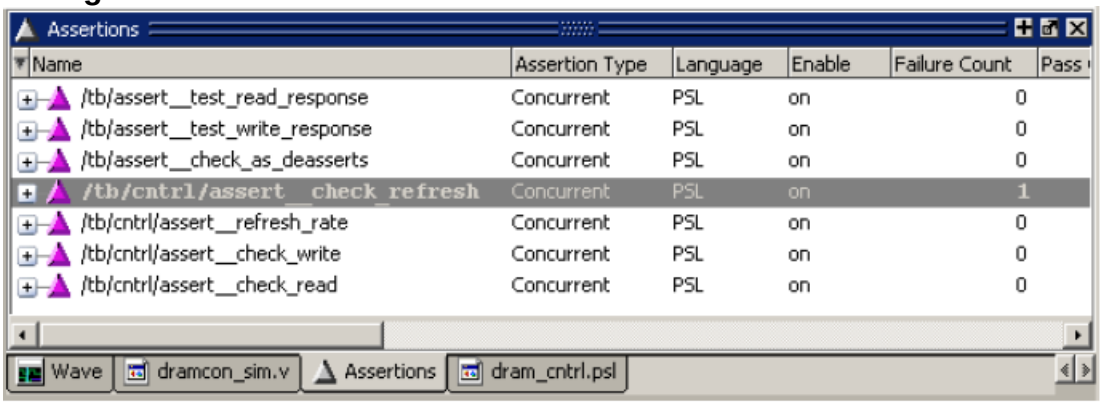


6. Simulation 진행과 결과 보기.

   A. Command 에 run –all 을 입력합니다.

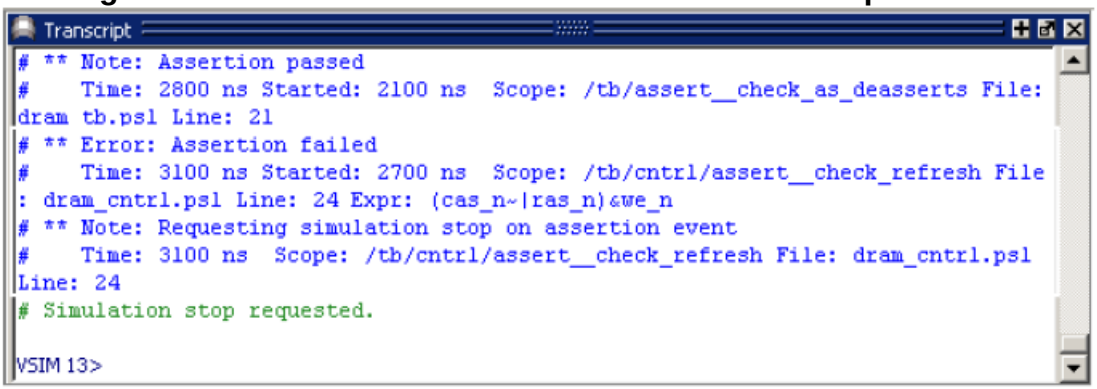
      Verilog : Transcript Window 에 dram_cntrl.psl 파일에 assert_check_refresh 의 assertion 이 3100 ns 에 failed 되었다는 정보를 보여줍니다.

Figure 14-5. Assertion Failure Indicated in the Assertions window



시뮬레이션은 그 시간에 정지했습니다. 테스트 벤치는 Assertion 을 하지 않아 267,400 ns 까지 failure 를 보고 하지 않았습니다. (80배 이상 빨리 Failure 된 Simulation 시간을 보여줍니다.)

Figure 14-6. Assertion Failure Shown in the Transcript Window



VHDL : Transcript Window 에 dram_cntrl.psl 파일에 assert_check_refresh 의 assertion 이 3800 ns 에 failed 되었다는 정보를 보여줍니다. 시뮬레이션은 그 시간에 정지했습니다. 테스트 벤치는 Assertion 을 하지 않아 246,800 ns 까지 failure 를 보고 하지 않았습니다. (60배 이상 빨리 Failure 된 Simulation 시간을 보여줍니다.)

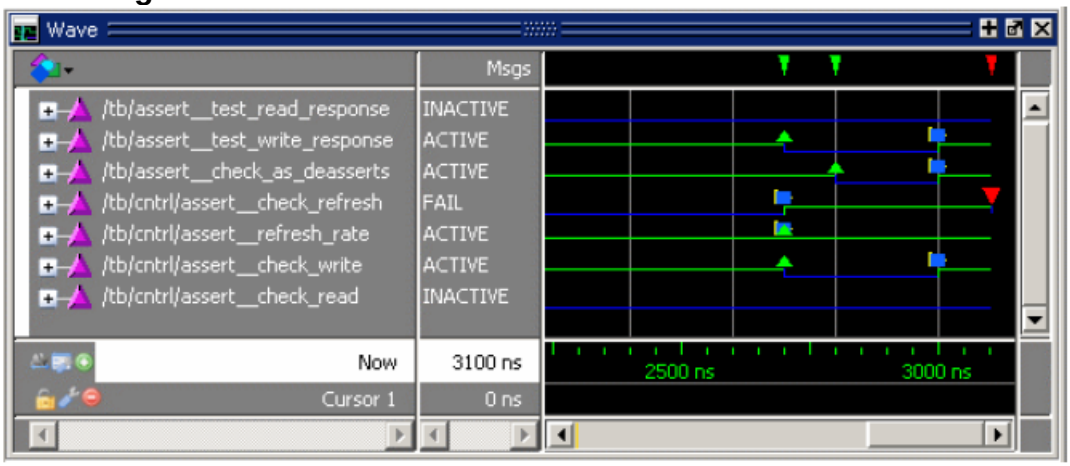
B. Source Window 에서 dram_cntrl.psl 을 클릭합니다.

Simulation 이 정지된 위치를 Source Window(dram_cntrl.psl)에서 파란색 화살표가 표시 됩니다. -24번 line 의 check_refresh 의 assertion

C. Wave 탭을 클릭하여 Wave Window 를 엽니다.

Wave Window 는 Simulation 이 Break 된 시점에서 FAIL 된 부분을 빨간색 삼각형으로 나타냅니다. 녹색 삼각형은 Pass 를 나타냅니다.

Figure 14-7. Assertion Failure Indicated in Wave Window



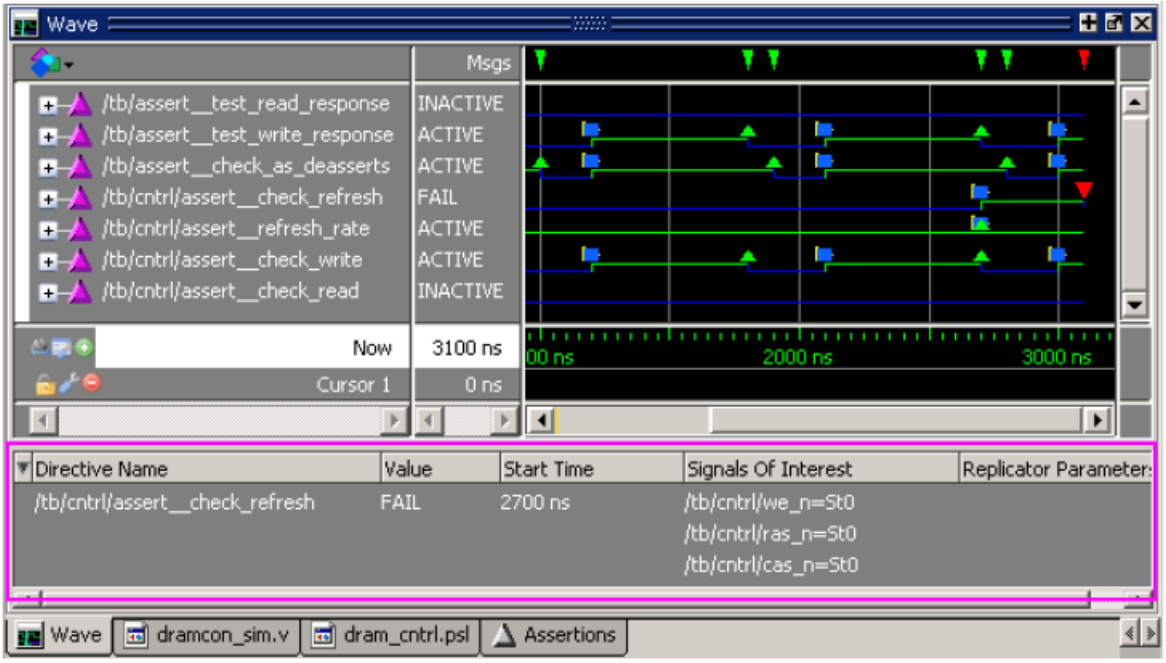
Waveform 에서 파란색 부분은 비활성 된 Assertion 을 나타냅니다. 녹색은 활성 된 Assertion 을 나타냅니다.

7. Wave window 를 통해 failure 된 Assertion 을 디버그 창을 통해 확인.
   Simulator 를 호출 하기 위해 vsim Command 를 입력할 때 –assertdebug 를 사용하여, Wave Window 를 통해 Assertion Failure 을 디버깅 할 수 있습니다.

   A. Wave>Assertion Debug 를 선택합니다.

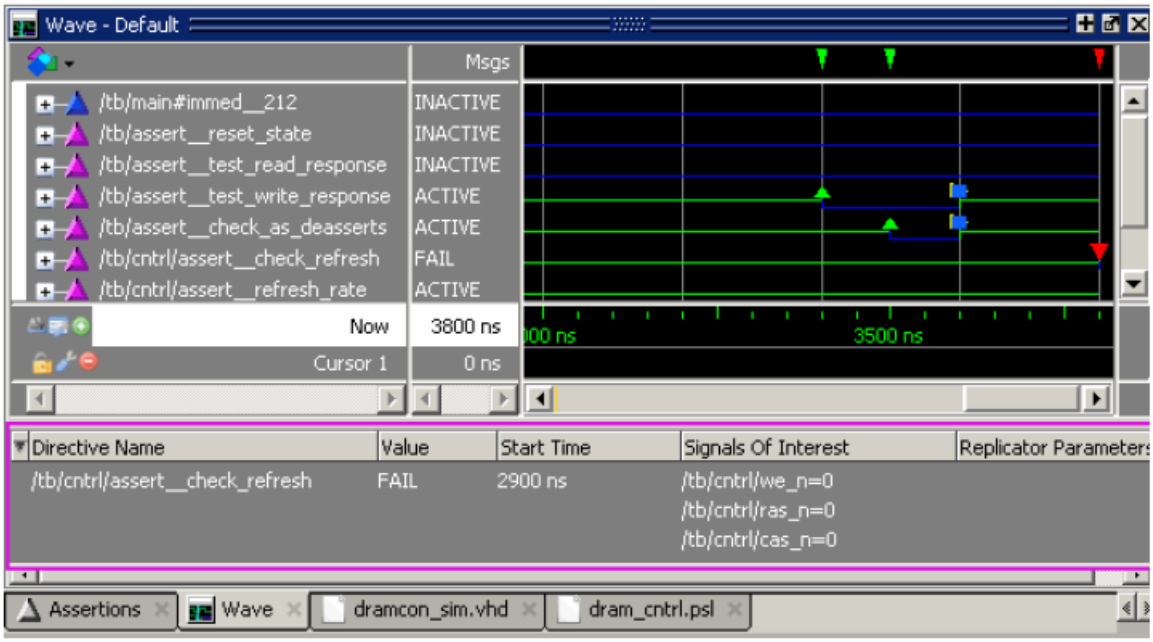
   Verilog : Wave Window 의 assertion 창을 보면 Start Time 에 2700 에서 FAIL 생겼다고 보여줍니다.

Figure 14-8. The Failed Assertion Details for Verilog Design



VHDL : Wave Window 의 assertion 창을 보면 Start Time 에 2900 에서 FAIL 생겼다고 보여줍니다.

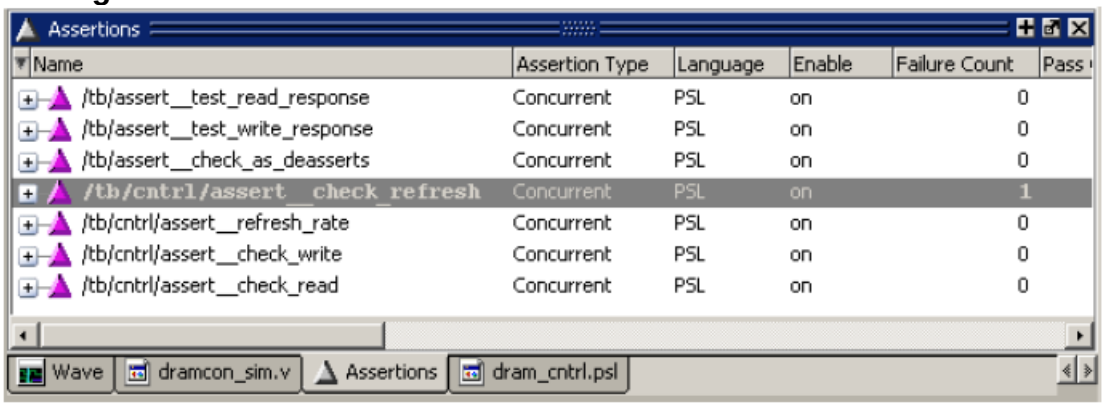
Figure 14-8. The Failed Assertion Details for VHDL Design



8. Assertions Window 를 통해 assertion failure 확인.

   A. Assertions Window 에서 assert_check_refresh 를 선택합니다. 선택하면 하이라이트로 표시 됩니다.

Figure 14-9. Assertion failure indicated in the Assertions window



Debugging the Assertion Failure

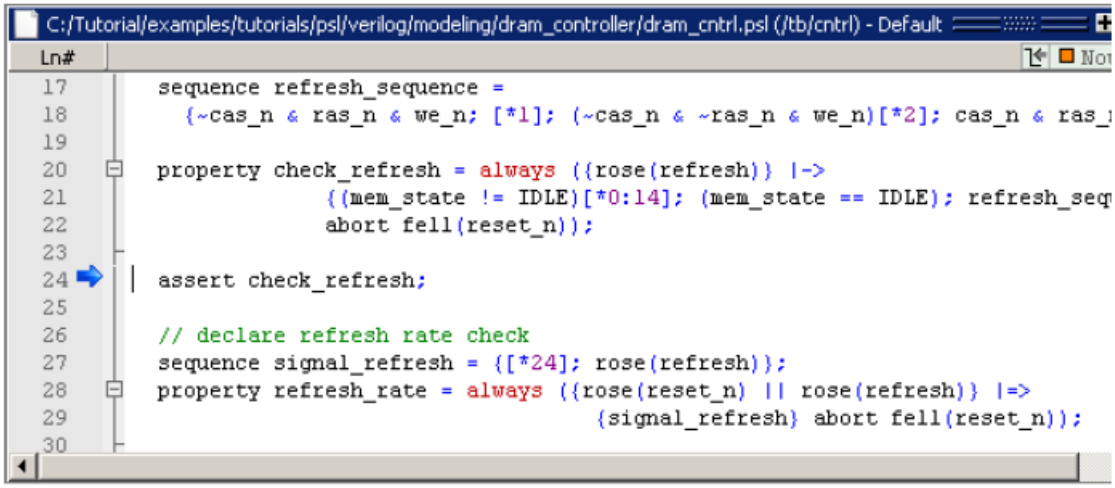
지금부터 Assertion Failure 를 Debugging 하겠습니다.

1. Assertion fail 된 source code 를 확인합니다.(dram_cntrl.psl)

   24 번 Line 을 보면 Assertion 이 Fail 된 부분에 파란색 화살표로 표시 됩니다. 이

assertion 은 20 ~ 22 번 라인까지 정의 되어 있으며, check_refresh 속성을 체크하는 것으로 구성되어 있습니다. check_refresh 속성을 보면 reflash signal 이 활성화가 되려면 memory controller state 가 IDLE 이 될 때까지 기다립니다. 읽기와 쓰기를 수행 할 때 가장 긴 부분이 14 cycles 이며, Controller 가 IDLE 일 때는 0 cycles 을 기다립니다. Controller 가 IDLE 이 되면 refresh sequence 는 다음 사이클에 시작됩니다.

Figure 14-10. Source Code for Failed Assertion



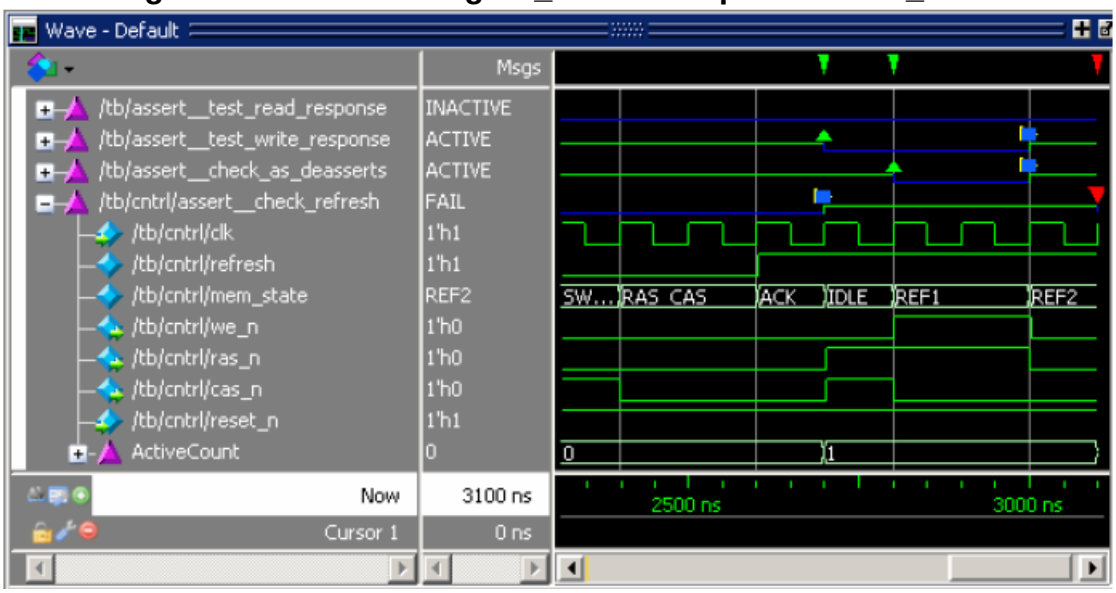
Refresh sequence 는 18번 Line 에 정의되어 있습니다. 여기에서 refresh protocol 의 핵심은 entire refresh cycle 를 위해 we_n 을 high 로 유지해야 합니다.

2. Write enable signal 인 we_n 을 Wave Window 를 통해 확인을 해 보면, REF1 과 REF2 에서 high 로 유지합니다.

   A. Wave Window 에서 assertion 을 참조하는 모든 Signal 을 보기 위해 assert_check_refresh 를 확장합니다.

   B. Zoom 과 scroll 을 통해 Wave Window 에서 we_n 과 mem_state 를 확인 합니다

Figure 14-11. Examining *we_n* With Respect to *mem_state*



we_n 이 high 일 때, mem_state 의 REF1 은 high 상태 이지만, REF2 는 low 인 것을 쉽게 볼 수 있습니다.

we_n 을 더 살펴보겠습니다.

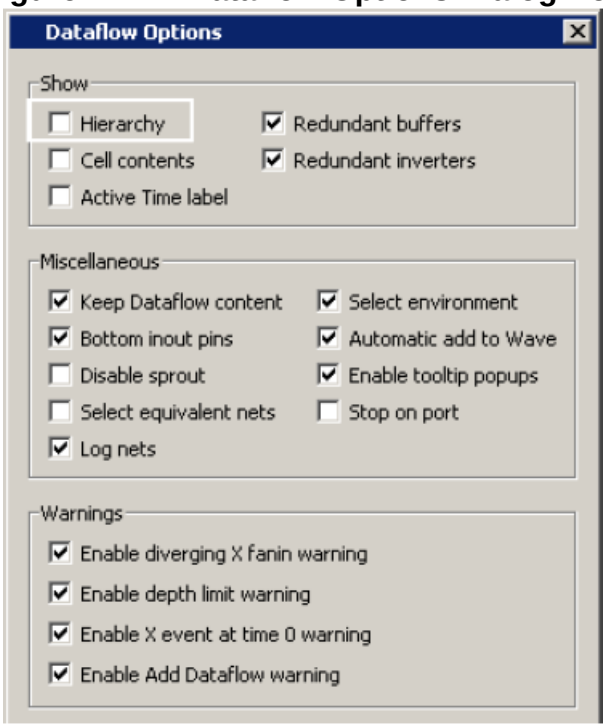
3. Dataflow 와 Source window 를 이용하여 we_n 확인.

   A. 메인 메뉴에서 View>Dataflow 를 선택하여 Dataflow Window 를 활성화 합니다.

   B. 메뉴에서 Dataflow>Dataflow Preferences>Options 을 선택하면 옵션 Dialog Box 가 열립니다. (만약에 Dataflow Window 가 Undock 이면, Dataflow Window 메뉴에서 Tool>options 을 선택합니다.)

   C. 옵션 Dialog Box 에서 Show 항목에서 Hierarchy 를 체크 하지 않습니다.

Figure 14-12. Dataflow Options Dialog Box

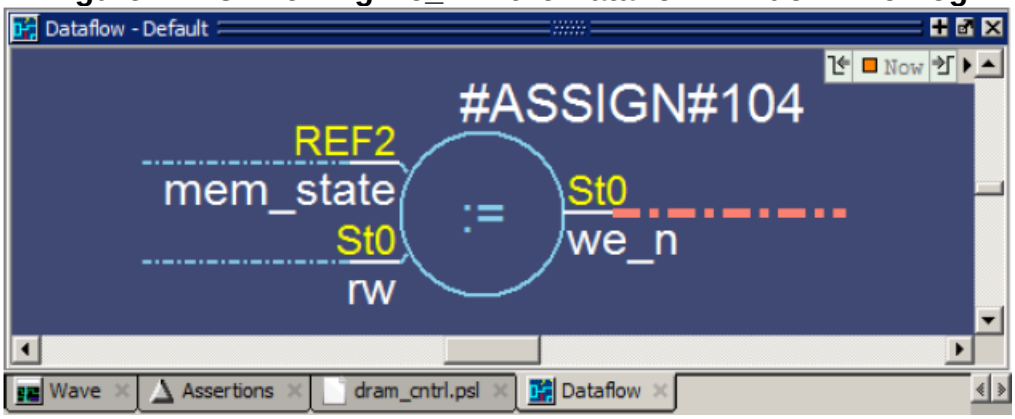


D. Wave Window 에서 we_n 을 선택합니다.

E. 메인 메뉴에서 Add>To Dataflow>Selected Item 을 선택합니다.

Verilog : Input rw 와 mem_state 를 가진 #ASSIGN#104 process 에 의해 we_n 이 동작하는 것을 Dataflow Window 를 통해 확인 할 수 있습니다. 3100 ns 에서 시뮬레이션이 정지하였으며, 그 시점에서 각각의 Signal Value 는 노란색으로 보여줍니다. mem_state 가 REF2 일 때 we_n 이 St0 인 것을 볼 수 있습니다. 하지만 we_n 은 St1 이 되어야 합니다. 이런 이유로 assertion failure 이 되었습니다.

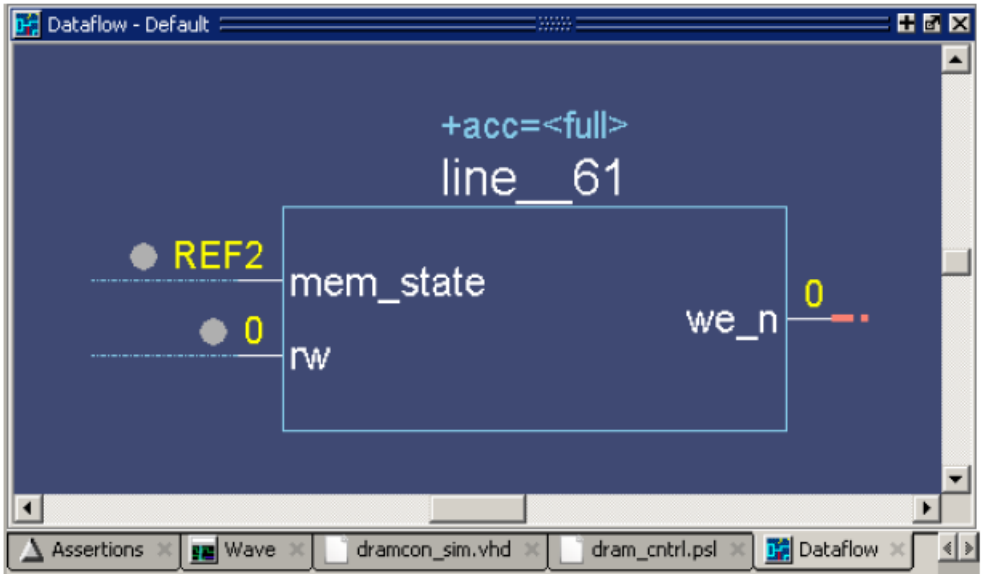
Figure 14-13. Viewing we_n in the Dataflow Window - Verilog



VHDL : Input rw 와 mem_state 를 가진 line_61 에 의해 we_n 이 동작하는 것을 Dataflow Window 를 통해 확인 할 수 있습니다. 3800 ns 에서 시뮬레이션이 정지하였으

며, 그 시점에서 각각의 Signal Value 는 노란색으로 보여줍니다. mem_state 가 REF2 일 때 we_n 이 St0 인 것을 볼 수 있습니다. 하지만 we_n 은 St1 이 되어야 합니다. 이런 이유로 assertion failure 이 되었습니다.

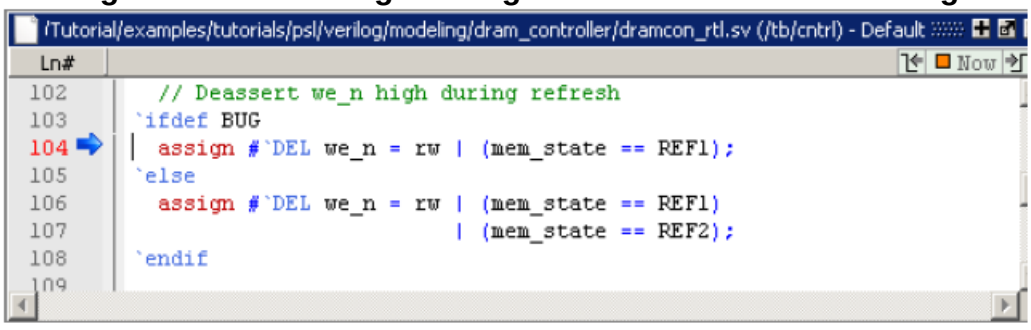
Figure 14-14. Viewing we_n in the Dataflow Window - VHDL



F. Source Window 를 통해 source code 를 확인 하기 위해 we_n 을 동작하는 process 을 더블 클릭합니다.

Verilog : Source Window에서 dramcon_rtl.sv 파일의 104 번 Line 에 파란색 화살표를 확인 할 수 있습니다. 이 라인을 통해 we_n 이 assign 되었을 때, REF2 가 account 되지 않는 것을 볼 수 있습니다.

Figure 14-15. Finding the Bug in the Source Code - Verilog

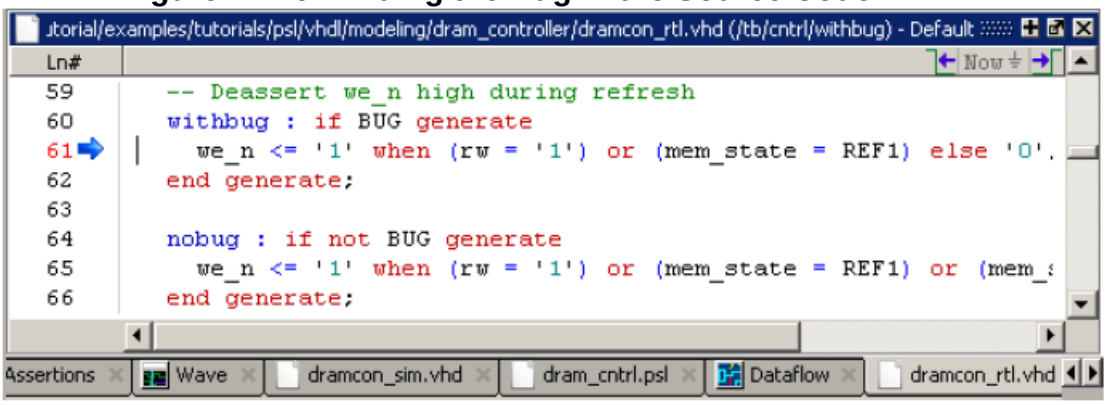


106 ~ 107 을 보면 refresh cycle 의 두 state 를 통해 we_n 이 high 로 되는 예제 코드를 확인 할 수 있습니다.

VHDL : Source Window에서 dramcon_rtl.sv 파일의 61 번 Line 에 파란색 화살표를 확인 할 수 있습니다. 이 라인을 통해 REF2 가 account 되지 않는 것을 볼 수 있습니다.

잘못된 assign 을 사용하는 코드로 확인 할 수 있습니다. 65 Line 을 통해 두 State 를 통해 we_n 이 high 가 되는 코드를 볼 수 있습니다.

Figure 14-16. Finding the Bug in the Source Code - VHDL



G. Wildcard filter 를 Default setting 으로 되돌리기.

set WildcardFilter "default"

H. Simulation 을 종료합니다.

quit-sim command 를 입력하여 Simulation 을 종료합니다.

Chapter 15
SystemVerilog Assertions and Functional Coverage

이번 챕터에서는 Questa 의 SystemVerilog Assertion 과 Functional Coverage 기능을 사용하여 기능적인 하드웨어 검증에 대해 살펴볼 것입니다.

- error 에 도달하기 전에 얼마나 시뮬레이션을 진행했는지 확인 하기 위해 assertion failure tracking 을 비활성화 하여 Design 을 Simulate.
- assertion failure tracking 을 활성화하면 assertion failure 의 error 위치와 빠른 디버깅을 할 때 도움이 됩니다. 활성화하여 다시 Simulation 을 진행합니다.
- test bench 를 위해 cover directive 와 covergroup 을 사용하여 functional coverage 활성화.
- Graphic interface 를 사용하는 functional coverage report 생성

이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

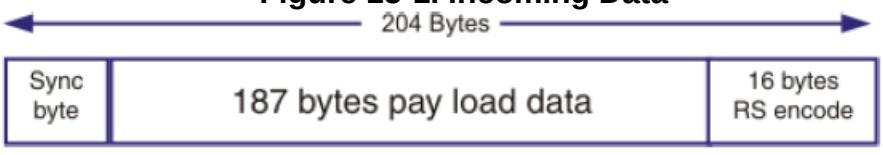
Verilog – */<install_dir>/examples/tutorials/systemverilog/vlog_dut*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Verification with Functional Coverage 를 보시면 더욱 자세히 알 수 있습니다.

Understanding the Interleaver Design

Interleaver 는 Reed Solomon/Viterbi 와 같은 에러 검출 및 정정 방식을 돕기 위해 들어오는 데이터의 바이트 순서를 뒤섞습니다. 이 챕터에서 사용되는 Incoming data 는 sync byte(0xb8, 0x47) 로 구성되어 있으며, 이어서 packet data 는 203 byte 로 되어 있습니다. 203 byte 에서 이미 Reed Solomon encoder 가 16 byte 를 갖고 있기 때문에, 데이터는 187 byte 로 되어 있습니다.

Figure 15-1. Incoming Data



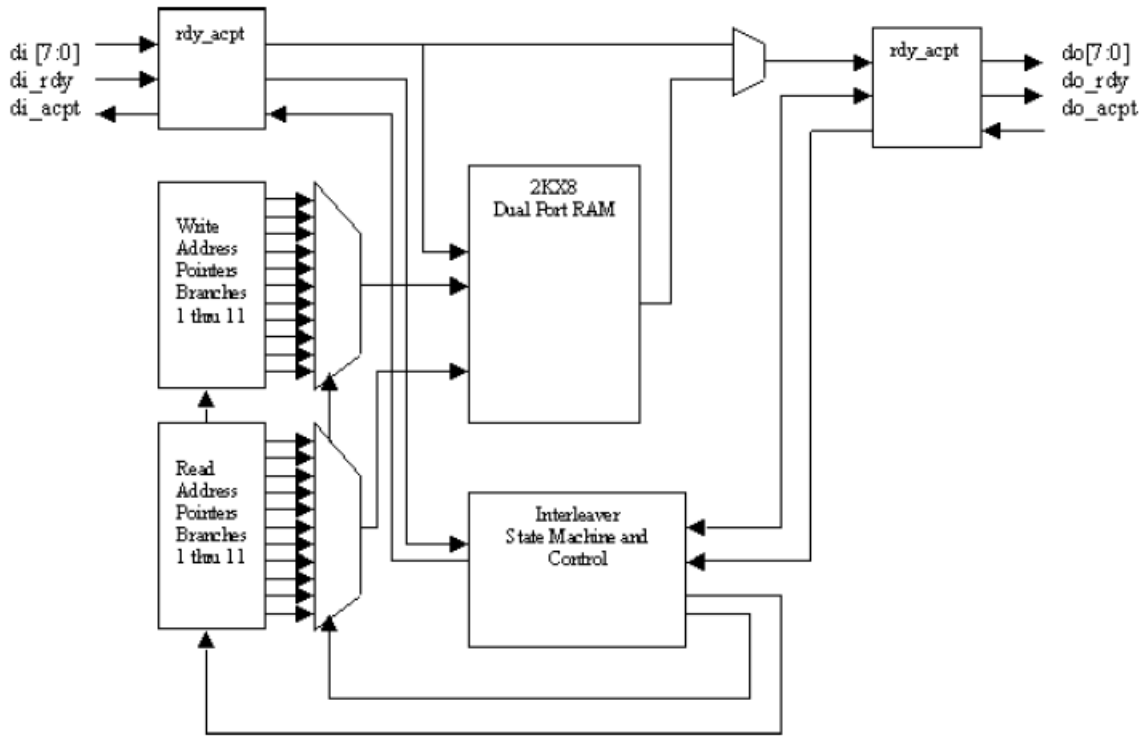
Interleaver 는 0 ~ 11 인 12 level number 가 있습니다. 각 레벨은 처음을 제외하고 개념적으로

FIFO shift register 로 생각 할 수 있습니다. 각 register 의 depth 는 이전 level 보다 17 이상입니다. 처음 level(level 0) 은 0 의 depth 를 가지며, level 1은 17 의 depth 를 가집니다. level 2 는 34. 이렇게 순차적으로 올라가 11 level 은 187 의 depth 를 가집니다. 패킷의 Sync byte 는 level 0 을 통해 연결됩니다. 바이트가 각 level 의 FIFO shift register 로 로드 될 때, byte shift 는 해당 level 에 따라 Interleaver 에 의해 출력됩니다.

실제 Register 대신에 single 2KX8 RAM 을 FIFO shift register 를 사용하여 진행합니다. RAM 은 11개의 다른 섹션으로 구분되며, 각 level 은 별도의 읽기 및 쓰기 address register 를 가지고 있습니다. State machine control 의 level 은 address register 의 level 에 의해 읽기, 쓰기를 결정하며, 실제 RAM address input 을 동작합니다.

rdy_acpt block 은 각각의 interleaver 의 각 data in(di) 과 data out(do) 포트를 동작하는데 사용됩니다. rdy_acpt 라는 일반적인 블록은 각각 수신(DI) 및 데이터 아웃 (DO) 포트에 데이터 interleaver를 구동하는데 사용됩니다. rdy_acpt 블록을 통해 간단한 handshake protocol 을 구현할 수 있습니다. 그것에 interleaver Deriver 데이터로부터 upstream 장치의 데이터가 구동되는 경우 ready signal이 (di_rdy)가 asserted 됩니다. Upstream block 은 rdy 신호와 함께 데이터를 downstram 이 받아 (di_acpt) 신호를 assert 할 때까지 asserted 되어야 합니다. rdy 와 acpt 신호 모두가 Clock 의 rising edge 에서 전송 될 때까지 assert 된 것으로 간주되지 않습니다. rdy_acpt 블록의 양쪽이 handshake protocol 을 따릅니다. Interleaver 의 block diagram 이 보여집니다.

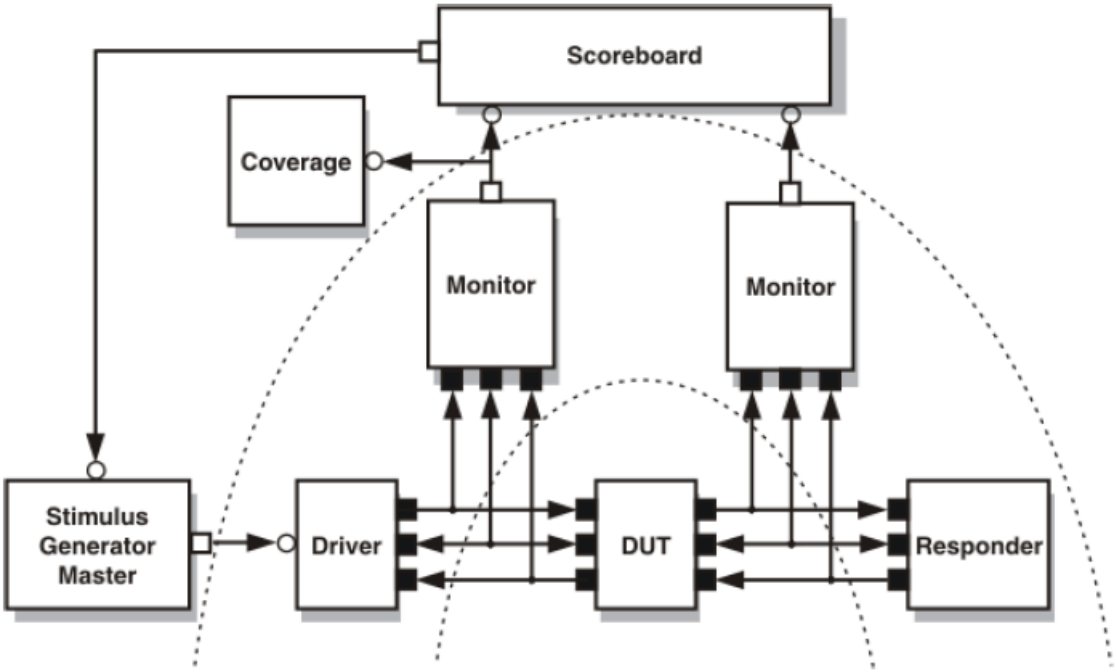
Figure 15-2. Block Diagram of the Inteleaver



The Test Bench

아래의 그림은 test bench 가 연결되는 방법을 보여줍니다. Stimulus generator 는 임의의 데이터 패킷을 생성하고 Driver 에게 보냅니다. Test bench 를 기반으로 하더라도, stimulus generator 는 여전히 Transaction based(SV class) 패킷을 생성합니다. Advanced Verification Methodology (AVM)가 제공하는 큰 장점입니다 – 유저 분이 완전한 객체 지향 프로그래밍 환경으로 진행하면 Test bench 를 변환 할 필요 없이 향상된 transaction level modeling(TLM) 기술을 활용할 수 있습니다.

Figure 15-3. Block Diagram of the Test Bench



드라이버는 TLM 패킷 들을 가지고, 핀 레벨 신호로 변환합니다. 드라이버는 장치에 전달 된 패킷의 타이밍을 변화시키는 randomization 을 사용합니다.

모니터는 DUT 입력 및 출력 핀의 레벨 활성을 가지고 Coverage collector 및 scoreboard 로 사용하기 위해 다시 transaction 을 activity 로 변환합니다.

scoreboard 는 실제 output 장치와 비교하는 것이 interleaver 의 "golden" reference 모델에 포함되어 있습니다. 테스트가 완료되면 stimulus generate 의 scoreboard 로부터 feedback loop 가 있습니다.
coverage collector 는 테스트가 완료되면 functional coverage 정보를 결정하는 데 도움이 됩니다.. 이것은 패킷 전송을 사용 할 때 다른 여러 delay value 의 종류를 측정합니다.

마지막으로 responder(실제 테스트 벤치 드라이버의 일부입니다) 는 패킷 전송에 필요한

handshaking ready/accept 신호를 제공합니다.

Run the Simulation without Assertions

1. 새로운 디렉토리를 만들고 튜토리얼 파일 복사하기.

   새로운 디렉토리에 */<install_dir>/examples/tutorials/systemverilog/vlog_dut* 를 복사합니다.

2. Questa 를 실행하기.

   A. UNIX shell prompt 에서 vsim 을 입력하거나, Window 의 Questa 아이콘을 더블클릭 합니다.

   B. 메인 메뉴의 File>Change Directory 를 선택하여 step 1 에서 생성한 디렉토리로 변경합니다.

3. .DO File 과 함께 시뮬레이션 실행하기.

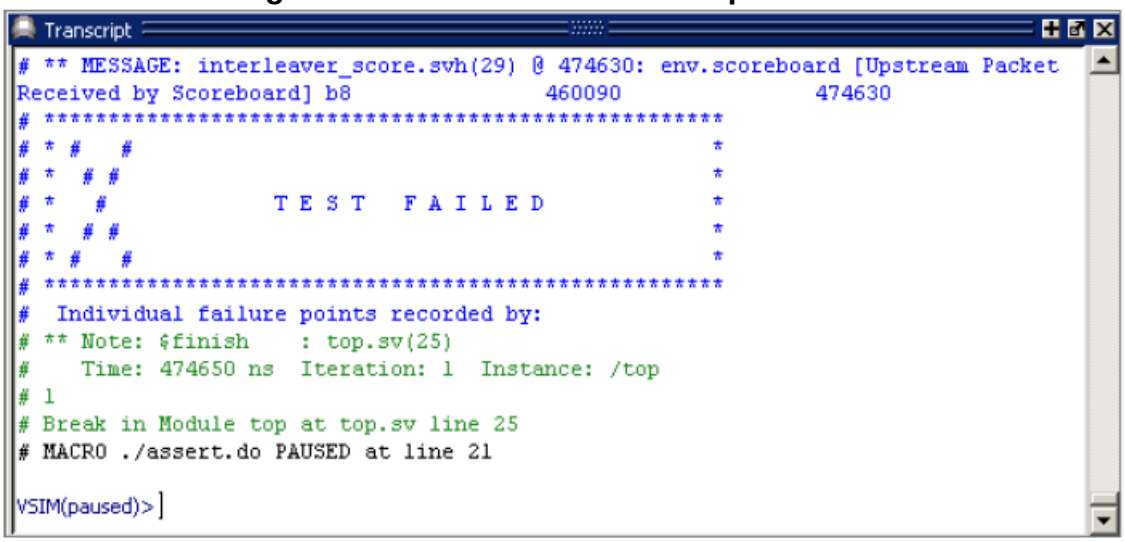
   A. Command 에 do assert.do 를 입력합니다.

      DO File 은 시뮬레이션 결과를 확인하면서 일시 정지를 하고, 컴파일 하여 디자인을 로드 합니다. 그리고 assertion 없이 시뮬레이션을 실행합니다. (만약 "Finish Vsim" dialog box 가 나타나 "Are you sure you want to finish?" 라고 물으면 NO 를 클릭하면 됩니다.)

      유저 분은 assertion 과 함께 simulation 을 다시 실행 할 수 있도록 resume 명령어를 입력합니다.

      디자인이 로드 된 후 top.sv module 은 $fisish 를 찾을 때까지 시뮬레이션이 실행됩니다. 그리고 Transcript Window 를 보면 "Test Failed" massage 가 보여집니다. summery 정보를 보면 22 packet 이 올바르게 scoreboard 에 수신되었다는 것을 나타냅니다. 이것은 self-checking test bench 에서 일반적인 메시지입니다.

Figure 15-4. First Simulation Stops at Error



이 시점에서 유저 분은 일반적으로 test failure 디버깅을 위한 waveform 을 생성합니다. 그러나 그 정보는 문제의 원인에 대한 정확한 source 를 제공하지 않습니다. 그렇기 때문에 assertion 을 사용하여 디버깅을 해야 합니다.

Run the Simulation with Assertions

유저 분은 디버깅을 위해 Assertion 을 사용하여 시뮬레이션을 실행하겠습니다.

1. Assertion 을 사용하여 시뮬레이션을 다시 실행하기.

   A. Transcript window 에서 resume 명령어를 입력합니다.

2. 디자인 로드 후에 모든 assertion 을 "Break on Failure" 로 구성하기.

   A. Assertion window 를 열기 위해 View>Coverage>Assertions 를 선택합니다.

   assertion 에 대한 pass 와 failure 을 모두 사용할 수 있는지 확인합니다. Assertion 기능이 default 로 되어 있지 않기 때문에 확인을 해야 합니다.

   Wave window 를 Count 와 Visual indication 을 통해 Assertion pass와 failure 를 확인할 수 있습니다.

   vsim –assertdebug 를 통해 시뮬레이션을 진행 할 때 assertion 기능을 사용 할 수 있습니다.(이것은 assert.do file 의 명령어에 있습니다.)

   B. Assertions tab 을 클릭하여 Assertion Window 를 활성화합니다. "Assertions" 는 menu bar 에 표시됩니다.

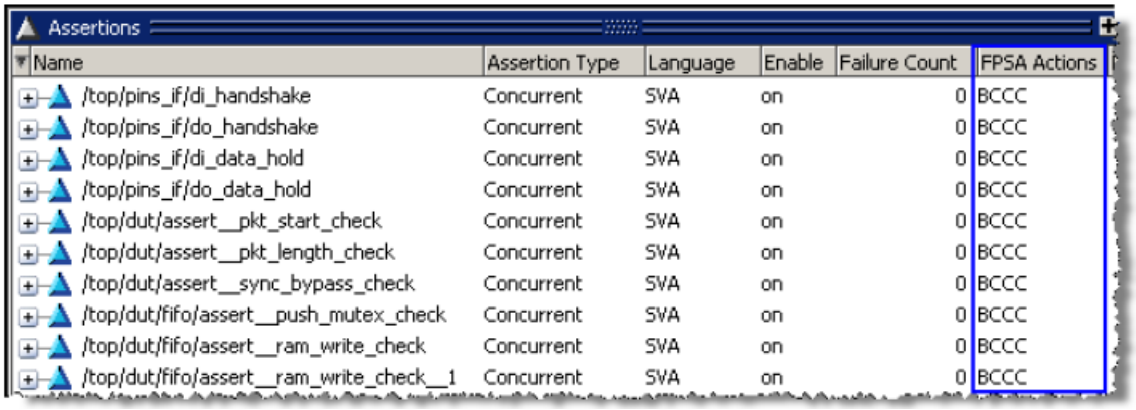
C. Assertion 이 모두 선택되었는지 확인합니다. Edit>Unselect all

D. Command 를 입력하여 실행합니다.

assertion action -cond fail -exec break -r *

FPSA Actions 를 통해 "B" 가 "break" 이며, assertion failure 를 확인 할 수 있습니다. (FPSA 는 Failures, passes, starts, antecedents 를 보여줍니다). Column 을 보면 4 문자가 표시되는데, 첫 번째 문자는 Failures, 두 번째 문자는 Passes, 세 번째 문자는 Starts, 네 번째 문자는 Antecedents 입니다. (Actions 는 continue, break, exit, tcl.) 만약에 FPSA 가 표시되지 않으면 열 탭을 선택하여 간격을 조정하여 볼 수 있습니다.

Figure 15-5. Assertions Set to Break on Failure



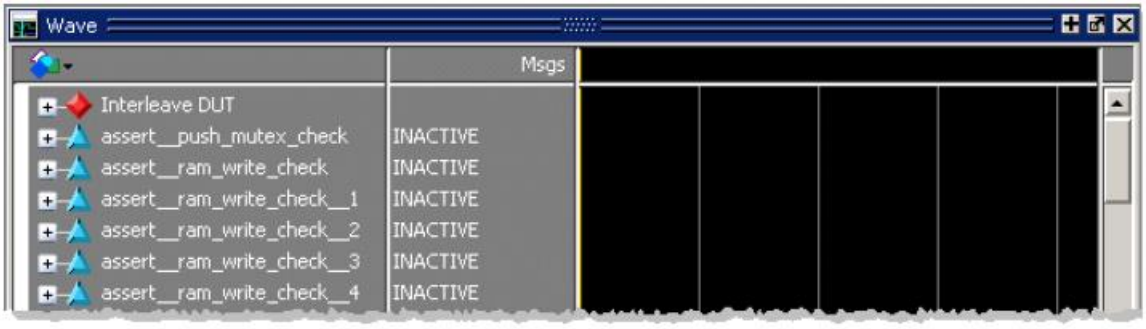
3. Wave Window 에 /top/dut/fifo 에 관련된 모든 assertion 을 추가합니다.

A. Assertion Window 에서 /top/dut/fifo/assert_push_mutex_check 를 선택합니다.

B. Shift 키를 누른 채 /top/dut/fifo/assert_ram_read_check_10 을 선택합니다. /top/dut/fifo 에 관련된 모든 assertion 을 선택해야 합니다.(파란색으로 표시)

C. Add>To Wave>Selected Objects 를 선택합니다. 선택된 Assertion 이 Wave Window 에 나타납니다.

Figure 15-6. Assertions in Wave Window



Debugging with Assertions

시뮬레이션을 실행하고 Assertion failure 를 디버깅합니다.

1. Assertion failure tracking 을 활성화 하고 시뮬레이션 실행.

   A. Transcript Window 에서 run –all 을 입력합니다.

   B. 시뮬레이션이 중지되면 run 0 을 입력합니다.

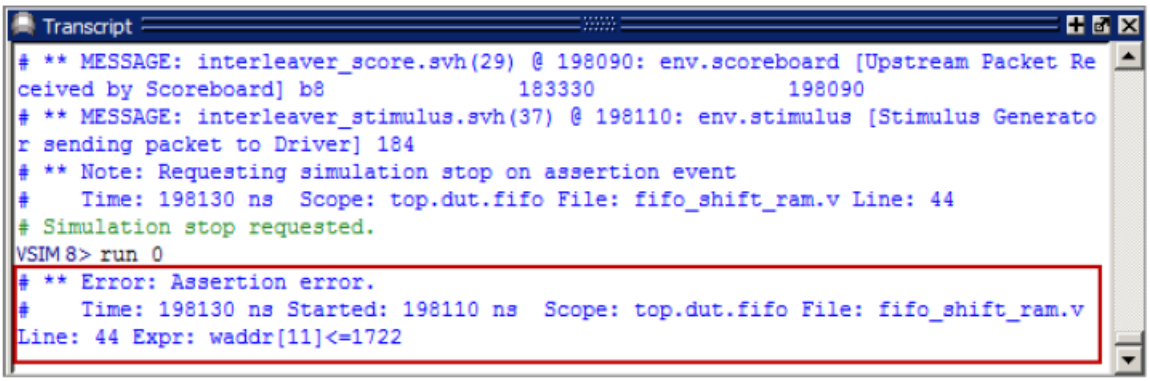
      run 0 명령어는 assertion failure action 이 break 가 설정되어 있는 경우 massage 를 출력하는데 사용합니다. "Break" 는 active event queue 에 발생합니다. Assertion massage 는 observed region 에 scheduled 되어 있습니다. Observed region 은 다음 time step 입니다. run 0 은 time step 끝으로 이동합니다.

2. Transcript window 를 통한 output 확인하기.

   Assertion failure massage 가 failing expression 을 제공하는 것을 알 수 있습니다.

   –assertdebug 를 vsim 과 함께 사용하는 경우 이 기능을 사용 할 수 있습니다. (dl 명령은 assert.do 파일에 있습니다.)

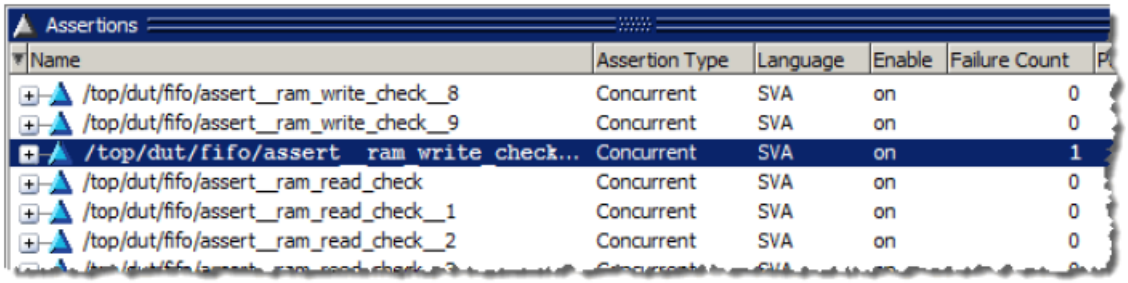
Figure 15-7. Assertion Failure Message in the Transcript



3. Assertion Window 를 통해 assertion failure 확인하기.

   실패한 assertion 은 하이라이트 와 "1" 로 볼 수 있으며, Verilog 디자인의 assertion Failure count 를 열을 통해 확인 할 수 있습니다.

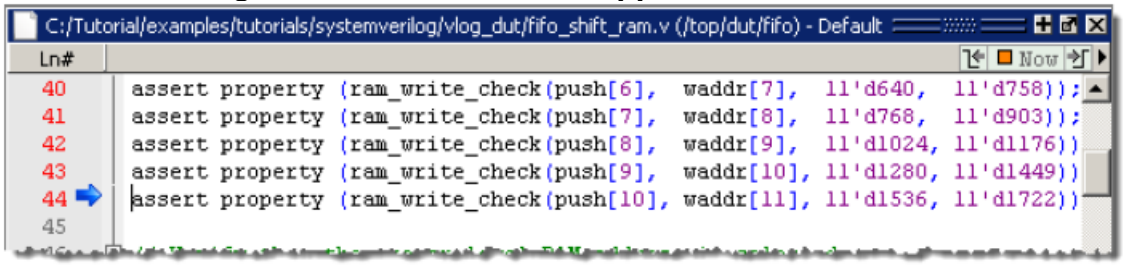
Figure 15-8. Assertions Tab Shows Failure Count



4. fifo_shift_ram_propss.v 소스 코드를 확인합니다.

   fifo_shift_ram.v 를 확인해 보면 44 번 라인에 assertion failed 된 부분을 파란색 화살표로 가리킵니다.

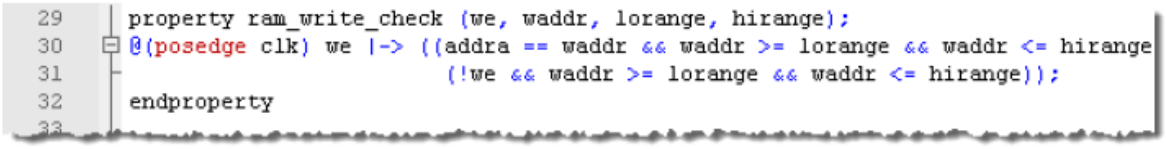
Figure 15-9. Simulation Stopped at Blue Pointer



parameterized property 가 선언 된 29 번 line 부터 시작합니다.

A. fifo_shift_ram.v 에서 property 가 선언된 29번 라인으로 이동합니다.

Figure 15-10. Assertion Property Definition



Property 를 보면 (push[10]) 은 같은 주기에 assert 되는 것을 알 수 있습니다.

- Ram address bus 는 level 11(waddr[11]) 에 대한 write address bus 와 같아야 합니다.
- 그리고 waddr[11] 은 1536 ~ 1722 범위 내에 있어야 합니다.

다음 주기 입니다.

- we 는 assert 를 하지 않습니다.

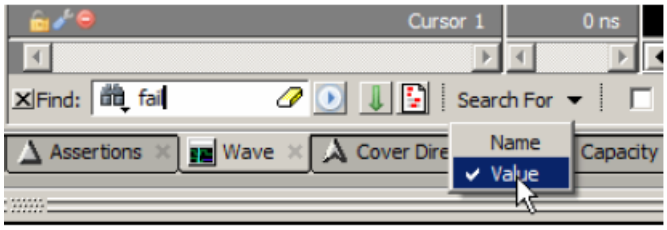
- 그리고 waddr[11] 의 다음 값은 1536 ~ 1722 범위 내에 있어야 합니다.

5. Assertion failure 를 보려면 Wave window 를 선택하기.

   A. Wave Window 에서 Edit>Find 를 선택하여 search bar 를 볼 수 있습니다.

   B. Search bar 에서 Search for>Value 를 선택합니다.

Figure 15-11. Search For Value

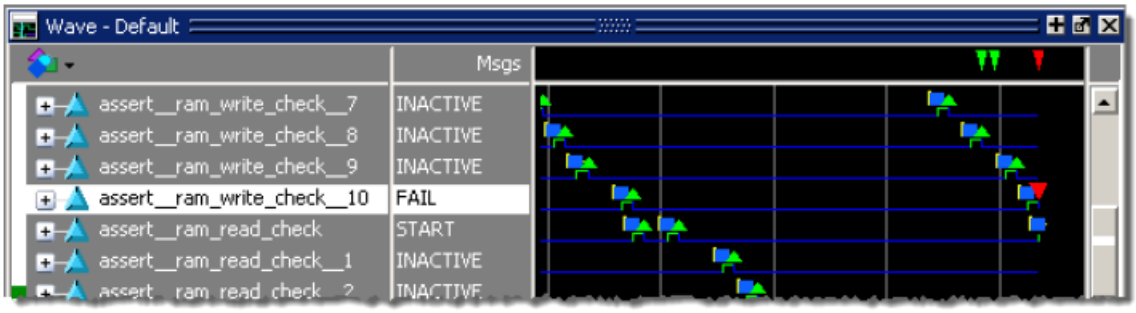


   C. Search bar text entry box 에 fail 을 입력합니다.

   검색은 사용자 입력으로 시작되며, "FAIL" 값이 하이라이트로 나타납니다.

   Waveform 에서 빨간색 삼각형은 assertion failure 를 나타냅니다.

Figure 15-12. Inverted Red Triangle Indicates Assertion Failure

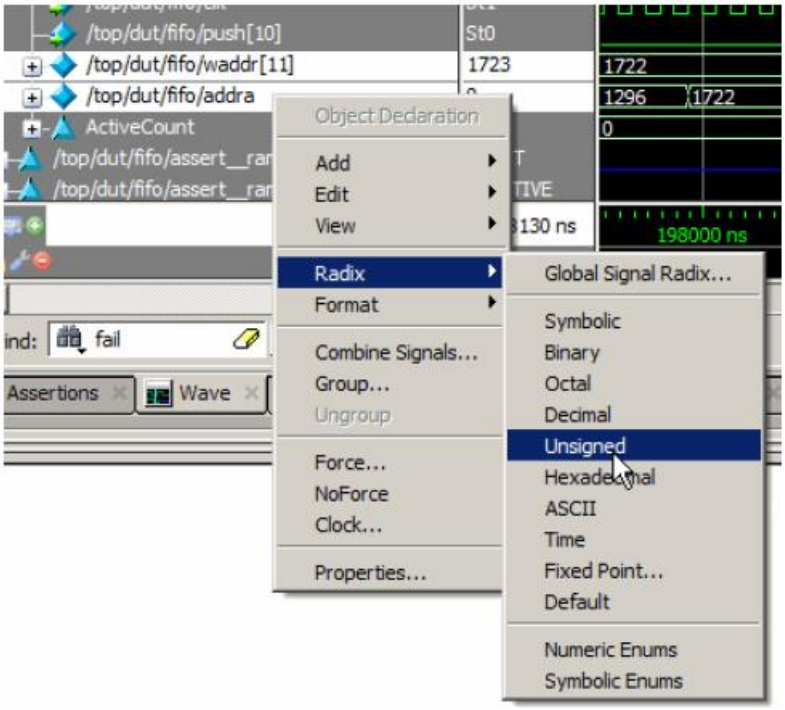


- Assertion 이 활성화되면 중간에 녹색이 나타나며, 비 활성화인 경우 파란색 선이 나타납니다.
- Assertion thread 시작은 파란색 사각형으로 나타납니다.
- 녹색 삼각형은 assertion pass 를 나타냅니다. vsim 명령어와 함께 –assertdebug 옵션을 사용하면 pass 가 표시됩니다.(assert.do 파일을 참조하시면 됩니다.)

   D. Wave window 를 Zoon in 하여 assert_ram_write_check_10 을 확인 후 [+] 를 눌러 확장합니다.

E. addra 와 waddr 신호를 선택 한 후, 오른쪽 마우스를 클릭하면 팝업 메뉴가 나타나는데, 팝업 메뉴에서 Radix>Unsigned 를 선택합니다.

Figure 15-13. Setting the Radix

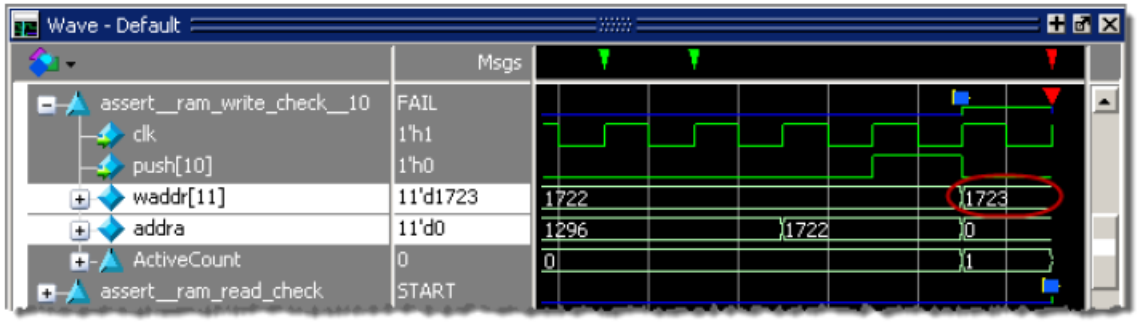


waddr[11] 값은 address 범위를 벗어나 1723으로 증가했습니다. 범위를 벗어나 failing expression 된 것은 transcript window 에 assertion violation 에 대한 내용이 보여집니다.

6. Dataflow window 를 통해 신호를 검사하기.

A. waddr[11] 신호를 찾아 [+] 를 눌러 waddr 신호를 확장합니다.

Figure 15-14. Diagnosing Assertion Failure in the Wave Window

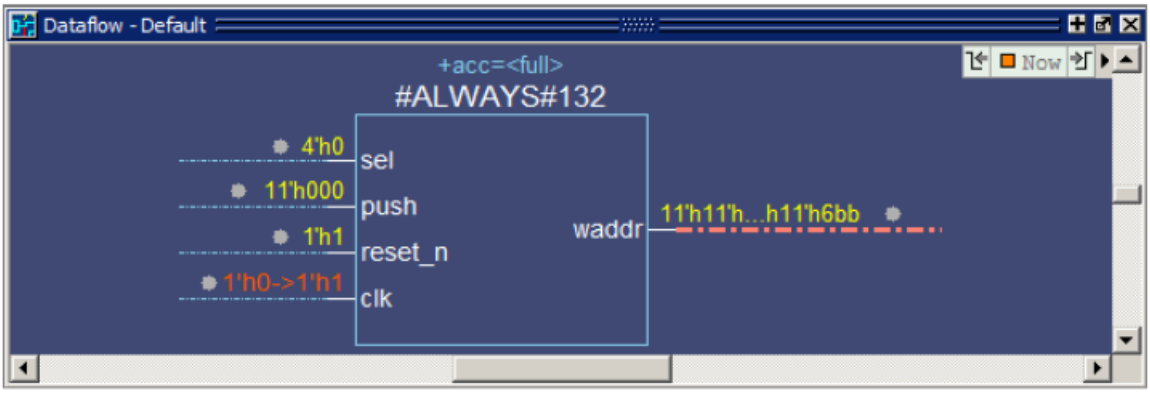


B. waddr[11] 신호를 선택한 후 Add>To Dataflow>Selected Items 를 선택합니다.

그러면 선택된 신호가 Dataflow Window 에 열립니다.

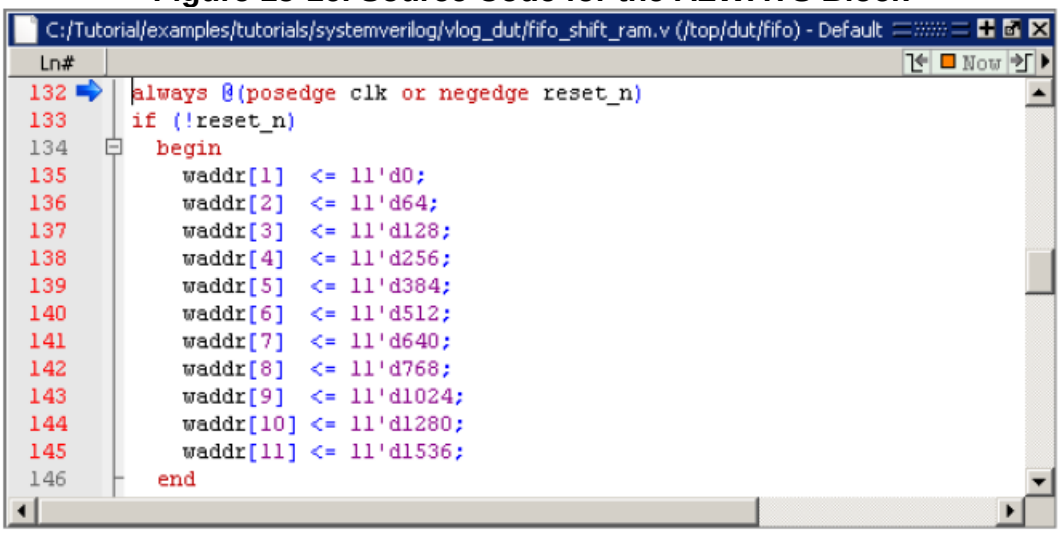
waddr[11] 신호는 하이라이트로 보여지며, ALWAYS Procedure 블록으로 볼 수 있습니다.

Figure 15-15. The *waddr11* Signal in the Dataflow Window



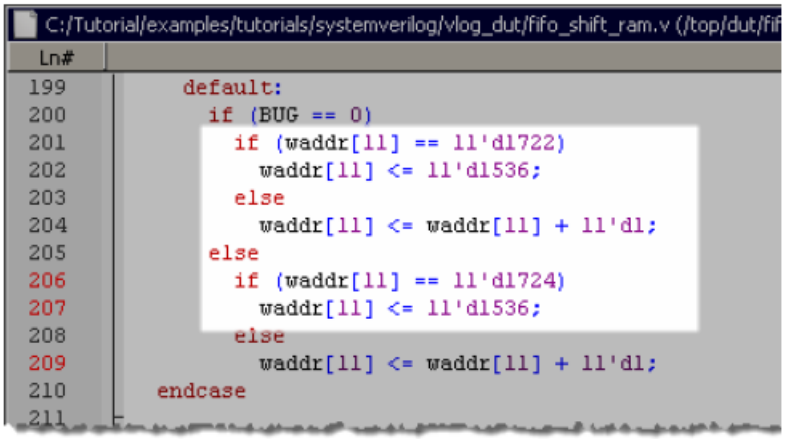
C. Dataflow Window 에서 ALWAYS 블록을 더블 클릭합니다. 그러면 fifo_shift_ram.v 의 소스코드가 나타나며 해당 부분을 파란색 화살표로 표시합니다.

Figure 15-16. Source Code for the ALWAYS Block



waddr[11] 을 보면 11'1722 대신 11'1724 로 되어있는 것을 확인 할 수 있습니다. 이것이 에러의 원인입니다.

Figure 15-17. Source Code for *waddr[11]*



7. 시뮬레이션 종료하기.

   A. Command 에 quit –sim 을 입력합니다.

Exploring Functional Coverage

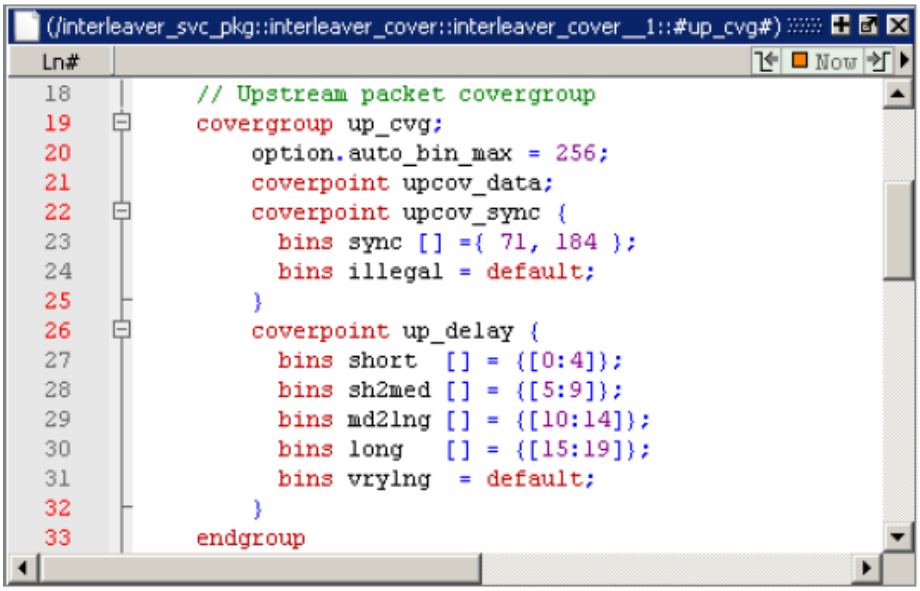
SystemVerilog 의 functional coverage 기능을 사용하면 유저 분이 만든 디자인의 functional level 을 확인 할 수 있습니다.

1. 다시 interleaver 로드하기.

   A. Command 에 do fcov.do 를 입력합니다.

      Interleaver 는 interleaver 되는 유효한 패킷의 수를 결정하기 위해 80 으로 설정된 parameter(PKT_GEN_NUM) 을 사용합니다. Scoreboard 는 80 개의 패킷이 stimulus generate 와 driver 를 멈추고 test controller 를 interleaver 되었다고 알려줍니다. 시뮬레이션 동안 coverage collector 는 여러가지로 전송 된 각 패킷에 대한 several metrics 와 interleaver 에 대한 output 을 기록합니다. up_cvg 의 소스코드를 확인합니다.

Figure 15-18. Covergroup Code



모니터로 upstream transaction 을 capture 하여 transaction 에 저장된 정보를 covergroup 에 기록합니다. Transaction 은 packet payload data, sync byte, 각각의 data payload 의 데이터 전송 시간을 포함합니다.

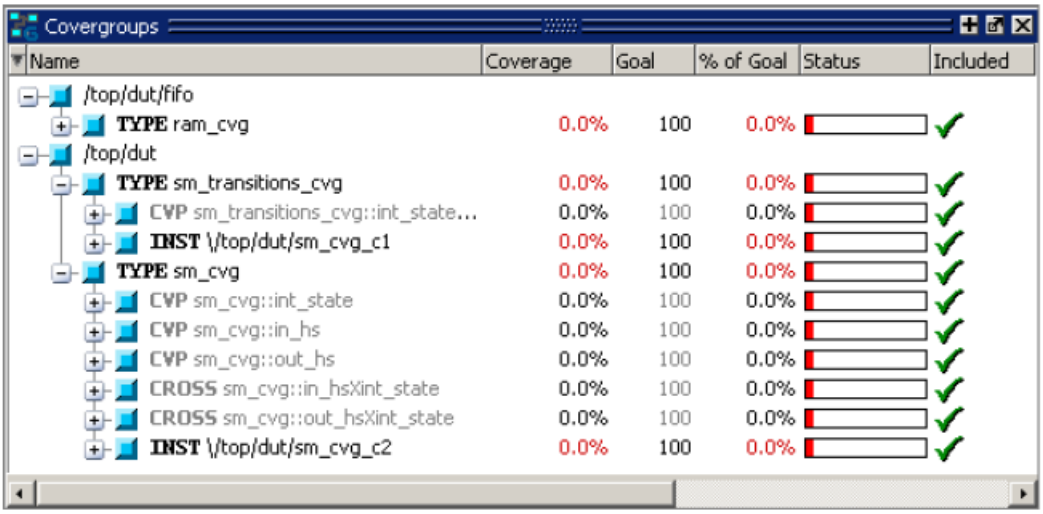
SystemVerilog 를 LRM 에 의해 정의하여 만든 auto bins 의 기본번호는 (64)로부터 각각의 데이터 값을 생성하기 위해 option.auto_bin_max = 256 을 지정합니다. Sync byte 값은 71 과 184 이며, 8'h47 과 8'hb8 입니다.

2. run 0

   시뮬레이션이 실행될 때까지 covergroup 은 표시되지 않습니다. 이 단계는 단순히 covergroup 창의 covergroup 을 볼 수 있습니다.

3. Covergroups Window(View>Coverage>Covergroups) 에서 /top/dut/ 를 선택하여 [+] 를 클릭하여 확장합니다. 그러면 interleave state machine 을 모니터 하여 추가 신호 인 sm_transitions_cvg 와 sm_cvg 를 찾을 수 있습니다.

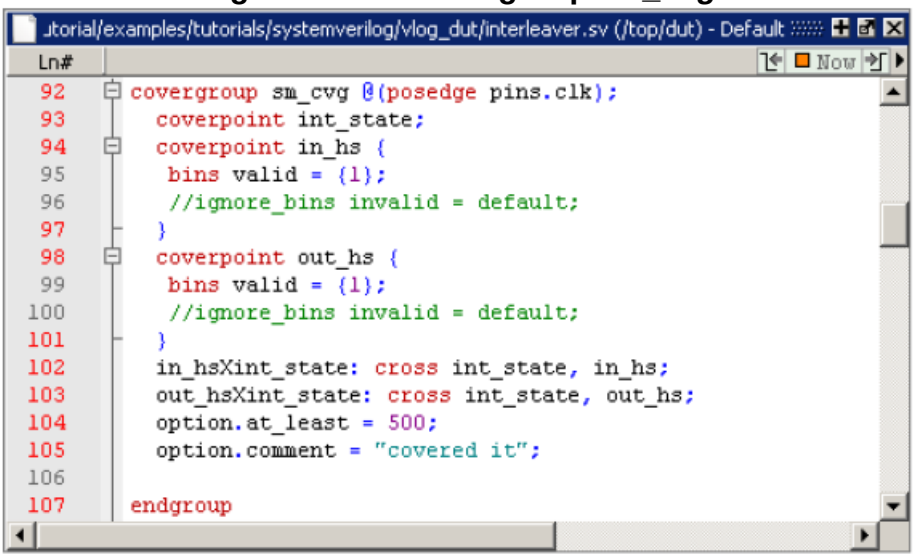
Figure 15-19. Covergroup Bins



sm_cvg record 는 state machine 이 제대로 수신 데이터를 받아 적절한 상태에서 output 데이터를 동작하면서, sm_transitions_cvg 의 state machine transition 을 기록합니다.

sm_cvg covergroup 의 소스 코드를 확인해보겠습니다.

Figure 15-20. Covergroup *sm_svg*

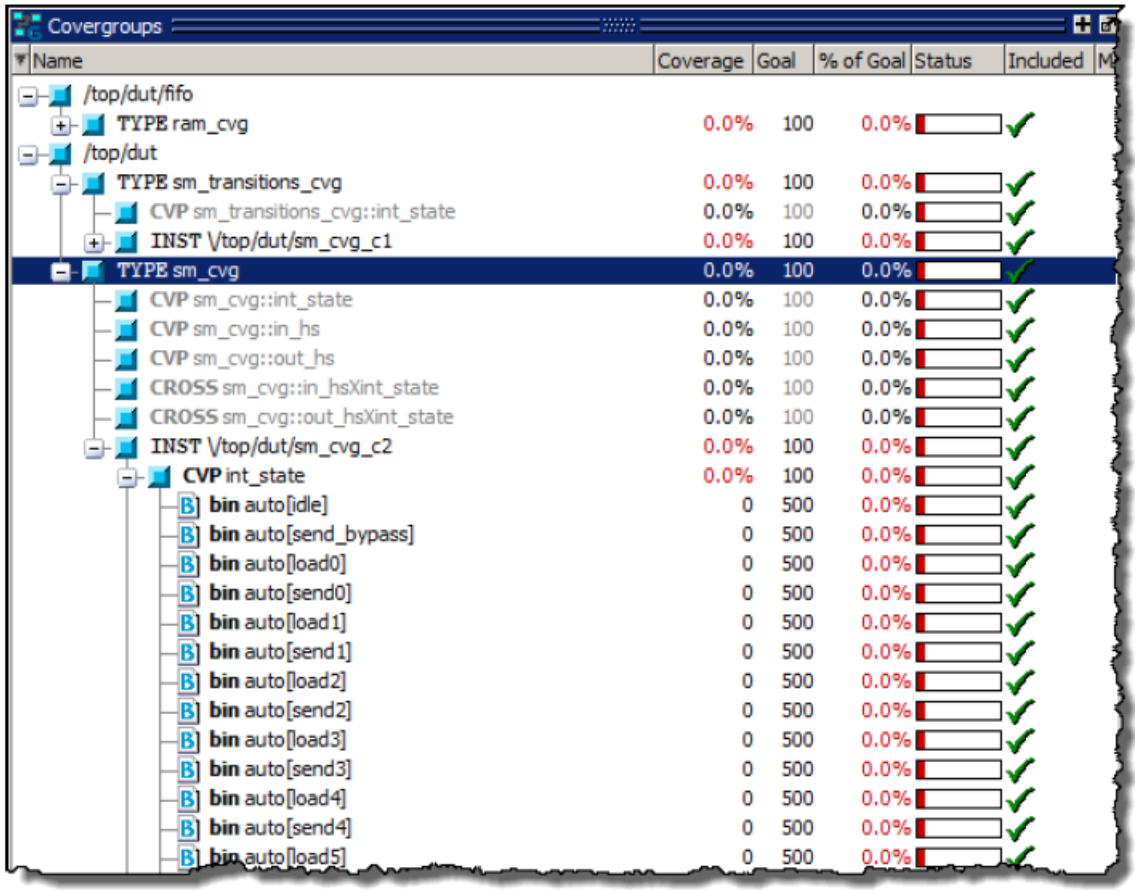


in_hs 신호와 out_hs 신호는 AND 연산에 의해 in_acpt 와 in_rdy, out_acpt 와 out_rdy 로 각각 derive 됩니다. idle, load_bypass 일 때, 또는 10 개의 state 중 하나의 load state 일 때 in_acpt 를 assert 하며, send_bypass 나 10 개 state 중 하나를 보낼 때 oup_rdy 를 assert 합니다.

State machine 은 idle, load_bypass 또는 10 개의 state 중 하나인 경우 in_hs 를 assert 합니다. load_bypass 거나 10 개의 state 중 하나를 보내면 마찬가지로 out_hs 를 assert 합니다. in_hs 와 int_state, 그리고 out_hs 와 ini_state 를 Crossing 하여 동작을 확인 할 수 있습니다.

int_state 의 coverpoint 와 sm_cvg 의 covergroup 모든 bin 을 볼 수 있도록 확장합니다. bin 값은 열거된 state 의 이름을 보여줍니다.

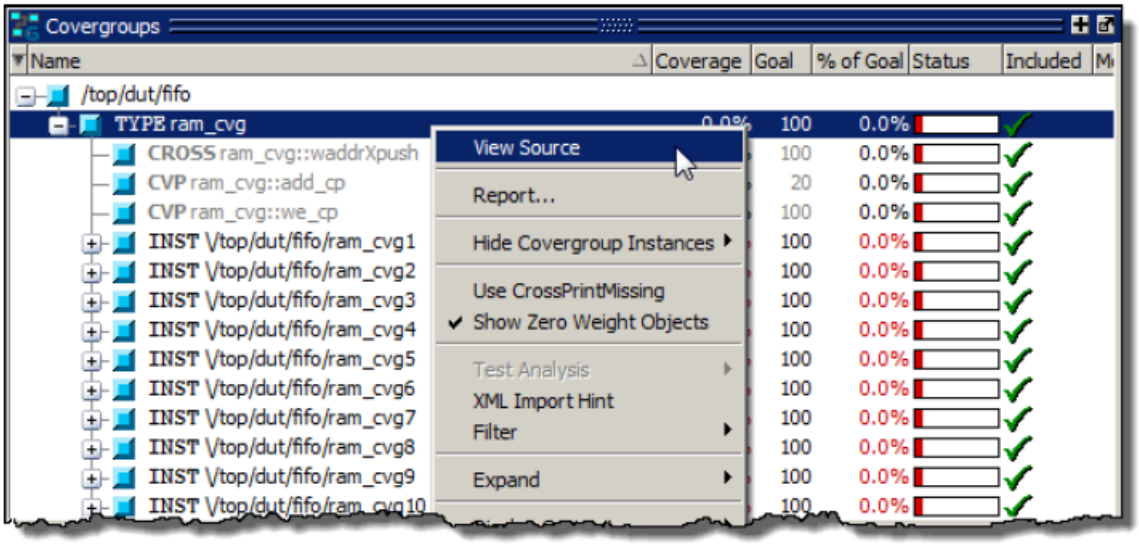
Figure 15-21. Bins for the *sm_cvg* Covergroup



4. ram_cvg covergroup 에서 /top/dut/fifo 를 [+] 버튼을 클릭하여 확장합니다. Covergroup 의 TYPE sm_cvg 는 여러 인스턴스를 포함하는 것을 알 수 있습니다.(INST 로 지정)

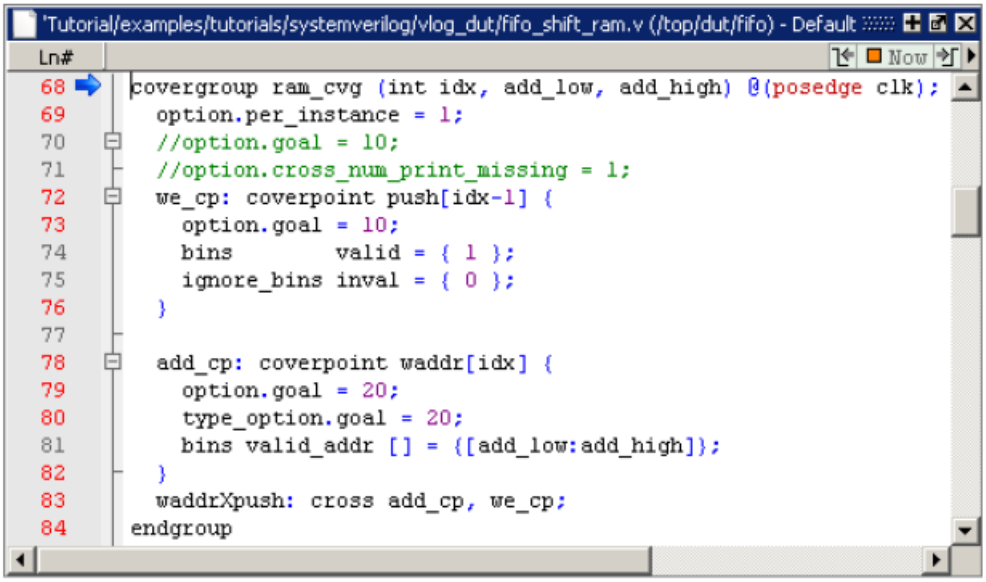
   A. Covergroup 안에 있는 인스턴스의 이름을 마우스 오른쪽 버튼으로 클릭하고, 팝업 메뉴에서 view source 를 클릭하여 TYPE ram_cvg 의 소스 코드를 볼 수 있습니다.

Figure 15-22. Viewing the Source Code for a Covergroup



fifo_shift_ram.v 의 코드를 확인하기 위해 source window 를 엽니다.

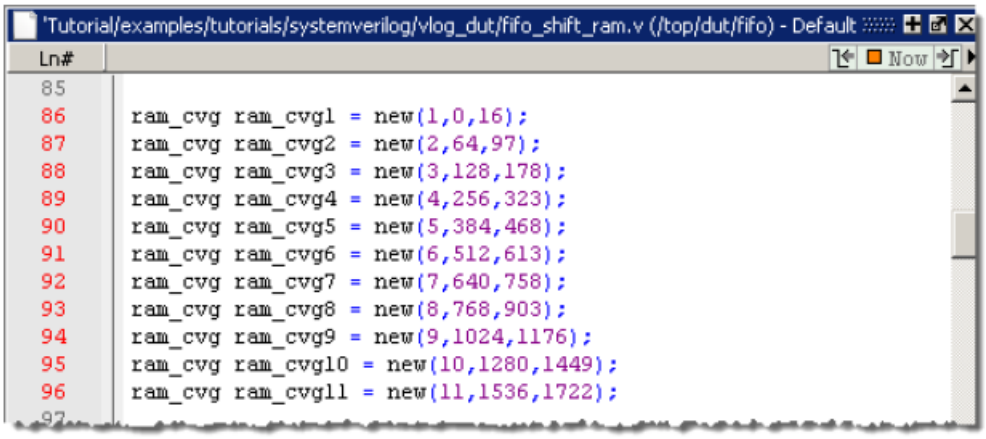
Figure 15-23. Source Code for *ram_cvg* Covergroup



Interleaver level 은 각 레벨에 대한 별개의 RAM 주소 범위와 단일 RAM 을 사용하여 구현되기 때문에, covergroup 에만 유효 address 위치를 읽고 있는지 확인합니다.

하나의 Covergroup 이 있는 것을 확인 할 수 있지만, constructor 에 전달 할 때 다른 값으로 구성되는 11개의 covergroup instance 가 있습니다.

Figure 15-24. Covergroup Instances for *ram_cvg*



covergroup 에 option.per_instance = 1 구문이 포함되어 있기 때문에 시뮬레이터는 생 constructor 에 전달 된 값만 다루고 각 인스턴스에 대해 별도의 covergroup 을 만듭니다.

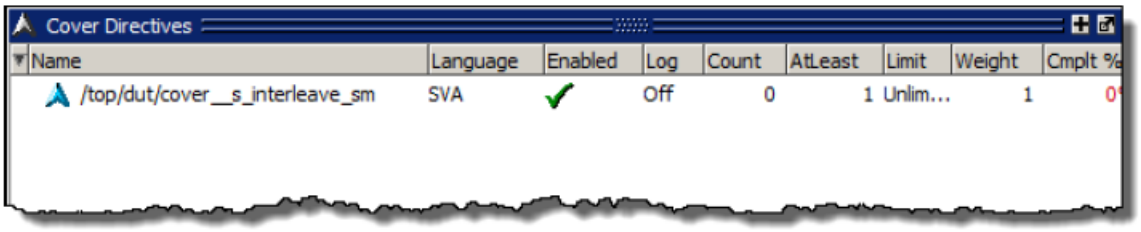
TYPE ram_cvg 의 covergroup 은 개별 covergroup instance 의 모든 값의 조합입니다.

5. Cover Directives window 를 열고 cover directive 에 대한 소스코드 보기.

   A. View>coverage>cover Directives 를 선택하여, Cover Directives window 를 엽니다.

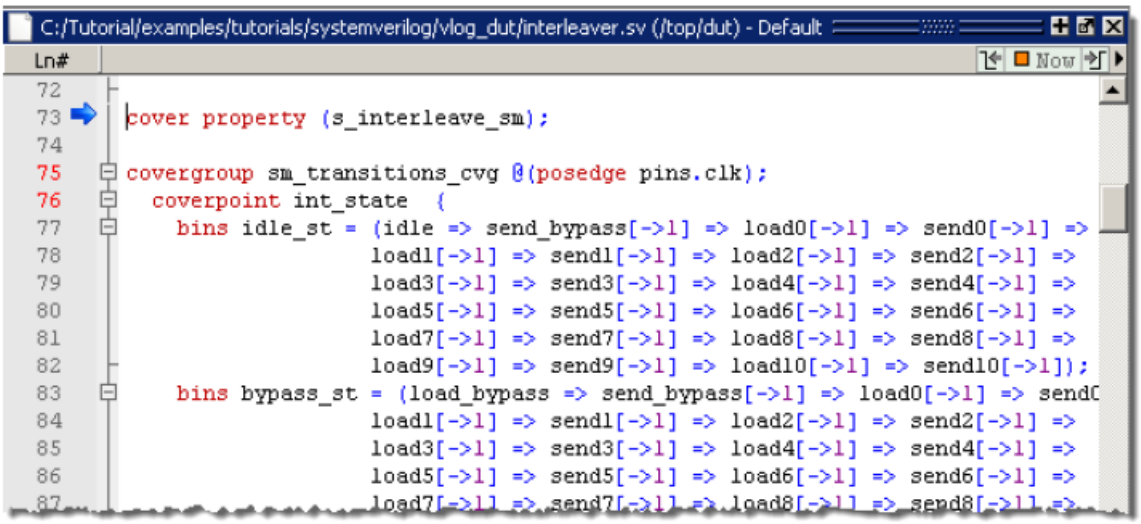
      Cover Directives tab 에는 하나의 cover directive 가 포함되어 있습니다.

Figure 15-25. Cover Directive for the Interleaver Design



   B. Cover directive 를 오른쪽 마우스 버튼으로 클릭하고 View Source 를 선택합니다. Cover directive 는 interleaver state machine transition tracking 을 보여줍니다.

Figure 15-26. Source Code for the Cover Directive



System Verilog 디자인에 중요한 항목을 추가하기 위해 여러 가지 방법을 제공합니다. Wave Window 에 directive 가 hit 했을 때 표시하는 기능을 제공합니다. Covergroup 이 이벤트가 적용되는 정확한 시간을 제공하지는 않지만, 일반적으로 데이터 값을 포함하는데 훨씬 뛰어납니다. SystemVerilog 의 coverage 기능이 모두 covergroup 의 oriented value 데이터를 샘플링하는 시기를 결정하기 위해 cover directive 를 시간적 성질을 이용하여 강력한 조합을 제공합니다.

6. Wave Window 로 cover directive 를 추가 합니다.

   A. Cover Directives Window 로 돌아가서 /top/dut/cover_s_interleave_sm 을 선택하고 마우스 오른쪽 버튼을 클릭합니다. 그리고 Add Wave>Selected Functional Coverage 를 선택합니다.

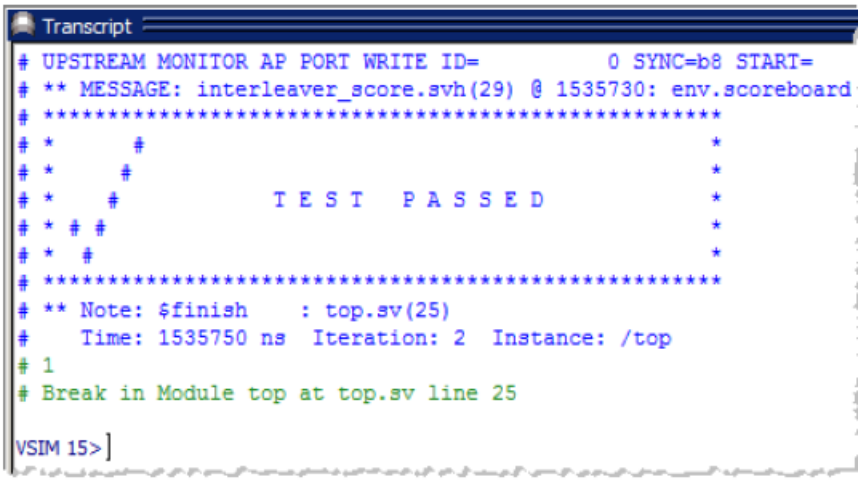
7. 시뮬레이션을 실행하고 functional coverage 정보보기.

   A. Cover Directive 가 기록될 수 있도록 WildcardFilter 를 변경합니다. Command 입력을 통해 "Cover" 를 기본 필터링에 포함하지 않습니다.

      set WildcardFilter "Variable Constant Generic Parameter SpecParam Memory Assertion Endpoint CellInternal ImmediateAssert"

   B. Transcript Window 에 run –all 을 입력합니다. "TEST PASSRED" 메시지가 나올 때까지 디자인을 실행합니다. Transcript window 에 scoreboard 정보가 표시됩니다.

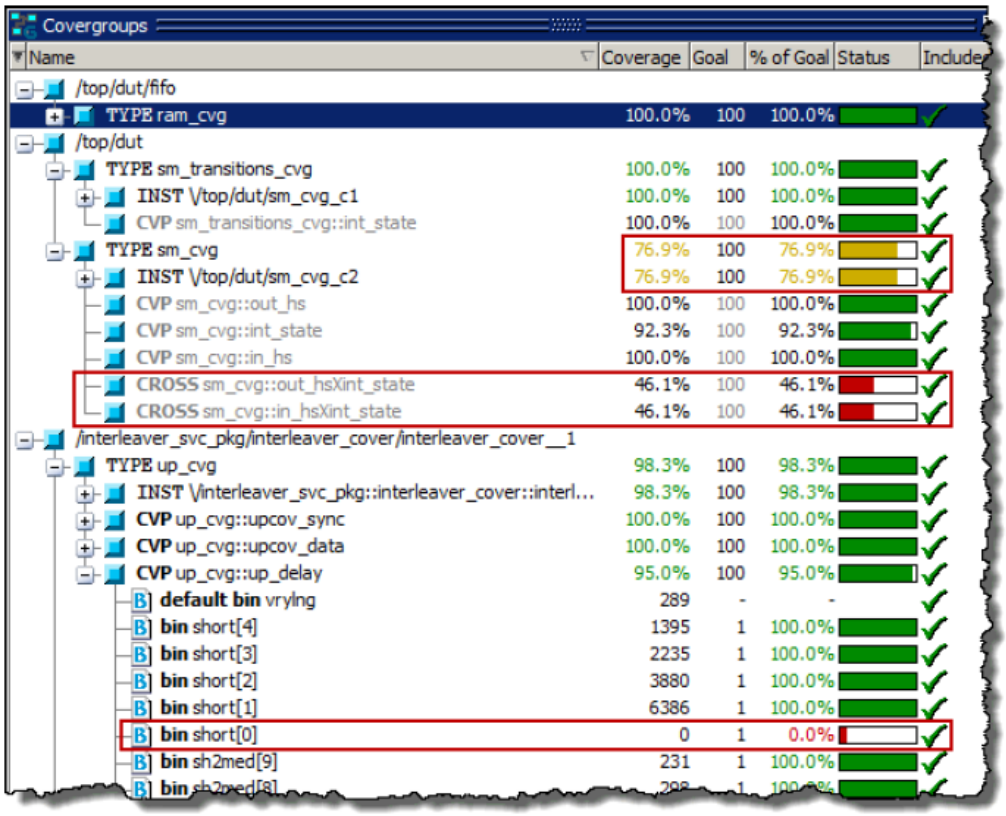
Figure 15-27. Scoreboard Information in the Transcript



C. Covergroups window 의 functional coverage 정보를 확장합니다. (창 오른쪽 하단의 상태 표시줄에서와 같이) interleaver 디자인의 전반적인 covergroup 의 coverage 는 거의 95% 입니다. No hit covergroup 은 up_delay 이며, 1 번의 적은 빈도수가 있습니다. 현재 드라이버는 패킷의 payload data 를 동작하는 한 단어 사이에 적어도 하나의 cycle 을 insert 합니다.

sm_cvg 는 in_hsXint_state 와 out_hsxint_state 로 인해 적은 coverage(76.9%) 를 보여줍니다. in_hs 신호는 idle, load_bypass 일 때, 또는 10 개의 state 중 하나의 load state 일 때 assert 하며, send_bypass 나 10 개 state 중 하나를 보낼 때 out_hs 신호를 assert 합니다. 이러한 빈도수가 테스트를 더 필요로 하는 영역을 가리키도록 나타날 수 있지만, coverage 의 부족은 실제로 적절한 동작이 이루어졌음을 보여주고 있습니다.

Figure 15-28. Covergroup Coverage in the Cover Groups Window

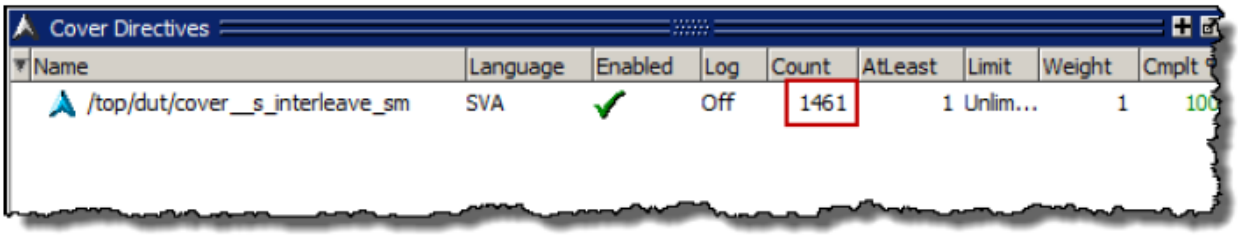


유저 분은 covergroup 의 sm_trasitions_cvg 를 확장해서 보면 1461 번 interleaver state transition 된 것을 볼 수 있습니다.(idle loop 일 때 86번, bypass loop 일 때 1375 번)

D. Cover Directives Tab 열기.

Cover directive count 를 보면 1461 번 transition 이 된 것을 확인 할 수 있습니다.

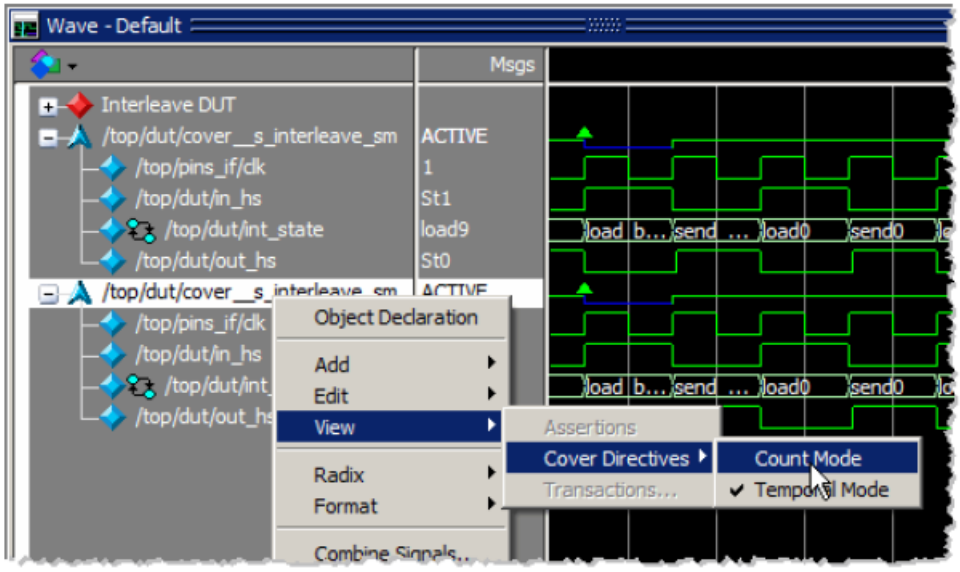
Figure 15-29. Cover Directive Counts State Transitions



8. Wave Window 에서 Temporal to count Mode 변경하기.

A. 두 번째 directive 를 선택한 후 오른쪽 마우스 버튼을 클릭하여 View>Cover Directives>Count mode 를 선택합니다.

Figure 15-30. Changing the Cover Directive View to Count View



B. Wave Window 를 통해 Cover Directive 를 확인합니다.

아래에 두 개의 Cover Directive 이미지가 있습니다. 위의 그림은 temporal mode 로 보았을 때 이며, 아래의 그림은 count mode 로 보았을 때 입니다. 두 이미지를 비교해 보면 다른 부분을 쉽게 확인 하실 수 있습니다. 위의 그림은 시작과 끝 사이가 1240 ns 이며, 아래의 그림에서는 780 ns 를 확인 할 수 있습니다.

Figure 15-31. First Temporal and Count Mode Views of Cover Directive

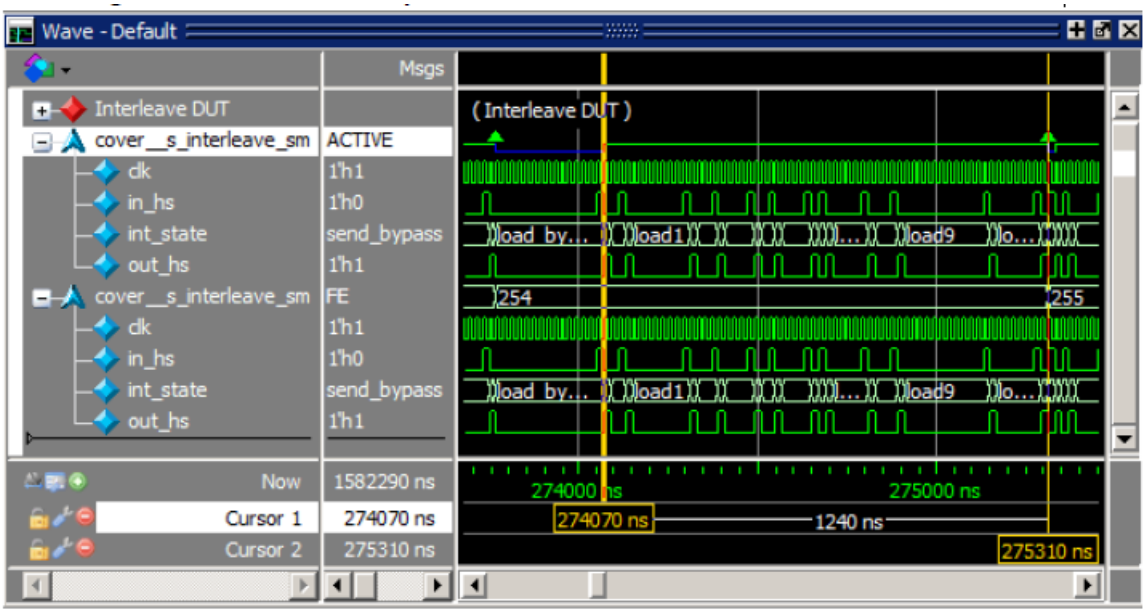
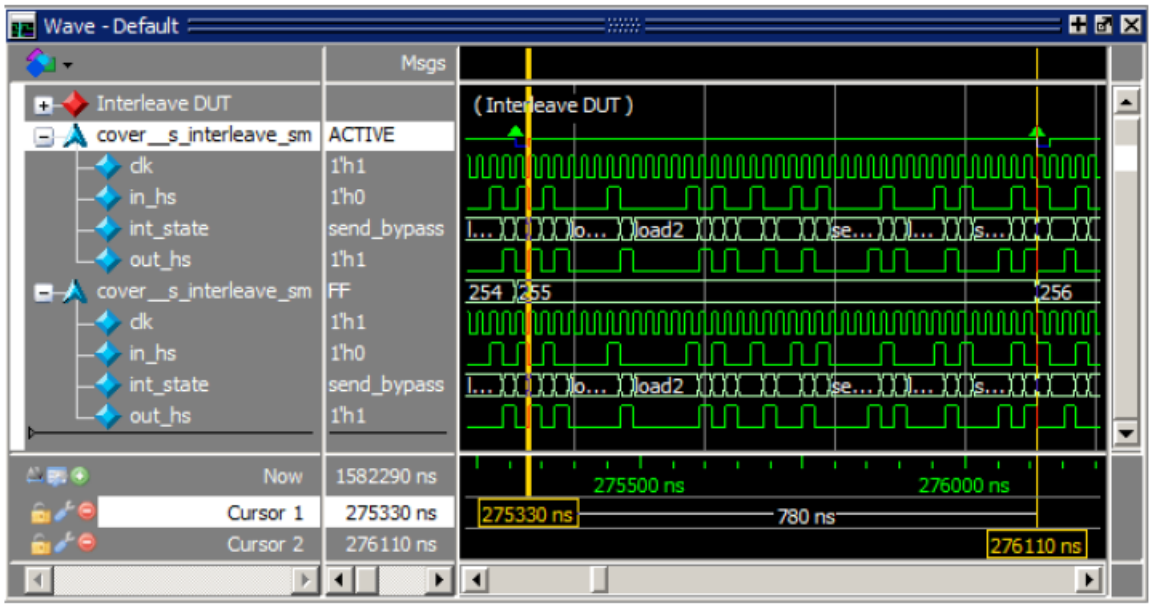


Figure 15-32. Second Temporal and Count Mode Views of Cover Directive



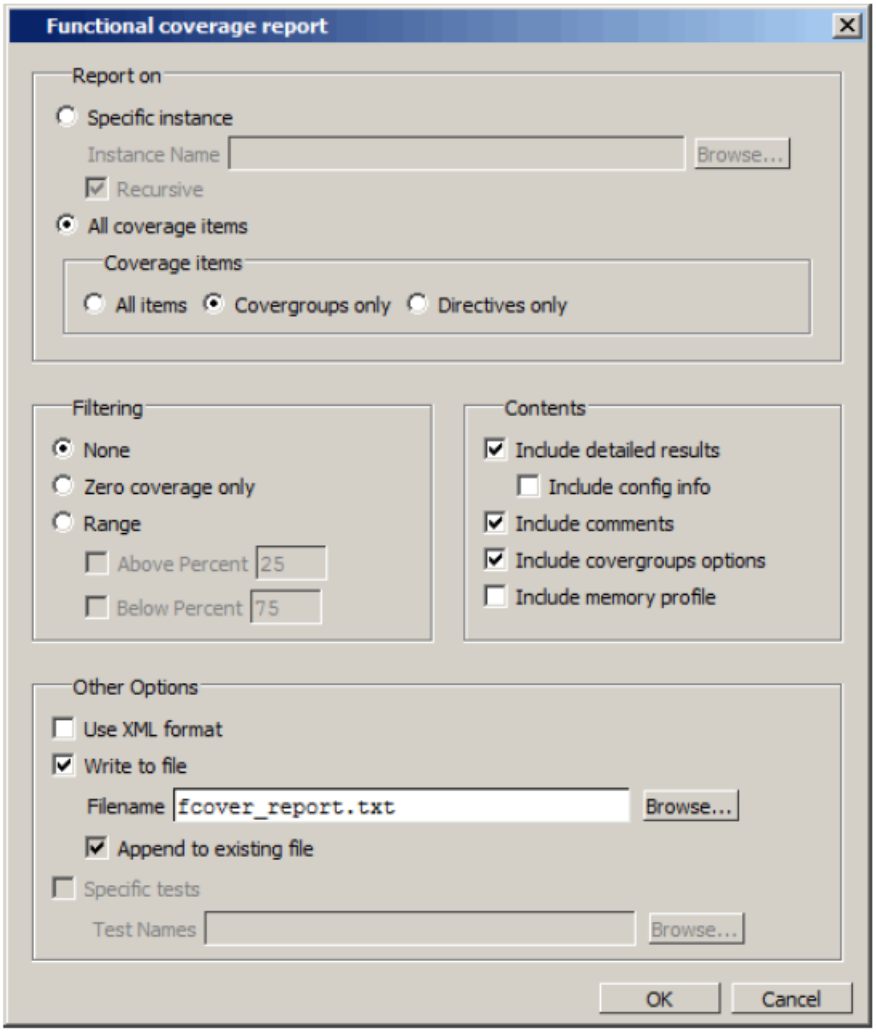
Creating Functional Coverage Reports

유저께서는 GUI 혹은 command 를 통해 functional coverage report 를 생성 할 수 있습니다.

1. GUI 를 통한 functional coverage report 생성하기.

   A. Coverage Directives Window 에서 오른쪽 마우스를 클릭하여 Report 를 선택합니다. 그러면 functional coverage report dialog box 가 열립니다.

Figure 15-33. Functional Coverage Report Dialog Box



B. Report on 항목에서 All coverage items 를 선택 한 후 Covergroups only 로 변경합니다.

C. Contents 항목에서 Include Covergroups options 을 체크합니다.

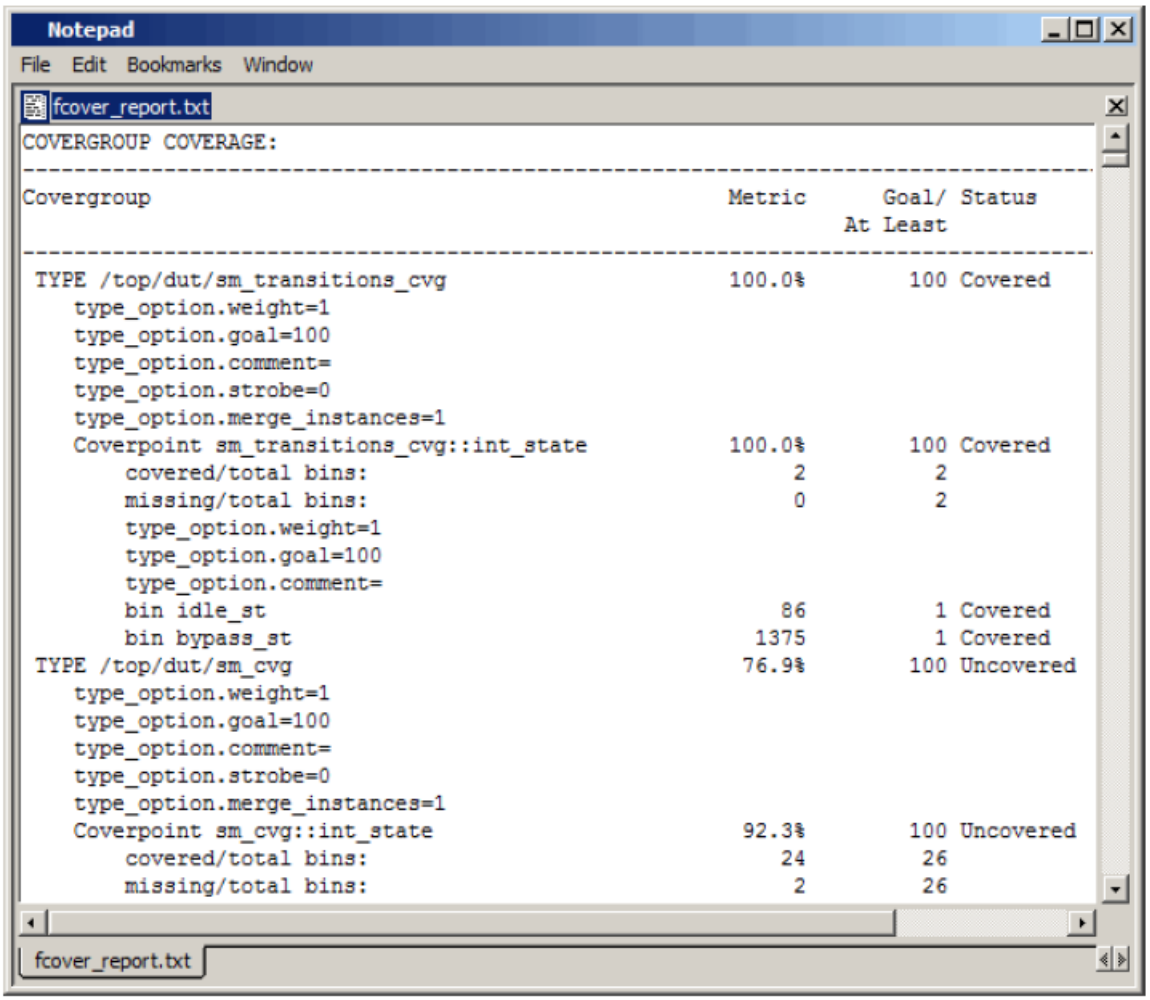
D. OK 를 선택하여 fcover_report.txt 파일을 만듭니다.

GUI 에서 진행한 내용은 Transcript Window 에 나타납니다.

```
coverage report -detail -cvg -comments –option
                -file fcover_report.txt -r /
```

Notepad 를 통해 만든 report 를 확인합니다.

Figure 15-34. The Functional Coverage Report



또한 Tools>Coverage>Report 메뉴를 선택하여, textual, html, exclusion coverage report 를 생성 할 수 있습니다.

File>Quit 를 통해 종료합니다.

Chapter 16
Using the SystemVerilog DPI

이번 챕터에서는 SystemVerilog 를 Direct Programming Interface(DPI) 를 사용하여 설계하는 기본 방법에 대해 알아보도록 하겠습니다. foreign language 로 작성된 code 와 Verilog Simulation 을 Simulation control flow 방법으로 작은 디자인부터 시작하여 보도록 하겠습니다. 일반적으로 Verilog Simulation 과 인터페이스를 하는데 사용하는 foreign language 는 C 로 작성된 코드를 사용합니다.

디자인은 교차로를 만들도록 하겠습니다. GUI 를 통해 디자인을 불러오고 신호등을 나타내는 신호의 waveform 을 모니터 하겠습니다. 시뮬레이션을 실행하고 Verilog 와 C 로 작성된 함수를 호출하여 빛의 색상을 변경하는 방법을 해보겠습니다.

이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

<install_dir>/examples/tutorials/systemverilog/dpi_basic

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Verilog Interfaces to C 를 보시면 더욱 자세히 알 수 있습니다.

Examine the Source Files

시작하기 전에 주요 디자인 source file 을 살펴보아 simulation flow 를 확인하고, DPI 에 대한 기본 요구사항 일부를 보도록 하겠습니다.

Test editor 를 통해 tset.sv 를 열어 확인을 해보도록 하겠습니다.

Figure 16-1. Source Code for Module *test.sv*

```
 1 module test ();
 2
 3 typedef enum {RED, GREEN, YELLOW} traffic_signal;
 4
 5 traffic_signal light;
 6
 7 function void sv_GreenLight ();
 8 begin
 9       light = GREEN;
10 end
11 endfunction
12
13 function void sv_YellowLight ();
14 begin
15       light = YELLOW;
16 end
17 endfunction
18
19 function void sv_RedLight ();
20 begin
21       light = RED;
22 end
23 endfunction
24
25 task sv_WaitForRed ();
26 begin
27       #10;
28 end
29 endtask
30
31 export "DPI-C" function sv_YellowLight;
32 export "DPI-C" function sv_RedLight;
33 export "DPI-C" task sv_WaitForRed;
34
35 import "DPI-C" context task c_CarWaiting ();
36
37 initial
38 begin
39       #10 sv_GreenLight;
40       #10 c_CarWaiting;
41       #10 sv_GreenLight;
42 end
43
44 endmodule
45
```

Line 1 – test 라는 top-level module 을 사용하여 모든 시뮬레이션을 진행 할 것입니다.

Line 3 – 데이터 값 빨간색, 녹색, 노란색을 포함하는 traffic_signal 이라는 새로운 데이터 유형을 선언합니다.

Line 5 – 새로운 traffic_signal 타입의 object 를 선언하고, 이름을 light 로 합니다.

Line 7-11 – 리턴 값이 없는 sv_GreenLight 라는 Verilog Functional 을 정의 합니다. 단순히 녹색 값으로 light 를 설정합니다. 순서대로 function 이름에 sv_ 를 앞에 접두사로 두고 C 의 Functions 와 SystemVerilog 의 tasks/functions 를 구별합니다.

Line 13-17 – sv_YellowLight 라는 function 을 선언하여, 노란색으로 light 를 변경합니다.

Line 19-23 – sv_RedLight 라는 function 을 선언하여, 빨간색으로 light 를 변경합니다.

Line 25-29 – Verilog Task 인 sv_WaitForRed 는 단순히 10 ns Delay 합니다.

Line 31-33 – 이 라인은 Verilog code 로 보이지 않습니다. 몇 가지 추가 정보를 가지고, "export" 키워드로 시작합니다. 이 부분은 export 선언문입니다. Verilog Compiler 를 알리기 위한 기본 메커니즘은 특별한 방식으로 처리할 필요가 있습니다. DPI 의 경우 특별한 처리는 지정된 task 또는 function 이 foreign language 로 특별한 이름이 공간에 배치되어야 표시할 수 있는 것을 의미합니다.

이 선언문은 SystemVeriog 의 LRM 에 정의되어 있습니다. 작동 방법은 간단한 규칙에 있습니다.

SystemVerilog 의 시뮬레이션을 실행하고 foreign(C) code 를 이용하기 위해 DPI 를 사용해야 하는 경우, Verilog Code(즉 모든 Verilog 코드 중심으로 전개)를 중심으로 간주 되어야 합니다.

이 라인에서 GreenLight function 을 뺀 두 가지의 신호를 보냅니다.

Line 35 – Import declaration 은 Verilog World 로 foreign(c) world 의 코드를 불러오는데 사용합니다.

이 경우 C world 에서 c_CarWaiting 이라는 이름의 task 를 가져옵니다.(task/function 이 발생한 위치를 추적 할 수 있도록 C_ 접두사를 참고) 이것은 중요한 개념입니다. 만약 foreign task/function 을 호출하지만, task/function 에 해결되지 않은 참조를 할 경우 시뮬레이션을 불러올 때 에러가 표시됩니다.

Line 37-42 – 시뮬레이션을 실행하고 light 변화 시나리오를 통해 작은 초기블록을 사용합니다. 기본적으로 light 시작은 RED out 을 기본으로 하며, light type 정의가 첫 번째 값입니다. 시뮬레이션이 시작되면, 10 ns 를 기다린 후 sv_GrrenLight 기능을 통해 녹색으로 light 를 변경합니다. Verilog world 에서 발생하여, sv_GreenLight function 을 export 할 필요가 없습니다. Foreign world 는 아무것도 수행되지 않습니다.

다음으로 10 ns 를 기다린 후 c_CarWaiting 을 호출합니다. Import declaration 의 이전 discussion 에서 Verilog task 를 import 합니다. C function 을 알고 있습니다. 작업을 호출 할 때 실제로 Foreign world 를 통해 stepping 으로 일부 C code 를 검사해야 합니다. 다른 소스 파일을 통해 시뮬레이션이 진행되는 동안 무슨 일이 일어나는지 확인을 합니다.

foreign.c 를 text editor 를 통해 열어보도록 하겠습니다.

Figure 16-2. Source Code for the *foreign.c* File - DPI Lab

```
1 int c_CarWaiting()
2 {
3    printf("There's a car waiting on the other side. \n");
4    printf("Initiate change sequence ...\n");
5       sv_YellowLight();
6       sv_WaitForRed();
7       sv_RedLight();
8       return 0;
9 }
10
```

Line 1 – c_CarWaiting 에 대한 function 정의입니다. 이것은 int 형으로 0을 반환합니다.

Line 3-4 – 함수 내부의 문은 자동차가 교차로의 반대편에서 대기하고 우리가 light 의 변화 순서를 시작해야 한다는 것을 나타내는 메시지를 출력합니다.

Line 5 – SystemVerilog 의 sv_YellowLight Function 을 호출 합니다. 이 함수가 종료되면 반환 될 때까지 Verilog 로 제어를 할 수 있습니다. Veriog world 를 호출하고 거기에서 task/function 을 실행 할 수 있습니다. export 선언과 Verilog code 를 내보내 C code 가 sv_YellowLight 의 존재를 알고 있습니다.

시뮬레이션을 test.sv 파일에서 13 ~ 17 번 라인의 sv_YellowLight function 과 함께 진행합니다. YELLOW 의 값으로 light 를 변경 한 후 foreign.c 제어를 전달하고 sv_YellowLight function 을 호출 한 후 다음 라인으로 이동합니다.

Figure 16-3. The sv_YellowLight Function in the *test.sv* File

```
13 function void sv_YellowLight ();
14 begin
15       light = YELLOW;
16 end
17 endfunction
```

Line 6 – test.sv 의 25~29 번 라인에 정의 된 sv_WaitForRed 의 SystemVerilog Task 를 호출합니다.

Figure 16-4. The sv_WaitForRed Task in the *test.sv* File

```
25 task sv_WaitForRed ();
26 begin
27       #10;
28 end
29 endtask
```

이 task 구문은 는 10 ns delay 를 지정합니다. 이 절차와 관련된 시간 delay 가 존재하기 때문에 task 구문이 있어야 합니다. foreign world 에서 호출하는 경우 기본 Verilog 의 task 와 function 에 관련된 모든 규칙이 적용됩니다. 독립적으로 두 개의 소스파일 (Verilog compiler 하나와 C compiler 하나)을 컴파일 한 이후에 한 언어의 규칙은 다른 컴파일러가 알 수 없습니다. Import/export 를 결정할 때 주의하여야 합니다.

여기에서 주의해야 할 중요한 부분은 foreign(C) world 에서 SystemVerilog sv_WaitForRed() 작업을 호출 한 것입니다. 시뮬레이션 시간을 소비할 경우 C 는 SystemVerilog 를 디자인이나 시뮬레이션 시간 단위에 대해 아무것도 알 수가 없습니다. 그래서 이러한 동작을 수행하기 위해 다시 Verilog 를 통해 호출을 해야 합니다.

sv_WaitForRed는 시뮬레이션의 10 ns 를 가지며, foreign.c 다음 행으로 진행합니다. 그리고 C 를 다시 제어하여 반환합니다.

Line 7 – light 를 RED 로 변경하기 위해 sv_RedLight SystemVerilog Function 을 호출합니다. test.sv 의 Function 을 보도록 하겠습니다.

Figure 16-5. The *sv_RedLight* Function in the *test.sv* File

```
19 function void sv_RedLight ();
20 begin
21        light = RED;
22 end
23 endfunction
```

foreign.c 에서 c_CarWaiting Function 의 마지막 구문입니다. 함수가 종료되면 반환되어 Verilog 를 제어합니다.

시뮬레이터는 처음에 C function 을 호출하며 test.sv 40 번 라인으로 반환합니다. 이 라인에서는 아무것도 수행하지 않습니다. 시뮬레이션 실행의 다음 행으로 내려갑니다. 10 ns 를 기다린 후 sv_GreenLight function 을 호출합니다. 이 기능은 Verilog world 로 변경되며 lignt 가 녹색으로 변경됩니다. 그러면 시뮬레이션이 모두 종료됩니다.

Figure 16-6. Function Calls in the *test.sv* File

```
37 initial
38 begin
39        #10 sv_GreenLight;
40        #10 c_CarWaiting;
41        #10 sv_GreenLight;
42 end
```

Exploring the Makefile

Makefile 은 Unix 와 Linux 사용자가 컴파일부터 디자인을 시뮬레이션 할 때, 한 번의 명령어를 통해 모든 과정을 실행하는데 도움이 됩니다. Makefile 을 통해 처음부터 실행하거나, 다시 실행할 경우 설정된 환경을 바로 사용할 수 있는 장점이 있습니다.

Figure 16-7. Makefile for Compiling and Running on UNIX or Linux Platforms

```
 1 worklib:
 2    vlib work
 3
 4 compile: test.sv
 5    vlog test.sv -dpiheader dpi_types.h
 6
 7 foreign: foreign.c
 8    gcc -I$(QUESTA_HOME)/include -shared -g -o foreign.so foreign.c
 9
10 foreign_32:  foreign.c
11    gcc -I$(QUESTA_HOME)/include -shared -fPIC -m32 -g -o foreign.so
foreign.c
12
13 optimize:
14    vopt +acc test -o opt_test
15
16 foreign_windows: foreign.c
17    vsim -c opt_test -dpiexportobj exports
18    gcc -I$(QUESTA_HOME)/include -shared -g -o foreign.dll foreign.c
eports.obj -lmtipli -L$(QUESTA_HOME)/win32
19
20 sim:
21    vsim opt_test -sv_lib foreign
22
23 all:
24    worklib compile foreign optimize sim
25
26 all_windows:
27    worklib compile optimize foreign_windows sim
28
29 clean:
30    rm -rf work transcript vsim.wlf foreign.so foreign.dll exports.obj
31
```

Makefile 을 확인 해 보겠습니다.

Line 1-2 – vlib command 는 컴파일 된 파일이 위치 할 work 라이브러리를 만듭니다.

Line 4-5 – vlog command 로 test.sv 파일을 컴파일 합니다.

Line 7-11 and 16-18 – gcc command 를 통해 foreign.c 를 컴파일 할 컴파일러를 호출하고 시뮬레이션 하는 동안 로드 됩니다.

Line 13-14 – vopt command 로 디자인을 최적화합니다. +acc 옵션을 통해 Full Visibility 를 통해 디자인을 디버깅 할 수 있습니다. –o 옵션을 통해 최적화 된 디자인의 이름을 적용합니다.

Line 20-21 – vsim Command 를 통해 최적화 된 디자인인 opt_test 를 시뮬레이터를 통해 실행합니다. -sv_lib 옵션을 통해 shared object 를 시뮬레이션 중에 불러옵니다. 옵션을 사용하지 않으면 시뮬레이터는 미리 정의한 C Function 을 찾을 수 없습니다.

Exploring the *windows.bat* File

windows.bat file 은 window 사용자가 사용할 수 있습니다.

Figure 16-8. The windows.bat File for Compiling and Running in Windows - DPI Lab

```
 1 vlib work
 2
 3 vlog test.sv -dpiheader dpi_types.h
 4
 5 vopt +acc test -o opt_test
 6
 7 vsim -c test -dpiexportobj exports
 8
 9 gcc -I %QUESTA_HOME%\include -shared -g -o foreign.dll foreign.c
exports.obj -lmtipli -L %QUESTA_HOME%\win32
10
11 vsim -i opt_test test -sv_lib foreign -do "add wave light; view source"
12
```

Windows.bat 파일을 확인 해 보겠습니다.

Line 1 – vlib command 는 컴파일 된 파일이 위치 할 work 라이브러리를 만듭니다.

Line 3 – vlog command 로 test.sv 파일을 컴파일 합니다.

Line 5 – vopt command 로 디자인을 최적화합니다. +acc 옵션을 통해 Full Visibility 를 통해 디자인을 디버깅 할 수 있습니다. –o 옵션을 통해 최적화 된 디자인의 이름을 적용합니다.

Line 7 – vsim command 에서 gcc command 를 사용하여 exports 를 object 를 불러옵니다.

Line 9 – gcc command 로 이전 command 를 통해서 만든 exports.obj 과 foreign.c 소스 파일을 compile 과 link 합니다. –o 옵션을 통해 foreign.dll 을 Output library 에 만듭니다.

Line 11 – vsim Command 를 통해 최적화 된 디자인인 opt_test 를 시뮬레이터를 통해 실행합니다. -sv_lib 옵션은 SystemVerilog 를 시뮬레이션에 사용할 수 있도록 C 디자인에 대한 foreign.dll library 를 Simulator 에서 볼 수 있도록 하는 옵션입니다. –do “add wave light; view source” 옵션을 통해 light 신호를 wave window 에 추가하고, source window 를 띄웁니다.

Compile and Load the Simulation

이 과정을 진행하기 위해 컴파일을 한 후 environment 변수를 설정하고 디자인을 로드해야 합니다. 이 경우 컴파일과 로드가 한번에 수행됩니다.

1. 새로운 디렉토리를 만들고 tutorial file 복사하기.

   <install_dir>/questasim/examples/tutorials/systemverilog/dpi_basic

2. 새로운 디렉토리로 디렉토리를 변경한 후 environment 변수 설정하기.

   A. Environment 변수는 questa 설치 디렉토리에 있습니다.

   B. 윈도우를 사용하는 경우 gcc-4.2.1-mingw32vc9 compiler 가 설치되어 있지 않은 경우 SupportNet (http://supportnet.mentor.com/) 에서 다운로드 받으실 수 있습니다. 설치는 Questa 설치 디렉토리에 하며, 환경변수 Path 설정을 해주어야 합니다.
   C:₩<Install_directory>₩gcc-4.2.1-mingw32vc9₩bin

3. Unix 와 Linux : make utility 를 사용하여 compile 과 디자인 로드를 진행합니다.

   Windows : windows.bat 를 더블 클릭하여 compile 과 디자인 로드를 진행합니다.

```
vlib work
vlog test.sv -dpiheader dpi_types.h foreign.c
vopt +acc test -o opt_test
vsim -i opt_test -do "add wave light; view source"
```

Run the Simulation

Object Window 의 light object 가 어떻게 되는지 시뮬레이션을 진행 해보겠습니다.

1. Unix and Linux : 드래그 앤 드롭을 통해 Wave Window 에 light object 를 추가합니다.

   Windows : Wave Window 에 이미 light object 가 추가되어 있습니다.

2. 시뮬레이션 모드에서 단계별로 시뮬레이션을 진행할 수 있습니다. 10 ns 단위마다 light 신호의 파형이 어떻게 변하는지 살펴 보도록 하겠습니다. Object Window 를 통해 light object 의 초기 값(RED)을 볼 수 있습니다. 만약 object Window 가 열려 있지 않으면, View>Object 를 통해 열 수 있습니다.

Figure 16-9. The *light* Signal in the Objects Window

3. 10 ns 시뮬레이션 진행하기.

   A. Command 에 run 10 ns 를 입력합니다. Object 와 Wave Window 를 보면 light 값이 GREEN 인 것을 확인 할 수 있습니다.

   B. 여러 번 시뮬레이션을 진행하여 변경되는 light 값을 확인합니다.

Figure 16-10. The *light* Signal in the Wave Window

4. 시뮬레이션 Restart 하기.

   A. Restart icon 을 클릭합니다. 

   B. Restart dialog box 의 OK 버튼을 클릭합니다.

5. 10 ns 시뮬레이션 진행하기.

   A. Command 에 run 10 ns 를 입력합니다.

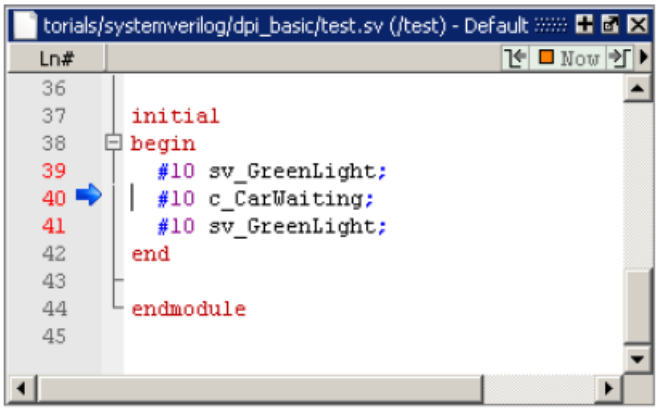
6. test.sv 를 source window 로 보기.

   A. test.sv tab 을 선택합니다.

7. Code 를 Step 으로 확인하기.

A. Step icon  을 클릭하여 source window 에서 test.sv 와 foreign.c code 에서 파란색 화살표가 움직이는 것을 확인합니다. 이것은 시뮬레이션을 단계별로 source file 이 있는 곳을 추적 할 수 있습니다.

Figure 16-11. Source Code for *test.sv*



8. 시뮬레이션 종료하기.

Simulation>End Simulation 을 선택하여 시뮬레이션을 종료합니다.

Chapter 17
Using SystemVerilog DPI for Data Passing

이번 챕터에서는 SystemVerilog 에 대한 Direct Programming Interface 에 대해 살펴볼 것입니다. 이전 챕터에서는 기본 인터페이스의 요소와 방법을 간단한 함수를 만들어 Verilog 와 C 를 호출하였습니다. 이번 챕터에서는 Interface 에 초점을 맞추어 살펴보도록 하겠습니다.

DPI 또한 foreign language 와 마찬가지로 Verilog 를 Interface 할 수 있으며, 이 과정에서는 C 를 맞추어 진행됩니다.

Mapping Verilog and C

C 에서 Verilog 또는 Verilog 에서 C 로 object 값을 보낼 때 마다 그 값은 이중성을 가지게 됩니다. 값을 보낼 때는 함수 호출을 통해 초기화 한 후 값을 주고 받습니다.

C 타입과 Verilog 타입을 매핑하는 테이블이 필요하게 됩니다. 다행히 Verilog 와 SystemVerilog 에 들어가는 대부분의 data 타입은 C 타입과 일치하는 것을 목적으로 되어 매핑이 간단합니다. 그러나 매핑 중 일부는 좀더 복잡하여 Verilog object 가 C 에 매핑 되는 방법을 알고 있어야 합니다.

SystemVerilog 는 유저를 위해 mapping 을 정의하고, 시뮬레이터는 이러한 이중성을 모두 처리하도록 설정되어 있어 따로 매핑을 할 필요가 없습니다. 예를 들어 Verilog 에서 32 bit 메모리에 저장되어 있는 2-state int 인 경우, 값은 0 과 1 로 할당 되지만 X 또는 Z 는 허용되지 않습니다.

그렇기 때문에 C 의 int 처럼 간단하게 매핑을 할 수 있습니다.

이 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

<install_dir>/examples/tutorials/systemverilog/data_passing

1. Environment 변수는 questa 설치 디렉토리에 있습니다.

2. 윈도우를 사용하는 경우 gcc-4.2.1-mingw32vc9 compiler 가 설치되어 있지 않은 경우 SupportNet (http://supportnet.mentor.com/) 에서 다운로드 받으실 수 있습니다. 설치는 Questa 설치 디렉토리에 하며, 환경변수 Path 설정을 해주어야 합니다.
   C:₩<Install_directory>₩gcc-4.2.1-mingw32vc9₩bin

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Verilog Interfaces to C 와 Verification with Functional Coverage 를 보시면 더욱 자세히 알 수 있습니다.

Examine the Source Files

시작하기 전에 두 개의 C function 에 대한 정의를 포함하는 foreign.c 파일을 살펴보도록 합니다.

Figure 17-1. Source Code for the *foreign.c* File - Data Passing Lab

```
1  #include "dpi_types.h"
2
3  void print_int(int int_in)
4  {
5      printf("Just received a value of %d.\n", int_in);
6  }
7
8  void print_logic(svLogic logic_in)
9  {
10     switch (logic_in)
11     {
12         case sv_0: printf ("Just received a value of logic 0.\n");
13             break;
14         case sv_1: printf ("Just received a value of logic 1.\n");
15             break;
16         case sv_z: printf ("Just received a value of logic Z.\n");
17             break;
18         case sv_x: printf ("Just received a value of logic X.\n");
19             break;
20     }
21 }
22
```

Line 1 – dpi_types.h 라는 헤더파일을 포함하고 있습니다.

Line 3 – 정수를 출력하는 print_int function 에 대한 정의입니다.

Line 8 – svLogic 에 따라 메시지를 출력하는 print_logic function 에 대한 정의입니다.

이제 System Verilog 의 Source Code 를 살펴보도록 하겠습니다. Text editor 를 이용하여 test.sv 를 확인해 보겠습니다.

Figure 17-2. Source Code for the *test.sv* Module

```
1  module test ();
2
3  import "DPI-C" context function void print_int (input int int_in);
4  import "DPI-C" context function void print_logic (input logic logic_in );
5
6  int int_var;
7  bit bit_var;
8  logic logic_var;
9
10 initial
11 begin
12        print_int(int_var);
13        int_var = 1
14        print_int(int_var);
15        int_var = -12;
16        print_int(int_var);
17        print_int(bit_var);
18        bit_var = 1'b1;
19        print_int(bit_var);
20        bit_var = 1'bx;
21        print_int(bit_var);
22        logic_var = 1'b1;
23        print_int(logic_var);
24        logic_var = 1'bx;
25        print_int(logic_var);
26        print_logic(logic_var);
27        logic_var = 1'bz;
28        print_logic(logic_var);
29        logic_var = 1'b0;
30        print_logic(logic_var);
31 end
32
33 endmodule
34
```

Line 3-4 – 이 라인은 Verilog code 처럼 보이지 않습니다. 이 문은 import 선언 문 입니다. Import declaration 은 Verilog 에서 특별하게 취급해야 할 필요가 있으므로 compiler 를 알리기 위해 사용됩니다. DPI 의 경우 특별한 처리는 지정된 task/function 이 foreign language 에서 SystemVerilog 가 visible 되도록 이 이름이 특별한 이름공간에 배치될 필요가 있습니다.

이 선언문은 SystemVeriog 의 LRM 에 정의되어 있습니다. 작동 방법은 간단한 규칙에 있습니다.

SystemVerilog 의 시뮬레이션을 실행하고 foreign(C) code 를 이용하기 위해 DPI 를 사용해야 하는 경우, Verilog Code(즉 모든 Verilog 코드 중심으로 전개)를 중심으로 간주되어야 합니다.

Verilog Code 는 foreign world 에서 먼가 access 를 하기 위해서는 "import" 가 필요합니다. foreign world 에서 Verilog 를 access 를 할 때도 마찬가지로 "export" 가 필요합니다. 그래서 이 라인에 prit_int & print_logic 을 통해 import 와 export 를 합니다.

Line 6-8 – 두 function 을 가지는 세 변수를 선언 합니다. int, bit, logic 이 세 변수는 SystemVerilog 타입으로 정의하는 방법을 참고합니다.

Line 10-31 – 이 초기 블록은 단순히 각 function 을 호출하고 우리가 디자인을 실행할 때 동작을 하게 됩니다. 각 변수 값에 대해 설정합니다.

Compile and Load the Simulation

이 과정을 진행하기 위해 컴파일을 한 후 environment 변수를 설정하고 디자인을 로드 해야 합니다. 이 경우 컴파일과 로드가 한번에 수행됩니다.

1. 새로운 디렉토리를 만들고 tutorial file 복사하기.

   <install_dir>/questasim/examples/tutorials/systemverilog/dpi_basic

2. 새로운 디렉토리로 변경한 후 environment 변수 설정하기.

3. Unix 와 Linux : make utility 를 사용하여 compile 과 디자인 로드를 진행합니다.

   Window : windows.bat 를 더블 클릭합니다.

Explore the Makefile

Makefile 은 Unix 와 Linux 사용자가 컴파일하고 디자인을 시뮬레이션을 실행하는데 필요한 정보가 이 단원에 포함되어 있습니다.

Figure 17-3. Makefile for Compiling and Running on UNIX and Linux Platforms

```
1  worklib:
2      vlib work
3
4  compile: test.sv
5      vlog test.sv -dpiheader dpi_types.h
6
7  foreign: foreign.c
8      gcc -I$(QUESTA_HOME)/include -shared -g -o foreign.so foreign.c
9
10 optimize:
11     vopt +acc test -o opt_test
12
13 sim:
14     vsim opt_test test -sv_lib foreign
15
16 all:
17     worklib compile foreign optimize sim
18
19 clean:
20     rm -rf work transcript vsim.wlf foreign.so dpi_types.h
21
```

Makefile 의 다섯가지 목표입니다.

Line 1-2 – vlib command 를 통해 컴파일 된 파일이 위치할 work library 를 생성합니다.

Line 4-5 – vlog command 를 통해 test.sv source file 을 컴파일 합니다.

Line 7-8 – gcc command 를 통해 foreign.c source file 을 컴파일 하며 시뮬레이션을 하는 동안 로드 됩니다. 공유 객체(foreign.so) 를 작성합니다.

Line 10-11 – vopt command 를 통해 디자인을 최적화 합니다. +acc 옵션을 통해 full visibility 로 디버깅을 할 수 있습니다. –o 옵션을 통해 최적화 된 디자인 object 의 이름을 변경 할 수 있습니다.

Line 13-14 – vsim Command 를 통해 최적화된 디자인을 사용하여 simulator 를 호출합니다. –sv_lib 옵션은 시뮬레이션 하는 동안 불러올 수 있는 공유 객체를 지정합니다. 이 옵션을 사용하지 않으면, C function 을 불러올 수 없습니다.

Explore the *windows.bat* File

Windows.bat file 은 Window 사용자에 포함되어 있습니다.

Figure 17-4. The *windows.bat* File for Compiling and Running in Windows - Data Passing Lab

```
 1 vlib work
 2
 3 vlog test.sv -dpiheader dpi_types.h
 4
 5 vopt +acc test -o opt_test
 6
 7 gcc -I %QUESTA_HOME%\include -shared -g -o foreign.dll foreign.c -lmtipli -L
%QUESTA_HOME%\win32
 8
 9 vsim -i opt_test -sv_lib foreign -do "view source"
10
```

Window.bat file 은 다음과 같이 컴파일과 시뮬레이션을 진행합니다.

Line 1 – vlib command 를 통해 컴파일 된 파일이 위치할 work library 를 생성합니다.

Line 3 – vlog command 를 통해 test.sv source file 을 컴파일 합니다.

Line 5 – vopt command 를 통해 디자인을 최적화 합니다. +acc 옵션을 통해 full visibility 로 디버깅을 할 수 있습니다. –o 옵션을 통해 최적화 된 디자인 object 의 이름을 변경 할 수 있습니다.

Line 7 – gcc command 를 통해 foreign.c source file 을 컴파일 하는데 사용합니다. –I 옵션을 사용하여 include 파일을 검색 할 디렉토리를 지정합니다. –shared 옵션은 출력으로 공유라이브러리를 생성하는 gcc 를 알려줍니다. –g 옵션은 출력에 디버깅 code 를 추가 합니다. –o 옵션은 foreign.dll 이라는 출력 라이브러리를 작성합니다. –lmtipli 옵션은 C

에서 사용되는 모든 기능들을 해결하기 위해 시도 할 때 C/C++ 을 포함하는 컴파일 된 라이브러리를 지정하기 위해 사용됩니다. –L 옵션은 –l 옵션에 지정된 라이브러리를 검색할 디렉토리를 지정합니다.

Line 9 – 두 번째 vsim Command 를 통해 최적화된 디자인인 opt_test 를 사용할 수 있도록 simulator 에 호출 합니다. –sv_lib 옵션을 통해 SystemVerilog Simulation 을 사용할 수 있도록 C design object 에 사용하는 foreign.dll library 를 볼 수 있습니다. –do "add wave light; view source" 옵션을 통해 light 신호를 Wave Window 에 추가하고 source Window 를 통해 볼 수 있습니다.

Compile and Load the Simulation

컴파일과 디자인을 시뮬레이터를 통해 불러오기 위해 Unix 와 Linux 는 Makefile 을 사용합니다. Window 사용자의 경우 windows.bat 파일을 사용하여 진행합니다.

1. 새로운 디렉토리를 만들고 tutorial file 복사하기.

   <install_dir>/questasim/examples/tutorials/systemverilog/data_passing

2. 새로운 디렉토리로 변경한 후 environment 변수 설정하기.

3. Unix 와 Linux : make utility 를 사용하여 compile 과 디자인 로드를 진행합니다.

   Windows : windows.bat 를 더블 클릭합니다.

```
vlib work
vlog test.sv -dpiheader dpi_types.h foreign.c
vopt +acc test -o opt_test
vsim -i opt_test -do "view source"
radix -symbolic
```

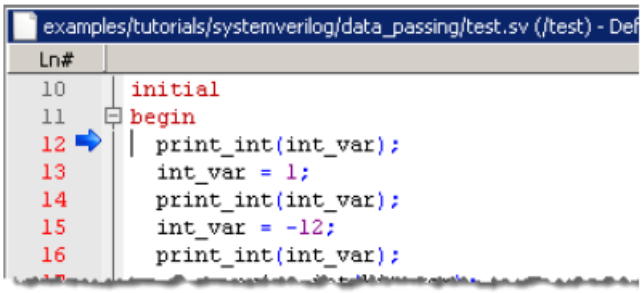
Run the Simulation

test.sv 모듈이 시뮬레이션에 로드 된 후에는 시뮬레이션을 통해 Step Over Command 를 버튼을 사용 할 수 있습니다. 이 것은 단순히 Verilog object 의 다른 유형의 값으로 설정하고 화면에 print out 을 위한 C 를 통해 데이터를 보냅니다.

1. Structure(Sim) window 에서 test instance 를 마우스 오른쪽 버튼으로 클릭하고, 나타나는 팝업메뉴에서 View Declaration 을 선택합니다. 그리고 source window 를 열어 test.sv 의 source code 를 확인합니다.

2. Step Over 버튼  을 클릭합니다. test.sv 의 12번 라인에 파란색 화살표가 표시됩니다.

Figure 17-5. Line 12 of *test.sv* in the Source Window



초기값 0 을 가져야 하므로 int_var 에 아직 값이 할당 되지 않았습니다. Object Window 를 보면 int_var 값이 0 인 것을 확인 할 수 있습니다.

Figure 17-6. The Value of int_var is Currently 0

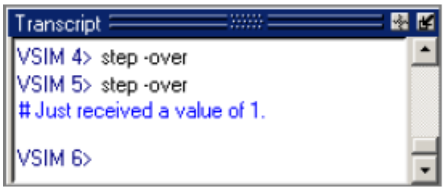
3. 다시 Step Over 버튼을 클릭합니다. 이 input parameter 로 int_var 를 가져온 C function print_int 를 호출합니다. 이 행이 실행 된 후에 Transcript Window 를 보면 다음과 같은 메시지를 확인 할 수 있습니다.

```
Just received a value of 0.
```

4. 다음 Step Over 버튼을 클릭하면 Object Window 에 int_var 값이 1로 변화되는 것을 확인 할 수 있습니다.

5. 더 Step Over 버튼을 클릭하면 Transcript Window 에 1 이 출력되는 것을 확인 할 수 있습니다.

Figure 17-7. The Value of *int_var* Printed to the Transcript Window



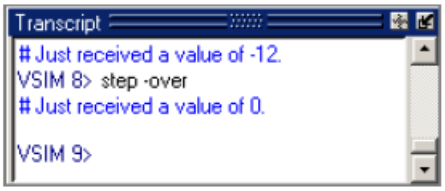
6. 다음 두 단계에서(Step Over 버튼을 2번 클릭) int_var 가 변경됩니다. Positive 와 negative integer 를 설정하고 올바르게 print 됩니다.

다음으로 7번 라인의 bit 타입으로 bit_var 의 값을 print_int 기능을 사용하여 출력합니다. 또한 0 의 초기값을 가지고 있기 때문에 print out 을 추측할 수 있습니다.

7. 다시 Step Over 버튼을 클릭하여 Transcript Window 를 확인합니다.

Figure 17-8. The Value of *bit_var* is 0.

Figure 17-9. Transcript Shows the Value Returned for *bit_var*



8. bit_var 를 1로 설정하고 transcript window 에서 Step Over 를 두 번 클릭합니다.

9. bit_var 를 X로 설정하고 Step Over 를 클릭합니다.

   값은 X 로 변경되지 않습니다.

10. print_int function 을 위해 Step Over 를 클릭하여 0 의 값이 print 되는지 확인합니다.

    4-state value 를 사용해봅니다. Logic 타입의 4-state 가 1이 22번 라인에 할당되어 있습니다.

11. Step Over 를 클릭하여 23 번 라인으로 이동합니다. Object Window 의 logic_var 를 보면 1 에서 X 로 변경된 것을 확인 할 수 있습니다.

12. Step Over 를 클릭하여 print_int 를 호출하고 Transcript window 를 통해 logic_var 의 값을 출력합니다.

13. logic_var 를 X 로 설정하고 Step Over 를 클릭합니다.

14. Step Over 를 클릭하여 logic_var 를 print 합니다. 26 번 라인에 화살표가 있어야 합니다. 0 값 대신 X 가 인쇄된 것을 확인 할 수 있습니다. Source Code 를 살펴보겠습니다.

    Import function 을 위한 C code 인 foreign.c 를 보면 3 번 라인에 print_int function 이 integer(int) 를 input 으로 하고 있습니다. 정수를 보내면 제대로 동작합니다. 이전에 X 와

Z 값을 가지는 4-state 타입으로 진행하였습니다.

SystemVerilog Language 는 데이터 매핑을 정의합니다. 컴파일 시 C 의 헤더 파일을 통해 모든 function prototypes, data type definitions 를 참조합니다.

Makefile 을 보면 compile 에 vlog command 에서 –dpiheader 를 호출하여 argument name 을 파일로 출력할 수 있습니다. Compiler 를 통해 Verilog source file 을 컴파일하고, DPI import/export 를 분석하며, imported/exported, functions/task 의 prototype 을 정의하여 C 헤더파일을 생성하고 이 과정에서 dpi_types.h 파일을 호출합니다.

Figure 17-10. The *dpi_types.h* File

```
1  /* MTI_DPI */
2
3  /*
4   * Copyright 2004 Mentor Graphics Corporation.
5   *
6   * Note:
7   *   This file is automatically generated.
8   *   Please do not edit this file - you will lose your edits.
9   *
10  * Settings when this file was generated:
11  *   PLATFORM = 'win32'
12  *       Info = SE 6.1c 2005.11
13  */
14 #ifndef INCLUDED_DPI_TYPES
15 #define INCLUDED_DPI_TYPES
16
17 #ifdef __cplusplus
18 extern "C" {
19 #endif
20
21 #include "svdpi.h"
22
23 DPI_DLLESPEC
24 void
25 print_int(
26     int int_in);
27
28 DPI_DLLESPEC
29 void
30 print_logic(
31     svLogic logic_in);
32
33 #ifdef __cplusplus
34 } /* extern "C" */
35 #endif
36
37 #endif  /* INCLUDED */
38
```

파일의 상단에 내부 DPI 의 정보를 볼 수 있습니다. 25번 라인으로 가면 print_int function 에 대한 function prototype 을 볼 수 있습니다. Input parameter 는 int 타입입니다.'

print_logic function 은 prototype 입니다. 이 파일은 SystemVerilog Language 일부이며, C 컴파일을 위한 명령어가 포함되어 있습니다. svLogic 타입은 unsigned char 입니다.

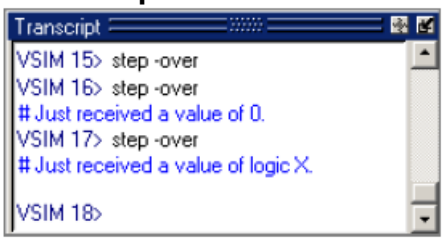
C source file에 #include dpi_type.h 를 넣었을 때, 모든 function prototype 및 data type 을 사용 가능하게 할 수 있습니다. DPI 를 통해 Verilog 와 C code 를 작성 할 때 이 파일을 사용하는 것이 좋습니다.

test.sv 파일에서 DPI import 구문을 다시 확인합니다. print_int 에 대한 print_logic 이 하나 있습니다. vlog compiler 는 이 구문에서 보이는 function 의 이름은 해당 parameter 와 반환 값을 함께 가져오고, 다음 DPI 헤더파일을 만듭니다. print_logic function 의 경우 input parameter 타입이 "logic" 입니다. 그래서 헤더파일에 "svLogic" 을 넣어 logic 에 대응합니다. 이제 특정 object 의 이중성을 정의하고 모든 것을 C 를 해 전달합니다.

시뮬레이션으로 돌아가서 26 번 라인에 데이터의 특정 타입에 대한 잘못된 기능을 사용하고 있어 0 에 X 값이 잘못 들어가는 것을 확인했습니다. 그래서 대신 print_logic function 을 사용합니다.

이 라인을 실행하기 위해 Step Over 를 클릭합니다. X 값이 print out 됩니다. foreign.c 파일을 확인 합니다.

Figure 17-11. The Transcript Shows the Correct Value of logic X



기본적으로 4-state 값들은 0, 1, 2, 3 과 같이 표현됩니다. 이것을 유지하기 위해 print_logic function 의 switch 문에서 참조 값은 svdpi.h 파일의 #define'd 입니다. C code 에서 DPI 헤더 파일을 사용하려면 이와 같이 사용할 수 있으며, 모든 것이 제대로 동작하는 것을 알 수 있습니다.

앞으로 이동한 후, 일부 구문을 step 을 통해 logic_var 를 다른 4-state 값으로 설정합니다. 그 후 print_logic 기능을 사용하여 print 하여 확인합니다.

Simulation>End Simulation 을 통해 종료합니다.

Chapter 18
Comparing Waveforms

이번 챕터에서는 Waveform compare 을 통해 기준이 되는 신호와 테스트 신호의 타이밍 차이를 계산하여 비교하는 것에 대해 살펴 보도록 하겠습니다.

- Simulation 또는 Dataset 을 선택하여 비교
- 특정 신호나 영역을 비교
- Compare 실행
- Compare 결과 보기

이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

Verilog – *<install_dir>/examples/tutorials/verilog/compare*
VHDL – *<install_dir>/examples/tutorials/vhdl/compare*

이번 챕터에서 진행되는 내용은 유저 매뉴얼의 Waveform Compare 와 Recording Simulation Results With Datasets 를 보시면 더욱 자세히 알 수 있습니다.

Creating the Reference Dataset

Reference Dataset 은 Waveform Log File 인 .wlf 확장자를 가지는 파일로 waveform compare 를 할 수 있는 dataset 입니다. Reference Dataset 는 미리 저장해 놓은 data, 현재 진행중인 Simulation 의 data, 현재 진행중인 Simulation 의 일부 data 입니다.

1. 새로운 디렉토리를 생성하고 Tutorial 파일 복사하기.

   이번 챕터를 진행하기 위한 directory 를 만듭니다.

   그리고 tutorial file 들을 새로 만든 디렉토리로 복사합니다.

2. ModelSim/Questa 가 실행되었으면 Change directory 를 이용하여 1번 과정의 directory 로 이동하기.

   A. 바탕화면의 단축 아이콘 혹은 vsim command 를 이용하여 ModelSim/Questa 를 구동합니다.

   B. File > Change Directory 를 이용하여 1번 과정에서 만든 directory 를 지정합니다.

3. Command 입력하기.
    - Verilog

```
vlib work
vlog *.v
vopt +acc test_sm -o opt_test_gold
vsim -wlf gold.wlf opt_test_gold
add wave *
run 750 ns
quit -sim
```

    - VHDL

```
vlib work
vcom -93 sm.vhd sm_seq.vhd sm_sram.vhd test_sm.vhd
vopt +acc test_sm -o opt_test_gold
vsim -wlf gold.wlf opt_test_gold
add wave *
run 750 ns
quit -sim
```

    -wlf 명령어는 vsim command 와 같이 사용할 수 있으며, default 로 출력되는 vsim.wlf 의 파일명을 변경할 수 있는 명령어 입니다.

Creating the Test Dataset

test dataset 은 Reference dataset 과 마찬가지로 .wlf 파일이며, Reference dataset 과 비교 할 대상입니다.

Verilog

1. test bench 수정
    A. File>Open 으로 test_sm.v 파일을 선택하여 열기.
    B. 122번 line 으로 가기.

```
@ (posedge clk) wt_wd('h10,'haa);
```

    C. 'haa' 를 'hab' 로 Data pattern 변경하기.

```
@ (posedge clk) wt_wd('h10,'hab);
```

    D. File>Save 로 파일 저장하기.
2. 수정된 파일을 다시 컴파일 한 후 시뮬레이션 하기.

```
vlog test_sm.v
vopt +acc test_sm -o opt_test_sm
```

```
vsim opt_test_sm
add wave *
run 750 ns
```

VHDL

1. Test bench 수정

   A. File>Open 으로 test_sm.vhd 파일을 선택하여 열기.

   B. 151번 line 으로 가기.

   ```
   wt_wd ( 16#10#, 16#aa#, clk, into );
   ```

   C. 'aa' 를 'ab' 로 Data pattern 변경하기.

   ```
   wt_wd ( 16#10#, 16#ab#, clk, into );
   ```

   D. File>Save 로 파일 저장하기.

2. 수정된 파일을 다시 컴파일 한 후 시뮬레이션 하기.

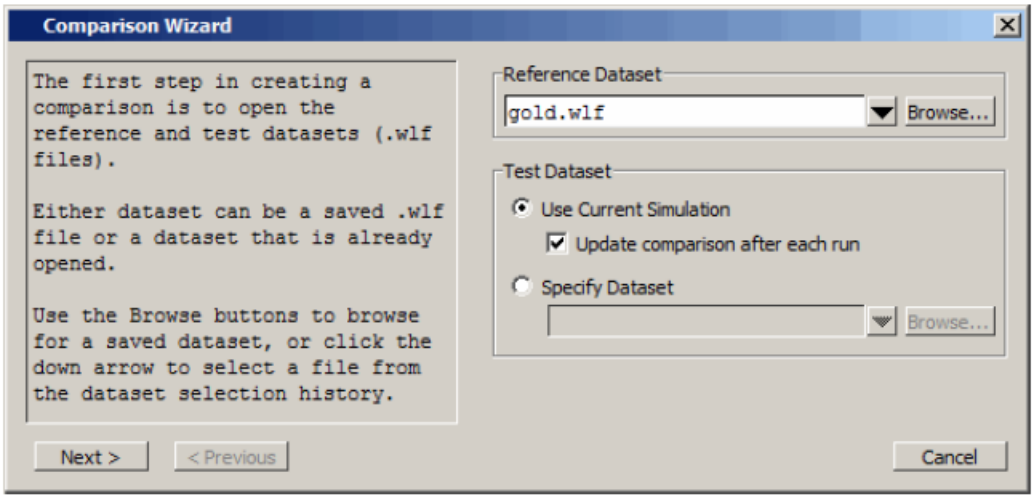
```
vcom test_sm.vhd
vopt +acc test_sm -o opt_test_sm
vsim opt_test_sm
add wave *
run 750 ns
```

Comparing the Simulation Runs

Comparison Wizard 를 통해 waveform compare 를 진행을 해보겠습니다.

1. Comparison Wizard 를 사용하여 comparison 생성
   A. Tool>Waveform Compare>Comparison Wizard 선택하기.
   B. Browse 버튼을 클릭하고 처음 Simulation 을 통해 생성된 reference dataset 인 gold.wlf 을 선택하여 불러옵니다.

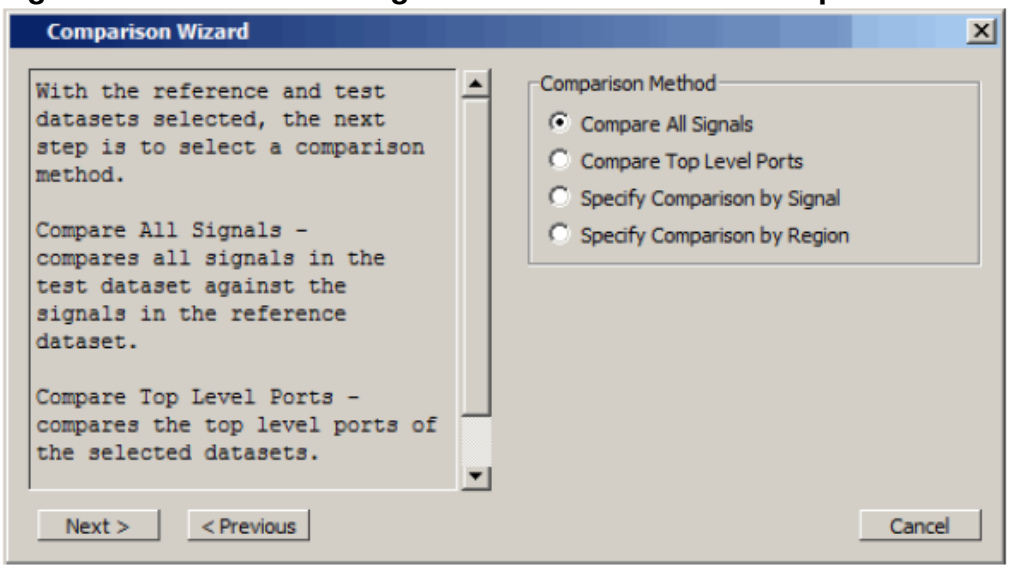
Figure 18-1. First Dialog Box of the Waveform Comparison Wizard



C. 현재 Simulation 을 test dataset 으로 설정하기 위해 Use Current Simulation 을 선택한 후 Next > 버튼을 클릭합니다.

D. 모든 신호를 비교하기 위해 Compare All Signals 를 을 선택한 후 Next > 버튼을 클릭합니다.

Figure 18-2. Second Dialog Box of the Waveform Comparison Wizard



E. Compute Differences Now 를 클릭한 후, Finish 버튼을 클릭합니다.

Compare 를 수행하고 Waveform Window 에 compare 신호가 표시 됩니다.

Viewing Comparison Data

Comparison data 는 Structure(compare), Transcript, Objects, Wave, List Window 에 보여집니다.

Compare Window 는 비교 영역을 보여줍니다.

Transcript Window 는 reference dataset 과 test dataset 을 비교하여 차이가 생긴 횟수가 보여집니다.

Objects Window 는 Structure(compare) Window 에서 comparison object 를 선택하면 비교하여 차이점을 보여줍니다.

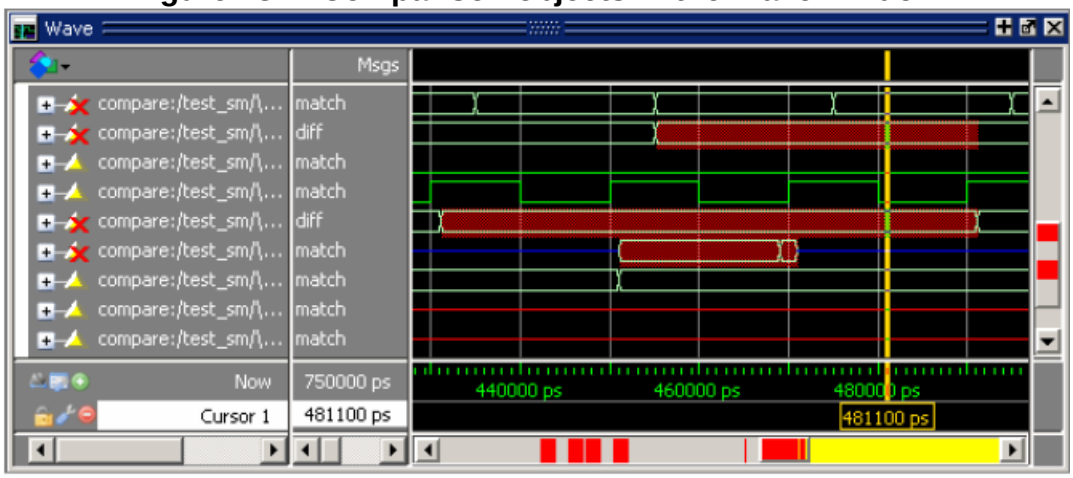
Figure 18-3. Comparison information in the compare and Objects windows

Comparison Data in the Wave Window

Wave Window 는 그래픽 포맷으로 비교 정보를 표시합니다.

- Wave Window 에서 타이밍 오차가 발생하면 pathname 아이콘 옆에 빨간색 X 표시가 됩니다.

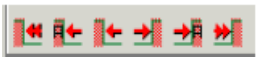
Figure 18-4. Comparison objects in the Wave window



- 타이밍 오차가 발생하면 Waveform 에서는 다른 부분 빨간색 영역으로 표시 됩니다.
- 또한 다른 부분의 스크롤 바 위치에 빨간색 라인이 표시됩니다.
- Annotated 가 다른 부분은 파란색 하이라이트로 표시됩니다.

아래 6개의 아이콘을 통해 빠르게 다른 부분을 찾을 수 있습니다.

Figure 18-5. The compare icons



왼쪽 아이콘부터 설명을 드리도록 하겠습니다. 첫 번째 아이콘은 가장 처음 다른 영역을 찾습니다. 두 번째 아이콘은 기준 커서에서 이전 Annotate 가 다른 부분을 찾습니다. 세 번째 아이콘은 기준 커서에서 이전 다른 영역을 찾습니다. 네 번째 아이콘은 기준 커서에서 다음 다른 영역을 찾습니다. 다섯 번째 아이콘은 기준 커서에서 다음 Annotate 가 다른 부분을 찾습니다. 여섯 번째 아이콘은 가장 마지막 다른 영역을 찾습니다.

특정 신호를 선택하여 아이콘을 통해 쉽게 다른 부분을 찾을 수 있습니다.

Viewing Comparison Data in the List Window

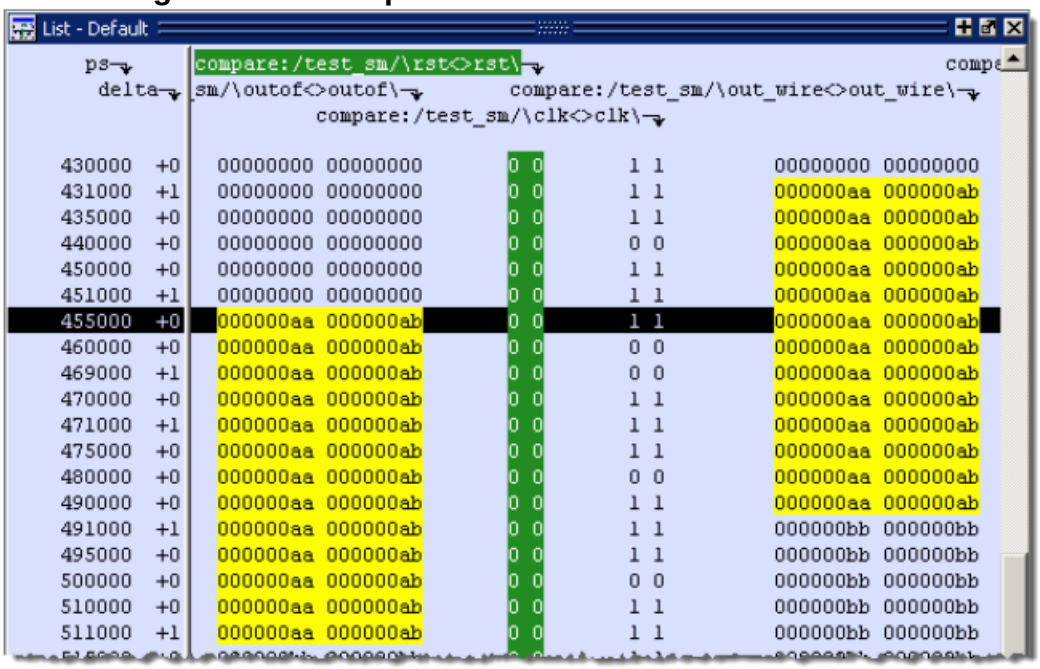
List Window 를 통해 Waveform comparison 결과를 확인 할 수 있습니다.

1. List Window 에 Comparison Data 추가하기.
   A. Menu bar에서 View>List 선택합니다.
   B. Compare tab 에서 comparison object 인 test_sm 을 List Window 로 드래그 합니다.

C. 스크롤을 내려 다른 부분을 찾습니다.

다른 부분은 노란색 하이라이트로 표시되며, annotate 가 다른 부분은 빨간색 하이라이트로 표시됩니다.

Figure 18-6. Compare differences in the List window



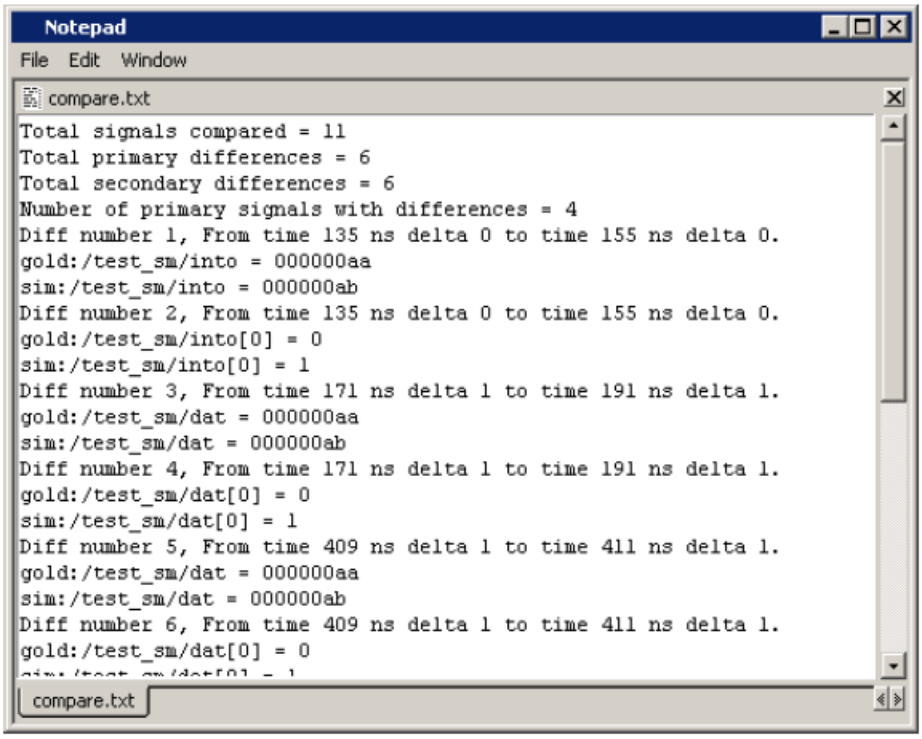
Saving and Reloading Comparison Data

유저 분은 Comparison Data 를 나중에 다시 보기 위해 저장하거나 불러 올 수 있습니다.

Comparison data 를 나중에 다시 불러오려면 2개의 파일을 저장해야 합니다. 하나의 파일에는 오차 계산을 저장하고, 다른 파일에는 비교 조건을 저장합니다. Comparison data 를 불러오려면 reference dataset 이 열려 있어야 합니다.

1. Text file 에 comparison data 를 저장하기.
   A. Main Window 에서 Tool>Waveform Compare>Differences>Write Report 를 선택합니다.
   B. Save 를 클릭합니다. 그럼 현재 디렉토리에 compare.txt 파일이 저장됩니다.
   C. Command 에 notepad compare.txt 를 입력하면, report 가 보여집니다.

Figure 18-7. Coverage data saved to a text file



D. report 를 분석 한 후 Notepad 를 닫습니다.

2. 다시 불러 올 수 있도록 comparison data 파일을 저장하기.

   A. Tools>Waveform Compare>Differences>Save 를 선택합니다.

   B. Save 를 클릭합니다. 그럼 현재 디렉토리에 compare.dif 가 저장됩니다.

   C. Tools>waveform Compare>Rules>Save 를 선택합니다.

   D. Save 를 클릭합니다. 그럼 현재 디렉토리에 compare.rul 이 저장됩니다.

   E. Tools>Waveform Compare>End Comparison 을 선택합니다.

3. Comparison data 를 불러오기

   A. Structure(sim) Window 를 활성화 한 상태에서, File>Open 을 선택합니다.

   B. Files of type 을 Log Files(*.wlf) 로 변경합니다.

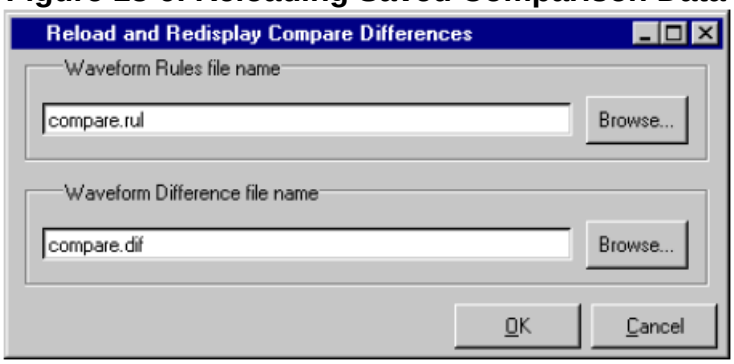
Figure 18-8. Displaying Log Files in the Open Dialog Box

C. Dataset 인 gold.wlf 를 선택하여 Open 을 클릭합니다.

D. Tools>Waveform Compare>Reload 를 선택합니다.

기본 파일 이름을 사용하여 데이터를 저장하고 있기 때문에 Rules 파일과 Difference 파일을 따로 지정하지 않으셔도 됩니다.

Figure 18-9. Reloading Saved Comparison Data



E. OK 를 클릭합니다.

Comparison Data 가 Reload 된 것을 확인 할 수 있습니다. Wave 나 List Window 에 comparison object 를 드래그하여, 비교 할 수 있습니다.

F. Command 에 quit –sim 을 입력하여 시뮬레이션을 종료합니다.

G. Command 에 dataset close gold 를 입력하여 dataset 을 종료합니다.

Chapter 19
Automating Simulation

ModelSim/Questa 는 GUI 메뉴나 Command 입력을 통해 동작을 하게 되는데, 하나의 명령을 차례대로 수행을 하게 됩니다. 만약 반복적인 작업을 해야 할 경우, DO File 을 사용하여 생산성을 향상 시킬 수 있습니다.

DO File 은 한번에 여러 개의 command 를 수행할 수 있는 스크립트입니다. 간단한 반복부터 Tcl 변수나 프로그램, 조건부 실행에 사용됩니다. GUI 에서 불러올 수 있고, Command 를 통해 실행 할 수 있습니다.

Creating a Simple DO File

DO File 은 Text File 에 command 를 입력하는 것으로 간단하게 생성을 할 수 있습니다. 이번 챕터에서는 DO File 생성을 통해 디자인을 불러오거나, Wave Window 에 신호를 추가하거나, 신호에 Stimulus 를 제공하여, 향상된 Simulation 을 할 수 있다. 또한 저장된 Transcript 파일을 가지고 DO File 을 생성 할 수 있습니다.

이번 챕터에서 진행되는 내용은 Reference 매뉴얼의 Saving a Transcript File as a Do file 을 보시면 더욱 자세히 알 수 있습니다.

1. Basic Simulation 에서 만든 directory 로 변경하기.
2. DO File 을 생성하여 Wave Window 에 신호를 추가하고, force signal 을 적용 후 에 시뮬레이션 실행하기.

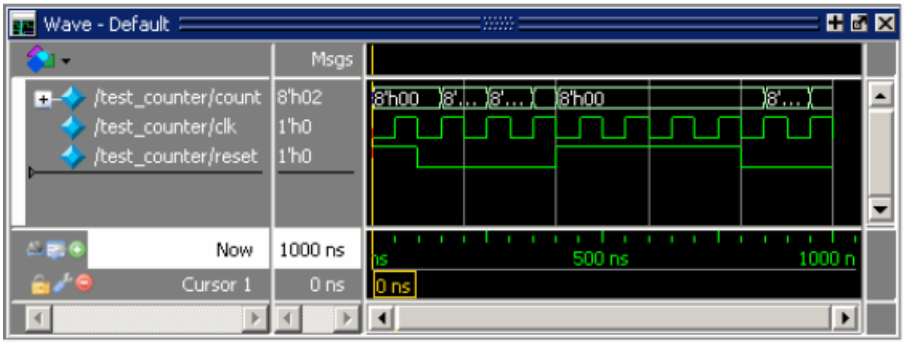
   A. 새로운 DO File 을 생성하기 위해 File>New>Source>DO 를 선택합니다.

   B. Source Window 에 아래의 Code 를 입력합니다.

```
vsim testcounter_opt
add wave count
add wave clk
add wave reset
force -freeze clk 0 0, 1 {50 ns} -r 100
force reset 1
run 100
force reset 0
run 300
force reset 1
run 400
force reset 0
run 200
```

3. 파일 저장하기.
   A. File>Save As 를 선택합니다.
   B. 파일명을 sim.do 로 입력합니다. 그러면 현재 디렉토리에 파일이 저장됩니다.
4. DO File 실행하기.
   A. Command 에 do sim.do 를 입력합니다.
      디자인이 로드 되면서 저장된 command 가 실행합니다. 그리고 Wave Window 에 Wave 가 그려진 것을 볼 수 있습니다.

Figure 19-1. Wave Window After Running the DO File



   B. File>quit 를 통해 종료합니다.

Running in Command-Line Mode

GUI 없이 DOS/UNIX 에서 Simulation 을 실행하기 위해 "Command-Line Mode" 를 사용합니다.

여러 Command(예를 들어 vsim, vlib, vlog 등) 호출을 통해 실제 독립적으로 실행이 가능합니다.

또한 Command 를 포함하는 DO File 을 생성하여, 시뮬레이터를 호출 할 때 해당 파일을 지정할 수 있습니다.

1. 이번 챕터에서 사용할 예제 파일의 위치는 다음과 같습니다.

   - */<install_dir>/examples/tutorials/verilog/automation/counter.v*
   - */<install_dir>/examples/tutorials/verilog/automation/stim.do*

   만약 VHDL license 가 있으신 경우 */<install_dir>/examples/tutorials/vhdl/automation* 디렉토리의 counter.vhd 와 stim.do 를 사용하시면 됩니다.

2. 새로운 디자인 라이브러리와 컴파일을 하기 위한 Source 파일 생성하기.

   유저 분이 1단계에서 만든 새 디렉토리에서 DOS/UNIX prompt 에서 다음 명령을 입력합니다.

A. DOS/UNIX prompt 에 vlib work 를 입력합니다.

B. 컴파일을 하기 위해 Verilog 는 vlog counter.v 를 입력하고, VHDL 은 vcom counter.vhd 를 입력합니다.

3. DO File 생성하기.

A. Text editor 를 엽니다.

B. 파일에 아래의 내용을 입력합니다.

```
# list all signals in decimal format
add list -decimal *

# read in stimulus
do stim.do

# output results
write list counter.lst

# quit the simulation
quit -f
```

C. 현재 디렉토리 위치에 파일명을 sim.do 로 저장합니다.

4. Counter design unit 최적화하기

A. DOS/UNIX prompt 에 아래의 내용을 입력합니다.

vopt +acc counter -o counter_opt

5. Command line mode 를 통해 Simulation 을 실행합니다.

A. DOS/UNIX prompt 에 아래의 내용을 입력합니다.

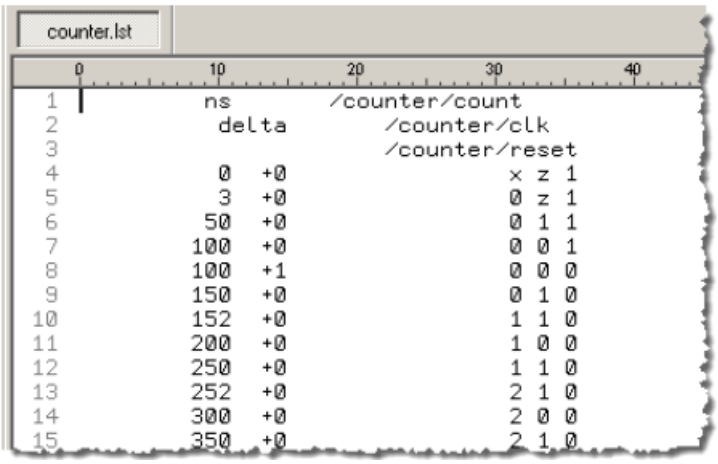
vsim -c -do sim.do counter_opt -wlf counter_opt.wlf

-c 는 GUI 를 호출하지 않고, ModelSim/Questa 를 동작합니다. –wlf 를 통해 시뮬레이션 결과를 .wlf 파일에 저장합니다. 이것은 디버깅을 위해 GUI 에서 시뮬레이션 결과를 볼 수 있습니다.

6. Output 을 list 로 보기.

A. counter.lst 를 열어 시뮬레이션 결과를 볼 수 있습니다. Verilog Version 을 통해 나온 output 을 확인 할 수 있습니다.

Figure 19-2. Output of the Counter



만약에 VHDL Version 을 사용 한 경우에 출력이 약간 다를 수 있습니다.

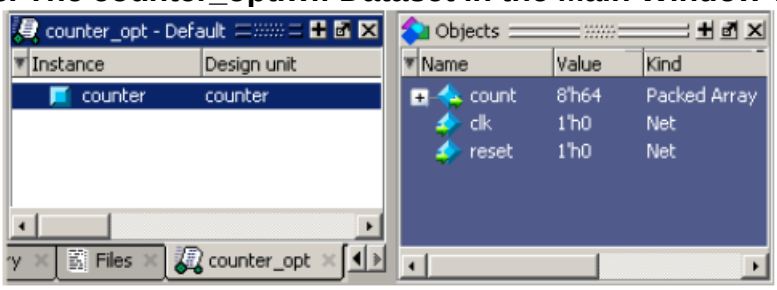
7. GUI 를 통해 결과 확인하기.

   시뮬레이션 결과는 counter_opt.wlf 에 저장되어, vsim 에 –view 옵션을 호출하여 GUI 를 통해 확인 할 수 있습니다.

   A. DOS/UNIX prompt 에 vsim –view counter_opt.wlf 을 입력합니다.

      GUI 가 열리면서 counter_opt 이름을 가진 dataset tab 이 보입니다.

Figure 19-3. The counter_opt.wlf Dataset in the Main Window Workspace



   B. Counter 를 선택하여 오른쪽 마우스를 클릭하고 Add wave 를 선택합니다.

      Waveform Window 에 Waveform 이 보여집니다.

8. File>Quit 를 선택하여 종료합니다.

Using Tcl with the Simulator

이전 단계에서는 DO File 을 Tcl Script 로 만들어 Simulatioin 을 진행 하였습니다. Tcl construct 를 통해 procedures, conditional operators, math and trig functions, regular expression 등을 포함 하고 있습니다.
이번 단계에서는 신호의 특정 값을 테스트하고 그 값이 존재하는 경우 Wave Window 를 확대하고, 북마크를 추가하는 간단한 Tcl Script 를 만들어 보겠습니다. 북마크를 통해 Wave Window 의 특정 zoom 범위와 스크롤 위치를 저장합니다.

Tcl Script 는 Main Window 에 북마크 버튼을 생성합니다.

1. Script 생성하기.

   A. text editor 를 통해 새로운 파일을 생성하여 아래의 구문을 입력합니다.

```
proc add_wave_zoom {stime num} {
 echo "Bookmarking wave $num"
 bookmark add wave "bk$num" "[expr $stime - 100] [expr $stime + 50]" 0
 add button "$num" [list bookmark goto wave bk$num]
}
```

   위의 command 를 살펴보겠습니다.

   - stime, num 이라는 두 개의 argument 를 가지는 새로운 procedure 인 add_wave_zoom 을 생성합니다.
   - 현재 시뮬레이션 시간에서 -100 time 과 +50 time 에 Zoom 범위와 북마크를 생성합니다.
   - Main Window 에 북마크 버튼을 생성합니다.

   B. 구문 맨 아래에 다음 구문을 추가합니다.

```
add wave -r /*
when {clk'event and clk="1"} {
    echo "Count is [exa count]"
    if {[examine count]== "8'h27"} {
        add_wave_zoom $now 1
    } elseif {[examine count]== "8'h47"} {
        add_wave_zoom $now 2
    }
}
```

   위의 command 를 살펴보겠습니다.

   - Wave Window 에 모든 신호를 추가합니다.
   - clk 이 1 인지 식별하기 위해 when 구문을 사용합니다.

- count 값을 검사하여 특정 값일 경우 북마크를 추가합니다.

C. 기존 Basic Simulation 단계에서 만들었던 directory 에 add_bkmrk.do 라는 이름으로 Script 를 저장합니다.

2. test_count 를 로드하여 radix 가 binary 로 설정 되어 있는지 확인하기.

A. ModelSim/Questa 를 실행합니다.

B. File>Change Directory 를 선택하고, DO File 을 저장했던 디렉토리로 변경합니다. (이전 단계의 Basic Simulation 을 생성한 Directory)

C. Command 에 radix –binary 를 입력합니다.

D. Command 에 vsim testcounter_opt 를 입력합니다.

3. DO File 을 실행하여 디자인 진행하기.

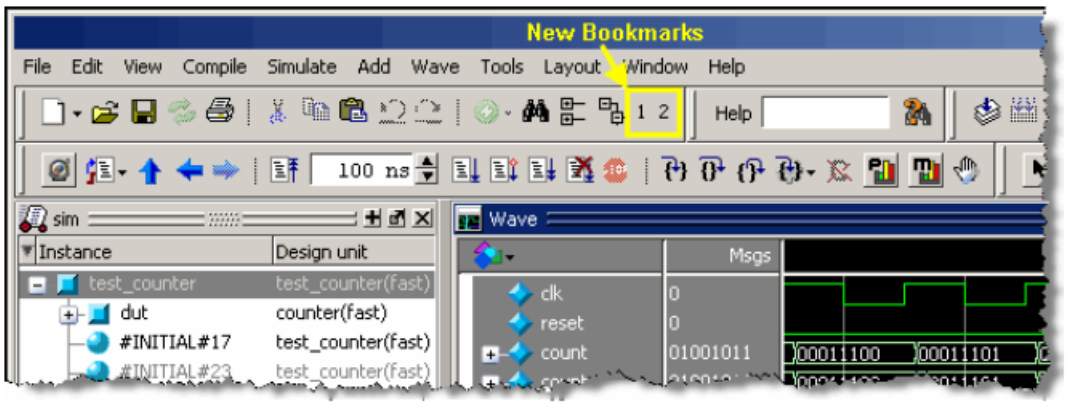
A. Command 에 do add_bkmrk.do 를 입력합니다.

B. Command 에 run 1500 ns 를 입력합니다.

시뮬레이션이 실행되고 DO 파일은 두 개의 북마크를 만듭니다.

Main Window 의 Toolbar 에 북마크 버튼 1, 2 가 생성됩니다.

Figure 19-4. Buttons Added to the Main Window Toolbar



C. 버튼을 클릭하면 count 가 특정 값이 될 때, Wave Window 의 Zoom in 과 해당 시간에 스크롤 되는 것을 볼 수 있다.

D. File>Quit 를 통해 종료합니다.

이 챕터에서 진행되는 내용은 유저 매뉴얼의 Tcl and DO Files 를 보시면 더욱 자세히 알 수 있

습니다.

Chapter 20
Getting Started With Power Aware

이번 챕터에서는 RTL 디자인의 Power Aware Simulation (Questa PASim) 실행하는 방법에 대해서 살펴볼 것입니다.

- Low Power Design intent 를 정의하는 Unified Power Format(UPF) 파일 작성
- 사용자 정의 Assertions, Power intent UPF File 관리, Power Aware Verification Flow 를 통한 작업
- 정확하게 Register Transfer Level(RTL) 에서의 power up/down 및 보존 동작을 Power Aware retention flip-flop 모델로 역할을 관찰.

이번 챕터의 예제는 Test bench 에 관련된 Clock-driven memory interleaver 입니다. 디렉토리 위치는 <*install_dir*>/examples/tutorials/pa_sim/ 입니다.

```
pa_sim
    |
  example_one
    |
    |
    |
    +-- Libraries ........ Verilog and SystemVerilog library source
    |     |
    |     +-- io
    |     |
    |     +-- sram_256x16
    |
    +-- Questa ........... Simulation directory
    |     |
    |     +-- scripts .... Compilation & simulation commands
    |
    +-- RTL .............. Source files for interleaver design
    |
    +-- UPF .............. UPF file for power intent
```

이 예제의 경우 example_one 디렉토리에서 모든 시뮬레이션을 실행합니다.

Script Files

Questa/scripts/directory 에는 컴파일 과 모든 시뮬레이션을 실행 할 수 있는 DO File 을 포함하고 있습니다.

- compile_rtl.do — Compile RTL source

- analyze_rtl.do — Analyze UPF and extract PA netlist
- doit_rtl.do — Run RTL simulation
- sim.do — Simulation commands

Create a Working Location

디자인을 시뮬레이션 하기 전에 work directory 에 복사본을 만들어 라이브러리를 생성하고 해당 라이브러리에 소스코드를 컴파일 해야 합니다.

1. 설치 디렉토리 외부에 새로운 디렉토리를 생성하여, 디자인 파일을 복사합니다.
2. Questa 를 실행하기.
   A. UNIX shell prompt 에서 vsim 을 입력하거나, Window 의 Questa 아이콘을 더블클릭 합니다.
   B. 메인 메뉴의 File>Change Directory 를 선택하여 아래의 Directory 로 변경합니다.

      <*my_tutorial*>/pa_sim/example_one

      my_tutorial 은 Step 1 에서 생성한 디렉토리 입니다.

Compile the Source Files of the Design

컴파일 단계에서는 HDL 디자인을 처리하고 시뮬레이션을 하기 위한 코드를 생성합니다. 이 단계는 Power Aware 와 non-Power Aware 모두 동일합니다. 각각의 시뮬레이션 둘 다 같은 output 을 사용합니다.

1. 모든 RTL source Code 파일을 컴파일 하기 위해 Transcript Window 에 command 를 입력합니다.

   ```
   do ./Questa/scripts/compile_rtl.do
   ```

   DO File 에는 실행 command script 가 있습니다.

   vlib work

   vlog -novopt -f ./Questa/scripts/compile_rtl.f

이러한 명령어들은 Power Aware 에 관련한 특별한 동작을 제공하게 됩니다.

Annotate Power Intent

Power Annotation 단계는 설계와 관련된 파일과 Unified Power Format(UPF) 파일을 처리하여, 그 파일에서 power intent 를 추출하여, power intent 를 반영하도록 컴파일 된 HDL model 을 확장합니다.

- Power distribution network 구축 (supply ports, nets, sets, and switches)
- Power control architecture 구축 (retention registers, isolation cells, level shifters, and their control signals)
- Power-related behavior 삽입 (retention, corruption, and isolation clamping on power down; restoration on power up)
- Automatic assertions to check for power-related error conditions 삽입 (such as current control signal sequencing)

1. UPF 를 분석하고 power annotation 을 수행하기 위해 transcript window 에 입력합니다.

   ```
   do ./Questa/scripts/analyze_rtl.do
   ```

   do 파일을 보면 아래의 구문이 적혀 있습니다.

   **vopt rtl_top **

   **-pa_upf ./UPF/rtl_top.upf **

   **-pa_prefix "/interleaver_tester/" **

   **-pa_replacetop "dut" **

   **-pa_genrpt=u+v **

   **-pa_checks=i+r+p+cp+s+uml **

   **-pa_enable=nonoptimizedflow **

   -o discard_opt

   vopt command 는 power annotation process 를 제어합니다.

   - -pa_upf – UPF 에 작성된 power intent 파일의 위치를 지정합니다.
   - -pa_prefix – (Power annotation 이 수행되고 있는) DUT 가 인스턴스 될 test bench 의 이름을 지정합니다.
   - -pa_replacetop – Top-level DUT 의 instance 의 이름을 지정합니다.

- -pa_genrpt – 현재 Directory 에 저장되어 있는 Power Aware report file 을 생성
- -pa_checks – Assertion check 를 활성화 합니다.

Specifying Power Aware Options

vopt command 의 argument 는 Power Aware Simulation 을 위한 많은 옵션이 있습니다. Power Aware argument(all begin with –pa_) 에 대한 정보는 Reference Manual 의 vopt command 를 참조하시면 되겠습니다.

-pa_checks argument 의 일환으로 "s"를 지정하면 level shifter 의 삽입을 위한 static check 가 켜집니다. 분석하는 동안 누락된 level shifter 의 massage 가 출력됩니다. vopt 를 통한 실행을 했을 때 output 을 확인 할 수 있다.

```
** Note: (vopt-9851) [ UPF_LS_STATIC_CHK ] Found Total 30 Valid level
shifters.
```

Simulate the Power Aware Design

Power Aware simulation accurately model 는 power architecture 의 동작 및 HDL 디자인의 power architecture 효과입니다.

1. Power Aware simulation 을 진행하기 위해 Transcript window 에 command 를 입력합니다.

   ```
   do ./Questa/scripts/doit_rtl.do
   ```

   do 파일을 보면 아래의 구문이 적혀 있습니다.

   **vsim interleaver_tester **

   **-novopt **

   **+nowarnTSCALE **

   **+nowarnTFMPC **

   **-L mtiPA **

   **-pa **

   **-l rtl.log **

   **-wlf rtl.wlf **

   **-assertdebug **

**+notimingchecks **

-do ./scripts/sim.do

시뮬레이션에서 vsim command 에 –pa 옵션을 설정하여 Power Aware mode 가 Simulator 에 호출 됩니다. mtiPA 라이브러리는 corruption, isolation, retention 을 포함하는 미리 컴파일 된 기본 라이브러리 입니다. 이 라이브러리는 –L 옵션을 통해 로드 됩니다.

2. Main Window 는 structure window 함께 Object, Wave, Source Window 를 추가 할 수 있습니다.

3. Structure Window 에서 test bench 인 interleaver_tester 가 나올 때까지 스크롤을 맨 위로 이동합니다.

4. Sim tab 에서 interleaver_tester 를 더블 클릭하면 Source Window 가 열리면서 test bench(interleaver_teset.sv) 를 확인 할 수 있습니다.

5. Source Window 에서 "Simulation control" 이라는 이름의 섹션을 찾습니다. 이 블록은 power management block 의 추상적인 표현을 제공하며, 다음 테스트를 수행합니다.

    - power_down_normal – (Test 1, line 97) Normal power-down cycle where retention, isolation, and clock gating are done correctly.
    - power_down_no_iso — (Test 2, line 101) Power-down cycle with the isolation control signal not toggled correctly.
    - power_down_no_clk_gate — (Test 3, line 105) Power-down cycle where the clock is not gated properly.
    - sram_PWR — (Test 4, line 90/92) Toggles the built-in power control signal for the SRAM models.

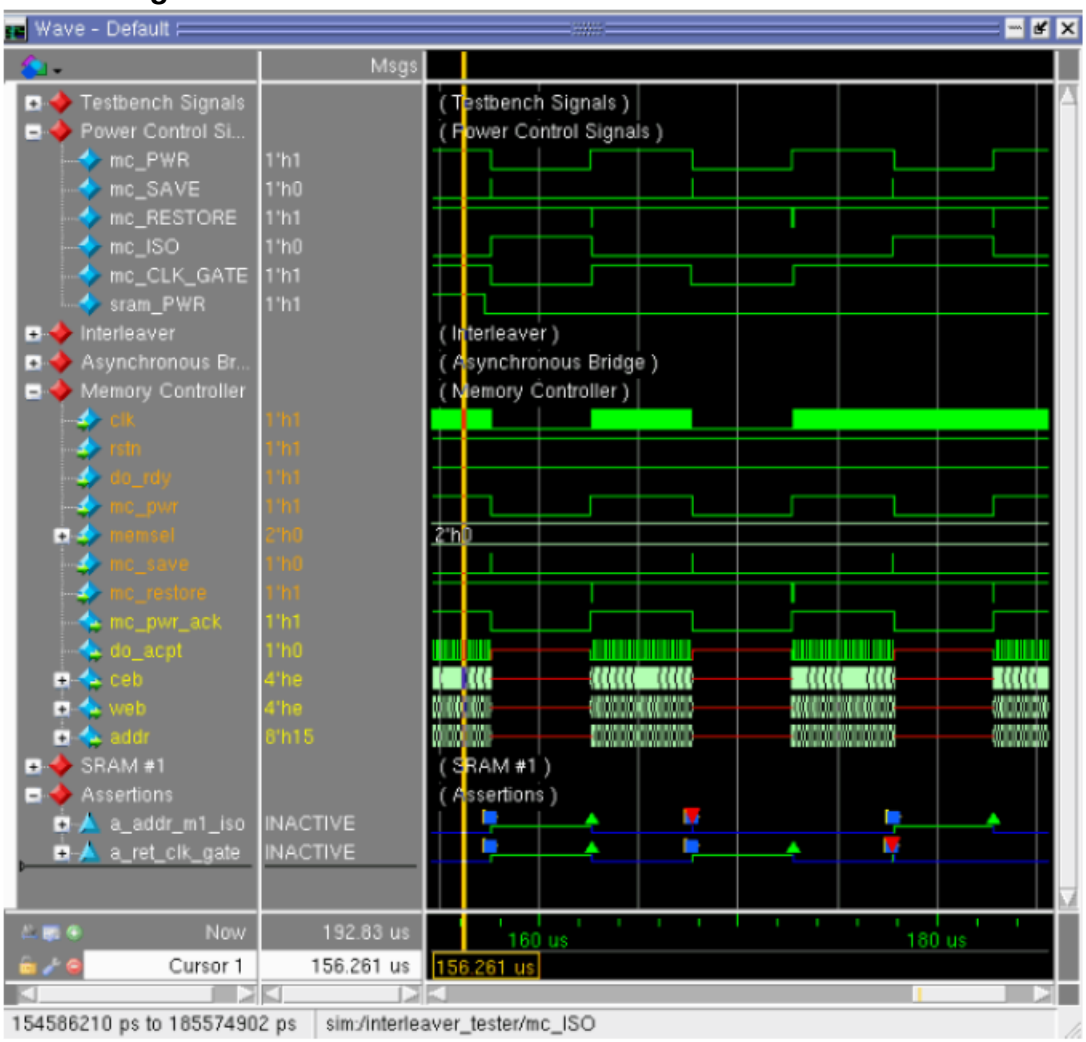
Analyze Results

Power Aware 디자인의 Simulation 결과는 그래픽 인터페이스를 통해 분석을 할 수 있습니다.

1. Wave Window 에서 Simulation 결과를 볼 수 있도록 Main window 에서 wave tab 을 클릭합니다.

2. Wave Window 에서 처음 3개의 test(155 us to 185 us) 를 볼 수 있도록 Zoom level 을 조절합니다.

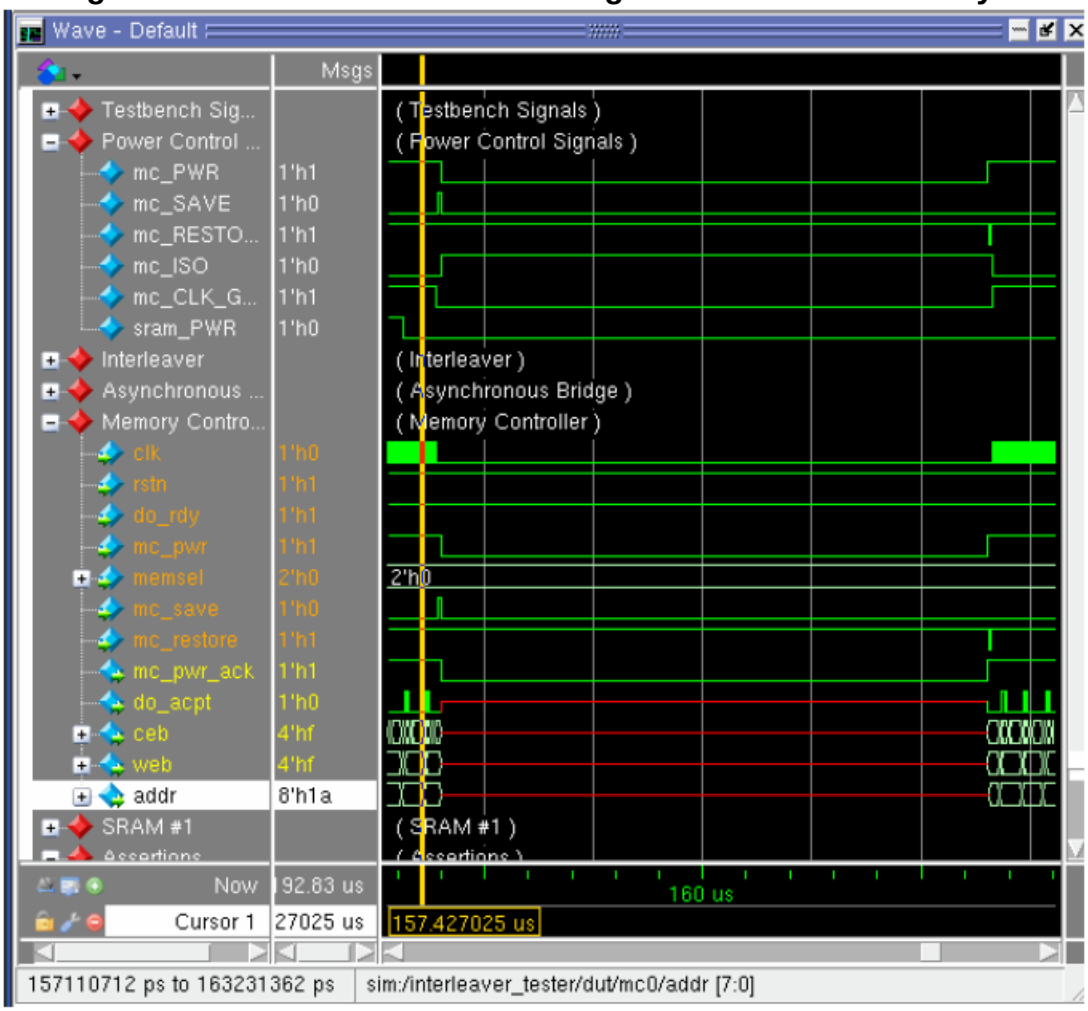
Figure 20-1. Results of the Power Aware RTL Simulation



Test 1(power_down_normal) 의 결과

1. 조금 더 확대하여 Test 1(155 us to 163 us) 에 초점을 맞춥니다. 이 test 는 정상적으로 power-down cycle 을 확인 할 수 있다.

Figure 20-2. Retention of addr During Normal Power Down Cycle



이 예제의 isolation strategy 는 isolation insertion point 를 위해 "parent" 를 지정했습니다. "Memory Controller" 의 모든 output 을 알 수 없습니다. Memory controller 의 output 으로부터 downstream block 을 보면 isolation value 를 확인 할 수 있습니다. SRAM 들에 대한 input 에서 address 는 0에 고정되고, chip 과 write enable 은 1 에 고정됩니다.

2. Memory controller 의 addr output 을 보면 (이 블록의 전원이 차단되기 직전에) 알 수 없는 상태의 왼쪽에 있는 마지막 값은 00011011 입니다. 전원을 재 투입하기 직전에 같은 신호를 보게 되면 값은 00011011로 복원됩니다. 이것을 통해 복원 동작을 볼 수 있습니다.

Results from Test 2 (power_down_no_iso)

167 us 에 다음 test 를 시작하기 위해 조금 이동합니다. 이 테스트는 디자인을 다시 power down 하며, isolation 을 하지 않습니다. 이 테스트에서 SRAM 모델의 address 입력이 0 로 고정되지 않는 것을 알 수 있습니다. 이 문제로 인해 메모리 컨트롤러에서 알 수 없는 값을 SRAM 에서 보내고 있습니다.

이 문제를 해결하기 위해 assertion 을 사용합니다. 이 경우 vopt command 를 이용하여 –pa_check=i 를 활성화합니다.

1. Transcript 열기.

   View>Transcript

   발생한 문제가 메시지로 표시됩니다.

   ```
   ** Error: (vsim-8918) MSPA_ISO_EN_PSO: Isolation control (0) is not
   enabled when power is switched OFF for the following:
   Port: /interleaver_tester/dut/mc0/addr.
   ```

   문제를 해결하기 위해 `Assertion` 구문을 작성하도록 하겠습니다.

2. Assertion Window 열기.

   View>Coverage>Assertions

   모든 assertions 에서 fired 는 빨간색 하이라이트로 표시됩니다. 아래 메시지를 생성되는 즉시 assertion 에 나타납니다.

   ```
   /mspa_top/PA_ISO_CHECK_1_2_BLK/QSPA_ISO_EN_PSO_1
   ```

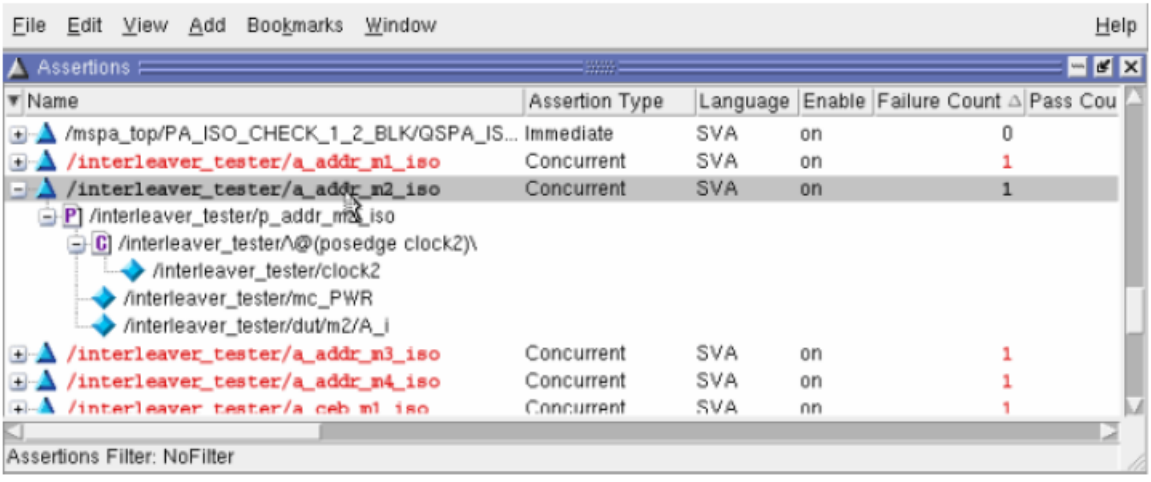
   이것은 Built-in Assertion 이며, 일부 failed 된 user-defined assertion 도 있습니다.

3. Assertion Window 를 Undock 한 후 Assertion 이름을 볼 수 있습니다.

   ```
   /interleaver_tester/a_addr_m2_iso
   ```

   이것은 user-defined assertion 입니다. Assertion Window 에서 Assertion 과 Assertion expression, 다양한 카운터를 구성하는 신호를 볼 수 있는 assertion 으로 확장합니다.

Figure 20-3. The Assertions Window



4. 해당 assertion 을 선택하고 팝업 메뉴를 표시하기 위해 마우스 오른쪽 버튼을 클릭합니다.

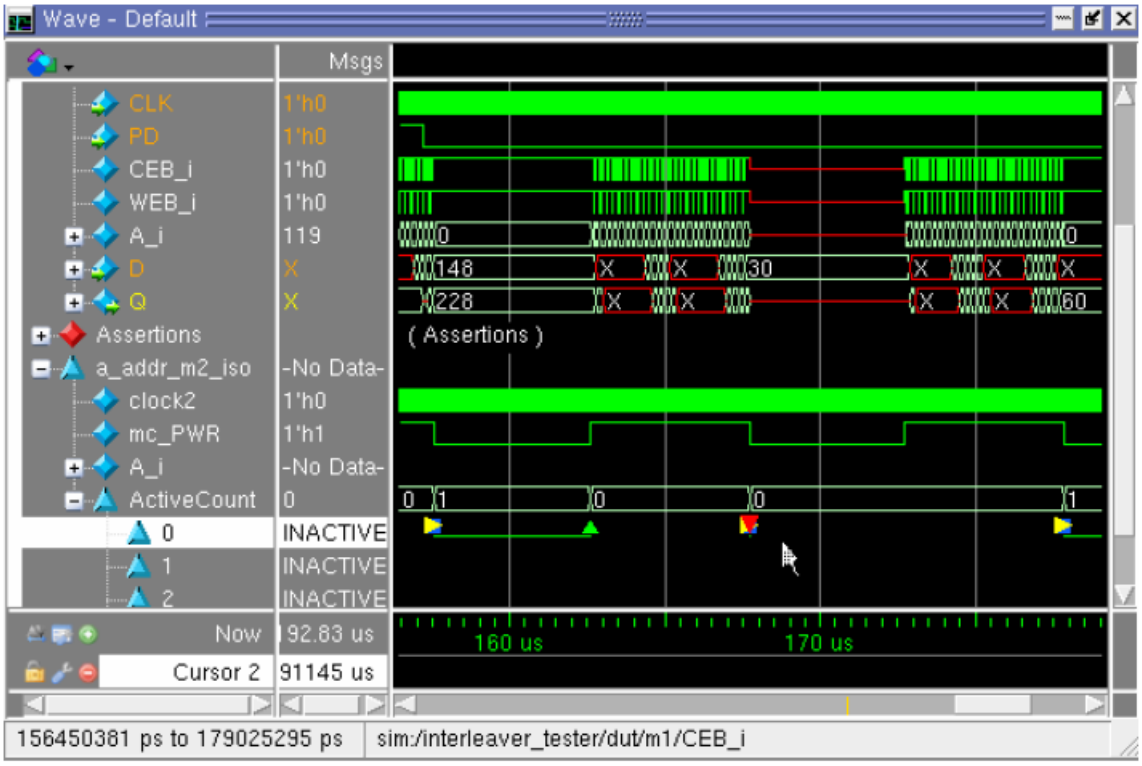
5. Add Wave>Selected Objects 를 선택합니다. Wave Window 의 왼쪽 에 있는 pathname 창에 assertion 그룹이 추가 됩니다.

6. 3 가지의 모든 Test 를 Wave Window 에서 Zoom out 을 통해 확인 할 수 있는데, 녹색과 빨간색 삼각형은 각각 Assertion Pass 와 failure 를 나타냅니다.

   Test 1 을 보면 정상적으로 Power-down cycle 을 하는 Simulate 를 하는 동안 power-down 에서 활성화(녹색 선)와 비활성화(파란 선) 을 통해 Assertion 의 변화를 볼 수 있습니다.

   Test 2 를 보면 isolation 을 활성화 하지 않았습니다. Assertion 은 power-down 에서 시작합니다. SRAM 의 Address input 에 올바른 값이 들어가지 않기 때문에, 다음 Clock 에서 Fail 이 나타납니다. 그렇기 때문에 빨간색 삼각형을 볼 수 있습니다.

Figure 20-4. User-Defined Assertion Failure (red triangle)



Results from Test 3 (power_down_no_clk_gate)

이 예제에서 사용 된 모델은 Clock 가 save/restore 과정에서 gated Low 가 되어야 합니다. 세 번째 테스트는 Clock 이 제대로 gated Low 가 되었는지 확인을 합니다. RTL description 에서는 적절한 clock gating 을 위해 assertion 을 사용 할 수 있습니다.

Assertion Window 에서 interleaver_tester/a_ret_clk_gate 라는 이름을 가지는 assertion 이 Fail 입니다.

1. 팝업 메뉴를 보기 위해 해당 assertion 을 선택 한 후 오른쪽 마우스를 클릭합니다.
2. Add Wave>Select Objects 를 선택하고, Wave Window 에 추가합니다.
3. 해당 Assertion 을 보면 3 개의 Test 중에 처음 2 개의 Test 에서 Fail 이 난 것을 확인 할 수 있습니다.
4. Cursor 를 177780 ns 에 위치시키면 debug window 에서 assertion failed 를 표시합니다.

   `/interleaver_tester/dut/mc0/clk=1'h1`

Results from Test 4 (sram_PWR)

SRAM 은 output 이 0 로 고정된 모델이며, built-in power-down 을 가지고 있습니다. 이 테스트는 기능을 테스트 하기 위해 모델의 power signal 을 전환합니다.

1. Wave Window 에서 152260 ns 로 이동합니다.
2. pathname 창에서 SRAM #1 신호를 찾은 후 [+] 를 눌러 확장합니다.
3. Power control signal m1/PD 를 찾은 후 이 시점에서 전환을 봅니다. PD 신호가 활성화 되어있는 동안 SRAM 의 output m1/Q 가 0으로 고정됩니다.

Simulate>End Simulation 을 통해 종료합니다.