手把手课堂：FPGA 101

用 Vivado HLS 为软件提速

任何为代码瓶颈而苦恼的人

都应探索高层次综合工具与

Zynq SoC 的组合出击。

作者：David C. Black
Doulos 高级技术人员
david.black@doulos.com

在编写软件时，您有没有遇到过无论怎么努力编码，软件都不能按您期望的速度运行？我遇到过。您有没有想过，“有没有什么简单而且成本不高的方法可将一些代码输入多个定制处理器或定制硬件？”毕竟，您的应用只是众多应用中的一个，而且创建定制硬件需要花费时间和成本。是不是这样？

最近听说了赛灵思的高层次综合工具 Vivado®HLS 后，我开始重新思考这一问题。高层次综合工具与 Zynq®-7000 All Programmable SoC 的结合为设计开辟了新的可能性，其中 Zynq®-7000 All Programmable SoC 结合了带有 FPGA 架构的双核 ARM®Cortex™-A9 处理器。这类工具可以用 C 语言，C++语言或 SystemC 源代码创建高度优化的 RTL。近年来，出现很多这项技术的提供商，且其采用率也不断提高。如果我只用 Vivado HLS 便能完成要求更高的计算，那么将那些慢速代码迁移到硬件中会有多难？毕竟我经常用 C++语言编写代码，而 Vivado HLS 将 C/C++语言作为输入语言。ARM 处理器内核意味着我可以在常规环境下运行多数软件。事实上赛灵思还提供了一款软件开发工具（SDK）以及 PetaLinux 来帮您实现这一目的。

架构问题

从软件角度思考这一转变，我开始更加担心软件接口问题。毕竟，HLS 创建的硬件专注于处理硬件接口。我需要一些易于访问的工具（如协处理器或硬件加速器）来加快软件运行速度。而且，我不想编写新的编译器。为了方便与软件的其它部分交换数据，这个接口应该类似于简单的存储单元，我们可以在其中输入信息并稍后读取结果。

然后我有了新的发现。Vivado HLS 支持以相对较小的努力轻松创建 AXI 从接口。这让我开始思考，创建加速器也许没有那么难。于是，我编码了一个简单的实例来探索这种可能性。探索的结果让我惊喜不已。

下面看看我用了什么方法，并思考这种方法所得出的结果。

在我的实例中，我选择了对一系列简单的矩阵运算（如加法和乘法）进行建模。我不想将它限制在固定的大小，因此，我必须同时提供输入阵列及各阵列的尺寸大小。理想的接口会将所有数值作为函数的自变量，例如图 1 中的代码。

硬件接口需要用一种简单的方法将函数自变量映射到存储单元。图 2 是支持这一映射的存储器配置。寄存器中保存了有关矩阵的排列方式以及所需运算的信息。*指令*寄存器将指示执行何种运算。这样我便可以将一些简单的运算融合到一个硬件中。可以用*状态*寄存器来查看是否正在进行运

算或是已经成功完成运算。此外，器件最好还能提供中断支持。

回到硬件设计，我了解到 Vivado HLS 为阵列自变量留出空间以指定小容量内存。这样，图 3 所示函数便说明了这种函数性。

假设能够综合 AXI 从接口，怎样将它用在软件上？我将正常编码环境假设为 Linux。还好赛灵思提供 PetaLinux，而且 PetaLinux 提供一种叫做用户 I/O 器件的机制。UIO 可以用简单的方法将新硬件映射到用户内存空间，并具备中断等待能力。这意味着您可以省去编写器件驱动程序所耗费的的时间和流程。图 4 显示了这个系统。

这种方式当然也有缺陷。例如，无法在 DMA 中使用 UIO 器件，因此您必

图 1 – 调用加速器示例

地址	寄存器名称	目录	位元	内容	
0	Matrix0_ptr	RW	32	Matrix 0 数据地址	
4	Matrix0_shape	RW	32	Matrix 0 行	Matrix 0 列
8	Matrix1_ptr	RW	32	Matrix 1 数据地址	
12	Matrix1_shape	RW	32	Matrix 1 行	Matrix 1 列
16	Matrix2-ptr	RW	32	Matrix 2 数据地址	
20	Matrix2_shape	RW	32	Matrix 2 行	Matrix 2 列
24	Matrix3_ptr	RW	32	Matrix 3 数据地址	
28	Matrix3_shape	RW	32	Matrix 3 行	Matrix 3 列
32	-reserved-	-	32		
36	-reserved-	-	32		
40	Command	RW	32	0	enum
44	Status	RW	32	0	enum

图 2 – 寄存器汇总表

图 3 – 加速器函数 API

:内存中构建矩阵，并在构建完成后手动复制出来。如果需要，将来可以通过定制器件驱动程序解决这个问题。

用 Vivado HLS 综合硬件

现在回到综合 AXI 从接口的话题。它的综合难度有多大？我发现这些编码限制非常合理。除内存的动态分配以外，大多数 C ++语言都可以使用。

毕竟硬件在运行过程中不能生产其本身。这限制了标准模板库（STL）功能的使用，因为这里大量使用了动态分配。只要数据保持静态，多数功能都可以使用。起初这项任务似乎非常繁重，但我发现这并不是什么大事。另外，Vivado HLS 允许 C++ 类、模板、函数和运算符重载。我的矩阵运算可轻易归入定制矩阵分类。

增加 I/O 来创建 AXI 从接口非常简单。只需增加一些能

够指示包含哪些端口以及　　即可。
使用哪些协议的编译指示

只要我不按下所有旋钮，运行这款综合工具非常简单。

只要我不按下所有旋钮，运行这款综合工具非常简单。图 5 展示了其中各个步骤，在此我不再详细解释。需要就目标技术和时钟速度对 Vivado HLS 进行一些引导。之后涉及的程序会密切关注违反政策的报告并研究分析报告以确保 Vivado HLS 按我所期望的方式运行。工具用户必须对硬件方面有所了解，但有一些技术课程可以解决这个问题。还存在综合前后运行仿真以检验预期行为的问题。

Vivado IP Integrator 让 AXI 从接口连接到 Zynq SoC 硬件变得轻而易举，并打消了对信号连接错误的顾虑。赛灵思甚至拥有我所使用的 ZedBoard 开发系统的系统概述，并用 IP Integrator 导出数据用于软件开发套件。

清除瓶颈

我对结果非常满意，我希望能用这款芯片与工具集的组合做更多事情。我并没有探索所有的可能性。例如，Vivado HLS 还支持 AXI 主接口。AXI 会允许加速器从外部存储器复制矩阵（尽管这样也可能存在安全问题）。不过我强烈建议所有面临代码瓶颈的人都能考虑这个工具集。这里提供足够的培训课程、资源和材料以实现快速匀变，其中包括 Doulos 提供的课程、资源和材料。如需了解更多信息，敬请访问：www.doulos.com。

图 4 – 系统框图

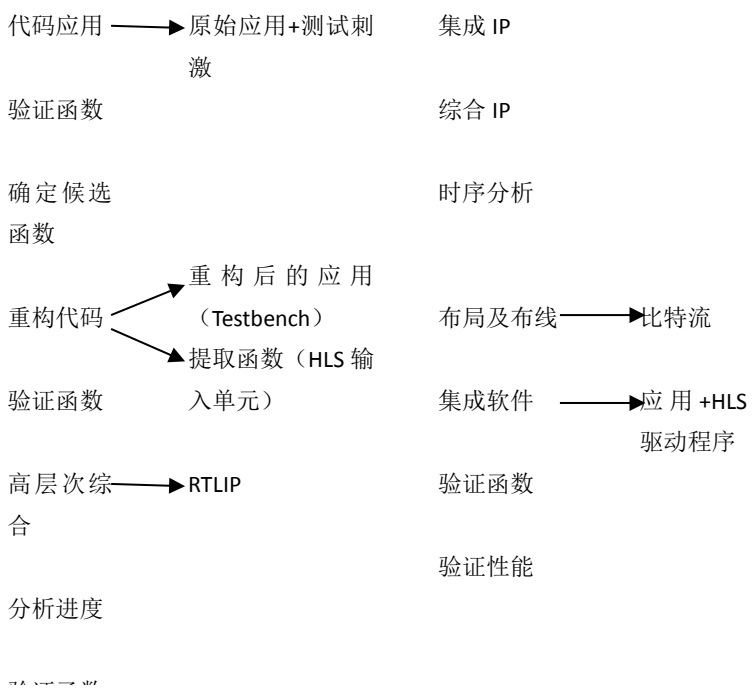

图 5 – 设计流程图