第二届
全国大学生集成电路创新创业大赛
NCICC

项目设计报告

参赛题目：中星微杯

队伍编码：1196

团队名称：教练我想学

目　　录

第一章　设计架构 …… 1

第二章　流水线设计及子模块 …… 12

第三章　优点与创新 …… 19

第四章　验证与测试 …… 22

第五章　性能评估 …… 31

第一章 设计架构

一、 设计原型

1. **公式变换：**

1） **变换目的 ：**

a） 直接根据定义式,硬件实现 e 指数,面临着值域范围过大的问题,难以轻量实现.

b） 利用简单的数学变换，可以简化计算,易于实现.

2） **变换过程：**如右图 1.1.1，首先分子分母同除以$e^{x_{max}}$，然后再对分子分母每一个项变换，使得原来以 e 为底变为以$\frac{1}{e}$为底。那么硬件所需拟合的函数就变为$y = (\frac{1}{e})^x$

$$
\begin{aligned}
f(x_i) &= \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}} \\
&= \frac{e^{x_i}}{e^{x_{min}} + \cdots + e^{x_{max}}} \\
&= \frac{e^{x_i - x_{max}}}{e^{x_{min} - x_{max}} + \cdots + e^{0}} \\
&= \frac{\left(\frac{1}{e}\right)^{x_{max} - x_i}}{\left(\frac{1}{e}\right)^{x_{max} - x_{min}} + \cdots + \left(\frac{1}{e}\right)^{0}} \\
&= \frac{(\frac{1}{e})^{x_{max} - x_i}}{\sum_{k=1}^{N} (\frac{1}{e})^{x_{max} - x_k}}
\end{aligned}
$$

图 1.1.1

3） **变换结果：**需实现的指数值域变为(0,1],范围大大缩小,同时,指数的幂从 0 开始,这将有效限制指数实现模块的输入范围.

2. **设计目标：**

$$
f(x_i) = \frac{(\frac{1}{e})^{x_{max} - x_i}}{\sum_{k=1}^{N} (\frac{1}{e})^{x_{max} - x_k}}
$$

图 1.1.2

由变换后的公式图 1.1.2,我们可以明确设计目标,一方面,我们需要比较得到最大值,进而实现 1/e 指数的运算,另一方面,我们需要将运算结果累加最终参与除法运算.

3. **基本流程：**

下图 1.1.3 所示。

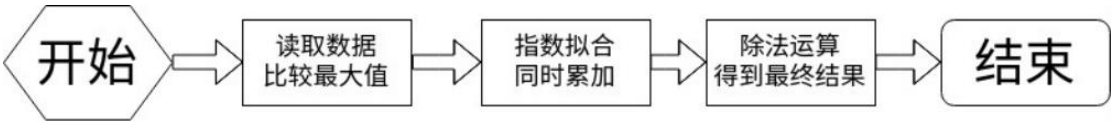
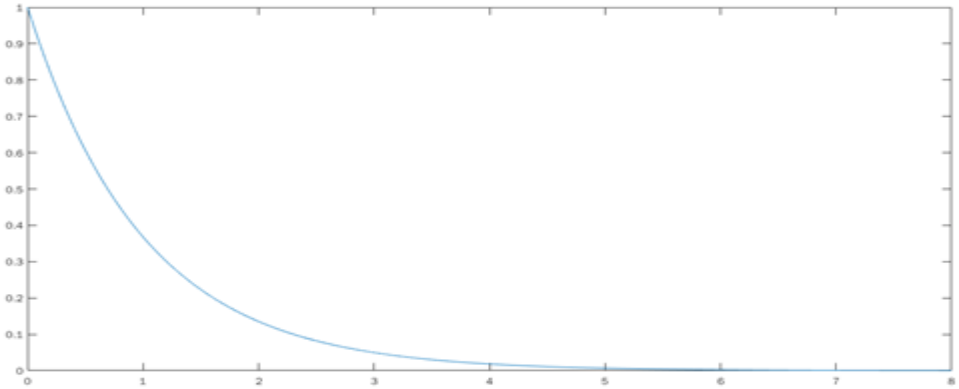


图 1.1.3

二、 定点化方案

（本节讨论定点方案时，默认是采用 16bit 数据格式.而采用 8bit 数据格式时，可以看做对 16 定点进行截取 8 位.）

1. **函数的截取：**通过硬件实现 y=(1/e)^x，需要注意的是，当 x=8，y 近似为 1/4096.当 16bit 数据格式时，将 1/4096 以 3 位整数位+12 位小数位方式定点化，得到的定点值为 1（定点范围是 0 到 4095）.采取同样的定点化方案，当 x>8 时候可以近似为 0，对输入大于 8 直接表示为 0.所以对这部分为 0 值进行截断，保留[0,8)为非零运算结果， 下图 1.2.1 所示。



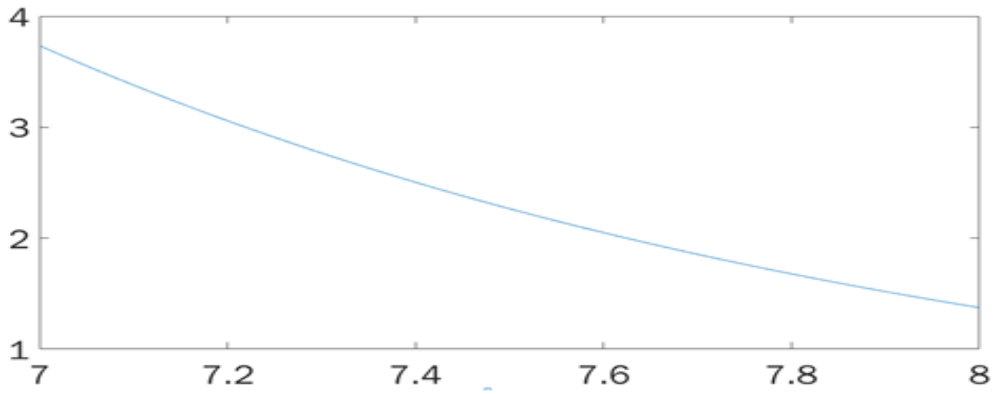


图 1.2.1

（a）定义域为[0，8]的函数图像（上）
（b）截取中定义域为[7:8],纵坐标定点化（转换为 3 位表示整数，12 位表示小数（下）

2. **拟合器输入输出的范围：**通过截断,拟合器的输入浮点表示范围[0.0,8.0)

   当支持 16 位由 1 的定点表示方案得到,定点范围为[0,2^15),为了方便起见,拟合器的输出也用该定点表示方案,得到输出范围为［0,4096］.

3. **除法器输入输出范围：**拟合器的输出数据作为除法器的被除数,定点方案相同.而除数作为被除数的累加,定点方案也相同.商作为输出结果,浮点范围是[0.0,1.0),定点采用 0 个整数位,16 个小数位表示,有效利用 16 位的数据格式.

4. **浮点数的定点化方案：**软件实现浮点数转为定点数后,需要用 8bit 定点值表示定点方案,用 fixed_q 表示.指数拟合器的输入对应的 fixed_q 值为 131，右图 1.2.2 所示.在实现中,我们采取的策略是,所有的输入定点数据共享同一个 fixed_q.fixed_q 由绝对值最大的输入数据决定.毫无疑问,这将使得定点表示浮点的成本降低,虽然有时会造成较小数值被忽略,但是较小数值对最终结果影响微小,可以舍去。

fixed_q	范 围
.	.
.	.
.	.
132	(-16，16)
131	(-8，8)
130	(-4，4)
129	(-2，2)
128	(-1，1)
127	(-1/2, 1/2)
.	.
.	.
.	.

图 1.2.2

5. **拟合器前的定点转换：**输入定点数据的 fixed_q 不一定为 131,而拟合器的输入的 fixed_q 却是 131.初始输入的定点数据势必要进行移位转换,对齐定点方案.所以在我们的实际设计中,输入数据先与最大值做减法,再进行移位,对扩展的高位判断得到溢出位,如果溢出的话,拟合结果直接为 0，具体过程如下图 1.2.3 所示.

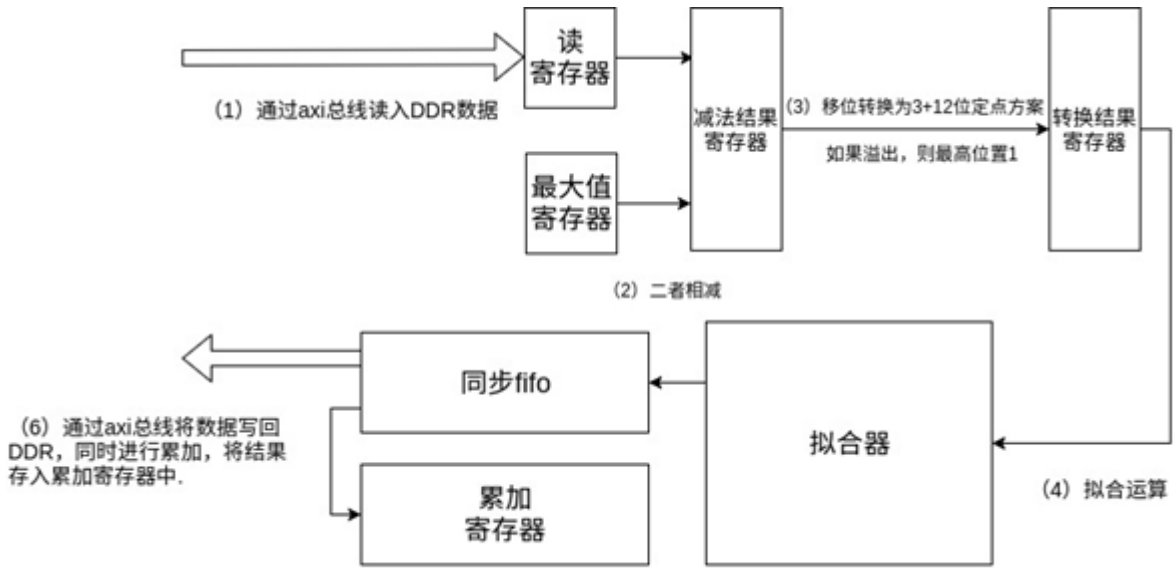


图 1.2.3

三、 拟合方案：

1. **8bit 数据格式 指数拟合方案**：此时,最简单高效的便是直接查表，用一个 8 位宽，深度为 256 的 rom 存储相对应的 rom 值，可以一个周期得到相应的指数值.由 case 语句实现,比较简单,不做赘述。

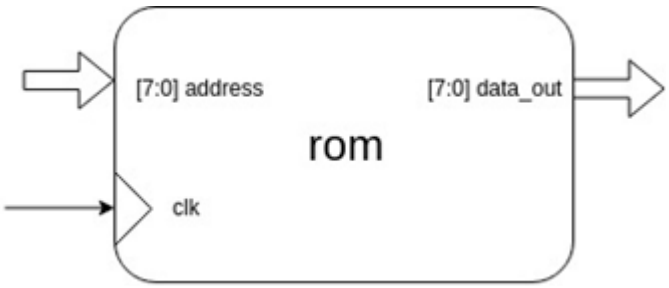


图 1.3.1

2. **16bit 数据格式 指数拟合方案**

a) **方案确定**:当 16 位输入时，如果要计算指数时，通过直接查表将耗用大量存储空间，最终我们采用查表+二阶拟合通过较少的资源消耗达到较为精确的运算目标.

b) **移位对齐**:在图 1.3.2 中所示，浮点转换为定点后直接相乘后是需要进行移位的，这是因为该乘法本质上是整数运算，而我们的意图是用整数运算实现小数运算，所以势必要进行移位使得小数点处于应处的位置，从而使得其定点运算时小数点对齐。

$$y = Ax^2 + Bx + C$$

$$\Downarrow$$

$$y_{定} = A_{定} * x_{定} * x_{定} \gg 24 \ + B_{定} * x_{定} \gg 12 \ + C_{定}$$

$$\Downarrow$$

$$y_{定} \ll 24 = A_{定} * x_{定} * x_{定} \ + B_{定} * x_{定} \ll 12 \ + C_{定} \ll 24$$

图 1.3.2

3. **流水结构**:通过调整运算的顺序，实现了二阶多项式的流水化.

乘法器既可以使用 LUT 和 FF 实现，也可以用 DSP 硬核实现，下图 1.3.3。

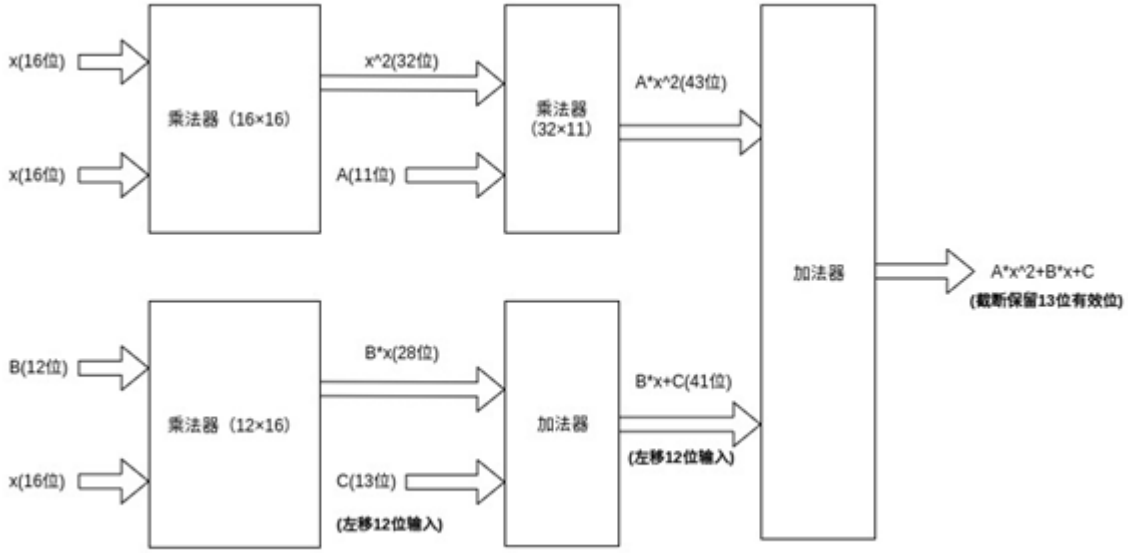


图 1.3.3

4. **系数表优化**:采用二阶多项式进行拟合，拟合的系数由 MATLAB 得出。由于横坐标越小，纵坐标的变化越大，所以我们采用的拟合区间不是等额区间，我们将 0 到 8 划分为 15 个区间，（0,1）为 4 个区间，（1,2）为 4 个区间，（2,3）为 2 个区间，（3,4）为 2 个区间，（4,5）为 1 个区间，（5,6）为 1 个区间，（6,8）为 1 个区间。再单独划分一个区间作为溢出区间，溢出区间系数为 0，合计共 16 个区间，在下图 1.3.4 所示。

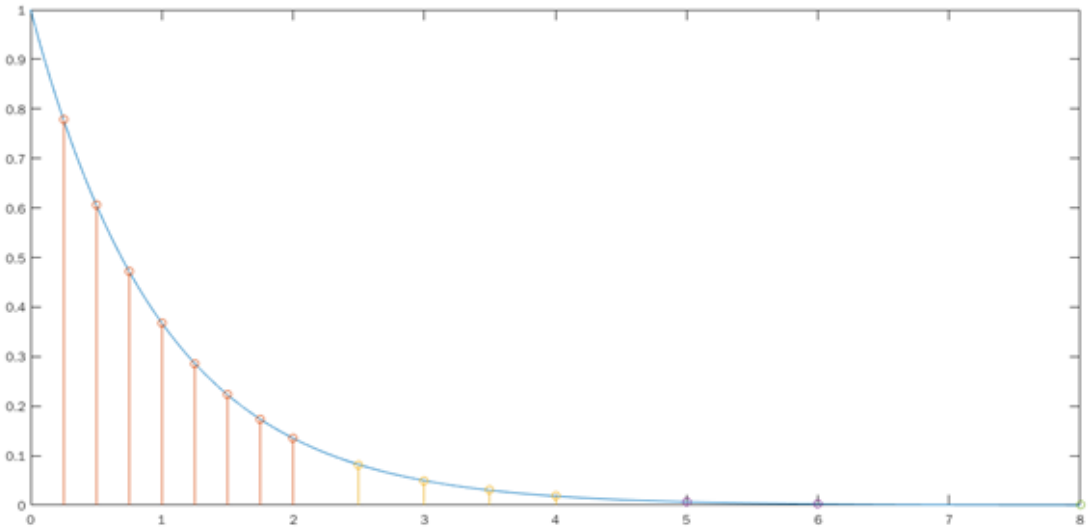


图 1.3.4

四、 设计结构

1. 基本输入输出

m00_axi_aclk	时钟信号
m00_axi_aresetn	复位信号
m00_axi_init_axi_txn	启动信号
m00_axi_txn_done	完成信号
M00_AXI	遵从 axi4-full 协议的接口

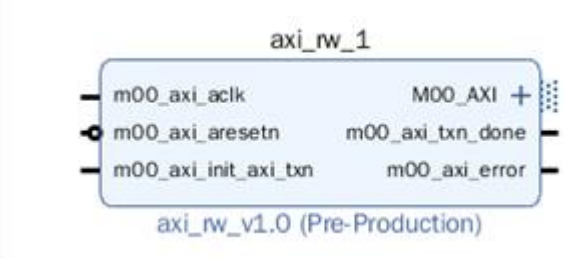


图 1.4.1

2. **代码层次**:下图所示，表明 verilog 文件在 project 里的结构。整个 IP 核设计不借助任何 IP，纯 verilog 实现。顶层模块包括 4 个 16/8 位复用 e 指数运算器，4 个 16/8 位复用除法器，4 个 8 位专用除法器，4 个 8 位专用 e 指数器，1 个同步 fifo，一个比较最大值模块和一个累加模块。

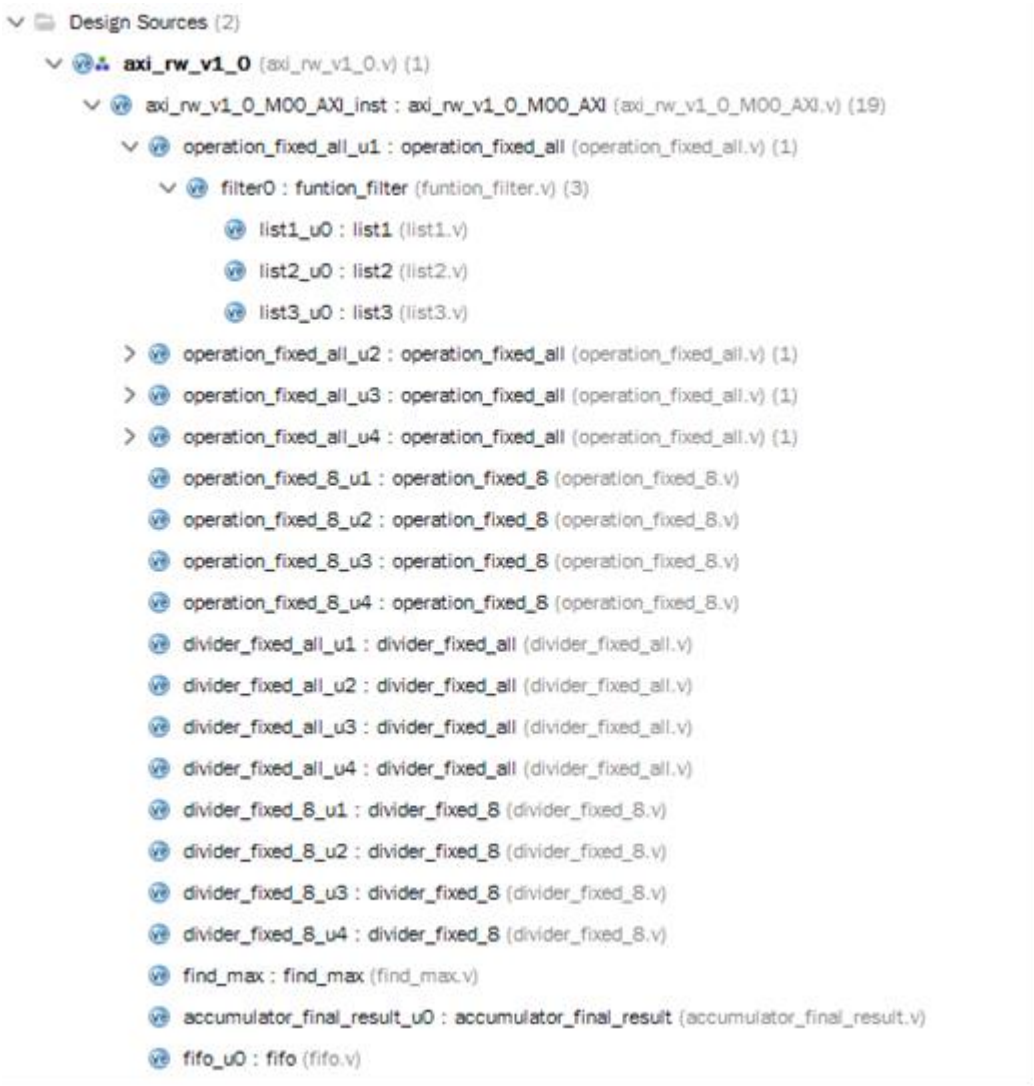


图 1.4.2

3. 模块功能介绍：

（1） 控制模块(axi_rw_v1_0_M00_AXI_inst)：

(a)包含顶层状态机,用于控制子状态机

(b)也包含读全子状态机、写全子状态机、读控制寄存器状态机三个自状态机,用于处理涉及 axi4-full 通信的相关部分

(c)联结或选通子模块之间的信号与数据

（２） 16 位/8 位复用拟合器(operation_fixed_all):兼容 8 位和 16 位数据格式,用于数据与最大值做减法,移位对齐以及指数的拟合

（３） 8 位专用指数拟合器(operation_fixed_8):8 位专用,用于数据与最大值做减法,移位对齐以及指数的拟合

（４） 16 位/8 位复用除法器(divider_fixed_all):兼容 8 位和 16 位数据格式,用于对输入数据除法运算

（５） 8 位专用除法器(divider_fixed_8):8 位专用,用于数据与最大值做减法,移位对齐以及指数的拟合

（６） 比较模块(find_max):包含了单通道比较的子模块,以及用于比较多通道结果的子状态机.

（７） 累加模块(accumulator_final_result):用于累加和,包含了累加子状态机

（８） 同步 fifo:用于外部读写数据之间的缓冲

4. 控制寄存器

控制寄存器大小为 64 位，在 DDR 中的地址可在 IP 核 parameter 中配置。控制寄存器的命名和功能下图所示

控制寄存器										
name	无	skipdivision	无	control_max	无	fixed_shift	无	register_number	fixed_q	reg_max
位数	[63:49]	[48:48]	[47:41]	[40:40]	[39:33]	[32:32]	[31:28]	[27:24]	[23:16]	[15:0]
功能	空	1：跳过除法阶段，输出拟合值 0：正常输出	空	1：PL端自行比较最大值 0：直接读取reg_max	空	1：PL端为8通道8位数据格式 0：PL端为4通道16位数据格式	空	写入值范围为0到8，用于通知PL端需要处理数据的个数，支持2^12,2^13,2^14,2^15,2^16,2^17,2^18,2^19,2^20九种数据个数，从而达到支持4096到2^20个数据的目标	用于通知PL端数据range	PS写入16位定点值，由control_max决定是否读取作为最大值

图 1.4.3

5. **状态机：**将复杂的控制分割给子状态机,由顶层统筹调用,一方面可以简化逻辑,另一方面实现了模块的复用.顶层状态机和子状态机通过启动和结束信号确认任务完成

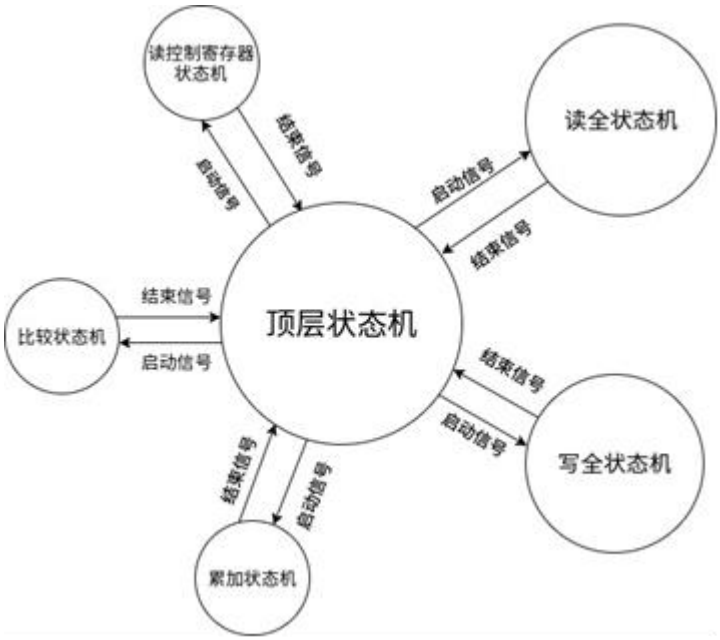


图 1.4.4

读控制寄存器状态机进行单次突发长度为 1 的读 64 位操作，从而读入控制寄存器。读全状态机直接读入 DDR 中所有数据，需要同步 fifo 的信号控制，而写全状态机写入 fifo 的数据，与读全状态机协同完成数据处理的流水线操作。累加和比较状态机是在多通道累加或者多通道比较之后，将多通道的 4 或者 8 个数据再进行累计或者比较。

五、 内存管理

1. **单个数据走向：**单个数据组逐个写回 ddr,存储在原先的地址中,不占用多余的 ddr 空间.（ 数据组由 4 个或者 8 个单个数据组成,个数视 8/16 定点方式决定 ）

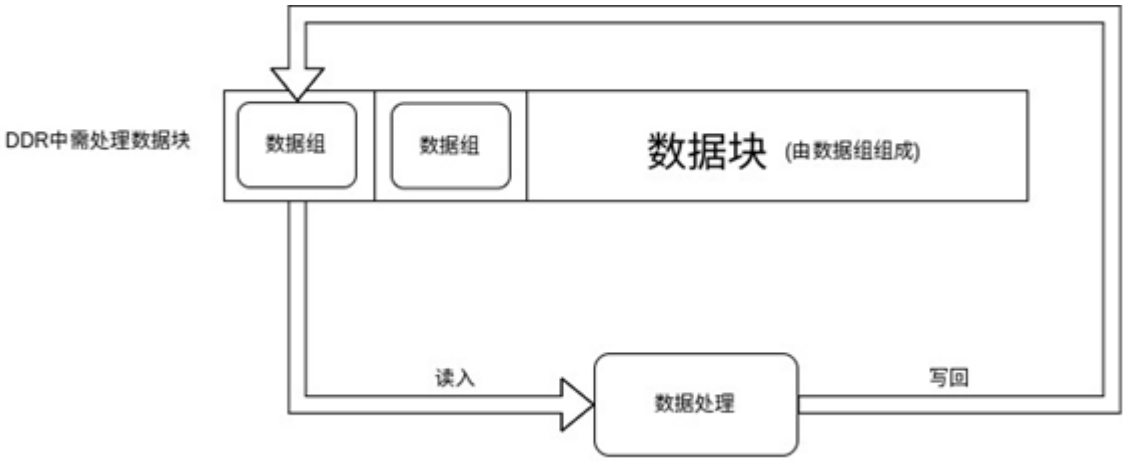


图 1.5.1

2. **内部缓存占用资源少：**读写之间通过位宽为 64,深度为 32 的同步 fifo 实现缓冲. 在 ip 核内部,不进行大数据的存储,大大节省资源消耗.

图 5.2

3. **总线利用：**

(1)8bit 数据格式：

图 1.5.3

(2)16bit 数据格式：

图 1.5.4

利用同步 fifo,读通道写通道同时进行,大大加快速度

64 位宽的数据总线利用率 100%,每次读均是 64 位全读,每次写均是 64 位全写

第二章 流水线设计及子模块

一、 数据流水线实现

简要描述:

借助读握手信号作为运算模块的门控时钟，从而使得每读一次，流水线向前推进一次，再借助同步 fifo 的 full 和 almost full 信号短时暂停读，方便将 fifo 的数据写出去。也就是说，满信号控制着读，空信号控制着写，在必要的情况下暂时停止，随后满足相应条件再重新激活，实现读写的同步，在顶层状态机的协助下截断有效数据，舍弃无用数据，使得数据一个不被遗漏的高效快速地完成正确的读写。

1. 门控时钟

输入数据是不连续的,所以无法连续工作.另一方面也是为了降低功耗,所以采用门控时钟.

当有输入数据时,流水线正常工作,当无输入时,流水线暂停工作.

简而言之,每进入一个数据,流水线向前推一步,同时输出一个数据,如果不进入数据,流水线保持不变,不做推进,输出不变.

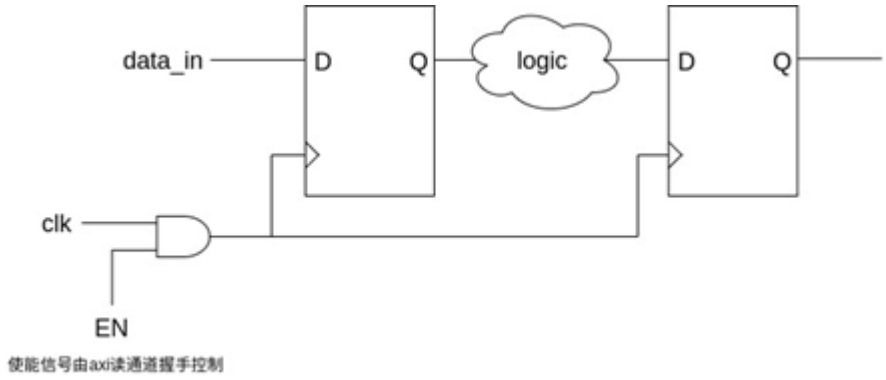


图 2.1.1

为了能形象描述门控时钟在我们设计中的作用,我们绘制了一系列典型的波形图，下图 2.1.2 到 2.1.4 所示。

Clock 作为主时钟,nclock 作为主时钟的反相时钟,en 为门控使能,gated_clock 为门控时钟,data_in 作为基于 axi-full 通信协议接收的数据,同时将作为数据处理模块的输入数据,data_out 是数据处理模块输出的数据,data_out 随后被置入 fifo_data 中,而 fifo_data 指的是同步 fifo 中的数据.同步 fifo 随后输出到 axi 写通道数据总线中.

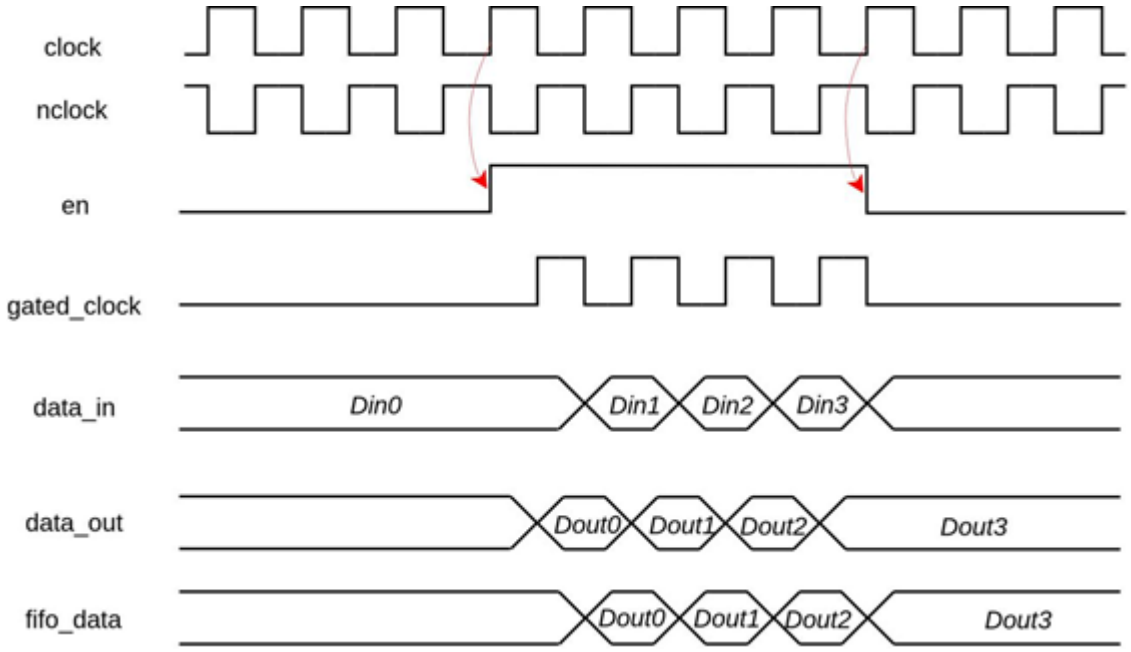


图 2.1.2

使能由主时钟控制

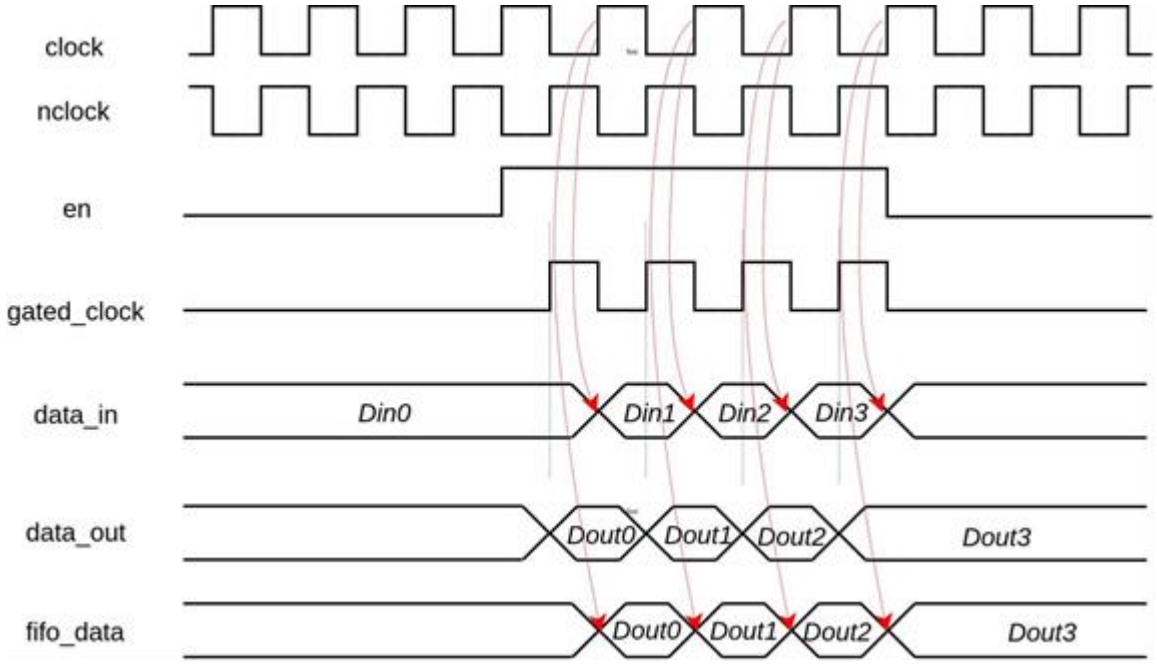


图 2.1.3

数据接收经过 axi 读通道，所以是主时钟控制的，fifo 时钟也是主时钟，便于输出数据于写通道. 被门控时钟驱动的数据处理模块置入 fifo 中.

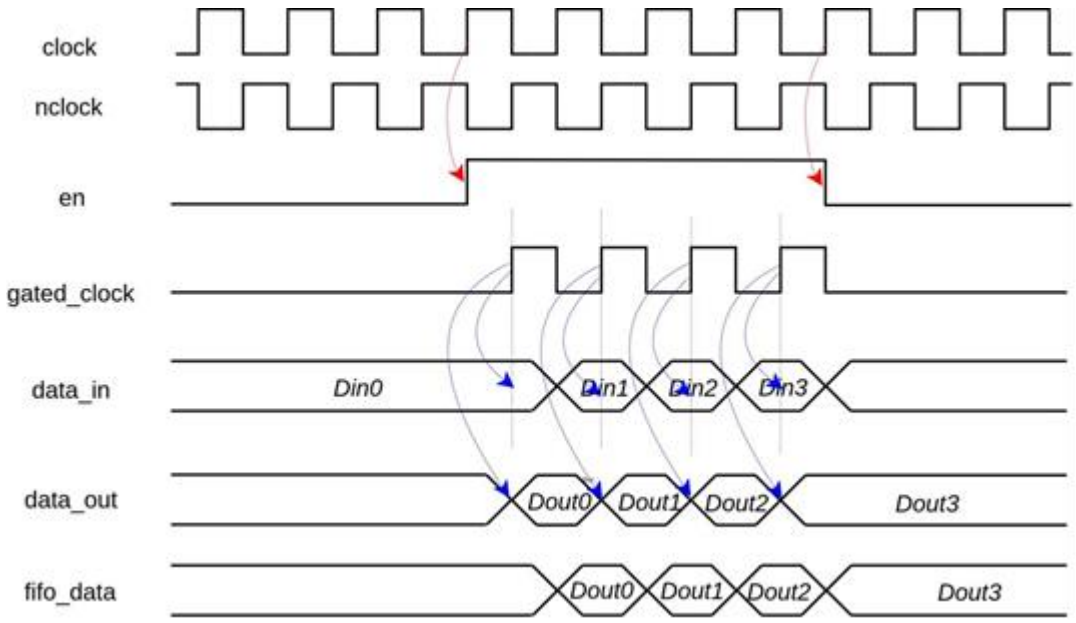


图 2.1.4

数据处理模块采样 data_in, 输出为 data_out

2. 数据处理流水线

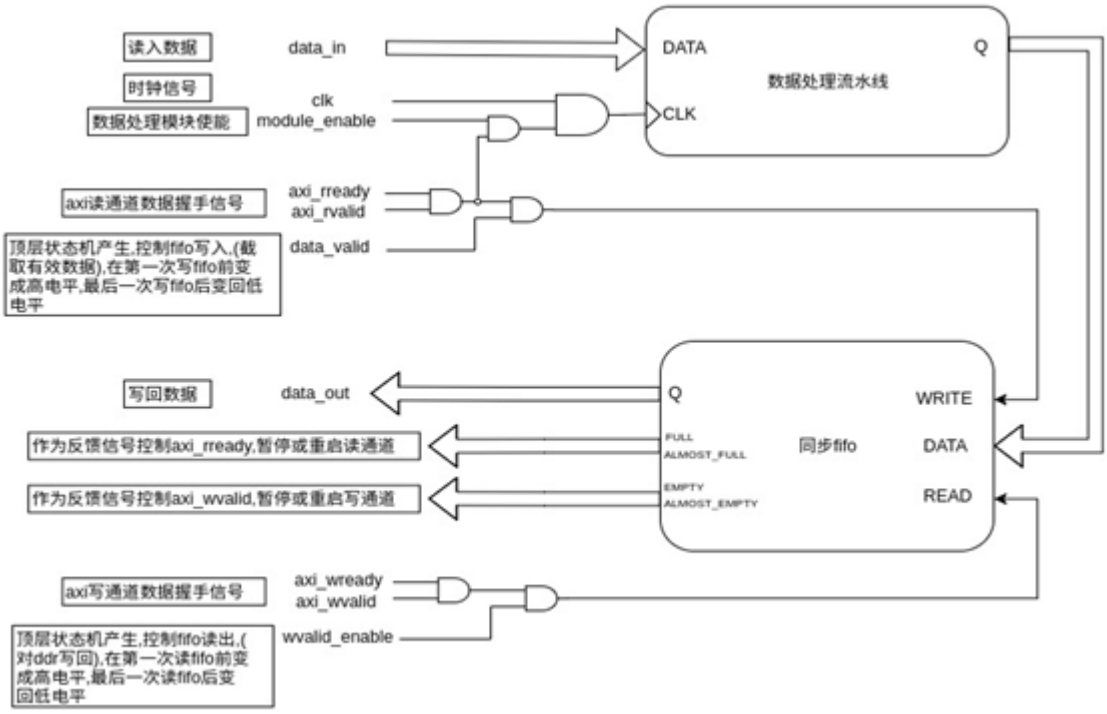


图 2.1.5

(1)数据一一对应读入的地址写回, 不漏首个数据, 也不漏最后一个数据.

(2)在合理有序的控制信号下,读写将同时进行,关键得益于反馈信号的有效控制.

二、 子模块

1. **控制寄存器读入：** 在顶层状态机中，通过启动信号驱动子状态机完成控制寄存器读入。子状态机通过 axi 协议发起单次突发长度为 1 的读操作，读地址为 ip 核可配置的 parameter。读完成后，子状态机发出 done 信号，再由顶层状态机确认进入下一个状态。

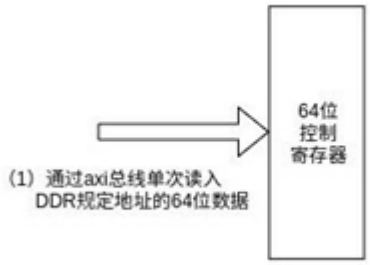


图 2.2.1

2. **寻找最大值：**在顶层状态机中，通过启动信号，启动次顶层读状态机，发起由单次突发长度为 16 组成的可以遍历所有输入数的全读操作。将读入的 64 位数拆分，再输入给多个用于比较大小的子模块。在读完全部输入数后，次顶层读状态机发出 done 信号，由顶层状态机确认进入下一个状态。进入下一个状态后，再启动另外一个子状态机，该子状态机用于将 4 或者 8 通道的对应数在进行比较，最终得到真正的最大值，再发出 done 信号由顶层状态机进行确认。

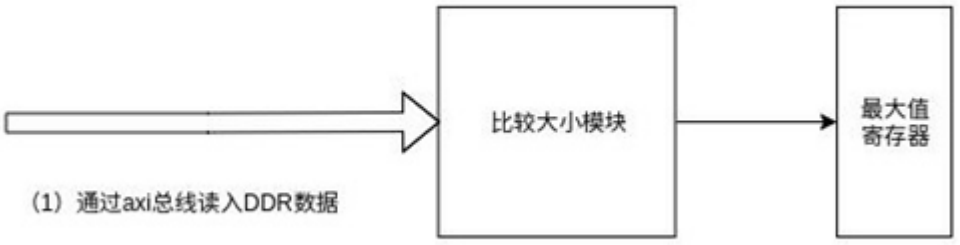


图 2.2.2

3. **定点拟合模块**

16 位定点采用二阶多项式进行拟合，拟合的系数由 MATLAB 得出。在实际拟合中，横坐标越小，纵坐标的变化越大，所以我们采用的拟合区间不再是等额区间，我们将 0 到 8 划分为 15 个区间，（0,1）为 4 个区间，（1,2）为 4 个区间，（2,3）为 2 个区间，（3,4）为 2 个区间，（4,5）为 1 个区间，（5,6）为 1 个区间，（6,8）为一个区间。再单独划分一个区间作为溢出区间，溢出区间系数为 0，合计共 16 个区间，所有系数右表所示。

address	A	B	C
4'b0000	0	0	0
4'b0001	1809	-4073	4096
4'b0010	1409	-3876	4070
4'b0011	1097	-3568	3994
4'b0100	855	-3206	3858
4'b0101	666	-2830	3671
4'b0110	518	-2463	3442
4'b0111	404	-2120	3185
4'b1000	314	-1808	2913
4'b1001	217	-1410	2507
4'b1010	132	-987	1981
4'b1011	80	-678	1521
4'b1100	48	-460	1140
4'b1101	23	-255	724
4'b1110	9	-111	369
4'b1111	2	-32	131

4. 包含拟合的减法移位模块

单单包含拟合是不能直接算出所需实现函数的分子，还需要通过定点方案与 3+12 定点方案做减法得到的结果对数据进行移位。通过对高位相或，得到溢出位，然后将转换后的 3+12 的定点数作为输入，提供给拟合器。

输出输入给同步 fifo，当 fifo 输入到 DDR 中时，进行多通道累加，累加后，再通过子状态机，累加多通道的累加结果。

实现过程下图所示。

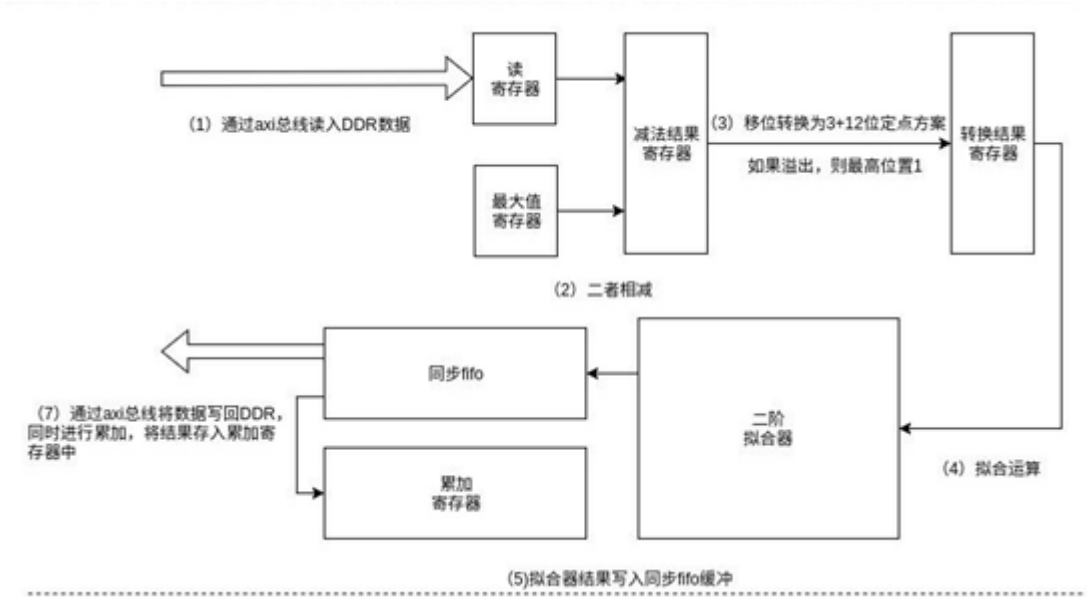


图 2.2. 3

5. 除法器

8 位和 16 位的除法器基本相同，实现原理很简单,通过移位比较判断是否相减实现,每一个上升沿得到结果中的一位，将之前的运算保留，通过复制 always 语句实现流水线。

下图显示了除法器在流水线中的作用。

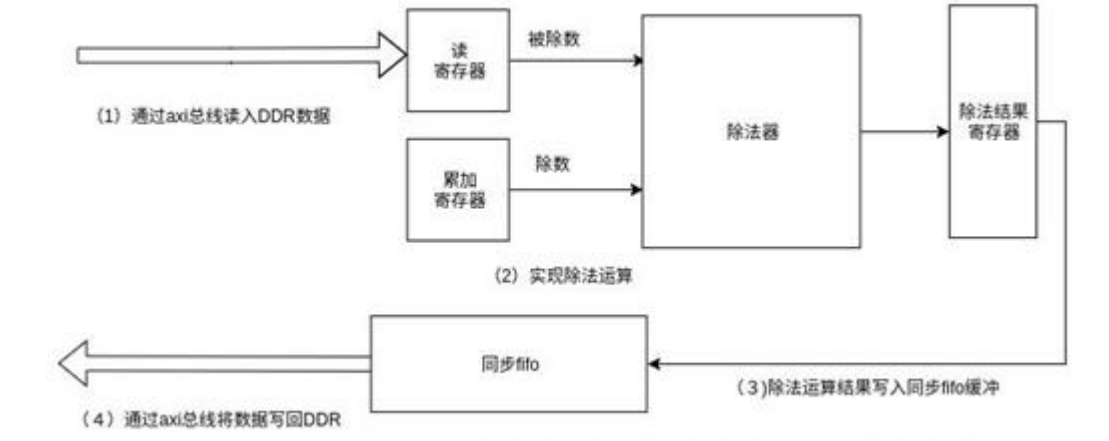


图 2.2. 4

6. 同步ｆｉｆｏ

同步 fifo 比较简单，通过纯 verilog 实现了先进先出的数据结构，同时通过对读写计数，从而得到 full,empty,almost full,almost empty 信号，功能上和其他同步 fifo 大同小异。

第三章 优点与创新

一、 多模式控制与模块复用

通过写入控制寄存器实现 8 定点到１６定点的切换.

1. １６位 e 指数拟合器兼容 8 位 e 指数运算，并且１６定点除法器也兼容 8 定点输入，通过这两者的实现，达到了模块复用的目的，从而实现 8 位输入运算 8 通道运算时，复用原本属于１６位的四个通道，降低大量的资源消耗.
2. 控制寄存器方面，提供一位用来控制最大值读写是否进行，在６４位控制寄存器中，最低１６位可以用来写入最大值（前提是打开），从而缩减ＰＬ查找最大值的运算.

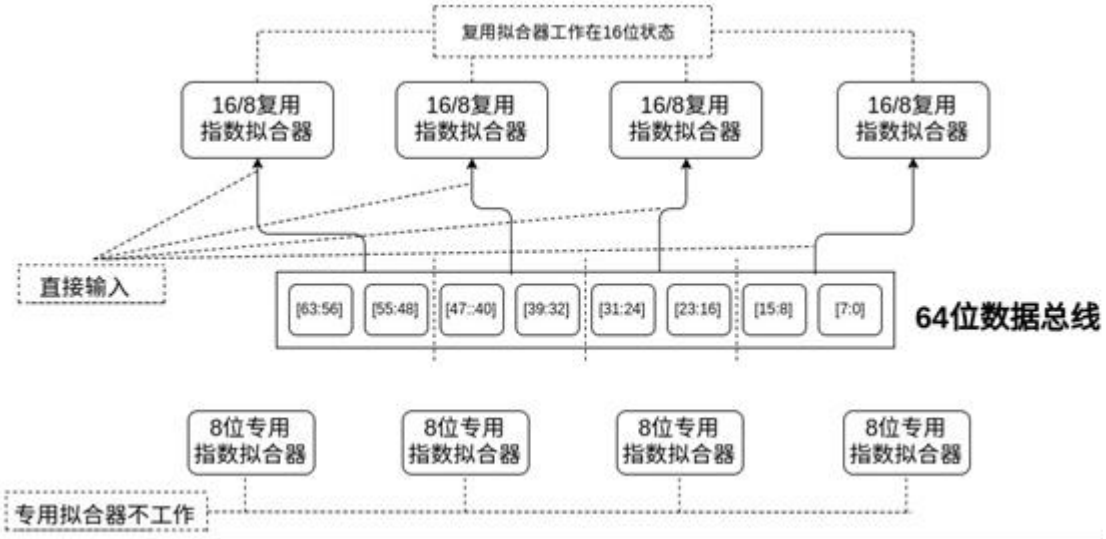


图 3.1. 1

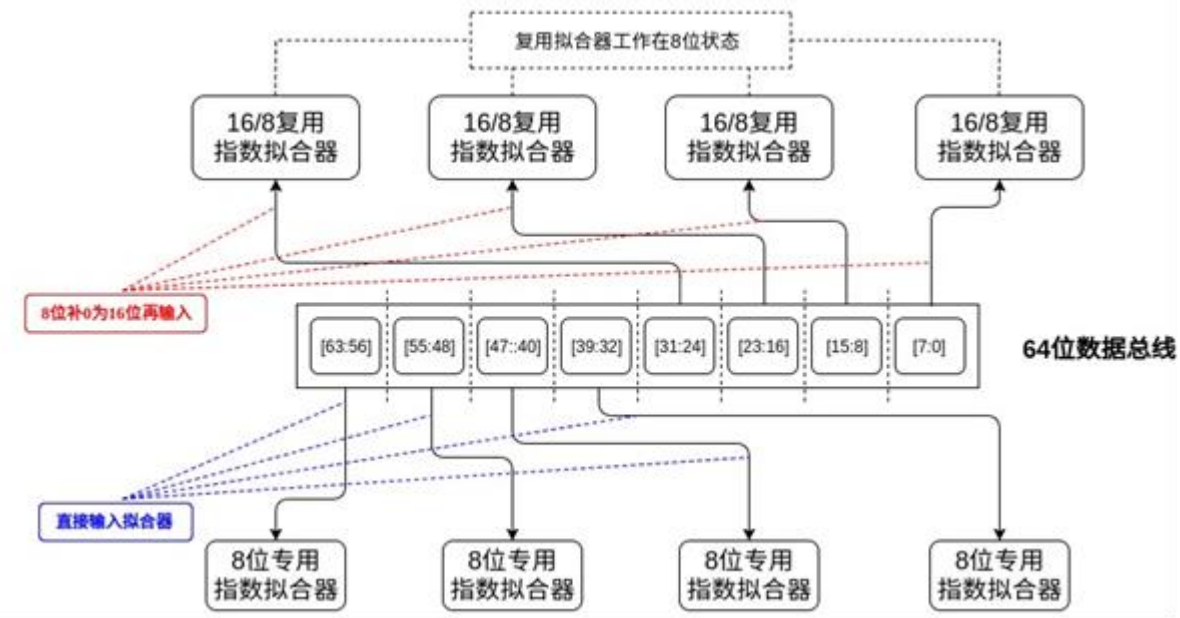


图 3.1. 2

二、 软件部分灵活

考虑到在输入浮点数的范围十分大时，势必引起定点的精度损失，并且该精度损失会随着输入浮点数范围越大而变得越大，这时可以通过用软件实

现查找最大值，关掉控制寄存器相应的使能位，再将输入值减去最大值＋８，再实现定点化，从而实现精度的大大提高．而此时，ＰＬ部分对这种情况是兼容的，它既支持直接定点化，也支持做完减法（并且加上８）再定点化.

三、 拟合运算资源占用小

拟合计算时，根据各自特点，１６位采用查找表和ｄｓｐ结合二阶拟合进行运算，单独８位时则采用ｒｏｍ查找表，大大降低资源消耗。

四、 缓冲面积小而速度快

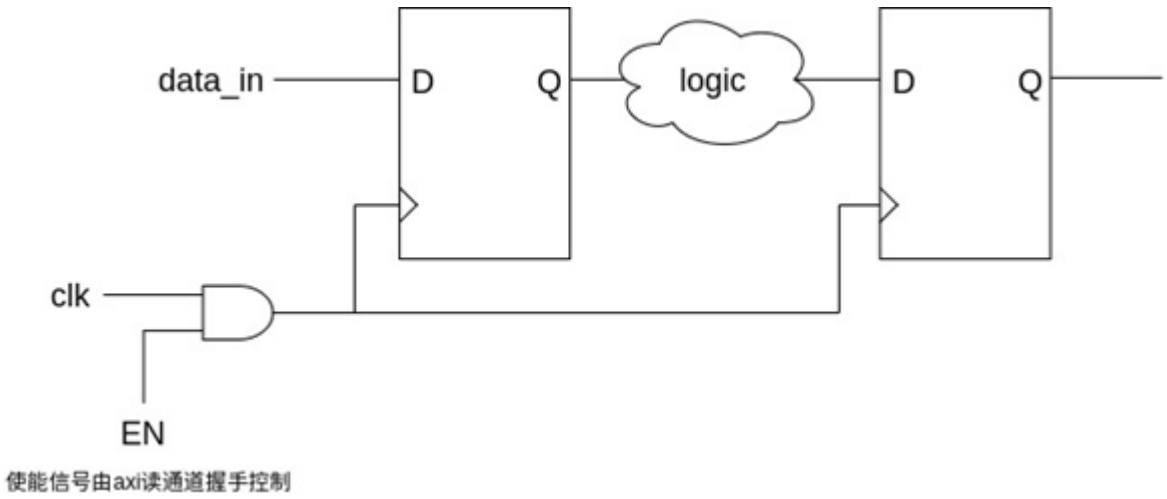


图 3.4.1

使用门控时钟,降低低功耗的同时也可以使得数据流水线跟随数据的输入，同时进行输出，两者之间通过６４×３２大小的同步 fifo 实现缓冲。并且通过 full，empty,almost_full,almost_empty 四根信号线来协调控制，full,almost_full 高电平信号来暂停读过程,直到转为低电平后唤醒读过程，empty 和 almost_empty 信号来暂停写过程,直到转为低电平后唤醒写过程。一开始会出现读写的不均衡，导致读写的某一方小几个ｃｌｋ的暂停，但这种暂停是很短的，同时随着读写过程的持续，读写将最终走向均衡，因满空信号高电平而导致的短时暂停将基本不存在．通过极少的ＰＬ部分内部存储

空间，完成运算的缓冲。

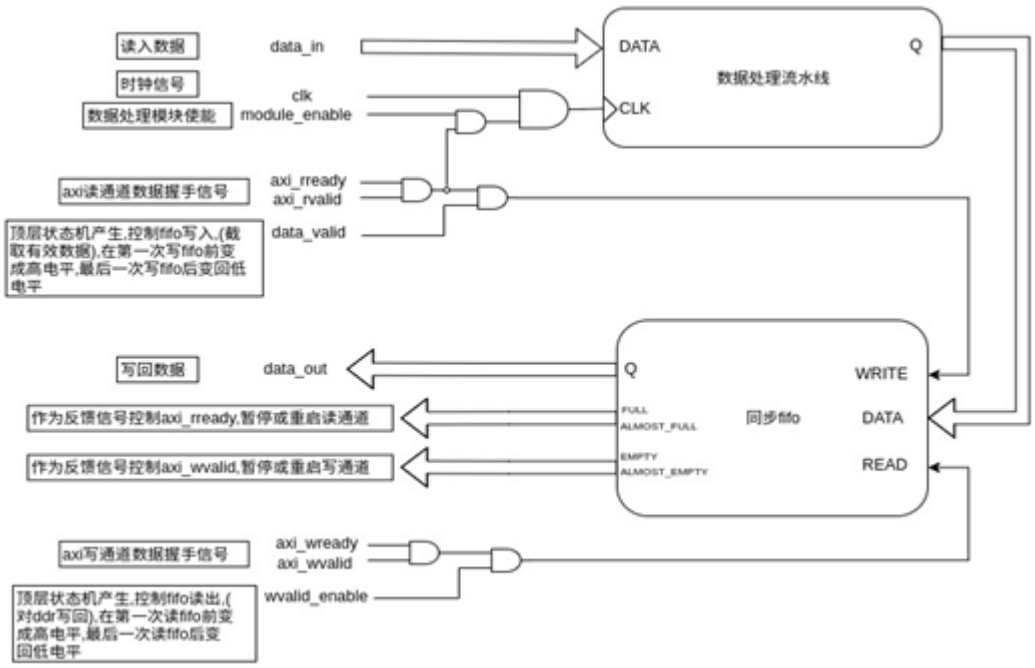


图 3.4.2

第四章 验证与测试

1. 子模块行为级仿真

为验证子模块基本功能,我们编写了 testbench,成功测试子模块的基本功能.

operation_fixed_8_tb.v

operation_fixed_all_tb.v

divider_fixed_8_tb.v

divider_fixed_all_tb.v

accumulator_final_result__tb.v

fifo_tb.v

2. 顶层行为级仿真

我们编写了测试代码,搭建顶层行为级仿真.

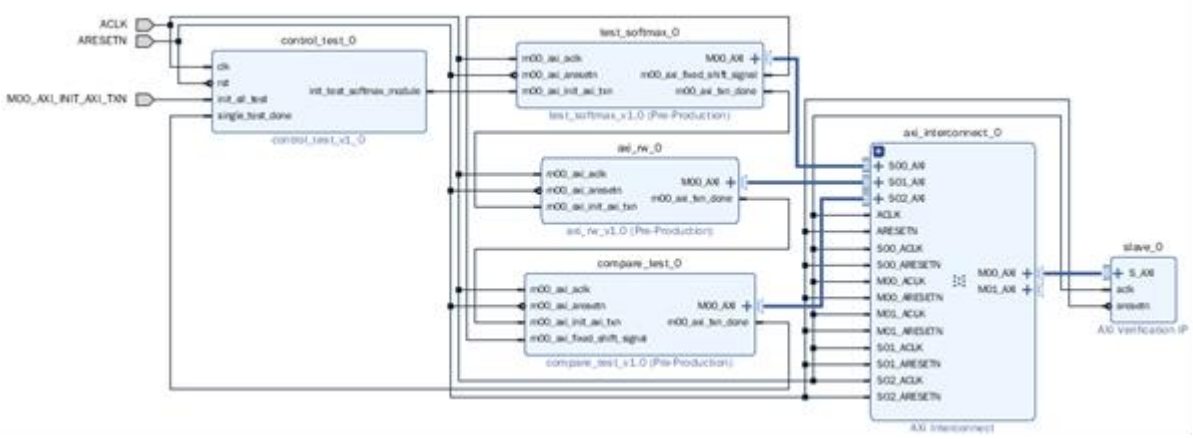


图 4.1.1

（1） **模块介绍**

重复发出启动信号,每次启动刷新将要输入的数据、控制寄存器值和用于验证的标准结果,检测到 done 信号后,记录验证的结果.

通过 axi-full 协议写入从机数据,包括用于被测模块读取寄存器和所需处理的数据.

被测模块,也就是我们设计的 ip 核.

读取从机中处理完成的数据,与标准结果比较,记录误差,并统计分布.

从机,存储数据,供主机读写用于主从机相连

（2） **测试效果**:

通过搭建完善的验证平台,一键仿真,自动重复启动,共启动 8 次(验证 16/ 8 位两种数据格式),自行输入赛方提供的四个测试集和对应的标准结果,然后处理、比较和记录误差.

最终得到记录的误差集和统计分布,证明了我们设计的 ip 核功能正确,并且误差较低,精度较高.

（3） **误差分析**:

```
The result of test_rand0.1 (16bit):
cnt_differ_0 =        1948
cnt_differ_1 =        2148
cnt_differ_2 =           0
cnt_differ_more =           0
The sum is        4096

The result of test_rand1 (16bit):
cnt_differ_0 =        2119
cnt_differ_1 =        1977
cnt_differ_2 =           0
cnt_differ_more =           0
The sum is        4096

The result of test_rand5 (16bit):
cnt_differ_0 =        2791
cnt_differ_1 =        1305
cnt_differ_2 =           0
cnt_differ_more =           0
The sum is        4096

The result of test_rand10 (16bit):
cnt_differ_0 =        3168
cnt_differ_1 =         906
cnt_differ_2 =          22
cnt_differ_more =           0
The sum is        4096

The result of test_rand0.1 (8bit):
```

```
cnt_differ_0 =        4096
cnt_differ_1 =           0
cnt_differ_2 =           0
cnt_differ_more =           0
The sum is        4096

The result of test_rand1 (8bit):
cnt_differ_0 =        4096
cnt_differ_1 =           0
cnt_differ_2 =           0
cnt_differ_more =           0
The sum is        4096

The result of test_rand5 (8bit):
cnt_differ_0 =        4096
cnt_differ_1 =           0
cnt_differ_2 =           0
cnt_differ_more =           0
The sum is        4096

The result of test_rand10 (8bit):
cnt_differ_0 =        4015
cnt_differ_1 =          81
cnt_differ_2 =           0
cnt_differ_more =           0
The sum is        4096
```

再整合数据在一张表上，如下表所示。

	16位数据格式				8位数据格式			
误差	rand0.1	rand1	rand5	rand10	rand0.1	rand1	rand5	rand10
0	1948	2119	2791	3168	4096	4096	4096	4015
1	2148	1977	1305	906	0	0	0	81
2	0	0	0	22	0	0	0	0
more	0	0	0	0	0	0	0	0
sum	4096	4096	4096	4096	4096	4096	4096	4096

表中列出 8 次重复启动后的数据误差分布，可以看到数据的误差集中在 0 和 1，说明误差相当小（等同于浮点误差 1/2^16 以内）。

3. FPGA 开发板验证

开发板品牌:米联客

芯片型号:xilinx zynq-7010

速度等级:-1

封装类型:clg400

图 4.1.2

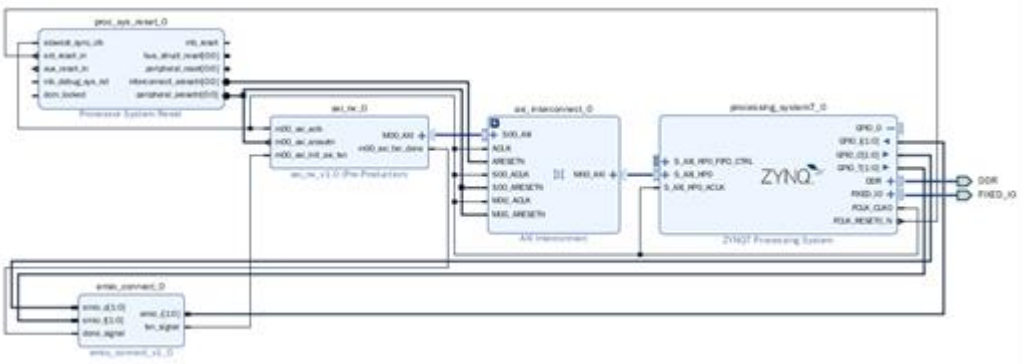


图 4.1.3

4. sdk c 语言测试:

（1） 初始化 emio 和 sd 卡,关闭 cache

（2） 读取 sd 卡中的 rand0.1 测试集(浮点),转换为定点值,写入 ddr

通过 emio 启动 ip 核,再通过 emio 循环检测完成信号,当检测到完成信号,退出循环

计算 softmax 标准值(cmodel 值),读取 ddr,两者比较,记录误差写入 sd 卡,进行误差统计

（3） 重复 2 的步骤,测试 rand1、rand5 和 rand10

5. 开发板验证结果(统计分布):

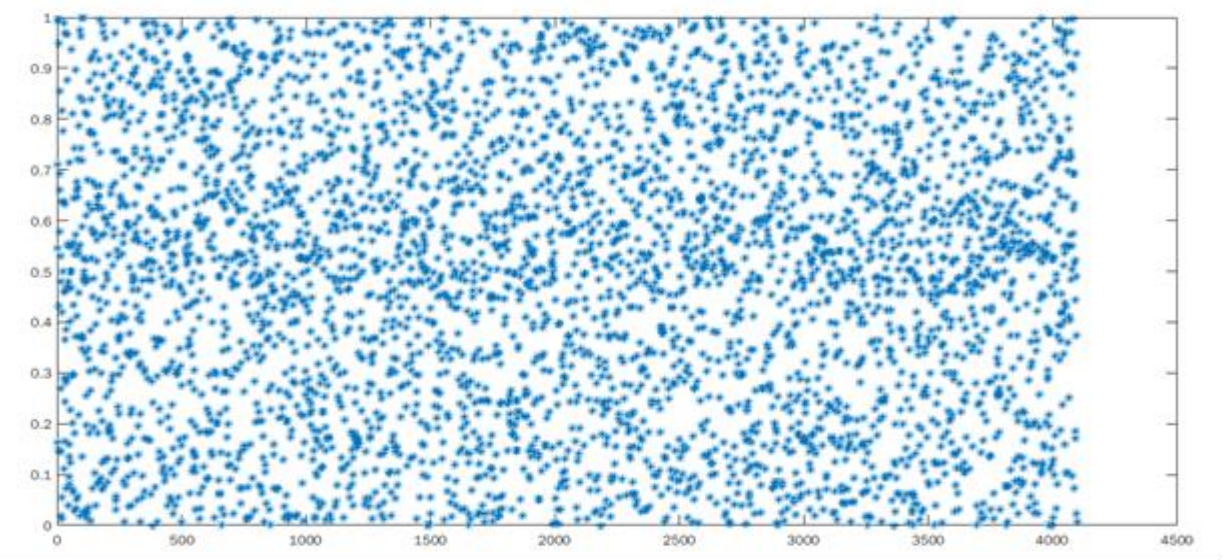
```
Start read rand0.txt.
start exp,and max:0.099900
Success! Complete rand0.txt.
show the error counter:
0 to 1:4096
1 to 2:0
2 to 3:0
more :0
rand0.txt error total:4096

Start read rand1.txt.
start exp,and max:0.999000
Success! Complete rand1.txt.
show the error counter:
0 to 1:4082
1 to 2:14
2 to 3:0
```

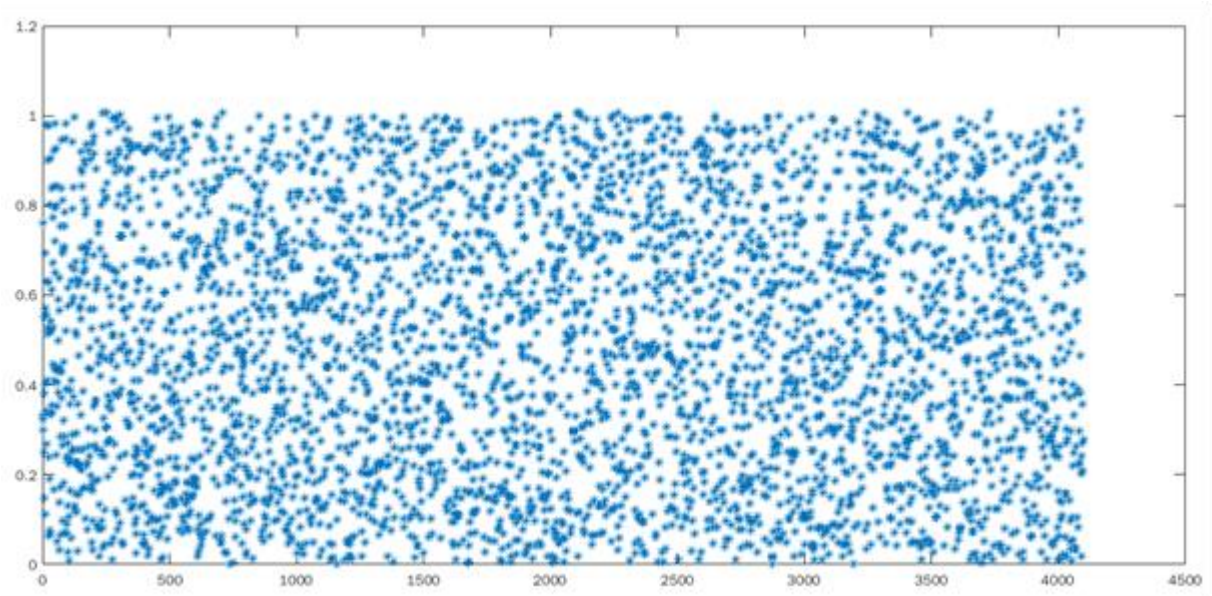
```
more :0
rand1.txt error total:4096

Start read rand5.txt.
start exp,and max:4.995000
Success! Complete rand5.txt.
show the error counter:
0 to 1:3830
1 to 2:266
2 to 3:0
more :0
rand5.txt error total:4096

Start read rand10.txt.
start exp,and max:9.990000
Success! Complete rand10.txt.
show the error counter:
0 to 1:3803
1 to 2:293
2 to 3:0
more :0
rand10.txt error total:4096
```

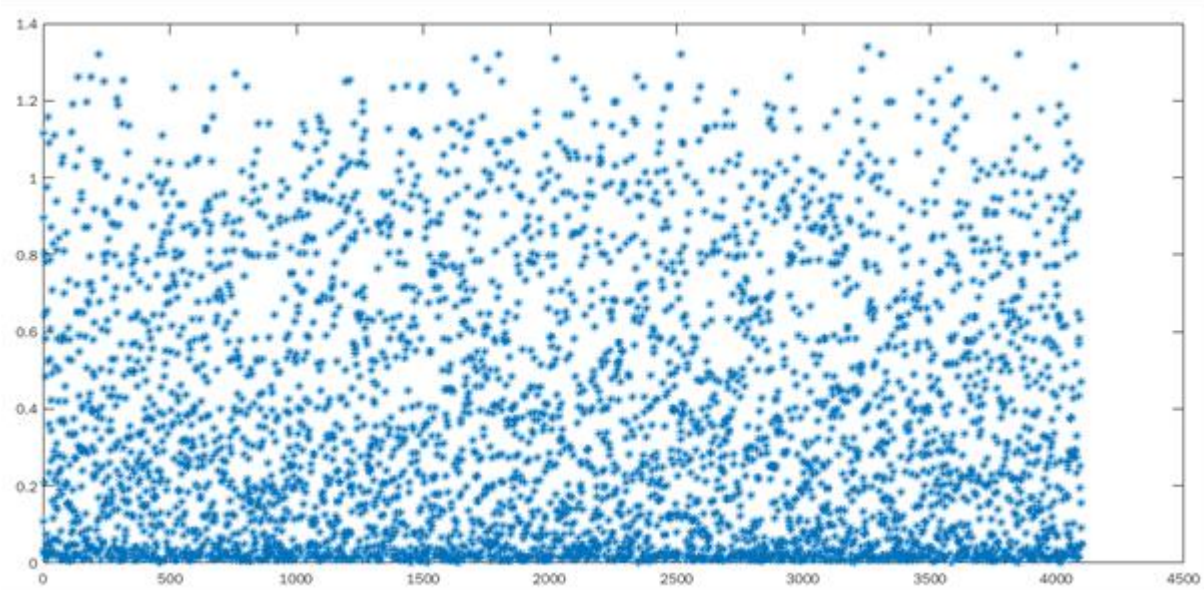
写回数据通过 matlab 显示误差离散点：

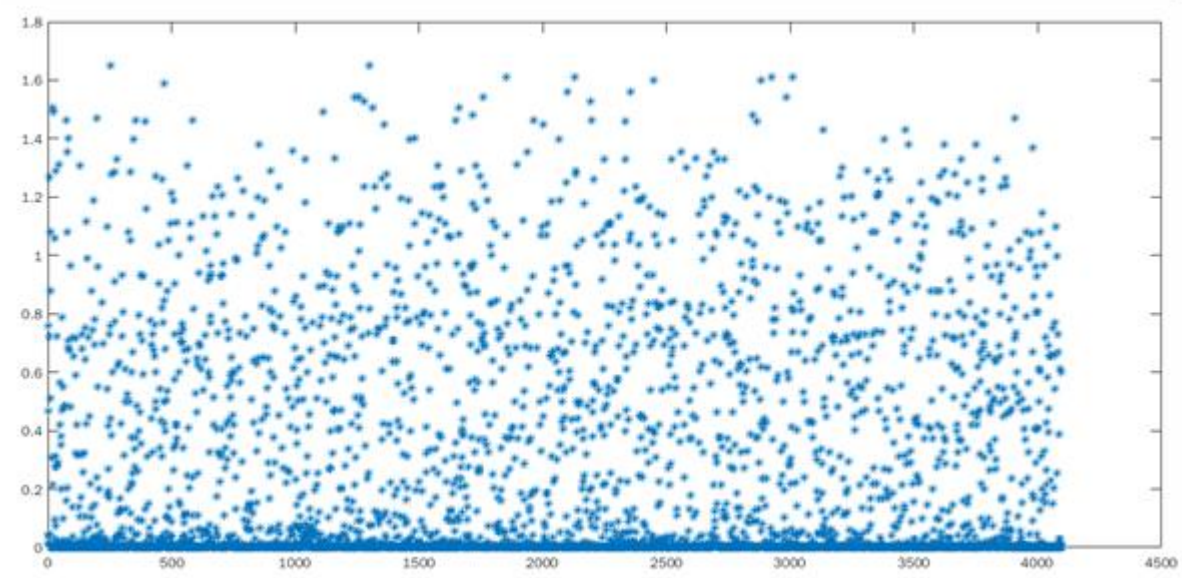
下图分别是 rand0.1，rand1，rand5，rand10 的 16 位数据格式的结构。

需要注意的是，误差的纵坐标是折算成与定点输出一致的单位，也就是乘以 2 的 16 次方的结果。









通过观察，这四个误差少部分最大不超过 2/2^16，普遍集中于 1/2^16 以内。因为 fpga 计算出来的结果是用定点表示的，所以 1/2^16 以内无法避免，最终可以得到我们结果精度足够高的结论。

第五章 性能评估

一、 频率

时序优化:

在初赛提交后,我们对主时钟与门控时钟上边沿交换数据的关键路径优化,大大提高了频率.

频率测试:

125M vivado 未出现时钟违例,最差路径 slack>0,如下图 6.1.1 所示。

125M 实际板上验证,测试结果符合,说明时序满足

130M synplify 极限频率,最差路径 slack 接近 0, 如下图 6.1.2 所示。

133.3M vivado 出现时序违例,最差路径 slack<0

150M 实际板上验证,测试结果与 125M 一致,说明时序满足

175M 实际板上验证,测试结果与 125M 一致,说明时序满足

200M 实际板上验证,测试结果与 125M 一致,说明时序满足

225M 实际板上验证,测试结果与 125M 不一致,时序不满足

250M 实际板上验证,测试结果与 125M 不一致,时序不满足

(备注:开发板中提供的接口时钟最大为 250M)

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.204 ns	Worst Hold Slack (WHS):	0.037 ns	Worst Pulse Width Slack (WPWS):	2.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6942	Total Number of Endpoints:	6942	Total Number of Endpoints:	3625

All user specified timing constraints are met.

6.1.1

```
Requested Frequency:    130.0 MHz
Wire load mode:         top
Paths requested:        5
Constraint File(s):
@N:MT320 :  | Timing report estimates place and route data. Please look at the place and route timing report for final timing.

@N:MT322 :  | Clock constraints cover only FF-to-FF paths associated with the clock.


Performance Summary
*******************

Worst slack in design: 0.006

                           Requested     Estimated     Requested     Estimated               Clock        Clock
Starting Clock             Frequency     Frequency     Period        Period        Slack     Type         Group
------------------------------------------------------------------------------------------------------------------
axi_rw_v1_0|m00_axi_aclk   130.0 MHz     130.2 MHz     7.692         7.680         0.006     inferred     Inferred_clkgroup_0
==================================================================================================================



Clock Relationships
*******************

Clocks                                          |    rise  to  rise   |    fall  to  fall   |    rise  to  fall   |    fall  to  rise
------------------------------------------------------------------------------------------------------------------------------------
Starting                 Ending                 |  constraint  slack  |  constraint  slack  |  constraint  slack  |  constraint  slack
------------------------------------------------------------------------------------------------------------------------------------
axi_rw_v1_0|m00_axi_aclk  axi_rw_v1_0|m00_axi_aclk  |  7.692       2.628  |  7.692       2.220  |  3.846       0.006  |  3.846       3.050
====================================================================================================================================
 Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
       'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.
```

6.1.2

预计开发板实际跑得最大频率准确值在 200M 到 225M 之间，可以得出，我们的设计实际跑得的频率是相当高的.另一方面，我们的设计中充分利用 axi4-full 读写通道可以同时进行的特性，而且 16 位数据格式 4 路并行，8 位数据格式 8 路并行，将速率提高到极致。毫无疑问，一个很高的时钟频率，再加上多路并行，这将大大缩短处理的时间。也可以说，我们的设计速度上达到了接近极限速度。

二、 面积

LUT	FF	BRAM	URAM	DSP
5192	3094	1	0	16

Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Block RAM Tile (60)	DSPs (...	Bonded IOPADs (130)	BUFGCTRL (32)
axi_rw_0 (axi_arm_r...	5192	3084	18	1909	5044	148	1653	1	16	0	2
inst (axi_rw_v1_0)	5192	3084	18	1909	5044	148	1653	1	16	0	0
axi_rw_v1_0_...	5123	3084	18	1900	4975	148	1618	1	16	0	0
accumulato...	435	39	0	429	435	0	35	0	0	0	0
divider_fixe...	170	93	0	71	162	8	49	0	0	0	0
divider_fixe...	170	93	0	69	162	8	49	0	0	0	0
divider_fixe...	170	93	0	71	162	8	49	0	0	0	0
divider_fixe...	170	93	0	70	162	8	49	0	0	0	0
divider_fixe...	628	333	0	225	614	14	171	0	0	0	0
divider_fixe...	632	333	0	224	618	14	175	0	0	0	0
divider_fixe...	632	333	0	230	618	14	175	0	0	0	0
divider_fixe...	632	333	0	230	618	14	175	0	0	0	0
fifo_u0 (fifo)	170	16	1	88	126	44	11	0	0	0	0
find_max (f...	158	108	8	79	158	0	17	0	0	0	0
operation_f...	28	42	0	25	24	4	12	0.5	0	0	0
operation_f...	26	42	0	25	22	4	12	0	0	0	0
operation_f...	27	42	0	20	23	4	12	0.5	0	0	0
operation_f...	26	42	0	25	22	4	12	0	0	0	0
operation_f...	105	120	0	68	105	0	44	0	4	0	0
operation_f...	108	120	0	61	108	0	44	0	4	0	0
operation_f...	108	120	0	70	108	0	44	0	4	0	0
operation_f...	658	129	0	238	658	0	76	0	4	0	0

三、 精度

在顶层行为级仿真和开发板验证，我们已经进行了精度上的分析，这里不做赘述。