WP491 (v1.0) 2017 年 3 月 30 日

将浮点转为定点
大幅降低功耗和成本

作者：Ambrose Finnerty 和 Hervé Ratigner

赛灵思器件和工具支持从二进制到双精度在内的多种数据类型。UltraScale ™ 架构的可扩展精度提供极大灵活性，便于优化功耗和资源利用，同时满足设计性能目标要求。

摘要

在数据中心、航空航天与军用、5G 无线以及汽车等领域，客户必须满足高级驾驶员辅助 (ADAS)、雷达和深度学习等应用中严峻的散热、功耗和成本要求。

要实现这些目标，一种极为有效的方法是用定点数实现信号处理链。赛灵思 FPGA 和 SoC 具备固有的可变精度支持，允许客户轻松调整以适应不断演变的朝更低精度解决方案发展的这种行业趋势。

赛灵思提供一种包含 Vivado® 高层次综合 (HLS) 的工具流程，允许客户方便地评估 C/C++ 设计的更低精度实现方案，诸如定点等。

简介：赛灵思支持的数据类型

赛灵思 All Programmable 器件和工具支持从二进制到双精度浮点在内的多种数据类型。用定点实现的设计总是比用浮点实现的同一设计更加高效，因为定点实现方案所占用的资源和消耗的功耗更少。若将设计迁移到定点，功耗和占用面积缩减一半并不稀奇。

相较于浮点，定点数据类型的优势包括：

- 逻辑资源占用减少
- 功耗降低
- 材料成本降低
- 时延缩短

赛灵思所有器件均支持客户采用浮点数据类型提供的动态范围，能实现高达 7.3TFLOPs 的单精度浮点 DSP 性能。

业界领先的赛灵思工具套件提供浮点支持。Vivado® 高层次综合 (HLS) [参考资料 1] 和 System Generator for DSP [参考资料 2] 本身均支持可变的浮点精度，包括半精度 (FP16)、单精度 (FP32) 和双精度 (FP64)；System Generator 还支持自定义精度，具备更大的灵活性。这些工具本身还支持可变定点数据类型。

***表格 1：*赛灵思工具支持浮点和定点数据类型**

赛灵思工具	FP16	FP32	FP64	自定义 FP	定点
Vivado HLS	Y	Y	Y	N	Y
System Generator for DSP	Y(1)	Y	Y	Y	Y
浮点运算符 IP	Y	Y	Y	Y	Y(2)

注：

1. System Generator for DSP 原本不支持 FP16，但支持自定义 FP16。
2. 浮点运算符内核支持 → 定点到浮点转换、浮点到定点转换以及浮点到浮点的精度变换。

赛灵思器件和工具均能支持可变精度数据类型，以便客户能够简单、灵活地调整和适应行业趋势的变化，例如图像分类只要求 INT8 或更低的定点计算，以保持可接受的推断精度 [参考资料 3][参考资料 4]。

用于高计算强度工作负载的其他器件，例如 GPU，在传统上其结构决定了只能有效支持单精度浮点。这些厂商现在也开始重新设计产品，以应对趋势的变化。赛灵思的可扩展架构允许客户调整信号处理链的精度，以便满足日新月异的行业需求。

当选择实现浮点还是定点信号处理链时，客户必须在功耗、成本、生产力和精度之间仔细权衡。

赛灵思灵活的 DSP48E2 Slice 可使用所有数据类型进行重要的 DSP 计算。当实现新的定点设计或针对某些应用（适用浮点到定点转换）将现有设计从浮点转换成定点时，DSP Slice 与赛灵思工具集相结合能够带来巨大优势和灵活性 [参考资料 5]。

对于采用 C/C++ 语言设计的客户，赛灵思提供 Vivado HLS 并支持任意精度定点数据类型，使客户能够方便地采用定点进行设计或者将现有的 C/C++ 设计转换成定点。

浮点转换为定点的优势

对于目前几乎所有的设计，最小化功耗是需要优先处理的问题。大多数应用产品必须首先满足严格的功耗和散热范围要求，才能投产。

普遍接受的一个原则是，浮点设计较之低精度设计而言，功耗更大 [参考资料 6][参考资料 7]。这对 FPGA 来说也一样，其中的浮点 DSP 模块已被硬化在 FPGA 中，另外客户必须使用提供的 DSP 资源和其它 FPGA 资源来实现软解决方案。浮点方案与同等的定点解决方案相比需占用更多的 FPGA 资源。资源占用增多，功耗随之增大，最终会增加设计实现的总成本。

将浮点设计转换为定点设计有助于满足严格的规范，具体体现在以下几个方面：

- 减少 FPGA 资源占用
  - 使用定点数据类型时，所需的 DSP48E2、查找表 (LUT) 和触发器更少。
  - 存储定点数字所需的存储容量更小。
- 功耗更低
  - 减少 FPGA 资源利用自然就会降低功耗。
- 材料成本降低
  - 设计人员可利用额外的资源以相同成本在应用中实现附加功能。
  - 资源的节约能大幅提升 FPGA 的计算能力。计算能力的提升可以让很多应用受益，例如机器学习 DNN。
  - 资源的节省还可能减小设计所需的器件尺寸。
- 降低时延
  - 当实现 FIR 时减少所用的资源，尤其是减少 DSP48E2 Slice 的占用，能降低定点设计的时延。
- 相近的性能和精度
  - 对于不需要用浮点实现动态范围的设计和应用，定点方案能提供相近的结果和精度。 有些情况下，结果甚至更好。

过去由于缺乏工具支持，难以将设计从浮点转换为定点。对于针对赛灵思 All Programmable 器件的 C/C++ 开发人员来说，可使用 Vivado HLS 减少转换过程中遇到的挑战。

这种转换能带来多种优势，在适用情况下应认真考虑——尤其是不需要利用浮点来实现动态范围和浮点精度的设计，而且很小的可预见的精度损失不会在部署后的应用中导致无效性。

实例：将浮点 FIR 滤波器转换为定点

Vivado HLS 中简单的 FIR 滤波器设计 [参考资料 8] 可用来展示浮点 FIR 设计转换为定点设计如何减少所用资源和功耗并实现相近的结果精度。

单精度浮点 FIR

在 C++ FIR 函数代码中，顶层函数将 FIR.h 报头文件中找到的类 CFir 文件 (class CFir) 实例化。

```
#include "FIR.h"


// Top-level function with class instantiated
fp_acc_t fp_FIR(fp_data_t x) {
    #pragma HLS PIPELINE
    static CFir<fp_coef_t, fp_data_t, fp_acc_t> fir1;
    return fir1(x);
}
```

CFir 类 (CFir class) 是主要的 FIR 算法，在报头文件 FIR.h 中定义。

```
// FIR main algorithm
template<class coef_T, class data_T, class acc_T>
acc_T CFir<coef_T, data_T, acc_T>::operator()(data_T x) {
//caller uses #pragma HLS PIPELINE which makes this function pipelined as
needed.
#pragma HLS ARRAY_PARTITION variable=c complete dim=1
#pragma HLS ARRAY_PARTITION variable=shift_reg complete dim=1
   int i;
   acc_T acc = 0;
   data_T m;

   loop: for (i = N-1; i >= 0; i--) {
      if (i == 0) {
         m = x;
         shift_reg[0] = x;
       } else {
         m = shift_reg[i-1];
         if (i != (N-1)) {
            shift_reg[i] = shift_reg[i - 1];
         }
      }
      acc += m * c[i];
   }
   return acc;
}
```

此函数包含重要的 ARRAY_PARTITION 编译指示，以确保设计的所有实现方案都是 II=1（迭代间隔为 1）[参考资料 9]。PIPELINE 编译指示也被应用到顶层函数调用。

这些编译指示、并行产品实现以及用于执行累加的加法器树，能够在整个 FIR 函数中确保最低时延（无论数据类型如何），同时保持 II = 1。

在 fp_FIR 函数中，fp_coef_t、fp_data_t 和 fp_acc_t 都被定义为浮点类型，即 C++ 默认的单精度浮点数据类型。

```
// float

typedef float fp_coef_t;
typedef float fp_data_t;
typedef float fp_acc_t;
```

通过报头文件中的 include 命令加载滤波器系数。

```
template<class coef_T, class data_T, class acc_T>
const coef_T CFir<coef_T, data_T, acc_T>::c[] = {
   #include "FIR_fp.inc"
};
```

用系数创建一个对称 FIR 滤波器，但本例中，未使用 DSP48E2 Slice 中的预加法器。如果使用预加法器，会实现更高效率。

以下是针对 85 抽头 FIR 滤波器得到的结果，在 Vivado HLS 中运行 C 综合与实现，并采用 XCVU9P-2FLGB2104 器件上的 400MHz 时钟（2.5ns 时钟周期）。见表 2。

表 2： **单精度浮点 FIR 的实现后结果**

单精度浮点 (FP32)	
F_{MAX}	500MHz
时延（时钟周期）	91
迭代间隔 (II)	1
DSP48E2	423
查找表 (LUT)	23,101

本例中，需要 423 个 DSP48E2 以及约 23,000 个 LUT 来实现单精度浮点 FIR。实现后，时延为 91 个时钟周期，F_{MAX} 为 500MHz（远远高于 400MHz 的目标）。

转换到定点 FIR 滤波器

为实现最高 DSP 效率，浮点到定点的转换必须考虑 DSP Slice 的总线宽度，即 27x18 位乘法器和 48 位累加器。将总线宽度进一步缩减到到设计允许的最低水平，从而尽可能减少资源占用和功耗。

针对这个 FIR 滤波器实例，定义以下定点数据类型以匹配 DSP48E2 Slice 中的总线大小，即 18 位系数中 1 个整数位和 17 个小数位；27 位数据中 15 个整数位和 12 个小数位；以及 48 位累加器中 19 个整数位和 29 个小数位。

```
// fixed points
#include <ap_fixed.h>


typedef ap_fixed<18,1> fx_coef_t;
typedef ap_fixed<27,15> fx_data_t;
typedef ap_fixed<48,19> fx_acc_t;
```

要使用 Vivado HLS 固有的 ap_fixed 数据类型，必须包含 ap_fixed.h 报头文件，以定义任意定点数据类型 [参考资料 9]。

再次采用 400MHz 时钟（2.5ns 时钟周期）和 XCVU9P-2FLGB2104 器件，定点 FIR 设计的 C 综合与实现产生的结果如表 3 所示。

表 3： **比较两种设计的实现后结果**

	单精度浮点	定点	定点的优势
F_{MAX}（实现后）	500MHz	580MHz	加快 16%
时延	91	12	降低约 7.5 倍
迭代间隔 (II)	1	1	–
DSP48E2	423	85	DSP 效率提高 5 倍
查找表 (LUT)	23,106	1,973	逻辑效率提升 11 倍

正如结果显示的那样，重视时延和 FPGA 资源利用率，能获得明显的改进。

在 UltraScale 架构中，通过将多个 DSP48E2 Slice 级联，必要时仍可支持更高总线宽度。采用级联 DSP48E2 Slice 的定点设计，与浮点方案相比仍可显著减少资源占用和功耗。

比较滤波器精度

在 System Generator for DSP 中使用 Vivado HLS 模块（来自赛灵思模块集），在 MATLAB®/Simulink® 环境中比较 FIR 滤波器的两种实现方案。见图 1。

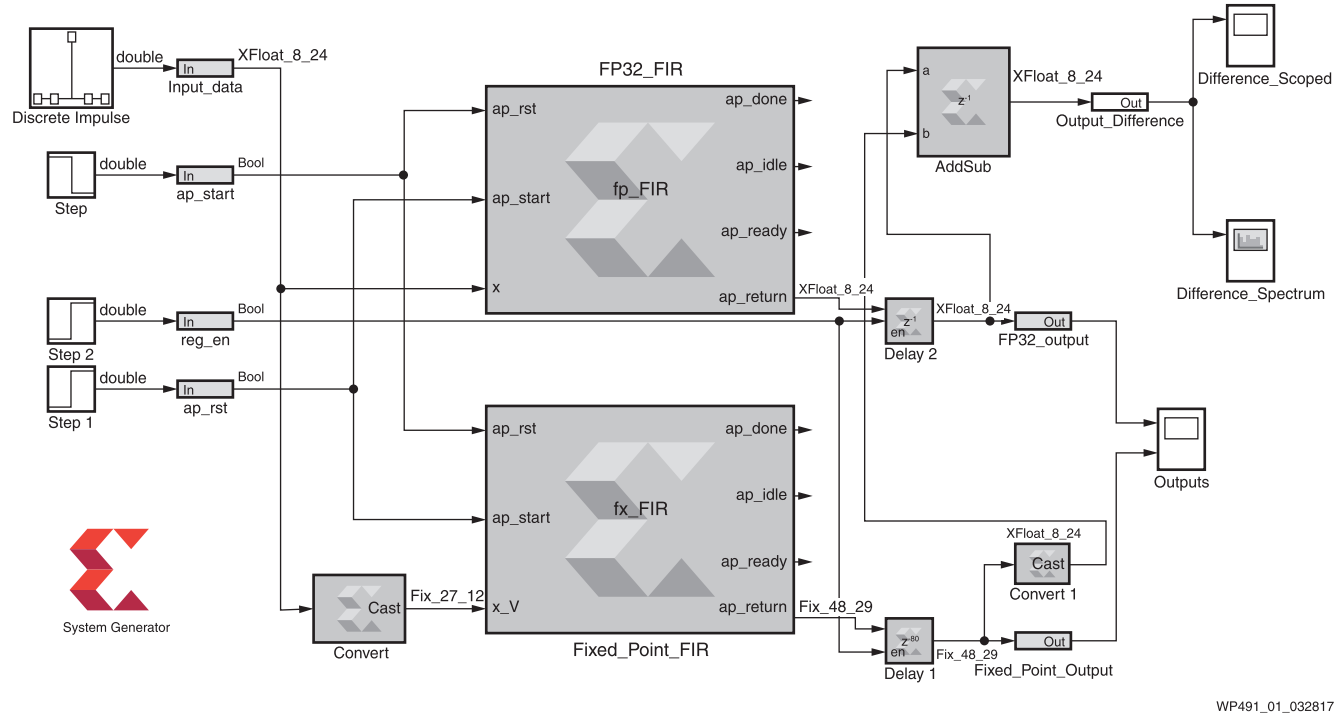


图 1： **System Generator for DSP 模型 - 使用两种 HLS 解决方案进行分析**

System Generator 模型由两个 Vivado HLS 模块构成，都经过配置以包含来自 Vivado HLS 的单精度浮点 (FP32) 和定点 FIR 解决方案。两个模块具有相同的输入和离散脉冲信号，然后在 Simulink 示波器上比较每个 FIR 的输出。见图 2。

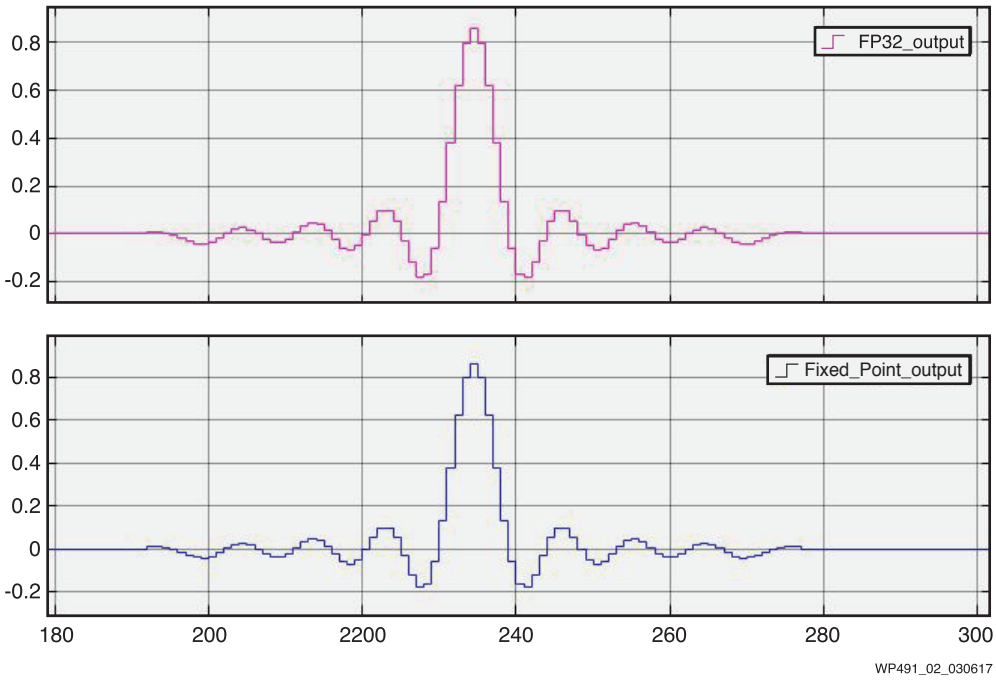


图 2： **System Generator 中两个 HLS 设计的输出**

为了方便比较输出，有必要延迟定点结果，以按照两种解决方案之间的时延差进行对比。

正如预期的那样，两种 FIR 滤波器产生的结果几乎相同，差异很小。

为进一步分析信号，将两个输出相减。得到的信号表明精度损失非常小，处在如图 3 所示频谱分析图中的 -100dBm 至 -160dBm 范围。

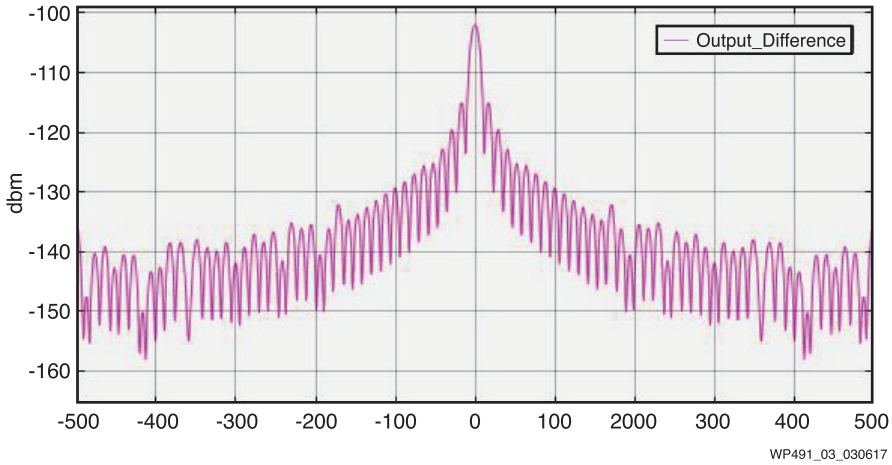


图 3： **两个输出差的 dB 图**

关键优势

当把原始单精度浮点 FIR 滤波器的结果与转换后定点 FIR 滤波器的结果进行比较时，发现定点设计不仅减少了资源占用，降低了时延，同时还能保持甚至提高设计的最大频率 (F_{MAX})。见图 4。

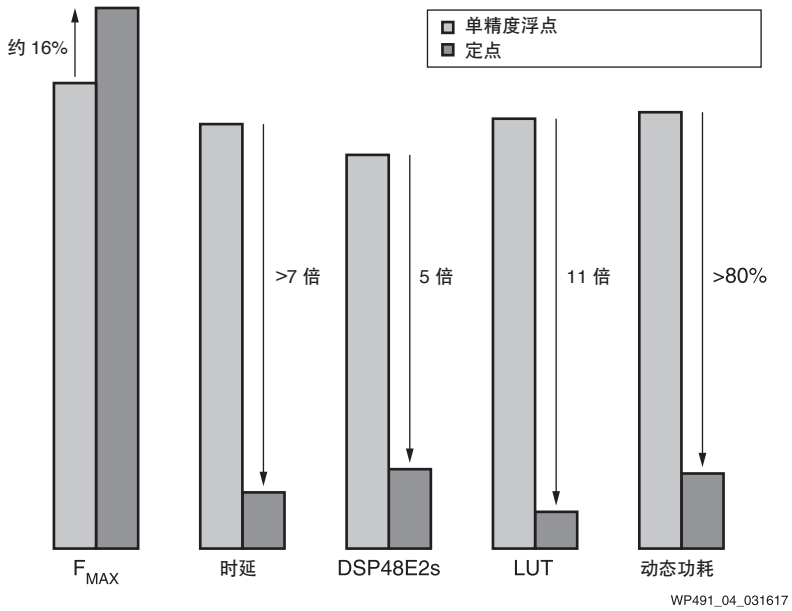


图 4： **定点 - 性能相近，并减少时延、资源占用和功耗**

大大减少 FPGA 资源占用

本例中的定点 FIR 所占资源不足原始浮点 FIR 的五分之一。

选择总线宽度，实现到硬件中 DSP48E2 slice 的最佳映射。这样允许每次相乘都在一个 DSP48E2 Slice 中完成，并针对 85 个系数并行完成。这将 DSP48E2 Slice 的使用量减小至浮点解决方案的 20%。

实现 FPGA 架构中 LUT 的大幅节省（约 90%），因为在定点方案中，无需额外的 LUT 来构建浮点运算。

如果一个设计有 10 个 FIR 滤波器，预计功耗会随设计而扩展。表 4 显示了 10 FIR 滤波器设计的单精度和定点实现方案的 XCVU9P FPGA 资源占用。当将单精度浮点的资源占用与定点实现方案进行对比时，会看到明显的差异。

表 4： **两种数据类型 10 FIR 滤波器所使用的资源**

	DSP48E2		LUT	
	资源数量	器件利用率	资源数量	器件利用率
单精度浮点	4,230	62%	231,060	20%
定点	850	12%	19,730	2%

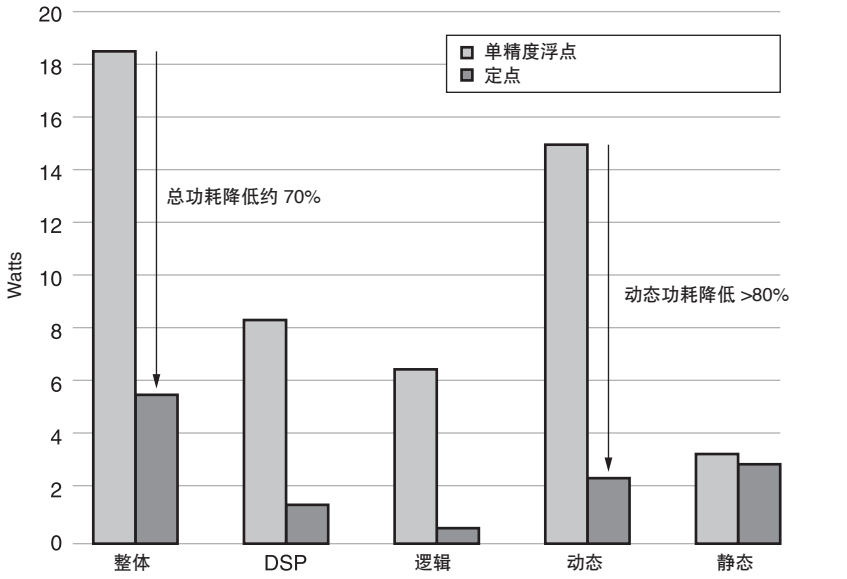
显著的资源节省能实现多种优势，为设计人员带来深远的影响，有助于他们改进设计特性集、设计功耗、设计性能和设计成本。

实现显著的功耗节省

显著的资源节省能够相应地降低功耗量。

将本白皮书中单个 FIR 滤波器的两种实现方案的功耗估算结果进行比较，发现定点 FIR 的功耗减少 1.4W。两种方案中，器件的静态功耗都是略高于 3W，单个单精度浮点 FIR 设计的总功耗为 4.7W。这表明该设计的动态功耗节省 80% 以上，定点 FIR 功耗为 3.3W。

再看看 10 FIR 滤波器设计，可利用赛灵思功耗估算器 (XPE) 和表 4 中的资源来获得两种实现方案的功耗估算结果。图 5 对节省效果进行了比较。



注：2016.4 XPE，假设 DSP48E2 Slice 翻转率最差为 25%，而且 F_{max} 为 400MHz

WP491_05_031617

图 5： **10 FIR 滤波器实例：利用定点实现显著的功耗节约**

这个 10 FIR 滤波器实例中，把设计转换成定点数据类型后，总功耗节省 70%。如果设计具有大量浮点信号处理，需占用大量 FPGA 资源，则通过将部分或全部浮点信号处理链转换成定点，能够实现巨大的功耗节省。

削减材料成本

将浮点设计转换成定点方案，能大大减少 FPGA 资源占用。FPGA 资源的削减能降低材料成本。通过三种方法来实现。

1 可利用最新可用的 FPGA 资源来增加应用特性集。

2 由于所用 FPGA 资源大量减少以及通过数据路径提高 F_{MAX}，因此 FPGA 的总体计算能力显著提高。

3 由于所需 FPGA 资源减少，因此设计可迁移到更小型的赛灵思 FPGA 中。

相近的精度

通过比较单个 FIR 滤波器设计两种实现方案的输出，会发现定点实现方案提供相近的滤波器精度，精度损失仅为 -100dBm 至 -160dBm，同时能降低功耗和成本。

然而，定点方案无法获得相同的动态范围，导致设计中出现可预测的精度损失。对于很多设计来说这不

成问题，因为只需要最低标准的精度。与单个 FIR 实例类似，这类设计很适合转换为定点。

对于需要更大精度值的设计，有时可将信号处理链中的中间值从浮点转换为定点。这种方案使设计人员能够将设计的特定部分（而非全部）转换为定点。最终，这使设计人员能够在需要时保持动态范围，确保维持数据路径的精度，同时充分发挥定点实现方案带来的部分优势。

降低时延

对于单个 FIR 设计实例，可通过滤波器降低时延——定点实现方案为 12 个时钟周期，浮点设计为 91 个时钟周期。随着资源用量减少，尤其是 DSP48E2 Slice 减少，有望降低时延。

除了降低时延，在单个 FIR 实例中，还能提升 F_{MAX}，实现之后能将 F_{MAX} 提升 16%。

结论

赛灵思 All Programmable 器件和工具支持多种数据类型，包括多种精度的浮点和定点。使用浮点的设计与使用定点的同一设计相比，资源用量和功耗都要更高，无论针对 FPGA 还是其他架构（例如 GPU）都是如此。

某些应用领域已开始放弃浮点数据类型，例如深度学习推断工作负载会尽可能多地使用 INT8 或更低精度，这已经成为明显的行业趋势。

如今的设计环境极富挑战，散热和功耗要求越来越难以满足，因此，设计人员必须评估所有可能的方法来降低功耗。其中一种选择就是将浮点设计转换为定点。

如果是使用 C/C++，Vivado HLS 等赛灵思工具有助于简化转换过程。

设计人员必须对定点数据类型的转换进行充分权衡，并充分理解这样做所能带来的巨大优势。

继续使用浮点是通向市场的捷径，但是成本高。投入时间和精力转换成定点，这样能够降低资源使用量、成本和功耗，而且性能损失很小，从而获得巨大优势。

如需了解更多信息，敬请登录：china.xilinx.com 网站，访问 DSP 网页：
https://china.xilinx.com/products/technology/dsp.html

参考资料

1. Xilinx Landing Page *Vivado High-Level Synthesis*
2. Xilinx Landing Page *System Generator for DSP*
3. Gysel et al, *Hardware-oriented Approximation of Convolutional Neural Networks*, ICLR 2016
4. Han et al, *Deep Compression:Compressing Deep Neural Networks With Pruning, Trained Quantization And Huffman Coding*, ICLR 2016
5. *Deep Learning with Int8 Optimization on Xilinx Devices (WP486)*
6. Gupta et al, *Deep Learning with Limited Numerical Precision*
7. Ying Fai Tong et al, *Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic*
8. Github, location for the *design files*
9. Xilinx Software Manual, *Vivado Design Suite User Guide, High-Level Synthesis*

修订历史

下表列出了本文档的修订历史：

日期	版本	修订描述
2017 年 3 月 30 日	1.0	赛灵思初始版本

免责声明

本文向贵司 / 您所提供的信息（下称“资料”）仅在选择和使用赛灵思产品时供参考。在适用法律允许的最大范围内：(1) 资料均按“现状”提供，且不保证不存在任何瑕疵，赛灵思在此声明对资料及其状况不作任何保证或担保，无论是明示、暗示还是法定的保证，包括但不限于对适销性、非侵权性或任何特定用途的适用性的保证；且 (2) 赛灵思对任何因资料发生的或与资料有关的（含对资料的使用）任何损失或赔偿（包括任何直接、间接、特殊、附带或连带损失或赔偿，如数据、利润、商誉的损失或任何因第三方行为造成的任何类型的损失或赔偿），均不承担责任，不论该等损失或者赔偿是何种类或性质，也不论是基于合同、侵权、过失或是其他责任认定原理，即便该损失或赔偿可以合理预见或赛灵思事前被告知有发生该损失或赔偿的可能。赛灵思无义务纠正资料中包含的任何错误，也无义务对资料或产品说明书发生的更新进行通知。未经赛灵思公司的事先书面许可，贵司 / 您不得复制、修改、分发或公开展示本资料。部分产品受赛灵思有限保证条款的约束，请参阅赛灵思销售条款：http://china.xilinx.com/legal.htm#tos；IP 核可能受赛灵思向贵司 / 您签发的许可证中所包含的保证与支持条款的约束。安全保护功能，不能用于任何需要专门故障安全保护性能的用途。如果把赛灵思产品应用于此类特殊用途，贵司 / 您将自行承担风险和责任。请参阅赛灵思销售条款：http://china.xilinx.com/ legal.htm#tos。

汽车应用免责申明

汽车产品 (产品部件号中标识为“XA”) 不保证用于安全气囊的开发或用于影响车辆控制的应用 (“安全应用”), 除非在该赛灵思产品中具备故障安全保护或者额外功能，符合 ISO 26262 汽车安全标准 (“安全设计”)。为安全起见，客户应在使用或分销任何集成有该产品的系统之前，对这些系统进行全面测试。在没有安全设计的安全应用中使用产品的风险完全由客户承担，仅受有关产品责任的适用法律和法规限制。