**ESL设计，即电子系统级（Electronic System Level）设计方法学。ESL设计可以为芯片设计工程师提供开发、优化、验证芯片架构与嵌入软件的方法，以及提供验证服务。**

# 前言

随着半导体行业的高速发展，ESL(Electronic System Level)建模或芯片建模在大算力高性能芯片领域显得越来越重要，国内外EDA或芯片设计公司都在构建自己的芯片评估工具或平台，以应对复杂芯片设计所带来的挑战。希望通过这篇文章让大家对ESL建模以及相关的基本概念和设计方法有一定的认识，了解ESL建模的方法和流程，理解工具并利用工具创造价值。本文主要介绍ESL相关的基础知识、概念、术语、以及ESL发展历史。基于这些新的理念和方法，ESL建模得到了长足发展并且逐渐趋于完善。与其它技术类似，ELS或芯片建模采用的工具平台也包括商业和开源两大阵营。此外，本文还介绍了芯片建模标准规范，IEEE1666-2011 SystemC+TLM2.0。在此基础上，还介绍了芯片建模的主要应用领域，有些应用比较成熟，比如性能模型(架构探索/评估/分析)，功能模型(软件提前开发)，功耗模型(功耗评估)，混合仿真(功能或性能模型)，软硬件协同设计，验证与划分等。有些应用比较新颖，比如高阶综合(HLS)，主流的EDA供应商已经发布了相应的工具，由于应用领域局限性而尚未得到广泛应用。限于篇幅，本文只涉及性能模型和功能模型两个应用方向，其它方向的应用后续有机会再作介绍。在本文中，ESL、ESL建模、芯片建模，名称不同但是含义相当。

# 术语

|  |  |  |
| --- | --- | --- |
| **简称** | **全称** | **注释** |
| ESL | Electronic System Level |  |
| VP | Virtual Prototyping/Platform | 虚拟原型/平台 |
| AT | Approximately-timed model | 时序近似模型，用于性能模型 |
| LT | Loosely-timed model | 无时序模型，用于功能模型 |
| TLM | Transaction Level Modeling | TLM2.0 |
| VDK | Virtualizer Development Kits | Synopsys |
| CA | Cycle Accuracy | 周期精度 |
| PV | Programmer View model |  |
| PEQ | Payload Event Queue | SystemC+TLM2.0 |
| FM | FastModel | arm |
| LISA | Language for Instruction Set Architectures | 指令集架构编程语言 |
| OSCI | Open SystemC Initiative | 开源组织 |
| PRD | Product  Requirement Document |  |
| NoC | Network on Chip |  |
| RTL | Register Transfer Level |  |
| AI | Artificial  Intelligence |  |
| GPU | Graphic  Processing Unit |  |
| NPU | Neural-network  Processing Unit |  |
| HLS | High-Level Synthesis |  |
| SoC | System on Chip |  |
| PCB | Printed Circuit Board |  |
| DDR | Double Data Rate |  |
| DMI | Direct Memory Interface |  |
| AMBA | Advanced Microcontroller Bus Architecture | Arm 总线协议 |
| AXI | Advanced eXtensible Interface | Arm AMBA |
| CHI | Coherent Hub Interface | Arm AMBA |

# ESL发展简介

早在20年前，随着芯片行业的发展，芯片结构越来越复杂，版本迭代越来越快速，应用场景越来越多样化，这些变化给芯片设计带来了巨大的挑战。人们注意到传统的设计方法已经不能再满足芯片设计需求，亟需一种全新的设计方法和理念，既能处理复杂的内部结构，又能应对功能的快速迭代，由此ESL应运而生。2001年，ESL率先由Gartner Dataquest定义和发布，并被大众所熟知，至今，ESL已经在芯片设计、验证以及仿真领域得到广泛应用。如果深入研究ESL的本质，它不是某种具体的设计语言、工具或平台，而是一种通用的设计和验证方法学，根据芯片各阶段的设计要求，在规格定义阶段(前期)、中期、后期利用这种方法对芯片的性能，功能，功耗进行建模，通过仿真得到某些参数或指标，作为芯片设计和优化参考。  
ESL主要应用领域：

* 性能/架构模型(CA/AT)：架构探索，评估与优化，架构设计阶段
* 功能模型(TL)：软件提前开发，软硬件并行开发，缩短研发周期
* 功耗模型：功耗评估
* 系统设计(System Level Design)：软硬件协同设计、验证与划分
* Co-simulation(Model+RTL)混合仿真：包括功能和性能仿真
* 高阶综合(HLS)：使用工具将C++/SystemC+TLM2.0模型直接综合成RTL代码

ESL之所以能应用于不同领域，是因为采用了高级编程语言，建模灵活，快速迭代，可以实现不同抽象层次(Abstraction Level)的模型。如果非要从学术层面来定义ESL，可以理解为“利用适当的抽象来增加对系统的理解，并以具有成本效益的方式提高成功实现功能的可能性”。这里面涉及系统、抽象、软硬协同设计和验证以及功能实现等。业界根据这些设计方法和思想，经过多年发展，已经形成了开源和商用两大阵营，以下是业界具有代表性的仿真工具与平台。

* 开源

**·**SystemC+TLM2.0

**·**QEMU

**·**GEM5

**·**……

* 商用

**·**FastModel, arm

**·**VDK, Synopsys

**·**Simics, Wind River/Intel

**·**……

除了以上列出来的仿真工具和平台之外，当然还包括其它开源和商用仿真工具，不胜枚举，详情可以参考[15]。由于商用工具需要购买license，有些公司与其IP深度绑定，形成了完整的生态链，处于垄断地位，授权价格昂贵且授权方式完全由供应商主导，对国内芯片公司不是特别友好。因此国内大多数公司把目标都转向开源仿真工具，开源工具中使用比较广泛的是GEM5和QEMU，两者都提供了丰富的模型和完整的软件，但他们之间也存在差异。GEM5和QEMU都有完整的处理器，总线和外设功能模型，可以快速搭建用户需要的模型，另外，还提供相应的驱动软件和系统软件包，可以更快的实现系统boot功能，用户只需要关注软件开发工作，这些只是功能模型方向的应用。另外GEM5还提供了业界主流架构的处理器(core)模型，如x86/arm/risc-v等，总线(NoC)等架构模型，可以进行性能仿真和评估。在学术界也非常流行，使用GEM5进行计算机或芯片架构探索，性能评估等，基于GEM5仿真所发表的学术论文特别多。对于多数自研IP的公司来说，比如AI/GPU/NPU等，通常还是基于C++/SystemC+TLM2.0建模，根据自研芯片规格来实现自定义架构模型，也会借用GEM5里面现成的模型。顺便提一下，SystemC+TLM2.0之所以重要，是因为商用仿真平台多数都是参考或基于SystemC+TLM2.0开发。比如大家所熟知的Synopsys VDK仿真平台，就是在开源SystemC+TLM2.0基础上，增加和增强了很多新的功能，以图形界面方式呈现，专门为用户提供了二次开发的接口和方法，究其最低层技术还是基于SystemC+TLM2.0标准实现的，另外Candence Xcelium，一种综合逻辑仿真器，也支持SystemC+TLM2.0模型仿真，SystemC+TLM2.0是芯片建模和仿真的重要基础。

# IEEE1666-2011标准

为了能够更全面地认识SystemC+TLM2.0, 我们可以从产生背景、编程语言、参考仿真器等多个方面来了解。根据前面介绍，早在2000年左右，人们就提出了抽象层次设计概念以应对复杂芯片设计挑战，但是当时并没有统一的实现语言和工具，比如LISA/LISA+早在90年代就被提出了，是最早架构建模语言之一，迄今为止，在arm的FastModel里面仍然可以见到其身影，除了FastModel之外业界很少直接使用LISA/LISA+语言建模。直到2004年SystemC诞生，架构建模语言逐渐趋于收敛和统一，2011年OSCI发了SystemC 2.3.0，目前最新版本为2.3.4，随后被纳入IEEE标准，并正式发布IEEE1666-2011标准版本。直到今天，还是由OSCI开发和维护，同时OSCI隶属于标准组织Accellera并成为该组织下面众多技术标准之一。首先，SystemC+TLM2.0是全面兼容C/C++的类库(class library)，支持C++11/14特征(Feature)，所以可以理解为SystemC+TLM2.0是基于C/C++的建模语言(Modeling language)，作为一门编程语言，它一直都在不断升级和发展，根据官方资料显示，主要朝着多线程(Multithreading)、功能安全(Functional Safety)、错误注入(Error Injection等方向进行发展和演变。其次，SystemC+TLM2.0还是一种开源参考仿真器，可以通过离散事件实现进程之间通信和同步，在这里，进程可以理解为模块里面具备独立功能的逻辑电路，使用进程旨在描述独立功能的逻辑电路的“并行性”，如果从这个角度来讲，SystemC+TLM2.0编程具有协同程序(Co-routines)的思想，因此在建模的时候，总是要考虑进程(逻辑电路)之间是顺序关系还是并行关系。通过几张图(均来自IEEE1666-2011)介绍SystemC+TLM2.0的主要内容和构成。图1展示了SystemC Language Architecture，最底层是标准C++编程语言，在此之上包括两大块核心内容Core Language 和 Data Type。核心语言部分主要包括module(sc\_module), port(sc\_port/sc\_export), interface(sc\_interface), channel(sc\_channel),clock(sc\_clock), 调度器(scheduler)，进程(thread/SC\_THREA/SC\_METHOD)，括号里面是相应的类名、函数或宏名。Data Type用于描述硬件中的数据位宽，逻辑状态(高电平，低电平，高阻态，不确定态)，以及数据和状态之间的操作等。在两大核心的基础上还包括进程之间的控制和交互使用的接口(Interface)和通道(Channel)，如signal、buffer、mutex、semaphore等。具备这些建模的基本要素之后就可以根据芯片规格(Specification)对其建模了。文本

描述已自动生成

图1. SystemC Language Architecture (IEEE1666-2011)

如果只用SystemC提供的组件来建模，开发者不仅需要关注模块内部结构和功能实现，还需要关注模块之间的通信功能实现。这对开发者来说，无疑是增加了开发负担，也不利于开发更高抽象层次的模型开发，因为SystemC里面模块之间通信以channel为主，比如signal/fifo，这种方式虽然可以描述更加精准的硬件行为(CA)，但是仿真效率低，对于芯片架构探索或开发前期来说并不适用。为了解决不能开发任意抽象层次模型和仿真效率低两大问题，OSCI还推出了TLM(Transaction Level Modeling)。在OSCI官方文档中对SystemC+TLM2.0最常用的描述为SystemC with TLM2.0，SystemC/TLM2.0或SystemC+TLM2.0。SystemC+TLM2.0实际上包括两块独立功能的内容，建模时会同时使用。参考图2，SystemC又是TLM2.0的基础，包含TLM1.0和TLM2.0，两个版本在实现方式和细节上有所不同，TLM1.0使用相对较少，本文不做介绍。由于TLM2.0设计更具有通用性，获得了广泛应用，TLM2.0不仅提供blocking/non-block/Debug/DMI transport接口，还实现了通用数据类型(generic payload)和通用的相位(phase)，由这两个通用模块构成基础协议(Base protocol)。接口和基础协议组成通用层(Interoperability layer)，通用层除了极大地方便建模外，还让不同公司开发的模型可以集成到同一个系统中。除了通用层之外，TLM2.0还提供了utilities class，可以作为golden simple，包括socket/callback register/blocking/ non-blocking transport等。其中simple\_initiator\_socket/ simple\_target\_socket极为常用。这些接口效率高且使用方便，应用非常广泛，对于大多数建模场景而言，使用自带utilities class基本满足要求，当然如果有特殊需求，例如增加时序或时钟信息，或对新的通信协议进行建模，可以在通用层的基础上开发全新的socket，比如AMBA协议簇里面的AXI /CHI等。图形用户界面, 文本, 应用程序

描述已自动生成  
图2. TLM2.0 Class (IEEE1666-2011)

根据前文对SystemC+TLM2.0的介绍，现在可以研究一下SystemC+TLM2.0建模抽象层次与具体应用之间的关系。图3展示了建模方式(Coding Styles)以及抽象层次，应用场景可分为四类，即软件开发、软件性能分析、架构性能分析、硬件验证。可以根据仿真需求选择相应的coding styles和TLM2.0通用层来实现，比如transport/ socket等。建模抽象层次应根据要求而定，精度和抽象层次更是需要权衡考虑(Trade-off)，同时还需要兼顾仿真效率。总的来说，对精度要求更高，则对建模抽象层次要求也更低，其仿真效率就越低(仿真慢)，反之亦然。除了SytemC+TLM2.0存在这个客观局限之外，其它EDA工具也类似，纵观芯片设计链路中的EDA工具中，没有任何一个EDA工具可以全部覆盖设计和验证功能，都会存在自身的优势和劣势。即便如此，也可以使用不完美的工具得出相对满意的结果，如何充分利用好SystemC+TLM2.0需要综合考虑模型精度和仿真效率。以模型对时钟或时序要求为例，每一种应用场景对时钟或时序要求差异很大，硬件验证则需要确保时钟周期完全一致，架构探索和软件性能分析则只需要时钟或时序近似，软件开发则完全不需要考虑时序信息，只要保证数据正确即可。图形用户界面

中度可信度描述已自动生成图3. Use cases, coding styles, and mechanism(IEEE1666-2011)

# 性能模型

图4是简化版本的芯片设计流程，忽略其它步骤和细节，对于前端和后端而言，EDA公司可以提供明确的测试与验证平台(工具)，包括方法和流程。但是对于架构设计来说，如何验证，使用什么工具或平台呢？假如芯片架构师收到了来自市场或客户的产品需求文档(Product Requirement Document, PRD)，需要根据产品需求文档量化芯片的参数和指标，即定义芯片规格，确定芯片里面某些子模块的具体参数和行为，比如L1/L2/L3 cache size，总线带宽，关键模块之间的时延，内存架构等，芯片研发初期，此时RTL还未开始开发，更是没有任何平台配合应用场景来验证参数的正确性和合理性，因此如何验证架构设计成为芯片挑战之一。

特别是AI/GPU/NPU等类似专用的芯片，对架构验证平台的需求尤为突出。架构模型或性能模型成为验证架构设计的主要工具或平台，在规格定义阶段并可以开始设计架构模型，迭代，测试和优化，将结果反馈给芯片架构师，不断精细化芯片参数，架构或性能模型应当具备以下几点要求：

* 模型精确性，抽象层次，可定量或定性分析
* 模型迭代，参数可配置，可快速迭代，可探索全新的架构(下一代)

图示

描述已自动生成图4. 芯片设计流程(简化)参考图4，芯片架构验证就是建模、测试、校验和优化的过程。尽管芯片架构验证处于芯片设计初期，实际上这一过程伴随芯片设计整个周期。  
架构探索可以分为单元架构(处理器微架构/存储架构)和系统架构，单元架构模型更聚焦于内部实现细节和交互信息，比如处理器5级流水，取指、解码、执行、访存、写回。而系统架构则侧重于整体性能评估，将应用场景实例化之后，直接加载到架构模型上运行，通过收集数据处理信息，比如Latency/Bandwidth/Efficiency/Event/Timing等等，定量分析架构的性能和参数。抽象层次其实是架构模型在建模之前需要考虑的重点。从周期精度模型(CA model)，时间近似模型(AT model)到行为模型(LT/Behavior Model)，需要根据评估目标而定，无关性能的模型，可以采用行为模型，需要关注性能的模型可以使用周期精度或时间近似模型，将这些不同抽象层次的模型集成到同一系统，还能得到混合(Hybrid)模型，该模型可以满足某些特殊场景的测试和验证需求。图5展示了建模与仿真流程，以及与架构设计之间的关系。无论是软件还是硬件设计，都有一种设计理念，称为可执行规格(Executable Specifications)，对于芯片设计而言，其意是指架构设计、架构建模和架构验证过程。可以直接将架构规格说明转化为可执行程序，定量分析架构设计。**图示

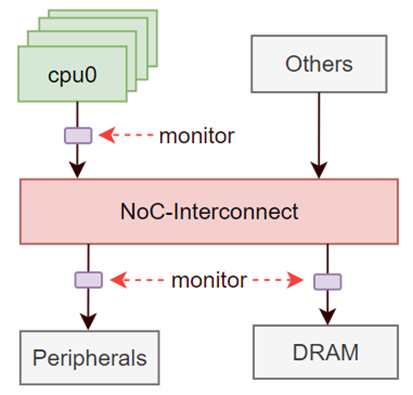
描述已自动生成**图5. 建模和仿真流程架构建模包含的内容很多，流程虽然简单，但是整个建模和仿真过程，可以粗略也可以精细，灵活多变，结果呈现方式和分析方法也是多种多样。经过仿真除了可以得到带宽(Bandwidth)、时延(Latency/Delay)、波形(Waveform)等最基本的数据之外，还可以获得状态、事件统计等信息。架构设计和架构建模互为补充，伴随参数调整、仿真迭代、仿真优化和结果分析等过程，直到最终得到满足应用要求的参数和结果。

图6. 简单的NoC参数和性能分析用例

图6给出了一个简单的NoC性能分析用例，在NoC的上行和下行接口上增加监测(Monitor)，收集数据传输过程的参数。假设总线协议是AXI，CPU发送的数据包来自某个应用场景。可以调整BL(Burst Length)长度、Burst Size、Burst Type等以适配不同的NoC配置，直到满足要求的结果。

# 功能模型

我们知道任何产品都有窗口期，每家公司都希望自己的产品尽快上市(Time-to-Market)，更快更多地抢占市场。为了最大化地缩短开发和生产周期，每个行业都在使用新进的工具来进行并行开发。芯片设计也不例外，如果简单地将芯片设计划分为硬件和软件两部分，传统的设计流程是先硬件后软件，以串行方式进行。

因为软件开发依赖于硬件平台，为了实现硬件与软件并行开发，功能模型使软硬件并行开发成为可能。功能模型(Function model)也称为虚拟原型(Virtual Prototyping VP)，虚拟平台(Virtual Platform VP)，它们的本质都是一样，如果说非要有些区别，虚拟原型更倾向于片上系统(System on Chip, SoC)，虚拟平台涉及更大的系统，除了包括SoC之外，还包括其它子模型如NoC，DDR，Peripherals等，在这里，可以理解为板级(Printed Circuit Board，PCB)虚拟系统。软件设计左移(Shift-left)在芯片设计领域越来越被重视，左移就是为了实现软硬件并行开发。参考图7，如果将芯片设计简单分为架构，前端和后端设计三个阶段，使用传统设计流程，软件需要在RTL开发快结束的时候，才能在FPGA/Emulator平台开始软件开发和调试，参考图7中Emulation starting point。如果采用功能模型或虚拟原型(VP)，在时间上可以提前至少8~12个月，给软件团队足够的时间来开发和优化软件，利用功能模型，可以开发firmware/driver/API/boot flow/Linux/software stack等，如果功能模型增加了时序和追踪(Trace)功能，还可以运行应用程序，用来分析软件的性能。通过两种方式对比，在时间上有绝对的优势，因此获得了广泛应用，比如VDK、Simics、利用SystemC+TLM2.0开发的功能模型等。**日程表

描述已自动生成**图7. 芯片研发周期示意图

与性能模型相比，功能模型是程序员视角模型(Programmer View, PV)，只要保证寄存器精度和数据传输正确即可，无需关注时序以及链路上数据的具体传输过程。如果有额外的需求，比如分析软件性能，可以在模型里面增加时序(Timing)、事件(Event)等信息，建模非常灵活，完全可以根据需求增加相应的特征(Feature)或功能。图示

描述已自动生成图8. 功能模型内部结构

由于功能模型不需要考虑时序和传输过程等因素，与性能模型比起来，相对简单很多，参考图8，典型的功能模型用例主要包括输入/输出接口，数据队列，控制逻辑，回调函数，寄存器，状态信息，中断信号等等。针对无时序模型SystemC+TLM2.0专门提供了blocking和DMI接口，即b\_transport和get\_direct\_mem\_ptr，通过调用这两个接口函数可以直接返回结果，调用接口函数过程中不会被其它模块或进程打断。此外，SystemC+TLM2.0还提供了时间解耦(Temporal Decoupling)，仿真时钟精度(Time Resolution)，全局量子时间(Global Quantum)等特征，充分利用non-blocking/DMI接口和这些特征可以巨大地提升功能模型的仿真效率。

# 总结

首先，本文介绍了ESL发展历史，以及商用和开源两大阵营中主流仿真工具和平台。其次，还介绍了SystemC+TLM2.0标准规范，最后介绍了两个主要应用场景：性能模型(架构模型)和功能模型(虚拟原型)。尽管可以使用SystemC+TLM2.0创建任意抽象层次的模型，但是也存在不足之处，目前SystemC+TLM2.0只支持单线程，正是这个原因导致复杂的或抽象层次较低的模型仿真很慢(仿真效率低)，从官方渠道可以得知，SystemC+TLM2.0正在朝着多线程、功能安全、错误注入等方向发展和演变，以解决仿真效率问题和拓展更多的应用领域。