# SystemC

#### sc\_port

sc\_port是SystemC中的端口，用于建立模块之间的连接。sc\_port可以连接到一个或多个sc\_interface，它是sc\_module（或其派生类）内部的成员。通过连接sc\_port和sc\_interface，不同的模块之间可以进行信号传递和方法调用，实现数据交换。sc\_port提供了一种简单的机制，用于在模块之间进行通信和互联。

sc\_in、sc\_out继承自sc\_port

#### sc\_interface

sc\_interface是SystemC中定义接口的类。它描述了一组可以在模块之间进行通信的方法和信号。sc\_interface定义了在模块之间进行消息传递的协议和接口规范。sc\_interface可以被模块内的sc\_port连接和实现，从而提供了在不同模块之间进行通信的能力。通过定义不同的sc\_interface，可以在不同模块之间定义不同的通信协议和接口规范。

sc\_port是连接模块之间的端口，用于建立连接。而sc\_interface是描述通信协议和接口规范的类，作为sc\_port的接口。通过sc\_port和sc\_interface的组合，SystemC可以建立起模块之间的通信和互联机制。

sc\_signal、sc\_fifo等继承自sc\_interface。可以自定义继承自sc\_interface的接口类作为通信的方法，例如总线bus。

#### sc\_export

sc\_export是SystemC中的导出接口，它用于连接模块内部的子模块与外部模块之间的通信。

在SystemC中，一个模块内部可以包含多个子模块，每个子模块具有自己的接口和行为。通过使用sc\_export，可以将子模块的接口导出到模块的外部，使得外部模块可以连接到这些子模块并与其进行通信。

sc\_export可以看作是一种中介，它帮助建立了模块内部子模块的接口与模块外部的连接。可以将sc\_export看作是一个sc\_port，它的作用是扮演外部模块和子模块之间的接口，以便进行通信。

在使用sc\_export时，需要先在模块中定义并实例化子模块，并将子模块的接口与sc\_export进行连接。然后，在外部模块中，可以通过连接到sc\_export来访问和操作子模块。  
总的来说，sc\_export提供了一种在模块内部将子模块接口导出到模块外部的机制，用于建立模块内部和外部的通信连接。它可以使得模块的设计更加模块化和灵活，方便进行模块间的通信和互联。

sc\_export和sc\_signal是SystemC中的两个不同的概念，它们在功能和用法上有所不同。

1.功能区别：  
-sc\_export：sc\_export用于将模块内部的子模块接口导出到模块的外部，以便模块可以与外部模块进行通信。它可以看作是模块内部子模块与外部模块之间的连接接口。  
-sc\_signal：sc\_signal是SystemC中的信号类型，可以用于模块内的信号传输。它可以用于连接模块内的子模块或模块内的具体行为。sc\_signal可以在SystemC仿真中进行值的传递和通信。

2.用法区别：  
-sc\_export：在使用sc\_export时，需要先在模块内部定义并实例化子模块，然后通过连接子模块的接口和sc\_export来导出子模块的接口。在外部模块中，可以通过连接到sc\_export来访问和调用子模块的接口和行为。  
-sc\_signal：在使用sc\_signal时，需要先定义sc\_signal变量，并将其连接到模块内的各个子模块或信号源和接收端。sc\_signal可以通过连接到多个子模块的端口来进行信号传输和通信。

总结来说，sc\_export用于在模块内部将子模块的接口导出到模块外部，以便进行模块间的通信。而sc\_signal是一种用于模块内部的信号类型，用于连接模块内的子模块或信号源和接收端，实现模块内的信号传输和通信。

SC\_ONE\_OR\_MORE\_BOUND

SC\_ZERO\_OR\_MORE\_BOUND

#### sc\_channel

分层通道

分层通道跟sc\_module一个级别，可以跑单独的进程

#### sc\_prim\_channel

基础通道：sc\_signal

#### sc\_fifo

write()、nb\_write()、read()、nb\_read()

#### peq\_with\_get

模拟RTL中的pipeline等场景下，必不可少

peq是一个可以设置路径延时的管道，一个transaction在入口处设置好延时，延时一到就从出口处可以看到这个transaction了。

同一拍可以在peq的出口处get到多个transaction，相应transaction的时间一到，在出口处肯定可以看到，保证过程的延时正确，且不会漏掉transaction。

peq的源码路径在：tlm\_utils/peq\_with\_get.h

# TLM 2.0

## Transaction Level Modeling using OSCI TLM 2.0

### Introduction

业界一般使用事务级建模来解决电子系统设计、开发、部署期间的各种实际问题。

包括：

* 为软件开发提供一个早期平台
* 帮助软件/硬件集成
* 系统级设计架构分析(System Level Design architecture)
* 硬件功能性验证

随着抽象级别的提高，OSCI TLM 1.0标准变得不太适用，并且对任务来说不够速度不够快

影响当前方法的另一个问题是缺乏模型互操作性，不同的供应商创建的模型将由一个共同的客户使用；由后者来连接组件，以便它们能够相互交流。

TLM 2.0标准的主要目标是通过定义一组可靠的API和建议的数据结构，来解决这些问题。当按照建议使用时，可以实现模型互操作性。其意图是由不同的人编写的两个模型，在不了解对方的情况下，根据标准编写时，将是可交互操作的。

该API基于模板，提供了一个通用框架，以便在建议的数据结构被证明不适用于正在建模的协议时使用。在这种情况下，互操作性将受到影响，将需要适配器来与基于标准TLM 2.0的组件接口。

TLM 2.0标准分阶段实施，路线如下：

TLM 2.0：

* 用于事务执行的通用TLM APIs和数据结构。
* 可交互性的内存映射总线(API + data + protocol sematics)，适用于loosely-timed和approx-timed编码风格。
* 支持非侵入式(non-intrusive)/debug事务。
* 支持不显眼的监控或探查事务活动(analysis port)
* 关于通用数据类型的建议

TLM 2.0 stretch：

* Direct-memory interface
* Model synchronization

After TLM 2.0：

* PV 同步和中断
* 时间解耦
* Model-model 内存映射转移(memory map transfer)
* Language reference model
* Cycle accurate编码风格
* 更多通用的(SystemC/Component) debug APIs
* 包含内存映射的更多通用配置APIs
* Profiling APIs
* Hardware watchpoints
* Register/memories

## OSCI TLM 2.0 proposal

事务级模拟的不同使用模型需要不同级别的时序准确性和资源争用建模。举个例子，对于软件开发，可以忽略大多数（如果不是全部）资源争用问题，假设硬件会解决它。也就是说，软件采用的代码路径通常不依赖于CPU事务可能遇到的任何资源争用。

另一方面，硬件性能分析需要为感兴趣的组件（甚至可能是一些周围的组件）完全模拟争用。TLM 2.0为事务级建模定义了两种编码风格，取决于时序和数据模型的相关性(timing-to-data model dependency)

### API Architecture

TLM 2.0 API有以下部分构成：

* 基于user-defined模板的通用TLM API
* 推荐的Low-level 数据类型，适用于user-defined 模板
* TLM API的数据结构使其完全specialized，并为“通用”内存映射总线建模

通用内存映射总线可能对模拟一些“真实的”协议足够好，在loosely-timed functional level。某些其他的协议可能有一些功能性的细节不能被映射到建议的数据结构，在这些情况下，直接的可交互性将会被限制，bridges将会被需要。

### OSCI TLM 2.0 Model coding styles

根据模型必须遵守的时间-数据(timing-to-data dependency)依赖关系，模型可以分为两种编码风格。有时这些编码风格与“抽象层次”相混淆，但是所提出的编码风格既可以用于非常详细的模型，也可以用于几乎不包含建模组件细节的模型。

#### Loosely-timed models

松散时间模型有时也被称为PV模型，这些模型在timing和data之间包含loose dependency，并且能够在事务开始时提供timing信息和所请求的数据。

这种模型并不依赖于时间的推进来产生response。通常来说资源争用和仲裁不会使用这种风格来做建模。由于有限的依赖性和最小的上下文切换，这些模型可以运行得最快，并且对于在虚拟平台上进行软件开发特别有用。达到每秒5000万次交易的模拟速度使软件开发人员能够在几秒钟内启动操作系统并运行测试代码。

#### Approximately-timed models

近似时间模型在timing和data之间有更强的依赖性。这种模型在事务开始时不能提供timing信息以及所请求的数据。

这种模型在提供response之前可能依赖于内部/外部事件触发and/or时间推进。资源争用和仲裁可以很容易地用这种风格建模。

由于这种模型必须在处理事务之前对它们进行同步或排序，因此它们会被迫在模拟中触发多个上下文切换，从而导致性能损失。

#### Mixes of models

在某些情况下，需要创建能够以任何一种风格运行的模型，推迟决定使用哪一种作为运行时决策。当以这种混合风格编码时，总线和仲裁模型特别有用。

这使得能够以有限的精度高速执行模拟，同时允许用户切换到更精确的模式进行进一步检查。

为了实现模型的最大重用和功能硬件验证，能够混合两种风格的模型是有很必要的。

TLM 2.0通过跨编码风格共享API和数据结构实现了这一点。以一种风格编写的模型可以直接连接到以另一种风格编写的模型，而不需要适配器。

这种模型可以很容易地从松散时间逐渐细化到近似时间。

相反，我们也能够利用可用的近似时间模型进行软件开发。

### Compatibility with TLM 1.0

TLM 2.0标准兼容TLM 1.0

## TLM 2.0 Recommended Methodology

### Developing new components

在使用TLM 2.0开发组件时，推荐使用以下方法学：

* 从loosely-timed模型开始创建组件，尽管可能最后的模型会是approximately-timed。

包含：

1. Component functionality
2. Debug functionality
3. Memory map functionality

这个初始模型可用于启动目标软件开发、功能验证和系统模型集成。

* 在loosely-timed模型中添加timing细节，如果可能，对contention进行建模。也不是所有的模型都会需要被重定义为approximately-time风格，只有那些关键(interest)路径上的模型才需要。

### Adapting existing components

适配现有组件的复杂性主要依赖于原始组件的编码风格。常见的情况可能是cycle-driven模型。对于这些模型，需要编写一个bridge来接收传入的事务并将其传递给模型。bridge的一部分应该随着SystemC时间的推移为模型计时，并且在收到response时产生时间通知，并转换为TLM 2.0数据结构。

## Details and Areas under discussion

### Pass by pointer vs. pass by value

传输事务数据通过指针会最小化数据拷贝和最大化运行速度。事务数据应该通过指针传输，并且希望ownership来定义指针生存周期的规则。

### Direc Memory Interface

为模型提供对另一个模型的内存存储的直接访问绕过了函数调用，从而允许更快的模拟执行

### Temporal decoupling of modules

允许模型先行运行，然后等待系统跟上，从而利用主机缓存并最小化上下文切换

### On the fly switching to more accurate mode

为了以可接受的速度进行软件性能分析，TLM 2.0建议大多数需要approximately-timed风格的高流量组件采用“combination”风格编码。这将创建一个模拟，该模拟可以在不感兴趣的部分（例如目标软件解压缩）期间快速执行，然后切换到更准确的模式（例如驱动程序fedding硬件加速器）。

### Model synchronization

当处理松散时间系统时，有必要强制模型同步以使模型以可预测的方式推进时间。TLM 2.0提供了模型同步的方法。

### Blocking in the master vs. Blocking in the slave

Approx-timed模型无法在启动时完成事务，因为它们依赖于系统的其他部分来完成事务。模型中的事务调用应该是阻塞的还是非阻塞的，或者两种风格都支持，还没有决定。

### Extension mechanisms

尽管人们一致认为对数据结构的扩展会增加标准的灵活性，但似乎没有实用的扩展机制可以处理可能需要的可选和必备扩展属性的混合。

### Mmoments in time

六个“moments in time”似乎足以代表大多数协议的阶段。然而，这些“moments”的交流方式尚未决定。

## Summary

TLM 2.0为创建模型和事务级建模提供了一种标准化的方法。该标准支持模型交换和公共的基础接口。

一个简单而坚实的架构使TLM的新来者能够快速上手并开发可互操作的模型。对于经验丰富的TLMers来说，该标准为日益棘手的模型互操作性问题提供了解决方案