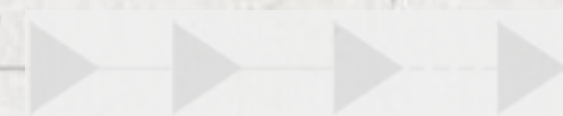



芯动力——硬件加速设计方法

第六章 FPGA硬件加速案例

邸志雄@西南交通大学

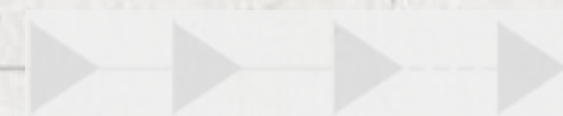
zxdi@home.swjtu.edu.cn





基于Xilinx PYNQ平台的深度学习激活函数 SoftMax设计优化及实现

本项目工程文件已在github共享 https://github.com/9334swjtu/PYNQ_softmax



Xilinx

ZYNQ

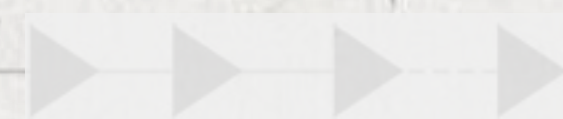
- FPGA中嵌入了一颗双核ARM
 - 发挥FPGA的定制特性
 - 发挥处理器的通用特性

接口丰富

官方文档齐全

可玩性极高

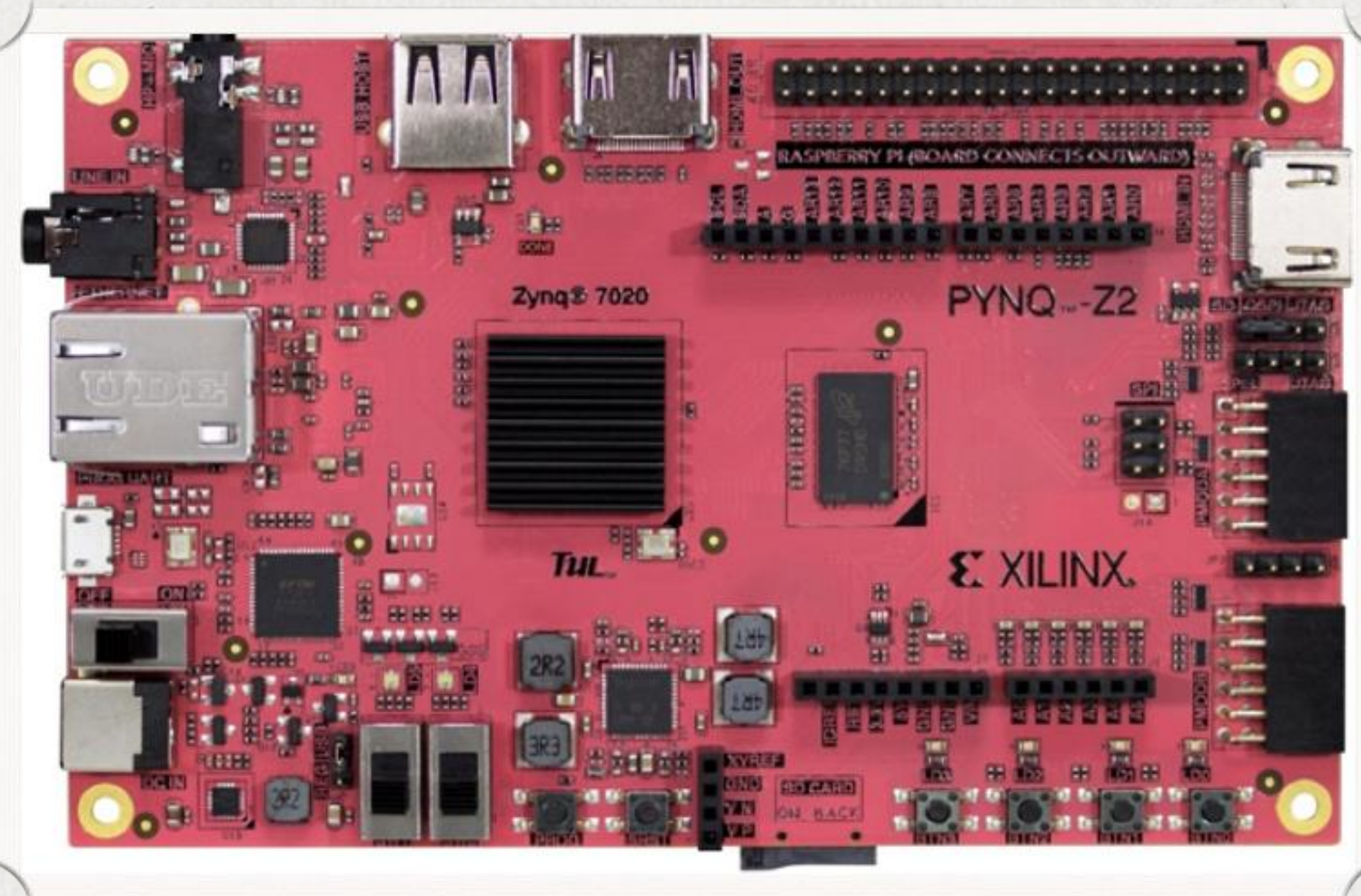
- 传统的ZYNQ开发板（Zedboard等）需要结合vivado和SDK两大工具分别对PL、PS端进行硬件和软件开发。
- PYNQ开发板可使用Python将软件操作和硬件控制进行无缝衔接，目前已经出到Z2版本。



PYNQ

Xilinx

- 旨在使用Python让SoC开发更加简单的项目
- 传统的ZYNQ开发板也能使其变为PYNQ开发板，当引入Python后就可以使用一些Python中强大的第三方库，例如Numpy、Matplotlib等。



PYNQ社区<http://www.pynq.io/>

入门操作https://pynq.readthedocs.io/en/latest/getting_started.html



一、工作概述

二、总体设计

三、方案分析

四、详细设计方案

五、测试与性能评估

六、创新点总结



一、工作概述

参赛题目： 激活函数SoftMax 进行设计优化并实现

$$S_i = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$$

- 支持AXI总线, DDR中读数
- 定点数输入输出, 支持8/16bit数据
- 最大输入范围 [-10, 10]

主要完成工

作:

- 所有电路均采用可综合 VerilogHDL 代码描述, 未使用FPGA 工具内置的IP 核
- 采用了ZYNQ、SIMC65nm标准单元工艺库分别完成了功能验证和性能评估

本项目工程文件已在github共享 https://github.com/9334swjtu/PYNQ_softmax

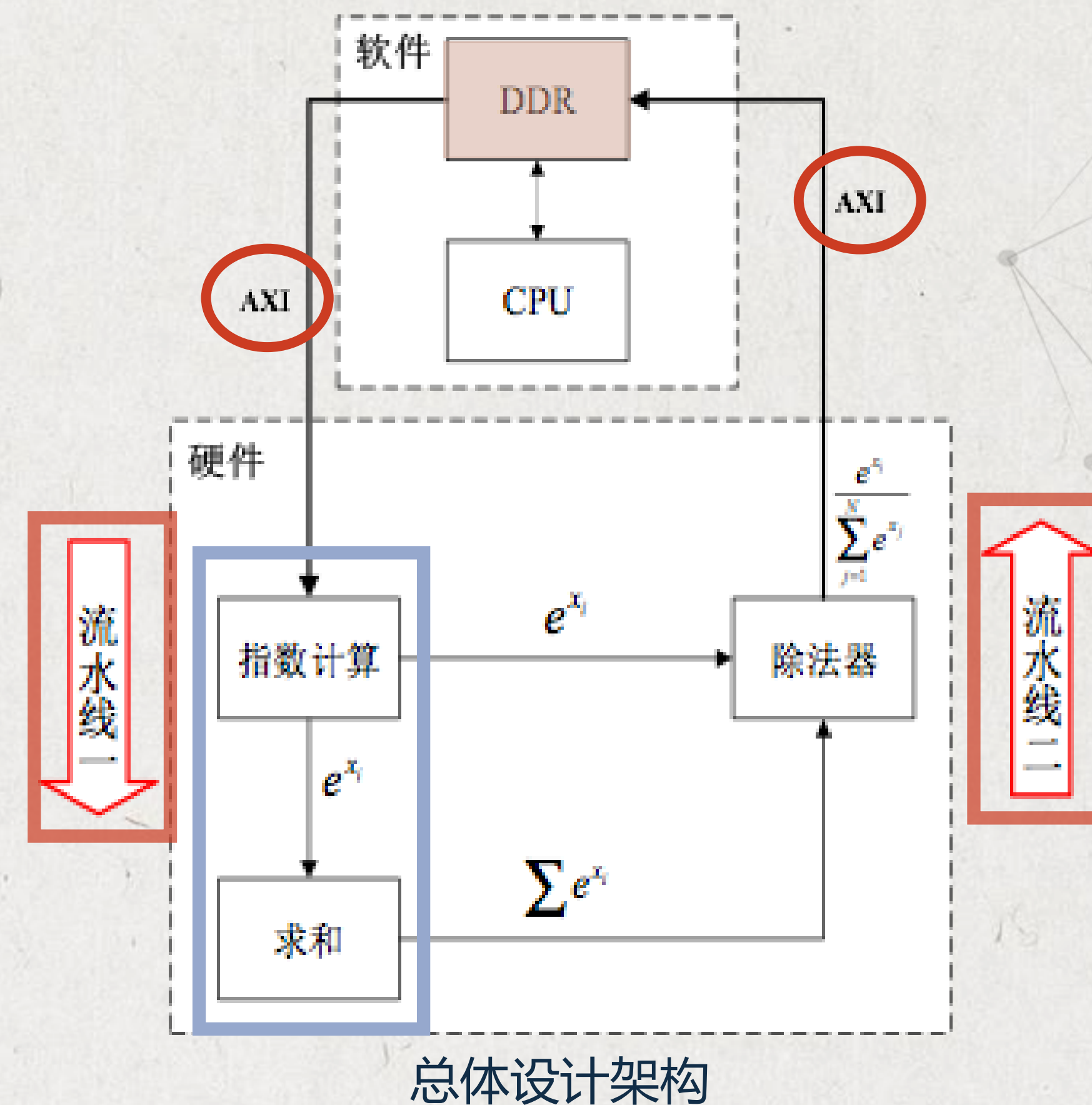
二、总体设计

软件: 在CPU中对DDR数据进行预处理, 将浮点数转化为定点数。

硬件: 完成SoftMax函数的定点数计算

$$S_i = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}} = \frac{e^{x_i - x_{max}}}{\sum_{j=1}^N e^{x_j - x_{max}}}$$

- 硬件结构采用了基底查表法, 对指数函数划分了5个基底, 从对应ROM表中查出, 用乘法器得到 e^{x-max} 。然后使用移位减除法器得到SoftMax函数最终值。



三、现有方案对比分析

查找表法:

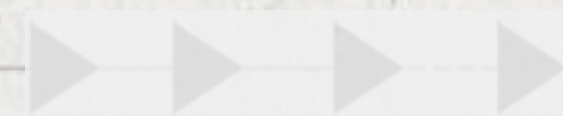
- 提前将所有的计算结果保存在一个ROM中。此方法计算量很小，速度快。但随着函数计算区间的增大，查找的ROM表将会十分巨大，十分浪费资源。

CORDIC算法:

- 即坐标旋转数字计算法。通过多次迭代将一些复杂的运算转换为简单的运算。但随着精度的增高，其算法迭代次数会大幅度增加，计算速度会降低。

Talyor级数展开法:

- 将SoftMax函数采用泰勒级数转化为多项式，将系数保存在ROM中。这种方法结合了前两种方法，但在精度要求较高的情况下，乘法器和ROM将会消耗大量的资源。



三、现有方案对比分析

- 本作品针对指数计算的特性，设计了一种原理简单的计算方案，根据其计算特性称其为**基底查表法**。

$$e^i = e^{ax} * e^{by} * e^{cz}$$

二进制

十进制

3.68

e^3

$e^{0.6}$

$e^{0.08}$

基底拆分

个位

十分位

百分位

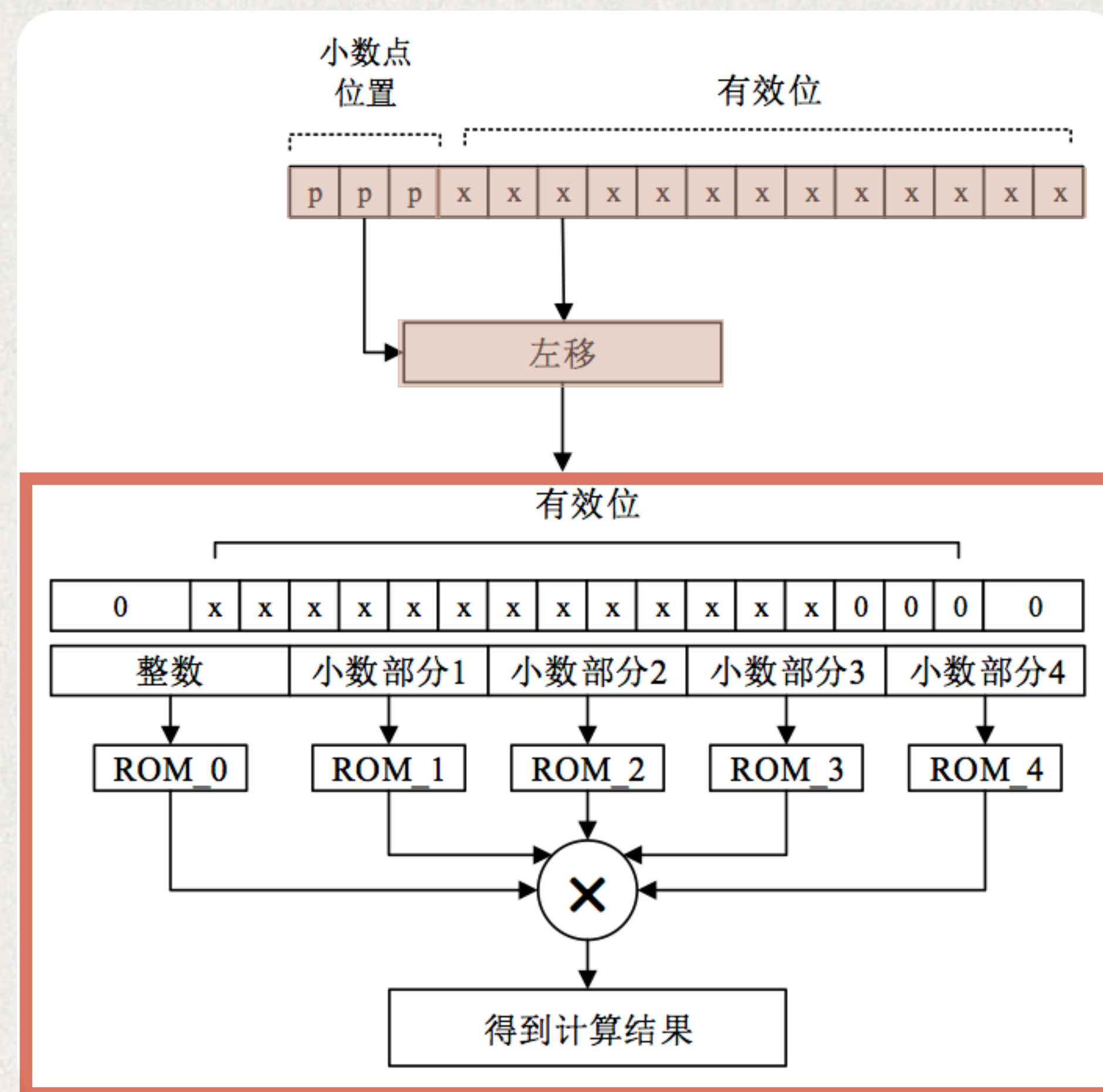
- 并作为ROM的寻址地址，最后将查表得到的值进行乘法操作即可得到最终结果。

三、现有方案对比分析

- 待计算的16bit数据，高三位表示的是小数点的位置信息，低十三位为数据位。经过左移并按照人为设计的格式输出即可还原数据。

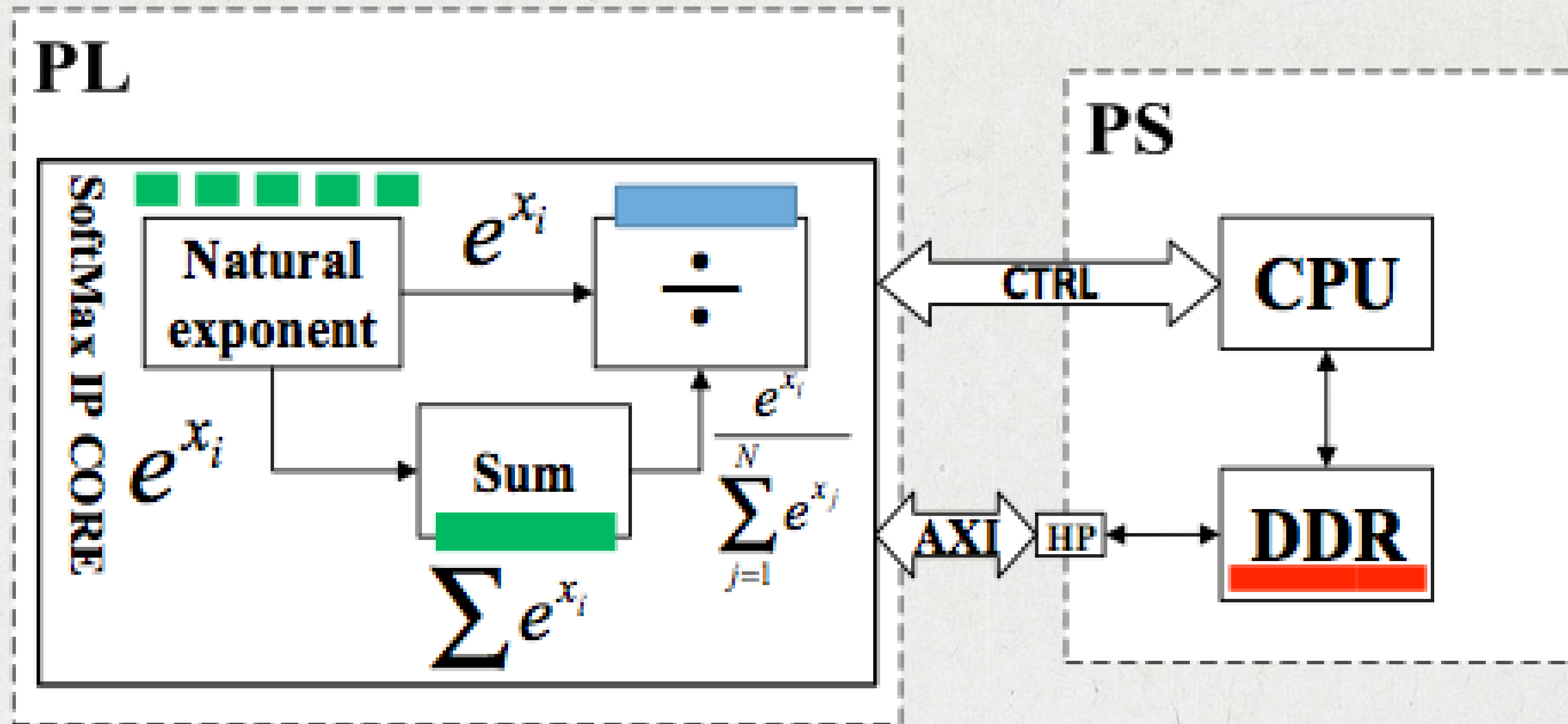
采用移位操作进行基底划分

- 硬件结构简单、传播延时小
- 直接存放 e^i 的值，计算精度损失很小



基于二进制移位操作的基底拆分示意图

四、详细设计方案



总体结构设计

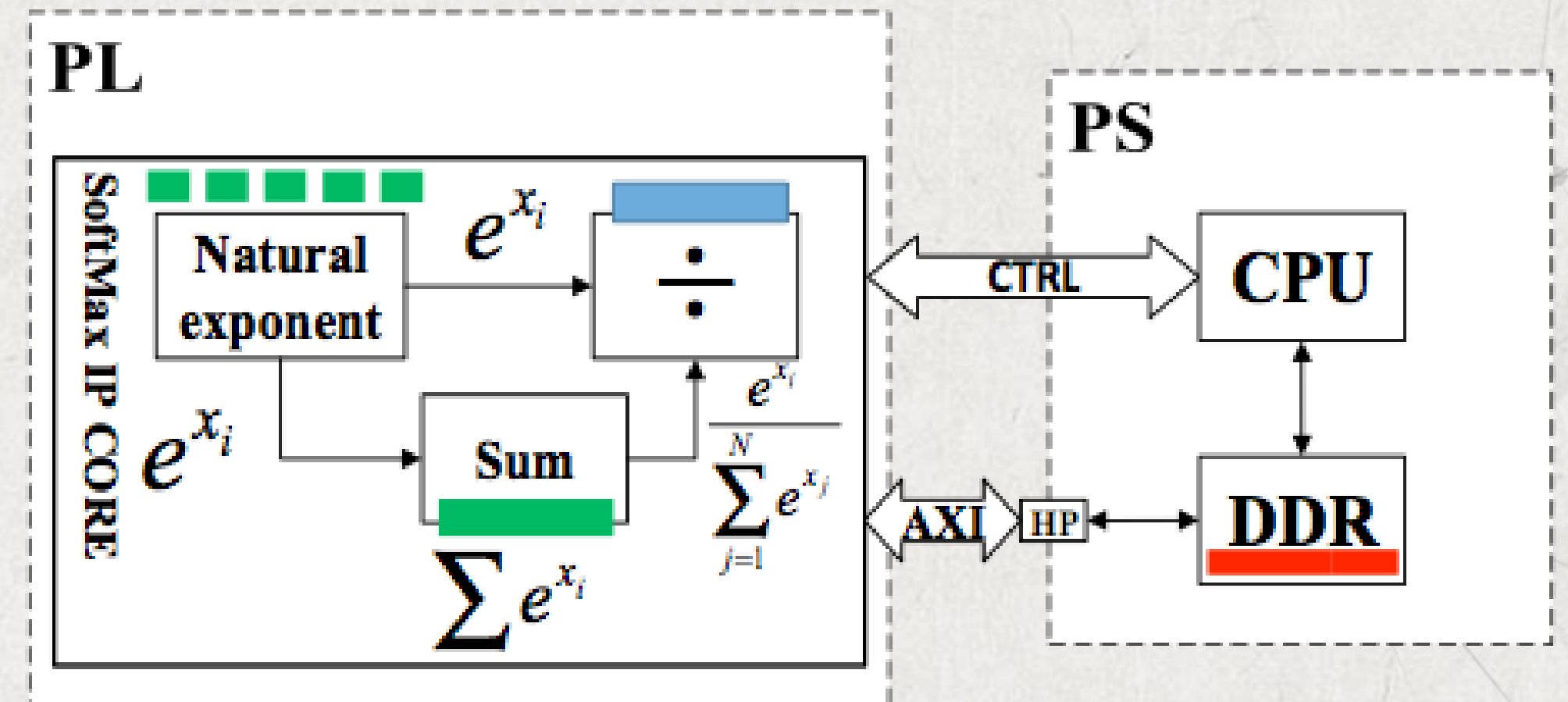
四、详细设计方案

软件部分:

- ①遍历, 求DDR中最大值 x_{\max} , 计算 $|x_i - x_{\max}|$
- ②将 $x_i - x_{\max}$ 进行定点化处理
- ③结果写入DDR

硬件部分:

- ④定点化数据查找表, 查得的5个数据相乘得到 $\alpha_i = e^{x_i - x_{\max}}$
- ⑤将计算得到的结果 α_i 相加, 得到累加值 $\sum_{j=1}^N e^{\alpha_j}$
- ⑥采用移位减除法器进行除法运算, 得到SoftMax的结果
- ⑦SoftMax结果写入DDR



总体结构设计

四、详细设计方案

- 软件预处理

硬件电路计算之前，先由软件进行数据预处理：

符号统一化

数据定点化

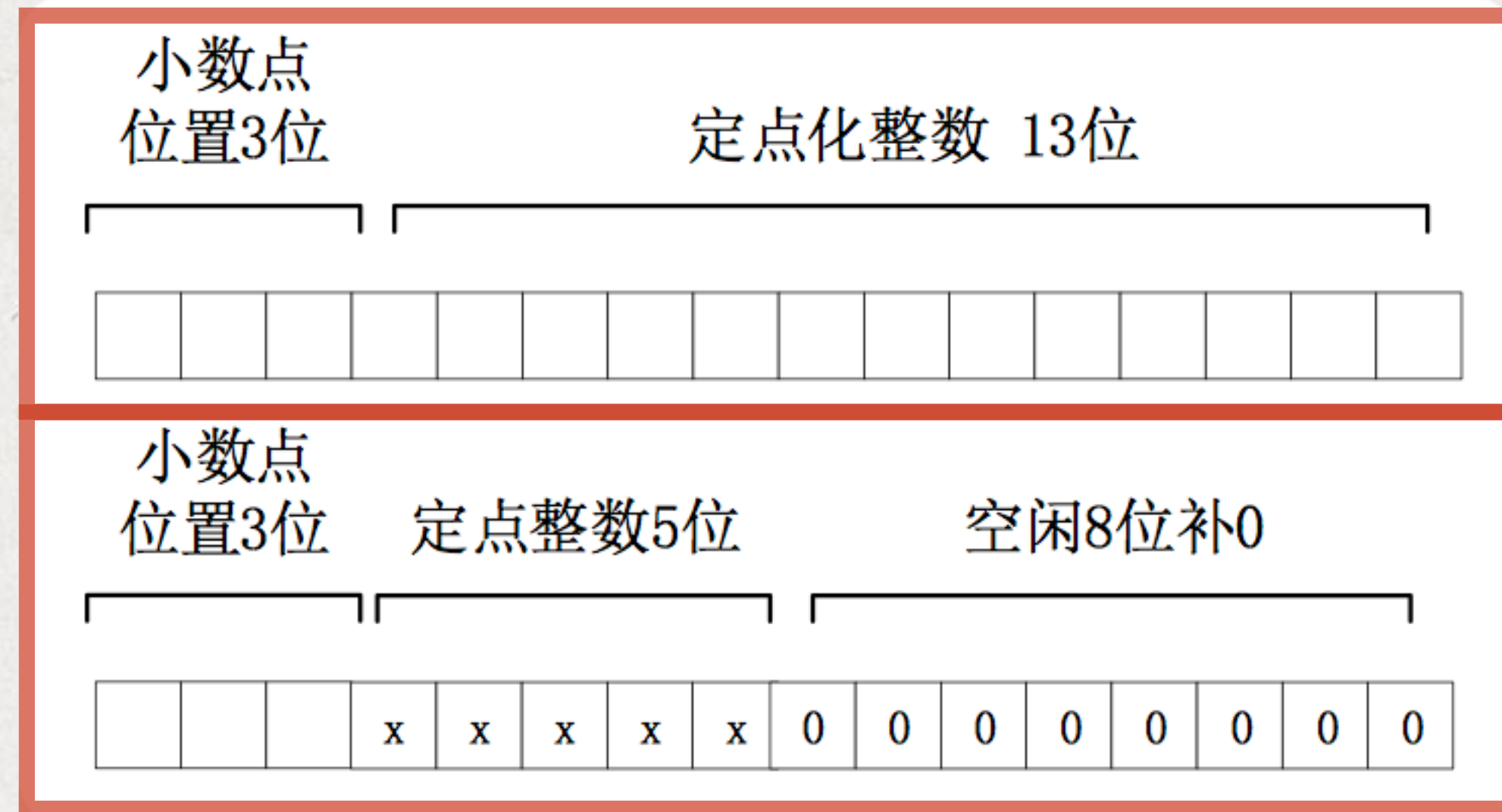
定点化方案

- AXI 输入的数据为定点数，总位宽为32bit，可分为两组，16bit为一组（兼容8bit，此时低8位补零）；
- 16位的最高3位为小数点位置保存位。
- 经CPU 处理后， $\alpha_i = x_i - x_{max}$ 范围为-20~0，小数点位置只可能是0~5，故用16位的高3位保存小数点位置，其余13 位为有效数字。
- 在进行符号统一化之后，所有数值均为负数，因此可以省略定点数的符号位，其大小等于原数值的绝对值。

四、详细设计方案

- 软件预处理

定点化方案



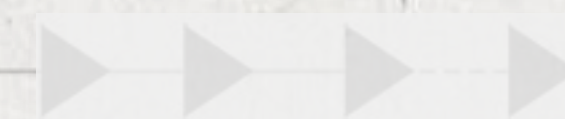
16/8位定点数划分方案

浮点数
7.3

二进制表示为 $(111.0100110011001)_2$

小数点位于第三位后

- 16bit定点化结果为: 011-1110100110
- 8bit定点化结果为: 011-11101



x-max

$$e^{x-\max} \leq 1$$



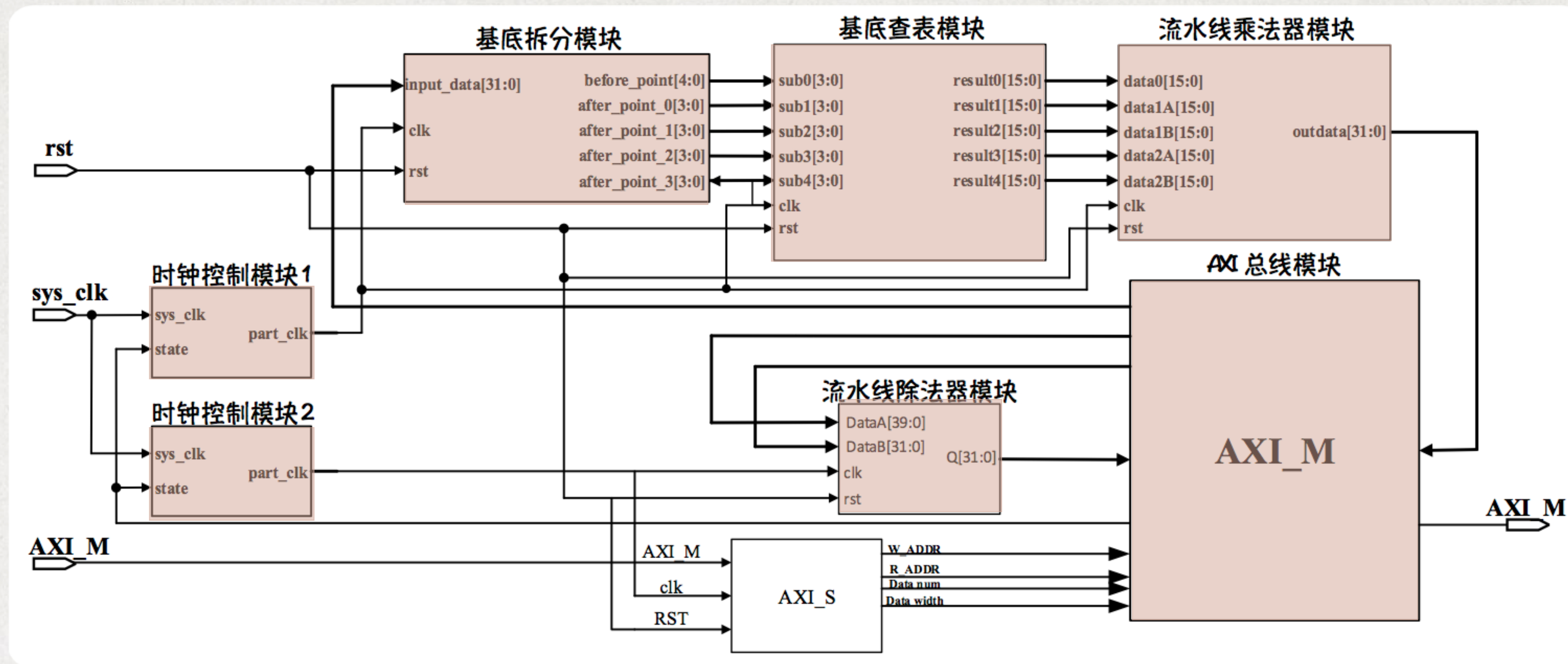
16bit

1bit整数位

15bit小数位

四、详细设计方案

- 硬件部分



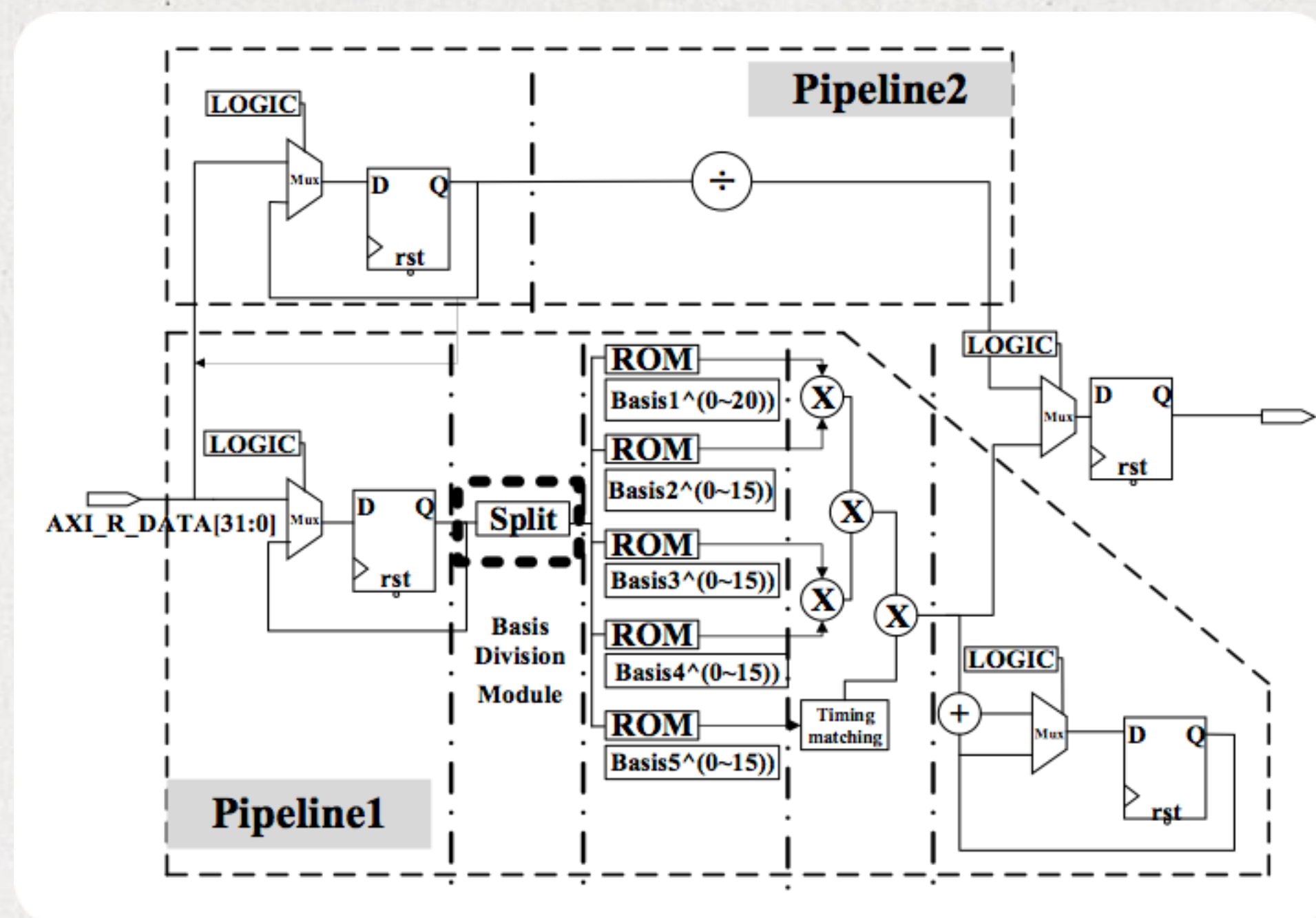
硬件部分整体框架图

四、详细设计方案

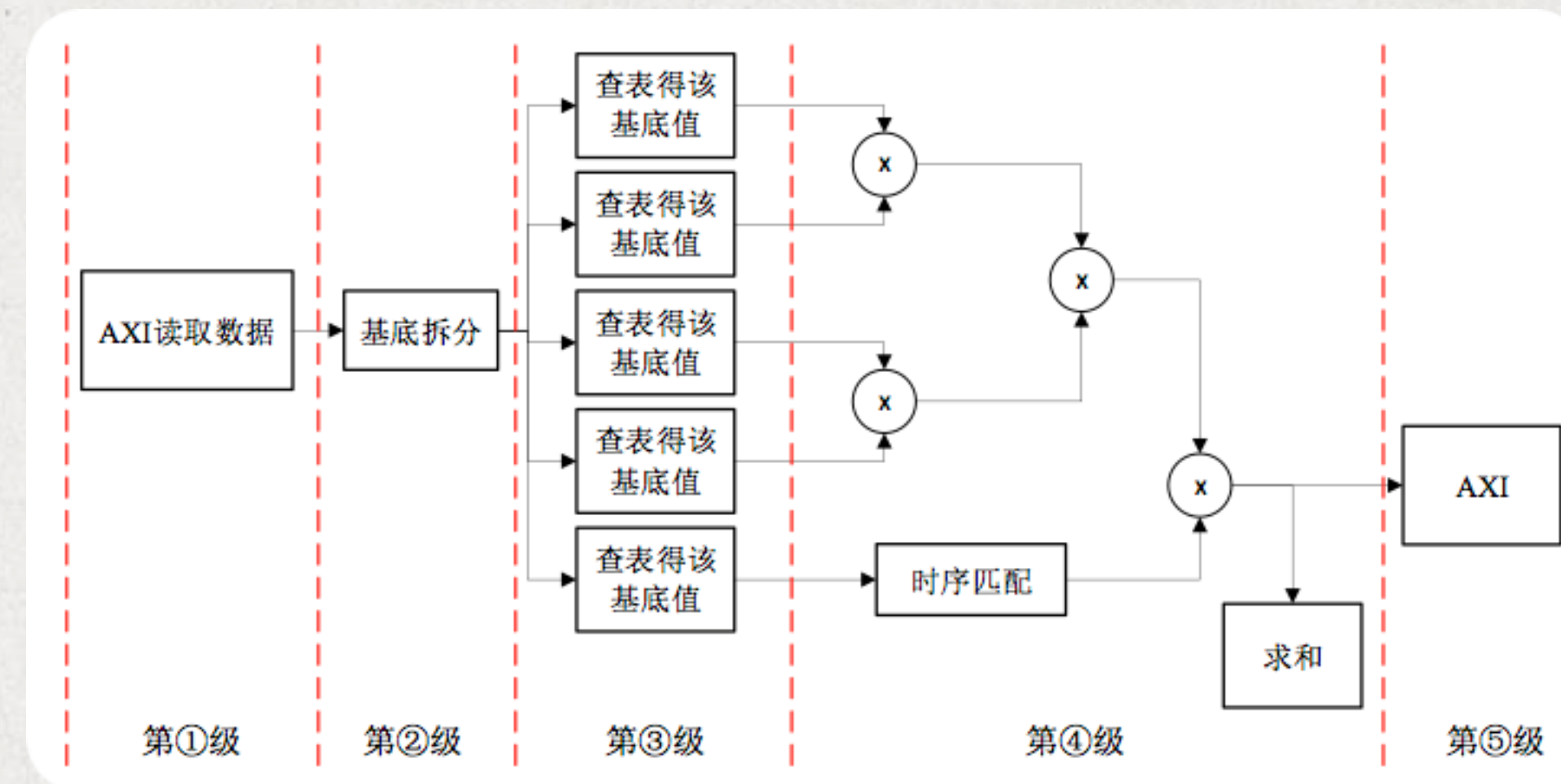
- 硬件部分

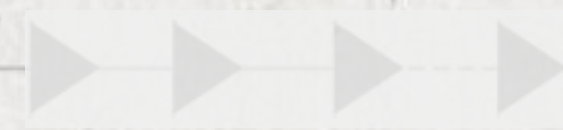
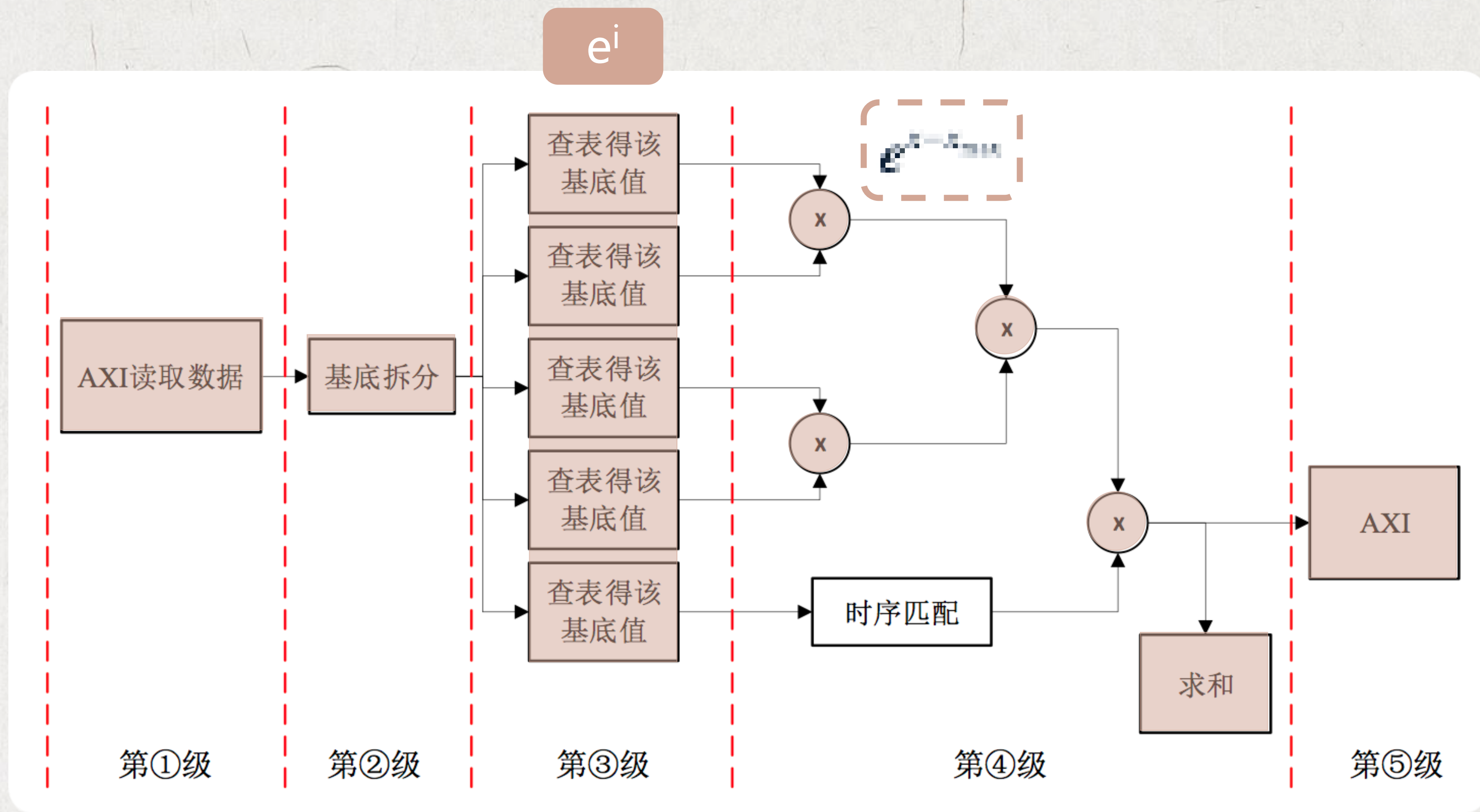
流水线1: 计算 e^x 并求和 $\sum e^x$

流水线2:



硬件部分流水线结构



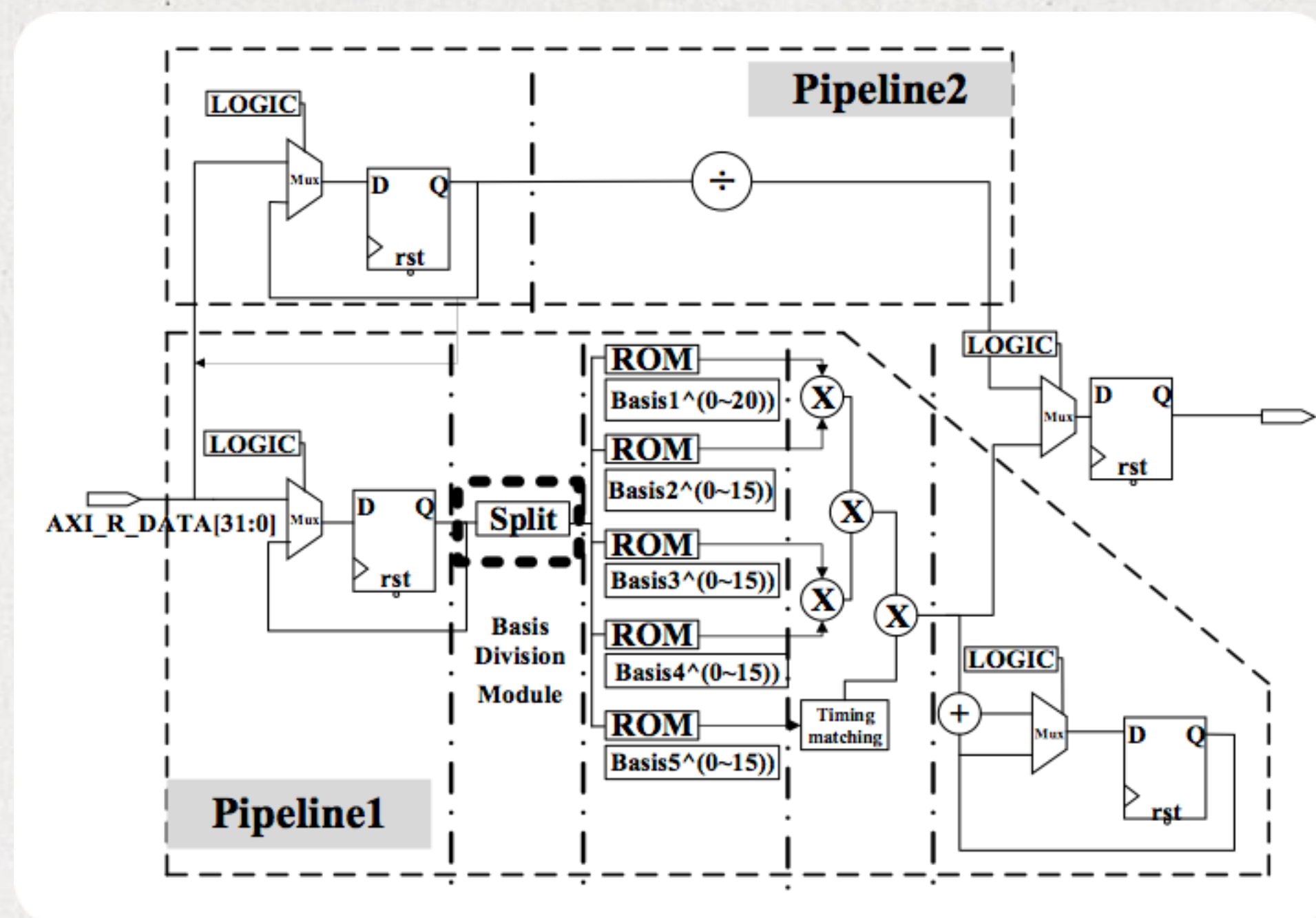


四、详细设计方案

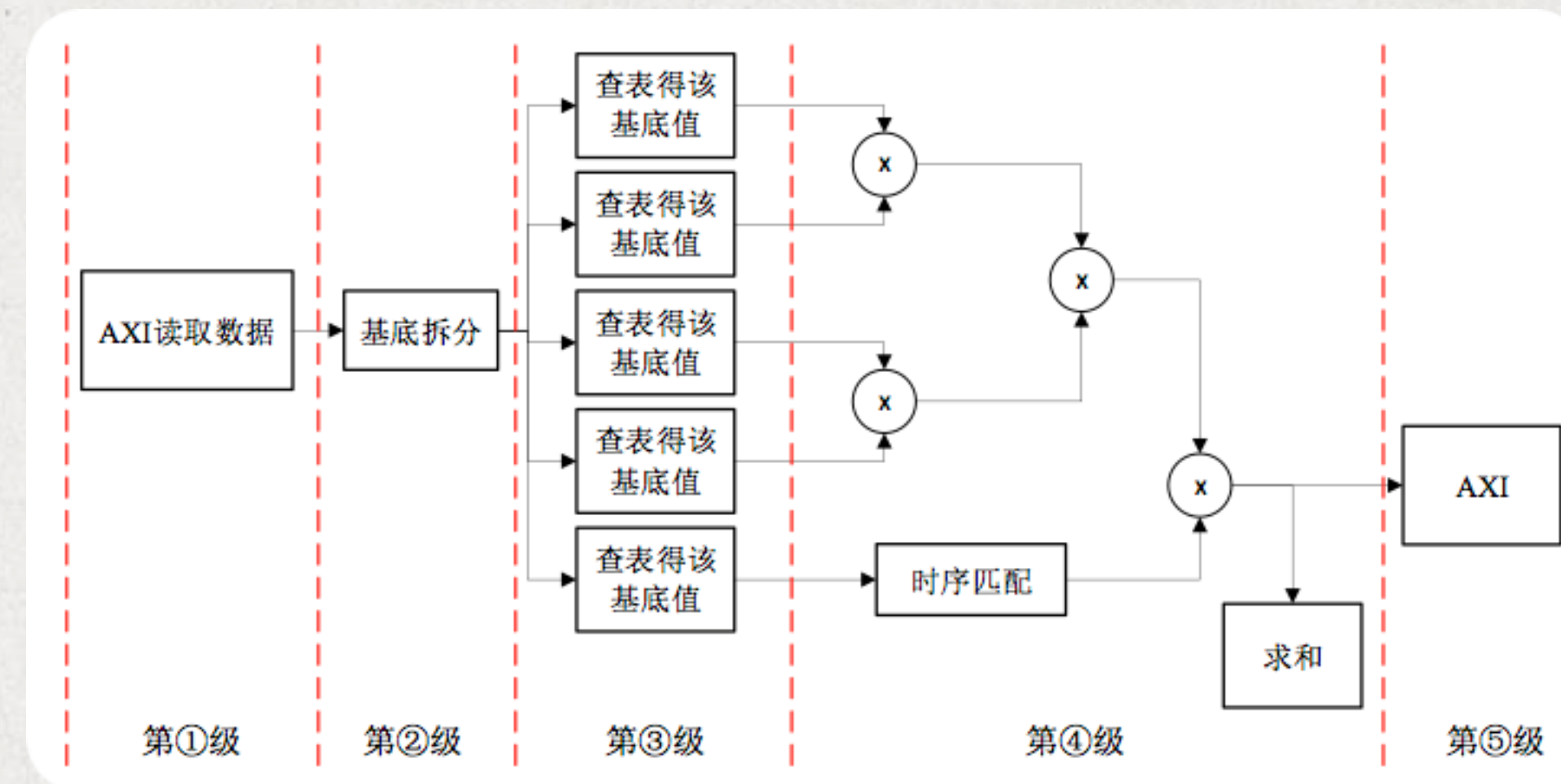
- 硬件部分

流水线1: 计算 e^i 并求和 $\sum e^i$

流水线2: 除法运算, 计算 $e^i / \sum e^i$

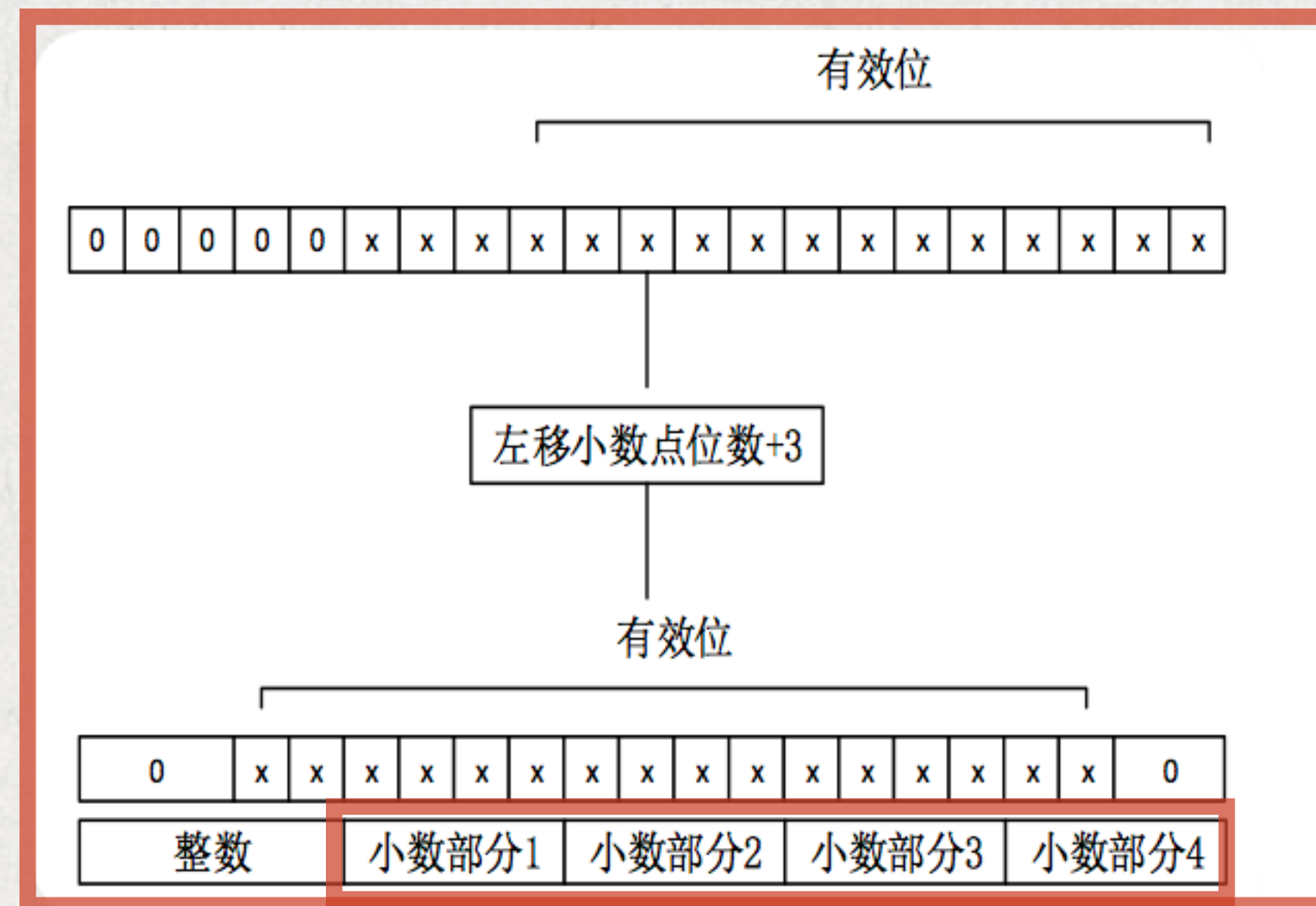


硬件部分流水线结构

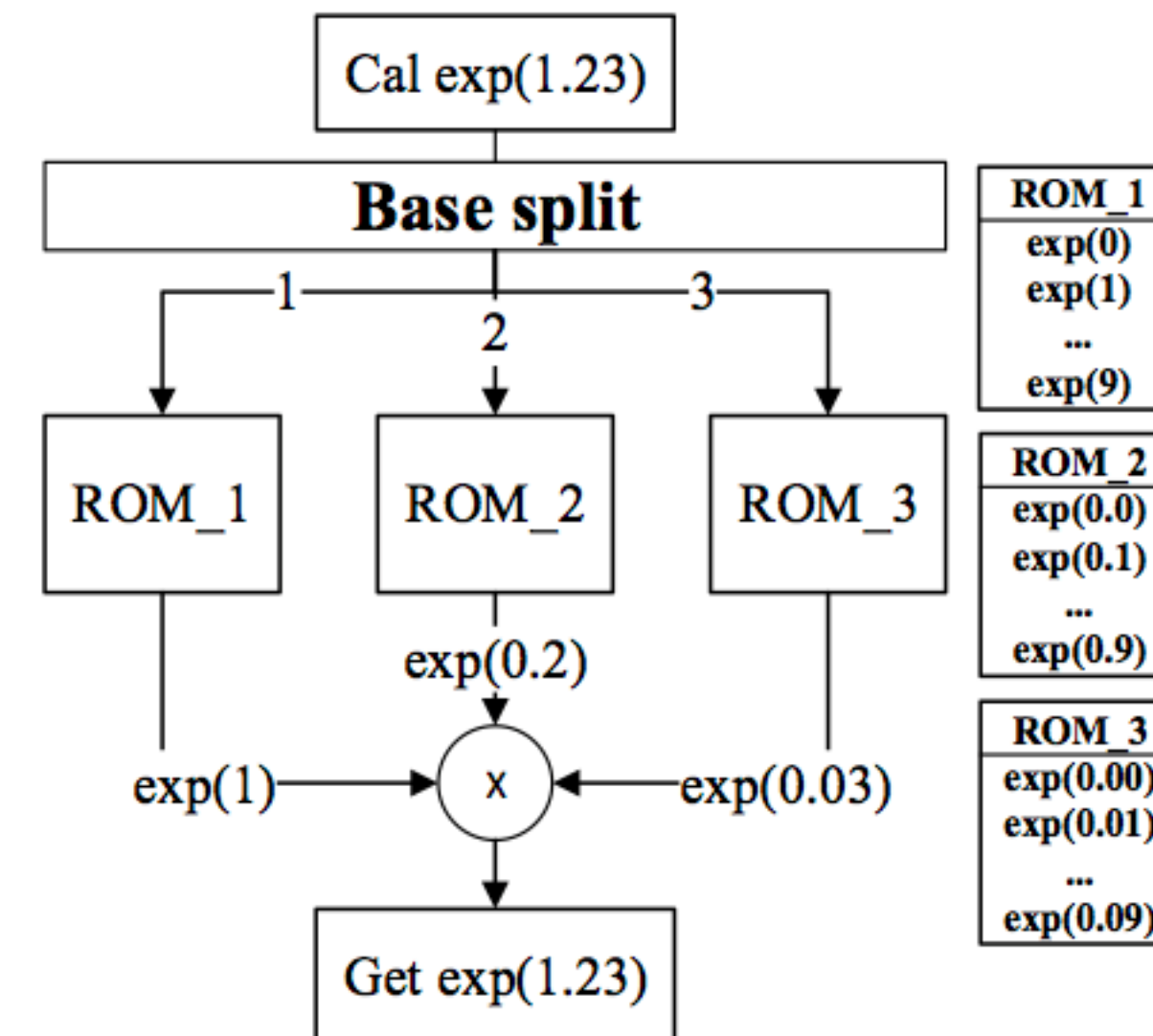


四、详细设计方案

- 硬件部分：基底划分模块



基底拆分过程



AXI读入的数据

定点数的范围是
[0, 20]

- 低13bit为定点数据

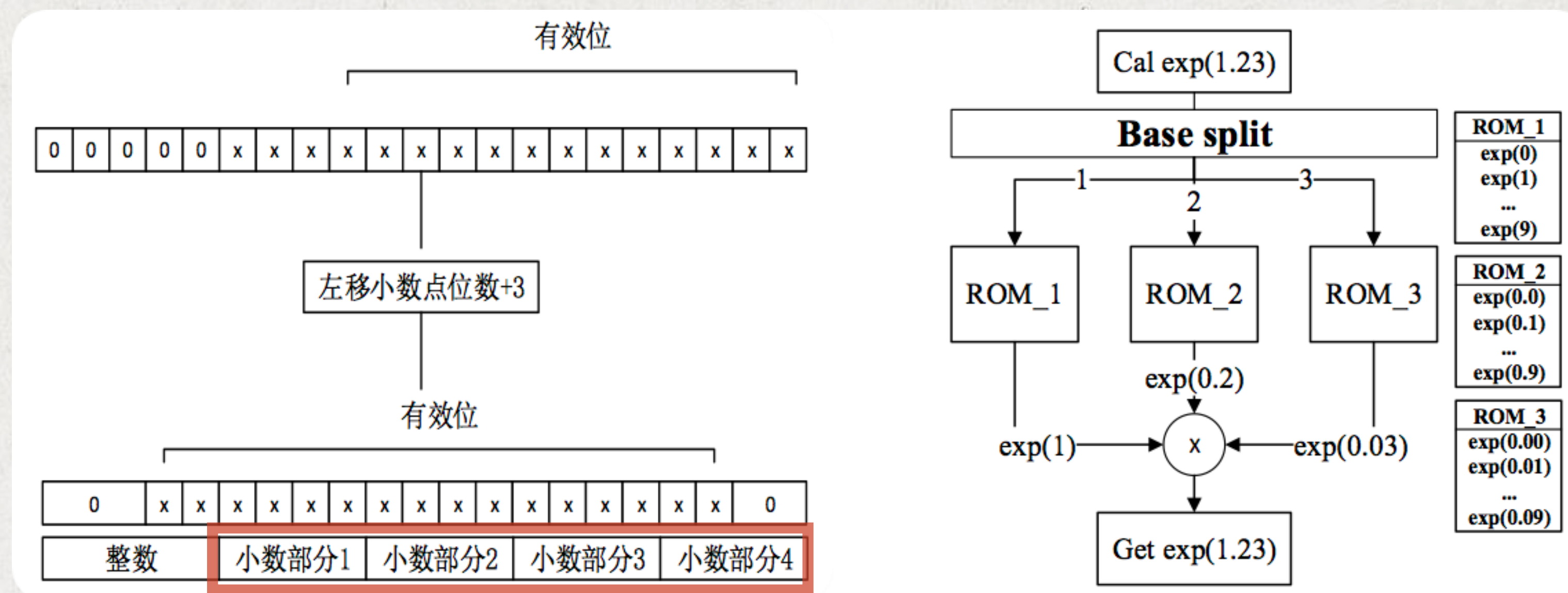
- 整数部分位宽最大为5bit

- 最高3bit表示小数点位置

- 小数部分位宽最大为16bit

四、详细设计方案

- 硬件部分：基底划分模块



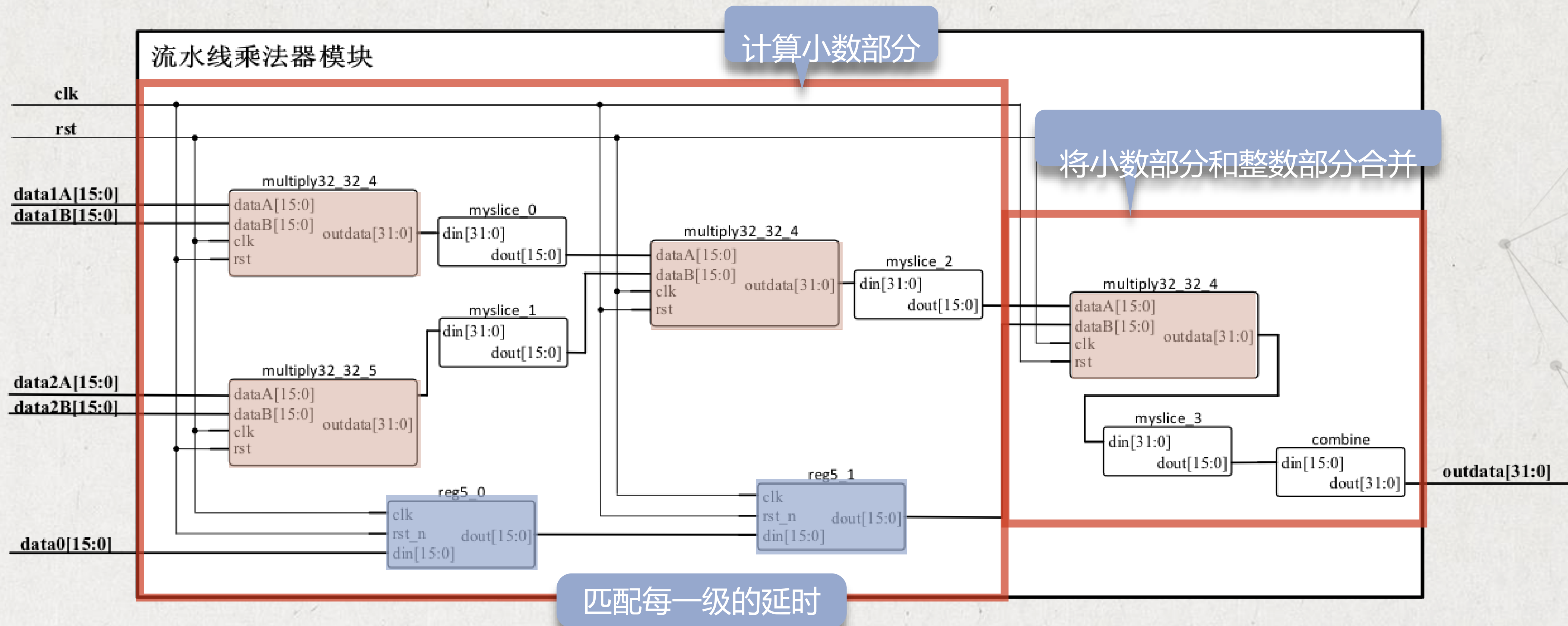
基底拆分过程

整数部分划分为一组 位宽5bit 小数部分划分成4组 每组4bit

- 本文方案采用了基底拆分的思想，如果直接查找表，则需要2097152个数据。将数据分组进行查表，可以减少查找表面积，只需要 $2^5 + 2^4 \times 4 = 96$ 个数据。极大的节省了存储空间。

四、详细设计方案

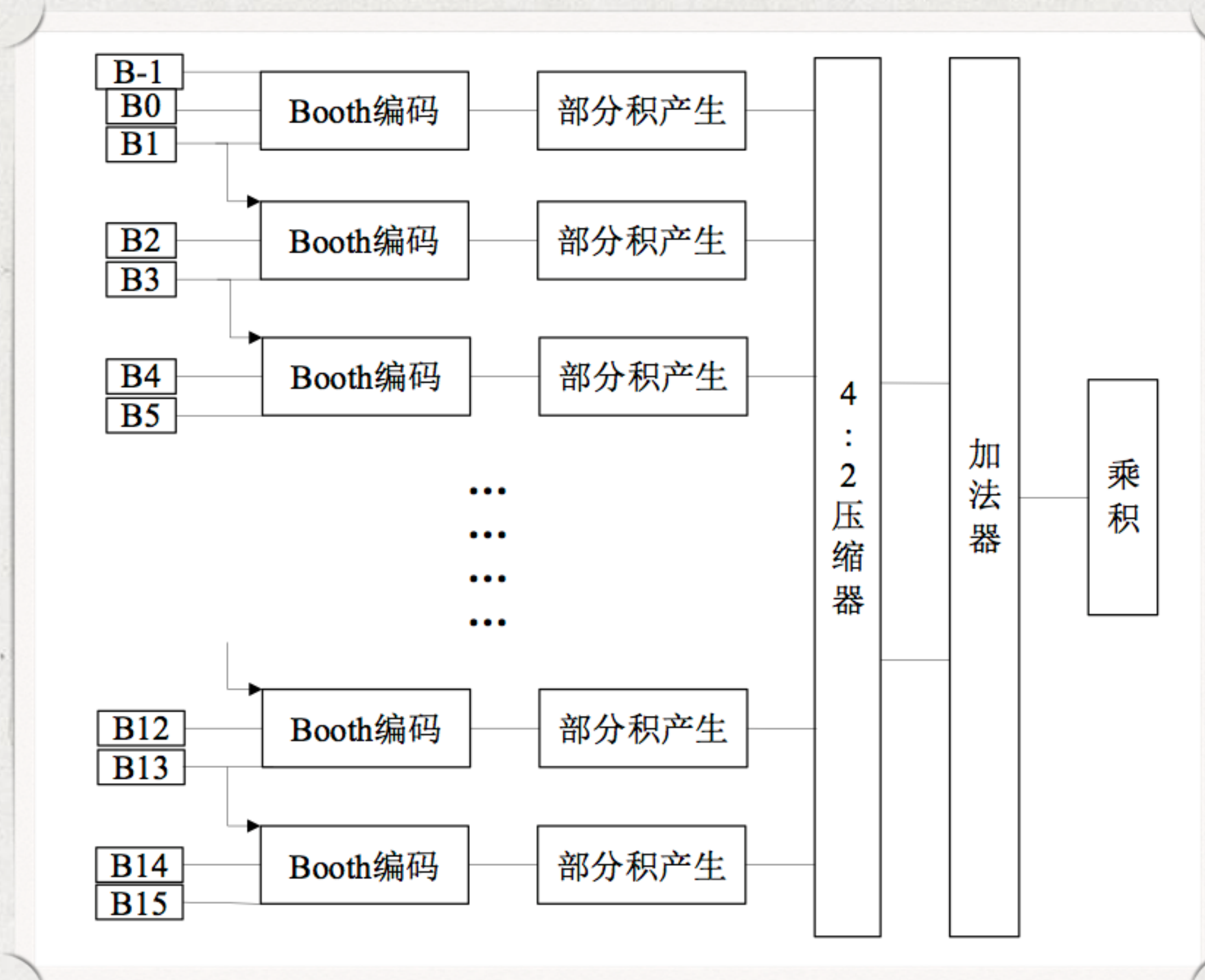
- 硬件部分：流水线乘法器模块



- 每个乘法器输入位宽为16bit，输出位宽为32bit。每个乘法器输出端连接slice模块，截取数据的高16位。

四、详细设计方案

- 硬件部分：流水线乘法器模块



Booth乘法器

四、详细设计方案

- 硬件部分：流水线除法器模块

此模块从AXI总线上读取数据进行除法运算，得到 $s_i = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$

(32bit)

$$\sum_{j=1}^N e^{x_j - x_{\max}}$$

(16bit)

$$e^{x_i - x_{\max}}$$



$$s_i = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}} = \frac{e^{x_i - x_{\max}}}{\sum_{j=1}^N e^{x_j - x_{\max}}}$$

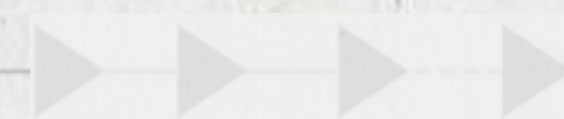


0

$$e^{x_i - x_{\max}} < \sum_{j=1}^N e^{x_j - x_{\max}}$$

移位除法器包含两部分：

数据预处理



移位除法器包含两部分：

数据预处理

(32bit)

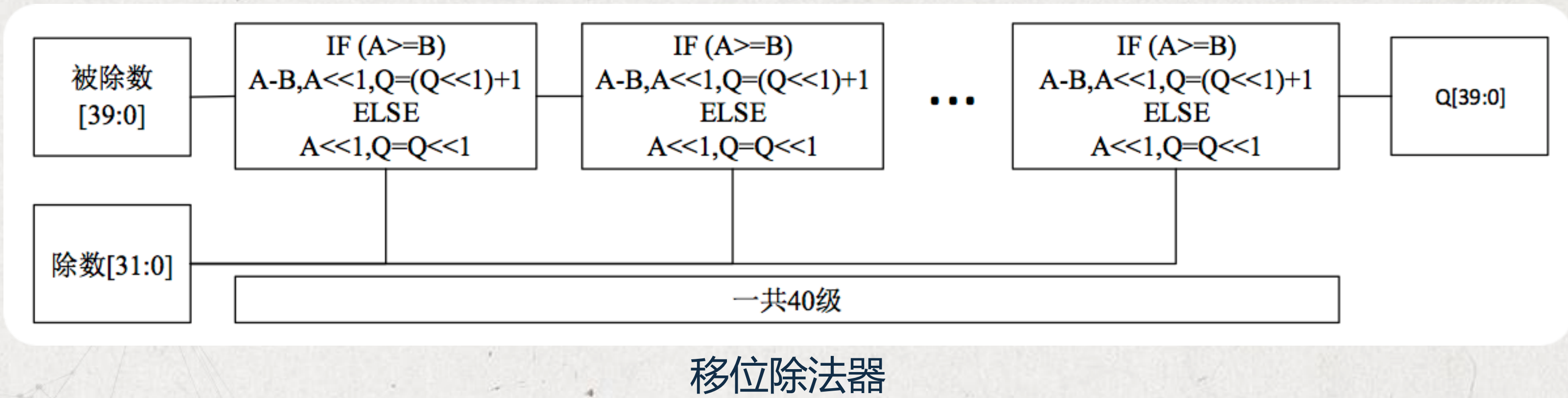
$$\sum_{j=1}^N e^{x_j^2 - x_{j-1}^2}$$

(16bit)

$$e^{x_j^2 - x_{j-1}^2}$$

- 左移24位

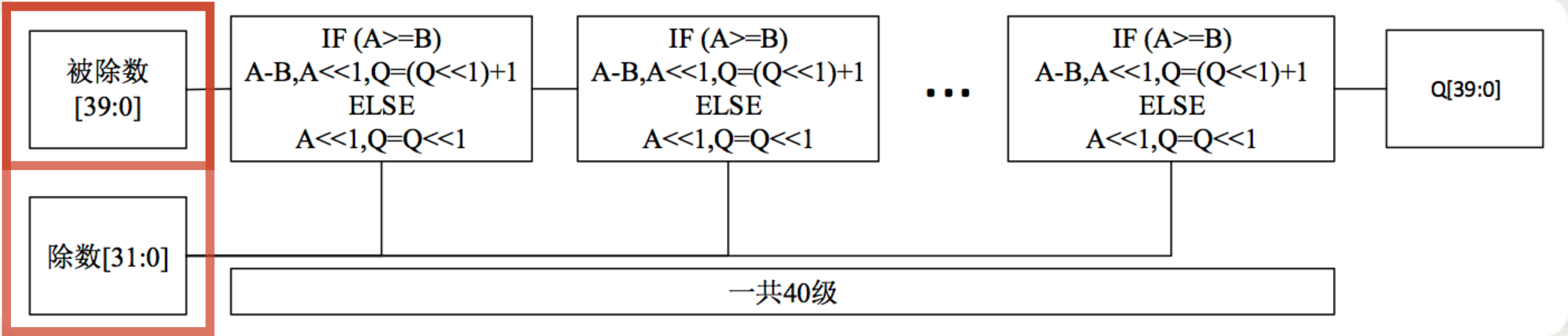
在采用移位除法器时，被除数小于除数，对于定点数除法器商值为0，因此需要将除数进行放大，本文针对[-10, 10]的范围将被除数左移24位构成40bit的被除数。



移位除法器包含两部分：

数据预处理

移位除法运算



移位除法器

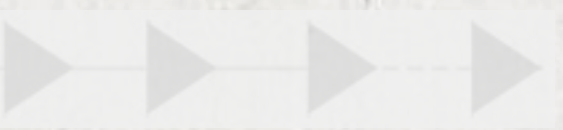
除数 < 被除数

商1

否则

商0

- 将除数左移一位



四、详细设计方案

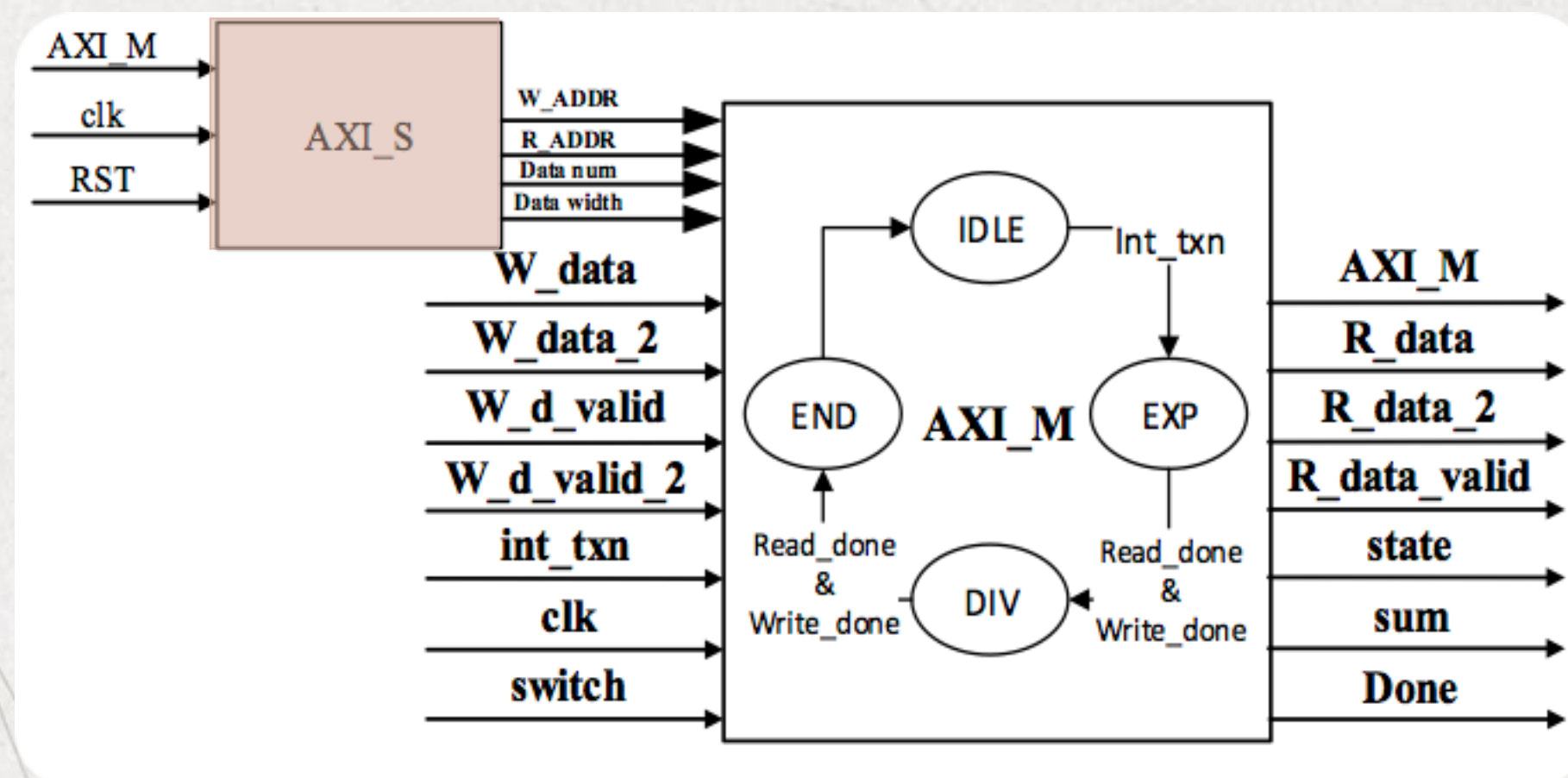
- 硬件部分：AXI和时钟控制模块

AXI读写模块:

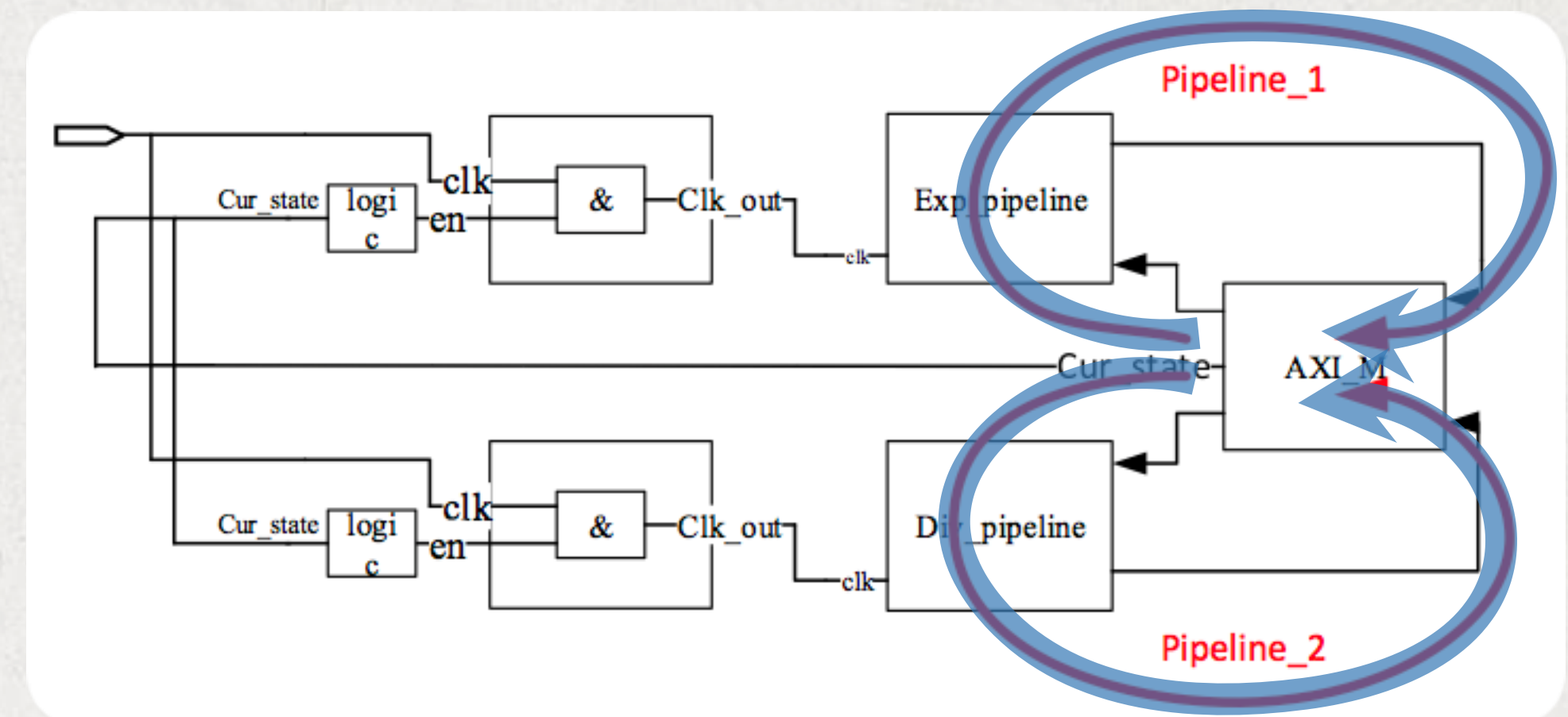
- ①AXI-Lite Master作为AXI读写部分核心，AXI-Lite Slave传递信息，减少ARM工作量
- ②使用一沿流水线传递的有效位指明数据有效性，降低硬件复杂度
- ②内嵌三段式状态机，提高系统稳定性

时钟控制:

当流水线需要运作时才将时钟信号输出给该条流水线，以达到控制流水线工作与停止



AXI读写模块



时钟控制

五、仿真与性能评

功能验证

FPGA测试

逻辑综合

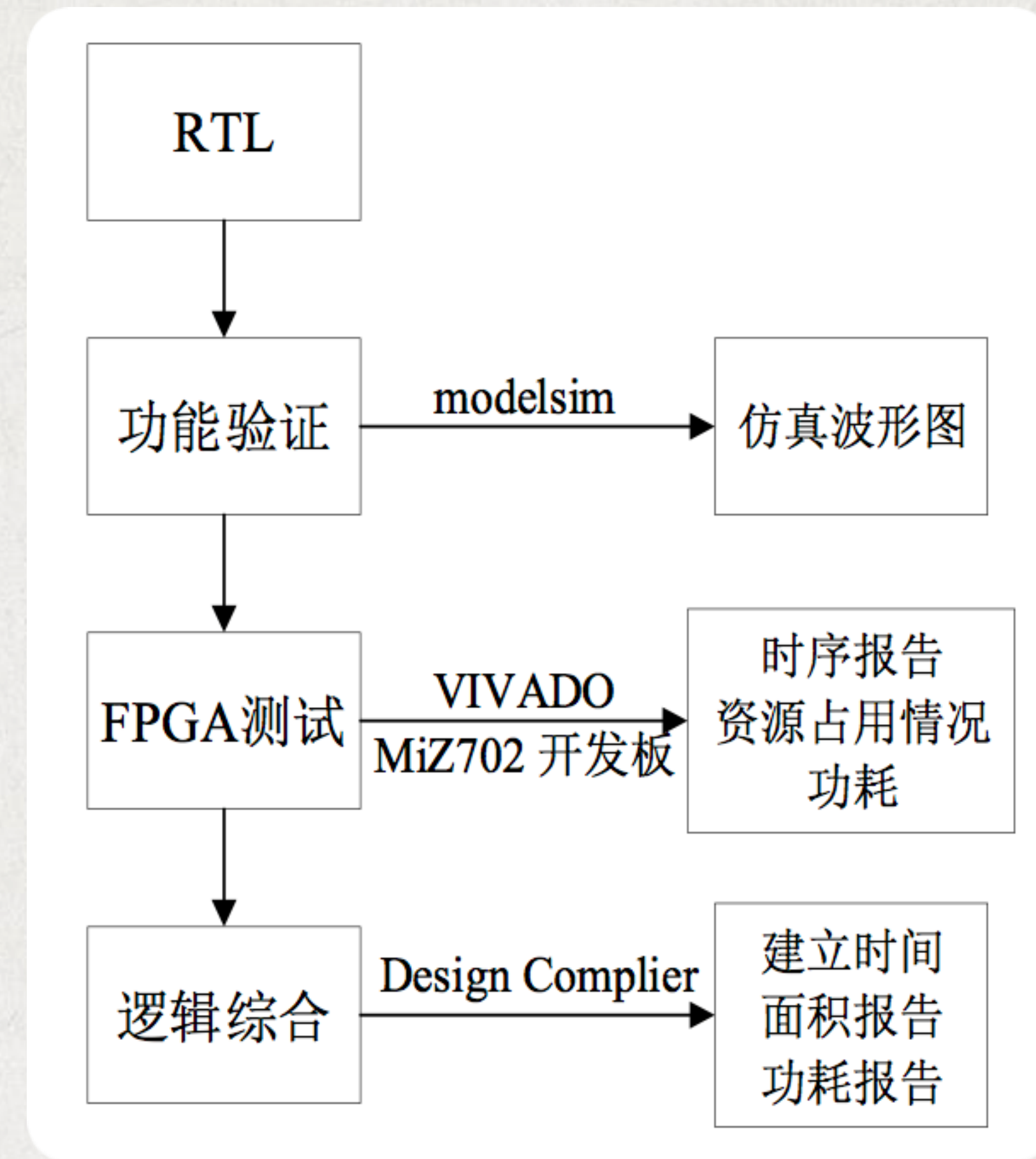
FPGA测试： 测试平台为MiZ702 与PYNQ开发板

- 首先由ARM 将需要计算的数据准备在指定地址的DDR 中，然后由ARM 进行数据定点化操作；
- FPGA 再对DDR 中进行函数计算再将数据覆盖原地址，计算结束返回DONE信号。
- 针对给出的数据范围，分别对[-5,5] 、 [-10,10]两个区间中4096个数据进行了测试。
- 测试完成后，与软件计算结果对比，分析计算误差。
$$error = (y_{\text{软件结果}} - x_{\text{FPGA}}) / y_{\text{软件结果}}$$

五、仿真与性能评

逻辑综合：

基于SMIC65nm标准单元工艺库，使用Synopsys Design Compiler工具完成逻辑综合。



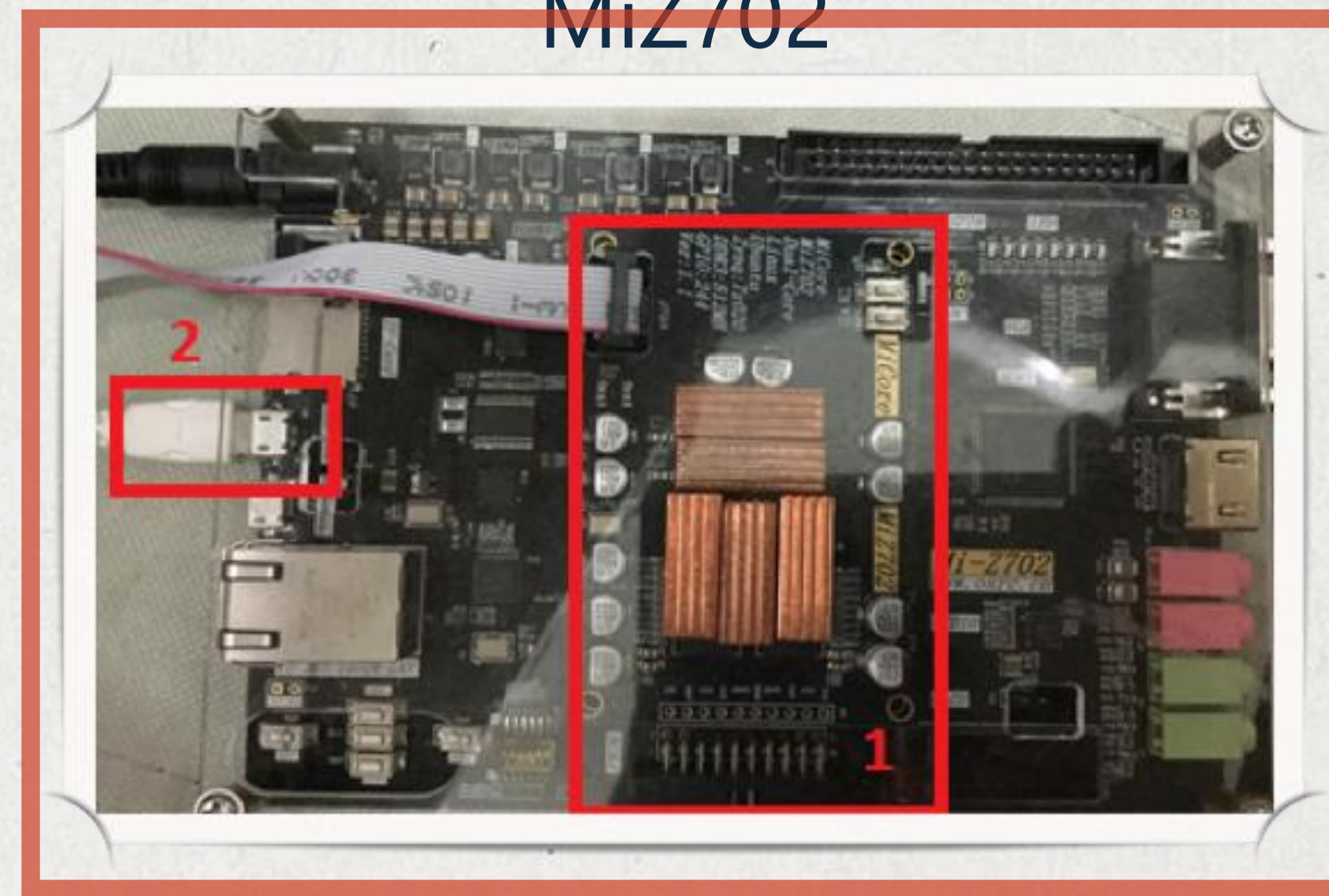
整体验证方案框图

五、两种测试平台

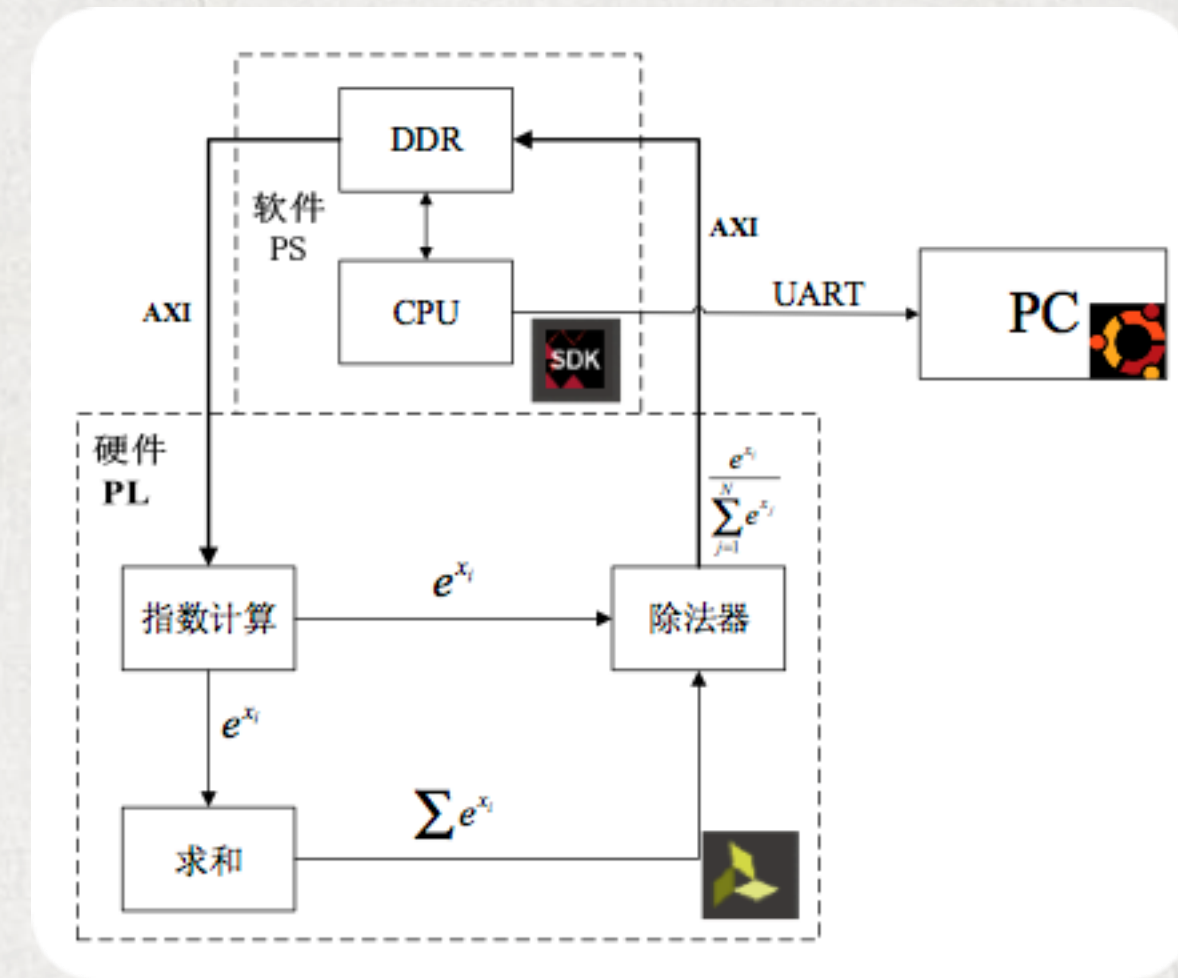
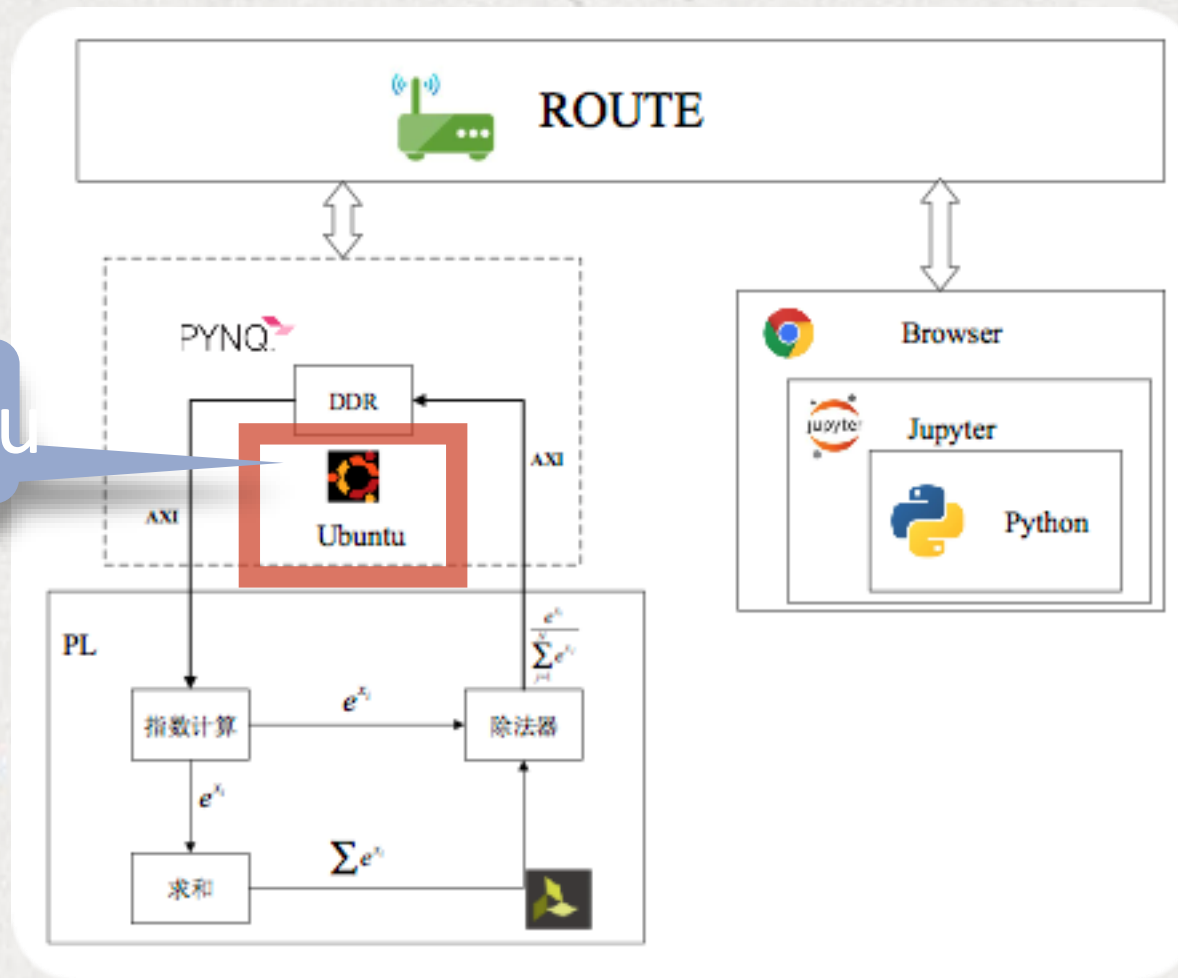
PYNQ-Z1



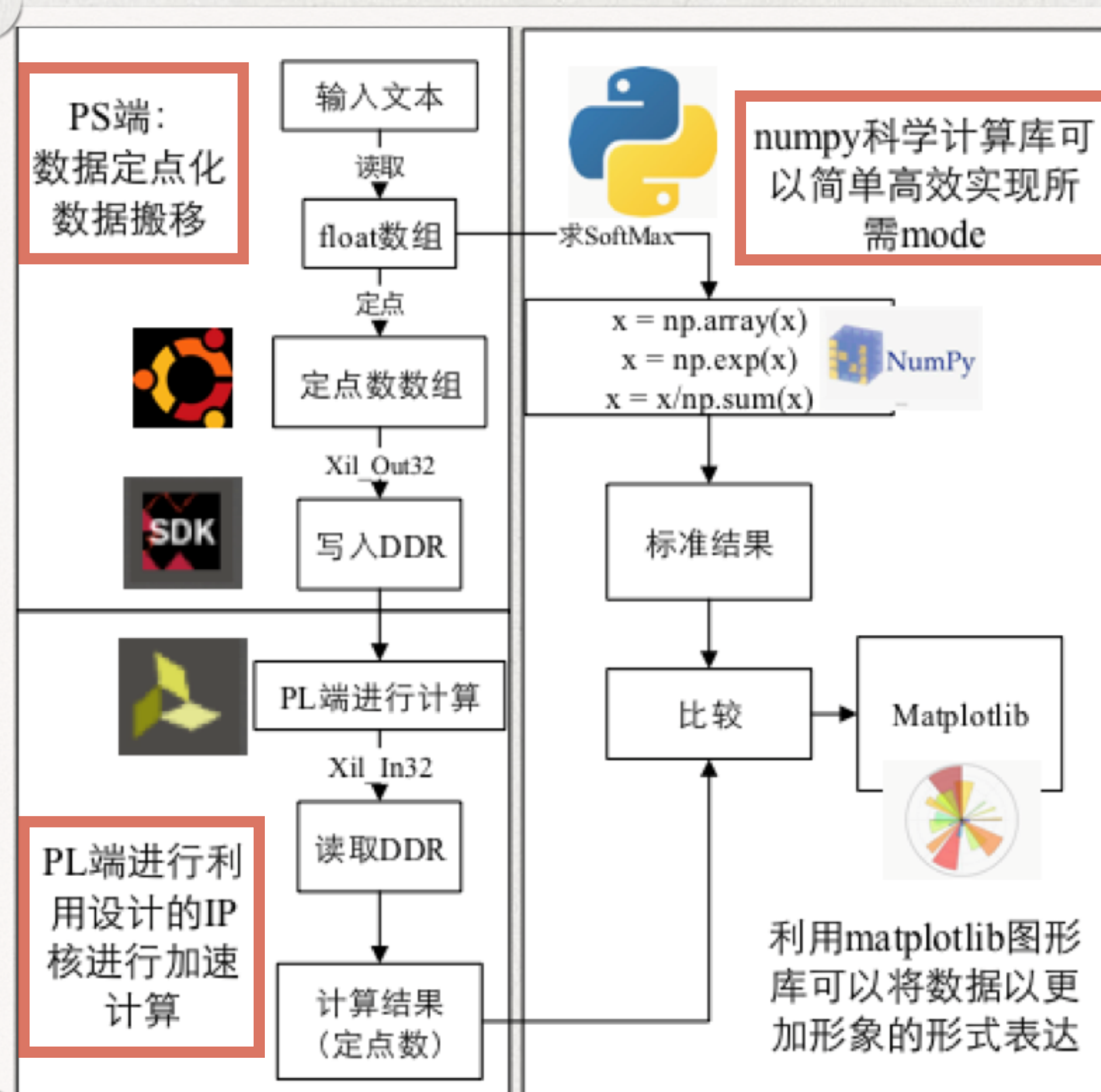
MiZ702



Ubuntu



五、Python Mode



五、Python Mode

matplotlib库

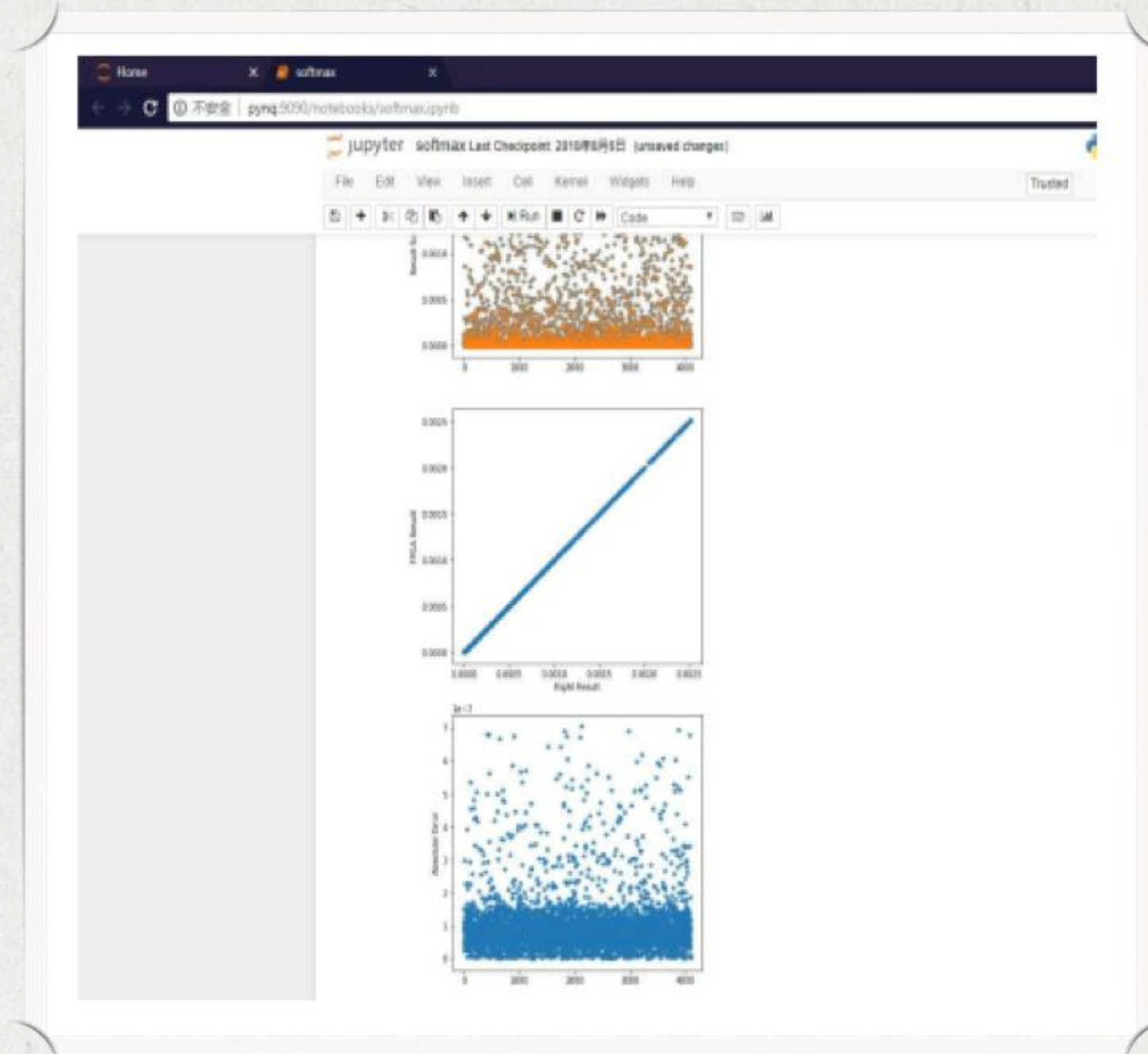


软件计算结果

硬件计算结果



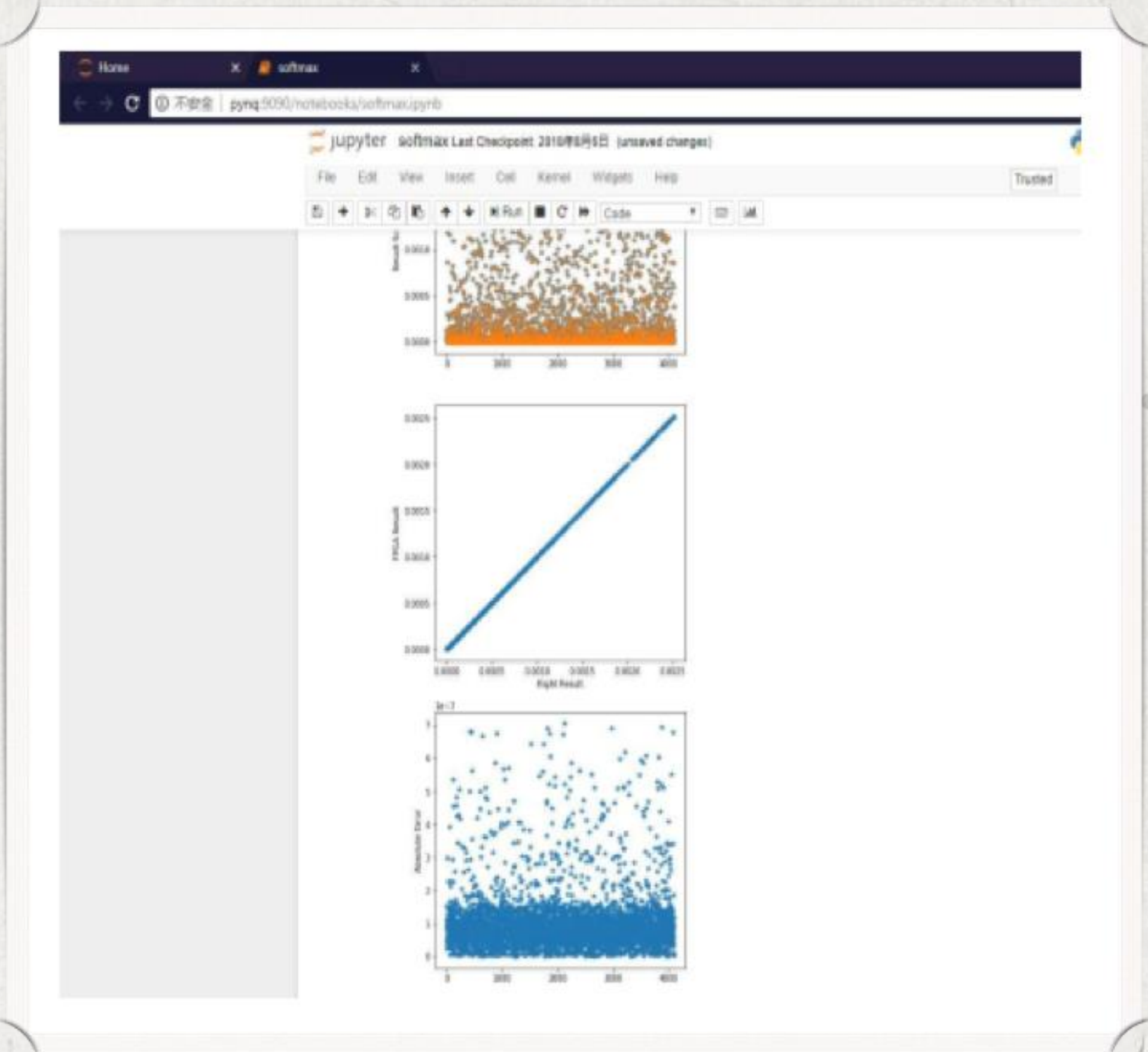
绝对误差计算



五、Python Mode

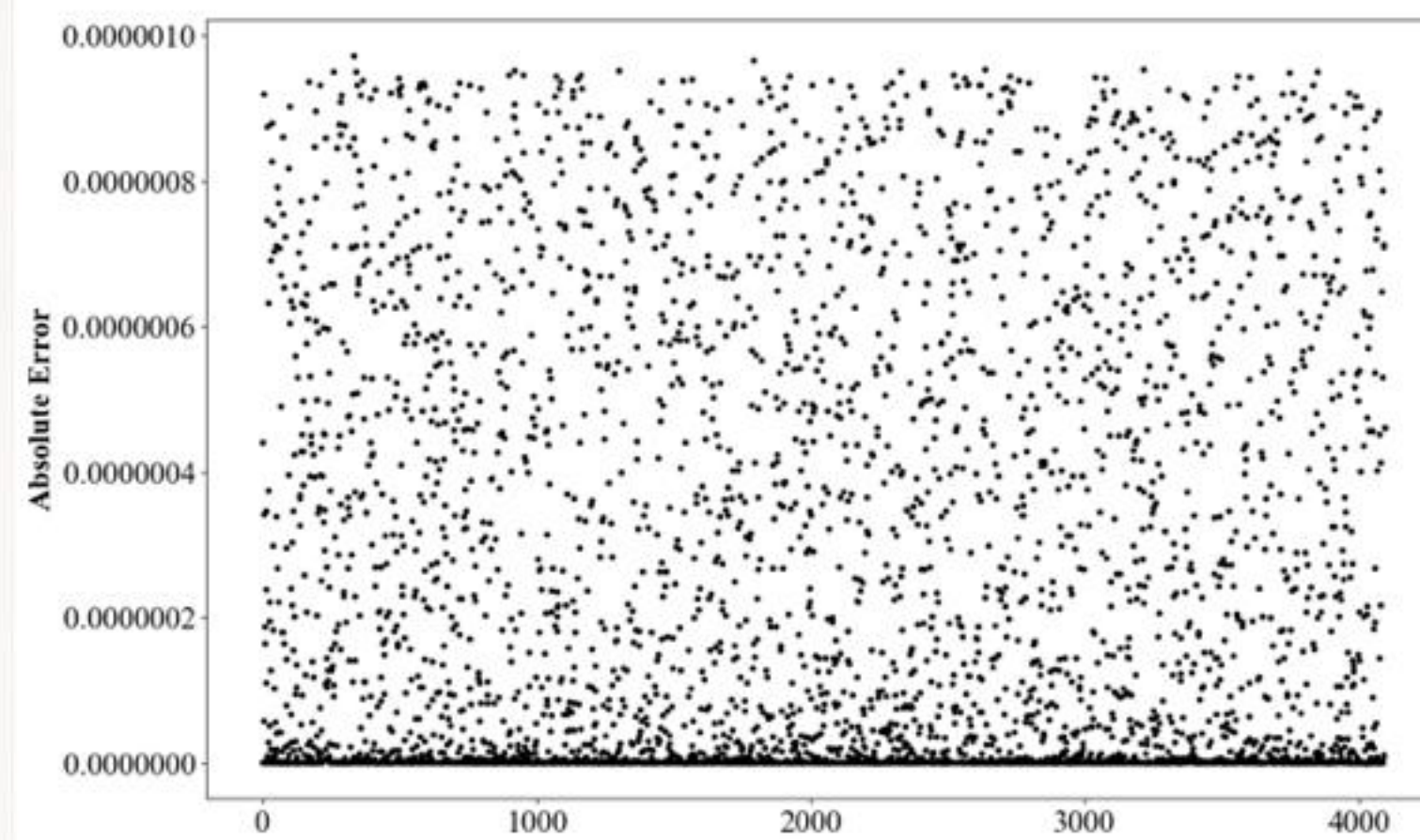
PYNQ

- 从SD卡中读入数据
- 数据定点化
- 数据写入DDR
- 控制PL端进行加速计算
- 读DDR
- 将软件结果与硬件结果对比通过误差散点图来测试设计的IP核的精度

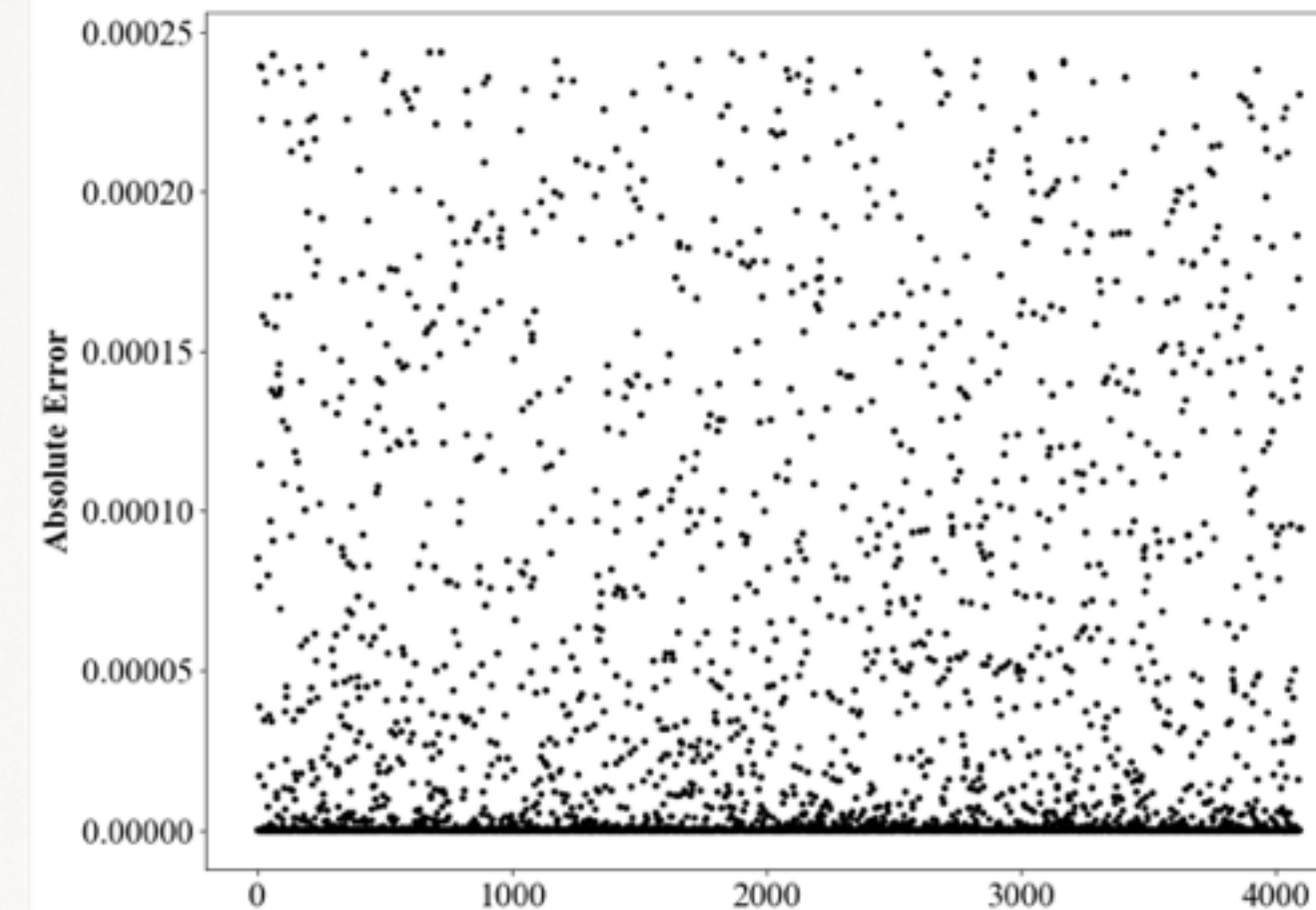


五、测试与性能评估

- FPGA误差分析



(a) [-10,10] 16bit误差分析



(a) [-10,10] 8bit误差分析

$$\text{error} = \text{abs}(y_{\text{软件结果}} - x_{\text{FPGA结果}})$$

此误差反应了每个点上的
偏差绝对值

- 在输入数据为16bit时，此误差率为 10^{-6} 数量级，误差非常低。
- 当输入8bit时，误差率稍大。这是因为数据位宽减少，所含信息减少，定点数的精度也随之下降。

五、测试与性能评估

- 性能评估

FPGA:

FPGA原型开发时，使用Xilinx推出的VIVADO作为开发环境。

ASIC:

使用Design Compiler进行综合，工艺库为SMIC65nm。

性能评估结果

	ZYNQ	SMIC 65nm
频率	200MHz	1GHz
面积	LUT/FF/DSP: 3323/5645/4	444858
功耗	302mW	333.2824mW