

ZYNQ 实验报告

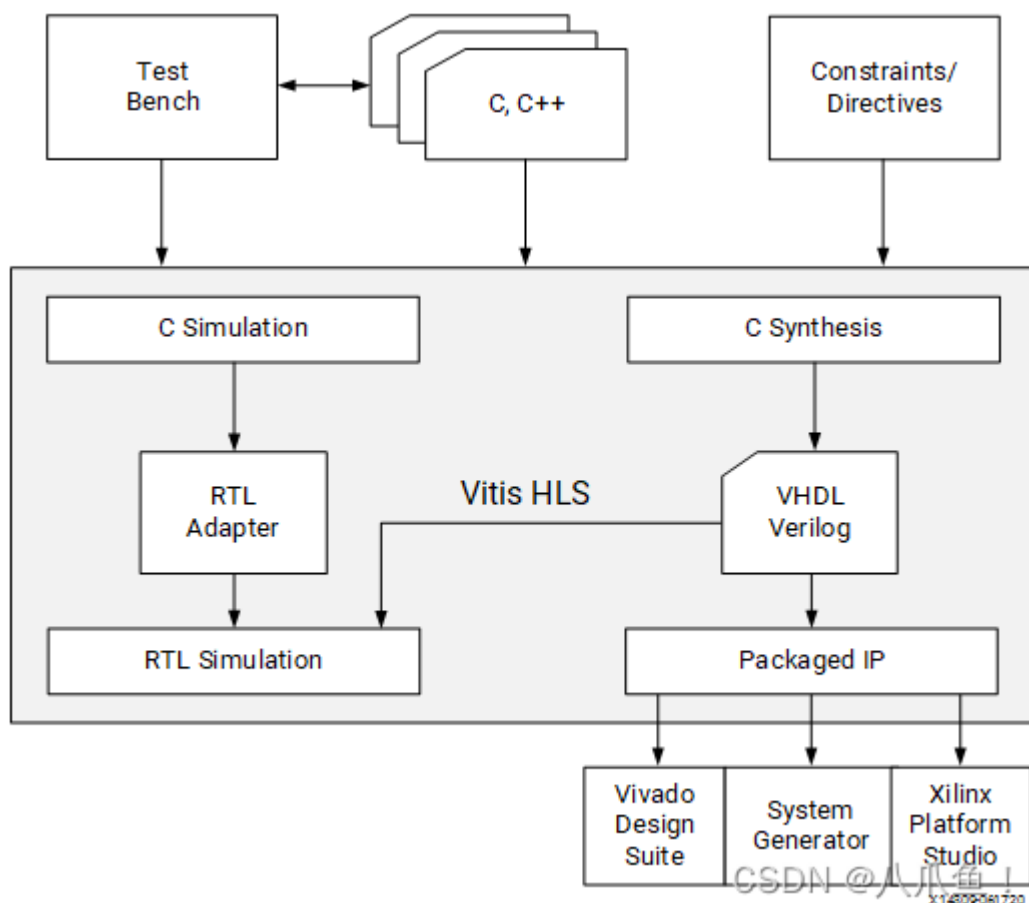
一、 Lab1: 设计一个 FIR 滤波器分离鸟类声音

1. 实验目标

- ①如何使用 Vitis HLS 构建一个项目；
- ②在 Vitis HLS 中进行仿真、综合与 IP 导出使用 Vivado 对 HLS 导出的 IP 进行集成；
- ③使用 PYNQ 构建一个简单的应用；

2. 实验流程

Vitis_HLS 设计流程:



Vitis_HLS 输入:

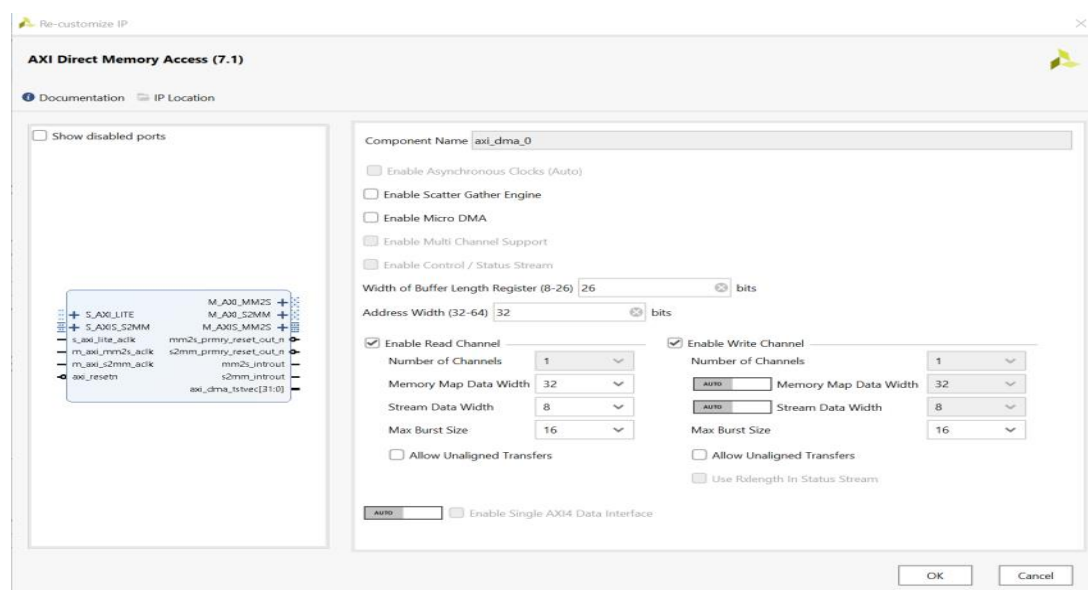
- * 用 C 或者 C++写的 C 函数。
- * 添加 RTL 黑箱函数中描述的带有 RTL 黑箱内容的 C 函数。
- * 指定时钟周期、时钟不确定性和设备目标的设计约束。
- * 指导合成过程实现特定的行为或优化的指令。
- * C 测试平台、在综合之前需要模拟 C 函数的相关文件，以及使用 C/RTL 联合仿真验证 RTL 的输出。

Vitis_HLS 输出:

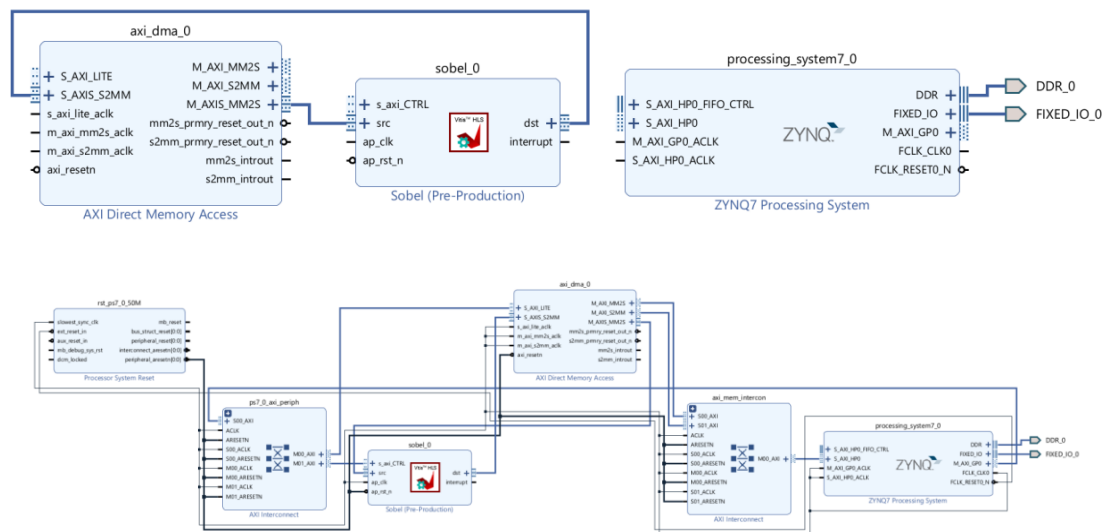
- * 编译的目标文件(.xo)。
- * 该输出允许您创建编译后的硬件函数，以便在 Vitis 应用程序加速开发流程中使用。当作为编译过程的一部分从 Vitis 工具流调用时，或者作为自底向上流中的独立工具调用时，Vitis HLS 会生成此输出。
- * 硬件描述语言(HDL)格式的 RTL 实现文件。
- * 由 Vitis HLS 生产的 RTL IP 可在 Verilog 和 VHDL 标准中使用，并且可以使用 Vivado 设计套件合成和实现到 Xilinx 设备中。
- * 报告文件。

3. 实验总结

在算法层面,高级语言通常比 HDL 开发效率更高, 因此使用 HLS 可以大大加速原型验证。高级语言进行程序设计可以脱离硬件, HLS 可以将项目的软件, 硬件开发分离, 更加灵活, 利于项目的管理。



* 连线&自动连接



* 导出比特流

3. 实验总结

* IP 核的重要性: Vivado 提供了许多 IP 核, 包括 DMA 核, 这些核能够极大地简化复杂的设计任务。通过使用现成的 IP 核, 我们可以节省大量的时间和精力, 同时确保设计的正确性和可靠性。

* IP 核的配置和连接: 学习如何配置和连接 IP 核是关键的一步。不同的 IP 核具有各自的参数和接口要求, 正确地配置和连接它们对于实现功能完整的设计至关重要。

* Block Design 视图的优势: 使用 Block Design 视图可以在图形化界面中直观地构建和管理设计。这种方式更易于理解和调整, 而且可以更快地完成设计原型的搭建。

* 边缘检测的重要性: 边缘检测在图像处理和计算机视觉领域中是一个重要的预处理步骤。它帮助我们捕捉图像中物体和纹理之间的边界, 从而更好地理解图像

的结构和内容。

* Sobel 算子的原理：Sobel 算子是一种常用的离散差分算子，用于计算图像中每个像素点处的梯度强度。通过对图像进行卷积操作，我们可以得到 X 和 Y 方向的梯度，然后将其合并得到综合的边缘强度图像。

* 处理 X 和 Y 方向的梯度：在 Sobel 算子中，通过分别使用横向和纵向的差分核来计算 X 和 Y 方向的梯度。对 X 和 Y 梯度取绝对值后，可以将其组合成一个综合的边缘图像，这样可以更好地捕捉图像中不同方向的边缘。

* 阈值处理：生成的边缘图像可能包含很多细微的边缘线条和噪点。在实际应用中，常常需要应用阈值处理来进一步筛选和增强图像中的边缘信息。

* 学会可视化：通过可视化原始图像、X 和 Y 方向的梯度以及最终的边缘图像，我更好地理解 Sobel 算子的作用和效果。可视化是学习图像处理算法的重要方法，它有助于直观地理解算法的工作原理。

三、 Lab3: MNIST 分类器

1. 实验内容

Vitis AI 是 Xilinx 开发的深度学习推理工具, 用于在 Xilinx FPGA 和 ACAP (Adaptive Compute Acceleration Platform) 设备上进行深度学习推理。

- * 安装 Vitis AI
- * 准备模型: 使用 Vitis AI 进行深度学习推理, 首先需要准备一个训练好的深度学习模型。Vitis AI 支持常见的深度学习框架, 如 TensorFlow 和 Caffe。可在 Vitis AI 中导入模型或使用支持的转换器将模型从其他框架转换为 Vitis AI 支持的格式。
- * 量化和优化模型: 为了在 FPGA 上进行高效的推理, 你可能需要量化和优化模型。量化可以将浮点模型转换为定点模型, 从而减少模型大小和计算量。优化可以针对目标硬件平台优化模型结构和参数。
- * 生成 DPU 文件: 使用 Vitis AI 工具链编译和生成 DPU (Deep Learning Processor Unit) 文件。DPU 是用于在 FPGA 上执行深度学习推理的特定硬件加速器。
- * 集成到 Vivado 或 Vitis 项目: 将生成的 DPU 文件集成到 Vivado 或 Vitis 项目中。这涉及在 Vivado 或 Vitis 中添加 DPU IP 核, 并将其与其他硬件模块进行连接。
- * 编写主机代码: 为了在 FPGA 上运行深度学习推理, 需要编写主机代码来控制和管理 DPU。主机代码可以运行在嵌入式系统或 PC 上。
- * 执行推理: 运行主机代码来初始化 DPU 并加载量化和优化后的模型。然后, 使用输入数据进行推理, 并获取输出结果。
- * 后处理和结果解释: 根据模型的输出, 进行后处理和结果解释。根据实际应用, 可能需要对输出进行解码、过滤或后处理。

2. 实验心得

我觉得学好 Vitis AI，需要学习以下内容：

要使用 Vitis AI 进行深度学习推理，你需要掌握以下一些关键内容：

- * 深度学习基础：理解深度学习的基本概念、神经网络结构（如卷积神经网络、循环神经网络）以及常见的深度学习任务（如分类、目标检测、分割等）。学习如何训练和优化深度学习模型，以及模型的评估和调优方法。
- * 深度学习框架：掌握至少一种常见的深度学习框架，如 TensorFlow、PyTorch 或 Caffe。了解如何构建、训练和导出深度学习模型，并熟悉框架提供的工具和功能。
- * FPGA 基础：了解 FPGA 的基本原理和体系结构。学习 FPGA 开发流程、硬件描述语言（如 VHDL 或 Verilog）以及 FPGA 工具链（如 Vivado）的使用。
- * Vitis 工具链：熟悉 Xilinx Vitis 工具链的使用，包括 Vivado 和 Vitis AI。学习如何创建和管理项目，进行综合、布局和布线，以及生成比特流文件和可执行文件。
- * Vitis AI 工具链：深入了解 Vitis AI 的工具链，包括模型转换器、量化工具和编译器。学习如何将深度学习模型转换为 DPU 支持的格式，并进行量化和优化，以达到在 FPGA 上进行高效推理的目的。
- * DPU 架构和编程模型：了解 Deep Learning Processor Unit (DPU) 的架构和编程模型。研究 DPU 的特性、指令集和硬件加速功能。学习如何在主机代码中初始化和控制 DPU，并进行数据传输和推理操作。
- * 性能优化和调试：学习如何优化深度学习模型的性能，以及如何调试和排查推理过程中的问题。掌握优化技术，如模型压缩、硬件/软件协同设计和并行计算等。

四、 自主实验：矩阵乘法加速器（与张文瑞同学共同完成）

1. 实验内容

- * 在 Vivado HLS 中生成矩阵乘法加速的 IP 核。
- * 在 Vivado 中完成 Block Design。
- * 在 Jupyter Notebook 上完成 IP 的调用。

2. 实验步骤

（代码文件已上传至 Github）

```
void matrixmul(
    mat_a_t a[SIZE],
    mat_b_t b[SIZE],
    result_t res[SIZE])
{
    int tempA[MAT_A_ROWS][MAT_A_COLS];
    int tempB[MAT_B_ROWS][MAT_B_COLS];
    int tempAB[MAT_A_ROWS][MAT_B_COLS];

    for (int ia = 0; ia < MAT_A_ROWS; ia++){
        for(int ja = 0; ja < MAT_A_COLS; ja++){
            tempA[ia][ja] = a[ia*MAT_A_ROWS+ja].data;
        }
    }
    for (int ib = 0; ib < MAT_B_ROWS; ib++){
        for(int jb = 0; jb < MAT_B_COLS; jb++){
            tempB[ib][jb] = b[ib*MAT_A_ROWS+jb].data;
        }
    }
    /* for each row and column of AB */

    row: for(int i = 0; i < MAT_A_ROWS; ++i) {

        col: for(int j = 0; j < MAT_B_COLS; ++j) {

            /* compute (AB)i,j */

            int ABij = 0;

            product: for(int k = 0; k < MAT_A_COLS; ++k) {
                ABij += tempA[i][k] * tempB[k][j];
            }
            tempAB[i][j] = ABij;
        }
    }

    for (int iab = 0; iab < MAT_A_ROWS; iab++){
        for(int jab = 0; jab < MAT_B_COLS; jab++){
            res[iab*MAT_A_ROWS+jab]=push_stream<int>(tempAB[iab][jab],iab==(MAT_A_ROWS-1)&&jab==(MAT_B_COLS-1));
        }
    }
}
```

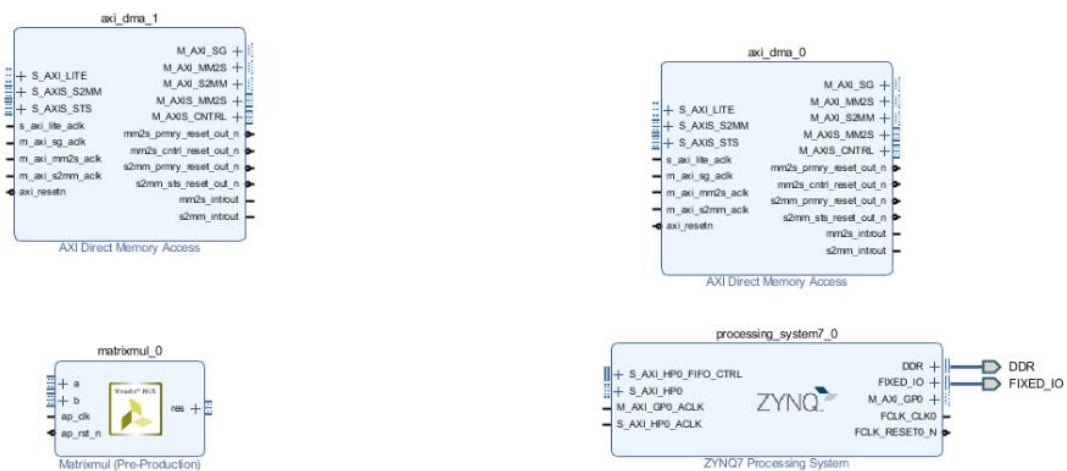

①Vivado HLS 生成矩阵乘法加速 IP

1) 在约束文件中添加接口约束和循环流水, 实现矩阵乘法的硬件加速

(IP 核使用 ap_ctrl_none 接口协议, 输入的 a,b 矩阵和输出的 res 矩阵均使用 axis 流数据, 注意流数据传输中 Block Design 中需添加 DMAip 核进行数据格式转换)

2) 导出 IP 核

②Vivado 进行 Block Design



③Jupyter Notebook 进行矩阵乘法加速 IP 的调用

3. 开发框架

* PYNQ 软件开发:

在 Python 中使用 PYNQ 库, 通过软件驱动来访问和控制 FPGA 上的硬件模块。

编写 Python 函数, 将输入数据传递给 FPGA 中的矩阵乘法加速器, 然后获取计算结果。

* Jupyter Notebook:

在 PYNQ 平台上, 您可以使用 Jupyter Notebook 来开发和测试矩阵乘法加速器

的功能。Jupyter Notebook 是一个交互式的开发环境，可以轻松编写和运行 Python 代码，并且与 PYNQ 完美结合。

4. 开发感想

当真正实操的时候，我发现很多内容都需要考虑到：

- * 仔细规划和设计硬件计算单元，包括乘法器和加法器的数量和布局，数据流和存储器的管理，以及高效的并行计算。
- * 考虑数据传输和接口设计，确保有效地将数据从主机系统传输到 FPGA，并将结果传回主机系统。
- * 进行性能优化，包括利用 FPGA 的并行性和流水线设计，以实现更高的运算速度和吞吐量。
- * 进行验证和测试，确保加速器在各种情况下都能正确运行，并与主机系统进行正确的数据交互。