# HyperBlue Semu-Ultra v3.0

**用户指导手册**

**PG185 April 9, 2019**

目录

[HyperBlue Semu-Utrascal v3.0 1](#_Toc5869168)

[关于本手册 4](#_Toc5869169)

[综述 4](#_Toc5869170)

[1. 简介 5](#_Toc5869171)

[1.1 Semu软件的特性 5](#_Toc5869172)

[1.2 Semu软件的工作流程 6](#_Toc5869173)

[1.3 Semu软件安装 6](#_Toc5869174)

[2. 快速入门 11](#_Toc5869175)

[3. 配置工程 13](#_Toc5869176)

[3.1 输入文件 14](#_Toc5869177)

[3.2 Project菜单 19](#_Toc5869178)

[3.3 Configure Hardware 22](#_Toc5869179)

[3.4 Configure RTL Pins 33](#_Toc5869180)

[3.5 Configure Testbench 41](#_Toc5869181)

[4. Build流程 48](#_Toc5869182)

[4.1 Build菜单 48](#_Toc5869183)

[4.2 编译 50](#_Toc5869184)

[4.4 命令行 52](#_Toc5869185)

[5. Board 56](#_Toc5869186)

[5.1 Board菜单 56](#_Toc5869187)

[5.2 Load Bit File By PCIE 57](#_Toc5869188)

[5.3 Load Bit File By JTAG 57](#_Toc5869189)

[5.4 Reset FPGA 58](#_Toc5869190)

[5.5 Change Emulation Speed 59](#_Toc5869191)

[5.6 Board Info 59](#_Toc5869192)

[6 Run 60](#_Toc5869193)

[7 Emulation流程 60](#_Toc5869194)

[7.1 波形获取方式对Emulation的影响 60](#_Toc5869195)

[7.2 Testbench类型对Emulation的影响 61](#_Toc5869196)

[7.3 Readback波形获取方式下的C++ Testbench的Emulation流程 61](#_Toc5869197)

[7.4 Readback波形获取方式下的Simulation Testbench的Emulation流程 69](#_Toc5869198)

[7.5 Probe波形获取方式下的C++ Testbench的Emulation流程 70](#_Toc5869199)

[7.6 Probe波形获取方式下的Simulation Testbench的Emulation流程 70](#_Toc5869200)

[7.7 命令行批量运行Emulation 71](#_Toc5869201)

[8 Simulation流程 71](#_Toc5869202)

[8.1 时钟控制 72](#_Toc5869203)

[8.2 DUT控制 73](#_Toc5869204)

[8.3 Message 73](#_Toc5869205)

[9 波形获取 73](#_Toc5869206)

[9.1 Readback波形获取 74](#_Toc5869207)

[9.2 Probe波形获取 74](#_Toc5869208)

[10 C++ Testbench的写法 79](#_Toc5869209)

[10.1 模板文件 79](#_Toc5869210)

[10.2 数据类型 80](#_Toc5869211)

[10.3 C++ Testbench Example 80](#_Toc5869212)

[10.4 Port接口的FIFO通信机制对发送/接收函数的影响 95](#_Toc5869213)

[10.5 Testbench的串行特性 97](#_Toc5869214)

[10.6 时钟控制 98](#_Toc5869215)

[11 Simulation Testbench的写法 99](#_Toc5869216)

[11.1模板文件 100](#_Toc5869217)

[11.2 Simulation Testbench Example 100](#_Toc5869218)

[11.3 用户自定义系统任务/函数的使用 107](#_Toc5869219)

[11.4 其他 109](#_Toc5869220)

[12 C-API 109](#_Toc5869221)

[12.1 接口函数 109](#_Toc5869222)

[12.2 初始化 114](#_Toc5869223)

[12.3 时钟和仿真控制 114](#_Toc5869224)

[附录 116](#_Toc5869225)

[sdt类型工程支持的接口类型和方法 116](#_Toc5869226)

## 关于本手册

### 综述

Semu是一款基于SCE-MI协议和FPGA验证平台的桌面型Emulator，主要用于加速验证过程、提高验证效率、缩短芯片前端设计研发的调试和验证周期。

本手册旨在向用户描述Semu软件的使用方法，主要包括以下几个方面的内容：软件特性、使用流程、配置工程的步骤及选项含义、Simulation编译和Emulation编译过程、Simulation运行过程、Emulation运行过程（包括添加信号、获取波形、断点功能等）、Testbench分类及写法、C-API接口函数的使用等。

## 简介

### 1.1 Semu软件的特性

Semu软件具有如下功能：

1. 支持高速仿真，FPGA中的DUT时钟频率可以达到50MHz；
2. 支持自动生成软硬件之间的高速互连接口；
3. 支持3种Testbench：Manual Testbench、C++ Testbench、Simulation Testench；
4. 支持2种工程类型：General、Sdt；
5. 支持两种波形获取方式：Readback、Probe；
   1. Readback波形获取方式下：
   * 支持获取设计中的所有寄存器类型信号的波形，生成VCD文件；
   * 支持硬件断点（仅在使用C/C++ Testbench时支持）；
   * 在启动Emulation之前设置待抓取的信号（可通过API函数从文件中获取或从界面添加信号），支持动态修改待抓取的信号，而不需要重新编译工程；
   * 速度较慢；
   * 待测设计输入的源文件可以是Vivado支持的任何一种源文件类型。
   1. Probe波形获取方式下：
   * 支持获取设计中的所有类型信号的波形，并生成VCD波形文件或FSDB波形文件；
   * 支持最多一次获取10752-bit信号的波形；
   * 支持硬件断点（仅在使用C/C++ Testbench时支持）；
   * 抓取不同的信号，需要重新编译工程；
   * 待测设计输入的源文件必须是Verilog RTL源码；
   * 速度较快。
6. 支持动态修改Testbench、受控时钟的频率，而不需要重新综合设计工程；
7. 支持Verilog语言设计文件、EDIF网表等设计文件；
8. 支持VCS、Modelsim第三方仿真器；
9. 支持三种FPGA板：Xilinx\_KC705、HyperSilicon\_B2000T、HyperSilicon\_K115、HyperSilicon\_E4000。

### 1.2 Semu软件的工作流程

Semu软件以待验证的RTL设计（Design Under Test or DUT）作为输入，生成一个完整的FPGA验证调试环境。用户可以使用一个已有的RTL Testbench（即Simulation Testbench：包括Verilog Testbench、SystemVerilog Testbench、UVM等）或使用Semu软件的API函数写C++ Testbench进行对DUT进行验证。所有的DUT都可以在Simulation环境（不支持Simulation Testbench）和Emulation环境下得到验证。

Semu软件的工作流程如下：

1. 工程配置
   * 定义工程变量，包括源文件、路径、顶层模块、Emulation板子型号、仿真器等；
   * 根据DUT顶层模块端口生成接口，连接Testbench和DUT；
   * 定义Testbench的类型，并指定Testbench文件。
2. 编译环境
   * 编译，生成一个完整的Simulation或Emulation运行环境。
3. 运行环境
   * 从Simulation或Emulation控制界面控制Simulation或Emulation环境运行的过程。

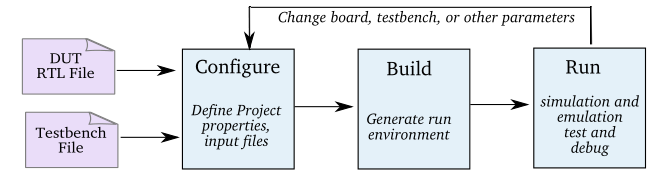


图 1.1 Semu软件工作流程

### 1.3 Semu软件安装

#### 1.3.1 推荐的最小系统配置

1. 操作系统：64位Centos 6系列；
2. CPU：1Ghz Pentium x86 处理器；
3. 最低内存大小：1GB；
4. 配置硬盘空间大小500G；
5. 支持Xilinx的开发板；
6. PCIExprress connector， 8 lane (gen 1 or gen 2)；
7. Python 2.5版本及以上；
8. Xilinx Vivado 2014.1或以上；
9. VCS 2014.04，Verdi 2014.04。

#### 1.3.2 软件安装步骤

在安装Semu软件之前需要预先安装Vivado、Xilinx、VCS、Modelsim等软件，并设置环境变量。下面是详细描述Semu软件的安装步骤。

1. 下载Semu安装包，并解压到安装目录下，您可以依照如下指令操作。
   * 创建一个安装文件夹HyperBlue；
   * 解压安装包到此目录下

命令如下：

mkdir /opt/HyperBlue

cd /opt/HyperBlue

tar –zxvf HyperBlue-Semu\_ultra.tar.gz

此时，软件已经安装完成，但是在启动软件之前还需要配置软件的运行环境。

1. 设置环境变量，BlueNoC配置及测试

用户可以将环境变量设置在你的根目录的.cshrc或.bashrc文件中，也可以用module\_files等工具对环境配置进行管理，下面介绍.bashrc文件环境变量的配置方式。

HYPERBLUEDIR和HYPERBLUE\_HOME都指向Semu的安装目录，此外还需要设置HYPERBLUE\_LICENSE\_FILE或者LM\_LICENSE\_FILE，license文件会由HyperBlue公司提供给用户。

export HYPERBLUE\_HOME=/opt /HyperBlue/current

export HYPERBLUEDIR=”$HYPERBLUE\_HOME/lib”

export PATH=$PATH:$HYPERBLUE\_HOME/bin

export LM\_LICENSE\_FILE=port@pcname (比如27000@hyperblue)

source /opt/Xilinx/Vivado/2014.1/settings64.sh > /dev/null

unset LD\_LIBRARY\_PATH

export XIL\_CSE\_PLUGIN\_DIR=$HOME/ .cse

环境变量设置完成后，需要安装支持Semu运行的Linux系统软件依赖包以及PCIE驱动文件。

1. 依赖包安装

首先进入Semu的依赖软件安装包目录。

cd /opt/HyperBlue/HyperBlue-Semu\_ultra/install/rpms/centos6

使用#rpm -ivh指令将安装包目录下的文件逐个安装，安装顺序如下：

dkms-2.2.0.2-1.noarch.rpm

flash-plugin-11.2.202.577-release.x86\_64.rpm

fxload-2008\_10\_13-10.3.x86\_64.rpm

Judy-1.0.5-5.1.x86\_64.rpm

libftdi-0.18-2.el6.x86\_64.rpm

libusb-devel-0.1.12-23.el6.x86\_64.rpm

libftdi-devel-0.18-2.el6.x86\_64.rpm

tcl-8.5.7-6.el6.x86\_64.rpm

tk-8.5.7-5.el6.x86\_64.rpmg

gtkwave-3.3.25-1.el6.x86\_64.rpm

xterm-253-1.el6.x86\_64.rpm

1. 驱动安装

进入Semu的驱动软件安装包目录。

cd /opt/HyperBlue/HyperBlue-Semu\_ultra/install/ Bluenoc

使用#rpm -ivh指令安装驱动，指令如下：

#rpm -ivh hyperblue-bluenoc-2017.05.beta1-1dkms.noarch.rpm

1. license安装

软件启动需要license支持，软件自带LICENSE管理器存放在在$BLUESPECDIR/目录下。对于license文件，您需要修改我们提供给您的license文件的开始部分，FCAA14E01A6C为当前主机的MAC地址。

SERVER PC FCAA14E01A6C

VENDOR HYPERBLUE HYPERBLUE

需要将PC修改成您的主机名，以PC主机名为默认值localhost.localdomain为例。

SERVER localhost.localdomain FCAA14E01A6C

VENDOR HYPERBLUE /opt/HyperBlue/HypreBlue-Semu\_ultra/util/flexlm/x86\_64\_re3/HYPERBLUE

然后在您的环境变量中加入以下内容。

export PATH=”/opt/HyperBlue/ HypreBlue-Semu\_ultra/util/flexlm/x86\_64\_re3:”$PATH

在环境变量中起license指令如下。

export PATH=”/opt/HyperBlue/ HypreBlue-Semu\_ultra/util/flexlm/x86\_64\_re3:”$PATH

#### 1.3.3 硬件安装注意事项



图 1.2 Semu硬件安装转接图

注意硬件安装时，如果用到PCIE转接卡，那么PCIE转接卡的方向应与图中的方向一致，否则主机无法识别PCIE设备。

## 快速入门

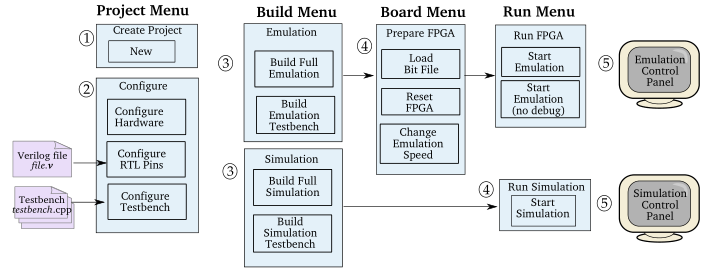


图 2.1 Emulation、Simulation流程

本小节简述Semu软件的使用方法。图2.1给出了Emulation、Simulation的流程及对应的菜单选项。Verilog文件（*file*.v）和testbench文件（*testbench*.cpp）是Semu软件的输入文件，另外如果工程类型为sdt，那么还需要手写t\_gen.tcl文件作为工程的输入文件。只有在使用C++ Testbench或Simulation Testbench时才需要Testbench文件作为输入。当使用Manual Testbench时，Semu软件会自动生成Manual Testbench。

在Linux的终端（terminal）中输入“semu”即可启动Semu软件的图形界面，或输入 “semu *path/*project.cfg” 启动Semu软件的图形界面并指定的打开工程。

**说明**：下面步骤的序号与图2.1 Emulation、Simulation流程中的序号一一对应。

1. 新建工程，Project->New；
2. 配置工程：
   * Project->Configure Hardware…，打开“Configure Hardware”界面：设置top-file、top-module、dut frequency(MHz)、noc frequency(MHz)、res\_fifo\_depth、fpga board、verilog-simulator、xrf\_debugging、project type、wave method、Build for signal debugging、Enable fifo depth、Enable dut dont touch、Build for timing debugging；
   * 点击“Configure Hardware”界面上的“Advanced Options…”按钮打开“Advanced Configuration Options”界面：设置其他的参数和选项；
   * Project->Configure RTL Pins…，打开“Pin Interface Definition”界面：设置RTL顶层模块端口的接口类型，Testbench与DUT之间通过设置的接口进行通信；
   * Project->Configure Testbench…，打开“Configure Testbench”界面：设置testbench的类型、名字，生成testbench的模板文件。
3. 编译Emulation（Build->Build Full Emulation）或编译Simulation（Build->Build Full Simulation）。在编译Emulation时，Linux系统必须已经安装Vivado软件；在编译Simulation时，Linux系统必须已经安装VCS或Modelsim软件。并且软件启动程序所在的路径必须添加到PATH变量中，以保证Semu能找到安装的软件。
4. 执行
   1. Emulation
   * Board->Load Bit File By PCIE（Board->Load Bit File By JTAG），有两种向FPGA板子加载BIT文件的方式，使用JTAG下载需要连接JTAG下载线；在加载完BIT文件后，FPGA中的电路会被Semu复位；
   * Run->Start Emulation，启动Emulation。
   1. Simulation
   * Run->Start Simulation，启动Simulation。

对于C++ Testbench、Simulation Testbench两种Testbench，运行Emulation、Simulation时存在界面上的差别：对于C++ Testbench，运行Emulation、Simulation时支持界面及对应操作；而对于Simulation Testbench，运行Emulation、Simulation时仅支持第三方仿真器的界面（目前仅支持VCS的DVE），原有“Emulation Control Panel”界面或“Simulation Control Panel” 界面上的一部分操作已经转化为API函数供用户使用，这些API函数也可以在C++ Testbench中使用。

对于C++ Testbench

1. 通过“Emulation Control Panel”界面或“Simulation Control Panel” 界面控制Emulation或Simulation过程，进行调试。
2. 从“Emulation Control Panel” 界面控制Emulation过程
   * 从信号列表中选择需要观察的信号添加到信号集合中，每次Emulation前可以动态选择被观察的信号；
   * 如果使用断点功能则设置断点；
   * Collection->Save Collection，保存信号集合；
   * 使能需要观察的信号；
   * 运行Emulation（RUN、RUN<N>），从“Emulation Control Panel”界面中控制时钟推进仿真时间。
3. 从“Simulation Control Panel”界面控制Simulation过程
   * 运行Simulation（RUN、RUN<N>），从“Simulation Control Panel”界面中控制时钟推进仿真时间。

对于Simulation Testbench

1. 可以选择是否启动第三方仿真器的界面（目前仅支持VCS的DVE）。如果不启动界面，那么Semu软件会打开一个新的xterm，并在其中运行仿真程序；如果启动界面，那么可以像操作第三方仿真器一样操作界面。

## 配置工程

本节描述工程的输入文件、配置过程、输出文件。

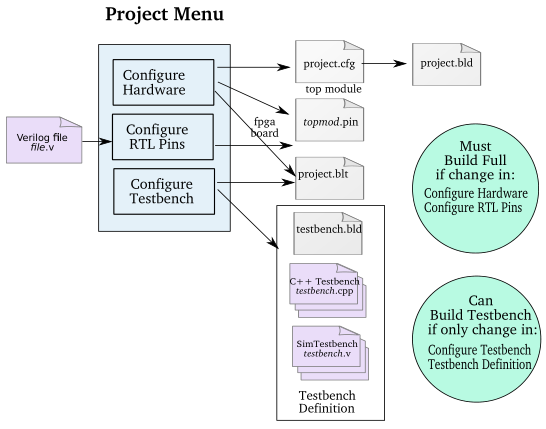


图 3.1 配置流程及输出文件

### 3.1 输入文件

Semu软件将FPGA器件中的待测试设计（DUT）通过计算机与用户的Testbench连接起来。其中，DUT的设计源文件和Testbench文件由用户提供，作为工程的输入文件。

#### 3.1.1 DUT设计源文件

DUT可以由一个或多个RTL文件组成。源文件可以放在任何文件夹中。顶层文件路径、模块名等信息在“Configure Hardware”界面中输入。

设计文件可以2种方式指定：

1. 在“Configure Hardware”界面中选择“Design Info”页，在第一项“Add File Style”下拉菜单中选择gui的方式，并在“Design Files”中选择对应文件夹中设计文件的路径；
2. 在“Configure Hardware”界面中选择“Design Info”页，在第一项“Add File Style”下拉菜单中选择cmd的方式，并在“Design Files”中选择对应文件夹选择设计文件列表（filelist）的路径。

另外，如果有include的文件，那么也需要在“Configure Hardware”界面中选择“Design Info”页，选择所需include文件的路径。具体添加方式详见文档[3.3.2 界面操作介绍](#_3.3.2_界面操作介绍)小节。

DUT的顶层模块是必须符合以下要求：

1. 顶层模块必须有且仅有一个时钟信号输入端口和一个复位信号输入端口。Semu软件目前仅支持单时钟域设计。如果DUT顶层模块端口中包含了多个时钟和复位，那么需要用户写一个Wrapper将原先的顶层模块封装成只有一个时钟信号输入端口和一个复位信号输入端口的模块，这个Wrapper就是新的顶层模块；
2. 目前不支持顶层模块包含inout类型端口，如果顶层模块包含inout端口，那么需要生成一个Wrapper将原先的顶层模块的inout端口分离成input、output端口，这个Wrapper就是新的顶层模块；
3. 例化的顶层模块的所有端口必须被连接，不能存在没有连接的端口；
4. 如果有include的文件，那么必须显式在每个使用到include中内容的Verilog文件中声明；
5. DUT的顶层文件必须是用Verilog语言描述的，且端口声明时每行仅允许声明一个端口。

#### 3.1.2 Testbench文件

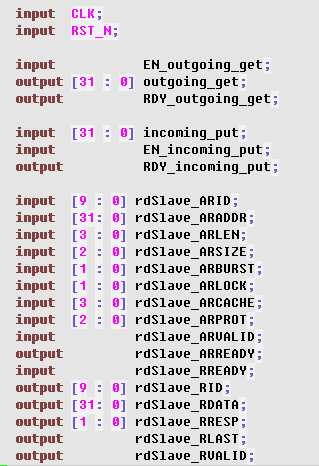
Semu支持以下3种类型的Testbench：

1. Manual Testbench：用户可以在“Simulation Control Panel”界面中手动输入发送到DUT的数据，同样在此界面中可以获得DUT的输出并显示。Manual Testbench可以在Simulation环境和Emulation环境中使用；
2. C++ Testbench：用户可以使用Semu的C-API和Semu生成的transactor写C++ Testbench，从而达到与DUT通信的目的。Semu生成针对待测DUT的接口文件，并提供模板文件。C++ Testbench可以在Simulation环境和Emulation环境中使用；
3. Simulation Testbench：用户直接在Testbench中将DUT的例化替换为Semu自动生成的DUT\_Interface的例化即可，实现原有Verilog/SystemVerilog Testbench到Semu的快速移植。Simulation Testbench的限制如下：
   * Simulation Testbench仅支持general类型工程中的port类型接口；
   * Simulation Testbench仅支持Emulation，不支持Simulation；
   * Simulation Testbench仅支持VCS仿真器；
   * Simulation Testbench仅支持VCS仿真器的DVE界面，不支持Semu原有的“Emulation Control Panel”界面或“Simulation Control Panel” 界面上的一部分操作已经转化为API函数供用户使用，这些API函数也可以在C++ Testbench中使用；

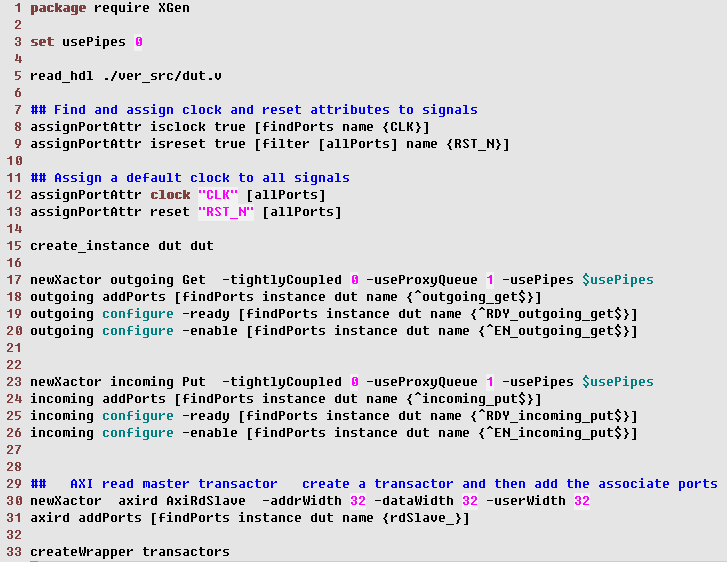
#### 3.1.3 t\_gen.tcl文件

如果工程类型是sdt，那么还需要用户手写t\_gen.tcl文件。t\_gen.tcl文件的作用是生成描述硬件工程的bsv文件和testbench模板文件。t\_gen.tcl文件应该存放在工程根目录下。t\_gen.tcl文件的写法举例如下，详细信息请参见附录。

DUT端口列表如下：



DUT的端口可以被分类为Get、Put、AxiRdSlave这3种。t\_gen.tcl文件如下：



第1行：添加需要用到的package；

第3行：设置一个后面会用到的变量；

第5行：读取DUT的顶层文件，注意不需要读取DUT的全部文件，仅需读取顶层文件即可；

第8-13行：设置Semu工程中DUT的时钟和复位；

第15行：给DUT顶层模块名取一个别名；

第17-20行：将DUT顶层模块端口中的3个端口组织成一个Semu的Get接口；其中第17行生成一个Get类型的接口，接口名字为outgoing，其后为接口的配置信息；第18行指定outgoing接口的数据端口；第19行指定outgoing接口的ready信号；第20行指定outgoing接口的enable信号；

第23-26行：将DUT顶层模块端口中的3个端口组织成一个Semu的Put接口；端口指定与Get接口一致；

第30-31行：将DUT顶层模块端口中的以rdSlave\_开头的所有端口组织成一个Semu的AxiRdSlave接口；端口指定与Get接口一致；

第33行：根据上述配置生成transactor等相关文件；

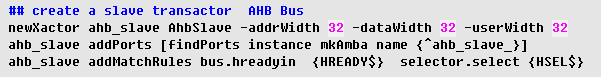
上述t\_gen.tcl中使用到的命令描述如表3-1所示：

表3-1 t\_gen.tcl中命令描述

|  |  |
| --- | --- |
| **命令** | **描述** |
| read\_hdl | 读取DUT顶层文件 |
| assignPortAttr | 指定时钟、复位端口及其属性 |
| allPorts | 获取所有的端口 |
| create\_instance | 给DUT顶层模块取别名或生成一个instance |
| newXactor | 生成一种接口实例，可用的配置项如下：  -addrWidth：地址位宽  -dataWidth：数据位宽  -userWidth：暂时不使用  -idWidth：id位宽 |
| addPorts | 给指定的接口实例添加端口，即将DUT中的端口组织分配给生成的接口实例 |
| findPorts | 从指定的instance中查找端口，端口的名字采用正则表达式匹配 |

下面给出其他几种常用接口的t\_gen.tcl文件的写法：











注意：t\_gen.tcl文件中指定的AMBA总线接口的信号个数和名字必须与工具内部规定的名字完全一致，位宽完全一致（可配置的除外），否则在生成接口时会报错。

### 3.2 Project菜单

#### 3.2.1 project菜单概述

Project菜单主要涉及对工程的新建、打开、关闭、配置等功能。



图 3.2 Project菜单

“Alt+标注下划线的字母（大小写均可）”可以展开相应的菜单，或执行相应的命令。比如：“Alt+P”展开Project菜单；在展开Porject时，“Alt+C”关闭工程。

#### 3.2.2 新建工程

Semu软件的基本工作单元是一个工程。当新建一个工程时，一个与工程同名的文件夹会同时被创建（如果同名文件夹已经存在，那么不重新创建文件夹），工程的所有输出文件都存放在该目录下（称为工程根目录）。DUT设计源文件、testbench等输入文件可以存放在任何目录。

新建工程，点击“Project->New”，弹出“New Project”对话框。

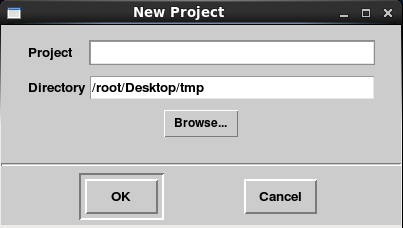


图 3.3 New Project对话框

点击Browse按钮修改Directory的路径，手动修改亦可。然后在Project选项框中输入工程名，点击OK即可。点击OK后，会弹出“Configure Hardware”对话框，具体设置方法在[3.3 Configure Hardware](#_3.3_Configure_Hardware)中详细描述。点击Cancel按钮可以取消新建工程。



图 3.4 工程配置过程

在一个工程的配置过程中，共有3个界面需要配置。这3个界面可以通过Project菜单打开，也可以在配置完成一个界面后直接进入下一个界面。这3个界面分别是：

1. Configure Hardware；
2. Configure RTL Pins；
3. Configure Testbench。

后面会详细介绍这3个界面的配置方法。

#### 3.2.3 打开工程

打开工程，点击“Project->Open”，弹出“Semu Project File”对话框。

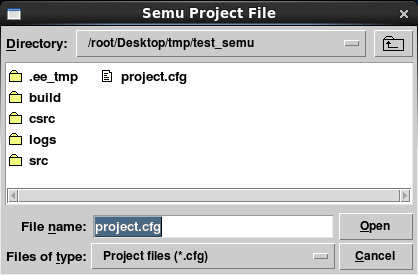


图 3.5 Semu Project File对话框

找到并选中对应工程的project.cfg文件，点击Open即可。另外，可以在启动Semu软件图形界面时打开工程，命令为“semu *path/*project.cfg”。

#### 3.2.4 Configure Hardware

在[3.3 Configure Hardware](#_3.3_Configure_Hardware)中详细描述。

#### 3.2.5 Configure RTL Pins

在[3.4 Configure RTL Pins](#_3.4_Configure_RTL)中详细描述。

#### 3.2.6 Configure Testbench

在[3.5 Configure Testbench](#_3.5_Configure_Testbench)中详细描述。

#### 3.2.7 Save as

工程另存为，点击“Project->Save as”，打开“Save as”对话框。将打开的工程的整个工程文件夹复制到指定的目录下，并可以重新命名。

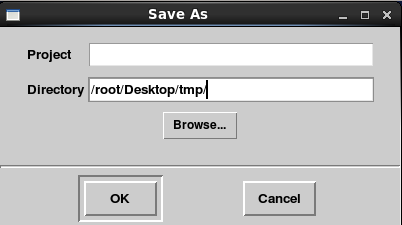


图 3.6 Save as对话框

点击Browse按钮修改Directory的路径，手动修改亦可。然后在Project选项框中输入工程名，点击OK即可，操作完成后会弹出保存完毕对话框。点击Cancel按钮可以取消Save as。



图 3.7 保存完毕对话框

### 3.3 Configure Hardware

#### 3.3.1 基本配置

“New Project”对话框配置完成并点击OK后，会自动弹出“Configure Hardware”对话框。也可以通过点击“Project->Configure Hardware”打开“Configure Hardware”对话框。“Configure Hardware”对话框分为三页，分别为：Project Info、Design Info、DDR3 Info。

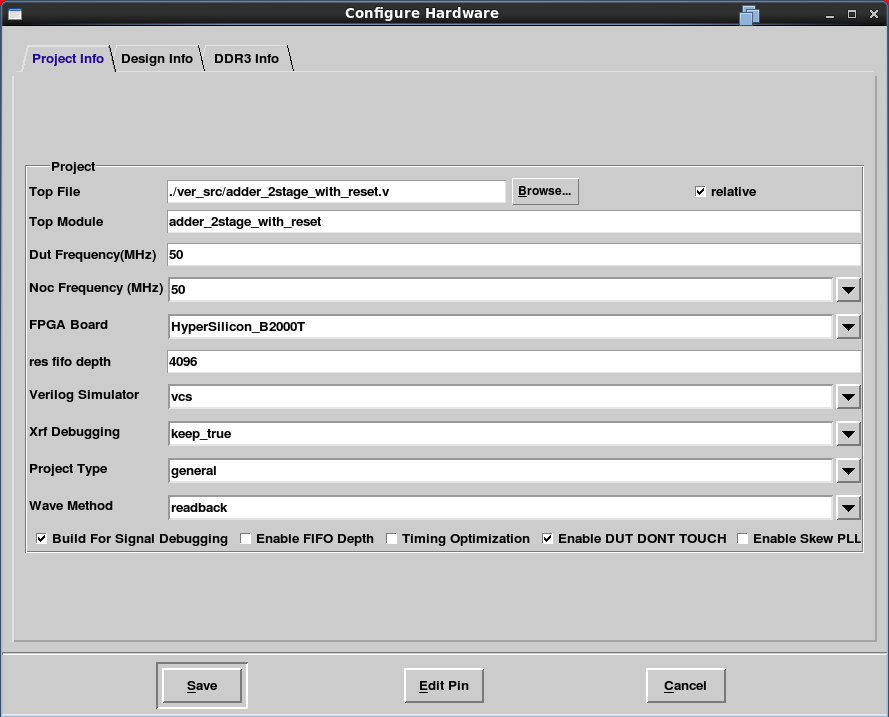


图 3.8 Project Info对话框

用户可以在“Configure Hardware”对话框中配置工程的硬件参数。这些参数将被写入project.cfg文件中。

在“Project Info”页面中的配置描述工程的信息：

* Top File：DUT设计工程中的顶层模块所在的文件（以下简称为顶层文件）的路径及文件名，顶层文件必须是Verilog文件。可以通过Browse按钮找到指定的文件，也可以手动输入。顶层文件可以随意存放在任何目录下。在设置顶层文件之前可以通过是否勾选relative选项决定采用相对路径还是决定路径，勾选relative选项则采用相对路径。默认勾选relative选项。
* Top Module：顶层文件中的顶层模块的模块名。生成的pin-file与top-module同名。
* Dut Frequency(Mhz)：FPGA中的DUT工程的受控时钟的频率。
* Noc Frequency(Mhz)：FPGA中的NoC时钟的频率。
* Fpga Board：从下拉菜单中选择Semu软件支持的Emulation板子的型号，目前仅支持Xilinx\_KC705、 HyperSilicon\_B2000T、HyperSilicon\_K115和HyperSilicon\_E4000三种硬件平台型号。
* res fifo depth：与Enable Fifo Depth单选项配合使用，缓冲DUT输入输出数据的FIFO的深度的配置，默认数值4096，Enable Fifo Depth单选项被选择时该数值才有意义。
* Verilog Simulator：从下拉菜单中选择Semu软件支持的且已经安装的仿真器。Semu软件通过查看PATH环境变量寻找已经安装的仿真器，并且显示所有支持的且已经安装的仿真器。如果PATH环境变量所指向的路径下找不到仿真器，那么下拉菜单中不会显示仿真器。目前Semu支持的仿真器包括：VCS、Modelsim。
* Xrf Debugging：选择是否对Semu硬件工程中的Verilog代码添加none、keep\_true和dont\_touch属性。
* Project Type：指定工程类型，可选的类型分为两种：general和sdt。其中sdt类型需要手写t\_gen.tcl文件，并存放在工程目录下。
* Wave Method：指定获取波形的方式，可选readback、probe。readback方式支持获取DUT内部所有寄存器类型信号的波形，每次获取新增信号的波形时不需要重新编译工程，支持硬件断点；probe方式支持获取DUT内部所有类型信号的波形（除了多维数组信号、信号路径中带有中括号的信号之外），每次获取新增信号的波形时需要重新编译工程，最多支持获取10752-bit信号的波形。
* Build For Signal Debugging：勾选时支持波形回读功能，默认勾选该选项。
* Enable Fifo Depth：勾选时允许用户配置缓冲DUT输入输出数据的FIFO的深度（深度由res fifo depth中的数值指定），默认不勾选。
* Timing Optimization：决定是否使用Vivado编译优化功能。当该选项被勾选时，提高运行频率的优化功能设置会被添加到Vivado工程中，但在readback波形回读方式下会削弱对信号的可见性，默认不勾选该选项。
* Enable Dut DONT TOUCH：勾选时软件会对DUT中的所有寄存器信号添加DONT\_TOUCH属性，使得尽量保持DUT中的寄存器不被优化掉，获得更好的信号可见性，默认不勾选。
* Enable Skew PLL：该选项用于选择是否在FPGA内部逻辑中使用PLL生成提供给DUT的时钟（时钟之间存在相位对齐），勾选后采用PLL。建议仅在DUT的输入时钟存在相位对齐关系时才勾选该项，否则不勾选，使用默认的时钟生成单元。

在“Desing Info”页面中的配置描述DUT的信息：

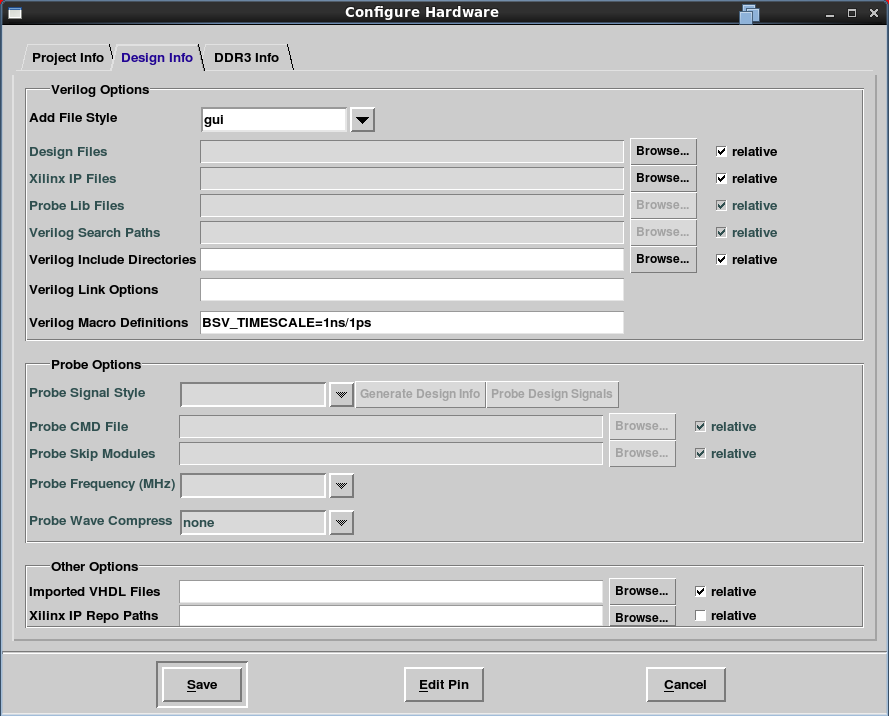


图 3.9 Design Info对话框

* Add File Style：添加工程源文件的方式选择。可选项：cmd、gui。其中cmd是指以文件列表的方式添加工程源文件，点击Browse按钮即可添加；gui是指通过界面添加工程源文件，点击添加文件的各项对应的Browse按钮即可打开添加工程源文件的界面。
* Design Files：DUT的设计文件列表，可以在readback、Probe两种波形获取方式的工程中使用，设计文件列表中的所有文件会被添加到Vivado工程中。在Probe波形获取方式下，此项必须指定；在readback波形获取方式下，此项可以指定，也可以不指定。如果指定，那么仅允许指定一个filelist。该选项与“Verilog Search Paths”类似，可以选择一种使用，也可以两种同时使用，但应避免重复添加文件。
* Xilinx IP Files：Vivado工程中例化的IP的文件列表，这些文件会被加入到Vivado工程中。也可以添加VHDL文件。
* Probe Lib Files：Probe获取波形方式下的库文件列表；库是指Vivado工程中使用Xilinx的原语或IP等，对于Xilinx原语（比如BUFG等）用户可以写对应的模型（仅包含模块名和端口声明即可，无需包含内部逻辑），确保Semu不会因为工程文件不全导致运行失败。在设计工程中没有除了Verilog源码之外的设计文件时，该项可以不设置；如果工程中包含了除了Verilog源码之外的设计文件，那么必须设置此项。
* Verilog Search Paths：Verilog设计文件的搜索路径，搜索路径下的所有文件会被添加到Vivado工程中。如果Verilog设计文件被存放在多个不同的目录下，那么可以将除了顶层文件所在的目录之外的所有目录的路径添加到这个选项中，各个路径之间使用空格隔开。点击Browse按钮可以选择指定的目录。relative选项决定写入project.cfg文件中的路径是决定路径还是相对路径。该选项与“Design File List”类似，可以选择一种使用，也可以两种同时使用，但应避免重复添加文件。
* Verilog Include Directories：Verilog文件中include的其他文件的搜索路径。
* Verilog Link Options：bsc的verilog link process的输入参数。可以向仿真器传递宏定义、仿真器命令选项。参见《Bluespec SystemVerilog and Bluespec Development Workstation User Guide.pdf》。
* Verilog Macro Definitions：设置DUT工程中的宏定义。另外，Semu工程默认DUT复位低电平有效，如果想修改为复位高电平有效，可以在此处设置宏定义BSV\_POSITIVE\_RESET即可。
* Probe Signal Style：添加Probe波形获取方式的信号时的添加方式选择。可选项：cmd、gui。其中cmd是指以文件的方式添加Probe信号，点击Browse按钮即可添加；gui是指通过界面添加Probe信号，点击添加文件的各项对应的Browse按钮即可打开添加Probe信号的界面。
* Probe CMD File：Probe获取波形方式下的PROBE信号描述文件；描述待采样信号的路径。
* Probe Skip Modules：Probe获取波形方式下的库模块名列表文件；库模块名是指库的模块名，Semu在获取指定层次及其以下的信号时会在遇到库模块时自动终止，即不会获取库模块及库模块以下层次的所有信号的波形。在设计工程中没有除了Verilog源码之外的设计文件时，该项可以不设置；如果工程中包含了除了Verilog源码之外的设计文件，那么必须设置此项。
* Probe Frequency (MHz)：Probe获取波形方式下的采样时钟频率。
* Probe Wave Compress：默认none，暂时无效。
* Imported VHDL Files：DUT设计文件中包含的VHDL文件的路径及文件名。多个文件之间通过空格隔开。点击Browse按钮可以选择指定的文件。relative选项决定写入project.cfg文件中的路径是决定路径还是相对路径。
* Supplemental xdc Files：用户书写的针对DUT的XDC约束文件的路径。

点击Save按钮保存配置或修改，点击Cancel按钮取消配置或修改。

在“DDR3 Info”页面中可以对FPGA板上外部DDR3的使用方式进行设置。

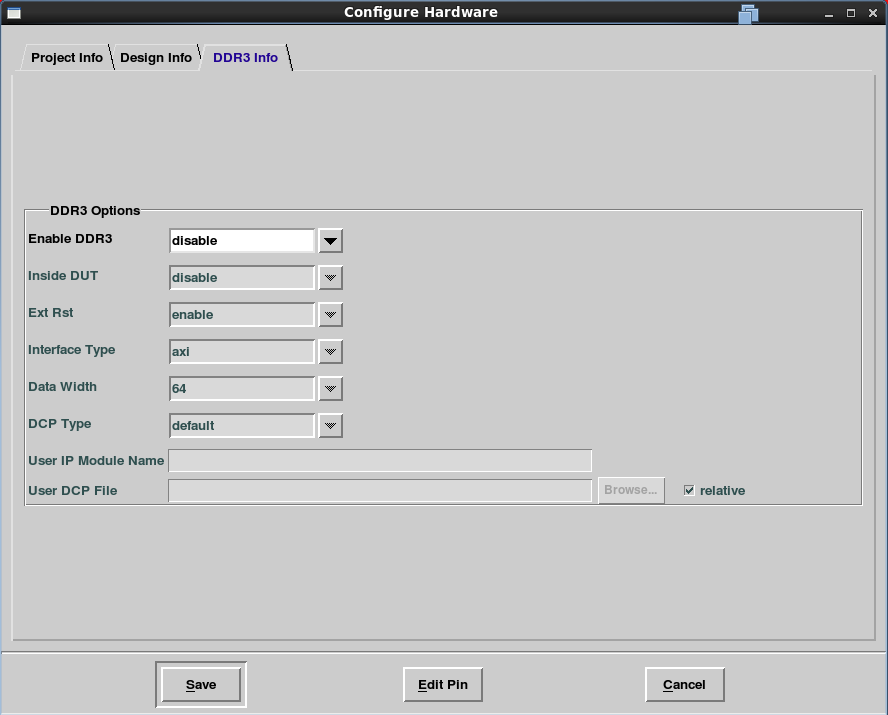


图 3.10 DDR3 Info对话框

* Enable DDR3：是否使用FPGA板上的DDR3。其中，disable为不使用；enable为使用。
* Inside DUT：DDR3的读写控制器是否在DUT内部，或者仅在DUT端口上留出DDR3控制器接口，通过工具自动添加DDR3控制器。其中，disable为后者；enable为前者。
* Ext Rst：DDR3控制器的复位是否使用外部按键复位。其中，disable为使用内部提供的复位信号，在每次启动Emulation时自动复位DDR3控制器；enable为使用外部按键复位，每次启动Emulation时不自动复位DDR3控制器。
* Interface Type：DDR3控制器的用户侧接口类型。可选为axi、native。
* Data Width：DDR3控制器的用户侧接口的数据位宽。可选为axi接口类型支持32、64、128、256、512；native接口类型支持256、512。
* DCP Type：DDR3控制器的类型选择。可选为default、custom。其中default为使用Semu默认的DDR3控制器IP例化生成的dcp文件（默认的DDR3 IP的配置请参见安装包下的lib/board\_support/bluenoc/hypersilicon/B2000T/ddr/ddr3/pdf路径中的DDR3控制器IP配置文档）；custom为使用用户生成的DDR3控制器IP（生成DDR3控制器IP时的管脚配置请加载安装包下的lib/board\_support/bluenoc/hypersilicon/B2000T/ddr/ddr3/xdc\_ucf/b2000t\_ddr3\_pin\_loc.ucf（B2000T）或者lib/board\_support/bluenoc/hypersilicon/KU115/ddr/ddr3/xdc\_ucf/example\_design\_J11J12J13.xdc（K115））。
* User IP Module Name：用户例化的DDR3控制器IP的模块名。
* User DCP File：用户例化的DDR3控制器IP的dcp文件的路径。

上述所有的工程配置存储在project.cfg文件中，以B2000T的配置为例，生成的project.cfg文件如下：



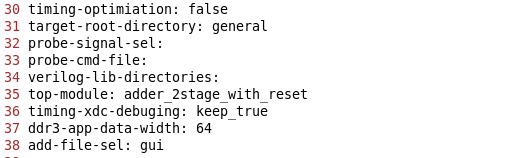


图 3.11 project.cfg文件

在“Configure Hardware”对话框中，点击Save按钮保存配置或修改，点击Cancel按钮取消配置或修改。点击Save按钮后，Semu软件会执行“build –i *path*/project.cfg”命令，生成project.bld、project.blt等文件。

每次修改工程配置之后，Semu软件会提示重新编译工程。并执行“build –i /*path*/project.cfg”，生成新的project.bld文件，旧的project.bld文件会被保存为project.bld~*x*（其中x是从1开始递增的数字，每次重新生成一次project.bld文件，x的值就自加1）

点击“Edit Pin”按钮可以打开“Configure RTL Pins”对话框，配置并生成pin-file文件。

#### 3.3.2 界面操作介绍

为了方便初学的用户使用Semu，用户可以通过界面添加设计工程源文件，也可以通过界面添加Probe波形获取方式中的信号。

配置界面如下图所示：

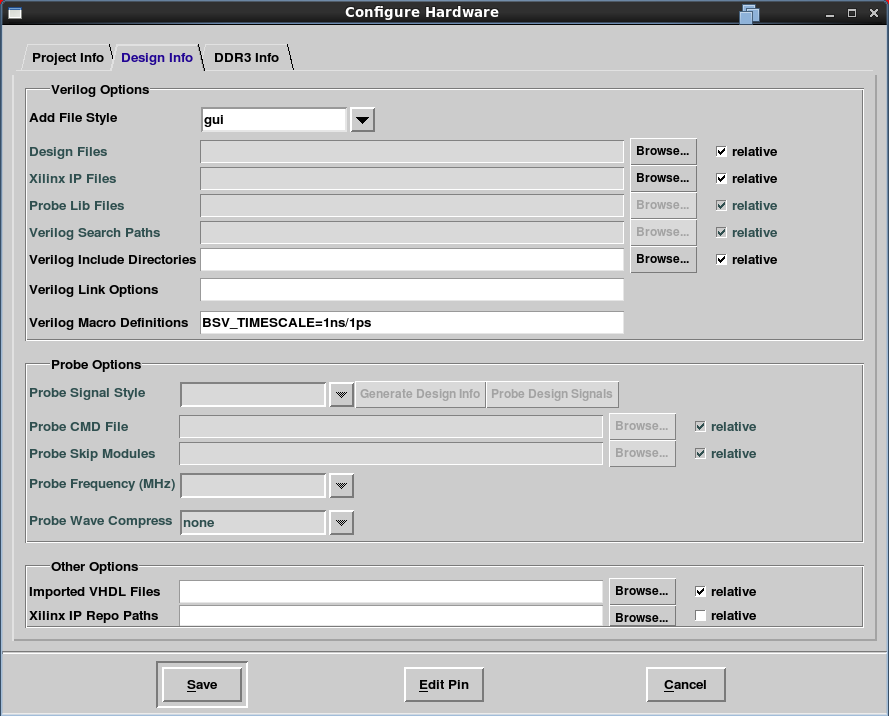


图 3.12 配置界面

**通过界面添加文件**

其中，通过配置Add File Style，可以选择添加工程源文件的方式。可选项：cmd、gui。当配置为gui时，可以通过界面添加工程源文件，点击添加文件的各项对应的Browse按钮即可打开添加工程源文件的界面。

例如，点击“Design File List”右侧的Browse按钮，会弹出如下图所示的添加文件的窗口。

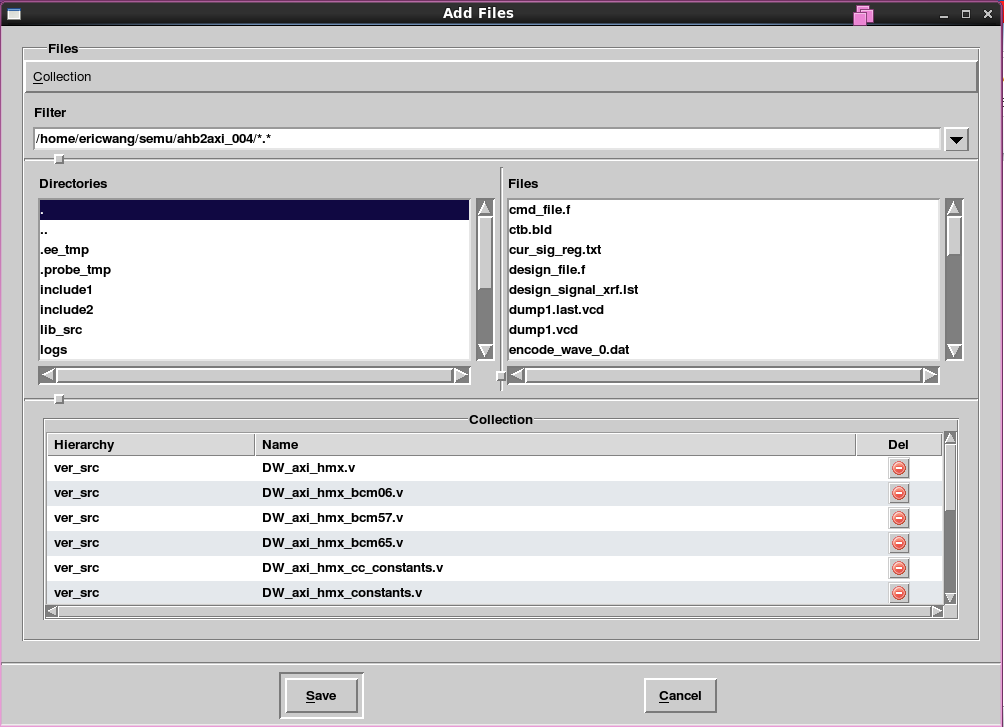


图 3.13 添加文件界面

该界面分为5个部分：

1. 菜单栏：包含了Collection菜单项，菜单中包括了对Collection窗格中信息的处理，具体如下：
   * Load Collection：将指定文件中的信息加载到Collection窗格中。
   * Save Collection：将Collection窗格中所有信息保存到默认的文件中。
   * Save Collection As：将Collection窗格中所有信息保存到指定的文件中。
   * Clear：删除Collection窗格中的所有信息。
2. Filter：文件过滤器，可以在该窗口栏中输入需要匹配的文件名（或信号名）的匹配字符串。比如，“\*.v”匹配在Filter中的指定目录下的所有以“.v”结尾的文件；“axi\_\*.v”匹配在Filter中的指定目录下的所有前缀为“axi\_”并以“.v”结尾的文件。
3. Directories：Filter中的指定目录（或层次）下的所有子目录（或子层次）。
4. Files：Filter中的指定目录（或层次）下的所有符合Filter过滤条件的文件（或信号）。
5. Collection：添加的文件（或信号）的列表。

添加文件的窗格支持两种操作：

* 双击文件：将单一文件添加到Collection窗格中。
* 多选文件后点击鼠标中键：将多个文件添加到Collection窗格中。在多选文件时支持Ctrl、Shift、Home、End等组合键。

**通过界面添加Probe信号**

添加Probe信号的操作步骤如下：

1. 添加工程源文件，包括“Design File List”、“Xilinx IP File List”（如果有）、“Verilog Include Directories” （如果有）、“Probe Lib File” （如果有）。
2. 点击“Generate Design Info”按钮，直到弹出如下提示框，确保正确获取设计工程中的层次及信号信息。



如果弹出如下提示框，那么提示获取设计工程中的层次及信号信息失败，需要根据Semu窗口中的错误提示信息排除错误后，再次点击“Generate Design Info”按钮，确保正确获取设计工程中的层次及信号信息。



1. 点击“Probe Design Signals”按钮，弹出如下对话框，添加指定的信号到“Signal Collection”窗格中，最后点击Save按钮，将添加的信号保存到默认的文件中。如果不保存，可以点击Cancel按钮，取消本次添加动作。

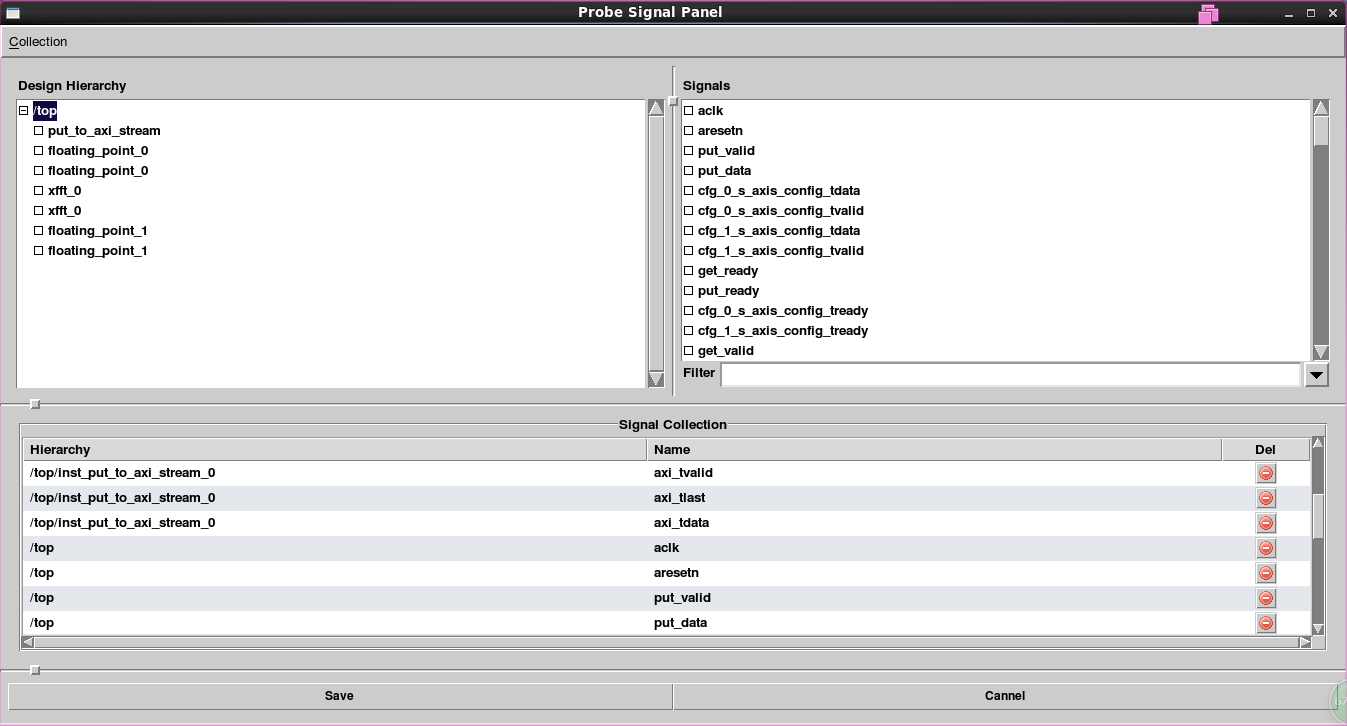


图 3.14 probe获取信号界面

添加信号时支持三种操作：

1. 在Signal窗格中点击鼠标左键选中一个信号，点击右键，点击“Add and Enable”，将选中的单一信号添加到Signal Collection窗格中。
2. 在Signal窗格中按住Ctrl键同时点击鼠标左键选中多个信号，点击右键，点击“Add and Enable”，将选中的多个信号添加到Signal Collection窗格中。
3. 在Design Hierarchy窗格中，选中指定层次后，点击鼠标右键，将该层次的指定类型的信号全部添加到Signal Collection窗格中。支持的信号类型如下：
   * Add All Signals：该层次中的所有信号。
   * Add Port Signals：该层次中的所有端口信号。
   * Add Input Signals：该层次中的所有输入信号。
   * Add Output Signals：该层次中的所有输出信号。
   * Add InOut Signals：该层次中的所有双向输入输出信号。
   * Add Inner Signals：该层次中的所有内部（非端口）信号。

注意：获取设计工程中的层次及信号信息功能要求安装Verdi，而且正确设置环境变量，参见“9.2.1 准备工作”。

### 3.4 Configure RTL Pins

Semu的工程类型分为两种：general和sdt。

对于general类型工程，pin-file文件将DUT顶层模块的输入、输出端口组织成接口。Semu软件根据pin-file中接口的分类自动生成testbench与DUT相互通信的transactor，不同的接口类型会生成不同的transactor；

对于sdt类型工程，用户需要手写t\_gen.tcl文件代替pin-file文件，t\_gen.tcl文件的详细信息请参见[3.1.3 t\_gen.tcl文件](#_3.1.3_t_gen.tcl文件)和[附录](#_附录)。

#### 3.4.1 接口类型

目前，Semu支持2种接口类型：Ports、Handshake (Ready/Enable)。

1. Port接口

Port接口是标准的RTL的输入输出端口。每个input、output端口通过Semu生成的transactor直接完成testbench与DUT之间的通信。Semu软件为每个端口生成C-API，通过C-API可以实现testbench分别向DUT指定端口发送数据或接收DUT指定端口返回的数据。设计中可以有0、1或更多的端口。目前，Semu不支持inout类型的端口。

1. Handshake (Ready/Enable)接口

Handshake接口实现了一个Ready/Enable协议。设计中可以包含0、1或更多的handshake接口。每个Handshake接口由3部分组成：

* + 一个或多个输入、输出数据端口；在Handshake接口中的所有的数据端口必须是同向的（输入或输出）。
  + 一个输出的单比特RDY（raedy）端口。
  + 一个输入的单比特EN（enable）端口。

Ready/Enable协议如下：

* + 当DUT准备好后，DUT将RDY信号拉高，并且RDY信号是否拉高与输入的EN信号和输入的数据总线无关。特别注意，从EN到RDY之间不应该存在组合逻辑。
  + 只有在DUT将RDY信号拉高后，Semu才可以将EN信号拉高。
  + 在RDY和EN都为高电平时才进行数据交换。
  + 在DUT将RDY信号拉高后，DUT可以在EN没有被拉高前将RDY信号拉低。

#### 3.4.2 接口定义

接口定义是在“Pin Interface Definition”界面中完成的，打开方式有如下两种：

1. 在“Configure Hardware”对话框中点击“Edit Pin”按钮
2. 在Project菜单中点击“Configure RTL Pins …”

“Pin Interface Definition”界面由5部分组成：

* + 标题栏；
  + 菜单栏；
  + Unconnected Pins列表栏：顶层模块的管脚，未组织成Interface；
  + Interfaces列表栏：顶层模块的管脚组织成的接口；
  + 快捷按钮栏。

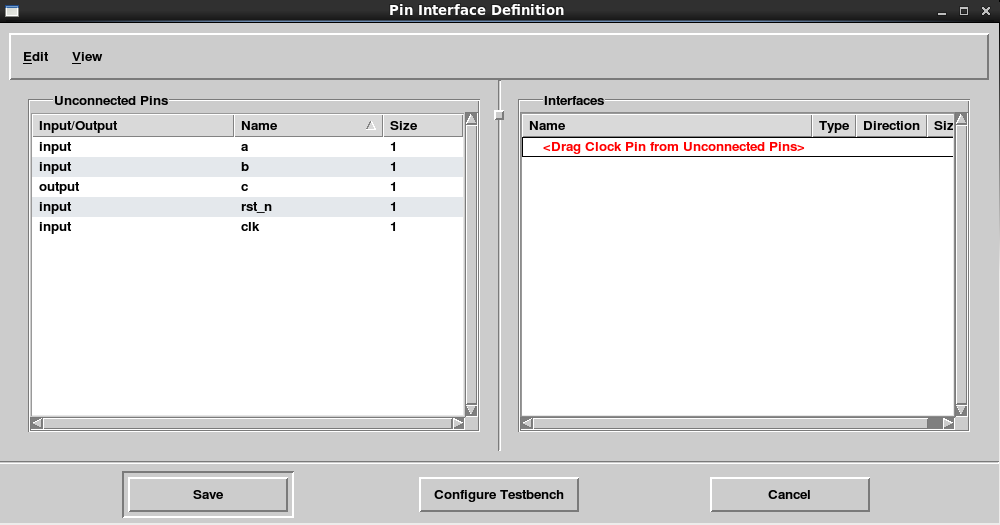


图 3.15 Pin Interface Definition界面

当第一次打开这个界面时，顶层模块的所有input、output端口都被列在“Unconnected Pins”栏中。当用户完成对顶层模块的管脚的配置后，“Unconnected Pins” 列表栏中的所有管脚都应被移到Interfaces列表栏中，并被组织成不同的接口。

在界面上进行接口定义的方法分为两种：拖拽、双击。

使用拖拽或双击的方式将“Unconnected Pins”列表栏中的1个端口移动到Interface列表中。Semu会指导用户完成定义接口的过程，并允许为新接口命名。

必须移动到Interface列表中的端口用红色显示，可选的端口用灰色显示。必须移动到Interface列表中的端口包括：

1. 时钟端口；
2. 复位端口；
3. 对于每个interface（ready/enable，memory request， memory response）：
   * ready端口；
   * enable端口；
   * data端口。

1个Handshake接口可以有1个或更多的数据端口。memory request和memory response接口只能有1个data端口。

时钟域由1个时钟信号和1个复位信号定义。每个接口必须隶属于1个指定的时钟域。目前，Semu仅支持单时钟域，即1个时钟信号和1个复位信号。如果DUT顶层模块端口包含了多个时钟和复位，那么需要生成一个Wrapper将原先的顶层模块封装成只有一个时钟管脚和一个复位管脚的模块，这个Wrapper就是新的顶层模块。

接口定义的步骤如下：

1. 定义时钟。将时钟信号从“Unconnected Pins”列表栏中拖拽到Interfaces列表中的红色区域或双击“Unconnected Pins”列表栏中的时钟信号。
2. 定义复位。将复位信号从“Unconnected Pins”列表栏中拖拽到Interfaces列表中的红色区域或采用双击的方式。这个复位信号是与已定义的时钟信号相关联的。

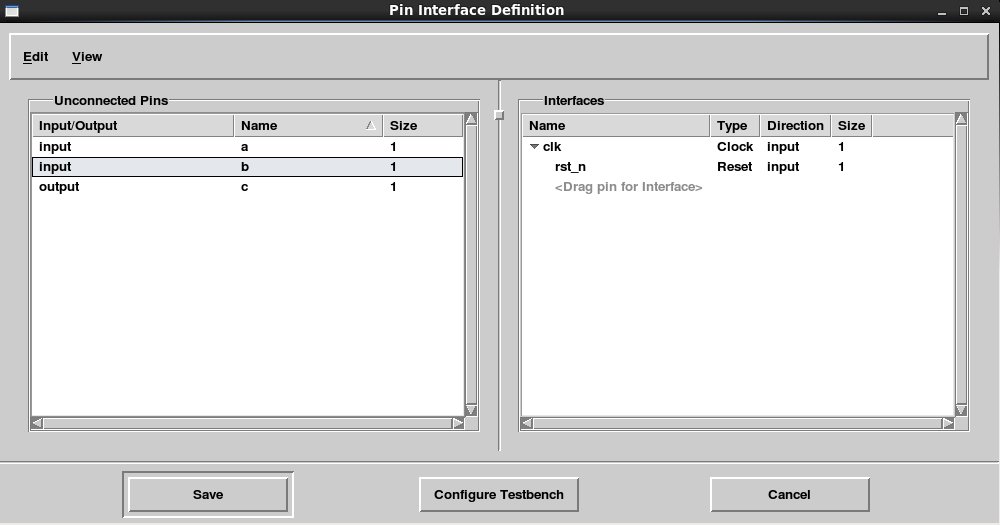


图 3.16 定义了时钟复位信号的Pin Interface Definition界面

1. 定义其他信号。将其他信号从“Unconnected Pins”列表中拖拽到Interfaces列表中的灰色区域或采用双击的方式，并指明接口的类型（Port，Handshake，Memory or Lockstep）。这些接口是在与已定义的时钟域中的。

选择的接口类型将会决定接下来显示的窗口内容。

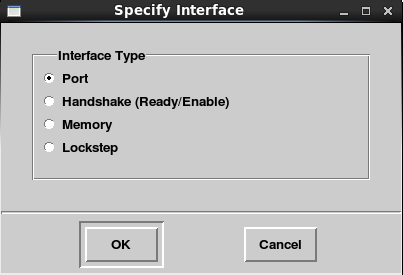


图 3.17 Specify Interface对话框

1. Port
2. Handshake
   * 如果端口信号的位宽大于1bit，那么一定是数据端口。
   * 如果端口信号的位宽为1bit，那么需要在弹出的“Specify Pin Type”对话框中选择端口类型：Data、Enable、Ready。其中Data类型始终有效，而Enable、Ready类型是否有效由端口方向决定。例如，Ready类型必须是输出端口；Enable类型必须是输入端口。
   * （可选）更改Handshake接口的名字。默认的接口名称是信号的名称。用户可以在“Specify Pin Type”对话框中的“Name of Interface”中修改，或在Interface窗格的接口名字中修改。
   * 举例：如果端口信号的位宽大于1bit，那么一定是数据端口。

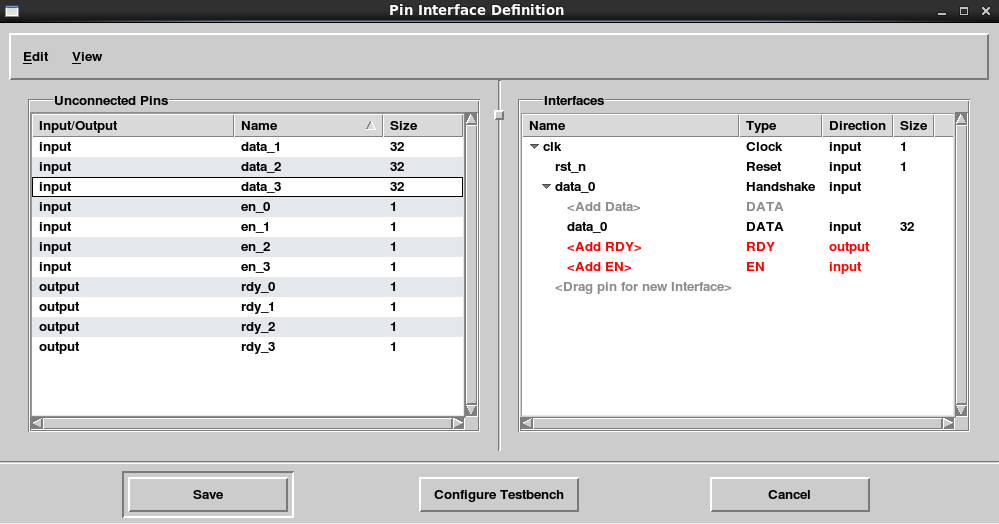


图 3.18 Handshake接口定义界面1

上图中的红色部分必须添加信号，而灰色部分可以不添加信号。Handshake接口必须有RDY和EN信号，选择信号拖拽到Interface窗格中的<Add RDY>行，选择信号拖拽到Interface窗格中的<Add EN>行。RDY和EN信号必须全部为1bit，否则会报错。Handshake接口至少需要有一个数据端口，如果有多个数据端口，那么一个Handshake接口内的所有数据端口必须全部为同向的，否则工具会报错。如果有多个数据端口，可以选择信号拖拽到Interface窗格中的<Add Data>行。上图中Interface窗格的第3行第1列为接口的名字，可以手动修改，默认为handshake接口第一个拖拽的信号的名字。

如果选择的信号为1bit的输入信号，那么会弹出如下窗口：

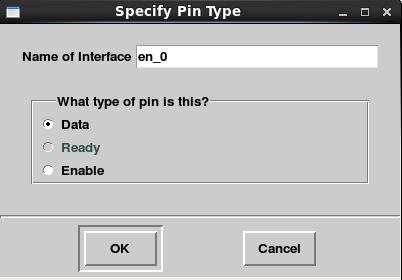


图 3.19 Handshake接口定义中1bit输入信号的Specify Pin Type对话框

“Name of Interface”中的内容为handshake接口的名字，可以手动修改，默认为信号的名称。接下来可以选择信号的种类：Data、Enable。由于en\_0信号为输入端口，所以Ready选项无效。选择Enable，点击OK按钮，可得到如下界面：

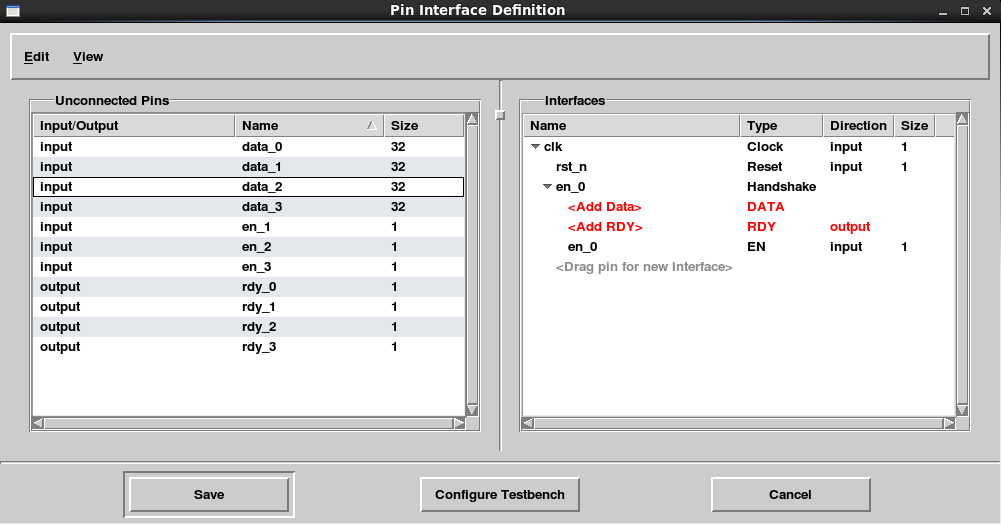


图 3.20 Handshake接口定义界面2

如果拖拽的信号为1bit的输出信号，那么会弹出如下窗口：

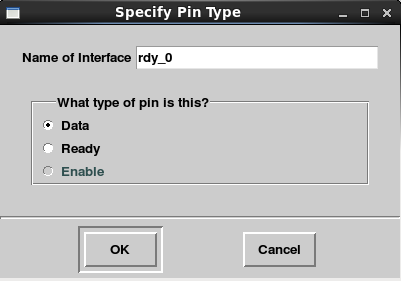


图 3.21 Handshake接口定义中1bit输出信号的Specify Pin Type对话框

“Name of Interface”中的内容为handshake接口的名字，可以手动修改，默认为信号的名称。接下来可以选择信号的种类：Data、Ready。由于rdy\_0信号为输入端口，所以Enable选项无效。选择Ready，点击OK按钮，可得到如下界面：

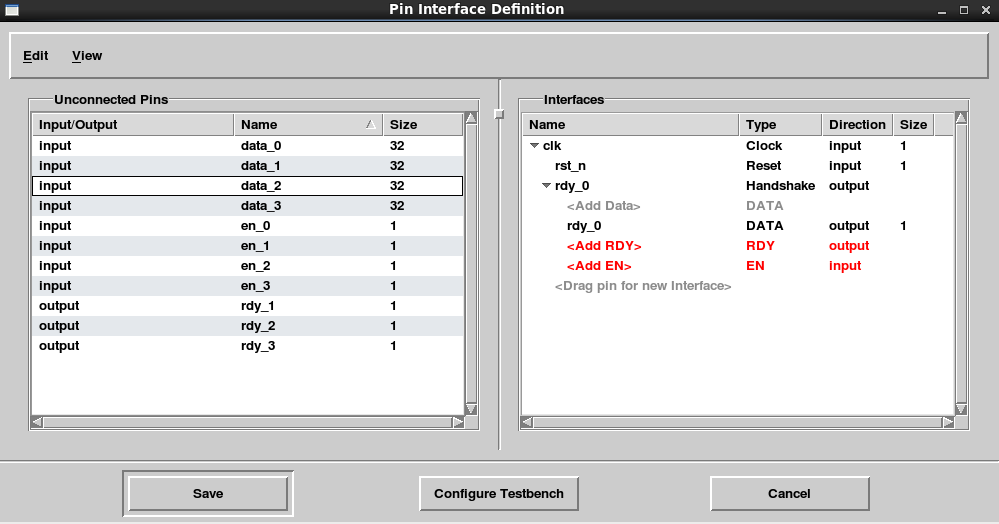


图 3.22 Handshake接口定义界面3

“Pin Interface Definition”界面的菜单栏包括“Edit”、“View”两个菜单。

1. Edit菜单
   * Move all unassigned pins to ports：将所有未分配的管脚设置为Port接口类型；
   * Make all pins Lockstep：将所有未分配的管脚设置为Lockstep接口类型。
2. View菜单
   * Expand All：展开所有接口层次；
   * Collapse All：折叠所有接口层次。

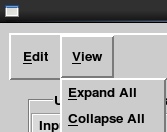
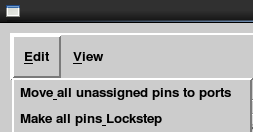


图 3.23 Pin Interface Definition界面的Edit、View菜单

如果工程目录下已经存在一个pin-file，那么点击“Project->Configure RTL Pins …”后，会弹出如下对话框：

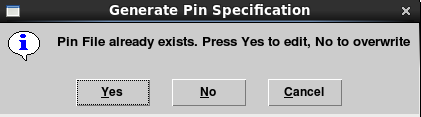


图 3.24 Generate Pin Specification对话框

点击Yes按钮，进入“Pin Interface Definition”界面，编辑pin-file；点击No按钮，进入“Pin Interface Definition”界面，清空pin-fie，重新设置pin-file；点击Cancel按钮，退出对话框。

#### 3.4.3 pin-file格式

pin-file格式要求如下：

1. 第一行必须以“//complete”开头，位于行首，无空白符。
2. 在port和 endport之间定义接口信号。其中port和endport各独占一行。
3. 时钟信号的定义格式如下：“端口类型 信号名 位宽 clock:CLK”。必须位于行首，无空白符。
4. 复位信号的定义格式如下：“端口类型 信号名 位宽 reset:RST\_N”。必须位于行首，无空白符。
5. Port接口信号的定义格式如下：“端口类型 信号名 位宽 CLK PIPE”。必须位于行首，无空白符。
6. Handshake接口信号的定义格式如下：“端口类型 信号名 位宽 CLK <RDY/EN/DATA> <PIPEPUT/PIPEGET> 接口名”。必须位于行首，无空白符。

注：上述的端口类型包括input、output，目前Semu软件不支持inout类型端口。

### 3.5 Configure Testbench

Semu允许在完成一次全部工程编译后多次重新编译testbench，而无需重新编译全部工程。当仅编译testbench时，Emulation板子的BIT文件和其他的目标文件不会被重新生成。这为每次修改testbench调试工程节省了时间。

Semu的运行环境包含1个DUT和1个testbench以完成对DUT的测试。在编译运行环境之前必须先指定testbench。指定testbench可以通过“Configure Testbench”界面完成。打开“Configure Testbench”界面的方法包括：

1. 点击“Pin Interface Definition”界面的“Configure Testbench”按钮；
2. 点击“Project->Configure Testbench …”；

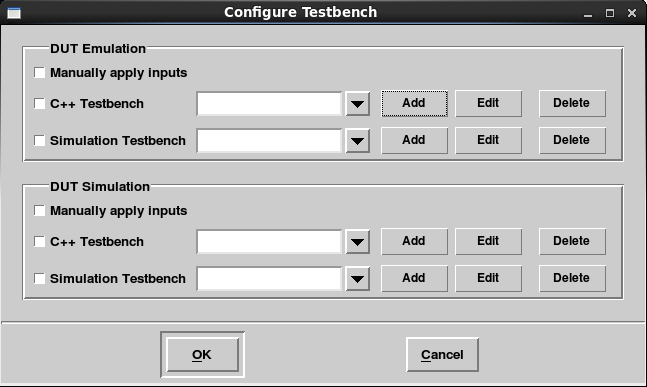


图 3.25 Configure Testbench对话框

“Configure Testbench”对话框分为两部分：DUT Emulation、DUT Simulation。每一部分都由3种testbench的选项及与testbench对应的其他功能按钮组成。在“DUT Emulation”窗格和“DUT Simulation” 窗格中，分别勾选2种testbench选项中的其中之一，然后可以通过对应的“Add”、“Edit”、“Delete”按钮及下拉菜单完成对testbench的配置。

1. 下拉框：选择当前工程使用的testbench；
2. Add：添加新的testbench，打开“C++ Testbench Definition”对话框；
3. Edit：编辑下拉框中的显示的testbench，打开“C++ Testbench Definition”对话框；
4. Delete：删除下拉框中的显示的testbench。

针对Emulation和Simulation，Semu目前支持2种类型的testbench（参见[3.1.1 Testbench文件](#_3.1.2_Testbench文件)），但每次只能编译和执行1个testbench。

Manual Testbench是在编译阶段由Semu生成的。C++ Testbench是用户提供的。工程中使用C++ Testbench和Simulation Testbench的步骤如下：

1. 点击“Project->Configure Testbench”，打开“Configure Testbench”对话框。勾选C++ Testbench或Simulation Testbench。点击Add按钮，打开“C++ Testbench Definition”或“Simulation Testbench Definition”对话框。
2. 在打开的对话框中设置添加的testbench的名称等信息，点击“Add Starter Templates”按钮，生成testbench模板文件。点击“Save”按钮，保存testbench的配置。
3. 在“Configure Testbench”对话框的下拉菜单中选择testbench的名称。点击“Save” 按钮，保存testbench的配置。
4. 在模板文件的基础上完成testbench（参见[10 C++ Testbench的写法](#_C++_Testbench的写法)章节）
5. 编译

#### 3.5.1 定义C++ Testbench

本小节描述如何生成工程的C++ Testbench。C++ Testbench的写法请参见[10 C++ Testbench的写法](#_C++_Testbench的写法)。

在Emulation运行环境中使用C++ Testbench时，勾选“DUT Emulation”窗格中的C++ Testbench以确定使用的testbench的类型。然后点击同一行的Add按钮，打开“C++ Testbench Definition”对话框，添加新的C++ Testbench。生成的testbench既可以提供给Emulation使用，也可以提供给Simulation使用。

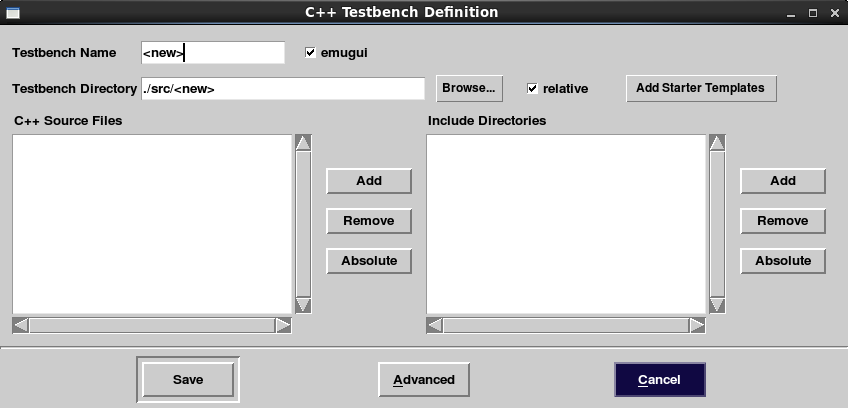


图 3.26 初始的C++ Testbench Definition对话框

1. Testbench Name：testbench的名字。emugui选项决定在运行Emulation时是否启动GUI界面，在不使用GUI界面时可以通过命令行批量运行Emulation。
2. Testbench Directory：testbench存放的目录，默认为工程目录下的src/testbench\_name文件夹。通过Browse按钮可以修改路径。relative选项决定是否采用相对路径。建议将testbench存放在默认目录下。
3. Add Starter Templates：生成testbench模板文件。
4. C++ Source Files：工程中添加的testbench文件，及其他操作选项。
5. Include Directories：C++ Testbench中的#include语句的搜索路径，及其他操作选项。
6. Add：添加文件或目录。
7. Remove：移除文件或目录。
8. ./ <=> /：在选中文件或目录的相对路径和决定路径之间进行切换。
9. Advanced：C++ testbench编译选项，一般情况下默认配置即可。
10. Save：保存配置。
11. Cancel：取消修改。

在Testbench Name选项框中输入testbench的名字，然后点击Add Starter Templates生成模板文件。如果需要添加其他的testbench源文件或include搜索路径，那么请通过相应按钮完成操作。

点击“Add Starter Templates”按钮后，Semu软件会根据pin-file在Testbench Directory文件夹下生成如下文件：

src/*testbench\_name*/TclTb.cpp

src/*testbench\_name*/gui\_dut.cpp

src/*testbench\_name*/usertb.cpp

usertb.cpp：是semu生成的testbench模板文件，用户可以在这个文件中添加自己的测试代码。

gui dut.tcl：实现在testbench中添加GUI界面。如果用户写了一个基于Tcl语言的可以嵌入到“Emulation Control Panel”或“Simulation Control Panel”界面中的GUI，那么需要编辑这个文件。

TclTb.cpp：这个文件不应该被修改。

同时，对于general类型工程，Semu会生成build/cpp/capi.h文件；对于sdt类型工程，Semu会生成transactors/transactors\_capi.h文件；这是针对于具体工程封装的C-API头文件，用户直接调用这个文件中的C-API函数即可。

点击“Add Starter Templates”按钮后，生成的CPP文件被添加到“C++ Source Files”中，生成的testbench目录被添加到“Include Directories”中。除了这些自动生成的文件和目录外，其他所有新增的testbench文件和目录都必须都过Add按钮添加到工程中。点击Save按钮保存testbench的配置，然后点击Edit按钮，再次打开“C++ Testbench Definition”对话框，才可以看到“C++ Source Files”和“Include Directories”中显示新添加的testbench的文件路径等信息，同时可以修改已存在的testbench的配置，但无法修改Testbench Name和Testbench Directory。

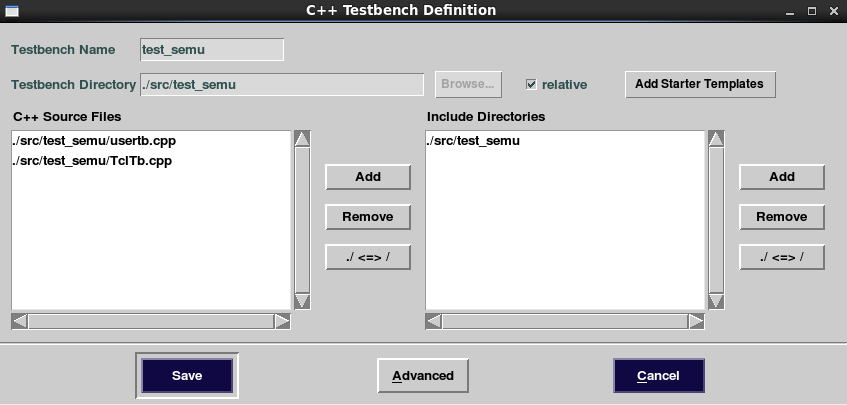


图 3.27 保存配置后的C++ Testbench Definition对话框

另外，点击“C++ Testbench Definition”对话框的Advanced按钮还可以完成C++ Testbench的高级选项等设置。

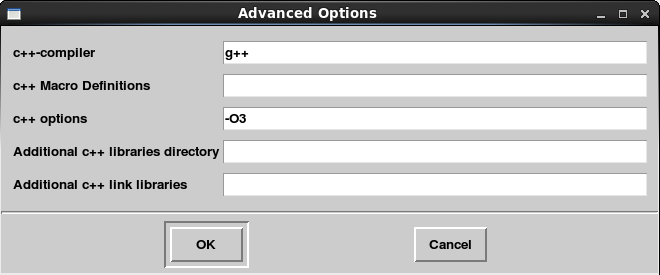


图 3.28 C++ Testbench的高级选项配置窗口

1. c++-compiler：c++的编译器。
2. c++ macro definitions：提供给c++工程的宏定义。
3. c++ options：c++编译器的参数，默认为-O3，对编译进行最高级别的优化。
4. Additional c++ link libraries directory：附加的c++链接库路径。（-L）
5. Additional c++ link libraries：调用附加的c++链接库的参数。（-l）

一但完成定义testbench的过程，就可以在“Configure Hardware”窗口的下拉框中选择已经定义的testbench。当执行“Build Full Emulation”、“Build Full Simulation”、“Build Emulation Testbench”、“Build Simulation Testbench”时，下拉框中选定的testbench就会被编译。如果工程已经被完整地编译过，那么可以仅执行“Build Emulation Testbench”或“Build Simulation Testbench”对testbench进行编译，缩短编译时间。

在“Testbench Configuration”窗口中的下拉框中选择一个testbench，点击Edit按钮，就可以对选定的testbench进行重新配置。但不能修改testbench name和testbench directory。当重新配置testbench并保存时，Emulation和Simulation的同名testbench的配置都被修改。

**注意**：所有testbench（C++ Testbench）的名字都应是独立的。

#### 3.5.2 定义Simulation Testbench

本小节介绍如何定义Simulation Testbench，Simulation Testbench的写法及用法请参考11Simulation Testbench的写法。

在Emulation运行环境中使用Simulation Testbench时，勾选“DUT Emulation”窗格中的Simulation Testbench以确定使用的testbench的类型。然后点击同一行的Add按钮，打开“Simulation Testbench Definition”对话框，添加新的Simulation Testbench。生成的testbench既可以提供给Emulation使用，也可以提供给Simulation使用。

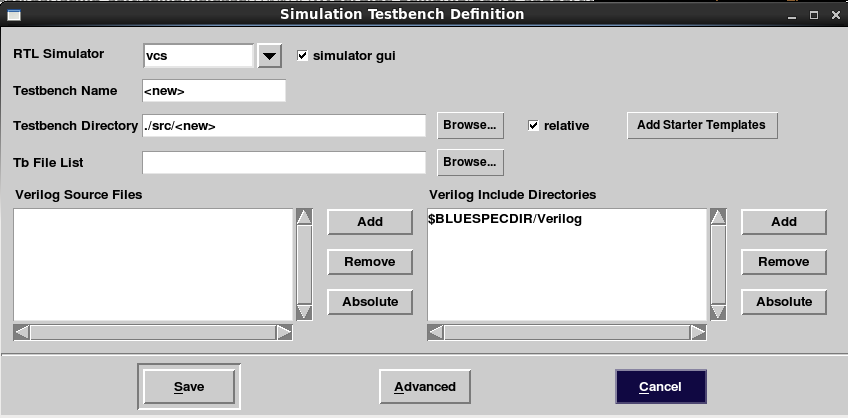


图 3.29 Simulation Testbench Definition对话框

1. RTL Simulator：Simulation Testbench的第三方仿真工具，目前仅支持VCS。
2. simulator gui：仿真时是否启动第三方仿真器的GUI界面。
3. Testbench Name：testbench的名字。
4. Testbench Directory：testbench存放的目录，默认为工程目录下的src/testbench\_name文件夹。通过Browse按钮可以修改路径。relative选项决定是否采用相对路径。建议将testbench存放在默认目录下。
5. Tb File List：Testbench的文件列表、搜索路径及其他编译选项等。相当于Verilog Source Files、Verilog Include Directories的功能组合，建议使用Tb File List指定Testbench的相关信息。
6. Verilog Source Files：工程中添加的testbench文件。
7. Verilog Include Directories：Simulation Testbench中的#include语句的搜索路径。
8. Add Start Templates：生成testbench的接口文件。
9. Add：添加文件或目录。
10. Remove：移除文件或目录。
11. Absolute：在选中文件或目录的相对路径和决定路径之间进行切换。
12. Advanced：C++ testbench编译选项，一般情况下默认配置即可。
13. Save：保存配置。
14. Cancel：取消修改。

在“Testbench Name”选项框中输入testbench的名字，然后点击“Add Starter Templates”生成模板文件。如果需要添加其他的testbench源文件或include搜索路径，那么请通过“Tb File List”相应按钮完成操作。

点击“Add Starter Templates”按钮后，Semu软件会根据pin-file在“Testbench Directory”文件夹下生成如下文件：

src/*testbench\_name*/verilog

src/*testbench\_name*/verilog/semu\_svtb\_dut\_interface.v

src/*testbench\_name*/vpi

src/*testbench\_name*/vpi/semu\_svtb\_init.cpp

src/*testbench\_name*/vpi/semu\_svtb\_advance\_clockB.cpp

src/*testbench\_name*/vpi/semu\_svtb\_vpi\_user.cpp

src/*testbench\_name*/vpi/semu\_svtb\_vpi.cpp

1. semu\_svtb\_dut\_interface.v：Testbench与Semu工程之间的接口，用户直接将Testbench中对DUT的例化修改为对本模块的例化即可完成移植工作。
2. semu\_svtb\_init.cpp：Semu工程的初始化函数文件。
3. semu\_svtb\_advance\_clockB.cpp：Semu工程的时钟推进函数文件。
4. semu\_svtb\_vpi\_user.cpp：Semu工程的VPI函数的注册文件。
5. semu\_svtb\_vpi.cpp：Semu工程的VPI函数的实现文件。

另外，点击“Simulation Testbench Definition”对话框的Advanced按钮还可以完成Simulation Testbench的高级选项等设置。

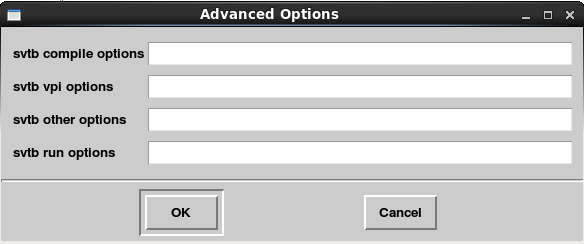


图 3.30 Simulation Testbench的高级选项配置窗口

下述的3种编译选项（svtb compile options、svtb vpi options、svtb other options）分别对应Testbench编译命令的不同位置。

1. svtb compile options：VCS工具的编译选项。
2. svtb vpi options：vpi编译选项。
3. svtb other options：动态连接库编译选项。
4. svtb run options：执行Simulation Testbench可执行程序时的选项参数。

**注意：**Simulation Testbench仅支持general类型工程中的port类型接口，仅支持VCS第三方仿真器。

## Build流程

Build操作将输入文件编译成一个Emulation运行环境或Simulation运行环境。

### Build菜单

Build菜单及功能介绍如下：

1. Build Full Emulation：编译Emulation环境
2. Build Emulation Testbench：编译Emulation环境的Testbench
3. Bit File Info：查看BIT文件的信息
4. Check Probe Size：在Probe回读波形方式下，检查待获取波形的信号的总比特数是否超过允许的上限值（10752-bit），仅在Probe回读波形方式下有效
5. Convert Probe VCD：在Probe回读波形方式下，对生成的编码波形文件进行解码，生成VCD波形文件或FSDB波形文件，仅在Probe回读波形方式下有效
6. Build Full Simulation：编译Simulation环境
7. Build Simmulation Testbench：编译Simmulation环境的Testbench
8. Stop：停止正在进行的编译过程



图 4.1 Build菜单及功能

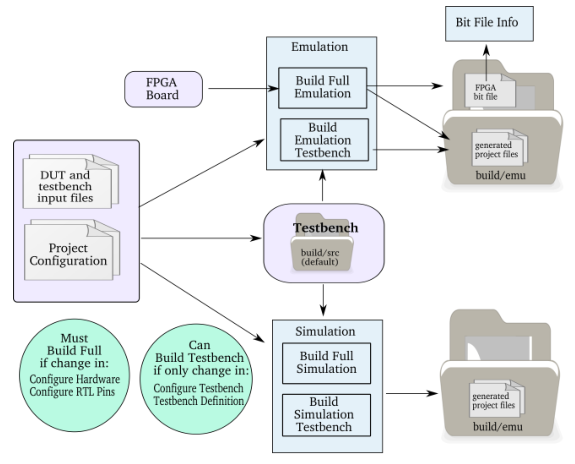


图 4.2 Build过程

### 4.2 编译

对于general类型工程，在编译工程时，Semu在工程根目录下生成一个build文件夹；对于sdt类型工程，在编译工程时，Semu在工程根目录下生成一个transactors文件夹。该文件夹下会生成3个文件夹：

1. cpp：对于general类型工程，包含testbench需要使用的transactors。该目录在生成C++ Testbench模板文件或编译工程时生成。C++ Testbench需要使用的C-API接口函数文件（capi.h）存放在此目录下。对于sdt类型工程，C-API接口函数文件的名字为transactors\_capi.h。
2. emu：包含了编译Emulation生成的所有文件。在“Build Full Emulation”时生成该目录。
3. sim：包含了编译Simulation生成的所有文件。在“Build Full Simulation”时生成该目录。

默认的存放Emulation和Simulation的testbench的文件夹为工程根目录下的src文件夹。用户可以随意设定src文件夹的位置，但不建议放在build或transactors文件夹下。每次编译工程时，build/emu或build/sim或transactors/emu或transactors/sim下的子文件夹会被删除并重新创建，所以存放在这些目录下的testbench在重新编译工程时可能会丢失。

#### 4.2.1 编译输入

编译过程的输入在Project菜单的选项中定义。硬件信息、DUT、仿真器选项在“Configure Hardware”界面中定义。pin-file在“Pin Interface Definition”界面中定义。Testbench类型和文件在“Configure Testbench”界面中定义。Semu将这些配置选项和文件作为输入生成完整的Emulation或Simulation环境。

#### 4.2.2 编译过程

对于Emulation编译，如果是第一次编译Emulation，那么点击“Build->Build Full Emulation”编译工程，编译完成后会生成完整的Emulation环境。如果已经完整地编译过工程，那么可以点击“Build->Build Full Emulation”重新编译工程，或点击“Build->Build Emulation Testbench”仅编译Emulation的Testbench。Emulation编译中耗时最长的步骤是生成BIT文件。如果仅需要编译Emulation的Testbench，那么运行“Build Emulation Testbench”即可，以节省编译时间。“Build->Bit File Info”选项显示Emulation板子信息和最新生成的BIT文件的生成时间信息。

对于Simulation编译，如果是第一次编译Simulation，那么点击“Build->Build Full Simulation”编译工程，编译完成后会生成完整的Simulation环境。如果已经完整地编译过工程，那么可以点击“Build->Build Full Simulation”重新编译工程，或点击“Build->Build Simulation Testbench”仅编译Simulation的Testbench以缩短编译时间。

“Build Emulation Testbench”和“Build Simulation Testbench”仅对C++ Testbench和Simulation Testbench有效，对Manual Testbench无效。如果需要编译Manual Testbench，那么需要执行“Build->Build Full Emulation”或“Build->Build Full Simulation”。

#### 4.2.3 编译输出

对于general类型工程， Emulation编译时，Semu会将build/emu文件夹重命名为build/oldemu，保存前一次Emulation编译的结果，然后重新创建emu文件夹及其子文件夹（scripts、rtl、rtl\_edited、obj、info、bsv、xilinx），完成编译过程。对于Simulation编译，Semu首先删除build/sim目录下的指定文件夹（scripts、rtl、rtl\_edited、obj、info、bsv），然后重新创建这些文件夹，完成编译过程。

对于sdt类型工程，对应的编译文件夹为工程根目录下的transactors文件夹。

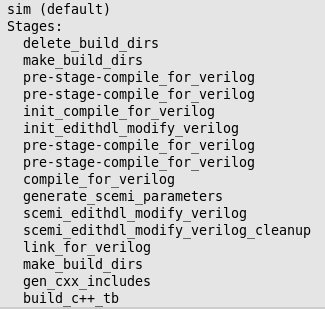
### 4.4 命令行

Semu软件的执行程序主体为安装目录下的lib/bin/build脚本。在终端中输入“build --doc”命令，可以得到build脚本使用方法。本小节仅对build流程做简单说明，详细信息请通过“build --doc”查看。

build脚本将编译过程组织为多个编译目标（target），并进一步细分为编译阶段（stage），对编译目标和编译阶段的组织和调用方式类似于Makefile。每个编译目标可以包含其他的一个或多个子目标，每个目标还可以包含若干个编译阶段，这些信息包含在bld文件中（包括：project.bld、testbench.bld、sim\_tb或用户自定义的其他bld文件），编译过程受bld文件的控制。用户可以使用“-p”选项指定一个bld文件，如果不指定，那么默认对project.bld文件进行操作。“build --doc”命令可以查看bld文件的格式和每项参数的含义。

project.cfg文件是工程的硬件配置信息文件，执行“build –i *path*/project.cfg”命令可以生成replay\_edits\_init.script、replay\_edits\_scemi.script、project.blt、project.bld文件。

使用“build –l [-p *xxx*.bld]”可以查看指定bld文件中的所有编译目标以及各个编译目标包含的编译阶段，不加“-p”选项时，默认对project.bld文件进行操作。如下：

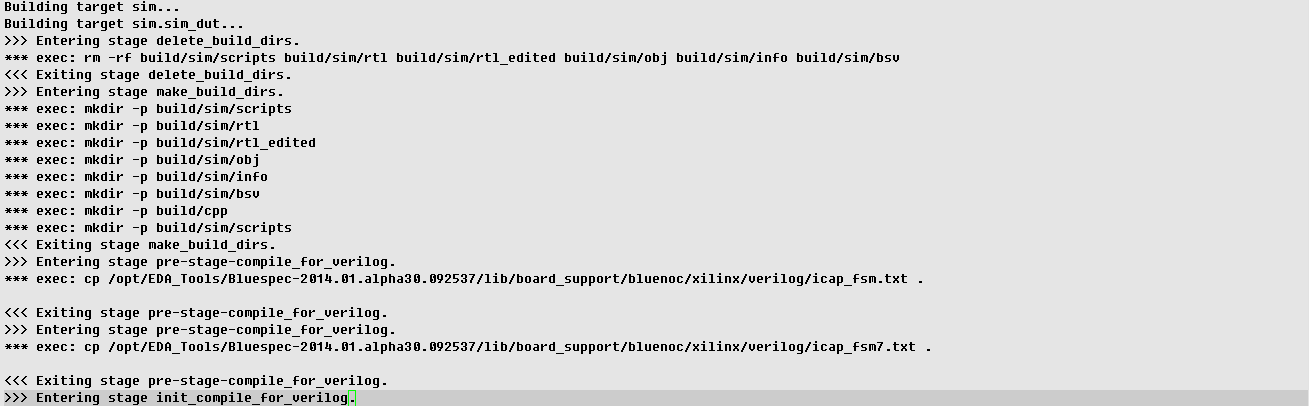


其中默认的编译目标为sim，即“Build->Build Full Simulation”，但仅编译Manual Testbench。

一般来说，为了生成完整的编译目标，编译目标中的所有编译阶段应全部按顺序执行。但是用户可以使用“build –f *start\_stage\_name* –t *end\_stage\_name*”仅执行从*start\_stage\_name*到end\_*stage\_name*的编译阶段。

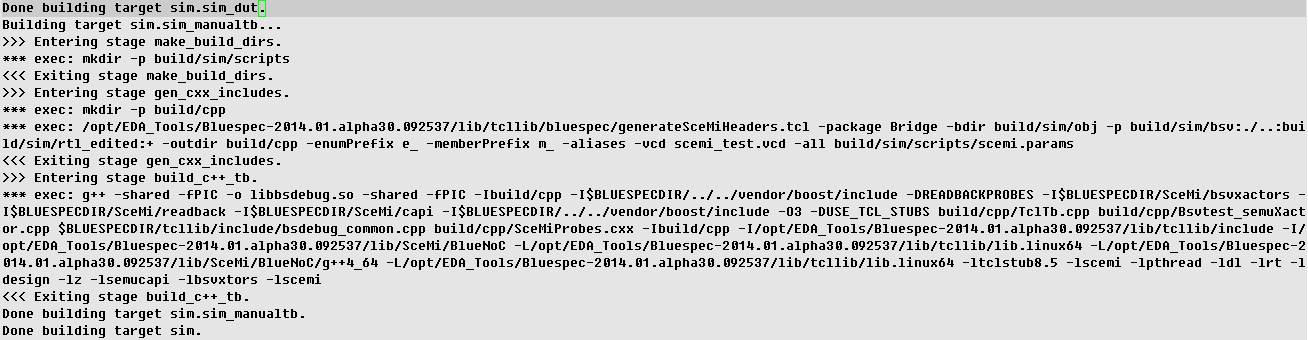
例如：

1. “build –f link\_for\_verilog –t build\_c++\_tb”：依次完成link\_for\_verilog、make\_build\_dirs、gen\_cxx\_includes、build\_c++\_tb这4个编译阶段。
2. “build sim –f compile\_for\_verilog”：从compile\_for\_verilog开始执行，直到build\_c++\_tb结束。
3. “build sim –t compile\_for\_verilog”：从delete\_build\_dirs开始执行，直到compile\_for\_verilog结束。
4. “build --dry-run [target\_name]”命令会显示指定编译目标的所有编译阶段的名字和执行的命令。如果target\_name缺省，那么默认为sim。如下：
5. 执行“build –dry-run sim”命令（Build->Build Full Simulation）的输出如下：

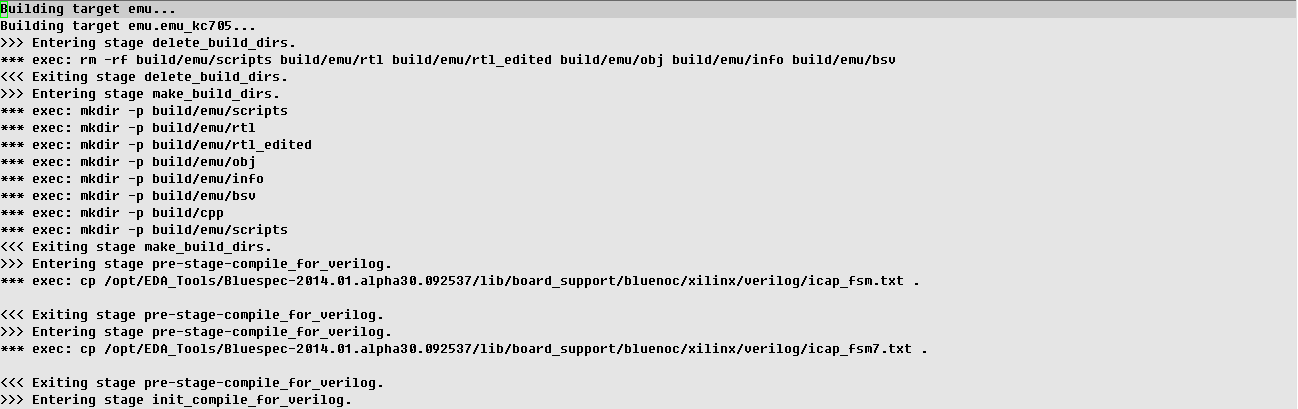






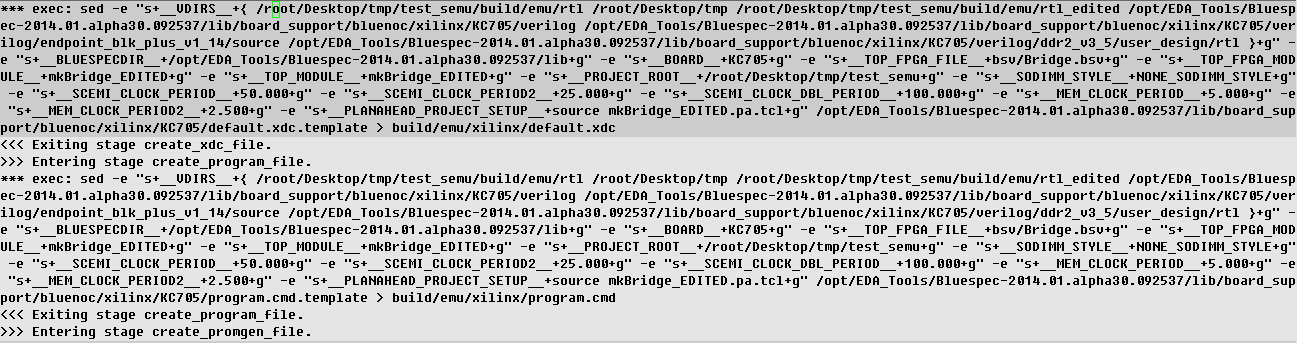


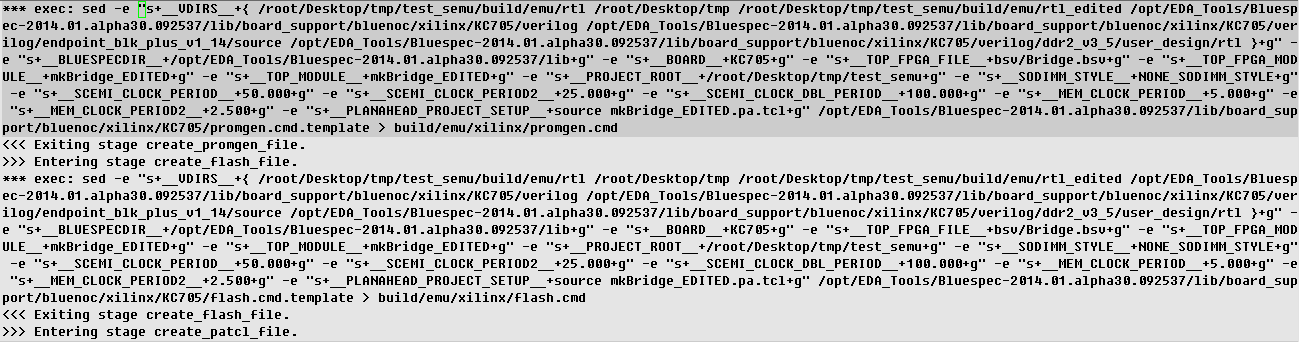
1. 执行“build –dry-run emu”命令（Build->Build Full Emulation）的输出如下：

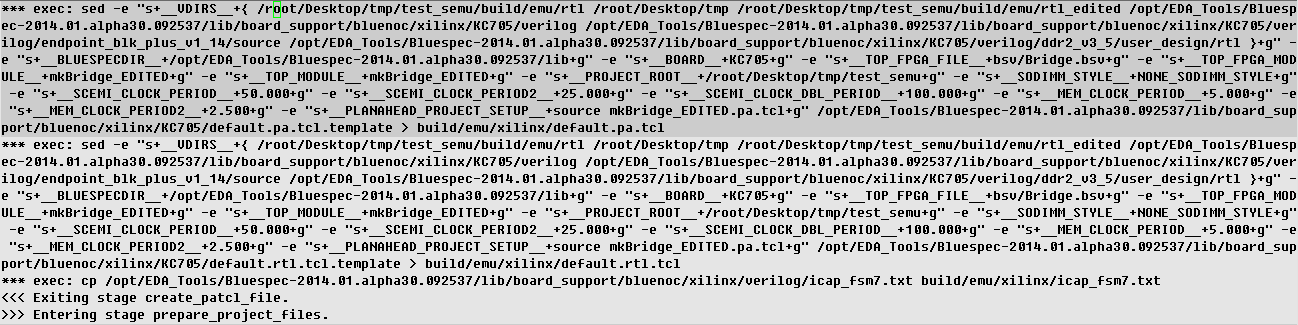


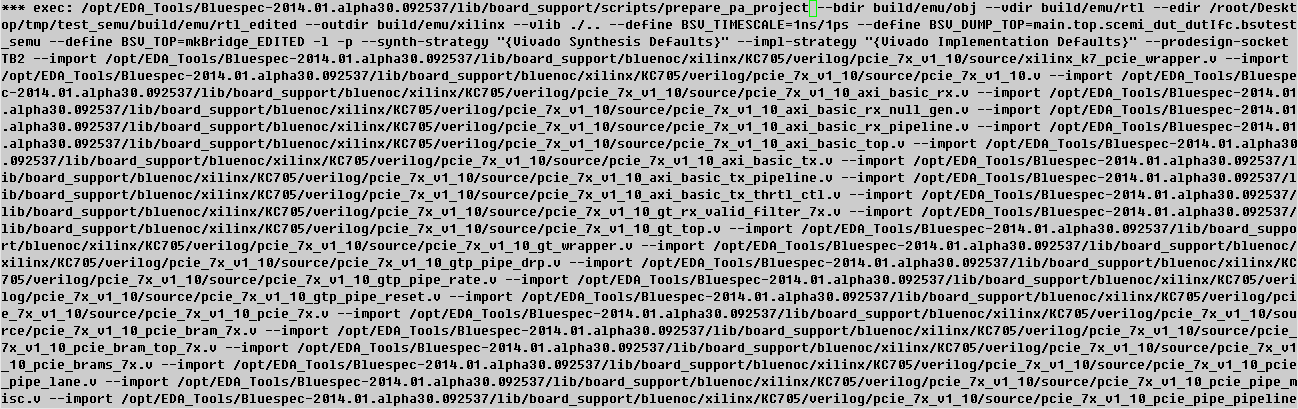


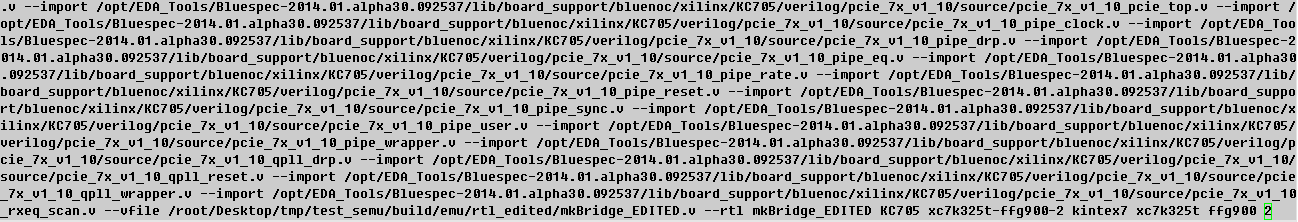




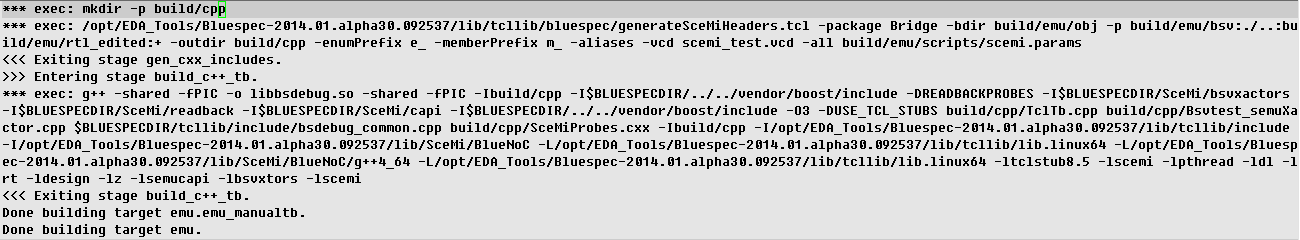












编译过程中，semu会将各个编译阶段的输出信息输出到工程目录下的logs文件夹下的<target\_name>\_<stage\_name>.log文件中。如果使用了“-v”或“-- verbose”参数，输出信息也会在Semu软件的窗口中显示。

## Board

### Board菜单

Board菜单包括设置、查询板子状态等功能。

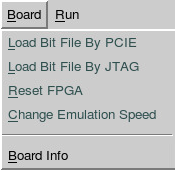


图 5.1 Board菜单

1. Load Bit File By PCIE：通过PCIE配置FPGA，速度很快；
2. Load Bit File By JTAG：通过JTAG配置FPGA，速度较慢；
3. Reset FPGA：复位FPGA中的逻辑；
4. Change Emulation Speed：动态修改Emulation时DUT的时钟频率；
5. Board Info：查看Semu硬件板子信息；

### 5.2 Load Bit File By PCIE

点击“Board->Load Bit File By PCIE”后，Semu软件会通过PCIE和SelectMap将Vivado生成的bin文件配置到FPGA中，配置速度很快，建议使用此种配置方式。

Semu向FPGA板上加载BIT文件的要求：已安装驱动（如果/dev/路径下是否可以找到bluenoc\_1则表示已成功安装驱动，否则需要重新安装驱动），PCIE可以正常通信。

加载BIT文件成功后，会弹出如下窗口，提示用户可以进行Emulation了。

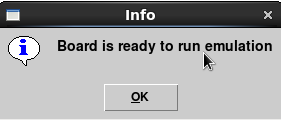


图 5.2 加载BIT文件成功时的提示

### 5.3 Load Bit File By JTAG

点击“Board->Load Bit File By JTAG”后，Semu软件会根据FPGA板子型号调用Impact工具加载生成的BIT文件。

Semu向FPGA板上加载BIT文件的要求如下：

1. 已安装Xilinx的Impact工具（此工具在包含在Xlinx软件中）；
2. FPGA板允许使用Impact工具通过USB加载BIT文件；
3. USB线连接正常；
4. 已安装驱动（如果/dev/路径下是否可以找到bluenoc\_1则表示已成功安装驱动，否则需要重新安装驱动）。

点击“Board->Load Bit File By JTAG”后，可以在Semu的Message窗口看到Semu执行了如下命令：“impact -batch mkBridge\_EDITED\_jtag.cmd”，加载的BIT文件为工程目录下的“build/emu/xilinx/mkBridge\_EDITED.bit”文件。

在加载BIT文件的过程中，Semu首先对当前的BIT文件和FPGA板子进行匹配验证。如果验证通过，BIT文件会被加载到FPGA中；如果验证失败，Semu会提示警告，但用户可以继续加载BIT文件，或取消加载动作。

如果工程配置更改而没有重新编译工程，那么在“Configure Hardware”对话框中指定的板子型号可能与当前的BIT文件不匹配。在这种情况下，Semu仍然会加载当前的BIT文件。

如果Semu不能找到连接的板子，那么可能是因为没有安装驱动（bootstrap），这种情况下，Semu仍然会尝试向FPGA加载BIT文件。

如果BIT文件加载失败，那么Semu会提示错误信息，并且Semu的Message窗口会提供更加详细的报告，如下图所示。



图 5.3 BIT文件加载失败的错误提示

加载BIT文件成功后，会弹出如图5.3的窗口，提示用户可以进行Emulation了。

### Reset FPGA

在完成加载BIT文件或改变Emulation受控时钟的频率后，Semu会自动复位FPGA。“Board-> Reset FPGA”允许用户在任一时刻复位FPGA。

### 5.5 Change Emulation Speed

大多数对“Configure Hardware …”对话框中工程配置的修改都需要重新编译工程生成BIT文件，这个过程会耗费较长的时间。“Board->Change Emulation Speed”允许用户动态改变Emulation时的受控时钟的频率，而不需要重新编译工程生成新的BIT文件。

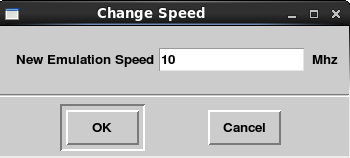


图 5.4 Change Speed对话框

受控时钟的频率必须设置在5~500MHz之间。为了动态改变Emulation中受控时钟的频率，用户需要点击“Board->Change Emulation Speed”，然后会弹出“Change Speed”对话框。“Change Speed”对话框中的数值为当前状态下的Emulation中受控时钟的频率，修改这个数值，并点击OK，完成修改。同时FPGA会被复位。

### Board Info

“Board->Board Info”会显示当前板子的信息。如下图所示：

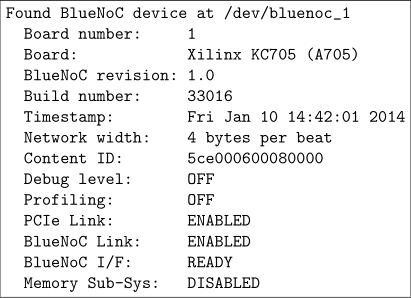


图 5.5 Board Info输出信息

## Run

当工程编译完成后，用户就可以通过Run菜单运行Emulation或Simulation环境对DUT进行测试了。



图 6.1 Run菜单及功能

当点击“Run->Start Emulation”或“Run->Start Simulation”时，首先显示的窗口是控制台界面。Emulation运行环境显示“Testbench Console”和“Emulation Control Panel”界面，而Simulation运行环境显示“Testbench Console”、“Simulation Console”、“Simulation Control Panel”界面。

## Emulation流程

一旦Emulation运行环境编译成功，用户就可以将生成的BIT文件加载到FPGA中开始验证、调试。Semu支持2种波形获取方式：Readback、probe，和3种Testbench。对于这2种波形获取方式和3种Testbench，运行Emulation的方式存在区别，下面分别进行介绍。

### 波形获取方式对Emulation的影响

1. Readback波形获取方式：
   * 支持获取设计中的所有寄存器类型信号的波形，生成VCD文件
   * 支持硬件断点（仅在使用C/C++ Testbench时支持）
   * 支持动态修改待抓取的信号，而不需要重新编译工程
   * 速度较慢
2. Probe波形获取方式：
   * 支持获取设计中的所有类型信号的波形，并生成VCD波形文件或FSDB波形文件
   * 支持最多一次获取10752-bit信号的波形
   * 支持硬件断点
   * 抓取不同的信号，需要重新编译工程
   * 待测设计输入的源文件必须是Verilog RTL源码
   * 速度较快

### Testbench类型对Emulation的影响

1. Manual Testbench
   * 功能简单，不建议使用，暂不介绍。
2. C++ Testbench
   * 支持所有类型工程中的所有类型接口；
   * 支持Emulation和Simulation；
   * Emulation时不需要第三方仿真器，Simulation时需要第三方仿真器（VCS、Modelsim）；
   * 支持Semu的“Emulation Control Panel”界面或“Simulation Control Panel” 界面及对应的所有操作。
3. Simulation Testbench
   * Simulation Testbench仅支持general类型工程中的port类型接口；
   * Simulation Testbench仅支持Emulation，不支持Simulation；
   * Simulation Testbench仅支持VCS仿真器；
   * Simulation Testbench仅支持VCS仿真器的DVE界面，不支持Semu原有的“Emulation Control Panel”界面或“Simulation Control Panel” 界面上的一部分操作已经转化为API函数供用户使用，这些API函数也可以在C++ Testbench中使用。

### Readback波形获取方式下的C++ Testbench的Emulation流程

在Readback获取波形方式下的Emulation过程中，用户可以动态选择需要获取波形的信号。

启动Emulation进行验证、调试的步骤如下：

1. 定义信号集合，或者加载一个已经存在的信号集合。
   * 从设计层次中选择的DUT信号。这些信号可以是定义断点需要使用的信号，或期望获取波形的信号。这些信号形成了一个信号集合。
   * 定义断点。用户可以对信号设置断点条件（包括bit-mask和value），将信号用作断点功能。一个信号集合中可以包含多个信号，每个信号又可以包含不同的断点条件。
   * 保存信号集合，以便于后续使用。
2. 使能信号，并将信号添加到Readback。
3. 选择信号添加到波形。
4. 使能断点。多个断点定义可以通过AND/OR组织成复合断点。同一时刻只能使能一个断点。
5. 使用C++ Testbench或Manual Testbench运行Emulation。“Emulation Control Panel”界面允许对时钟进行控制。
6. 根据需要修改选择的信号，并使能断点。

#### FPGA准备

Board菜单的功能是设置、查询板子的状态。

首先，用户必须加载BIT文件到FPGA中（Board->Load Bit File）。BIT文件在编译Emulation环境时已经生成。

在加载BIT文件成功后，FPGA会被自动复位。“Board->Reset FPGA”允许用户在任一时刻复位FPGA。

用户可以通过“Board->Change Emulation Speed”在运行Emulation之前动态修改Emulation的时钟频率。默认的Emulation时钟频率是“Configure Hardware”对话框中指定的值。

#### Emulation Control Panel界面

点击“Run->Start Emulation”可以启动Emulation，并打开“Testbench Console”窗口和“Emulation Control Panel”窗口。当Emulation结束时，关闭“Testbench Console”窗口，同时“Emulation Control Panel”窗口也会被关闭。

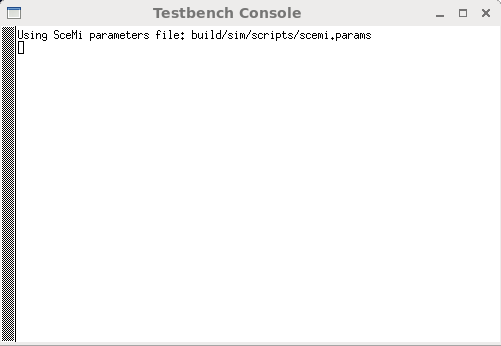


图 7.1 Testbench Console窗口

“Emulation Control Panel”界面被划分为5个窗格。每个部分都可以通过各部分之间的分割器上的手柄调整大小。

1. 时钟和Readback控制：提供实现时钟控制和Readback控制的按钮，并显示Emulation的速度和状态；
2. DUT控制：该窗格仅在使用Manual Testbench时才会显示；
3. Message：使用Manual Testbench时，显示DUT的输出信息；
4. 设计层次（debug only）：显示设计层次，从设计层次中可以选择与断点的信号和获取波形的信号；
5. 信号集合和断点（debug only）：列出从设计层次中选择的与断点的信号和获取波形的信号。

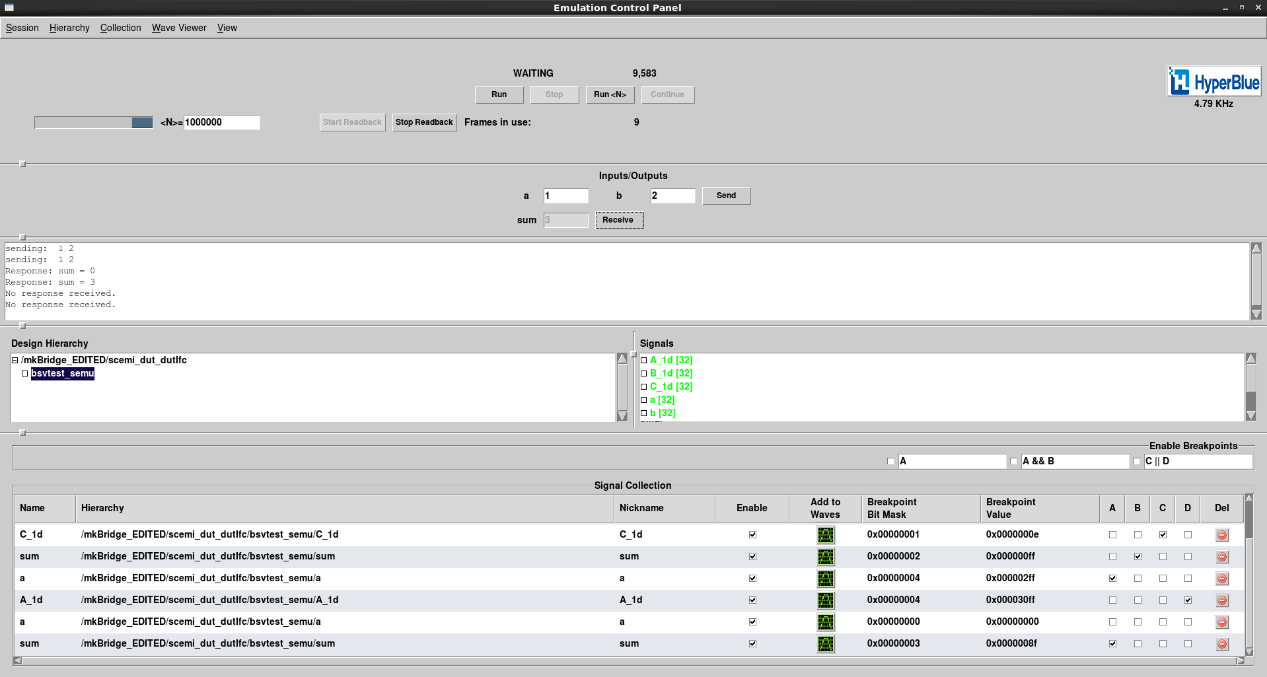


图 7.2 使用Manual Testbench的Emulation Control Panel界面

在View菜单中可以选择是否显示设计层次窗口、信号集合和断点窗口。

#### View菜单

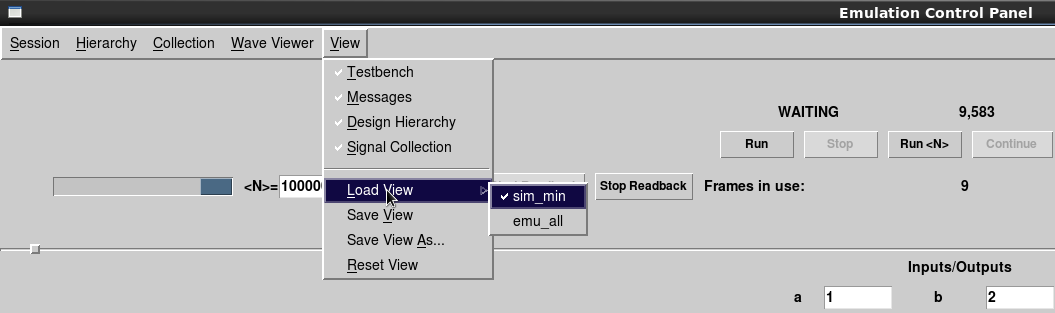


图 7.3 Emulation Control Panel界面中的View菜单选项

Semu允许用户修改并保存“Emulation Control Panel”和“Simulation Control Panel”界面。窗口的大小、位置、显示的窗格、每个窗格的大小都可以被修改。一但用户得到了一个最佳的窗口布局配置，可以保存窗口的布局特性以便于后续使用。

View菜单的选项控制“Emulation Control Panel”和“Simulation Control Panel”界面的显示情况，包括是否显示某个窗格。时钟控制窗格总是处于显示状态。Testbench窗格只有在testbench包含GUI时才会显示。“Design Hierarchy”窗格和“Signal Collection”窗格仅在debug模式下（仅在Emulation时有效）显示。

用户可以选择隐藏窗格，也可以调整窗格大小。窗口布局配置信息可以被保存和加载，以便于用户后续的使用。

如果需要显示某个窗格，那么可以在View菜单中的对应窗格的名字上打钩。如果不需要显示某个窗格，那么取消在View菜单中的对应窗格的名字的勾选。如果窗格的名字是灰色的，那么在当前运行环境下，灰色名字的窗格是不能显示的。例如，“Design Hierarchy”窗格和“Signal Collection”窗格仅在Emulation的debug模式下可以显示，在Simulation环境下无法显示。

当使用C++ Testbench时，由于模板文件中不包含testbench的GUI接口，所以默认Testbench窗格是不显示的。如果用户写了一个定制的GUI，那么可以选择显示Testbench窗格。

用户可以保存很多个窗口布局配置。使用“View->Save View”或“View->Save View as …”将窗口布局配置保存在layout文件（proj.lyt）中。也可以使用“View->Load View”加载指定的窗口布局配置。如果加载的窗口布局配置中含有无效的窗格，那么semu会输出提示信息并保持原有的窗口布局配置不变。

保存、加载窗口布局配置的选项在View菜单的下半部分。

1. Load View：加载保存的窗口布局配置，当点击这个选项时，选项的右边会列出已经保存的窗口布局配置的名字。从列表中选择一个即可完成加载；
2. Save View：保存当前的窗口布局配置，如果当前的窗口布局配置没有名字，那么semu会提示用户输入名字；
3. Save View As …：保存当前的窗口布局配置，并重新命名；semu会提示用户输入名字。
4. Reset View：加载系统默认的窗口布局配置。

每次打开“Emulation Control Panel”和“Simulation Control Panel”界面时，上一次保存的并显示的窗口布局配置被加载。如果没有保存窗口布局配置，那么会加载系统默认的窗口布局配置。用户可以点击“View->Reset”恢复系统默认的窗口布局配置。

#### 时钟和Readback控制

最上面的窗格包含了时钟控制和Readback控制功能。窗格中包含如下控制按钮：

1. Run：启动受控的时钟；
2. Stop：停止受控的时钟；
3. Run<N>：推进N个时钟周期；
4. Continue：再次启动受控的时钟，而不复位时钟周期数（N），只有当点击Run<N>按钮，而且时钟停止之前再次点击Stop按钮时，Continue按钮才会处于激活状态。

系统的运行频率在窗格的右上角显示。系统的运行状态和运行时钟周期数在按钮的上方显示。

“Start Readback”按钮和“Stop Readback”按钮提供对Readback功能的控制。在Readback按钮右边显示的是被选择的frame的个数。当使能Readback功能时（选中Start Readback），Semu每个时钟周期都会读取这些frame。当信号集合中的信号处于使能状态时，这些信号的值被写入VCD文件（工程根目录下的dump1.vcd）。

#### DUT控制

提供Manual Testbench的DUT的输入和控制。该窗格仅在使用Manual Testbench时才会显示。

#### Message

Manual Testbench的输出信息显示在Message窗格中，其他的输出信息显示在“Testbench Console”窗口中。

#### 设计层次

设计层次窗格中显示DUT的设计层次。从设计层次中可以选择与断点的信号和获取波形的信号添加到信号集合窗格中。左边的一半窗格显示DUT的设计结构，右边的一半窗格显示左边窗格中被选中的设计层次中的信号。并不是所有的信号都会显示在右边窗格中，只有readback功能支持的状态信号（state signals）才会显示。

从设计层次窗格的左半边窗格中点击右键可以得到如下菜单，功能如下：

1. Expand：展开指定层次
2. Collapse：折叠指定层次
3. Promote To Top：将指定的层次设置为顶层
4. Collapse All：折叠所有层次
5. Add and Enable All：将指定层次内的所有信号添加到信号集合中

#### 选择信号

选择信号有两种方式：从界面选择信号、使用C-API函数选择信号。区别在于：

1. 从界面选择信号：从界面选择每个信号进行添加；支持断点；
2. 使用C-API函数选择信号：从工程根目录下的“signals\_for\_wave.lst”文件中获取需要添加的信号，自动完成添加过程，C-API函数名为“do\_add\_and\_enable\_signals”；不支持断点；

下面仅介绍从界面选择信号的方式。

从设计层次窗格的左半边窗格中点击右键得到的“Add and Enable All”可以将指定层次内的所有信号添加到信号集合中。

从设计层次窗格的右半边窗格中选择信号并点击右键，会出现一个功能栏，功能栏中选项的如下：

1. Add and Enable：将选中的信号添加到信号集合中，并使能信号；
2. Add：将选中的信号添加到信号集合中；
3. Enable：使能选中信号的所有实例，但不添加一个新的实例到列表中；
4. Disable：禁止选中信号的所有实例。

如果希望针对信号应用readback功能，那么对应的信号必须是Enabled的。

#### 信号集合和断点

窗口的底部显示被添加到信号集合中的所有信号。当完成添加信号后，设计层次窗口就可以被隐藏起来，通过View菜单可以实现这个功能。

信号集合可以被重复使用。Collection菜单提供了保存信号集合、加载已经存在的信号集合等功能。一个工程可以拥有多个信号集合。用户可以在任何时刻修改一个信号集合，包括添加信号、删除信号等，修改后的信号集合可以被保存为另外一个名字的信号集合，以便后续使用。

信号集合列表中的每行均包含了一个信号和对应的断点等信息（bit-mask、value）。对于一个信号instance只能设置一个断点信息：设置bit-mask、value，并勾选A、B、C、D中的一列。如果期望对同一个信号设置多个断点信息，那么需要新增同一个信号的信号instance，并在新增的信号instance行设置其他的断点信息。

#### 断点

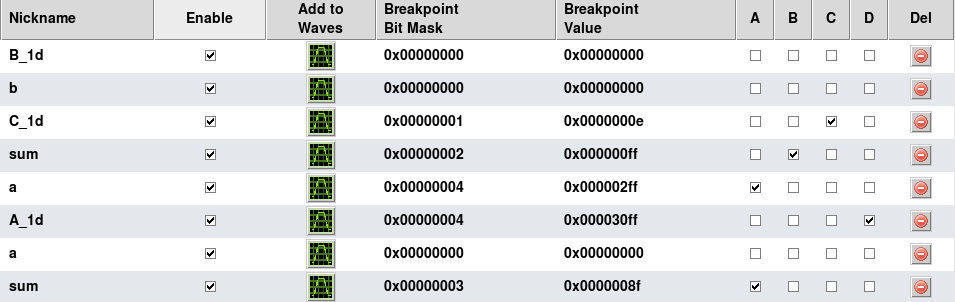


图 7.4 信号集合列表中的断点

信号集合窗格中的4列（A、B、C、D）定义了工程中的断点信息。一个断点包含一个或多个信号实例，一个信号实例包含了一个信号、一个断点bit-mask、一个断点value。信号实例中的条件必须全部为真，Semu才会认为断点是真（被触发）。

1. 断点bit-mask：决定断点value中的哪些位被用来与FPGA中的信号值相比较，当某位为1时，则不对断点value的对应位做比较；
2. 断点value：被比较的信号值，当FPGA中的信号值与断点value相等时（假设不考虑断点bit-mask）才会触发断点。

勾选对应信号行的A、B、C、D这4列中的方框即可定义断点，上图中的A列包含了两个信号实例：a和sum。

#### 使能断点

“Enable Breakpoints”部分允许一次定义3个断点，但任一时刻只能使能一个断点，断点条件表达式填写在“Enable Breakpoints”部分。一个断点条件表达式可以是断点列（A、B、C、D）的名字，比如图中的“Enable Breakpoints”部分的第1个断点定义。断点表达式还可以是断点列（A、B、C、D）的名字的逻辑运算，比如“Enable Breakpoints”部分中的第2、3个断点定义。当使能断点条件表达式时，对应的断点列（A、B、C、D）和列中被勾选的信号以蓝色高亮显示。

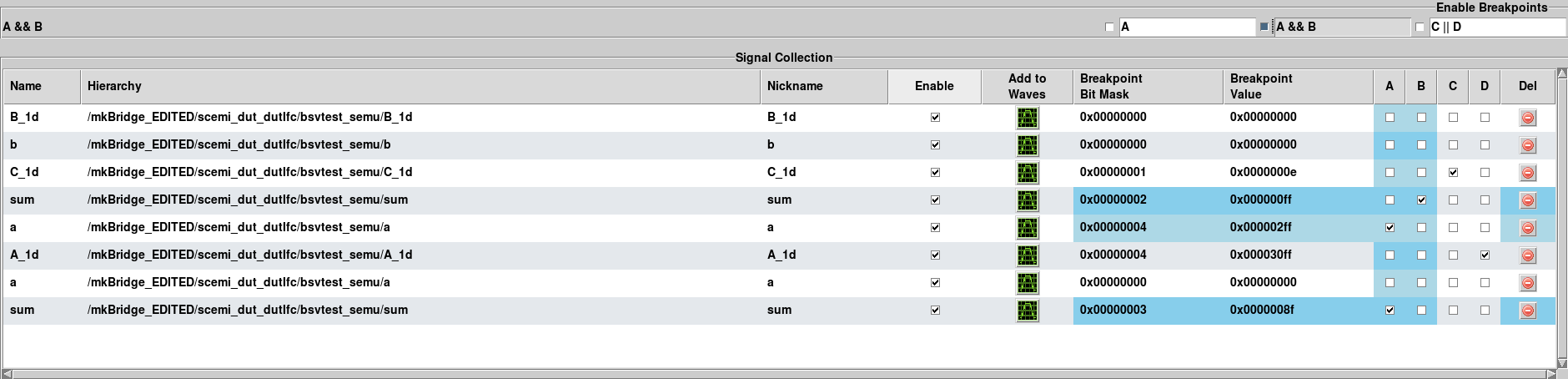


图 7.5 使能断点

#### 运行Emulation，查看波形

步骤如下：

1. 加载BIT文件到FPGA中；
2. 加载一个信号集合，或从设计层次窗口中选择信号添加到信号集合中；
3. 使能对信号的reaback功能，并将信号添加到波形中；
4. 启动readback（Start Readback）；
5. 使能或禁止断点；
6. 启动时钟运行，点击Run按钮或Run<N>按钮。

“Wave Viewer”菜单管理与波形查看软件之间的接口，可以实现如下操作：

1. Start Wave Viewer：启动波形查看软件gtkwave；
2. Attach：选择一个已经打开的波形查看器，并附在该波形查看器上；
3. Load Dump File：加载VCD文件；
4. Send Enabled Signals：将信号集合中处于使能状态的信号添加到gtkwave；
5. Reload VCD：重新加载已经加载了的VCD文件；
6. Allow XServer Connections：允许波形查看软件的xhost连接。

用户也可以在终端中启动gtkwave或verdi，打开已经生成的VCD文件查看波形。或通过verdi提供的vcd2fsdb命令将VCD文件转化成FSDB格式的波形文件，然后使用verdi查看。

### Readback波形获取方式下的Simulation Testbench的Emulation流程

在该方式下，点击“Run->Start Emulation”后就会启动Emulation。如果设置对应Testbench的“simulator gui”为true，那么会启动第三方仿真工具的界面（目前仅支持VCS的DVE界面）；如果设置为false，那么不会启动界面，而是直接开始跑仿真。

该方式与“Readback波形获取方式下的C++ Testbench的Emulation流程”的区别在于：

1. 仅支持使用C-API函数选择信号获取波形：从工程根目录下的“signals\_for\_wave.lst”文件中获取需要添加的信号，自动完成添加过程，C-API函数名为“do\_add\_and\_enable\_signals”；
2. 不支持断点；
3. 仅支持VCS的DVE界面。

### Probe波形获取方式下的C++ Testbench的Emulation流程

该方式与“Readback波形获取方式下的C++ Testbench的Emulation流程”的区别在于：

1. 通过在“Configure Hardware->Advance Options”中的“Probe CMD File”中指定的文件描述待获取波形的信号；
2. 支持使用C-API函数启动Probe波形功能：需要将工程根目录下的“design\_signal\_xrf.lst”中的任一DUT路径下的信号添加到工程根目录下的“signals\_for\_wave.lst”文件中，然后在testbench中（usertb.cpp中的do\_test函数的最开始处）调用do\_add\_and\_enable\_signals函数即可。

当Emulation结束后，执行“Build->Convert Probe VCD”可以对获得的编码波形文件解码生成VCD波形文件或FSDB波形文件。

### Probe波形获取方式下的Simulation Testbench的Emulation流程

在该方式下，点击“Run->Start Emulation”后就会启动Emulation。如果设置对应Testbench的“simulator gui”为true，那么会启动第三方仿真工具的界面（目前仅支持VCS的DVE界面）；如果设置为false，那么不会启动界面，而是直接开始跑仿真。

该方式与“Readback波形获取方式下的C++ Testbench的Emulation流程”的区别在于：

1. 通过在“Configure Hardware->Advance Options”中的“Probe CMD File”中指定的文件描述待获取波形的信号；
2. 支持使用C-API函数启动Probe波形功能：需要将工程根目录下的“design\_signal\_xrf.lst”中的任一DUT路径下的信号添加到工程根目录下的“signals\_for\_wave.lst”文件中，然后在./src/testbench\_name/vpi/semu\_svtb\_init.cpp中的semu\_svtb\_init\_compiletf函数中调用do\_add\_and\_enable\_signals函数即可；
3. 仅支持VCS的DVE界面；
4. 支持断点。

当Emulation结束后，执行“Build->Convert Probe VCD”可以对获得的编码波形文件解码生成VCD波形文件或FSDB波形文件。

### 命令行批量运行Emulation

上述的介绍的方法无法批量运行Emulation，为了批量运行Emulation需要做如下工作：

1. 在生成Testench时，对于C++ Testbench不勾选emugui，对于Simulation Testbench不勾选simulator gui；
2. 界面编译或命令行编译；
3. 界面编译和命令行编译的过程请参见[4 build流程](#_Build流程)；
4. 命令行批量运行。

用户在批量运行Emulation时可以采用方式如下：

1. 生成多个testbench并完成用例编写，编译，然后批量运行Emulation；
2. 生辰1个testbench并完成用例编写，编译，然后通过传参批量运行Emulation；
3. 前两种相结合。

批量运行的脚本需要用户结合Semu的命令行自行编写。

对于C++ Testbench来说，在命令行批量运行模式中，usertb.cpp文件中新增了main函数，作为可执行程序的入口；用户依然在do\_test函数中写Testbench，do\_test函数增加了两个参数：int argNum， char\*\* argValues，它们等效于main函数传入的参数，可以按照使用main函数参数的方法使用。编译生成的可执行程序为build/emu/scripts/*testbench\_directory\_name*.x或transactors/emu/scripts/*testbench\_directory\_name*.x，用户编写Shell脚本调用相关命令和生成的可执行程序（可传参），即可实现命令行批量运行Emulation。

对于Simulation Testbench，编译生成的可执行程序为build/emu/scripts/*testbench\_directory\_name*或transactors/emu/scripts/*testbench\_directory\_name*，可以像使用VCS一样通过编写Shell脚本调用相关命令和生成的可执行程序，实现命令行批量运行Emulation。

## Simulation流程

Semu支持3种Testbench：Manual Testbench、C++ Testbench、Simulation Testench。其中Semu不支持对Simulation Testench进行Simulation，仅支持Manual Testbench、C++ Testbench进行Simulation。下面介绍Simulation流程。

点击“Run->Start Simulation”即可打开“Simulation Control Panel”界面，启动Simulation，如下图所示。

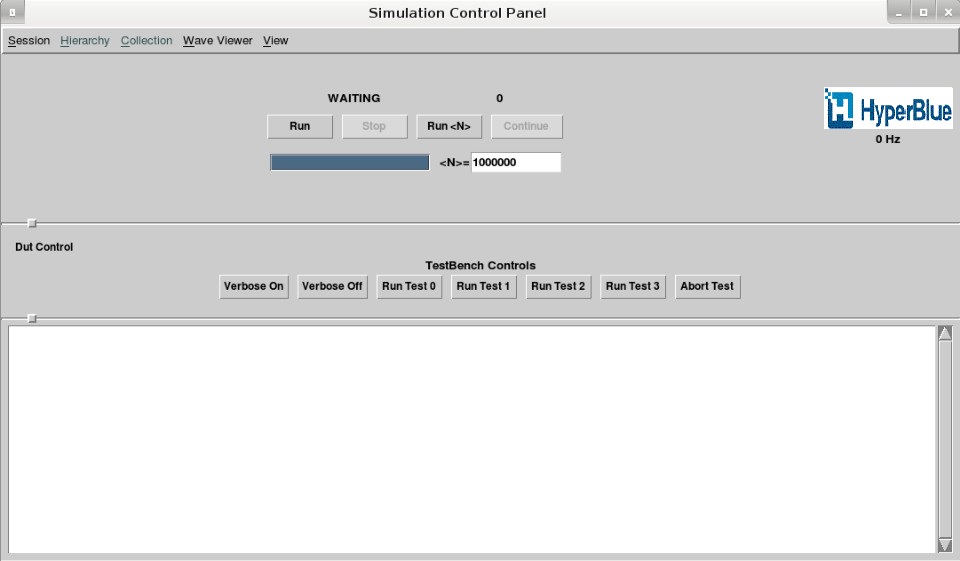


图 8.1 Simulation Control Panel界面

“Simulation Control Panel”界面被分为3部分。每个部分都可以通过各部分之间的分割器上的手柄调整大小。在View菜单中可以选择是否显示设计层次窗口、信号集合和断点窗口，与“Emulation Control Panel”界面中的操作相同。

1. 时钟控制：提供时钟控制功能，显示仿真速度状态。Simulation不支持readback功能，信号值被写入到VCD文件，用户可以打开VCD文件查看波形；
2. DUT控制：为Manual Testbench提供DUT的输入和控制；
3. Message：显示DUT输出的信息（仅在Manual Testbench和C++ Testbench时）。

### 时钟控制

通过最上一行的功能按钮可以实现时钟控制：

1. Run：启动受控的时钟；
2. Stop：停止受控的时钟；
3. Run<N>：推进N个时钟周期；
4. Continue：再次启动受控的时钟，而不复位时钟周期数（N），只有当点击Run<N>按钮，而且时钟停止之前再次点击Stop按钮时，Continue按钮才会处于激活状态。

系统的速度在右上角显示。系统的状态和时钟周期数在按钮的上方显示。

### DUT控制

该窗口包含DUT输入端口的入口域、控制数据传输的按钮、DUT输出端口的显示区域。对于Manual Testbench，这部分已经在编译工程时生成。对于C++ Testbench，这部分不显示。如果用户为C++ Testbench定义GUI，那么可以从View菜单中显示显示这部分。

### Message

Message窗口显示一些Semu的输出信息。其他的信息输出到“Testbench Console”和“Simulation Console”窗口中。

“Testbench Console”窗口显示C++ Testbench中的cout语句的输出信息。显示Manual Testbench中DUT输出的信息。

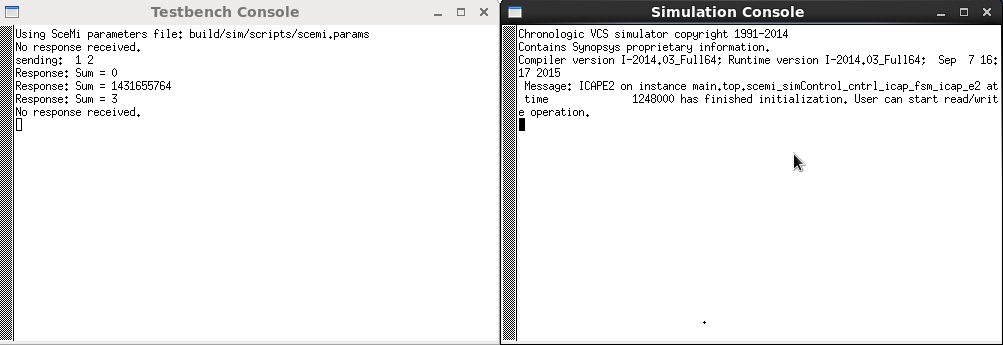


图 8.2 Simulation 窗口

使用C++ Testbench时，如果用户的C++ Testbench不带自己写的GUI，那么DUT控制窗格将会是空白的，Message窗格也不会显示任何信息。其他的部分与使用Manual Testbench时一致。

## 波形获取

Semu的波形获取方式分为两种：Readback、Probe。

1. Readback波形获取方式：
   * 支持获取设计中的所有寄存器类型信号的波形，生成VCD文件；
   * 支持硬件断点（仅在使用C/C++ Testbench时支持）；
   * 在启动Emulation之前设置待抓取的信号（可通过API函数从文件中获取或从界面添加信号），支持动态修改待抓取的信号，而不需要重新编译工程；
   * 速度较慢；
   * 待测设计输入的源文件可以是Vivado支持的任何一种源文件类型。
2. Probe波形获取方式：
   * 支持获取设计中的所有类型信号的波形，并生成VCD波形文件或FSDB波形文件；
   * 支持最多一次获取10752-bit信号的波形；
   * 支持硬件断点（仅在使用C/C++ Testbench时支持）；
   * 抓取不同的信号，需要重新编译工程；
   * 待测设计输入的源文件必须是Verilog RTL源码；
   * 速度较快。

### 9.1 Readback波形获取

Semu的Readback波形获取方式支持FPGA内部任意编译后存在的寄存器。由于Readback不需要对DUT进行预先综合，所以Semu的信号可见性和断点功能允许在运行Emulation期间动态选择需要观测的DUT内部寄存器。

每个被添加到Readback中的信号都会降低仿真速度，添加的信号越多，仿真速度越慢。

### 9.2 Probe波形获取

Probe波形获取方式支持获取DUT内部所有类型信号的波形，每次获取新增信号的波形时需要重新编译工程，最多支持获取10752-bit信号的波形，支持硬件断点。

#### 9.2.1 准备工作

在使用Probe波形获取方式之前，必须安装Verdi，并设置如下环境变量：

export NOVAS\_INST\_DIR=$NOVAS\_HOME

export NPI\_PLATFORM=LINUX64\_GNU\_422

export LD\_LIBRARY\_PATH=”$NOVAS\_INST\_DIR/share/NPI/lib/$NPI\_PLATFORM:”$LD\_ LIBRARY\_PATH

注：其中$NOVAS\_HOME是Verdi软件安装包的根路径；变量NPI\_PLATFORM也可以设置为LINUXAMD64等。

#### 9.2.2 工程配置

使用Probe波形获取功能的工程配置如下：

1. 在“Configure Hardware”中设置如下选项框：
   * wave method：指定获取波形的方式，可选Readback、Probe。Readback方式支持获取DUT内部所有寄存器类型信号的波形，每次获取新增信号的波形时不需要重新编译工程，支持硬件断点；Probe方式支持获取DUT内部所有类型信号的波形（除了多维数组信号、信号路径中带有中括号的信号之外），每次获取新增信号的波形时需要重新编译工程，最多支持获取10752-bit信号的波形，支持硬件断点。
2. 在“Configure Hardware->Advance Options”中设置如下选项框：
   * Design File List：Probe获取波形方式下的DUT的设计文件列表。
   * Probe CMD File：Probe获取波形方式下的PROBE信号描述文件；描述待采样信号的路径。
   * Probe Lib File：Probe获取波形方式下的库文件列表，其中库是指Vivado工程中使用Xilinx的原语或IP等。Design File List和Probe Lib File中的文件合并形成一个新的列表文件，这个列表文件是最终的Probe设计内部信号的设计文件列表。对于Xilinx原语（比如BUFG等）用户可以写一个端口描述模型（仅包含模块名和端口声明即可），并将其路径和文件名添加到Probe Lib File中，确保Semu不会因为工程文件不全而导致运行失败。
   * Probe Skip Modu File：Probe获取波形方式下的库模块名列表文件；库模块名是指库的模块名，Semu在获取指定层次及其以下的信号时会在遇到库模块时自动终止，即不会获取库模块及库模块以下层次的所有信号的波形。另外，可以添加设计中的其他模块，实现SKIP指定模块及模块一下层次的所有信号的波形的功能。
   * Probe Frequency (MHz)：Probe获取波形方式下的采样时钟频率。

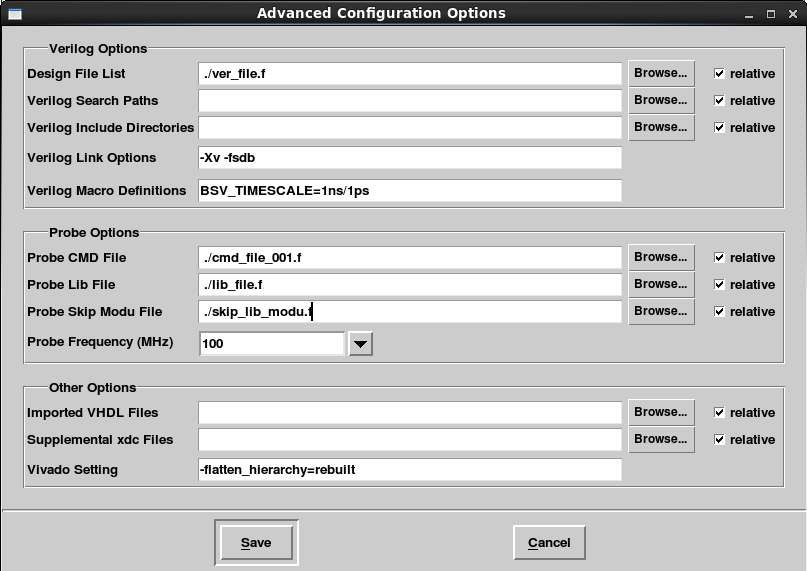


图9.1 Probe波形获取的硬件配置窗口

**PROBE信号描述文件格式**

PROBE信号描述文件是描述待采样信号的文件。

该文件支持以下描述信号的方法：

1. 写出信号的全路径，比如：

me\_top.U\_ME\_UNIT.U\_ADDRESS\_GENERATOR\_INST\_00\_00.snd\_address

1. 按照层次选取指定类型的信号，格式如下：

inst\_path [–max\_depth max\_value] [–input] [–output] [–inout] [–port]

inst\_path：从设计顶层到指定层次的层次路径

1. -max\_depth max\_value：从指定层次开始向下dump max\_value层；如果max\_value=-1，则表示dump从指定层次开始向下所有层次
2. -input：仅dump input类型信号
3. -output：仅dump output类型信号
4. -inout：仅dump inout类型信号
5. -port：仅dump端口信号（包括input、output、inout）

注意：中括号中的选项至少需要有一个，否则inst\_path会被认为是信号路径，而不是层次路径

1. 允许出现空白行或以“//”开头的注释行。

**PROBE信号描述文件举例**

// top.U\_ME\_UNIT -input -max\_depth 0

me\_top.U\_ME\_UNIT.U\_ADDRESS\_GENERATOR\_INST\_00\_00.snd\_address

me\_top.U\_ME\_UNIT.U\_ADDRESS\_GENERATOR\_INST\_00\_01.snd\_address

me\_top.U\_ME\_UNIT.U\_ADDRESS\_GENERATOR\_INST\_00\_02.snd\_address

me\_top.U\_ME\_UNIT.U\_ADDRESS\_GENERATOR\_INST\_00\_03.snd\_address

me\_top.U\_ME\_UNIT.U\_TOP\_CONTROLLER.io\_tmp

me\_top.U\_ME\_UNIT.U\_TOP\_CONTROLLER.rev\_tmp

me\_top.U\_ME\_MEM.U\_REF\_AGU\_INST\_00 –port

me\_top.U\_ME\_UNIT.U\_ADDER\_ARRAY\_INST\_00 -output -max\_depth -1

me\_top.U\_ME\_MEM.U\_REF\_AGU\_INST\_01 -max\_depth 0

**Probe回读波形对信号的限制**

在Probe获取波形方式下，不支持的信号的种类如下：

1. 带位域选择信息的信号，比如：sig\_a[7:0]；
2. 多维数组信号，比如：reg [7:0] sig\_a [0:3]; reg sig\_b [0:7]; reg sig\_c[4]；
3. Generate语句生成的模块等范围下面的所有层次中的信号；
4. Begin-end块、function块、task块、initial块等块中的所有信号。

**Probe波形回读的启动方法**

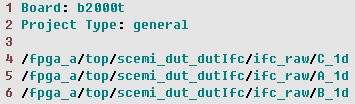
在Testbench中调用如下函数即可启动Probe波形回读功能。

unsigned int do\_add\_and\_enable\_signals(string filename, bool is\_full\_path);

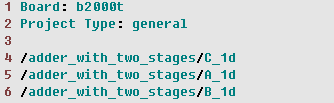
其中第一个参数为文件名，用户可以用工程根目录下的signals\_for\_wave.lst文件的文件名作为输入参数；第二个参数为路径类型标识，用于指明第一个参数对应的文件中的信号路径是否是Vivado工程中的全路径，值为1时表示信号路径是全路径，值为0时表示信号路径是DUT中的路径。

signals\_for\_wave.lst文件举例：

1. 全路径



1. DUT路径



注意：

1. 设计文件列表和库文件列表中不能包含例化了顶层模块的模块所在的文件（比如顶层模块的Testbench文件等），否则会出错。
2. 不能缺少源文件，否则会出错。比较常见的是Vivado中的Xilinx原语（比如BUFG等），需要用户写Verilog模型，并将其文件添加到库文件列表中，将其模块名添加到库模块列表中。
3. include的文件尽量不要出现在设计文件列表和库文件列表中。

#### 9.2.3 工程编译

在完成工程配置后，建议执行“Build->Check Probe Size”对PROBE信号描述文件中描述的待获取波形的信号的总比特数进行统计，查看是否超出上限值（10752-bit）。如果没有超出上限值，那么可以执行“Build->Build Full Emulation”编译工程，否则，修改PROBE信号描述文件直到其中描述的待获取波形的信号的总比特数小于等于上限值（10752-bit）之后再编译工程。

#### 9.2.4 工程运行

在界面上点击“Run->Run Emulation”即可。

#### 9.2.5 波形生成

在Probe波形获取方式下，当Emulation结束后，执行“Build->Convert Probe VCD”可以对获得的编码波形文件解码生成VCD波形文件或FSDB波形文件。

Probe样例及详细描述请参见Semu软件安装包下的lib/training/semu/example/probe中的文件。

## C++ Testbench的写法

在“Configure Testbench”窗口中可以添加新的testbench，并生成testbench模板文件。用户可以利用生成的testbench模板文件和Semu提供的C-API写C++ Testbench对DUT进行测试。

写C++ Testbench的步骤如下：

1. 定义接口：将DUT顶层模块的端口组织成接口，生成pin-file文件。参见[3.4 Configue RTL Pins](#_3.4_Configure_RTL)。pin-file是生成C-API的输入文件。C++ Testbench中不允许使用Lockstep transactors。
2. 定义testbench：在“Configure Testbench”界面中勾选C++ Testbench选项，并点击Add按钮新增testbench。在弹出的“C++ Testbench Definition”界面中输入testbench的名字，点击“Add Starter Templates”按钮，生成模板文件和C-API文件。
3. 编辑testbench：编辑生成的usertb.cpp文件，添加测试代码。

编译工程，然后就可以使用选定的C++ Testbench对DUT进行测试了。

### 模板文件

点击“Add Starter Templates”按钮后，Semu软件会根据pin-file在Testbench Directory文件夹下生成如下文件：

src/*testbench\_name*/TclTb.cpp

src/*testbench\_name*/gui\_dut.tcl

src/*testbench\_name*/usertb.cpp

usertb.cpp：是Semu生成的Testbench模板文件，用户可以在这个文件中添加自己的测试代码。

gui\_dut.tcl：实现在Testbench中添加GUI界面。如果用户写了一个基于Tcl语言的可以嵌入到“Emulation Control Panel”或“Simulation Control Panel”界面中的GUI，那么需要编辑这个文件。

TclTb.cpp：这个文件不应该被修改。

同时，对于general类型工程，Semu会生成build/cpp/capi.h文件；对于sdt类型工程，Semu会生成transactors/transactors\_capi.h文件；这是针对于具体工程封装的C-API头文件，用户直接调用这个文件中的C-API函数即可。

### 10.2 数据类型

BitT<n>类型是与Verilog中的reg/wire [n-1:0] signal\_name;对应的一种数据类型，其中n代表变量的位宽。BitT<n>类型不能直接参数算数和逻辑运行，如果需要对其进行运算，则需要进行强制类型转换。

### 10.3 C++ Testbench Example

#### 10.3.1 例子1

这个例子的DUT为一个加法器。代码如下：

module mkAdder(

CLK,

RST\_N,

start\_a,

start\_b,

start\_cin,

EN\_start,

RDY\_start,

EN\_take,

take,

RDY\_take

);

1. 定义接口

第一步是将DUT的顶层模块组织成接口。在这个例子中，我们可以定义两个handshake transactors：start和take。如下图所示。

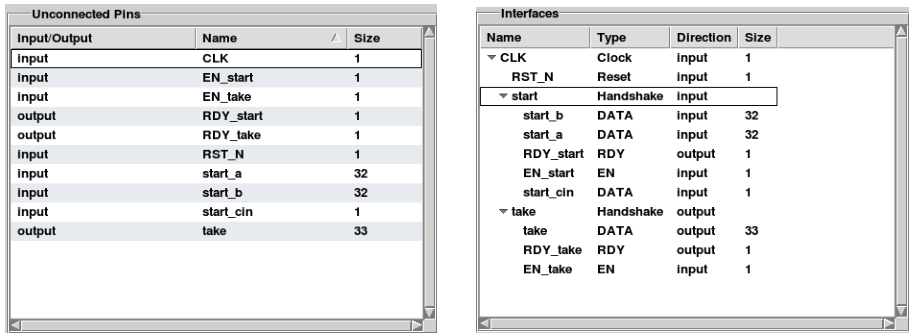


图10.1 DUT顶层模块组织

1. 定义Testbench

下一步是定义Testbench，参见[3.5.1 定义C++ Testbench](#_3.5.1_定义C++_Testbench)。本例子中的testbench名字为addertb。

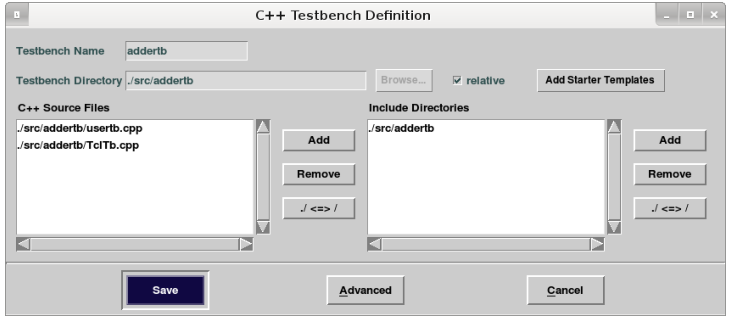


图10.2 定义Testbench界面

点击“Add Starter Templates”按钮生成模板文件和C-API文件。工程目录下的build/cpp/capi.h文件或transactors/transactors\_capi.h中包含了Semu生成的C-API函数，通过这些函数可以实现与DUT之间的通讯。Semu为每个接口生成了阻塞transactor和非阻塞transactor。

对于每个数据端口方向为输入的handshake接口，Semu生成了put interface transactors：

// send/put (non-blocking) and sendB/putB (blocking) to the DUT

bool semu\_put\_start(BitT<32> &start\_b, BitT<32> &start\_a, BitT<1> &start\_cin);

bool semu\_putB\_start(BitT<32> &start\_b, BitT<32> &start\_a, BitT<1> &start\_cin);

对于每个数据端口方向为输出的handshake接口，Semu生成了get interface transactors：

// receive/get (non-blocking) and receiveB/getB (blocking) from the DUT

bool semu\_get\_take(BitT<33> &take);

bool semu\_getB\_take(BitT<33> &take);

使用这些函数可以向DUT发送数据，或接收DUT输出数据。到目前为止，本文档已经介绍了需要写C++ Testbench所需要的所有信息。

1. 编译Testbench

本例中我们对Semu生成的testbench模板文件usertb.cpp进行修改，添加测试代码，得到最终的C++ Testbench文件。

**Include语句**

usertb.cpp文件以#include语句开头，如下：

#include <iostream>

#include "capi.h"

#include "usertb.h"

using namespace std;

**时钟控制**

usertb.cpp文件中存在一个do\_test函数，用户可以在这个函数中添加测试代码，实现测试功能。

首先，需要定义是否在testbench中控制时钟，如果不在testbench中控制时钟，那么当运行Emulation并点击“Emulation Control Panel”界面中的Run按钮时，DUT的时钟就一直处于运行状态。如果在testbench中控制时钟，那么用户需要在testbench中推进时钟，这可以精确地掌控与DUT通讯的时间点。默认testbench中控制时钟。

///////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////

//

// If you do NOT plan to actively control the DUT clock from within your C

// testbench, then you should UNcomment the following lines of code. When

// this code is active, then the DUT clock will start running as soon as

// the user clicks ’Run’ in the Emulation Control Panel.

//

// If, instead, you plan to control the clock (e.g. single-step the clock)

// from within your C testbench, then leave it ’as is’ (that is, commented

// out).

//

///////////////////////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////////////////////

下面的语句功能是取消testbench对时钟的控制。这些语句在生成的usertb.cpp文件中是被注释掉的。本例中的testbench不对时钟进行控制。

if (!semu\_start\_controlled\_clock()) {

cerr << "Something wrong, cannot start controlled clock" << endl;

return 0;

}

**用户测试代码**

接下来就可以写用户自己的测试代码了。

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \*\*\*\*\*\*\*\*\*\*\*\* Here’s where you insert testbench code \*\*\*\*\*\*\*\*\*\*\*\*\*

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

在这段注释之后，用户可以直接添加自己的测试代码或从外部文件中调用C++函数。但被使用的外部文件必须被添加到“C++ Testbench Definition”界面中的“C++ Source Files”窗格中。

接下来会用到capi.h中的semu\_put\_start和semu\_get\_take函数。

1. semu\_put\_start：向DUT发送数据

// Send

while (!semu\_put\_start(a\_data, b\_data, Cin\_data)) {

usleep(1);

}

1. semu\_get\_take：接收DUT输出的数据

// Receive

while (!semu\_get\_take(sum\_data)) {

usleep(1);

}

Example: 在do\_test函数中直接添加测试代码。

// \*\*\*\*\*\*\*\*\*\*\*\* Here’s where you insert testbench code \*\*\*\*\*\*\*\*\*\*\*\*\*

BitT<32> a\_data;

BitT<32> b\_data;

BitT<1> Cin\_data;

BitT<33> sum\_data;

BitT<33> sum;

int \*a, \*b, \*Cin;

long decsum;

// Initialize values for the next batch

a = new int[ARRAYSIZE];

b = new int[ARRAYSIZE];

Cin = new int[ARRAYSIZE];

int i;

for (i=0; i<ARRAYSIZE; i++) {

a[i] = rand() % 50;

b[i] = rand() % 50;

Cin[i] = rand() % 2;

}

cout << "Start Testing" << endl;

for (i=0; i<ARRAYSIZE; i++) {

// Set the values of payload

a\_data = a[i];

b\_data = b[i];

Cin\_data = Cin[i];

// Send

while (!semu\_put\_start(a\_data, b\_data, Cin\_data)) {

usleep(1);

}

cout << "Sent #" << std::dec << i << "(a=" << std::dec << a[i] << " b=" << std::dec << b[i] << " Cin=" << Cin[i]<< ")" << endl;

// Receive

while (!semu\_get\_take(sum\_data)) {

usleep(1);

}

decsum = sum = (a[i]+b[i]+Cin[i]);

if (sum == sum\_data) {

cout << " Result #" << std::dec << i << " Sum = " << std::dec << decsum << endl;

cout << " => GOOD!" << endl;

}

else {

cout << " Result #" << std::dec << i << " Sum should be = " << std::dec << decsum << " but got " << sum\_data << endl;

cout << " => Something wrong!" << endl;

}

}

cout << "Done Testing" << endl;

//////////////////////////////////////////////////////////////////////////////

done\_testing = true;

return 1;

}

Example: 在do\_test函数中调用外部文件中的函数。

// \*\*\*\*\*\*\*\*\*\*\*\* Here’s where you insert testbench code \*\*\*\*\*\*\*\*\*\*\*\*\*

sram\_test ();

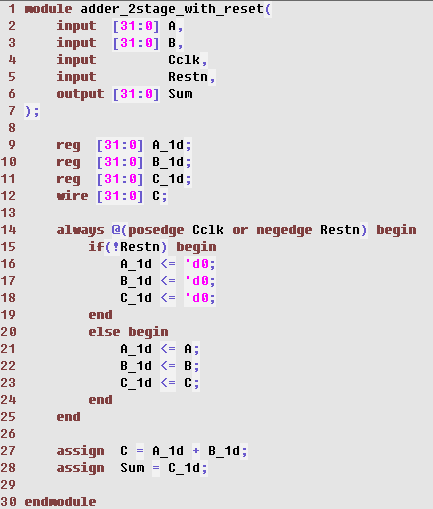
loopback\_test ();

return 1;

}

#### 10.3.2 例子2

本例子的DUT是一个两级寄存的加法器，Verilog代码如下：



1. 定义接口

第一步是将DUT的顶层模块组织成接口。在本例子中，我们将除了时钟和复位之外的端口信号全部定义为Port接口。如下图所示。

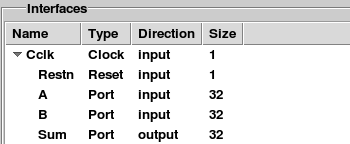
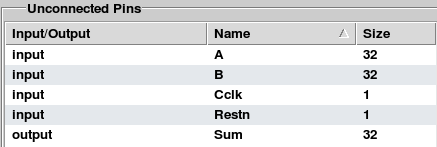


图10.3 接口定义

1. 定义Testbench

下一步是定义Testbench，参见[3.5.1 定义C++ Testbench](#_3.5.1_定义C++_Testbench)。本例子中的Testbench名字为adderTb。

1. 编译Testbench

本例中我们对Semu生成的Testbench模板文件usertb.cpp进行修改，添加测试代码，得到最终的C++ Testbench文件。

**capi.h文件或transactors\_capi.h**

Semu在工程目录下的build/cpp文件夹下生成了capi.h文件或在transactors文件夹下生成transactors\_capi.h文件，这个文件中包含了文件包含了Semu生成的C-API函数，函数名均以“semu\_”开头，通过这些函数可以实现与DUT之间的通讯。Semu为每个接口生成了阻塞transactors和非阻塞transactors。这些函数的返回值为bool类型，返回值为1表示执行成功，返回值为0表示执行失败。

本例中的DUT的顶层模块的端口（除了时钟和复位）全部被定义成Port接口。

对于每个输入端口，semu生成了send transactors。

**Functions for A\_data**

非阻塞send

* bool semu\_send\_A(BitT<32> &A\_data);

阻塞send，直到数据被发送

* bool semu\_sendB\_A(BitT<32> &A\_data);

带应答的阻塞send，直到DUT接收到数据

* bool semu\_sendBAck\_A(BitT<32> &A\_data);

**Functions for B\_data**

* bool semu\_send\_B(BitT<32> &B\_data);
* bool semu\_sendB\_B(BitT<32> &B\_data);
* bool semu\_sendBAck\_B(BitT<32> &B\_data);

对于每个输出端口，semu生成了receive transactors。

**Functions for Sum\_data**

非阻塞receive

* bool semu\_receive\_ Sum(BitT<32> &Sum\_data);

阻塞receive，直到接收到DUT输出的数据

* bool semu\_ receiveB\_Sum(BitT<32> & Sum\_data);

另外，capi.h文件中还生成了与时钟控制相关的函数。

**Functions for clock-control**

非阻塞时钟推进函数（仅能推进一个周期）

* bool semu\_advance\_controlled\_clock(unsigned int number\_of\_edges,

const char \*clock\_name=NULL);

阻塞时钟推进函数，可通过输入参数决定推进的周期数

* bool semu\_advance\_controlled\_clockB(unsigned int number\_of\_edges,

const char \*clock\_name=NULL);

取消时钟控制函数，DUT时钟不受控，不可以使用时钟推进函数控制仿真时间推进

* bool semu\_start\_controlled\_clock(const char \*clock\_name=NULL);

使能时钟控制函数，DUT时钟受控，可以使用时钟推进函数控制仿真时间推进

* bool semu\_stop\_controlled\_clock(const char \*clock\_name=NULL);

复位控制函数，将受控复位信号置为有效若干周期后再置为无效

* bool semu\_assert\_reset(unsigned char number\_of\_cycles,

const char \*reset\_name=NULL);

获取当前仿真时刻的受控时钟运行周期数函数

* bool semu\_get\_current\_controlled\_clock\_cycle(SceMiU64 &cycles,

const char \*clock\_name=NULL));

**Include语句**

usertb.cpp文件以#include语句开头，如下：

#include <iostream>

#include "capi.h"

#include "usertb.h"

using namespace std;

**添加宏定义**

为了方便Testbench的修改，并增强Testbench的可重用性，声明如下宏定义：

// the size of memory in this testbench

#define SIZE 256

// the number of stages in the pipeline

#define NSTAGE 2

// a working value based on the pipeline

#define SENTINDEX (i-(NSTAGE+1))

// the number of checks to run

#define NUMCHECKS 5000

**do\_test函数定义**

int do\_test() {

cout<<"Enter do\_test ..."<<endl;

**时钟控制**

usertb.cpp文件中存在一个do\_test函数，用户可以在这个函数中添加测试代码，实现测试功能。

首先，需要定义是否在Testbench中控制时钟，如果不在Testbench中控制时钟，那么当运行Emulation并点击“Emulation Control Panel”中的Run按钮时，DUT的时钟就一直处于运行状态。如果在Testbench中控制时钟，那么用户需要在Testbench中推进时钟，这可以精确地掌控与DUT通讯的时间点。默认Testbench中控制时钟。

下面的语句功能是取消Testbench对时钟的控制。这段代码在生成的usertb.cpp文件中是被注释掉的。本例中的Testbench不使用这段代码，即Testbench对时钟进行控制。

/\*

if (!semu\_start\_controlled\_clock()) {

cerr << "Something wrong, cannot start controlled clock" << endl;

return 0;

}

\*/

**用户测试代码**

接下来就可以写用户自己的测试代码了。

Example: 在do\_test函数中直接添加测试代码。

// \*\*\*\*\*\*\*\*\*\*\*\* Here's where you insert testbench code \*\*\*\*\*\*\*\*\*\*\*\*\*

BitT<32> a\_data;

BitT<32> b\_data;

BitT<32> sum\_data;

int \*Achk, \*Bchk;

long Sum\_chk, Sum\_rcv, Sum\_prev;

cout << "Start Testing" << endl;

// Initialize values for the next batch

Achk = new int [SIZE];

Bchk = new int [SIZE];

Sum\_chk = 0;

Sum\_prev = -1;

int i, k;

i = 0;

while (i <= NUMCHECKS) {

// Set the values of payload

a\_data = Achk[i%SIZE] = rand() % 100;

b\_data = Bchk[i%SIZE] = rand() % 100;

k = rand() % 1000;

k = 0; /\* Comment this line to inject errors \*/

if (k == 8) {

/\* inject errors to check that TB catches them \*/

a\_data = Achk[i%SIZE]+10;

if ((Achk[i%SIZE]+10+Bchk[i%SIZE]) == Sum\_prev) continue;

}

else {

/\* prevent same consecutive sum \*/

if ((Achk[i%SIZE]+Bchk[i%SIZE]) == Sum\_prev) continue;

}

// Send

if(DEBUG\_EN == 1) {

cout << std::dec << "Send A=" << a\_data << ", B=" << b\_data << endl;

}

semu\_sendBAck\_A(a\_data);

semu\_sendBAck\_B(b\_data);

// Advance the controlled clock 1 cycle

semu\_advance\_controlled\_clockB(1);

if (i > NSTAGE) {

// Receive

semu\_receiveB\_Sum(sum\_data);

if(DEBUG\_EN == 1) {

cout << std::dec << "Get Sum=" << sum\_data << endl;

}

Sum\_chk = Achk[SENTINDEX % SIZE] + Bchk[SENTINDEX % SIZE];

Sum\_rcv = sum\_data;

//cout << SENTINDEX << ": Rec Sum=" << Sum\_rcv << endl;

if ((i > 0) && (Sum\_chk-Sum\_rcv != 0)) {

cout << std::dec << "Error: i = " << SENTINDEX << " Achk = " <<

Achk[SENTINDEX % SIZE]

<< " Bchk = " << Bchk[SENTINDEX % SIZE] << " Sum\_chk = " <<

Sum\_chk

<< " not equal to Sum\_rcv = " << Sum\_rcv << endl;

}

else {

cout << i << ": compare is successed" << endl;

}

}

else if ((i == 0) && NSTAGE) {

semu\_receiveB\_Sum(sum\_data);

Sum\_rcv = sum\_data;

cout << "Receive first initialized output: " << Sum\_rcv << endl;

}

// Save the previous sum for comparison to avoid blocking

Sum\_prev = (Achk[i%SIZE]+Bchk[i%SIZE]);

i++;

}

cout << "Check the rest left in the pipeline" << endl;

// Check the rest left in the pipeline

for (; i<=(NUMCHECKS+NSTAGE); i++) {

// Receive sum output

semu\_receiveB\_Sum(sum\_data);

// Check sum output

Sum\_chk = Achk[SENTINDEX % SIZE] + Bchk[SENTINDEX % SIZE];

Sum\_rcv = sum\_data;

if (Sum\_chk-Sum\_rcv != 0) {

cout << std::dec << "Error: j = " << SENTINDEX << " Achk = " <<

Achk[SENTINDEX % SIZE]

<< " Bchk = " << Bchk[SENTINDEX % SIZE] << " Sum\_chk = " << Sum\_chk

<< " not equal to Sum\_rcv = " << Sum\_rcv << endl;

}

}

cout << "Done Testing" << endl;

/////////////////////////////////////////////////////////////////////////////

done\_testing = true;

return 1;

}

#### 10.3.3 其他

**批量数据传输C-API**

上面的两个例子分别为时钟不受控和时钟受控的例子，例子中使用的C-API均为单个数据的send和receive函数。除了这些单个数据的send和receive函数外，Semu还提供批量数据的send和receive函数，这些函数可以一次发送大批量数据，比单个数据的send和receive函数效率高，速度快，函数如下：

bool semu\_vector\_send\_A(std::vector<BitT<32> > &A\_data);

bool semu\_vector\_sendB\_A(std::vector<BitT<32> > &A\_data);

bool semu\_vector\_sendAck\_A(std::vector<BitT<32> > &A\_data);

bool semu\_vector\_send\_B(std::vector<BitT<32> > &B\_data);

bool semu\_vector\_sendB\_B(std::vector<BitT<32> > &B\_data);

bool semu\_vector\_sendAck\_B(std::vector<BitT<32> > &B\_data);

unsigned semu\_vector\_receive\_Sum(std::vector<BitT<32> > &Sum\_data, unsigned minReturned=0, unsigned maxReturned=0);

**AMBA总线接口C-API**

AMBA总线接口的C-API中也包含send、receive函数，只是其中的输入参数的数据类型与上述例子中的C-API中的不同。详情请参考软件安装目录下的lib/SceMi/tlmxactors/文件夹下的MasterProxy.h、SlaveProxy.h、TLMRequest.h、TLMResponse.h等头文件。

### 10.4 Port接口的FIFO通信机制对发送/接收函数的影响

在使用Port接口时，C++ Testbench和DUT之间可以通过send/receive类函数实现数据通信。从testbench角度观察，从testbench到DUT的数据通路称为发送方向，从DUT到testbench的数据通路称为接收方向。Testbench和DUT之间的数据通路可以抽象为两个FIFO，分别为发送方向的FIFO（简称SEND\_FIFO）和接收方向的FIFO（简称RECEIVE\_FIFO）。



图10.4 Port接口通信机制

1. 发送数据通路

当testbench中执行semu\_send\_*portname*、semu\_sendB\_*portname*、semu\_sendBAck\_*portname*函数时，testbench中的数据会存入SEND\_FIFO中，当DUT检测到SEND\_FIFO中非空时，DUT会从SEND\_FIFO中取出数据并发送到相应的DUT端口上。

1. 接收数据通路

DUT的输出数据会存入RECEIVE\_FIFO，当testbench中执行semu\_receive\_*portname*、semu\_ receiveB\_*portname*函数时，testbench会查询RECEIVE\_FIFO是否为空；如果RECEIVE\_FIFO为空，则非阻塞receive类函数返回0表示执行失败，阻塞receive类函数等待FIFO非空时从RECEIVE\_FIFO中读出数据并返回1表示执行成功；如果RECEIVE\_FIFO非空则Testbench从RECEIVE\_FIFO中读出数据存入Testbench中的指定变量。

这种依靠FIFO实现Testbench与DUT之间通信的机制被称为FIFO通信机制。然而，这种FIFO机制会引出一个问题：在某个仿真时刻，使用send类函数向DUT发送的数据不一定能立即呈现在DUT对应端口上，而可能需要推进若干仿真周期后才会呈现在DUT对应端口上；使用receive类函数接收到的数据不一定是DUT对应端口上的实时值，而可能是历史值。下面举例说明。

1. FIFO通信机制下的send类函数问题

多次连续调用send类函数向同一个端口发送不同的数据然后延时若干周期，与每个周期调用send类函数向同一个端口发送不同的数据的结果是一致的。多次连续调用send类函数发送的数据均存储在SEND\_FIFO中，每推进一个时钟周期，DUT会自动检测SEND\_FIFO中是否非空，如果非空，DUT会从SEND\_FIFO中取出数据并发送到相应的DUT端口上。

**说明：**建议不要在同一个仿真时刻多次连续调用send类函数向同一个端口发送数据。

1. FIFO通信机制下的receive类函数问题

在松耦合模式下（参见12.1.1 Port接口），每次DUT端口上的数据发生变化时会新数据存入RECEIVE\_FIFO中。假设DUT端口A的数据变化为0->1->2->3，且在此过程中Testbench没有调用receive类函数，即RECEIVE\_FIFO中存储了0、1、2、3四个数据，在DUT端口A上的数据变为3后，连续调用4次receive类函数，会一次得到这4个数据。即在某一时刻，使用receive类函数接收到的数据不一定是DUT对应端口上的实时值，而可能是历史值。

针对FIFO通信机制下的receive类函数问题可以采用如下方法解决：

在“global.h”中定义存储DUT各个输出端口数据的全局变量，每次期望读取DUT输出端口上的实时值时，调用“actual\_receive.h”文件中的对应函数即可将实时值存入对应的全局变量中。

**举例**：假设DUT有一个8-bit的输出端口port0o。

global.h中的全局变量定义：

BitT<8> port0o;

actual\_receive.h中的函数定义：

void actual\_receive\_port0o(BitT<8> &port0o)

{

BitT<8> tmp\_data;

bool valid;

valid = semu\_receive\_port0o(tmp\_data);

while(valid == 1) {

port0o = tmp\_data;

valid = semu\_receive\_port0o(tmp\_data);

}

}

**说明**：建议在testbench中不将“actual\_receive.h”文件中的函数与semu\_receive\_*portname*、semu\_receiveB\_*portname*函数混合使用，以防止获得错误的实时值。用户可以根据需要参考“actual\_receive.h”文件中的函数定制自己的receive函数。

### 10.5 Testbench的串行特性

C++ Testbench具有C++的串行特性，不具有Verilog语言的并行特性，比如：fork-join语句。而用户在对DUT进行测试时难免会期望用到并行特性以方便编写testbench、增强testbench的功能和灵活性，这时，C++ Testbench的串行特性Testbench的编写带来了一些不便。针对这个问题，用户可以使用C++的串行特性模拟Verilog的并行特性。具体方法是：将Verilog代码中并行的每个周期需要做的动作按照信号组织成独立的函数（在转换成函数的过程中，可以适当添加一些控制变量；如果多个信号之间关系比较紧密，也可以将多个信号的动作组织成一个函数），然后将这些函数放在while(1){}循环中，并在这些函数的后面添加时钟推进函数，推进1个时钟周期。在while(1){}循环中还可以添加循环控制函数，以便退出循环，结束仿真。下面举例说明。

1. 并行的Verilog伪码

always @(posedge clk …) begin

if(…)

data <= …;

…

end

always @(posedge clk …) begin

if(…)

valid <= …;

…

end

1. 串行的C++ Testbench伪码

while(1) {

*some\_loop\_control\_functions;*

cycle\_action\_data(); // data信号每个周期的全部动作函数

cycle\_action\_valid(); // valid信号每个周期的全部动作函数

semu\_advance\_controlled\_clockB(1); // 将仿真时间推进一个时钟周期

}

### 10.6 时钟控制

C-API中的函数可以分为两类：推进仿真时间的函数、不推进仿真时间的函数。在testbench中控制时钟的情况下，推进仿真时间的函数包括：semu\_advance\_controlled\_clockB()、semu\_advance\_controlled\_clock()、semu\_assert\_reset()，其他的函数（包括send/receive）都是不推进仿真时间的。

在testbench中控制时钟的情况下，推进仿真时间的函数只能将仿真时间停止在时钟有效沿之后很小的时间点处，如下图中的point1~point10（时钟上升沿为有效沿）。而由于C++ Testbench的串行特性，在执行推进仿真时间的函数时不能进行其他操作，所以除了推进仿真时间的函数外的函数或操作只能在非连续的时间点（本例中的point1~point10）执行。



如上所述，Semu对时钟的控制比较简单。这种简单的时钟控制机制会引发一些问题。下面举例说明。

C++ Testbench中不支持如下形式的Verilog写法：

reg [31:0] mem [0:255];

wire [7:0] address;

wire [31:0] data\_out;

assign data\_out = mem[address];

上述的verilog描述要求address的值变化时，data\_out的值立即更新，即是一个组合逻辑，要求时钟当拍返回数据。但由于C++ Testbench中所有的信号值的获取和信号赋值都必须发生在时钟沿之后，address值的获取和data\_out信号的赋值至少需要相差一个时钟周期。即在第N个时钟周期只能获得第N-1个时钟时address信号的值，然后在第N个周期根据address信号值对mem进行寻址获得data\_out值。

## Simulation Testbench的写法

Simulation Testbench是指一个已有的RTL Testbench，可以是Verilog Testbench、SystemVerilog Testbench、UVM等。在“Configure Testbench”窗口中可以配置Testbench为Simulation Testbench（即使用RTL Testbench对DUT进行验证），并生成Simulation Testbench与FPGA硬件端数据同步所需的Verilog接口模块和Semu提供的C-API对应的VPI函数。用户可以利用生成的Verilog接口模块对DUT进行验证。

配置Simulation Testbench的步骤如下：

1. 定义接口：将DUT顶层模块的端口组织成接口，生成pin-file文件。pin-file是生成Verilog接口模块和VPI函数的输入文件。Simulation Testbench中除时钟和复位端口外，其他端口只能定义为Port接口。
2. 配置Testbench：在“Configure Testbench”界面中勾选Simulation Testbench选项，并点击Add按钮新增Testbench。在弹出“Simulation Testbench Definition”界面中选择vcs作为RTL Simulator（目前仅支持VCS作为RTL Simulator），并根据需要选择是否使用vcs的gui界面；在Testbench Name中输入Testbench的名字，点击“Add Starter Templates”按钮，生成Verilog接口模块和VPI函数；在Tb File List中指定一个RTL Testbench的filelist。
3. 更改RTL Testbench中实例化DUT时的顶层模块名：使用第一步生成的Verilog接口模块名作为代替RTL Testbench中DUT顶层模块名。

编译工程，然后就可以使用选定的Simulation Testbench对DUT进行测试了。

### 11.1模板文件

点击“Add Starter Templates”按钮后，Semu软件会根据pin-file在Testbench Directory文件夹下生成如下文件：

src/*testbench\_name*/verilog

src/*testbench\_name*/verilog/semu\_svtb\_dut\_interface.v

src/*testbench\_name*/vpi

src/*testbench\_name*/vpi/semu\_svtb\_init.cpp

src/*testbench\_name*/vpi/semu\_svtb\_advance\_clockB.cpp

src/*testbench\_name*/vpi/semu\_svtb\_vpi\_user.cpp

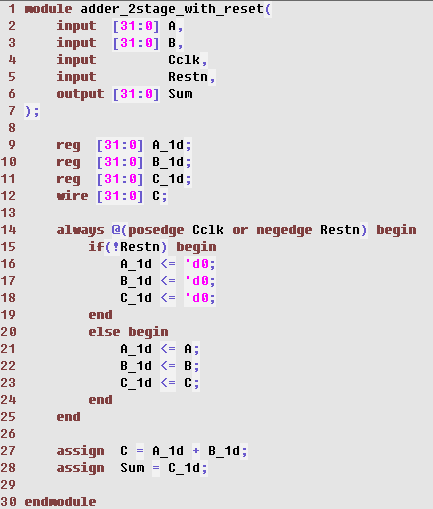
src/*testbench\_name*/vpi/semu\_svtb\_vpi.cpp

1. semu\_svtb\_dut\_interface.v：Testbench与Semu工程之间的接口，用户直接将Testbench中对DUT的例化修改为对本模块的例化即可完成移植工作。
2. semu\_svtb\_init.cpp：Semu工程的初始化函数文件。
3. semu\_svtb\_advance\_clockB.cpp：Semu工程的时钟推进函数文件。
4. semu\_svtb\_vpi\_user.cpp：Semu工程的VPI函数的注册文件。
5. semu\_svtb\_vpi.cpp：Semu工程的VPI函数的实现文件。

### 11.2 Simulation Testbench Example

#### 11.2.1 例子1

这个例子的DUT为一个两级加法器，说明Simulation Testbench的基本用法。代码如下：



1. 定义接口

第一步是将DUT的顶层模块组织成接口。在Simulation Testbench中，我们需要将除了时钟和复位之外的端口信号全部定义为Port接口。如下图所示。

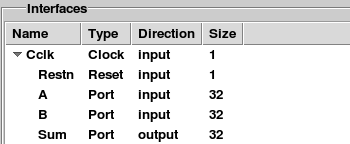
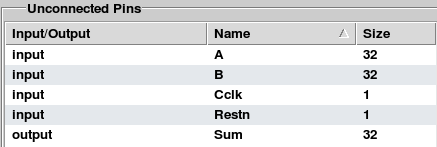


图11.1 DUT顶层模块接口

1. 配置Testbench

下一步是定义Testbench，参见3.5.1 定义C++ Testbench。本例子中的testbench名字为tb\_adder。

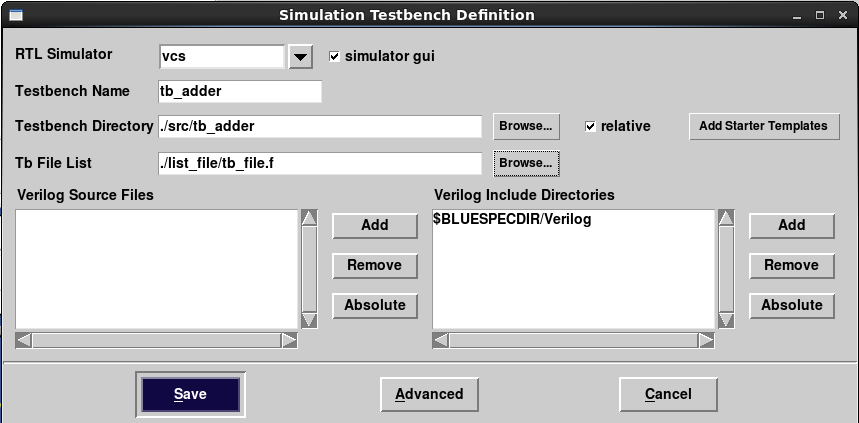


图11.2 Simulation Testbench配置

在Simulation Testbench配置时，选择vcs作为RTL Simulator（目前仅支持VCS作为RTL Simulator），并且本例中选择使用simulator gui，即使用VCS DEV界面（需安装VCS DEV界面）控制仿真流程。

点击“Add Starter Tesmplates”，在Testbench目录./src/tb\_adder下生成vpi和verilog两个文件夹。

**自定义系统任务**

vpi文件夹下包含Simulation Testbench与DUT之间数据交互所必须的自定义系统函数的实现与注册，本例中包含如下自定义系统函数：

1. $semu\_svtb\_init：用于初始化DUT和控制readback、probe的波形获取的过程。
2. $semu\_svtb\_advance\_clockB：控制DUT时钟推进，verilog中每次调用该任务，DUT时钟推进一个周期。
3. $semu\_svtb\_sendB\_A(A)：向DUT的输入端口A发送数据，每调用一次该任务，向端口A发送一次数据。
4. $semu\_svtb\_sendB\_B(B)：向DUT的输入端口B发送数据，每调用一次该任务，向端口B发送一次数据。
5. $semu\_svtb\_receive\_Sum(Sum)：从输出端口Sum接收数据，每次调用最多接收一个加法器运算结果。
6. $semu\_svtb\_flush\_VCD：写vcd文件。

**Testbench与Semu工程接口模块**

Verilog文件下包含Testbench与Semu工程之间实现操作匹配与数据同步的接口模块，该模块命名为semu\_svtb\_dut\_interface。为了在testbench中方便用该接口模块代替DUT进行仿真，该模块的输入输出端口位宽与命名均与DUT的输入输出端口位宽与命名相同。

module semu\_svtb\_dut\_interface( Cclk,

A,

B,

Sum,

Restn);

input Cclk;

input [ 31 : 0 ] A;

input [ 31 : 0 ] B;

output [ 31 : 0 ] Sum;

input [ 31 : 0 ] Restn;

接口模块通过调用自定义的系统任务实现Testbench与Semu工程的交互。

首先初始化Semu工程。

initial begin

$semu\_svtb\_init;

$semu\_svtb\_receive\_Sum(Sum);

end

推进DUT时钟并与DUT输入输出端口数据交互。

always @(posedge Cclk)

begin

$semu\_svtb\_advance\_clockB;

$semu\_svtb\_receive\_Sum(Sum);

end

always @(A)

$semu\_svtb\_sendB\_A(A)

always @(B)

$semu\_svtb\_sendB\_B(B)

控制每个500个周期刷新一次vcd文件。

parameter FLUSH\_VCD\_COUNT = 500;

reg [ 15: 0 ] clk\_count;

initial

clk\_count = 0;

always @(posedge Cclk)

if(clk\_count == FLUSH\_VCD\_COUNT)

begin

$semu\_svtb\_flush\_VCD;

clk\_count <= 0;

end

else

clk\_count <= clk\_count + 1;

**指定Testbench**

Testbench的指定可通过两种方式，一是在Tb File List中指定testbench的file list，即本例中的./list\_file/tb\_file.f；另一种方式是在Verilog Source Files以及Verilog Include Directories中指定相关的文件与文件夹。

1. 更改RTL Testbench中实例化DUT时的顶层模块名

更改RTL Testbench中实例化DUT时的顶层模块名，即将顶层模块名用semu\_svtb\_dut\_interface代替，这样Testbench就能与Semu工程通过semu\_svtb\_dut\_interface模块进行交互。

semu\_svtb\_dut\_interface DUT (

.Cclk (clk ),

.Restn (rst\_n ),

.A (data\_a ),

.B (data\_b ),

.Sum (data\_sum )

);

本例采用System Verilog编写Testbench，Testbench中激励的产生于结果校验如下所示：

initial begin

for(int i = 0; i<100000;i=i+1) begin

@(posedge clk) #1;

data\_a = $random;

data\_b = $random;

end

$display("\*\*\*\*\*\*\*\*\*\*\*\*\*\*simulation is done\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

$stop(0);

end

property check\_property;

@(posedge clk or negedge rst\_n)rst\_n |-> ##1 $past(data\_a+data\_b,2)==data\_sum;

endproperty

adder\_checker: assert property(check\_property)

else $display("[ERROR]:Calculate Error.");

#### 11.2.2 例子2

本例子简要介绍semu对uvm的支持，其DUT的Verilog代码如下：

module dut (clk, rst\_n, rxd, rx\_dv, txd, tx\_en);

input clk;

input rst\_n;

input [ 7:0 ] rxd;

input rx\_dv;

output [ 7:0 ] txd;

output tx\_en;

reg [ 7:0 ] txd;

reg tx\_en;

always @(posedge clk or negedge rst\_n)

if(~rst\_n) begin

txd <= 0;

tx\_en <= 0; end

else begin

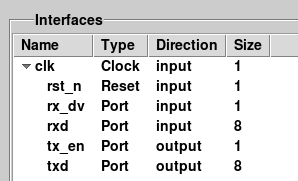
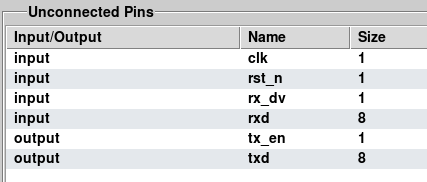
txd <= rxd;

tx\_en <= rx\_dv; end

endmodule

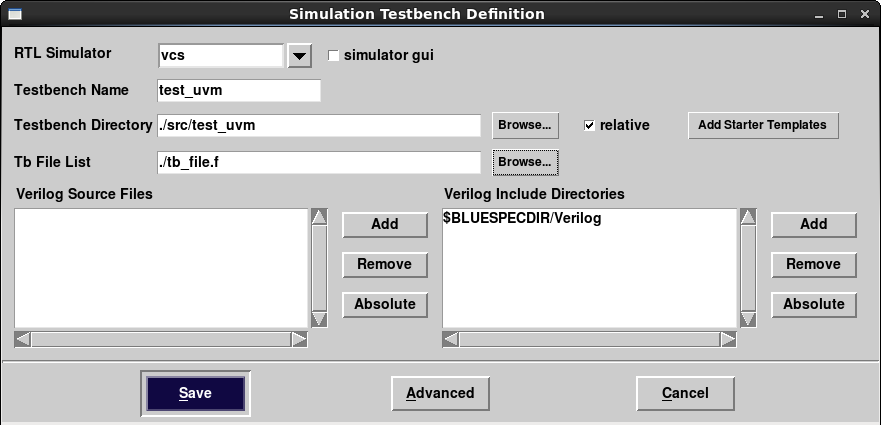
1. 定义接口

第一步是将DUT的顶层模块组织成接口，即将除了时钟和复位之外的端口信号全部定义为Port接口。如下图所示。



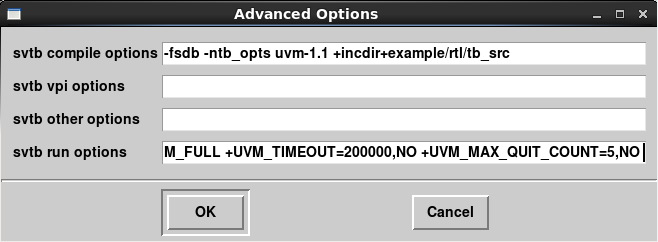
1. 配置Testbench

下一步是定义Testbench，参见3.5.1 定义C++ Testbench，Testbench名字为test\_uvm。



11.3 配置Testbench

本例中不使用VCS DEV界面控制仿真流程。为了使得uvm能够正确的编译和执行，我们还需要配置其他选项，这些选项可通过点击“Advanced”按钮进行配置。



11.4 “Advanced”按钮配置

首先配置“svtb compile options”选项，所配置的内容会作为vcs的编译选项，即在本例中vcs的编译命令中会加入“-fsdb -ntb\_opts uvm-1.1 +incdir+example/rtl/tb\_src”。在“svtb vpi options”和“svtb other options”配置的内容，也作为vcs的编译选项，只是在vcs编译命令中出现的位置不同，比如，在本例中可以在“svtb other options”配置“-debug”，并在最后一个选项“svtb run options”中配置“-gui”，就可以启用VCS DVE界面控制仿真流程。“svtb run options”中可配置执行Simulation Testbench时的选项，即在运行仿真过程中所需要的选项参数，本例中配置为“+UVM\_TESTNAME=my\_case0+UVM\_VERBOSITY=UVM\_FULL +UVM\_TIMEOUT=200000,NO +UVM\_MAX\_QUIT\_COUNT=5,NO”。

1. 更改RTL Testbench中实例化DUT时的顶层模块名

semu\_svtb\_dut\_interface my\_dut (

.clk (clk ),

.rxd (my\_my\_if.rxd ),

.rx\_dv (my\_my\_if.rx\_dv ),

.txd (my\_my\_if.txd ),

.tx\_en (my\_my\_if.tx\_en )

);

### 11.3 用户自定义系统任务/函数的使用

在semu\_svtb\_dut\_interface模块中使用多个自定义的系统任务实现Simulation Testbench与Semu工程的操作匹配与数据同步。当用户有自己的系统任务/函数时，需要做一些简单修改，以免与已有的自定义系统任务冲突。下面以$show\_value()为例说明用户自定义系统任务/函数的使用。

1. 如果用户自定义系统任务/函数没有才使用注册函数，首先编辑注册函数，本例中在$show\_value()的实现文件show\_value.c中编辑如下注册函数：

void RegisterMyTfs()

{

s\_vpi\_systf\_data tf\_data;

tf\_data.type = vpiSysTask;

tf\_data.sysfunctype = 0;

tf\_data.tfname = "$show\_value";

tf\_data.calltf = PLI\_ShowVal\_calltf;

tf\_data.compiletf = PLI\_ShowVal\_compiletf;

tf\_data.sizetf = NULL;

tf\_data.user\_data = NULL;

vpi\_register\_systf(&tf\_data);

return 0;

}

1. 在src/testbench\_name/vpi/semu\_svtb\_vpi\_user.cpp文件中添加用户自定义系统任务/函数的注册函数。如果用户自定义系统任务/函数是用C语言实现，则需要使用extern "C"，如下所示。

……

extern "C"{

void RegisterMyTfs(void);

}

void (\*vlog\_startup\_routines[])()={

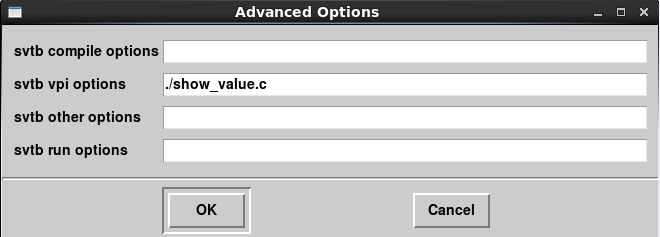
……

RegisterMyTfs,

……

}

1. 在“Advanced Options”中配置“svtb vpi options”为用户自定义系统任务/函数的实现文件，本例中配置为“./show\_value.c”。



11.5 用户自定义系统函数

### 11.4 其他

Readback和Probe波形的回读可通过控制src/*testbench\_name*/vpi/semu\_svtb\_init.cpp函数实现。在semu\_svtb\_init.cpp函数中通过调用do\_add\_and\_enable\_signals(“signals\_for\_wave.lst,1”)函数控制回读波形信号的添加。do\_add\_and\_enable\_signals(“signals\_for\_wave.lst,1”)函数被调用时，添加signals\_for\_wave.lst文件中的信号用于波形回读，否则不添加信号。在Readback波形回读方式下默认不调用do\_add\_and\_enable\_signals(“signals\_for\_wave.lst,1”)函数；Probe波形回读方式下默认调用do\_add\_and\_enable\_signals(“signals\_for\_wave.lst,1”)函数。

## C-API

对DUT进行全面测试的最好方法是使用C++ Testbench。Semu提供C-API可以实现testbench与DUT之间的通信。

C-API分为3种基本类型：

1. 发送或接收数据的接口transactor；
2. 初始化；
3. 时钟和仿真控制。

### 12.1 接口函数

C-API主要支持两种接口类型：

1. Port接口：通过DUT顶层模块的input或output端口实现向DUT发送数据或从DUT接收数据。
2. Handshake（Ready/Enable）接口：支持在发送和接收数据之间的Ready/Enable接口异步握手协议。
3. AMBA接口：包括AHB MASTER、AHB SLAVE、AXI MASTER、AXI SLAVE等。

一个给定的Semu模型可能同时包含Port和Handshake（Ready/Enable）接口。在pin-file中，DUT顶层模块的input、output端口被映射成Port和Handshake（Ready/Enable）接口。

#### 12.1.1 Port接口

Port接口transactors是针对pin-file中的每个Port接口定义的。对于每个DUT顶层模块的input端口，Semu会生成一个发送transactor。对于每个DUT顶层模块的output端口，Semu会生成一个接收transactor。Transactor分为两种：非阻塞transactor、阻塞transactor。非阻塞transactor返回一个布尔值表明发送或接收是否成功。

Port接口与时钟之间的关系由两方面决定：端口方向（input、output）、port与时钟之间的耦合方式（松耦合（Loosely Coupled）、紧耦合（Tightly Coupled））。默认的耦合方式是松耦合（Loosely Coupled）。

1. 输入（Inputs）
   * 紧耦合（LC）：只有当在指定时钟域中的DUT所有紧耦合的输入端口被赋值时，才允许推进受控时钟。
   * 松耦合（TC）：可以在任意时刻推进受控时钟。
2. 输出（Outputs）
   * 紧耦合（LC）：每个受控时钟的时钟周期都会将DUT输出端口的数值更新到FIFO中，与输出端口上的数值是否变化无关。
   * 松耦合（TC）：只有在DUT输出端口的数值变化时最新的数据才呈现在对应松耦合输出的port接口上。

**说明：**建议采用默认的耦合方式。

生成的函数（transactor）如表12.1所示。

表12.1 Port接口transactor

|  |  |  |
| --- | --- | --- |
| Port Transactor Definitions | | |
| Type | Transactor Name | Description |
| Input | semu\_send\_*portname* | 非阻塞发送，返回一个布尔值表明是否发送成功 |
| semu\_sendB\_*portname* | 阻塞发送，直到数据被发送出去才返回 |
| semu\_sendBAck\_*portname* | 带应答的阻塞发送，直到DUT接收到发送的数据才返回 |
| Output | semu\_receive\_*portname* | 非阻塞接收，返回一个布尔值表明是否接收成功 |
| semu\_receiveB\_*portname* | 阻塞接收，直到接收到DUT的数据才返回 |

Port接口的 transactor默认是松耦合的。如果想更改一个Port与时钟的耦合方式，可以使用semu\_set\_emulation\_type函数。

表12.2 Port接口的 transactor耦合修改

|  |  |
| --- | --- |
| Clock/Port Coupling Configuration | |
| semu\_set\_emulation\_type\_*portname*(EmulationType) | 非阻塞transactor |
| semu\_set\_emulation\_typeB\_*portname*(EmulationType) | 阻塞transactor |

**说明：**建议semu\_set\_emulation\_type\_*portname*和semu\_set\_emulation\_typeB\_*portname*仅在testbench的开始处使用，尽量避免在testbench中切换Port与时钟的耦合方式。

举例：

Verilog模块：

module testmodule (

input a,

input[7:0] b,

output[7:0] o

);

Semu生成的非阻塞transactors：

bool semu\_send\_a(BitT<1> &data);

bool semu\_send\_b(BitT<8> &data);

bool semu\_receive\_o(BitT<8> &data);

Semu生成的阻塞transactors：

bool semu\_sendB\_a(BitT<1> &data);

bool semu\_sendB\_b(BitT<8> &data);

bool semu\_sendBAck\_a(BitT<1> &data);

bool semu\_sendBAck\_b(BitT<8> &data);

bool semu\_receiveB\_o(BitT<8> &data);

#### 12.1.2 Handshake (Ready/Enable)接口

Ready/Enable类型接口将ports组织成Handshake transactor，以支持在发送端和接收端之间实现异步握手。

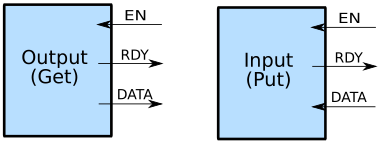


图12.1 Handshake接口模型

当DUT与testbench通过Handshake transactor连接时，一个AND门被添加到transactor之间，如下图所示。

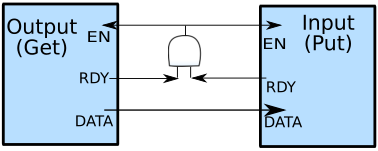


图12.2 Handshake接口连接方式

表12.3 Handshake接口transactor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Get/Put Transactor Definitions | | | | |
| 类型 | 信号 | 方向 | 非阻塞transactor | 阻塞transactor |
| Put | RDY | Output | semu\_put\_*interfacename* | semu\_putB\_*interfacename* |
| EN | Input |
| Data | Input |
| Get | RDY | Output | semu\_get\_*interfacename* | semu\_getB\_*interfacename* |
| EN | Input |
| Data | Output |

举例：

Verilog模块：

module testmodule (

output RDY\_request,

input EN\_request,

input a,

input [7:0] b,

output RDY\_response,

input EN\_response,

output [7:0] o);

生成的C-API：

在这个例子中的两个接口被命名为inputsAB、outputsO。

非阻塞transactors：

bool semu\_put\_inputsAB(BitT<1> &a, BitT<8> &b);

bool semu\_vector\_send\_inputsAB(std::vector<BitT<1> > &a, std::vector<BitT<8> > &b);

bool semu\_get\_outputsO(BitT<8> &o);

unsigned semu\_vector\_receive\_outputsO(std::vector<BitT<8> > &o, unsigned minReturned=0, unsigned maxReturned=0);

阻塞transactors：

bool semu\_putB\_inputsAB(BitT<1> &a, BitT<8> &b);

bool semu\_vector\_sendB\_inputsAB(std::vector<BitT<1> > &a, std::vector<BitT<8> > &b);

bool semu\_vector\_sendAck\_inputsAB(std::vector<BitT<1> > &a, std::vector<BitT<8> > &b);

bool semu\_getB\_outputsO(BitT<8> &o);

unsigned semu\_vector\_receive\_outputsO(std::vector<BitT<8> > &o, unsigned minReturned=0, unsigned maxReturned=0);

#### 12.1.3 AMBA接口

AHB SLAVE发送函数：

bool semu\_send\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMRequest<32u,32u,32u> &req, bool discardResponse = false);

bool semu\_sendB\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMRequest<32u,32u,32u> &req, bool discardResponse = false);

bool semu\_sendT\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMRequest<32u,32u,32u> &req, struct timespec &expiration, bool discardResponse = false);

bool semu\_sendT\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMRequest<32u,32u,32u> &req, const time\_t seconds, const long microseconds=0, bool discardResponse = false);

AHB SLAVE接收函数：

bool semu\_receive\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMResponse<32u,32u> &rsp);

bool semu\_receiveB\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMResponse<32u,32u > &rsp);

bool semu\_receiveT\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMResponse<32u,32u> &rsp, struct timespec &expiration);

bool semu\_receiveT\_ahb\_slave(SlaveProxy<32u,32u,32u> & slavep, TLMResponse<32u,32u> &rsp, const time\_t seconds, const long microseconds=0);

AXI MASTER发送函数：

bool semu\_sendB\_axi\_master(MasterProxy<32u,32u,32u> & masterp, TLMResponse<32u,32u> &rsp);

bool semu\_send\_axi\_master(MasterProxy<32u,32u,32u> & masterp, TLMResponse<32u,32u> &rsp);

bool semu\_sendT\_axi\_master(MasterProxy<32u,32u,32u> & masterp, TLMResponse<32u,32u> &rsp, const time\_t seconds, const long microseconds=0);

AXI MASTER接收函数：

bool semu\_receive\_axi\_master(MasterProxy<32u,32u,32u> & masterp, TLMRequest<32u,32u,32u> &req);

bool semu\_receiveB\_axi\_master(MasterProxy<32u,32u,32u> & masterp, TLMRequest<32u,32u,32u> &req);

bool semu\_receiveT\_axi\_master(MasterProxy<32u,32u,32u> & masterp, TLMRequest<32u,32u,32u> &req, const time\_t seconds, const long microseconds=0);

建议：discardResponse参数设置为true。

### 12.2 初始化

下面的C-API调用控制Emulation/Simulation环境的初始化。不需要用户使用这些初始化函数，生成的C++ Testbench模板中已经默认做了初始化工作。

bool semu\_start\_scemi(const char \*paramfile);

bool semu\_start\_simulation\_control();

bool semu\_start\_readback\_control();

bool semu\_start\_vcd\_writer(const char \*dumpfile=0);

semu\_start\_services函数调用启动所有列出来的项目，semu\_stop\_services函数终结所有的项目。

bool semu\_start\_services(const char \*paramfile);

bool semu\_stop\_services();

### 12.3 时钟和仿真控制

本小节的C-API函数调用具有控制时钟和复位的功能，包括：启动、停止、推进时钟。

1. <ClkName>是pin-file中指定的时钟信号的名字；
2. <ResetName>是pin-file中指定的复位信号的名字。

**Functions for clock-control**

非阻塞时钟推进函数（仅能推进一个周期）

* bool semu\_advance\_controlled\_clock(unsigned int number\_of\_edges,

const char \*clock\_name=NULL);

阻塞时钟推进函数，可通过输入参数决定推进的周期数

* bool semu\_advance\_controlled\_clockB(unsigned int number\_of\_edges,

const char \*clock\_name=NULL);

取消时钟控制函数，DUT时钟不受控，不可以使用时钟推进函数控制仿真时间推进

* bool semu\_start\_controlled\_clock(const char \*clock\_name=NULL);

使能时钟控制函数，DUT时钟受控，可以使用时钟推进函数控制仿真时间推进

* bool semu\_stop\_controlled\_clock(const char \*clock\_name=NULL);

复位控制函数，将受控复位信号置为有效若干周期后再置为无效

* bool semu\_assert\_reset(unsigned char number\_of\_cycles,

const char \*reset\_name=NULL);

获取当前仿真时刻的受控时钟运行周期数函数

* bool semu\_get\_current\_controlled\_clock\_cycle(SceMiU64 &cycles,

const char \*clock\_name=NULL));

**说明：**推进仿真时间的函数只能在Testbench中对时钟进行控制时才能使用，否则执行Testbench时会卡住。

举例：使用C-API函数实现与SceMi受控时钟的交互。函数执行成功，则返回1；反之，返回0。

// Advance controlled clock 1 cycle

if (semu\_advance\_controlled\_clock(1)) do\_something();

// Advance controlled clock 20 cycle

if (semu\_advance\_controlled\_clockB(20)) do\_something();

*or* semu\_advance\_controlled\_clockB(20);

// Start free-running controlled clock

if (semu\_start\_controlled\_clock()) do\_something;

// Stop controlled clock

if (semu\_stop\_controlled\_clock()) do\_something();

// Get the current cycle of the controlled clock

SceMiU64 clock\_cycles;

if (semu\_get\_current\_controlled\_clock\_cycle(clock\_cycles)) do\_something();

// Assert controlled reset for 8 cycles

int number\_of\_cycles = 8;

if (semu\_assert\_reset(number\_of\_cycles)) do\_something();

当执行C++ Testbench时，从“Emulation Control Panel”界面也可以控制时钟。

## 附录

### sdt类型工程支持的接口类型和方法

Created Module object: documentModule

Created Instance object: documentInstance

Top level commands:

allPorts {}

assignPortAttr {attr value ports}

check {}

clear {}

createBVIWrapper {outdir}

createCPP\_Proxies {outdir}

createEMUWrapper {outdir}

createProject {outdir}

createSystemC\_Proxies {outdir}

createTopWrapper {outdir}

createWrapper {outdir}

create\_instance {name module}

debug {args}

deleteXactor {name}

filter {objs args}

findPorts {args}

newXactor {name type args}

read\_hdl {filename}

Documentation for class: Port

Configure options:

reset {}

clock {}

instance {}

bsvtype {}

dir inout

isreset false

width 1

isclock false

name documentPort

module documentModule

Methods:

method bsv\_method\_name {}

method bvi\_entry {strname}

method cget {-option}

method check\_direction {}

method check\_name\_clash {}

method check\_type {}

method configure {?-option? ?value -option value...?}

method constructor {\_name \_dir \_mod args}

method debug {args}

method destructor {<undefined>}

method getInstName {}

method getInstanceObj {}

method getModule {}

method getName {}

method getType {}

method getTypeObj {}

method interfaceDef {strname p1}

method isa {className}

method isport {p}

method readOnly {field srcObj srcMeth}

method show {args}

method stdbool {field}

Documentation for class: IPort

Configure options:

reset {}

clock {}

instance documentInstance

bsvtype {}

dir inout

isreset false

width 1

isclock false

name documentPort

module documentModule

Methods:

method bsv\_method\_name {}

method bvi\_entry {strname}

method cget {-option}

method check\_direction {}

method check\_name\_clash {}

method check\_type {}

method configure {?-option? ?value -option value...?}

method constructor {\_modport \_instanceO}

method debug {args}

method destructor {<undefined>}

method getClockConfig {}

method getInstName {}

method getInstanceObj {}

method getModule {}

method getName {}

method getType {}

method getTypeObj {}

method interfaceDef {strname p1}

method isa {className}

method isport {p}

method readOnly {field srcObj srcMeth}

method setClockConfig {}

method show {args}

method stdbool {field}

Documentation for class: Module

Configure options:

Methods:

method addPort {name dir args}

method cget {-option}

method configure {?-option? ?value -option value...?}

method constructor {mname args}

method debug {args}

method deletePorts {delPorts}

method destructor {<undefined>}

method getName {}

method getPorts {}

method isa {className}

method show {args}

Documentation for class: Instance

Configure options:

Methods:

method cget {-option}

method configure {?-option? ?value -option value...?}

method constructor {iname mod}

method debug {args}

method destructor {<undefined>}

method getModule {}

method getName {}

method getPorts {}

method isa {className}

method show {args}

Transactor classes

Type --> Class

AhbMaster --> AhbMasterSceMiXactor

AhbSlave --> AhbSlaveSceMiXactor

ApbMaster --> ApbMasterSceMiXactor

ApbSlave --> ApbSlaveSceMiXactor

Axi4LMaster --> Axi4LRdWrMasterSceMiXactor

Axi4LRdMaster --> Axi4LRdMasterSceMiXactor

Axi4LRdSlave --> Axi4LRdSlaveSceMiXactor

Axi4LSlave --> Axi4LRdWrSlaveSceMiXactor

Axi4LWrMaster --> Axi4LWrMasterSceMiXactor

Axi4LWrSlave --> Axi4LWrSlaveSceMiXactor

Axi4Master --> Axi4RdWrMasterSceMiXactor

Axi4RdMaster --> Axi4RdMasterSceMiXactor

Axi4RdSlave --> Axi4RdSlaveSceMiXactor

Axi4Slave --> Axi4RdWrSlaveSceMiXactor

Axi4WrMaster --> Axi4WrMasterSceMiXactor

Axi4WrSlave --> Axi4WrSlaveSceMiXactor

AxiMaster --> AxiRdWrMasterSceMiXactor

AxiRdMaster --> AxiRdMasterSceMiXactor

AxiRdSlave --> AxiRdSlaveSceMiXactor

AxiSlave --> AxiRdWrSlaveSceMiXactor

AxiWrMaster --> AxiWrMasterSceMiXactor

AxiWrSlave --> AxiWrSlaveSceMiXactor

Custom --> Custom

Generic --> Custom

Get --> GetXactor

Input --> InputXactor

MasterAhb --> AhbMasterSceMiXactor

MasterApb --> ApbMasterSceMiXactor

MasterAxi --> AxiRdWrMasterSceMiXactor

MasterAxi4 --> Axi4RdWrMasterSceMiXactor

MasterAxi4L --> Axi4LRdWrMasterSceMiXactor

Output --> OutputXactor

Put --> PutXactor

RdMasterAxi --> AxiRdMasterSceMiXactor

RdMasterAxi4 --> Axi4RdMasterSceMiXactor

RdMasterAxi4L --> Axi4LRdMasterSceMiXactor

RdSlaveAxi --> AxiRdSlaveSceMiXactor

RdSlaveAxi4 --> Axi4RdSlaveSceMiXactor

RdSlaveAxi4L --> Axi4LRdSlaveSceMiXactor

SlaveAhb --> AhbSlaveSceMiXactor

SlaveApb --> ApbSlaveSceMiXactor

SlaveAxi --> AxiRdWrSlaveSceMiXactor

SlaveAxi4 --> Axi4RdWrSlaveSceMiXactor

SlaveAxi4L --> Axi4LRdWrSlaveSceMiXactor

WrMasterAxi --> AxiWrMasterSceMiXactor

WrMasterAxi4 --> Axi4WrMasterSceMiXactor

WrMasterAxi4L --> Axi4LWrMasterSceMiXactor

WrSlaveAxi --> AxiWrSlaveSceMiXactor

WrSlaveAxi4 --> Axi4WrSlaveSceMiXactor

WrSlaveAxi4L --> Axi4LWrSlaveSceMiXactor

Documentation for class: AhbMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AhbXtorMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Ahb}

ports {}

fieldnames {}

clockConf {}

type {AhbXtorMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AhbMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Ahb}

lastError ::vInterface0

pipeFifoDepth 32

providedIfc AhbMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: AhbSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AhbXtorSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Ahb}

ports {}

fieldnames {}

clockConf {}

type {AhbXtorSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AhbSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Ahb}

lastError ::vInterface3

pipeFifoDepth 32

providedIfc AhbSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: ApbMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type ApbXtorMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Apb}

ports {}

fieldnames {}

clockConf {}

type {ApbXtorMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2ApbMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Apb}

lastError ::vInterface6

pipeFifoDepth 32

providedIfc ApbMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: ApbSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type ApbXtorSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Apb}

ports {}

fieldnames {}

clockConf {}

type {ApbXtorSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2ApbSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Apb}

lastError ::vInterface8

pipeFifoDepth 32

providedIfc ApbSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4LRdMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4LRdMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4LRdMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4LRdMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface10

pipeFifoDepth 32

providedIfc Axi4LRdMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4LRdSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4LRdSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4LRdSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4LRdSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface11

pipeFifoDepth 32

providedIfc Axi4LRdSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4LRdWrMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4LRdWrMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4LRdWrMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4LRdWrMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface12

pipeFifoDepth 32

providedIfc Axi4LRdWrMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4LRdWrSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4LRdWrSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4LRdWrSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4LRdWrSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface15

pipeFifoDepth 32

providedIfc Axi4LRdWrSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4LWrMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4LWrMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4LWrMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4LWrMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface18

pipeFifoDepth 32

providedIfc Axi4LWrMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4LWrSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4LWrSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4LWrSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4LWrSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface19

pipeFifoDepth 32

providedIfc Axi4LWrSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4RdMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4RdMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4RdMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4RdMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface20

pipeFifoDepth 32

providedIfc Axi4RdMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4RdSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4RdSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4RdSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4RdSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface21

pipeFifoDepth 32

providedIfc Axi4RdSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4RdWrMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4RdWrMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4RdWrMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4RdWrMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface22

pipeFifoDepth 32

providedIfc Axi4RdWrMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4RdWrSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4RdWrSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4RdWrSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4RdWrSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface25

pipeFifoDepth 32

providedIfc Axi4RdWrSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4WrMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4WrMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4WrMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4WrMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface28

pipeFifoDepth 32

providedIfc Axi4WrMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Axi4WrSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type Axi4WrSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi4}

ports {}

fieldnames {}

clockConf {}

type {Axi4WrSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2Axi4WrSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi4}

lastError ::vInterface29

pipeFifoDepth 32

providedIfc Axi4WrSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: AxiRdMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AxiRdMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi}

ports {}

fieldnames {}

clockConf {}

type {AxiRdMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AxiRdMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi}

lastError ::vInterface30

pipeFifoDepth 32

providedIfc AxiRdMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: AxiRdSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AxiRdSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi}

ports {}

fieldnames {}

clockConf {}

type {AxiRdSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AxiRdSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi}

lastError ::vInterface31

pipeFifoDepth 32

providedIfc AxiRdSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: AxiRdWrMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AxiRdWrMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi}

ports {}

fieldnames {}

clockConf {}

type {AxiRdWrMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AxiRdWrMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi}

lastError ::vInterface32

pipeFifoDepth 32

providedIfc AxiRdWrMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: AxiRdWrSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AxiRdWrSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi}

ports {}

fieldnames {}

clockConf {}

type {AxiRdWrSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AxiRdWrSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi}

lastError ::vInterface35

pipeFifoDepth 32

providedIfc AxiRdWrSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: AxiWrMasterSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AxiWrMaster

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi}

ports {}

fieldnames {}

clockConf {}

type {AxiWrMaster #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AxiWrMasterSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi}

lastError ::vInterface38

pipeFifoDepth 32

providedIfc AxiWrMasterSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: AxiWrSlaveSceMiXactor

Configure options:

inPipeDepth 32

dataWidth 32

interleave\_depth 1

addrWidth 32

flow\_depth 0

general\_type AxiWrSlave

outPipeVis Fifo

big\_endian false

outPipeDepth 32

write\_bypass false

inPipeVis Fifo

keep\_bursts true

userWidth 0

bsvpackage {Xactors DefaultValue TLM3 Axi}

ports {}

fieldnames {}

clockConf {}

type {AxiWrSlave #(10,32,32,15,0)}

matchRule {}

resetPort {}

name n2AxiWrSlaveSceMiXactor

expandPortTypes false

pipeMode Fifo

usePipes true

imports {Xactors DefaultValue TLM3 Axi}

lastError ::vInterface39

pipeFifoDepth 32

providedIfc AxiWrSlaveSceMiXactor

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

method sysc\_i\_or\_t {}

Documentation for class: Custom

Configure options:

preInstanceProc {}

proxyConstrArgs {path {"%SI"}}

proxyClass UNDEFINED

moduleArguments {{clocked\_by %CC} {reset\_by %CR}}

cppHeaders {}

xactorModule NOT\_DEFINED

bsvpackage {}

ports {}

fieldnames {}

clockConf {}

type {}

matchRule {}

resetPort {}

name n2Custom

expandPortTypes false

pipeMode Fifo

usePipes false

imports {}

lastError {}

pipeFifoDepth 32

providedIfc NOT\_DEFINED

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {return false}

method isTLM {}

method isa {className}

method loadpackage {}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

Documentation for class: GetXactor

Configure options:

ready {}

enable {}

useProxyQueue false

tightlyCoupled false

structName S\_n2GetXactor

ports {}

fieldnames {}

clockConf {}

type Get#(S\_n2GetXactor)

matchRule {}

resetPort {}

name n2GetXactor

expandPortTypes false

pipeMode Fifo

usePipes false

imports {GetPut Connectable DefaultValue FShow SceMiXactors}

lastError {}

pipeFifoDepth 32

providedIfc Put#(S\_n2GetXactor)

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getCValueType {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {return false}

method isTLM {}

method isa {className}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

Documentation for class: InputXactor

Configure options:

allowFirstClock false

useProxyQueue false

tightlyCoupled false

structName S\_n2InputXactor

ports {}

fieldnames {}

clockConf {}

type Put#(S\_n2InputXactor)

matchRule {}

resetPort {}

name n2InputXactor

expandPortTypes false

pipeMode Fifo

usePipes false

imports {GetPut Connectable DefaultValue FShow SceMiXactors BypassReg}

lastError {}

pipeFifoDepth 32

providedIfc {Get# (S\_n2InputXactor)}

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getCValueType {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {return false}

method isTLM {}

method isa {className}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

Documentation for class: OutputXactor

Configure options:

useProxyQueue false

tightlyCoupled false

structName S\_n2OutputXactor

ports {}

fieldnames {}

clockConf {}

type Get#(S\_n2OutputXactor)

matchRule {}

resetPort {}

name n2OutputXactor

expandPortTypes false

pipeMode Fifo

usePipes false

imports {GetPut Connectable DefaultValue FShow FIFOF SceMiXactors}

lastError {}

pipeFifoDepth 32

providedIfc {Put# (S\_n2OutputXactor)}

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getCValueType {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {return false}

method isTLM {}

method isa {className}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}

Documentation for class: PutXactor

Configure options:

ready {}

enable {}

useProxyQueue false

tightlyCoupled false

structName S\_n2PutXactor

ports {}

fieldnames {}

clockConf {}

type Put#(S\_n2PutXactor)

matchRule {}

resetPort {}

name n2PutXactor

expandPortTypes false

pipeMode Fifo

usePipes false

imports {GetPut Connectable DefaultValue FShow SceMiXactors}

lastError {}

pipeFifoDepth 32

providedIfc Get#(S\_n2PutXactor)

Methods:

method addMatchRules {args}

method addPorts {prts}

method cget {-option}

method checkFieldPortMatch {}

method checkSanity {mode}

method checkTypeHandle {}

method check\_name {}

method configure {?-option? ?value -option value...?}

method constructor {cname args}

method cppIncludes {}

method cppPostConstr {pre objName}

method debug {args}

method debugFields {}

method destructor {<undefined>}

method generateInterfaceDef {strname p1}

method generateState {strname p1}

method generateTypeDef {strname}

method getAllPorts {}

method getCProxyClass {}

method getCValueType {}

method getClockConfig {}

method getInstance {}

method getInstanceObj {}

method getModuleArguments {}

method getName {}

method getPreInstance {}

method getProvidedIfc {}

method getProxyConstrArgs {}

method getResetPort {}

method getSceMiType {}

method getTypeHandle {}

method getTypeName {}

method getXactorInstance {}

method getXactorModule {}

method interfaceDecl {strname p1}

method isSCInitiator {}

method isSCTarget {return false}

method isTLM {}

method isa {className}

method matchFields {}

method proxyArgs {}

method removePorts {prts}

method scemiPath {}

method setTypeHandle {}

method show {args}