在 FPGA 中实施无线 MIMO 球形检测器

利用 AutoESL 高级综合工具可实现在 Xilinx Virtex-5 器件中构建复杂的宽带无线系统接收器。

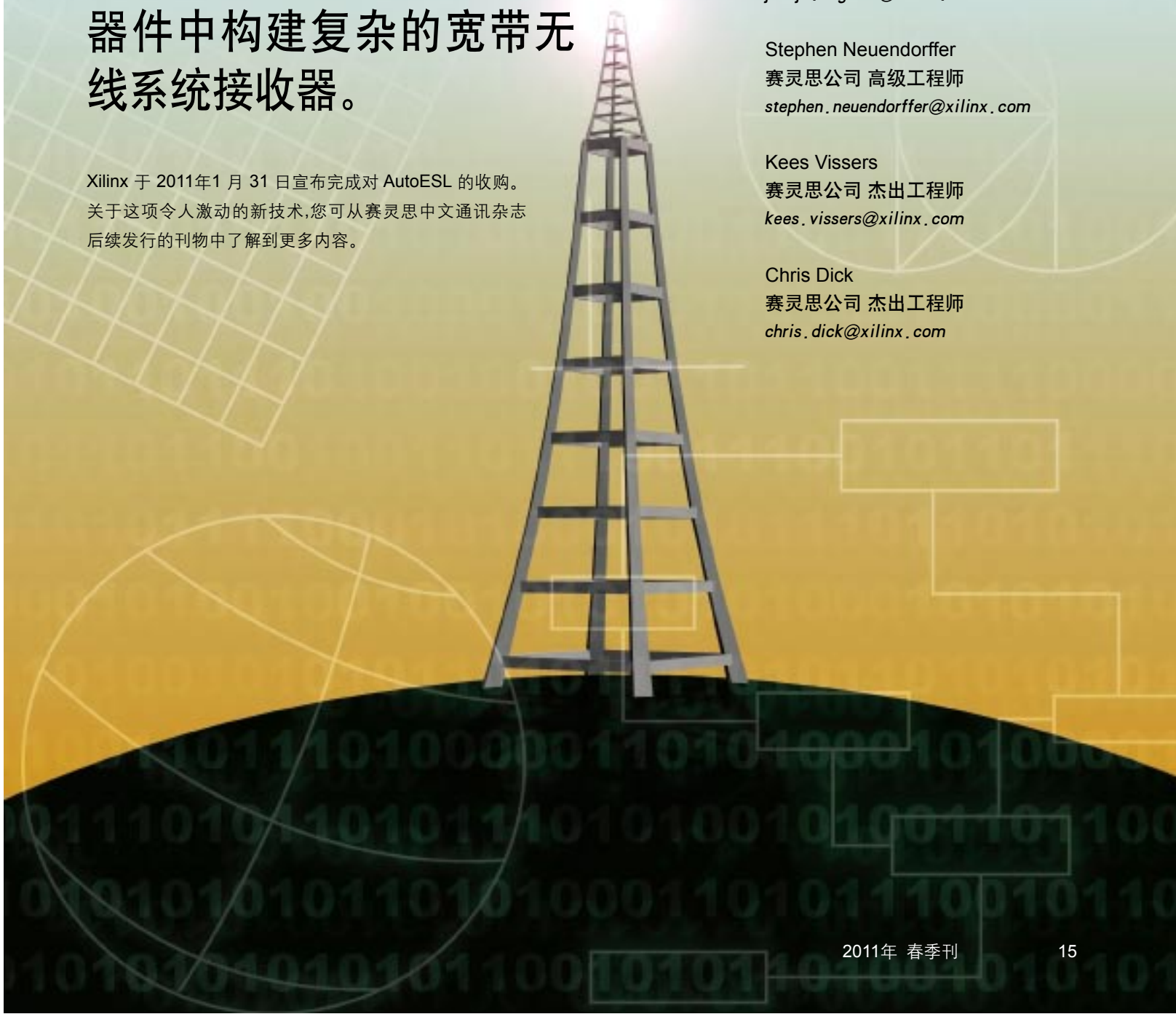
Xilinx 于 2011年1 月 31 日宣布完成对 AutoESL 的收购。关于这项令人激动的新技术,您可从赛灵思中文通讯杂志后续发行的刊物中了解到更多内容。

作者:
Juanjo Noguera
赛灵思公司 高级工程师
juanjo.noguera@xilinx.com

Stephen Neuendorffer
赛灵思公司 高级工程师
stephen.neuendorffer@xilinx.com

Kees Vissers
赛灵思公司 杰出工程师
kees.vissers@xilinx.com

Chris Dick
赛灵思公司 杰出工程师
chris.dick@xilinx.com



空分复用 MIMO 处理技术可显著提高无线通信系统的频谱效率，进而大幅增加无线通信系统的容量。正因如此，它已成为新一代 WiMAX 以及其它基于 OFDM 无线通信系统的核心组成部分。空分复用 MIMO 处理技术是一项计算密集型应用，可实现高要求的信号处理算法。

在 MIMO 系统中，空分复用技术的一个具体实例是球形解码。球形解码是一种解决 MIMO 检测问题的有效方法，其在比特误码率 (BER) 性能方面能与最佳的最大似然检测算法相媲美。但是，DSP 处理器的计算能力有限，不足以满足球形解码实时性方面的要求。

现场可编程门阵列 (FPGA) 是一个极具吸引力的平台，可实现如球形解码器这样的复杂 DSP 密集型算法。现代 FPGA 是一种高性能并行计算平台，可在保持可编程 DSP 处理器灵活性的同时，为系统提供所需的专用硬件。多项研究表明在多个信号处理应用中，FPGA 的性能比传统 DSP 处理器高 100 倍，性价比可提高 30 倍。

尽管 FPGA 的性能具有相当大的优势，但通常无法应用于无线信号处理，主要因为传统的 DSP 程序员认为它们不容易处理。事实上，造成 FPGA 无法在无线应用中得到广泛使用的真正阻碍是以硬件为中心的传统设计流程与工具。目前，要想利用 FPGA 来进行设计，需要具备丰富的硬件设计经验，包括熟悉 VHDL 与 Verilog 等硬件描述语言。

最近，新型高级综合工具可以作为 FPGA 的辅助设计工具。这些设计工具将高级算法描述作为输入，并生成可与标准 FPGA 实现工具（例如 Xilinx® ISE® design suite 与嵌入式开发套件）一起使用的 RTL。该工具可提高设计效率，缩短开发时间，同时实现高质量的设计。我们可利用该工具设计基于 FPGA 的复杂无线算法应用，即802.16e 系统下的空分复用 MIMO 球形检测器。我们专门选择 AutoESL 的 AutoPilot 高级综合工具作为时钟频率为 225MHz 的 Xilinx Virtex®-5 的辅助设计工具。

球形解码

球形检测作为解码过程的一部分，是一种用于简化空分复用系统检测复杂程度的有效方法，在 BER 性能方面可与复杂度更高的最佳最大似然 (ML) 检测算法相媲美。

如图 1 所示，在 MIMO 802.16e无线接收器的方框图中，我们假设接收器可以准确估计信道矩阵，该假设条件可通过传统的信道估算方法来实现。该实现的流水线具有 3 个构建模块：信道重新排序、QR分解以及球形检测器 (SD)。我们通过计算检测到的比特对数似然比 (LLR) 来产生软输出，从而为软输入、软输出信道解码器（比如涡轮解码器）的使用做准备。

信道矩阵重排序

球形检测器处理天线的顺序可对 BER 的性能产生较大影响。在进行球形检测之前，首先执行信道重排序。检测器利用信道矩阵预处理器实现了类似贝尔实验室分层空时 (BLAST) 结构上采用的连续干扰抵消处理技术，最终达到了接近 ML 性能。

该方法利用信道重排序处理来实现，通过多次迭代确定复杂信道矩阵最佳列检测次序。该算法根据迭代次数来选择范数最大或最小的行。欧几里德范数最小的行表示天线影响最强，而欧几里德范数最大的行表示天线影响最弱。这种新颖的方案首先处理最弱的数据流，随后依次迭代处理功率从高到低的数据流。

为了满足应用对高数据速率的要求，我们实现了图 2 所示的基于流水线架构的信道排序模块。该模块可以在时分复用 (TDM) 模式下同时处理 5 条信道。这种方案延长了同一信道不同矩阵元素之间的处理时间，同时可保持高数据吞吐量。

在图 2 中，G 矩阵计算是要求最高的组成部分。该过程的核心是矩阵求逆，可通过 QR 分解 (QRD) 来实现。实现 QRD 的常用方法是使用吉文斯旋转。该方法可执行对角线单元和非对角线单元的复杂旋转，这些单元都是我们所使用脉动阵列的基本计算单元。

改进的实数矩阵 QRD

在获取信道矩阵列的最佳排序之后，下一步应对实数矩阵系数进行 QR 分解。用于该 QRD 处理的功能单元与计算逆矩阵的 QRD 引擎类似，但有一些不同之处。本例子的输入数据是实数，因此脉动阵列结构的维数会相应更高（即 8×8 实数值而不是 4×4 复数值）。

为了满足时序约束的要求，输入数据消耗速率必须保持在每时钟周期 1 个输入采样。这种要求给处理时延带来了挑战，即我们无法在 5 条信道的 TDM 结

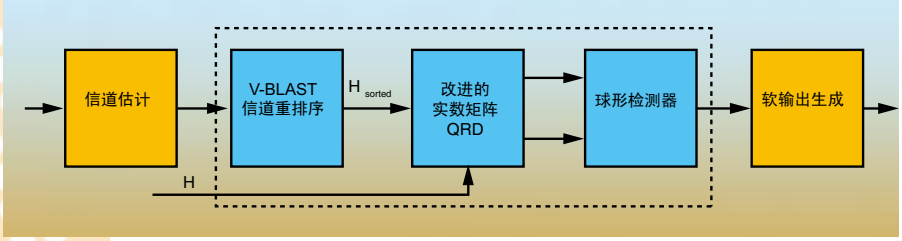


图 1 球形解码器方框图

构中处理该问题。因此，在 TDM 组中将信道数量增加至 15 条，以延长同一信道矩阵中连续元素之间的处理时间。

球形检测器设计

您可以将迭代球形检测算法视为遍历树过程，树 i 的每一层对应于第 i 个天线的处理符号。实现遍历树有几种可选方法。我们选择的是广度优先搜索法，因为该方案具有备受欢迎的硬件友好特征。在每一层，只选择具有最小部分欧氏距离的 K 节点来计算扩展情况。这类检测器称为 K-best 检测器。

范数计算是在球形检测器的部分欧氏距离 (PED) 模块中完成的。根据树的层次，我们可以利用 3 种不同的 PED 模块。根节点 PED 模块计算所有可能的 PED（树层次指数是 i = M = 8）。第二层 PED 模块分别对第一层计算得出的 8 个幸存路径进行计算，计算每个幸存路径的 8 个可能 PED。在树层次指数 i = 7 的情况下，将得出 64 个生成 PED。第三类 PED 模块用于树的其它层次，负责为上一级计算得出的所有 PED 计算出最近节点的 PED。这会将每一层的分支数量固定为 K = 64，如此传播至最后一层 i = 1，并产生 64 个最终 PED 以及它们的检测符号序列。

SD 流水线架构可允许在每个时钟周期中进行数据处理。因此，树的每一层只需要 1 个 PED 模块，从而使 PED 模块总数量与树的层数量相等。这样对 4×4 64-QAM 调制方式而言，PED 的总数量为 8。图 3 为 SD 结构图。

FPGA 性能实现目标

目标 FPGA 器件为 Xilinx Virtex-5，其目标时钟频率是 225MHz。如果为每个数据副载波估计信道矩阵，就会限制每个信道矩阵的处理时间。对于选定的时钟频率和 5MHz 通信带宽（在WiMAX 系统中相当于 360 个数据副载波），我们可以按如下公式计算每个信道矩阵间隔的可用处理时钟周期数：

$$num\_cycles = \frac{(102.9\mu s / 360)}{1/225MHz} \cong 64$$

如前所述，我们设计了计算量要求最高的 4×4 天线与 64-QAM 调制方案的配置结构。本方案的原始数据速率可达到 83.965Mb/s。

针对 FPGA 的高级综合

高级综合工具将具体算法的高级描述作为其输入，可执行并生成基于 FPGA 实现的 RTL 描述，如图 4 所示。该 RTL 描述可与参考设计、IP 核或已有 RTL 代码结合在一起，利用传统 Xilinx ISE/EDK 工具来创建完整的 FPGA 实现方案。

现代高级综合工具将非定时 C/C++ 描述作为输入规范。该工具对同一 C/C++ 代码执行 2 种解读：输入/输出行为的顺序语义与基于 C/C++ 代码与编译器指令的架构规范。这些高级综合工具根据 C/C++ 代码、编译器指令以及目标吞吐量的要求来生成高性能的流水线架构。高级综合工具还具有流水线级数的自动插入与资源共享等其它功能，这样可减少 FPGA 资源的消耗。基本上讲，高级综合工具提高了 FPGA 设计的抽象级，并将耗时且容易发生错误的 RTL 设计任务实现透明化。

我们将重点集中在 C++ 描述的使用，其目标是利用 C++ 模板类来代表任意精度的整数类型以及利用模板功能来代表架构中参数化的模块。

图 5 给出总体设计方案，出发点是从 MATLAB® 功能描述中获得的 C/C++ 参考代码。如图所示，在任何硬件目标平台中实现应用的第一步通常是重组 C/C++ 参考代码。我们将“重组”表示为将原始 C/C++ 代码以一种更适合目标处理引擎的格式进行重写（通常是为了使

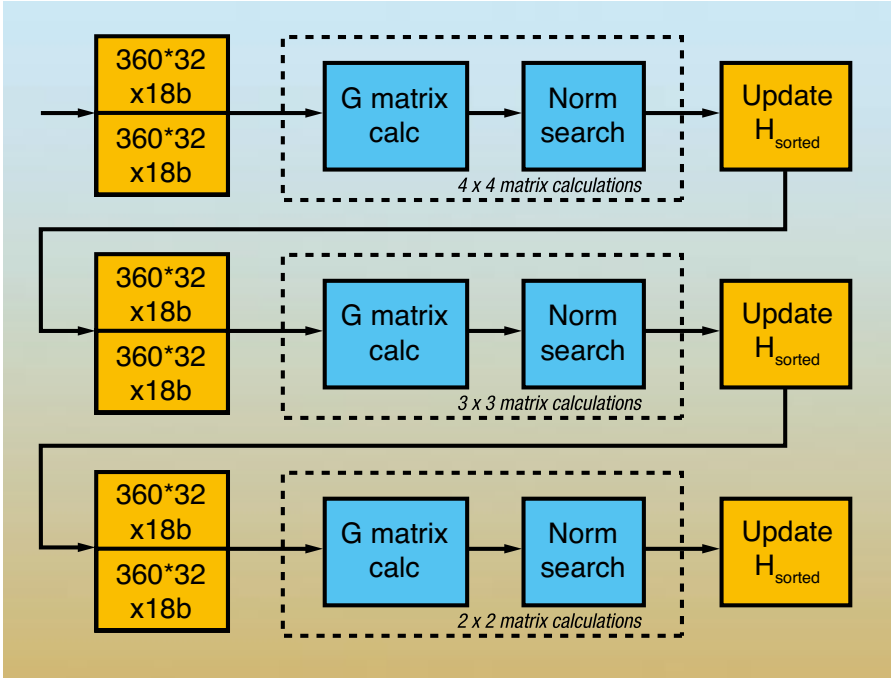


图 2 迭代信道矩阵重排序算法

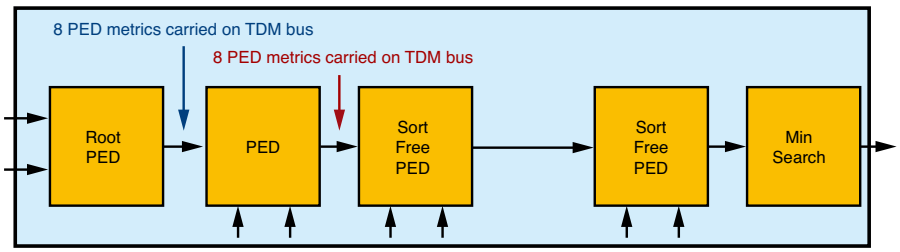


图 3 球形检测器处理流水线

代码清晰明了、易于概念理解而不考虑最优性能）。例如，必须重新排列 DSP 处理器中的应用代码，以便算法可以高效地利用缓存。当目标器件是 FPGA 时，重组可能包括：重写代码用以表示可达到预期吞吐量的架构规范，或重写代码以高效利用 FPGA 的特定功能，例如嵌入式 DSP 宏。

我们通过利用传统 C/C++ 编译器（例如 gcc），并重新利用适用于 C/C++ 参考代码校验的 C/C++ 级测试台，可实现对 C/C++ 执行代码的功能校验。C/C++ 执行代码是高级综合工具的主要输入。但是，额外输入会显著影响生成硬件、及其性能和 FPGA 资源的使用数量。因此存在 2 个基本约束条件，分别为目标 FPGA 系列产品与目标时钟频率，它们都会影响生成架构中流水线级数的数量。

此外，高级综合工具可接受编译器指令（例如，在 C/C++ 代码中插入注解），使设计者可以在不同 C/C++ 代码段中应用不同的指令类型。例如，应用于循环（例如，循环展开）和其它数组（例如，指定哪些 FPGA 资源必须用于执行数组操作）的指令。

根据这些输入，高级综合工具生成输出架构 (RTL)，并报告其吞吐量。然后设计者根据吞吐量的大小来修改指令，执行 C/C++ 代码。如果生成的架构满足吞吐量方面的要求，则 RTL 输出用于 FPGA 实现工具 (ISE/EDK) 的输入。只有在完成逻辑综合和布局布线之后，才报告可实现的最终时钟频率和 FPGA 资源使用数量。如果该设计不能满足时序和 FPGA 资源限制这 2 方面要求，则该设计与预期设计不相符，那么设计者应修改 C/C++ 执行代码或编译器指令。

SD 高级综合实现

我们已经利用 AutoESL 的 AutoPilot 2010.07.ft 工具实现了图 1 所示的 WiMAX 球形解码器的 3 个关键构建模块。需要重点强调的是，本方案所选择的算法与最近 SDR 会议论文中的算法一致，因此可以实现相同的 BER。在本节中，我们给出了用于该特殊实现的代码重写与编译器指令的具体示例。

从 MATLAB 功能描述中获得的原始 C 参考代码大约有 2000 行，包括综合 C 代码与验证 C 代码。

代码只包括使用 C 内置数据类型的定点运算。一个对FPGA 友好的实现几乎可以完成所有要求的浮点运算（例如 sqrt）。

除了描述 FPGA 综合功能的 C 参考代码之外，还有一个完整的 C 级验证测试平台。我们从 MATLAB 描述中生成输入测试矢量和重要的输出参考文件。

原始 C/C++ 参考代码符合 MATLAB 规范的比特精度要求，并可通过由多个数据集构成的回归分析套件。

该 C/C++ 参考代码经历了不同类型的代码重组。例如，图 5 显示了 3 个我们已经实现的代码重组的例子。我们重新使用 C 级验证设施来检验 C/C++ 代码执行中的任何变化。而且，我们是在 C 级执行所有验证，而不是在寄存器传送级，这样可避免非常耗时的 RTL 仿真，从而有助于减少总体开发时间。

宏架构说明

代码重构的最重要部分是重写 C/C++ 代码，以描述可有效实现特定功能的宏架构。换句话说，设计人员负责宏架构说明，而高级综合工具负责宏架构的生成。这类代码重组对得到的吞吐量和质量结果具有重大影响。

就球形解码器而言，有几个这类代码重组的实例。例如，为了满足信道排序模块的高吞吐量要求，设计人员应使用 C/C++ 来描述图 2 所示的宏架构。这类 C/C++ 代码由几个以数组为传递参数的函数构成。该高级综合工具可自动调用乒乓缓冲器中的数组，以实现在流水线中并行执行多个矩阵计算模块。

本级代码重组的另一个实例是决定特定模块的 TDM 结构中所使用信道的数量（例如，信道矩阵重排序模块使用 5 条信道，修正后的实部 QR 分解模块使用 15 条信道）。

图 6 是宏架构说明的一个实例。图 3 为描述球形检测器的 C++ 代码片段框图。我们注意到图中有一条调用 9 个函数的流水线，其中每个函数代表图 3 中的一个模块。函数之间的通信通过传递数组来完成，这些数组被第 5 行和第 7 行的适当指令 (pragmas) 映射至数据流接口（不是 FPGA 嵌入式 BRAM 存储器）。

参数化的重要性

参数化是代码重写的另一个关键实例。我们广泛利用 C++ 模板函数来表示架构中的参数化模块。

在球形解码器的实现过程中，该类代码重写有几个不同实例。一个具体

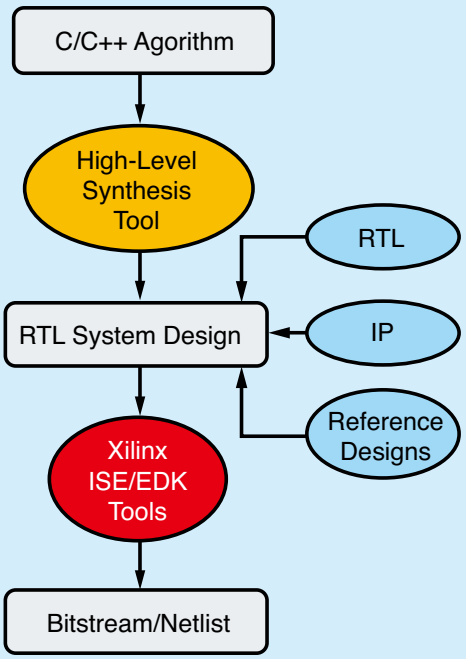


图 4 – 针对 FPGA 的高级综合

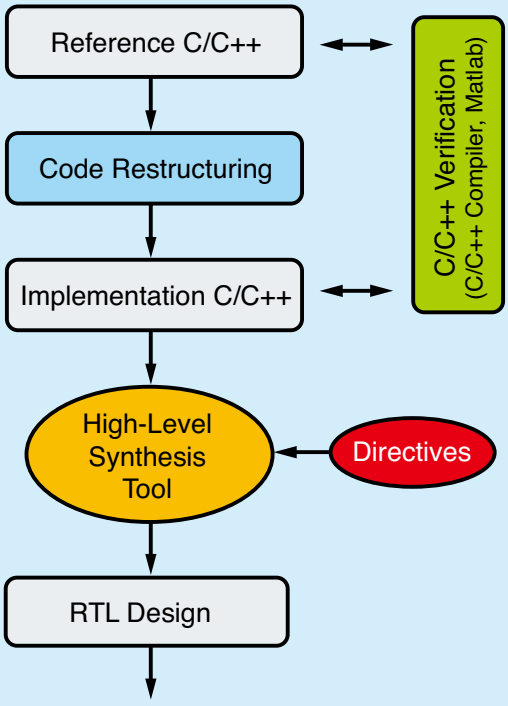


图 5 C/C++ 改进型迭代设计方案

实例是应用于信道重排序模块的不同类型矩阵操作。图 2 所示的矩阵计算模块（4×4, 3×3与2×2）就包含不同类型的矩阵操作，例如矩阵求逆或矩阵相乘。将这些模块进行编码并作为 C++ 模板函数，模板参数即矩阵的行数与列数。

图 7 为矩阵相乘的 C++ 模板函数。除了矩阵行数和列数这两个参数之外，该模板函数还有第三个参数，即 MM_II（矩阵相乘初始化间隔），该参数用于指定二次连续循环迭代之间的时钟周期数量。第 9 行的指令 (pragma) 用于对具体实例所需吞吐量进行参数化。这是一项很重要的功能，原因是它可对生成的微体系架构产生重要影响—也就是高级综合工具具备充分利用资源共享的能力，从而可以减少用于具体实现中的 FPGA 资源数量。例如，高级综合工具仅仅通过修改初始化间隔 (Initiation Interval) 参数并使用相同的 C++ 代码，即可在执行不同矩阵求逆 (4×4, 3×3, 2×2) 模块的过程中自动实现不同层面的资源共享。

FPGA 最优化

FPGA 最优化是代码重写的最后一个实例。设计人员可以重写 C/C++ 代码，以更高效地利用特定 FPGA 资源，从而可以改善时序并减小存储区域的使用。该类最优化方面的两个具体实例是比特宽度最优化与嵌入式 DSP 模块 (DSP48) 的高效使用。高效利用 DSP48 可以改善时序并提高 FPGA 资源利用率。

我们利用内置 C/C++ 数据类型（例如short、int）来编写 C/C++ 参考代码，同时利用 18 位定点数据类型来表示矩阵元素。我们已经利用 C++ 模板类来表示任意精度的定点数据类型，因此可减少 FPGA 资源的使用并将时序影响最小化。

```
1: void sphere_detector_top (…) {
2:  #pragma AP DATAFLOW
3:  // PED streams between pipeline blocks
4:  ap_int<18>     PED_7[RVD_MODULATION ];
5:  #pragma AP ARRAY_STREAM variable=PED_7 depth=1 stream
6:  ap_int<4>      symb_7[RVD_MODULATION ];
7:  #pragma AP ARRAY_STREAM variable=symb_7 depth=1 stream
8:  main_label :{
9:    RootPED (r_7, y_7, ..., symb_7);
10:   PED(r_6 , y_6, ..., symb_7, symb_6 );
11:   SortFreePED <5,2>(r_5 , y_5, ..., symb_6 , symb_5 );
12:   SortFreePED <4,3>(r_4 , y_4, ..., symb_5 , symb_4 );
13:   SortFreePED <3,4>(r_3 , y_3, ..., symb_4 , symb_3 );
14:   SortFreePED <2,5>(r_2 , y_2, ..., symb_3 , symb_2 );
15:   SortFreePED <1,6>(r_1 , y_1, ..., symb_2 , symb_1 );
16:   SortFreePED <0,7>(r_0 , y_0, ..., symb_1 , symb_0 );
17:   // find minimum PED
18:   min_finder (PED_0 , symb_0 , min_PED , min_symb_list );
19:  }
20:}
```

图 6 球形检测器宏架构描述

```
1:  template< int X_DIMENSION, int Y_DIMENSION, int MM_II >
2:  void matrix_multiply _(...) {
3:  #pragma AP ARRAY_PARTITION variable= chunk_in_re dim=1
4:  #pragma AP ARRAY_PARTITION variable= chunk_in_im dim=1
5:   // matrix multiplication of a   A'*A matrix
6:   for (index_a = 0; index_a < TDM_CHUNKS; index_a ++) {
7:     for (index_b = 0; index_b < X_DIMENSION; index_b ++) {
8:       for (index_c = 0; index_c < Y_DIMENSION; index_c ++) {
9:  #pragma AP PIPELINE II = MM_II
10:              <loop body >
11:   } } }
12: }
```

图 7 代码参数化实例

```
1:  template  <int Wa, int Wb, int Wc>
2:  ap_int<36> SUBMUL( ap_int<Wa> a, ap_int<Wb> b, ap_int<Wc> c) {
3:    #pragma AP LATENCY max=2
4:    #pragma AP INTERFACE ap_none port=return register
5:          .
6:    ap_int<36> c_36 = c; // sign extension
7:    return  c_36-a*b;
8:  }
```

图 8 针对 DSP48 有效利用的 FPGA 性能最优化

图 8 是一个先执行乘法后执行减法的 C++ 模板函数，而输入操作数宽度是需要设置的参数。可将这两个运算操作映射至嵌入式 DSP48 模块中。在图 8 中，有两条指令指示高级综合工具用最多两个时钟周期来调度这些操作并使用寄存器来存储输出返回值。

生产力度量指标

在图 9 中，我们绘出利用AutoESL的 AutoPilot 所生成设计的规模大小（即 FPGA 资源使用情况）随时间变化的曲线，并与传统系统生成器 (RTL)的实现过程相比较。利用高级综合工具，我们可以实现很多有效的解决方案，且这些解决方案的规模随时间而变化。因此，设计人员可根据代码重组的数量，在获取解决方案的速度与解决方案的规模大小之间做出权衡。另外，RTL 解决方案只有一种，而且开发周期较长。

我们已经观察到可以用相对较少的时间来获得几个明显比传统 RTL 解决方案使用更多 FPGA 资源（例如区域）的综合解决方案。另一方面，设计人员也可自行决定在工具专家等级下工作，通过执行更高级的 C/C++ 代码重组技术（例如特定 FPGA 的最优化），实现用更少的 FPGA 资源生成更多的解决方案。

最后，因为我们是在 C/C++ 级执行所有验证，从而可以避免耗时的 RTL。因此，在 C/C++ 级执行设计验证将明显缩短总体开发时间。

质量结果

在图 10 中，我们对比了分别利用高级综合工具与系统参考生成器（基本属于结构化的 RTL 设计，显示使用如 DSP48 模块的 FPGA 基元）这两种不同方法来实现的完整球形解码器，在其最终 FPGA 资源使用量和总体开发时间这两方面进行了比较。AutoESL 开发时间包括工具学习、产生结果、设计空间探测与详细验证所需要的时间。

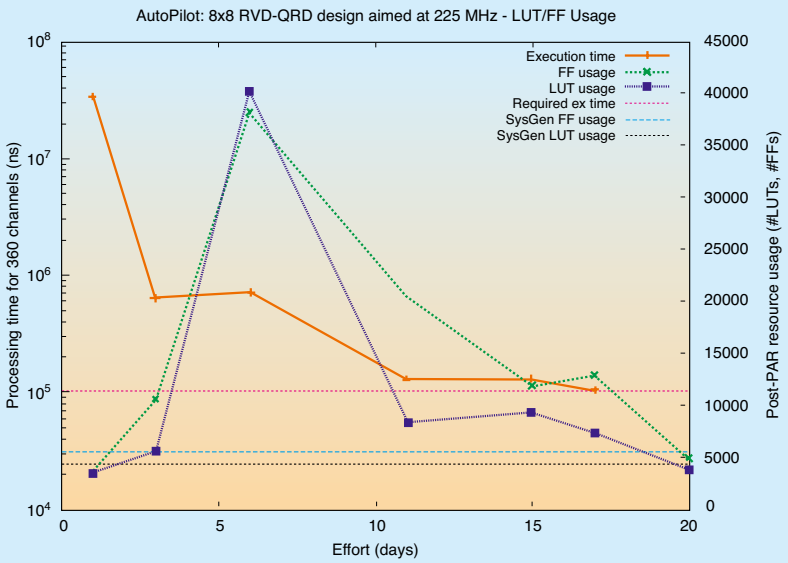


图 9 FPGA 资源用量随着开发时间增加而下降

度量标准	SysGen Expert	AutoESL Expert	% 差值
开发时间（周）	16.5	15	-9%
LUTs	27,870	29,060	+4%
寄存器	42,035	31, 000	-26%
DSP48s	237	201	-15%
18K BRAM	138	99	-28%

图 10 质量结果的度量标准体现了 AutoESL 优势。

为了更精确地比较，我们利用针对 Virtex-5 FPGA 的最新 Xilinx ISE 12.1 工具重新实现 RTL 参考设计。同样，我们利用针对同类 FPGA 的 ISE 12.1 来执行由 AutoESL AutoPilot 生成的 RTL。图 10 显示 AutoESL AutoPilot 节约FPGA 资源效果明显，主要是因为在实现矩阵求逆模块时实现了资源共享。我们也观察到寄存器的使用数量明显减少，查找表 (LUT) 的使用量略有提高。产生这种结果的部分原因在于 AutoESL 实现过程中延迟线被映射至 SRL16（例如 LUT），而在系统生成器方案中，则利用寄存器实现上述功能。

在其它模块中，我们交替使用 BRAM 与 LUTRAM，导致信道预处理器的 BRAM 使用率较低。

AutoESL AutoPilot 完成对底层 FPGA 实现细节的抽象（例如时序与流水线设计）。与使用传统 RTL 设计方案相比，其产生的质量结果更具竞争力。C/C++ 级验证避免使用耗时的 RTL 仿真，从而可以减少总体开发时间。但是，对于具有挑战性的复杂设计而言，如果想取得卓越的效果，则必须给出优异的宏架构定义，且必须具备扎实的 FPGA 设计工具知识，还要有理解与解释 FPGA 工具报告的能力。

赛灵思ISE 13 全面支持 7 系列 FPGA
利用全新Team Design Flow 强化系统级生产力
利用开放式行业标准的创新工具与即插即用 IP，将加速设计创建、验证、实施并降低系统功耗

赛灵思屡获殊荣的设计工具和 IP 套件ISE®13新增了许多增强特性，可以提高片上系统（SoC） 设计团队的生产力，针对 Spartan®-6、Virtex®-6 和 7 系列 FPGA 以及行业领先的容量高达 200 万个逻辑单元的 Virtex-7 2000T 器件，加速实现真正的即插即用 IP。针对减少开发时间和成本，ISE 13设计套件引入了加速验证、支持 IP-XACT 的即插即用 IP以及全新的 Team Design Flow，让多名工程师利用时序可重复功能同时开展工作，从而缩短设计周期。由于赛灵思已经推出系统门容量高达数百万的 FPGA，例如采用堆叠硅片互连技术 的Virtex-7 2000T 器件，能够将串行、并行和数字信号处理融合到一个芯片之上，并提供高达 28Gbps 的收发器速度，因此，生产力的需求在这些高度复杂的设计中极为重要。然而，根据《国际半导体技术发展蓝图》(International Technology Roadmap for Semiconductors)，若要维持生产力曲线，行业必须将周期缩短 50%。由于超过一半的设计周期都花在了验证环节上，ISE 13设计套件采用了新的硬件协同仿真功能和AMBA®4 AXI4（高级扩展接口）总线函数仿真模型，可以直接提高设计验证团队的生产力。