电子科技大学课程设计报告

报告题目：MCU 设计

报告完成人：刘涵章（2017000203008）

报告完成人：张天祺（2017020901008）

报告完成人：罗成学（2017040205019）

报告完成时间：2019 年 7 月 5 日

1 设计目标

1.设计一个程序在所设计 MCU 能够完成对 64 个随机数的排序（从小到大），包括正负数。具体要求如下：
①芯片为通用型架构，功能全部以 MCU 指令完成，不能做成专用硬件芯片；
②设计一个计数器用于计时；
③数据类型为定点带符号的 16 位整数；
④从外部 ROM 中读取数据，完成排序操作后，一次性将数据读出存储到外部的 RAM 中；
⑤设计需要上板实现。
2. 设计一个程序在所设计 MCU 完成对 8x8 矩阵乘以 8x1 向量的矩阵乘法。具体要求如下：
①芯片为通用型架构，功能全部以 MCU 指令完成，不能做成专用硬件芯片；
②设计一个计数器用于计时；
③输入数据为 16 位定点数（1 位符号位，3 位整数位，12 位小数位），输出数据为 16 位定点数（1 位符号位，6 位整数位，9 位小数位）
④从外部 ROM 中读取数据，完成排序操作后，一次性将数据读出存储到外部的 RAM 中；
⑤设计需要上板实现。

2 提高计算并行性：从算法开始

我们的 MCU 采用了六级流水、四倍超标量设计，四条指令并行执行。为了尽可能避免指令冲突，减少硬件冲突控制单元的面积，必须要选择适合并行计算的算法以及冲突最少的指令顺序。矩阵的乘法本身就是一个并行性较强的计算，因此无需特别选择算法，只需稍微调整指令顺序即可。在排序运算中，我们选择了双调排序算法。

2.1 双调排序算法

首先，我们来定义一个名词“双调序列”：一个先单调递增后单调递减（或者先单调递减后单调递增）的序列。将任意一个长为$2n$的双调序列A分为等长的两半X和Y，将X中的元素与Y中的元素一一按原序比较，即$a[i]$与$a[i+n]$ $(i < n)$比较，将较大者放入 MAX 序列，较小者放入 MIN 序列。则得到的 MAX 和 MIN 序列仍然是双调序列，并且 MAX 序列中的任意一个元素不小于 MIN 序列中的任意一个元素。这是 Batcher 定理。

假设我们有一个双调序列，则我们根据 Batcher 定理，将该序列划分成 2 个双调序列，然后继续对每个双调序列递归划分，得到更短的双调序列，直到得到的子序列长度为 1 为止。这时的输出序列按单调递增顺序排列。

以具有 16 个数的双调序列为例，前面八个数为升序排序，后面八个数为降序排列，如图 1 所示：

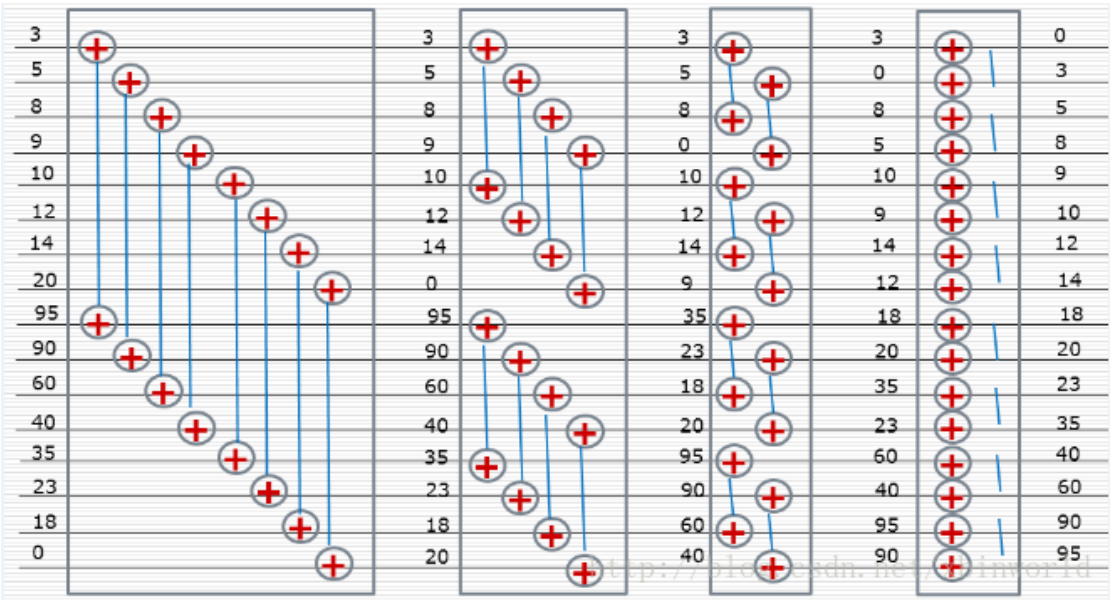

图 1 长为 16 双调序列的 Batcher 定理

首先进行第一轮，将第一个数和第九个数作比较，小的放上面。再将第二个数和第十个数作比较，小的放上面。如此类推，直至所有数都参与了比较。在第二轮中，将十六个数分成两组，每组八个数，在每一组内部执行与第一轮相同的操作。如此类推，在第三轮中分成四组，第四轮中分成八组……直至分组长度为 1 时停止。最终结果便是十六个数从小到大的排列。

上述的排序算法，只适用于双调序列，因此需要考虑如何将任意序列转成双调序列。下面是一种实现方法：将

两个相邻的，单调性相反的单调序列看作一个双调序列，每次将这两个相邻的、单调性相反的单调序列生成一个新的双调序列，然后对其进行双调排序。这样只要每次两个相邻长度为n的序列的单调性相反，就可以通过连接得到一个长度为$2n$的双调序列，然后对这个$2n$的序列进行一次双调排序变成有序，然后在把两个相邻的$2n$序列合并（在排序的时候第一个升序，第二个降序）。n开始为1，每次翻倍，直到等于数组长度，最后就只需要再一遍单方向（单调性）排序了。

同样以 16 个数的任意序列为例，展示如何生成双调序列，如图 2 所示：

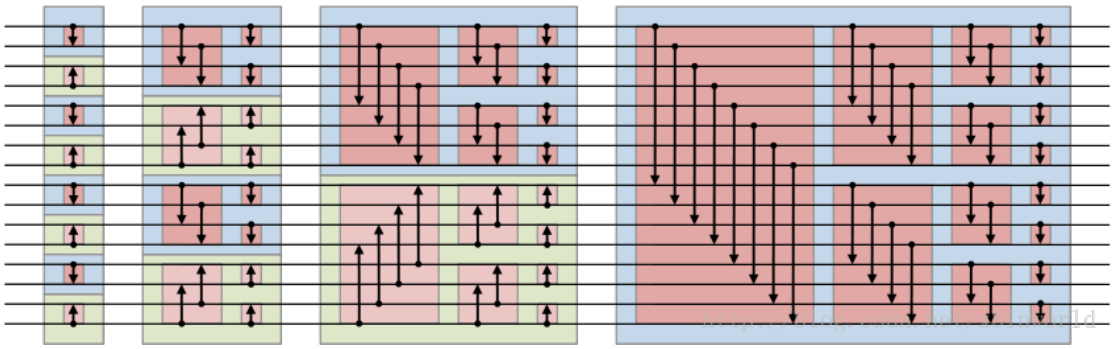

图 2 长为 16 任意序列生成双调序列

第一轮中，将第一第二个数按升序排列，将第三第四个数按降序排列，第五第六个数按升序排列……如此交替重复，直至所有数均参与排列。第二轮时，将第一个数到第二个数看作是四个数的双调序列，并对其进行双调排序。将第五个数到第八个数看作四个数的双调序列，进行“反双调排序”（即升序和降序和双调排序相反，最后得到的序列是从大到小排列的）……如此交替重复直至所有数均参与排列。后面轮次操作类似，分别是八个数的双调序列，十六个数的双调序列……直至在一个双调序列中包含原始序列的所有数，算法停止。最终得到一个先升序排列后降序排列的双调序列。如此的双调序列便可以应用双调排序算法进行排序。

从上述过程可以看出，不管是任意序列生成双调序列，还是双调序列的排序，在每一轮操作中，每两个数据是相互独立的，即一次操作的结果不会影响同一轮中的另一操作。因此，双调排序十分适合用于并行计算。

2.2 矩阵乘法

设计目标是完成8×8矩阵A与8×1向量B的乘法。矩阵A中每一行向量与B的积都是一个独立的 8 次乘累加运算，非常适合并行计算。

3 指令集、寄存器与汇编代码

3.1 排序指令

由于我们的寄存器和 ALU 是 32 位，而需要比较的数是 16 位，因此我们分别使用高十六位和低十六位装入两个不同的数据。有 64 个数，因此需要用到 32 个寄存器。根据上面对双调排序算法的分析，排序比较既可以在同一寄存器的高低位间发生，也可在不同寄存器的高位间或低位间发生；排序既有升序排序，也有降序排序，因此需要四条指令，如表格 1 所示。值得注意的是，我们的 32 位 ALU 可以同时比较两个寄存器，即两对 16 位数。

3.2 乘累加指令

矩阵乘法用到的操作较为简单，只有一条乘累加指令，具体操作见表格 1.

表格 1 指令集设计

指令	操作
mulcum $a, $b, $m	[$m] = [$a][31:16]*[$b][31:16]+ [$a][15:0]*[$b][15:0] + [$m]
cpia $a, $b	分别在 a、b 寄存器的高低位间进行升序排序
cpid $a, $b	分别在 a、b 寄存器的高低位间进行降序排序
cpxa $a, $b	在 a、b 寄存器的高位与高位、低位与低位之间进行升序排序
cpxd $a, $b	在 a、b 寄存器的高位与高位、低位与低位之间进行降序排序

3.3 寄存器分配

排序的原始数据是 64 个 16bit 数，需要 32 个寄存器存放。矩阵乘法的原始数据是 72 个 16bit 数，需要 36 个寄存器存放。累加结果可以覆盖掉已经读取矩阵元素数据的寄存器。但由于四倍超标量设计，要求 Register File 有大吞吐（同时进行 12 个寄存器的读取和 8 个寄存器的写入操作）的能力，因此 Register File 的数据存储结构进行了重

新设计，矩阵乘法共使用 64 个寄存器，具体设计方法见 5.2 节。

3.4 汇编代码

如 2.1 节中所述，双调序列的排序在每一轮操作中，每两个数据是相互独立的，即一次操作的结果不会影响同一轮中的另一操作。因此每一轮操作中的汇编指令可以任意调换顺序。我们找到了最佳的指令顺序，使得数据冲突仅需重定向就可以解决并且需要重定向的次数最少。排序的汇编代码详见附录 A，冲突控制模块的设计详见 4.2 节

4 从算法并行到硬件并行：超标量 Data Path 设计

4.1 未加入冲突控制单元的 Data Path

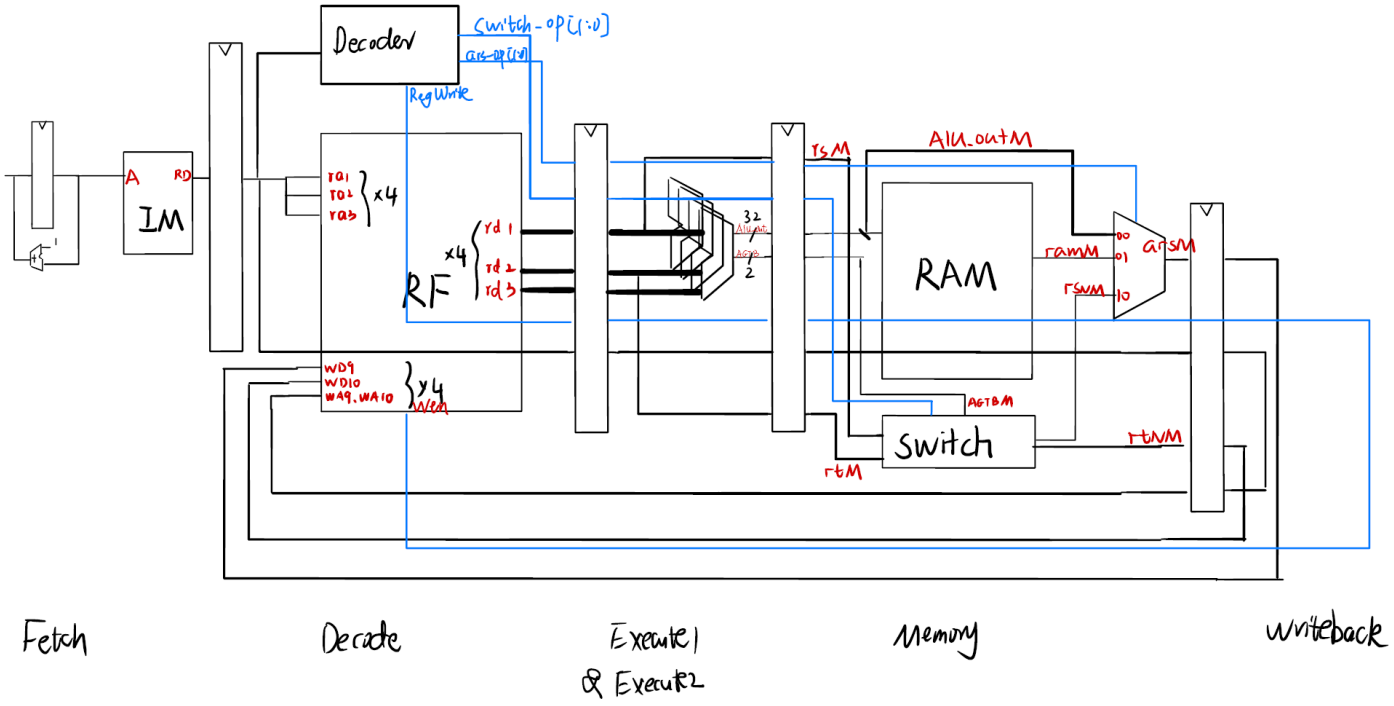

图 3 未加入冲突控制单元的 Data Path

未加入冲突控制单元四倍超标量 MCU 的 Data Path 如图 3 所示。ALU 有两级流水，因此整个流水线分为六个阶段，分别为 Fetch, Decode, Execute 1, Execute 2, Memory 以及 Writeback. 信号名最后的大写字母(F, D, E1, E2, M, W)表示该信号所处的流水线阶段。指令存储器接收一个地址，输出四条指令。因此，除指令存储器的输入信号外，图中所有信号都是 System Verilog 中深度为 4 的数组信号。Register File 能同时进行 12 个寄存器的读取操作和 8 个寄存器的写入操作。端口 ra1, ra2, ra3 读取的寄存器分别是表格 1 中的$a, $b 和$m 寄存器。译码器读取指令中 operation code，输出控制信号 switch_op, ars_op 和 RegWrite，分别作为 switch、Memory 阶段的 Mux 的控制信号和 Register File 的写使能信号。ALU 进行比较和乘累加运算，输出信号为乘累加结果(ALU_out)和比较结果(AGTB, A Greater Than B). Switch 模块接收 switch_op 和 AGTB 信号，在$a, $b 寄存器的高低位间进行相应的交换。$a, $b 寄存器交换后对应的信号为 rsNM 和 rtNM. Memory 阶段的 3-1Mux 从 ALU_out, RAM 读取的数据 ramM 和交换后$a 对应的数据 rsNM 之间选择一个输出。指令中$a, $b 和$m 寄存器的地址跟随流水线推进，并在 Writeback 阶段作为 Register File 的写入地址。

4.2 冲突控制单元

乘累加指令仅会产生$m 寄存器的冲突问题。由于 ALU 的累加操作在 Execute 2 阶段完成，而紧接着的下一阶段 M 阶段就能产生累加结果，因此用重定向就可以解决相邻两条使用同一个$m 寄存器的指令的冲突问题。乘累加指令的控制冲突单元设计如图 4 所示。ALU 为乘累加操作的重定向提供了一个数据端口和一个控制端口，连接 rd_reM 和 replace 信号。rd_reM 是从 M 阶段重定向的$m 寄存器的值，replace 信号为高电平时，ALU 的 E2 阶段选择重定向自 M 阶段的值作为操作数，为低电平时选择从 Register File 中读取的值作为操作数。冲突控制单元接收 E2 阶段 rd 寄存器的地址 ArsdE2 和 M 阶段 rd 寄存器的地址 ArsdM，以及指示当前 ALU 执行的操作是比较还是乘累加的 selcet 信号，若 E2 阶段和 M 阶段的地址一致，并且 ALU 在执行乘累加运算，则选择重定向的数据。

排序操作的冲突控制相对复杂一些，因为数据的重定向需要在 4 个 ALU 之间进行。通过对双调排序算法的分析，我们发现，任意一个寄存器，可能造成冲突的情况只有该寄存器在两个或三个阶段后再次被调用。此外，重定

向仅有可能是从一个 ALU 的数据通路重定向到另一个或者该 ALU，不可能出现一对多或多对一的情况。冲突单元接收 W 阶段寄存器的地址，与 E1 和 E2 阶段的地址进行比较，并产生控制信号。前缀为 replace 的信号高电平有效，决定 ALU 是否接收重定向的值，前缀为 mux 的信号选择重定向的数据来源。排序指令的冲突控制单元如图 5 所示，整个冲突控制单元的逻辑与实现见附录 A.

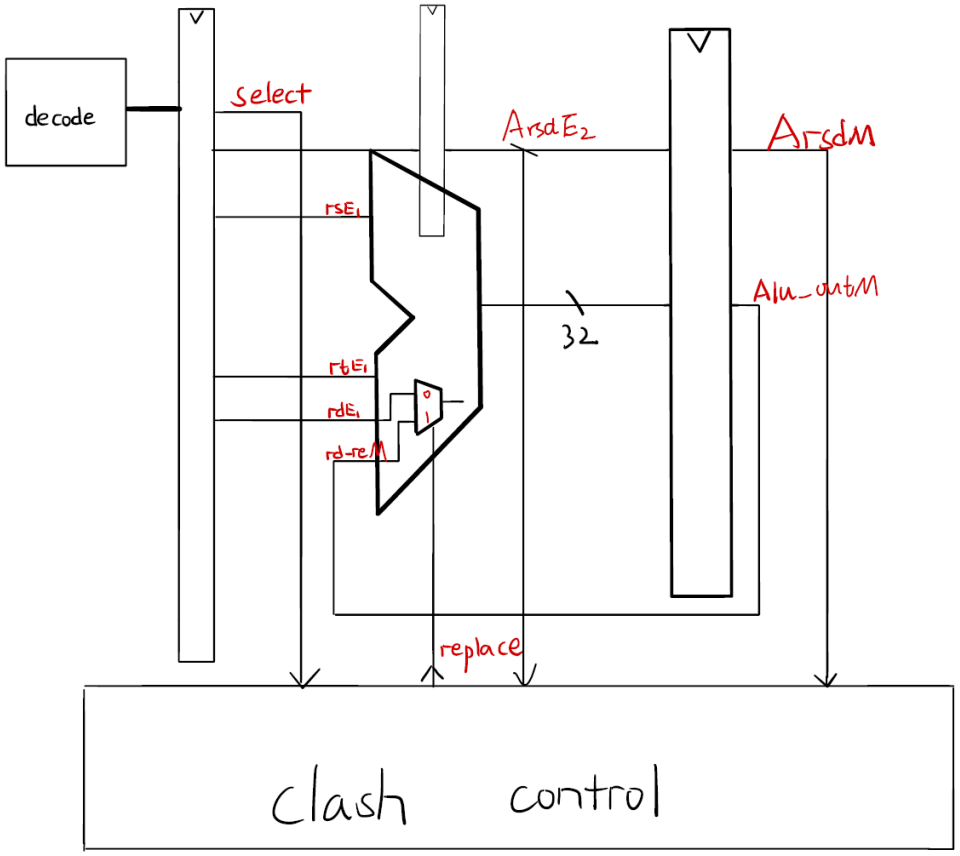

图 4 乘累加指令的冲突控制单元

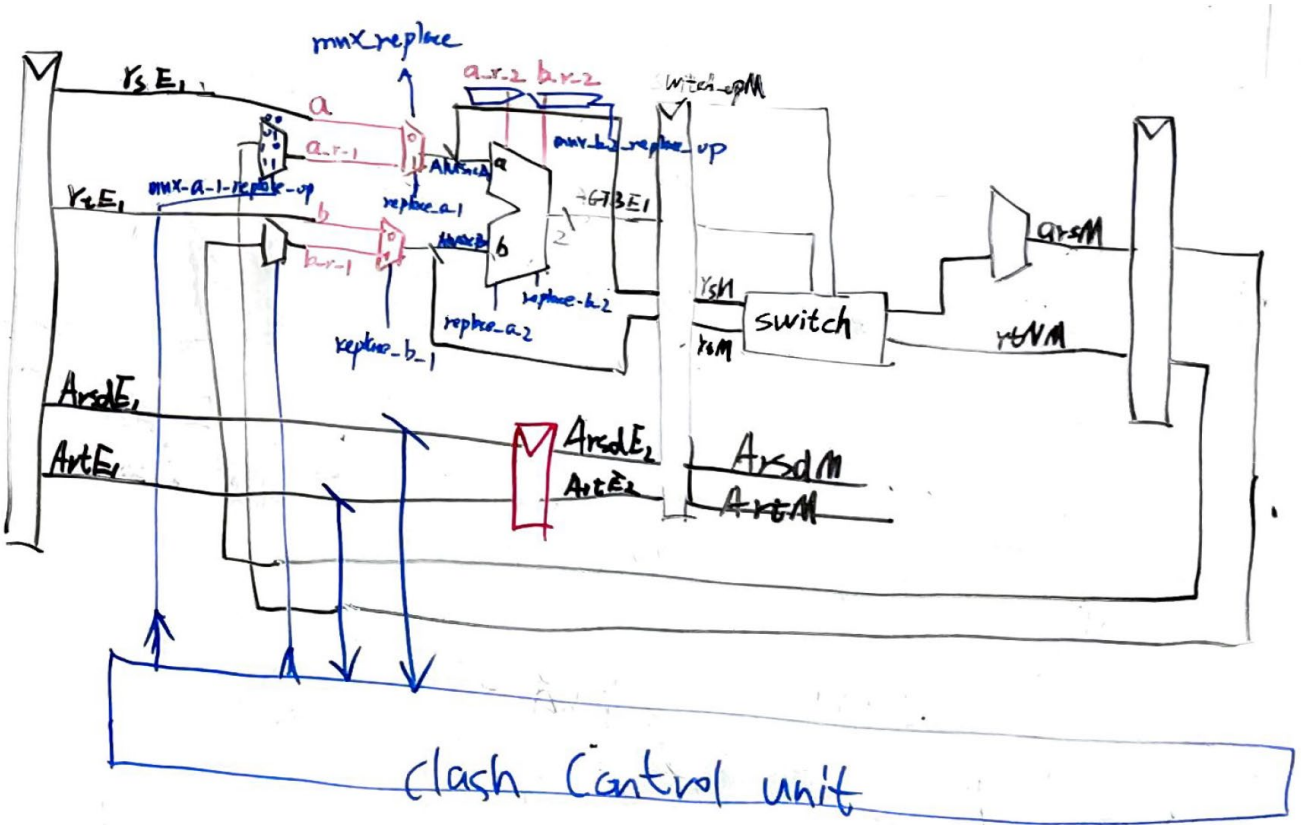

图 5 排序操作的冲突控制单元

5 提高速度的核心：存储与运算单元设计

乘加器使用一级超流水结构，完成两组 16bit 数积与第五个 32bit 数积的和运算，需要 524 个 LUT，36 个 Carry4，96 个 FF.关键路径延时 4.7ns. Register File 实现了最多 32bit 8 并入同时 32bit 12 并出操作，需要 6 个 36Kbit 的 BRAM 和 1022 个 LUT.

5.1 基于整数规划的 Radix-4 Booth 算法和 Generalized Parallel Counter (GPC)压缩器的乘加器设计

5.1.1 Radix-4 Booth 算法介绍

Booth 算法以补码为基础，可以完成带符号数的计算。将两个数 A、B 表示成补码形式。

$$\mathrm{A} = -\mathrm{a}_{\mathrm{n}-1} \cdot 2^{\mathrm{n}-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$B = -B_{n-1} \cdot 2^{n-1} + B_{n-2} \cdot 2^{n-2} + \cdots + B_1 \cdot 2^1 + B_0 \cdot 2^0$$
$$= -\mathrm{B}_{\mathrm{b}-1} \cdot 2^{n-1} + (2B_{n-2} - B_{n-2}) \cdot 2^{n-2} + \cdots + (2\mathrm{B}_0 - \mathrm{B}_0) \cdot 2^0$$
$$= \left(-\mathrm{B}_{\mathrm{n}-1} + \mathrm{B}_{(\mathrm{n}-2)}\right) \cdot 2^{\mathrm{n}-1} + (-\mathrm{B}_{\mathrm{n}-2} + \mathrm{B}_{\mathrm{n}-3}) \cdot 2^{\mathrm{n}-2} + \cdots + (-\mathrm{B}_1 + B_0) \cdot 2^1 + (-B_0 + 0) \cdot 2^0$$
$$= \sum_{\mathrm{i}=0,\mathrm{B}_{-1}=0}^{\mathrm{n}-1} (-B_i + B_{i-1}) \cdot 2^i$$
$$= \sum_{\mathrm{i}=0}^{\mathrm{n}-1} E_i \cdot 2^i$$

其中$E_i = -B_i + B_{i-1}, B_{-1} = 0$. 于是，可以将 AB 的积表示成

$$P = A \times B = A \times \sum_{i=1}^{\frac{n}{2}-1} E_i \cdot 4^i = \sum_{i=1}^{\frac{n}{2}-1} A \times E_i \cdot 4^i$$

于是可以将 Booth 编码表示成表格 2 的形式。

表格 2 基 4 Booth 编码

$\mathbf{B}_{i+1}\boldsymbol{B}_i\mathbf{B}_{i-1}$	$\mathbf{E_i}$	操作
000	0	+0
001	+1	+A
010	+1	+A
011	+2	+2A
100	-2	-2A
101	-1	-A
110	-1	-A
111	0	0

为了实现并行运算，需要同时产生所有的部分积项，将补码的产生的+1 运算保留在下一行。一个例子如下所示。被乘数的最后两位是 01，加上补充的-1 位产生结果$A \cdot 4^0$，第二行产生结果 $-2A \cdot 4^1$,类似的将产生 8 行结果。值得注意的是结果使用补码表示，需要对乘数 A 使用符号拓展。产生的低位结果中会有大量符号拓展不包含信息。可以将其压缩起来，同时为了规整整个结构，将补码的进位与上一行结果进行运算，产生的进位保留。压缩结果如图 6 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	-1
																1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0
																1	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	0
																-1		-1		2		-2		0		1		-2		1		
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	1	1	0	1	1		0	-2
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0		1			1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0					0
0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	1	1	0	1	1		0							-2
1	1	1	1	1	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0		1									2
0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	1	1	0	1		0											-1
0	0	0	1	1	0	0	1	0	0	0	1	1	0	1	1	0	1		1													-1
																	1															

图 6 Booth 乘法器部分积生成样例

5.1.2 设计总体结构

如图 7 所示，先并行生成部分积结果，再进行若干级的压缩器使输出结果位 3 位，最后将累加结果 c 和两个乘法结果进行就求和可以得到乘法结果。

为了方便实现超流水的重定向控制，将 c 的加法放在第二阶段进行，同时留出将 c 改变的接口以便于重定向。部分积生成器使用 Radix4 Booth 算法生成部分积。将把 m 输入加法压缩成 n 输入加法(m>n)的电路称为压缩器。GPC 是将2^N-1 bit 的无权数转换为N bit 的 2 进制数的器件，使用若干级 GPC 可以实现压缩器的功能。全加器可以视为（3;2）GPC。三进制加法器可以视为一次将计算 3 输入二进制数的和的器件。

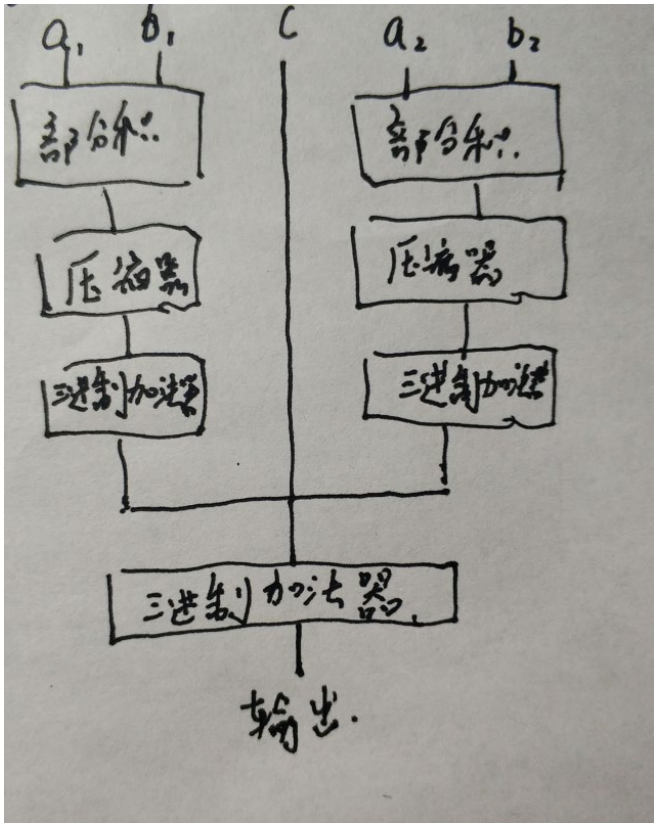

图 7 ALU 结构

5.1.3 部分积生成

由于 A7 系列 FPGA 的一个 LUT 单元最大可配置 6 输入 1 输出 LUT 或共享 5 输入 2 输出 LUT，可以将每一位的部分积结果使用 LUT 查表得到。A7 系列 FPGA 的 CLB 布局如图 8。

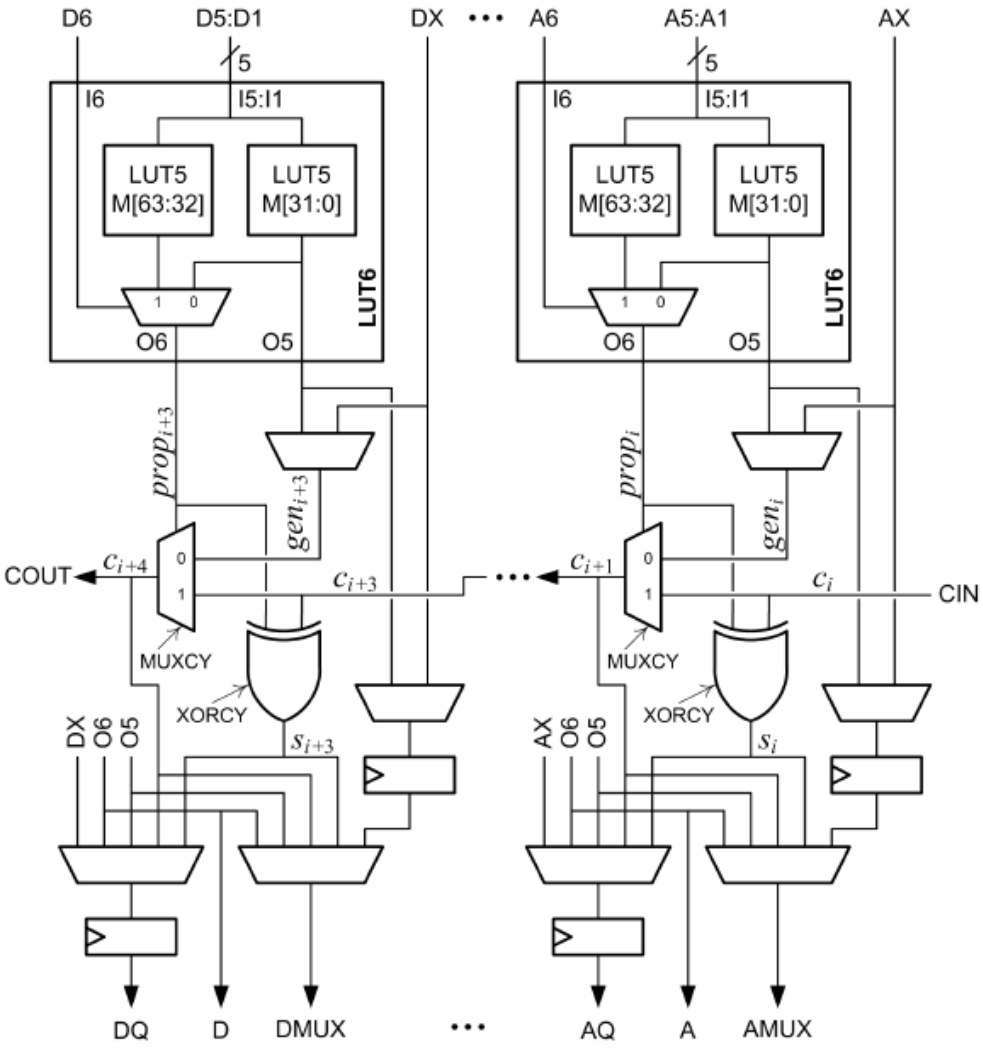

图 8 A7 系列 FPGA 的 CLB 布局

对于中间的部分称为 A 盒，输入 5 个数据$a_i, a_{i-1}, b_{i+1}, b_i, b_{i-1}$输出结果$c_{s,i}$. 与符号结果相关的输出称为 S 盒，S 盒的反称为 S'盒，输入 4 个数据$a_{15}, b_{i+1}, b_i, b_{i-1}$.输入输出关系如表格 3 所示。

表格 3 部分积生成盒

b_{i+1}, b_i, b_{i-1}	A 盒	S 盒	S'盒
000	0	0	0
001	a_i	a_{15}	$\overline{a_{15}}$
010	a_i	a_{15}	$\overline{a_{15}}$
011	a_{i+1}	a_{15}	$\overline{a_{15}}$
100	$\overline{a_{i-1}}$	$\overline{a_{15}}$	a_{15}
101	$\overline{a_i}$	$\overline{a_{15}}$	a_{15}
110	$\overline{a_i}$	$\overline{a_{15}}$	a_{15}
111	0	0	0

B 盒与 C 盒处理补码与最后 1bit 进位相关数据，输入$a_0, b_{i+2}, b_{i+1}, b_{i+1}, b_i, b_{i-1}$，同时将参考两级的 Booth 编码和乘数的最低 bit，将结果输出位 1 位。生成的部分积阵列如图 9 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												S'	-	-	S	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B
												S'	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	C	
										1	S'	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	C			
								1	S'	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	C							
						1	S'	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	C								
				1	S'	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	C										
		1	S'	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	C												
1	S'	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	C														
																C															

图 9 规整化的部分积生成矩阵

5.1.4 压缩器设计

一些常见的压缩器的点表示法在文献[1]中表示出来，如图 10 所示。

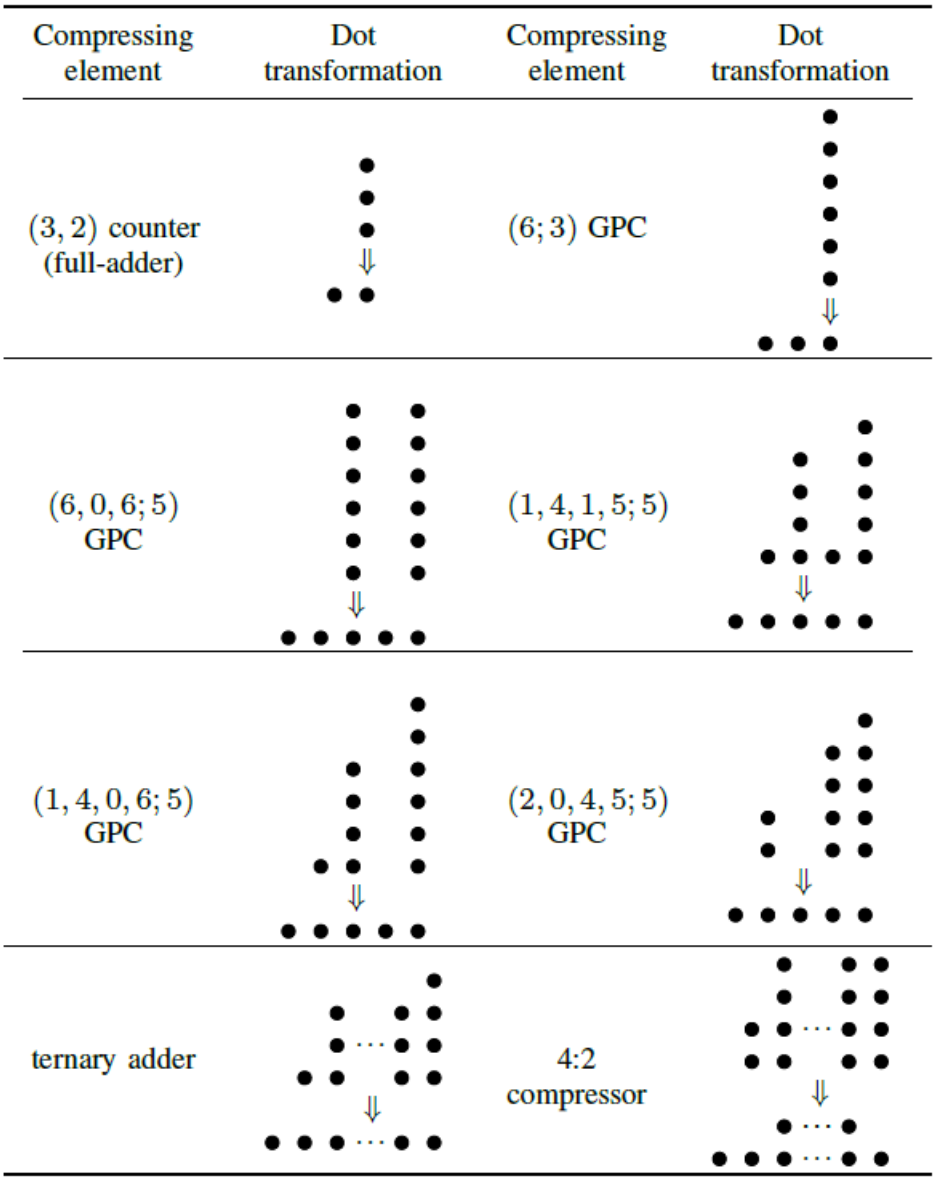

图 10 常见压缩器的点表示方法

将减少的 bit 数δ和使用的 LUT 数目 k 的比值称为 GPC 的效率。将查找表的延时记为τ_L，一级先行进位链的延时记为τ_{CC}，一个逻辑块的路由延时记为τ_R.在 A7 系列芯片中,数据手册给出了$\tau_{cc} \cong 20\tau_L \cong 20\tau_R$

表格 4 常见 GPC 的 LUT 数量、效率与逻辑延时

GPC	LUT	效率	逻辑延时
(3;2)	1	1	$\tau_L \cong \tau$
(6;3)	3	1	$\tau_L \cong \tau$
(1,5;3)	3	1	$\tau_L \cong \tau$
(2,3;3)	2	1.5	$\tau_L \cong \tau$
(7;3)	3	1.33	$2\tau_L + \tau_R + 3\tau_{CC} \cong 3\tau$
(5,3;4)	3	1.33	$2\tau_L + \tau_R + 3\tau_{CC} \cong 3\tau$
(6,2;4)	3	1.33	$2\tau_L + \tau_R + 3\tau_{CC} \cong 3\tau$
(5,0,6;5)	4	1.5	$\tau_L + 4\tau_{CC} \cong \tau$
(1,4,1,5;5)	4	1.5	$\tau_L + 4\tau_{CC} \cong \tau$
(1,4,0,6;5)	4	1.5	$\tau_L + 4\tau_{CC} \cong \tau$
(2,0,4,5;5)	4	1.5	$\tau_L + 4\tau_{CC} \cong \tau$
(6,0,6;5)	4	1.75	$2\tau_L + 4\tau_{CC} + \tau_R \cong \tau$
(1,3,2,5;5)	4	1.5	$\tau_L \cong \tau$

在文献[1][2][3][4]中使用 1 个 6 LUT 封装两个全加器，将其与 Carry4 连接构造出各种 GPC 电路，例如文献[2]中将（5,0,6;5）GPC 表示成图 11。

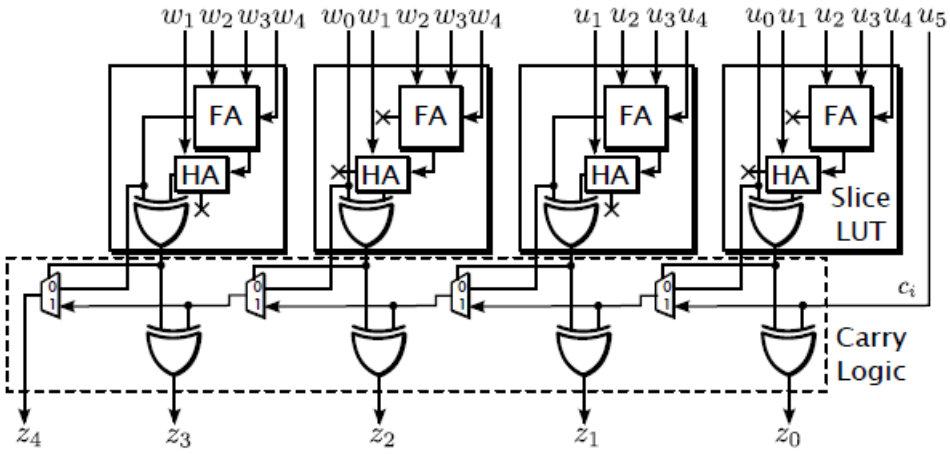

图 11 （5，0，6；5）GPC 结构

考虑到较多输入输出的 GPC 意味着较大的 fanout，测试发现当 fanout 达到 3 之后会产生极大的线延迟。所以在本设计中只使用前 4 种 GPC。结果可以直接使用 LUT6 查表或 LUT6_2 实现。

5.1.5 三进制加法器设计

由于在计算中需要使用 3 输入的加法器，所以可以使用三进制加法器一次完成这样的操作。

文献[5]中提出的三进制加法器采用如图 12 的设计。可以将其改进，使得将第一级的全加器和第二级全加器的半加器部分合并，加上第二级的全加器进位逻辑。改进结果如图 13，所需延时$\tau_L + k\tau_{CC}$

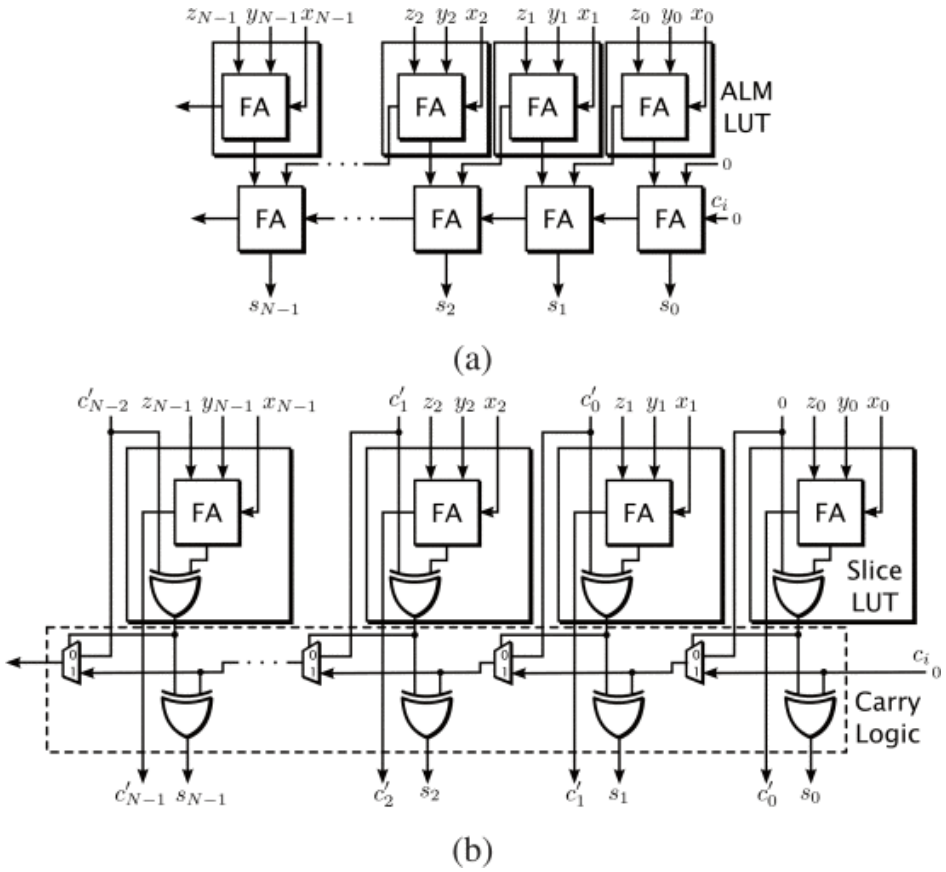

图 12 三进制加法器设计

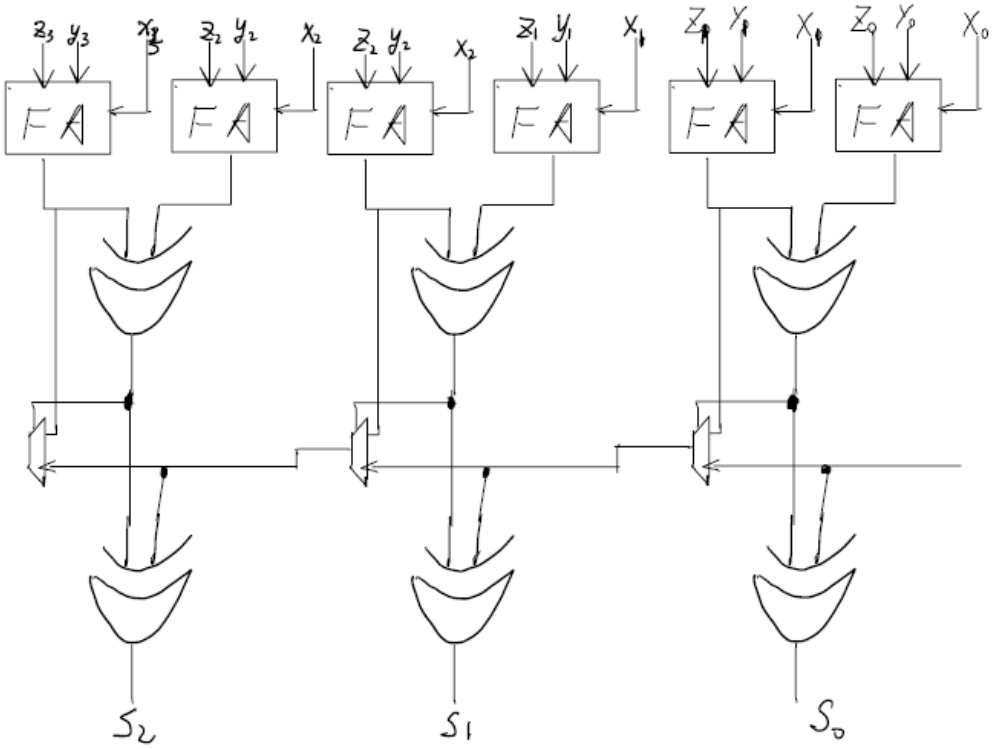

图 13 改进的三进制加法器设计

5.1.6 压缩器放置与整数规划

如何在最小的 GPC 级数下使用更少的 LUT 将 8 级的部分积压缩到 3 级是一个整数规划问题。定义符号s为级数，表明 GPC 在压缩器中的位置，e为 GPC 类型，c为 GPC 的最低位所在的列数。$K_{s,e,c}$表示在s级c列放置e种类 GPC 的数量，$M_{e,c}$表示为一个e种类 GPC 在c列压缩的个数，$K_{e,c}$表示e种类 GPC 在c列生成的个数，c_e.表示e压缩器

使用 LUT 的数量。$N_{s,c}$表示s级c列所存在的 bit 数。L是一个较大的整数，I是一个极大的整数。D_s指示第s级是否为最后一级，如果是则为 1.

目标函数：

$$z = min\{\sum_{s=0}^{S-1}\sum_{c=0}^{C-1}\sum_{e=0}^{E-1} c_e k_{s,e,c} + (s-1)LD_s\}$$

$$s.t.\begin{cases} N_{s-1,c} \le \sum_{e=0}^{E-1}\sum_{c'=0}^{C_e-1} M_{e,c+c'} k_{s-1,e,c+c'} + D_s I \\ N_{s,c} = \sum_{e=0}^{E-1}\sum_{c'=0}^{C_e-1} K_{e,c+c'} k_{s-1,e,c+c'} + D_s I \\ N_{s,c} \le 3 + I(1-D_s) \\ \sum_{s=0}^{S-1} D_s = 1 \\ N_{s,c}, k_{s,e,c} \in N_+ \\ D_s \in \{0,1\} \end{cases}$$

目标函数中sLD_s项指示了压缩器的级数，最小化了当D_s为 1 时s的值，第一项为 LUT 的总数量，最小化 LUT 使用。约束条件第一条表明一层 GPC 覆盖了所有的部分积项，第二条产生下一层的 bit 全部是来自与 GPC 的输出，第三条是约束边界为生成一个可以被三进制加法器接受的结果。

整数规划使用 LINGO14.0 求解，如果只使用前文所述的 4 种 GPC，将 16bit 乘法部分积结果压缩到每列不超过 3bit，最少需要 2 级 GPC 的压缩器电路。LINGO 生成优化程序详见附录 B。另外，提出了一种将 LINGO 输出结果自动生成 Verilog 代码的脚本，解决了使用 GPC 压缩器不规整，代码复杂的问题。

5.1.7 验证与测试

为了提高仿真的覆盖率，将数据使用$random 指令生成，同时使用行为级描述乘加操作作为对照组数据，两组数据进行判等。重复 10000 组数据，发现两数据始终相等（flag 信号始终为 1），认为已经覆盖了绝大多数情况，设计可靠。仿真结果如图 14：

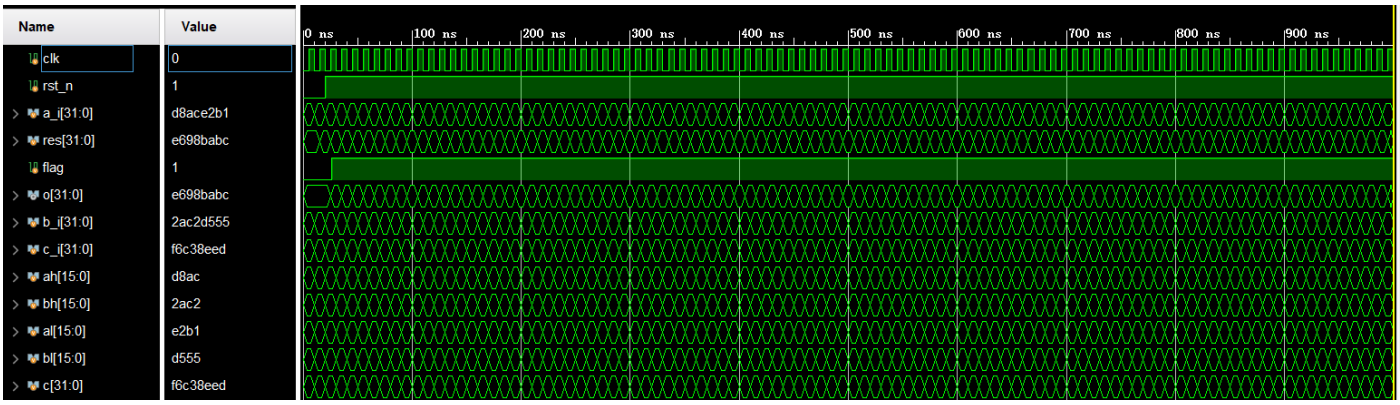

图 14 仿真测试

为了寻找 FF 的最佳位置，将乘加器卡在两级 FF 之间，第一级 FF 数据来自于 BRAM，第二级 FF 后接一个 BRAM。所有的综合器指令和布线器指令默认。

将 FF 插入不同的地方，关键路径延迟如表格 5 所示：

表格 5 FF 插入位置与关键路径延时关系

FF 位置	关键路径延时	关键路径位置
不插入	8.1ns	全局
部分积和第一级压缩器结果	7.3ns	II 阶段
第一级压缩器结果和第二级压缩器结果	6.4ns	II 阶段
第二级压缩器和第三级压缩器结果	5.3ns	I 阶段
第三级压缩器结果和第一级加法器结果	5.9ns	I 阶段
第一级加法器结果和第二级加法器结果	6.7ns	I 阶段

5.1.8 布线结果：桃心乘加器

布线器的指令设置如下，此时最佳情况关键路径延时 4.7ns.

```
set_property STEPS.PLACE_DESIGN.ARGS.DIRECTIVE ExtraNetDelay_high [get_runs impl_13]
set_property STEPS.PHYS_OPT_DESIGN.IS_ENABLED true [get_runs impl_13]
set_property STEPS.PHYS_OPT_DESIGN.ARGS.DIRECTIVE Explore [get_runs impl_13]
set_property STEPS.ROUTE_DESIGN.ARGS.DIRECTIVE Explore [get_runs impl_13]
set_property STEPS.POST_ROUTE_PHYS_OPT_DESIGN.IS_ENABLED true [get_runs impl_13]
set_property STEPS.POST_ROUTE_PHYS_OPT_DESIGN.ARGS.DIRECTIVE Explore [get_runs impl_13]
```

布线结果如图 15。有意思的是，它看起来像一个桃心。ALU 的详细代码详见附录 C.

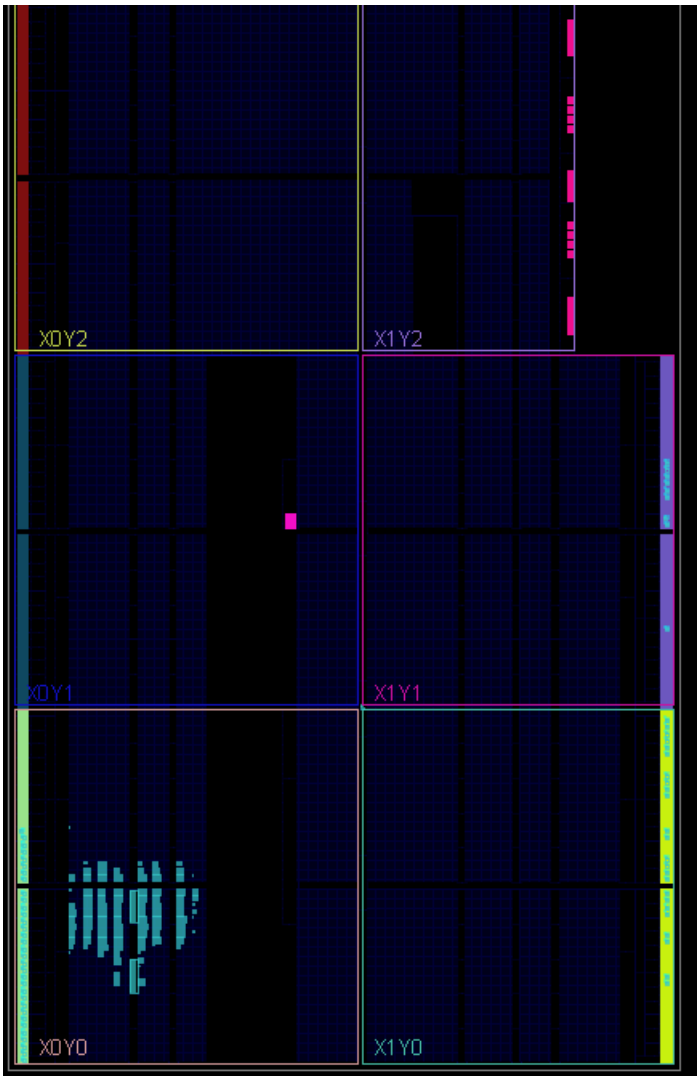

图 15 乘加器布线结果

5.2 8 并入 12 并出 Register File 设计

5.2.1 问题的提出

由于使用双调并行排序算法和乘累加并行算法，设计标量处理器运算个数为 4 个，所以乘加操作需要最多$3 \times 4 = 12$个并出项，双调排序操作需要最多$2 \times 4 = 8$个并入项。但是每一种算法都不要求访问全局寄存器。设计目标是完成一个允许部分访问的 register file，可以实现最大 8 并入 12 并出操作。

5.2.2 实现方法

由于这 8 个并入项是要求在一个周期内全部写入的，可以考虑采用跨时钟域的设计，让 register file 工作在更高的时钟频率。但是系统设计目标是 150Mhz，那么 register file 要求 300Mhz 是不现实的。所以考虑让每个 ALU 只允许访问部分 register file，同时共享一部分 register file 允许全局访问。为了使电路结构简单，可以要求 ALU 的 srcA、srcB 和 srcC 都只来自几个固定的 register file。一个可能的设计图如图 16。

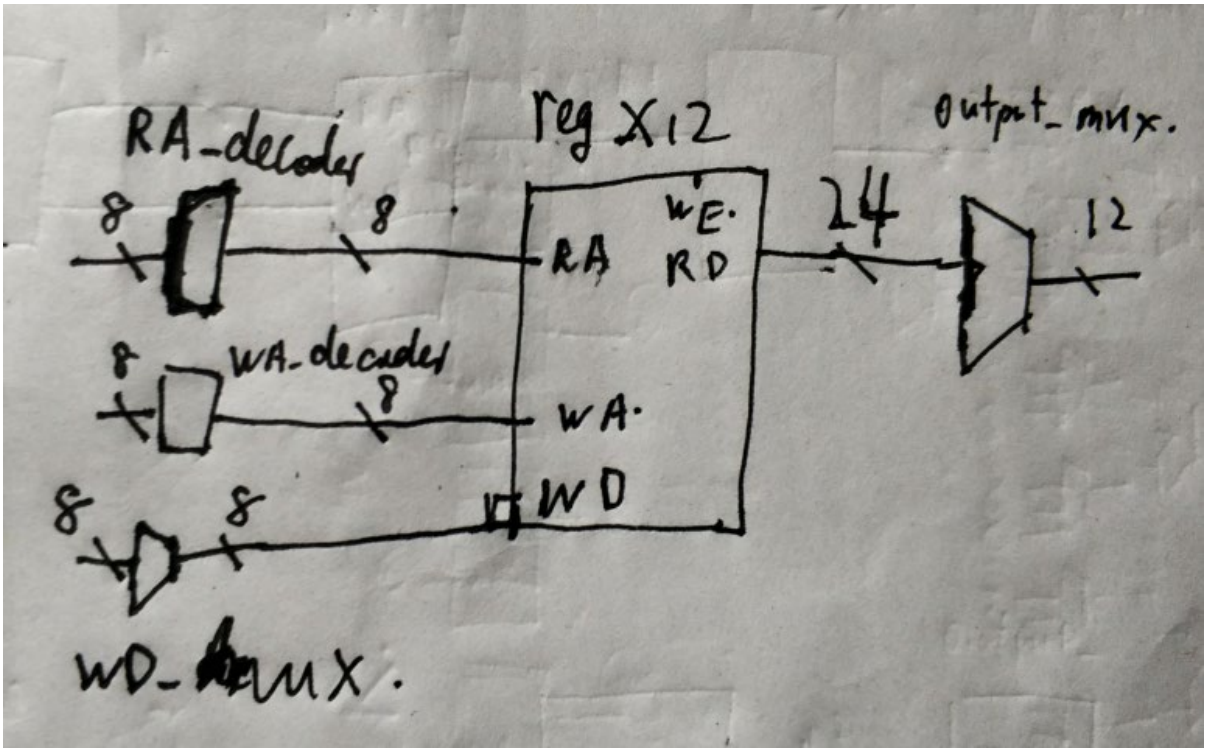

图 16 一种可能的 Register File 设计图

选取 12 个 simple dual ports ram。对于读操作，将外部寄存器编号译码成小寄存器编号，将从中读取到的数据使用一个 mux 电路选择输出到读数据端口。对于写操作，将写数据通过多路选择器选择到对应的 ram 的写数据端口，将地址译码，用地址信号和外部的写使能信号共同译码获得。不过如果允许只访问部分 register file，输出 mux 电路和输入 mux 电路可以获得极大的简化。简化结果如图 17 所示。对于编号为 A、B 的 ALU，srcA 只允许从 1、3、5 号 register file 中寻求数据，srcB 只允许从 2、4、6 中寻数据，srcC 只允许分别从 9、10 号中寻数据。对于编号为 C、D 的 ALU，srcA 只允许从 3、5、7 号 register file 中寻求数据，srcB 只允许从 4、6、8 中寻数据，srcC 只允许分别从 11、12 号中寻数据。也就是说寄存器 9~12 是独享寄存器，只允许被 1 个 ALU 访问；1、2、7、8 是半共享寄存器，前两个可以被 A、B 的 ALU 共享，后两个可以被 C、D 的 ALU 共享；3~6 为共享寄存器，可以被全局 ALU 访问。在 1~8 号 register file 中，srcA 只能访问奇数标号的，srcB 只能访问偶数标号的。srcC 只能访问与 ALU 对应的 1 个。这样，输出 mux 电路从 12 个 32bit 的 24-12 的 mux 阵列精简到 8 个 32bit 的 3-1mux 阵列。输入 mux 电路只需要 4 个 32bit 的 2-1 的 mux 和 4 个 432bit 的 4-1 的 mux 阵列。大大降低了控制的复杂度。读操作和写操作的地址信号都需要经过一个 decoder，将外部的寄存器编号与内部寄存器编号对应起来。写使能信号只有当外部写使能为 1 且选中内部 register file 之后才会为 1，否则总为 0.

需要注意的是，生成的 BRAM 的输入信号是上升延采样的，也就说输出级 mux 的控制信号需要经过一级 FF 延迟一拍才能使用。

5.2.3 布线与结果

测试激励信号手动给出，可以在设计范围内完成数据的读写。布线结果如图 18。总共使用 6 个 36Kbit 的 BRAM 和 1022 个 LUT。Register File 的 System Verilog 代码详见附录 D.

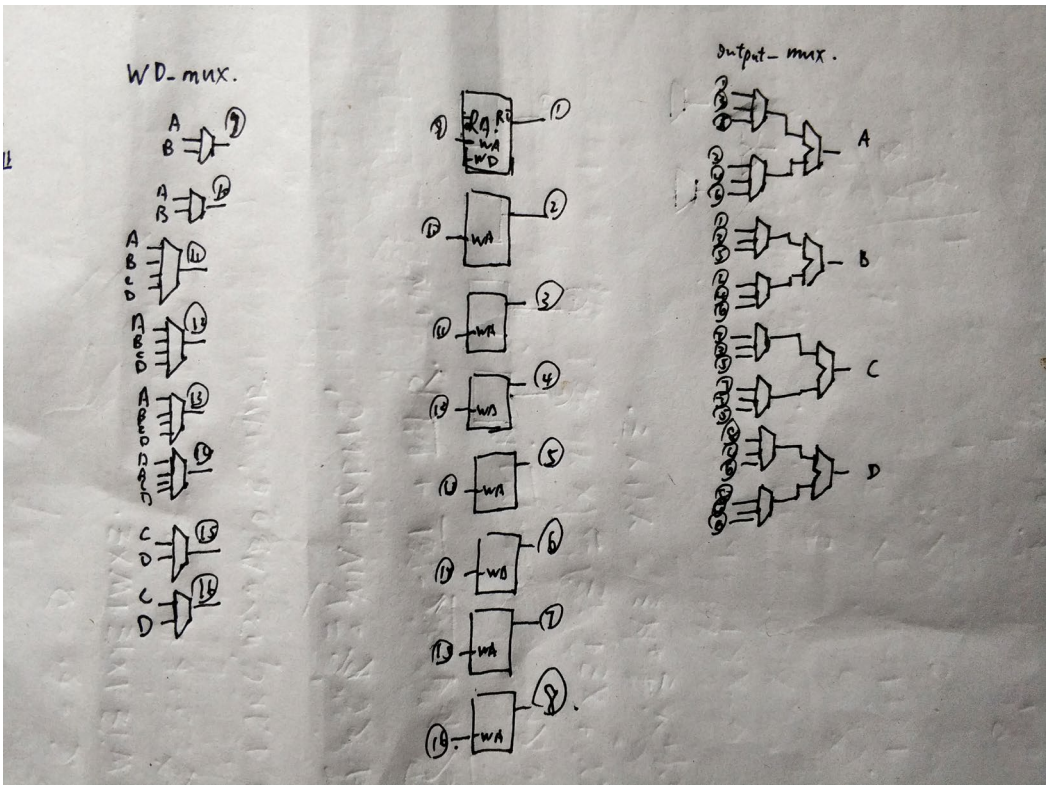

图 17 简化后的 Register File 结构

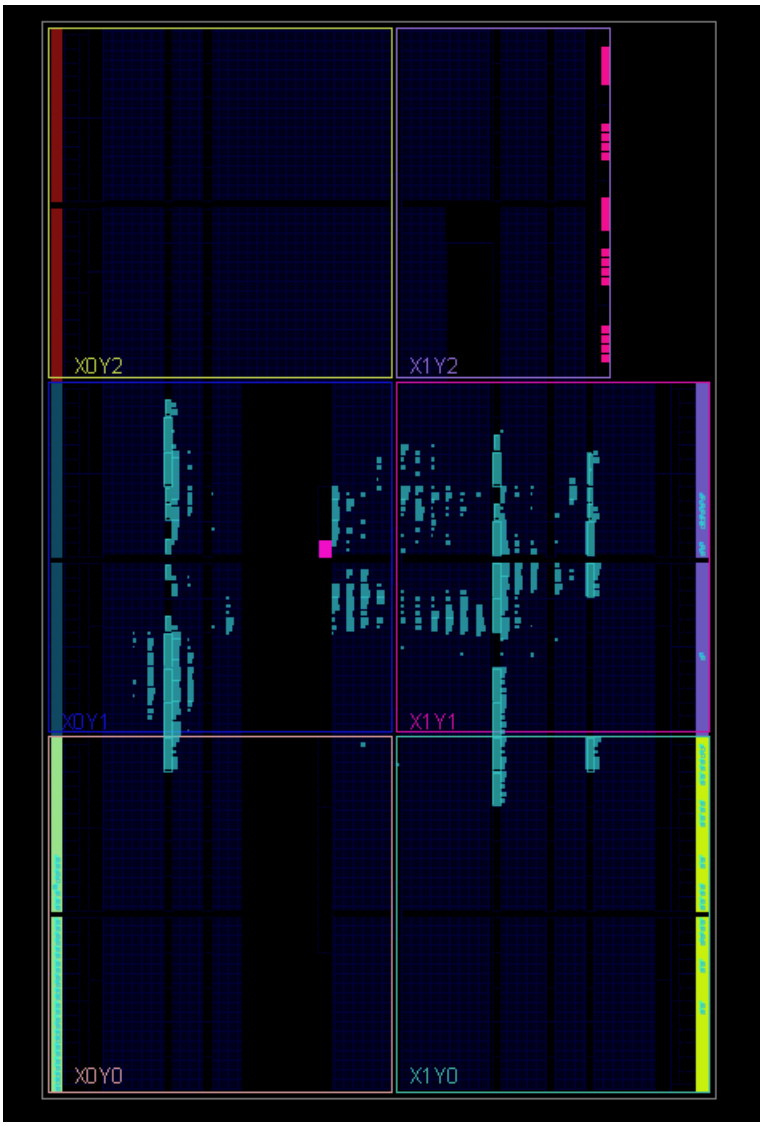

图 18 Register File 布线结果

6 参考文献

[1] M. Kumm and P. Zipf, “Pipelined compressor tree optimization using integer linear programming,” Conf. Dig. - 24th Int. Conf. F. Program. Log. Appl. FPL 2014, pp. 1–8, 2014.

[2] M. Kumm, S. Abbas, and P. Zipf, “An Efficient Softcore Multiplier Architecture for Xilinx FPGAs,” Proc. - Symp. Comput. Arith., vol. 2015–August, pp. 18–25, 2015.

[3] H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Improving synthesis of compressor trees on FPGAs via integer linear programming,” Proc. - Design, Autom. Test Eur. DATE, vol. 1, no. 1, pp. 1256–1261, 2008.

[4] S. Khan, K. Javeed, and Y. A. Shah, “High-speed FPGA implementation of full-word Montgomery multiplier for ECC applications,” Microprocess. Microsyst., vol. 62, no. June 2017, pp. 91–101, 2018.

[5] M. Kumm, “Efficient High Speed Compression Trees on Xilinx FPGAs,” no. January 2014, 2018.

[6] M. Langhammer and G. Baeckler, “High Density and Performance Multiplication for FPGA,” Proc. - Symp. Comput. Arith., vol. 2018–June, pp. 5–12, 2018.

[7] H. Parandeh-Afshar and P. Ienne, “Measuring and reducing the performance gap between embedded and soft multipliers on FPGAS,” Proc. - 21st Int. Conf. F. Program. Log. Appl. FPL 2011, pp. 225–231, 2011.

[8] K. K. Parhi, “VLSI digital signal processing systems: design and implementation,” 1999.

附录 A 冲突控制单元的 System Verilog 代码

```
module clash_control (
    input AddrSelectE1 [SUPERSCALAR_AMT-1:0],
    input [WIDTH_RF_ADDR-1:0] ArsdE1 [SUPERSCALAR_AMT-1:0], ArsdE2 [SUPERSCALAR_AMT-1:0], ArsdM [SUPERSCALAR_AMT-1:0], ArsdW
[SUPERSCALAR_AMT-1:0],
    input [WIDTH_RF_ADDR-1:0] ArtE1 [SUPERSCALAR_AMT-1:0], ArtE2 [SUPERSCALAR_AMT-1:0], ArtW [SUPERSCALAR_AMT-1:0],
    output reg replace [SUPERSCALAR_AMT-1:0],
    output reg replace_a_1 [SUPERSCALAR_AMT-1:0], replace_b_1 [SUPERSCALAR_AMT-1:0],
    output reg [1:0] mux_a_1_replace_op [SUPERSCALAR_AMT-1:0], mux_b_1_replace_op [SUPERSCALAR_AMT-1:0],
    output reg replace_a_2 [SUPERSCALAR_AMT-1:0], replace_b_2 [SUPERSCALAR_AMT-1:0],
    output reg [1:0] mux_a_2_replace_op [SUPERSCALAR_AMT-1:0], mux_b_2_replace_op [SUPERSCALAR_AMT-1:0]);

    genvar i;

    //乘法的重定向
    generate
        for(i=0;i<SUPERSCALAR_AMT;i=i+1)
        begin
            always @ (*)
                if(AddrSelectE1[i] && ArsdE1[i] == ArsdM[i])
                    replace[i] = 1;
                else
                    replace[i] = 0;
        end
    endgenerate

    //排序的重定向
    generate
        for(i=0;i<SUPERSCALAR_AMT;i=i+1)
        begin
            always @ (*)
                casex({ArsdE1[i] == ArsdW[0], ArsdE1[i] == ArsdW[1], ArsdE1[i] == ArsdW[2], ArsdE1[i] == ArsdW[3]})
                    {1'b1,1'bx,1'bx,1'bx}:
                        begin
                        replace_a_1[i] = 1;
                        mux_a_1_replace_op[i] = 2'b00;
                        end
                    {1'bx,1'b1,1'bx,1'bx}:
                        begin
                        replace_a_1[i] = 1;
                        mux_a_1_replace_op[i] = 2'b01;
                        end
                    {1'bx,1'bx,1'b1,1'bx}:
                        begin
                        replace_a_1[i] = 1;
                        mux_a_1_replace_op[i] = 2'b10;
                        end
                    {1'bx,1'bx,1'bx,1'b1}:
```

```
            begin
            replace_a_1[i] = 1;
            mux_a_1_replace_op[i] = 2'b11;
            end
        {1'b0,1'b0,1'b0,1'b0}:
            begin
            replace_a_1[i] = 0;
            mux_a_1_replace_op[i] = 2'bxx;
            end
        default:
            begin
            replace_a_1[i] = 0;
            mux_a_1_replace_op[i] = 2'bxx;
            end
    endcase

always @ (*)
    casex({ArtE1[i] == ArtW[0], ArtE1[i] == ArtW[1], ArtE1[i] == ArtW[2], ArtE1[i] == ArtW[3]})
        {1'b1,1'bx,1'bx,1'bx}:
            begin
            replace_b_1[i] = 1;
            mux_b_1_replace_op[i] = 2'b00;
            end
        {1'bx,1'b1,1'bx,1'bx}:
            begin
            replace_b_1[i] = 1;
            mux_b_1_replace_op[i] = 2'b01;
            end
        {1'bx,1'bx,1'b1,1'bx}:
            begin
            replace_b_1[i] = 1;
            mux_b_1_replace_op[i] = 2'b10;
            end
        {1'bx,1'bx,1'bx,1'b1}:
            begin
            replace_b_1[i] = 1;
            mux_b_1_replace_op[i] = 2'b11;
            end
        {1'b0,1'b0,1'b0,1'b0}:
            begin
            replace_b_1[i] = 0;
            mux_b_1_replace_op[i] = 2'bxx;
            end
        default:
            begin
            replace_b_1[i] = 0;
            mux_b_1_replace_op[i] = 2'bxx;
            end
    endcase
```

```
always @ (*)
    casex({ArsdE2[i] == ArsdW[0], ArsdE2[i] == ArsdW[1], ArsdE2[i] == ArsdW[2], ArsdE2[i] == ArsdW[3]})
        {1'b1,1'bx,1'bx,1'bx}:
            begin
            replace_a_2[i] = 1;
            mux_a_2_replace_op[i] = 2'b00;
            end
        {1'bx,1'b1,1'bx,1'bx}:
            begin
            replace_a_2[i] = 1;
            mux_a_2_replace_op[i] = 2'b01;
            end
        {1'bx,1'bx,1'b1,1'bx}:
            begin
            replace_a_2[i] = 1;
            mux_a_2_replace_op[i] = 2'b10;
            end
        {1'bx,1'bx,1'bx,1'b1}:
            begin
            replace_a_2[i] = 1;
            mux_a_2_replace_op[i] = 2'b11;
            end
        {1'b0,1'b0,1'b0,1'b0}:
            begin
            replace_a_2[i] = 0;
            mux_a_2_replace_op[i] = 2'bxx;
            end
        default:
            begin
            replace_a_2[i] = 0;
            mux_a_2_replace_op[i] = 2'bxx;
            end
    endcase

always @ (*)
    casex({ArtE2[i] == ArtW[0], ArtE2[i] == ArtW[1], ArtE2[i] == ArtW[2], ArtE2[i] == ArtW[3]})
        {1'b1,1'bx,1'bx,1'bx}:
            begin
            replace_b_2[i] = 1;
            mux_b_2_replace_op[i] = 2'b00;
            end
        {1'bx,1'b1,1'bx,1'bx}:
            begin
            replace_b_2[i] = 1;
            mux_b_2_replace_op[i] = 2'b01;
            end
        {1'bx,1'bx,1'b1,1'bx}:
            begin
```

```
                    replace_b_2[i] = 1;
                    mux_b_2_replace_op[i] = 2'b10;
                    end
                {1'bx,1'bx,1'bx,1'b1}:
                    begin
                    replace_b_2[i] = 1;
                    mux_b_2_replace_op[i] = 2'b11;
                    end
                {1'b0,1'b0,1'b0,1'b0}:
                    begin
                    replace_b_2[i] = 0;
                    mux_b_2_replace_op[i] = 2'bxx;
                    end
                default:
                    begin
                    replace_b_2[i] = 0;
                    mux_b_2_replace_op[i] = 2'bxx;
                    end
            endcase
        end
    endgenerate

endmodule
```

附录 B 整数规划程序的 Lingo 代码

```
{\colortbl ;\red0\green0\blue255;\red0\green0\blue0;\red0\green175\blue0;}
\viewkind4\uc1\pard\cf1\lang2052\f0\fs20 model\cf2 :\par
\cf3 ! cost_per_comp\par
   1\f1\tab\lang1033\f0 (          1:1)\lang2052\par
   2\tab (    \tab    3:2)\par
   3\tab (    \tab    6:3)\par
   4\tab (        1,5:3)\par
   5\tab (        2,3:3)\par
   6\f1          \lang1033\f0    (    6,0,6:5)\lang2052\par
   7\tab (1,3,2,5:5)\par
;\cf2\tab\par
\par
      \cf1 sets\cf2 :\par
\tab state/1..5/:Ds;\cf3 !\lang1033 output sign\lang2052 ;\cf2\par
\tab column/1..37/:N0;\par
\tab comp_type/1..5/:cost_per_comp;\par
\tab comp_change/1..5/;\par
\tab\par
\tab link_N\tab (state,column):N;\par
\tab link_change\tab (comp_type,comp_change):eliminate,gen;\par
\tab link_num\tab\lang1033 (\lang2052 state,comp_type,column\lang1033 ):num;\lang2052\par
      \cf1 endsets\cf2\par
\par
      \cf1    min\cf2        =    \cf1    @sum\cf2    (link_num(i,j,k):cost_per_comp(j)*num(i,j,k))    +    \cf1    @sum\cf2
(state(i):i*Large*Ds(i));\par
\par
      \cf1 data\cf2 :\par
\tab Large = 1000;\par
\tab Inf = 100\lang1033 00\lang2052 00;\par
\tab cost_per_comp = 0.0001,1,3,2,2;\cf3 !,4,4;\cf2\par
\tab eliminate = 1 0 0 0 0\par
\tab\tab\tab 3 0 0 0 0\par
\tab\tab\tab 6 0 0 0 0\par
\tab\tab\tab 5 1 0 0 0\par
\tab\tab\tab 3 2 0 0 0\par
\tab\tab\tab ;\cf3 !6 0 6 0 0 \par
\tab\tab\tab 5 2 3 1 0;\cf2\par
\tab\tab\tab\par
\tab gen = \par
\tab\tab 1 0 0 0 0\par
\tab\tab 1 1 0 0 0\par
\tab\tab 1 1 1 0 0\par
  \tab\tab 1 1 1 0 0 \par
\tab\tab 1 1 1 0 0\par
\tab\tab ;\cf3 !1 1 1 1 1\par
\tab\tab 1 1 1 1 1;\cf2\par
\pard\nowidctlpar\qj\lang1033\kerning2\f2\fs21\par
\par
```

```
\tab N0 =\tab 1\tab 2\tab 2\tab 3\tab 3\tab 4\tab 4\tab 5\par
\tab\tab 5\tab 6\tab 6\tab 7\tab 7\tab 8\tab 9\tab 8\par
\tab\tab 8\tab 8\tab 8\tab 8\tab 6\tab 6\tab 5\tab 5\par
\tab\tab 4\tab 4\tab 3\tab 3\tab 2\tab 2\tab 1\tab 1\par
\tab\tab 0\tab 0\tab 0\tab 0\tab 0;\par
\par
\par
\par
\par
\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\tab\par
\pard\lang2052\kerning0\f0\fs20\par
\tab\cf1 enddata\cf2\par
\par
      \cf1 @for\cf2 (link_N:\cf1 @gin\cf2 (N));\par
      \cf1 @for\cf2 (link_num:\cf1 @gin\cf2 (num));\par
      \cf1 @for\cf2 (state:\cf1 @bin\cf2 (Ds));\par
\par
      \cf1 @for\cf2 (state(i)|i#gt\lang1033 #1: \cf1 @for\cf2 (\lang2052 column(j)\lang1033 : \lang2052\par
\tab N(i,j) = \cf1 @sum\cf2 (link_change(k,offset)|offset#lt#j+1:gen(k,offset)*num(i-1,k,j+1-offset))\par
\tab    ));\par
\par
      \cf1 @for\cf2 (state(i)|i#gt\lang1033 #1: \cf1 @for\cf2 (\lang2052 column(j):\par
\tab N(i-1,j) <= \cf1 @sum\cf2 (link_change(k,offset)|offset#lt#j+1:eliminate(k,offset)*num(i-1,k,j+1-offset))\par
\tab    ));\par
\par
      \cf1 @for\cf2 (state(i)\lang1033 : \cf1 @for\cf2 (\lang2052 column(j):\par
\tab N(i,j) <= 3 + 100\f1 *\lang1033\f0 (1-Ds(i))\lang2052 ));\par
      \cf1 @sum\cf2 (state:Ds) = 1;        \par
\par
\lang1033         \cf1 @for\cf2 (column(j):N(1,j)=N0(j));\lang2052\par
\par
\par
\par
\par
```

附录 C ALU 的 Verilog 代码

```
module ALU (
        input clk,
        input rst_n,
        input [31:0] a_in,
        input [31:0] b_in,
        input [31:0] c_in,
        input [31:0] a_r_2_in,
        input [31:0] b_r_2_in,
        input [31:0] c_r_in,
        input  replace_a_2,
        input  replace_b_2,
        input  replace_c,
        output [1:0] a_gt_b_o,
        output [31:0] d_o
      );
parameter OP_MUL = 00;
parameter OP_CMP = 11;
parameter OP_SHF = 01;

// multiply
mul_top mul_top_u(
         .clk(clk),
         .rst_n(rst_n),
         .a_in(a_in),
         .b_in(b_in),
         .c_in(c_in),
         .c_re_in(c_r_in),
         .replace({replace_a_2,replace_b_2,replace_c}),
         .o(d_o)
       );

// compare
compare_top compare_top_u(
    .clk                ( clk             ),
    .rst_n              ( rst_n           ),
    .a_in               ( a_in      [31:0] ),
    .b_in               ( b_in      [31:0] ),
    .a_re_in            ( a_r_2_in    [31:0] ),
    .b_re_in            ( b_r_2_in    [31:0] ),
    .replace_a          ( replace_a_2       ),
    .replace_b          ( replace_b_2       ),
    .agtb_h_o           ( a_gt_b_o[1]        ),
    .agtb_l_o           ( a_gt_b_o[0]        )
       );

// shift

endmodule
```

```
module compare(
    input [15:0] a_in,
    input [15:0]  b_in,
    output wire o
    );
    genvar i;
    wire [7:0] gt;
    wire [7:0] eq;
    wire [3:0]carry_1;
    wire [3:0]c;
    for (i=0; i<16;i=i+2) begin
        GT GT_u(a_in[i+1:i],b_in[i+1:i],gt[i/2],eq[i/2]); end
    CARRY4 CARRY4_u1 (
      .CO(carry_1), // 4-bit carry out
      .O(), // 4-bit carry chain XOR data out
      .CI(0), // 1-bit carry cascade input
      .CYINIT(1), // 1-bit carry initialization
      .DI(gt[3:0]), // 4-bit carry-MUX data in
      .S(eq[3:0]) // 4-bit carry-MUX select input
    );
    CARRY4 CARRY4_u2 (
      .CO(c), // 4-bit carry out
      .O(), // 4-bit carry chain XOR data out
      .CI(carry_1[3]), // 1-bit carry cascade input
      .CYINIT(1), // 1-bit carry initialization
      .DI(gt[7:4]), // 4-bit carry-MUX data in
      .S(eq[7:4]) // 4-bit carry-MUX select input
    );
    assign o = c[3];
endmodule

module GT(
    input[1:0] a,
    input[1:0] b,
    output wire o_gt,
    output wire o_eq
    );
    assign o_gt =( a[0] & !b[0] & !a[1]) | (!a[1] & b[1]) |
(a[0] & !b[0] & b[1]);
    assign o_eq = (!a[0] & !b[0] & !a[1] & !b[1]) | (a[0] &
b[0] & !a[1] & !b[1])|( !a[0] & !b[0] & a[1] & b[1]) |( a[0]
& b[0] & a[1] & b[1]);
endmodule

module mul_comp_lcx(
        input clk,
        input rst_n,
```

```
        input [15:0]a,
        input [15:0]b,
        output [31:0]o_33
      );
////wire [19:0]l1, [19:1]l2, [21:3]l3, [23:5]l4, [25:7]l5,
[27:9]l6, [29:11]l7, [31:13]l8, [31:14]l9;
wire [31:0]prod_l[1:9];//部分积
wire [9:1]s1[0:31];
wire [9:1]s1_temp[0:31];
wire [7:1]s2[0:31];//部分积
wire [6:1]s3[0:32];//部分积
wire [3:1]s4[0:32];//部分积
wire [31:0]d;
wire [31:0]s;
wire [31:0]o;
wire [31:0]carryo;
//wire [32:0]o_33;
assign o_33 = o;
reg  [6:1]s3_r[0:32];


genvar i,j;

// * 生成
assign
{prod_l[3][21],prod_l[4][23],prod_l[5][25],prod_l[6][27],pro
d_l[7][29],prod_l[8][31]} = 6'b11_1111; // 最高位置 1
assign prod_l[1][18:17] = {prod_l[1][16],prod_l[1][16]}; //
生成第一行的 SS
//assign prod_l[9][31:15] = 0;

//* 调转
generate
  for (i = 1; i <= 9; i = i+1)
  begin
    for (j = 0; j <= 31; j = j+1)
    begin
      assign s1_temp[j][i] = prod_l[i][j];
    end
  end

  for (j = 0; j <= 19; j = j+1)
  begin
    assign s1[j][9:1] = s1_temp[j][9:1];
  end
  assign s1[20][7:1] = s1_temp[20][9:3];
  assign s1[21][7:1] = s1_temp[21][9:3];
  assign s1[22][6:1] = s1_temp[22][9:4];
  assign s1[23][6:1] = s1_temp[23][9:4];
  assign s1[24][5:1] = s1_temp[24][9:5];
  assign s1[25][5:1] = s1_temp[25][9:5];
  assign s1[26][4:1] = s1_temp[26][9:6];
  assign s1[27][4:1] = s1_temp[27][9:6];
  assign s1[28][3:1] = s1_temp[28][9:7];
  assign s1[29][3:1] = s1_temp[29][9:7];
  assign s1[30][2:1] = s1_temp[30][9:8];
  assign s1[31][2:1] = s1_temp[31][9:8];
endgenerate

// * 部分积
generate
  // * 生成 Abox
  for (j = 1; j<9; j = j+1)
    begin
      for (i = 1; i < 16; i = i+1)
        begin
          A_box A_box_u_prod_l(
                 .b_i((j==1) ? {b[1:0],1'b0} : b[2*j-1:2*j-
3]),
                 .a_i(a[i:i-1]),
                 .o(prod_l[j][i + (j-1)*2]) // 第 8 行从 15 开始
               );
        end
      if (j < 8)
        begin
          BC_box BC_box_u_prod_l_md(
                  .b_i((j==1) ? {b[1:0],1'b0} : b[2*j-1:2*j-
3]),
                  .a_i(a[0]),
                  .b_box_o(prod_l[j][(j-1)*2]),
                  .c_box_o(prod_l[j+1][j*2-1])
                );
        end
      else
        begin
          DE_box DE_box_u_prod_l_end(
                  .b_i(b[2*j-1:2*j-3]),
                  .a_i(a[0]),
                  .d_box_o(prod_l[8][14]),
                  .e_box_o(prod_l[9][14])
                );
        end
      if (j<=2)
        begin
          SnS_box SnS_box_u_prod_l(
                   .b_i((j==1) ? {b[1:0],1'b0} : b[2*j-1:2*j-
3]),
                   .a_i(a[15]),
```

```
                .s_box_o(prod_l[j][(j==1) ? 16 : 18]),
                .ns_box_o(prod_l[j][19])
              );
      end
    else
      begin
        SnS_box SnS_box_u_prod_l(
                .b_i(b[2*j-1:2*j-3]),
                .a_i(a[15]),
                .ns_box_o(prod_l[j][j*2+14])
              );  assign s2[31][1] = s1[31][1];
comp32 comp32_u_s1_0(
  .a_i(s1[5][4:2]),
  .o({s2[5][2],s2[6][2]})
);
comp32 comp32_u_s1_1(
  .a_i(s1[6][4:2]),
  .o({s2[6][3],s2[7][3]})
);
comp32 comp32_u_s1_2(
  .a_i(s1[7][5:3]),
  .o({s2[7][4],s2[8][3]})
);
comp32 comp32_u_s1_3(
  .a_i(s1[8][5:3]),
  .o({s2[8][4],s2[9][1]})
);
comp32 comp32_u_s1_4(
  .a_i(s1[10][3:1]),
  .o({s2[10][1],s2[11][2]})
);
comp32 comp32_u_s1_5(
  .a_i(s1[10][6:4]),
  .o({s2[10][2],s2[11][3]})
);
comp32 comp32_u_s1_6(
  .a_i(s1[12][4:2]),
  .o({s2[12][2],s2[13][3]})
);
comp32 comp32_u_s1_7(
  .a_i(s1[12][7:5]),
  .o({s2[12][3],s2[13][4]})
);
comp32 comp32_u_s1_8(
  .a_i(s1[18][5:3]),
  .o({s2[18][3],s2[19][1]})
);
comp32 comp32_u_s1_9(
  .a_i(s1[21][3:1]),
  .o({s2[21][1],s2[22][3]})
);
comp32 comp32_u_s1_10(
  .a_i(s1[21][6:4]),
  .o({s2[21][2],s2[22][4]})
);
comp32 comp32_u_s1_11(
  .a_i(s1[22][5:3]),
  .o({s2[22][5],s2[23][6]})
);

      end
    end
endgenerate

// * 压缩
generate
  assign s2[0][1] = s1[0][1];
  assign s2[1][1] = s1[1][1];
  assign s2[1][2] = s1[1][2];
  assign s2[2][1] = s1[2][1];
  assign s2[2][2] = s1[2][2];
  assign s2[3][1] = s1[3][1];
  assign s2[3][2] = s1[3][2];
  assign s2[3][3] = s1[3][3];
  assign s2[4][1] = s1[4][1];
  assign s2[4][2] = s1[4][2];
  assign s2[4][3] = s1[4][3];
  assign s2[5][1] = s1[5][1];
  assign s2[6][1] = s1[6][1];
  assign s2[7][1] = s1[7][1];
  assign s2[7][2] = s1[7][2];
  assign s2[8][1] = s1[8][1];
  assign s2[8][2] = s1[8][2];
  assign s2[11][1] = s1[11][1];
  assign s2[12][1] = s1[12][1];
  assign s2[13][1] = s1[13][1];
  assign s2[13][2] = s1[13][2];
  assign s2[16][1] = s1[16][1];
  assign s2[16][2] = s1[16][2];
  assign s2[17][1] = s1[17][1];
  assign s2[17][2] = s1[17][2];
  assign s2[18][1] = s1[18][1];
  assign s2[18][2] = s1[18][2];
  assign s2[22][1] = s1[22][1];
  assign s2[22][2] = s1[22][2];
  assign s2[23][1] = s1[23][1];
  assign s2[23][2] = s1[23][2];
  assign s2[23][3] = s1[23][3];
```

```
assign s2[23][4] = s1[23][4];
assign s2[23][5] = s1[23][5];
assign s2[24][1] = s1[24][1];
assign s2[25][1] = s1[25][1];
assign s2[28][1] = s1[28][1];
assign s2[28][2] = s1[28][2];
assign s2[29][1] = s1[29][1];
assign s2[29][2] = s1[29][2];
assign s2[30][1] = s1[30][1];
assign s2[31][1] = s1[31][1];
comp32 comp32_u_s1_0(
  .a_i(s1[5][4:2]),
  .o({s2[5][2],s2[6][2]})
);
comp32 comp32_u_s1_1(
  .a_i(s1[6][4:2]),
  .o({s2[6][3],s2[7][3]})
);
comp32 comp32_u_s1_2(
  .a_i(s1[7][5:3]),
  .o({s2[7][4],s2[8][3]})
);
comp32 comp32_u_s1_3(
  .a_i(s1[8][5:3]),
  .o({s2[8][4],s2[9][1]})
);
comp32 comp32_u_s1_4(
  .a_i(s1[10][3:1]),
  .o({s2[10][1],s2[11][2]})
);
comp32 comp32_u_s1_5(
  .a_i(s1[10][6:4]),
  .o({s2[10][2],s2[11][3]})
);
comp32 comp32_u_s1_6(
  .a_i(s1[12][4:2]),
  .o({s2[12][2],s2[13][3]})
);
comp32 comp32_u_s1_7(
  .a_i(s1[12][7:5]),
  .o({s2[12][3],s2[13][4]})
);
comp32 comp32_u_s1_8(
  .a_i(s1[18][5:3]),
  .o({s2[18][3],s2[19][1]})
);
comp32 comp32_u_s1_9(
  .a_i(s1[21][3:1]),
  .o({s2[21][1],s2[22][3]})
);
comp32 comp32_u_s1_10(
  .a_i(s1[21][6:4]),
  .o({s2[21][2],s2[22][4]})
);
comp32 comp32_u_s1_11(
  .a_i(s1[22][5:3]),
  .o({s2[22][5],s2[23][6]})
);
comp32 comp32_u_s1_12(
  .a_i(s1[24][4:2]),
  .o({s2[24][2],s2[25][2]})
);
comp32 comp32_u_s1_13(
  .a_i(s1[25][4:2]),
  .o({s2[25][3],s2[26][1]})
);
comp32 comp32_u_s1_14(
  .a_i(s1[26][3:1]),
  .o({s2[26][2],s2[27][1]})
);
comp32 comp32_u_s1_15(
  .a_i(s1[27][3:1]),
  .o({s2[27][2],s2[28][3]})
);
comp63 comp63_u_s1_0(
  .a_i(s1[9][6:1]),
  .o({s2[9][2],s2[10][3],s2[11][4]})
);
comp63 comp63_u_s1_1(
  .a_i(s1[11][7:2]),
  .o({s2[11][5],s2[12][4],s2[13][5]})
);
comp63 comp63_u_s1_2(
  .a_i(s1[13][8:3]),
  .o({s2[13][6],s2[14][1],s2[15][1]})
);
comp63 comp63_u_s1_3(
  .a_i(s1[14][6:1]),
  .o({s2[14][2],s2[15][2],s2[16][3]})
);
comp63 comp63_u_s1_4(
  .a_i(s1[15][6:1]),
  .o({s2[15][3],s2[16][4],s2[17][3]})
);
comp63 comp63_u_s1_5(
  .a_i(s1[16][8:3]),
  .o({s2[16][5],s2[17][4],s2[18][4]})
);
```

```
comp63 comp63_u_s1_6(
  .a_i(s1[17][8:3]),
  .o({s2[17][5],s2[18][5],s2[19][2]})
);
comp63 comp63_u_s1_7(
  .a_i(s1[19][6:1]),
  .o({s2[19][3],s2[20][1],s2[21][3]})
);
comp63 comp63_u_s1_8(
  .a_i(s1[20][6:1]),
  .o({s2[20][2],s2[21][4],s2[22][6]})
);
comp233 comp233_u_s1_0(
  .a0_i(s1[14][9:7]),
  .a1_i({s1[15][8:7]}),
  .o({s2[14][3],s2[15][4],s2[16][6]})
);
comp233 comp233_u_s1_1(
  .a0_i(s1[18][8:6]),
  .a1_i({s1[19][8:7]}),
  .o({s2[18][6],s2[19][4],s2[20][3]})
);
assign s3[0][1] = s2[0][1];
assign s3[1][1] = s2[1][1];
assign s3[1][2] = s2[1][2];
assign s3[2][1] = s2[2][1];
assign s3[2][2] = s2[2][2];
assign s3[3][1] = s2[3][1];
assign s3[3][2] = s2[3][2];
assign s3[3][3] = s2[3][3];
assign s3[4][1] = s2[4][1];
assign s3[4][2] = s2[4][2];
assign s3[4][3] = s2[4][3];
assign s3[5][1] = s2[5][1];
assign s3[5][2] = s2[5][2];
assign s3[6][1] = s2[6][1];
assign s3[6][2] = s2[6][2];
assign s3[6][3] = s2[6][3];
assign s3[7][1] = s2[7][1];
assign s3[8][1] = s2[8][1];
assign s3[9][1] = s2[9][1];
assign s3[9][2] = s2[9][2];
assign s3[10][1] = s2[10][1];
assign s3[10][2] = s2[10][2];
assign s3[10][3] = s2[10][3];
assign s3[11][1] = s2[11][1];
assign s3[11][2] = s2[11][2];
assign s3[12][1] = s2[12][1];
assign s3[15][1] = s2[15][1];
assign s3[19][1] = s2[19][1];
assign s3[21][1] = s2[21][1];
assign s3[24][1] = s2[24][1];
assign s3[24][2] = s2[24][2];
assign s3[26][1] = s2[26][1];
assign s3[26][2] = s2[26][2];
assign s3[27][1] = s2[27][1];
assign s3[27][2] = s2[27][2];
assign s3[28][1] = s2[28][1];
assign s3[28][2] = s2[28][2];
assign s3[28][3] = s2[28][3];
assign s3[29][1] = s2[29][1];
assign s3[29][2] = s2[29][2];
assign s3[30][1] = s2[30][1];
assign s3[31][1] = s2[31][1];
comp32 comp32_u_s2_0(
  .a_i(s2[7][4:2]),
  .o({s3[7][2],s3[8][2]})
);
comp32 comp32_u_s2_1(
  .a_i(s2[8][4:2]),
  .o({s3[8][3],s3[9][3]})
);
comp32 comp32_u_s2_2(
  .a_i(s2[11][5:3]),
  .o({s3[11][3],s3[12][2]})
);
comp32 comp32_u_s2_3(
  .a_i(s2[12][4:2]),
  .o({s3[12][3],s3[13][1]})
);
comp32 comp32_u_s2_4(
  .a_i(s2[13][3:1]),
  .o({s3[13][2],s3[14][1]})
);
comp32 comp32_u_s2_5(
  .a_i(s2[13][6:4]),
  .o({s3[13][3],s3[14][2]})
);
comp32 comp32_u_s2_6(
  .a_i(s2[14][3:1]),
  .o({s3[14][3],s3[15][2]})
);
comp32 comp32_u_s2_7(
  .a_i(s2[15][4:2]),
  .o({s3[15][3],s3[16][1]})
);
comp32 comp32_u_s2_8(
  .a_i(s2[16][3:1]),
```

```
    .o({s3[16][2],s3[17][1]})
  );
  comp32 comp32_u_s2_9(
    .a_i(s2[17][3:1]),
    .o({s3[17][2],s3[18][1]})
  );
  comp32 comp32_u_s2_10(
    .a_i(s2[19][4:2]),
    .o({s3[19][2],s3[20][1]})
  );
  comp32 comp32_u_s2_11(
    .a_i(s2[20][3:1]),
    .o({s3[20][2],s3[21][2]})
  );
  comp32 comp32_u_s2_12(
    .a_i(s2[21][4:2]),
    .o({s3[21][3],s3[22][1]})
  );
  comp32 comp32_u_s2_13(
    .a_i(s2[22][3:1]),
    .o({s3[22][2],s3[23][1]})
  );
  comp32 comp32_u_s2_14(
    .a_i(s2[22][6:4]),
    .o({s3[22][3],s3[23][2]})
  );
  comp32 comp32_u_s2_15(
    .a_i(s2[25][3:1]),
    .o({s3[25][1],s3[26][3]})
  );
  comp63 comp63_u_s2_0(
    .a_i(s2[18][6:1]),
    .o({s3[18][2],s3[19][3],s3[20][3]})
  );
  comp63 comp63_u_s2_1(
    .a_i(s2[23][6:1]),
    .o({s3[23][3],s3[24][3],s3[25][2]})
  );
  comp233 comp233_u_s2_0(
    .a0_i(s2[16][6:4]),
    .a1_i({s2[17][5:4]}),
    .o({s3[16][3],s3[17][3],s3[18][3]})
  );

// ! FF state3 -> state4
  assign s4[0][1] = s3_r[0][1];
  assign s4[0][2] = 0;
  assign s4[0][3] = 0;
  assign s4[1][1] = s3_r[1][1];
  assign s4[1][2] = s3_r[1][2];
  assign s4[1][3] = 0;
  assign s4[2][1] = s3_r[2][1];
  assign s4[2][2] = s3_r[2][2];
  assign s4[2][3] = 0;
  assign s4[3][1] = s3_r[3][1];
  assign s4[3][2] = s3_r[3][2];
  assign s4[3][3] = s3_r[3][3];
  assign s4[4][1] = s3_r[4][1];
  assign s4[4][2] = s3_r[4][2];
  assign s4[4][3] = s3_r[4][3];
  assign s4[5][1] = s3_r[5][1];
  assign s4[5][2] = s3_r[5][2];
  assign s4[5][3] = 0;
  assign s4[6][1] = s3_r[6][1];
  assign s4[6][2] = s3_r[6][2];
  assign s4[6][3] = s3_r[6][3];
  assign s4[7][1] = s3_r[7][1];
  assign s4[7][2] = s3_r[7][2];
  assign s4[7][3] = 0;
  assign s4[8][1] = s3_r[8][1];
  assign s4[8][2] = s3_r[8][2];
  assign s4[8][3] = s3_r[8][3];
  assign s4[9][1] = s3_r[9][1];
  assign s4[9][2] = s3_r[9][2];
  assign s4[9][3] = s3_r[9][3];
  assign s4[10][1] = s3_r[10][1];
  assign s4[10][2] = s3_r[10][2];
  assign s4[10][3] = s3_r[10][3];
  assign s4[11][1] = s3_r[11][1];
  assign s4[11][2] = s3_r[11][2];
  assign s4[11][3] = s3_r[11][3];
  assign s4[12][1] = s3_r[12][1];
  assign s4[12][2] = s3_r[12][2];
  assign s4[12][3] = s3_r[12][3];
  assign s4[13][1] = s3_r[13][1];
  assign s4[13][2] = s3_r[13][2];
  assign s4[13][3] = s3_r[13][3];
  assign s4[14][1] = s3_r[14][1];
  assign s4[14][2] = s3_r[14][2];
  assign s4[14][3] = s3_r[14][3];
  assign s4[15][1] = s3_r[15][1];
  assign s4[15][2] = s3_r[15][2];
  assign s4[15][3] = s3_r[15][3];
  assign s4[16][1] = s3_r[16][1];
  assign s4[16][2] = s3_r[16][2];
  assign s4[16][3] = s3_r[16][3];
  assign s4[17][1] = s3_r[17][1];
  assign s4[17][2] = s3_r[17][2];
```

```
  assign s4[17][3] = s3_r[17][3];
  assign s4[18][1] = s3_r[18][1];
  assign s4[18][2] = s3_r[18][2];
  assign s4[18][3] = s3_r[18][3];
  assign s4[19][1] = s3_r[19][1];
  assign s4[19][2] = s3_r[19][2];
  assign s4[19][3] = s3_r[19][3];
  assign s4[20][1] = s3_r[20][1];
  assign s4[20][2] = s3_r[20][2];
  assign s4[20][3] = s3_r[20][3];
  assign s4[21][1] = s3_r[21][1];
  assign s4[21][2] = s3_r[21][2];
  assign s4[21][3] = s3_r[21][3];
  assign s4[22][1] = s3_r[22][1];
  assign s4[22][2] = s3_r[22][2];
  assign s4[22][3] = s3_r[22][3];
  assign s4[23][1] = s3_r[23][1];
  assign s4[23][2] = s3_r[23][2];
  assign s4[23][3] = s3_r[23][3];
  assign s4[24][1] = s3_r[24][1];
  assign s4[24][2] = s3_r[24][2];
  assign s4[24][3] = s3_r[24][3];
  assign s4[25][1] = s3_r[25][1];
  assign s4[25][2] = s3_r[25][2];
  assign s4[25][3] = 0;
  assign s4[26][1] = s3_r[26][1];
  assign s4[26][2] = s3_r[26][2];
  assign s4[26][3] = s3_r[26][3];
  assign s4[27][1] = s3_r[27][1];
  assign s4[27][2] = s3_r[27][2];
  assign s4[27][3] = 0;
  assign s4[28][1] = s3_r[28][1];
  assign s4[28][2] = s3_r[28][2];
  assign s4[28][3] = s3_r[28][3];
  assign s4[29][1] = s3_r[29][1];
  assign s4[29][2] = s3_r[29][2];
  assign s4[29][3] = 0;
  assign s4[30][1] = s3_r[30][1];
  assign s4[30][2] = 0;
  assign s4[30][3] = 0;
  assign s4[31][1] = s3_r[31][1];
  assign s4[31][2] = 0;
  assign s4[31][3] = 0;
endgenerate

// * FF
generate
for(i=0; i<33; i=i+1)
begin
  always @(posedge clk or negedge rst_n)
    begin
      if (!rst_n)
        begin
            s3_r[i] = 0;
          end
      else
        begin
             s3_r[i] = s3[i];
        end
    end
end


endgenerate

// * 合并

for(i=0;i<32;i=i+1)
begin
assign d[i] = ^{s4[i][1],s4[i][2],s4[i][3]};
assign s[i] = i==0 ? d[i] : d[i]^((s4[i-1][1]&s4[i-
1][2])|(s4[i-1][3]&(s4[i-1][1]|s4[i-1][2])));
end
for(i=1; i<9; i = i+1)
  begin
    CARRY4 CARRY4_inst (
            .CO(carryo[i*4-1:i*4-4]), // 4-bit carry out
            .O(o[i*4-1:i*4-4]), // 4-bit carry chain XOR data
out
            .CI((i==1) ? 0 : carryo[(i-1)*4-1]), // 1-bit
carry cascade input
            .CYINIT(0), // 1-bit carry initialization
            .DI(d[i*4-1:i*4-4]), // 4-bit carry-MUX data in
            .S(s[i*4-1:i*4-4]) // 4-bit carry-MUX select
input
          );
  end

//assign o = o_33[31:15];
//assign o_33[0] = s4[0][1];
endmodule// mul_comp



module A_box(
         input [2:0]b_i,
```

```
        input [1:0]a_i,
        output wire o
       );
  LUT5 #(
       .INIT(32'h0E16_6870) // Specify LUT Contents
     ) LUT5_A_box (
       .O(o), // LUT general output
       .I0(b_i[0]), // LUT input
       .I1(b_i[1]), // LUT input
       .I2(b_i[2]), // LUT input
       .I3(a_i[0]), // LUT input
       .I4(a_i[1]) // LUT input
     );
endmodule // Abox

module BC_box(
        input [2:0]b_i,
        input a_i,
        output wire b_box_o,
        output wire c_box_o
      );
  LUT6_2 #(
          .INIT(64'h0000_1070_0000_6600)
        // .INIT(64'h0066_0000_0E08_0000) // Specify LUT
Contents
       ) LUT6_2_BC_box (
         .O6(c_box_o), // 1-bit Cbox @ LUT6 output
         .O5(b_box_o), // 1-bit Bbox @ LUT5 output
         .I0(b_i[0]), // LUT input
         .I1(b_i[1]), // LUT input
         .I2(b_i[2]), // LUT input
         .I3(a_i), // LUT input
         .I4(0), // LUT input
         .I5(1) // LUT input control mux
       );
endmodule //BC_box

module DE_box(
        input [2:0]b_i,
        input a_i,
        output wire d_box_o,
        output wire e_box_o
      );
  LUT6_2 #(
           .INIT(64'h0000_7070_0000_1670) // Specify LUT
Contents
         // .INIT(64'h0E68_0000_0E0E_0000) // Specify LUT
Contents
       ) LUT6_2_DE_box (
         .O6(e_box_o), // 1-bit Ebox @ LUT6 output
         .O5(d_box_o), // 1-bit Dbox @ LUT5 output
         .I0(b_i[0]), // LUT input
         .I1(b_i[1]), // LUT input
         .I2(b_i[2]), // LUT input
         .I3(a_i), // LUT input
         .I4(0), // LUT input
         .I5(1) // LUT input control mux
       );
endmodule //DE_box

module SnS_box(
        input [2:0]b_i,
        input a_i,
        output wire s_box_o,
        output wire ns_box_o
       );
  LUT6_2 #(
           .INIT('h0000_F18F_0000_0E70) // Specify LUT
Contents
         // .INIT('h0E70_0000_F18F_0000) // Specify LUT
Contents
       ) LUT6_2_SnS_box (
         .O6(ns_box_o), // 1-bit nSbox @ LUT6 output
         .O5(s_box_o), // 1-bit Sbox @ LUT5 output
         .I0(b_i[0]), // LUT input
         .I1(b_i[1]), // LUT input
         .I2(b_i[2]), // LUT input
         .I3(a_i), // LUT input
         .I4(0), // LUT input
         .I5(1) // LUT input control mux
       );
endmodule //  SnS_box


module comp32(
        input [2:0]a_i,
        output wire [1:0]o
       );
  LUT6_2 #(
           .INIT('h0000_00E8_0000_0096) // Specify LUT
Contents
         // .INIT('h6900_0000_1700_0000) // Specify LUT
Contents
       ) LUT6_2_comp32_box (
         .O6(o[0]), // 1-bit o1 @ LUT6 output
         .O5(o[1]), // 1-bit o0 @ LUT5 output
         .I0(a_i[0]), // LUT input
         .I1(a_i[1]), // LUT input
```

```
        .I2(a_i[2]), // LUT input
        .I3(0), // LUT input
        .I4(0), // LUT input
        .I5(1) // LUT input control mux
       );
endmodule

module comp63(
        input [5:0]a_i,
        output wire [2:0]o
      );
  LUT6 #(//第 i 位
       .INIT('h6996_9669_9669_6996) // Specify LUT Contents
     ) LUT6_comp63_0_box (
       .O(o[2]), // 1-bit o0 @ LUT6 output
       .I0(a_i[0]), // LUT input
       .I1(a_i[1]), // LUT input
       .I2(a_i[2]), // LUT input
       .I3(a_i[3]), // LUT input
       .I4(a_i[4]), // LUT input
       .I5(a_i[5]) // LUT input
     );
  LUT6 #(//第 i+1 位
       .INIT('h8117_177E_177E_7EE8) // Specify LUT Contents
       // .INIT('h177E_7EE8_7EE8_E881) // Specify LUT
Contents
     ) LUT6_comp63_1_box (
       .O(o[1]), // 1-bit o1 @ LUT6 output
       .I0(a_i[0]), // LUT input
       .I1(a_i[1]), // LUT input
       .I2(a_i[2]), // LUT input
       .I3(a_i[3]), // LUT input
       .I4(a_i[4]), // LUT input
       .I5(a_i[5]) // LUT input
     );
  LUT6 #(//第 i+2 位
       .INIT('hFEE8_E880_E880_8000) // Specify LUT Contents
       // .INIT('h0001_0117_0117_177F) // Specify LUT
Contents
     ) LUT6_comp63_2_box (
       .O(o[0]), // 1-bit o2 @ LUT6 output
       .I0(a_i[0]), // LUT input
       .I1(a_i[1]), // LUT input
       .I2(a_i[2]), // LUT input
       .I3(a_i[3]), // LUT input
       .I4(a_i[4]), // LUT input
       .I5(a_i[5]) // LUT input
     );
endmodule

module comp153(
        input a1_i,
        input [4:0]a0_i,
        output wire [2:0]o
      );
  LUT6 #(// 第 i 位
       .INIT('h9669_6996_9669_6996) // Specify LUT Contents
       // .INIT('h6996_9669_6996_9669) // Specify LUT
Contents
     ) LUT6_comp153_0_box (
       .O(o[2]), // 1-bit o0 @ LUT6 output
       .I0(a0_i[0]), // LUT input
       .I1(a0_i[1]), // LUT input
       .I2(a0_i[2]), // LUT input
       .I3(a0_i[3]), // LUT input
       .I4(a0_i[4]), // LUT input
       .I5(a1_i) // LUT input
     );
  LUT6 #(// 第 i+1 位
       .INIT('hE881_8117_177E_7EE8) // Specify LUT Contents
       // .INIT('h177E_7EE8_E881_8117) // Specify LUT
Contents
     ) LUT6_comp153_1_box (
       .O(o[1]), // 1-bit o1 @ LUT6 output
       .I0(a0_i[0]), // LUT input
       .I1(a0_i[1]), // LUT input
       .I2(a0_i[2]), // LUT input
       .I3(a0_i[3]), // LUT input
       .I4(a0_i[4]), // LUT input
       .I5(a1_i) // LUT input
     );
  LUT6 #(//第 i+2 位
       .INIT('hFFFE_FEE8_E880_8000) // Specify LUT Contents
       // .INIT('h0001_0117_177F_7FFF) // Specify LUT
Contents
     ) LUT6_comp153_2_box (
       .O(o[0]), // 1-bit o2 @ LUT6 output
       .I0(a0_i[0]), // LUT input
       .I1(a0_i[1]), // LUT input
       .I2(a0_i[2]), // LUT input
       .I3(a0_i[3]), // LUT input
       .I4(a0_i[4]), // LUT input
       .I5(a1_i) // LUT input
     );
endmodule

module comp233(
        input [1:0]a1_i,
```

```
        input [2:0]a0_i,
        output wire [2:0]o
      );
  LUT6_2 #(//第 i，i+1 位
       .INIT('hE817_17E8_9696_9696) // Specify LUT Contents
       // .INIT('h6969_6969_17E8_E817) // Specify LUT
Contents
     ) LUT6_2_comp233_0_1_box (
       .O6(o[1]), // 1-bit o1 @ LUT6 output
       .O5(o[2]), // 1-bit o0 @ LUT5 output
       .I0(a0_i[0]), // LUT input
       .I1(a0_i[1]), // LUT input
       .I2(a0_i[2]), // LUT input
       .I3(a1_i[0]), // LUT input
       .I4(a1_i[1]), // LUT input
       .I5(1) // 1bit control mux
     );
  LUT5 #(//第 i+2 位
       .INIT('hFFE8_E800) // Specify LUT Contents
       // .INIT('h0017_17FF) // Specify LUT Contents
     ) LUT5_comp233_2_box (
       .O(o[0]), // 1-bit o2 @ LUT5 output
       .I0(a0_i[0]), // LUT input
       .I1(a0_i[1]), // LUT input
       .I2(a0_i[2]), // LUT input
       .I3(a1_i[0]), // LUT input
       .I4(a1_i[1]) // LUT input
     );
endmodule

module xor2(// o1 = ^a1[1:0], o2 = ^a2[1:0]
      input [1:0]a1,
      input [1:0]a2,
      output wire o1,
      output wire o2
     );
    LUT6_2 #(
        .INIT('h0000_0FF0_0000_6666) // Specify LUT
Contents
        // .INIT('h6666_0000_0FF0_0000) // Specify LUT
Contents
       ) LUT6_2_xor2_box (
        .O6(o2), // 1-bit o1 @ LUT6 output
        .O5(o1), // 1-bit o0 @ LUT5 output
        .I0(a1[0]), // LUT input
        .I1(a1[1]), // LUT input
        .I2(a2[0]), // LUT input
        .I3(a2[1]), // LUT input
        .I4(0), // LUT input
        .I5(1) // 1bit control mux
       );
endmodule

module mul_top(
        input clk,
        input rst_n,
        input [31:0] a_in,
        input [31:0] b_in,
        input [31:0] c_in,
        input [31:0] c_re_in,
        input replace,
        output [31:0] o
       );

wire [31:0]prod[2:1];
wire [31:0]d;
wire [31:0]s;
wire [31:0]carryo;
wire [31:0]c;
reg  [31:0]c_reg;

assign c = (replace==1)? c_re_in :c_reg;

always @(posedge clk or negedge rst_n)
  begin
    if (!rst_n)
      begin
        c_reg <= 0;
      end
    else
      begin
        c_reg <= c_in;
      end
  end


mul_comp_lcx mul_comp_u_1(
          .clk(clk),
          .rst_n(rst_n),
          .a(a_in[31:16]),
          .b(b_in[31:16]),
          .o_33(prod[1])
        );
mul_comp_lcx mul_comp_u_2(
          .clk(clk),
          .rst_n(rst_n),
          .a(a_in[15:0]),
          .b(b_in[15:0]),
```

```
          .o_33(prod[2])
        );

genvar i;
for(i=0;i<32;i=i+1)
begin
assign d[i] = ^{prod[1][i],prod[2][i],c[i]};
assign s[i] = i==0 ? d[i] : d[i]^((prod[1][i-1]&prod[2][i-
1])|(c[i-1]&(prod[1][i-1]|prod[2][i-1])));
end
for(i=1; i<9; i = i+1)
  begin
    CARRY4 CARRY4_inst (
            .CO(carryo[i*4-1:i*4-4]), // 4-bit carry out
            .O(o[i*4-1:i*4-4]), // 4-bit carry chain XOR data
out
            .CI((i==1) ? 0 : carryo[(i-1)*4-1]), // 1-bit
carry cascade input
            .CYINIT(0), // 1-bit carry initialization
            .DI(d[i*4-1:i*4-4]), // 4-bit carry-MUX data in
            .S(s[i*4-1:i*4-4]) // 4-bit carry-MUX select
input
          );
  end


endmodule

module compare_top(
    input clk,
    input rst_n,
    input [31:0]a_in,
    input [31:0]b_in,
    input [31:0]a_re_in,
    input [31:0]b_re_in,
    input replace_a,
    input replace_b,
    output wire agtb_h_o,
    output wire agtb_l_o
);
reg [31:0] a_reg;
reg [31:0] b_reg;
wire [31:0]a;
wire [31:0]b;

assign a = (replace_a==1)? a_re_in :a_reg;
assign b = (replace_b==1)? b_re_in :b_reg;


compare compare_u_h(
    .a_in(a[31:16]),
    .b_in(b[31:16]),
    .o(agtb_h_o)
);
compare compare_u_l(
    .a_in(a[15:0]),
    .b_in(b[15:0]),
    .o(agtb_l_o)
);
always @(posedge clk or negedge rst_n)
  begin
    if (!rst_n)
      begin
        a_reg <= 0;
        b_reg <= 0;
      end
    else
      begin
        a_reg <= a_in;
        b_reg <= b_in;
      end
  end
endmodule // compare_top
```

附录 D Register File 的 System Verilog 代码

```
module reg_file(
        input clk,
        input rst_n,
        input [WIDTH_RF_ADDR-1:0] ra1_i [SUPERSCALAR_AMT-
1:0],ra2_i [SUPERSCALAR_AMT-1:0], ra3_i [SUPERSCALAR_AMT-
1:0],
        wa5_i [SUPERSCALAR_AMT-1:0] , wa6_i
[SUPERSCALAR_AMT-1:0] ,
        input  [31:0] wd5_i [SUPERSCALAR_AMT-1:0], wd6_i
[SUPERSCALAR_AMT-1:0],
        input  [SUPERSCALAR_AMT-1:0]we5_i,
        input  [SUPERSCALAR_AMT-1:0]we6_i,
        output [31:0] rd1_o [SUPERSCALAR_AMT-1:0], rd2_o
[SUPERSCALAR_AMT-1:0], rd3_o [SUPERSCALAR_AMT-1:0])
        ;

    genvar i;

    reg  [31:0] rf_w_d [1:8];
    wire [31:0] rf_r_d [1:8];
    reg  [2:0] rf_w_addr [1:8];
    reg  [2:0] rf_r_addr [1:8];
    reg  rf_reg_c_e[1:4];
    reg  rf_we [1:8];

    // addr Decoder
    reg_addr_decoder reg_addr_decoder_u_r(
                    .addr_1_i(ra1_i),
                    .addr_2_i(ra2_i),
                    .we({we6_i,we5_i}),
                    .addr_o(rf_r_addr)
                    );
    reg_addr_decoder reg_addr_decoder_u_w(
                    .addr_1_i(wa5_i),
                    .addr_2_i(wa6_i),
                    .we({we6_i,we5_i}),
                    .addr_o(rf_w_addr),
                    .we_o(rf_we [1:8])
                    );
    // output_mux
    output_mux output_mux_u_rd1(
    .clk(clk),
    .rst_n(rst_n),
        .addr__i(ra1_i),
        .d1_i(rf_r_d[1]),
        .d2_i(rf_r_d[3]),
        .d3_i(rf_r_d[5]),
        .d4_i(rf_r_d[7]),
        .d_o(rd1_o)
    );
    output_mux output_mux_u_rd2(
        .clk(clk),
    .rst_n(rst_n),
        .addr__i(ra2_i),
        .d1_i(rf_r_d[2]),
        .d2_i(rf_r_d[4]),
        .d3_i(rf_r_d[6]),
        .d4_i(rf_r_d[8]),
        .d_o(rd2_o)
    );
    // wd_demux
    reg_data_demux reg_data_demux_u(
        .addr_1_i(wa5_i),
        .addr_2_i(wa6_i),
        .da_1_i(wd5_i),
        .da_2_i(wd6_i),
        .da_o(rf_w_d)
    );
    // w_regc_en
    generate
        for (i=2;i<=8;i=i+2)
            begin
               assign rf_reg_c_e[i/2] = ((wa5_i[i/2-1]==54+i)
|| (wa5_i[i/2-1]==55+i)) ? we5_i[i/2-1] : 0;
            end
    endgenerate

    generate
        for(i=1;i<=8;i=i+1)
        begin
            if (i>=3&&i<=6)
                begin
                    blk_mem_shared blk_mem_u_3456 (
                        //write
                        .clka(clk),    // input wire clka
                        .ena(1),      // input wire ena
                        .wea(rf_we[i]),      // input wire [0 :
0] wea
                        .addra(rf_w_addr[i]),  // input wire
[1 : 0] addra
                        .dina(rf_w_d[i]),    // input wire
[31 : 0] dina
                        // read
                        .clkb(clk),    // input wire clkb
                        .enb(1),      // input wire enb
```

```
                    .addrb(rf_r_addr[i]),  // input wire [1 : 0] addrb
                    .doutb(rf_r_d[i])  // output wire [31 : 0] doutb
                    );
                end
            else
                begin
                    blk_mem_only blk_mem_u_1278 (
                    //write
                    .clka(clk),    // input wire clka
                    .ena(1),      // input wire ena
                    .wea(rf_we[i]),      // input wire [0 : 0] wea
                    .addra(rf_w_addr[i]),  // input wire [2 : 0] addra
                    .dina(rf_w_d[i]),    // input wire [31 : 0] dina
                    // read
                    .clkb(clk),    // input wire clkb
                    .enb(1),      // input wire enb
                    .addrb(rf_r_addr[i]),  // input wire [2 : 0] addrb
                    .doutb(rf_r_d[i])  // output wire [31 : 0] doutb
                    );
                end
            if (i<=4)
                begin
                    blk_mem_c blk_mem_u_c (
                    //write
                    .clka(clk),    // input wire clka
                    .ena(1),      // input wire ena
                    .wea(rf_reg_c_e[i]),      // input wire [0 : 0] wea
                    .addra(wd5_i[i-1]),  // input wire [2 : 0] addra
                    .dina(wd5_i[i-1]),    // input wire [31 : 0] dina
                    // read
                    .clkb(clk),    // input wire clkb
                    .enb(1),      // input wire enb
                    .addrb(ra3_i[i-1]),  // input wire [2 : 0] addrb
                    .doutb(rd3_o[i-1])  // output wire [31 : 0] doutb
                    );
                end
        end
    endgenerate

endmodule

module reg_addr_decoder(
    input [5:0]addr_1_i[3:0],
    input [5:0]addr_2_i[3:0],
    input [7:0]we,
    output reg [2:0]addr_o[1:8],
    output reg we_o[1:8]
  );
  always @(*)
  begin
      casex ({addr_1_i[0],addr_1_i[1]})
          {6'bx,6'd1},{6'd1,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd0};
          {6'bx,6'd4},{6'd4,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd1};
          {6'bx,6'd6},{6'd6,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd2};
          {6'bx,6'd7},{6'd7,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd3};
          {6'bx,6'd33},{6'd33,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd4};
          {6'bx,6'd36},{6'd36,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd5};
          {6'bx,6'd38},{6'd38,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd6};
          {6'bx,6'd39},{6'd39,6'bx}: {we_o[1], addr_o [1]} = {we[0]|we[1], 3'd7};
          default: {we_o[1], addr_o [1]} = {1'b0, 3'bx};
      endcase
      casex ({addr_2_i[0],addr_2_i[1]})
          {6'bx,6'd2},{6'd2,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd0};
          {6'bx,6'd3},{6'd3,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd1};
          {6'bx,6'd5},{6'd5,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd2};
          {6'bx,6'd8},{6'd8,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd3};
          {6'bx,6'd34},{6'd34,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd4};
          {6'bx,6'd35},{6'd35,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd5};
          {6'bx,6'd37},{6'd37,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd6};
          {6'bx,6'd40},{6'd40,6'bx}: {we_o[2], addr_o [2]} = {we[4]|we[5], 3'd7};
```

```
        default: {we_o[2], addr_o [2]} = {1'b0, 3'bx};
    endcase
    casex ({addr_1_i[2],addr_1_i[3]})
        {6'bx,6'd25},{6'd25,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd0};
        {6'bx,6'd28},{6'd28,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd1};
        {6'bx,6'd30},{6'd30,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd2};
        {6'bx,6'd31},{6'd31,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd3};
        {6'bx,6'd41},{6'd41,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd4};
        {6'bx,6'd44},{6'd44,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd5};
        {6'bx,6'd46},{6'd46,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd6};
        {6'bx,6'd47},{6'd47,6'bx}: {we_o[7], addr_o [7]} =
{we[2]|we[3], 3'd7};
        default: {we_o[7], addr_o [7]} = {1'b0, 3'bx};
    endcase
    casex ({addr_2_i[2],addr_2_i[3]})
        {6'bx,6'd26},{6'd26,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd0};
        {6'bx,6'd27},{6'd27,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd1};
        {6'bx,6'd29},{6'd29,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd2};
        {6'bx,6'd32},{6'd32,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd3};
        {6'bx,6'd42},{6'd42,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd4};
        {6'bx,6'd43},{6'd43,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd5};
        {6'bx,6'd45},{6'd45,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd6};
        {6'bx,6'd48},{6'd48,6'bx}: {we_o[8], addr_o [8]} =
{we[6]|we[7], 3'd7};
        default: {we_o[8], addr_o [8]} = {1'b0, 3'bx};
    endcase
    casex
({addr_1_i[0],addr_1_i[1],addr_1_i[2],addr_1_i[3]})

{6'd10,6'bx,6'bx,6'bx},{6'bx,6'd10,6'bx,6'bx},{6'bx,6'bx,6'd
10,6'bx},{6'bx,6'bx,6'bx,6'd10}:
{we_o[3],addr_o[3] }={ |we[3:0], 3'd0};

{6'd11,6'bx,6'bx,6'bx},{6'bx,6'd11,6'bx,6'bx},{6'bx,6'bx,6'd
11,6'bx},{6'bx,6'bx,6'bx,6'd11}:
{we_o[3],addr_o[3] }={ |we[3:0], 3'd1};

{6'd13,6'bx,6'bx,6'bx},{6'bx,6'd13,6'bx,6'bx},{6'bx,6'bx,6'd
13,6'bx},{6'bx,6'bx,6'bx,6'd13}:
{we_o[3],addr_o[3] }={ |we[3:0], 3'd2};

{6'd16,6'bx,6'bx,6'bx},{6'bx,6'd16,6'bx,6'bx},{6'bx,6'bx,6'd
16,6'bx},{6'bx,6'bx,6'bx,6'd16}:
{we_o[3],addr_o[3] }={ |we[3:0], 3'd3};
        default: {we_o[3],addr_o[3] }={1'b0,3'bx};
    endcase
    casex
({addr_2_i[0],addr_2_i[1],addr_2_i[2],addr_2_i[3]})

{6'd9,6'bx,6'bx,6'bx},{6'bx,6'd9,6'bx,6'bx},{6'bx,6'bx,6'd9,
6'bx},{6'bx,6'bx,6'bx,6'd9}:{we_o[4],addr_o[4] }={ |we[7:4],
3'd0};

{6'd12,6'bx,6'bx,6'bx},{6'bx,6'd12,6'bx,6'bx},{6'bx,6'bx,6'd
12,6'bx},{6'bx,6'bx,6'bx,6'd12}:{we_o[4],addr_o[4] }={ |we[7
:4],3'd1};

{6'd14,6'bx,6'bx,6'bx},{6'bx,6'd14,6'bx,6'bx},{6'bx,6'bx,6'd
14,6'bx},{6'bx,6'bx,6'bx,6'd14}:
{we_o[4],addr_o[4] }={ |we[7:4],3'd2};

{6'd15,6'bx,6'bx,6'bx},{6'bx,6'd15,6'bx,6'bx},{6'bx,6'bx,6'd
15,6'bx},{6'bx,6'bx,6'bx,6'd15}:{we_o[4],addr_o[4] }={ |we[7
:4], 3'd3};
        default: {we_o[4],addr_o[4] }={1'b0, 3'bx};
    endcase
    casex
({addr_1_i[0],addr_1_i[1],addr_1_i[2],addr_1_i[3]})

{6'd18,6'bx,6'bx,6'bx},{6'bx,6'd18,6'bx,6'bx},{6'bx,6'bx,6'd
18,6'bx},{6'bx,6'bx,6'bx,6'd18}:
{we_o[5],addr_o[5] }={ |we[3:0], 3'd0};

{6'd19,6'bx,6'bx,6'bx},{6'bx,6'd19,6'bx,6'bx},{6'bx,6'bx,6'd
19,6'bx},{6'bx,6'bx,6'bx,6'd19}:
{we_o[5],addr_o[5] }={ |we[3:0], 3'd1};

{6'd21,6'bx,6'bx,6'bx},{6'bx,6'd21,6'bx,6'bx},{6'bx,6'bx,6'd
21,6'bx},{6'bx,6'bx,6'bx,6'd21}:
{we_o[5],addr_o[5] }={ |we[3:0], 3'd2};

{6'd24,6'bx,6'bx,6'bx},{6'bx,6'd24,6'bx,6'bx},{6'bx,6'bx,6'd
```

```
24,6'bx},{6'bx,6'bx,6'bx,6'd24}:
{we_o[5],addr_o[5] }={ |we[3:0], 3'd3};
        default:  {we_o[5],addr_o[5] }={ 1'b0, 3'bx};
      endcase
      casex
({addr_2_i[0],addr_2_i[1],addr_2_i[2],addr_2_i[3]})

{6'd17,6'bx,6'bx,6'bx},{6'bx,6'd17,6'bx,6'bx},{6'bx,6'bx,6'd
17,6'bx},{6'bx,6'bx,6'bx,6'd17}:
{we_o[6],addr_o[6] }={ |we[7:4], 3'd0};

{6'd20,6'bx,6'bx,6'bx},{6'bx,6'd20,6'bx,6'bx},{6'bx,6'bx,6'd
20,6'bx},{6'bx,6'bx,6'bx,6'd20}:
{we_o[6],addr_o[6] }={ |we[7:4], 3'd1};

{6'd22,6'bx,6'bx,6'bx},{6'bx,6'd22,6'bx,6'bx},{6'bx,6'bx,6'd
22,6'bx},{6'bx,6'bx,6'bx,6'd22}:
{we_o[6],addr_o[6] }={ |we[7:4], 3'd2};

{6'd23,6'bx,6'bx,6'bx},{6'bx,6'd23,6'bx,6'bx},{6'bx,6'bx,6'd
23,6'bx},{6'bx,6'bx,6'bx,6'd23}:
{we_o[6],addr_o[6] }={ |we[7:4], 3'd3};
        default:{we_o[6],addr_o[6] }={1'b0, 3'bx};
      endcase
  end
endmodule // reg_decoder

module reg_data_demux(
    input [5:0]addr_1_i[3:0],
    input [5:0]addr_2_i[3:0],
    input [31:0]da_1_i[3:0],
    input [31:0]da_2_i[3:0],
    output reg [31:0]da_o[1:8]
    );
    always @(*)
    begin
        casex ({addr_1_i[0],addr_1_i[1]})

{6'd4,6'bx},{6'd1,6'bx},{6'd6,6'bx},{6'd7,6'bx},{6'd33,6'bx}
,{6'd36,6'bx},{6'd38,6'bx},{6'd39,6'bx}: da_o[1] =
da_1_i[0];

{6'bx,6'd1},{6'bx,6'd4},{6'bx,6'd6},{6'bx,6'd7},{6'bx,6'd33}
,{6'bx,6'd36},{6'bx,6'd38},{6'bx,6'd39}: da_o[1] =
da_1_i[1];
            default: da_o[1] =32'bx;
        endcase

        casex ({addr_1_i[0],addr_1_i[1]})

{6'd4,6'bx},{6'd1,6'bx},{6'd6,6'bx},{6'd7,6'bx},{6'd33,6'bx}
,{6'd36,6'bx},{6'd38,6'bx},{6'd39,6'bx}: da_o[1] =
da_1_i[0];

{6'bx,6'd1},{6'bx,6'd4},{6'bx,6'd6},{6'bx,6'd7},{6'bx,6'd33}
,{6'bx,6'd36},{6'bx,6'd38},{6'bx,6'd39}: da_o[1] =
da_1_i[1];
            default: da_o[1] =32'bx;
        endcase

        casex ({addr_2_i[0],addr_2_i[1]})

{6'd2,6'bx},{6'd3,6'bx},{6'd5,6'bx},{6'd8,6'bx},{6'd34,6'bx}
,{6'd35,6'bx},{6'd37,6'bx},{6'd40,6'bx}: da_o[2] =
da_2_i[0];

{6'bx,6'd2},{6'bx,6'd3},{6'bx,6'd5},{6'bx,6'd8},{6'bx,6'd34}
,{6'bx,6'd35},{6'bx,6'd37},{6'bx,6'd40}: da_o[2] =
da_2_i[1];
            default: da_o[2] =32'bx;
        endcase

        casex ({addr_1_i[2],addr_1_i[3]})

{6'd25,6'bx},{6'd28,6'bx},{6'd30,6'bx},{6'd31,6'bx},{6'd41,6
'bx},{6'd44,6'bx},{6'd46,6'bx},{6'd47,6'bx}: da_o[7] =
da_1_i[2];

{6'bx,6'd25},{6'bx,6'd28},{6'bx,6'd30},{6'bx,6'd31},{6'bx,6'
d41},{6'bx,6'd44},{6'bx,6'd46},{6'bx,6'd47}: da_o[7] =
da_1_i[3];
            default: da_o[7] =32'bx;
        endcase

        casex ({addr_2_i[2],addr_2_i[3]})

{6'd26,6'bx},{6'd27,6'bx},{6'd29,6'bx},{6'd32,6'bx},{6'd42,6
'bx},{6'd43,6'bx},{6'd45,6'bx},{6'd48,6'bx}: da_o[8] =
da_2_i[2];

{6'bx,6'd26},{6'bx,6'd27},{6'bx,6'd29},{6'bx,6'd32},{6'bx,6'
d42},{6'bx,6'd43},{6'bx,6'd45},{6'bx,6'd48}: da_o[8] =
da_2_i[3];
            default: da_o[8] =32'bx;
        endcase

        casex
({addr_1_i[0],addr_1_i[1],addr_1_i[2],addr_1_i[3]})
```

```
{6'd10,6'bx,6'bx,6'bx},{6'd11,6'bx,6'bx,6'bx},{6'd13,6'bx,6'
bx,6'bx},{6'd16,6'bx,6'bx,6'bx}: da_o[3] = da_1_i[0];

{6'bx,6'd10,6'bx,6'bx},{6'bx,6'd11,6'bx,6'bx},{6'bx,6'd13,6'
bx,6'bx},{6'bx,6'd16,6'bx,6'bx}: da_o[3] = da_1_i[1];

{6'bx,6'bx,6'd10,6'bx},{6'bx,6'bx,6'd11,6'bx},{6'bx,6'bx,6'd
13,6'bx},{6'bx,6'bx,6'd16,6'bx}: da_o[3] = da_1_i[2];

{6'bx,6'bx,6'bx,6'd10},{6'bx,6'bx,6'bx,6'd11},{6'bx,6'bx,6'b
x,6'd13},{6'bx,6'bx,6'bx,6'd16}: da_o[3] = da_1_i[3];
        default:da_o[3] = 32'bx;
    endcase

    casex ({addr_2_i[0],addr_2_i[1],addr_2_i[2],addr_2_i[3]})

{6'd9,6'bx,6'bx,6'bx},{6'd12,6'bx,6'bx,6'bx},{6'd14,6'bx,6'b
x,6'bx},{6'd15,6'bx,6'bx,6'bx}:da_o[4] = da_2_i[0];

{6'bx,6'd9,6'bx,6'bx},{6'bx,6'd12,6'bx,6'bx},{6'bx,6'd14,6'b
x,6'bx},{6'bx,6'd15,6'bx,6'bx}:da_o[4] = da_2_i[1];

{6'bx,6'bx,6'd9,6'bx},{6'bx,6'bx,6'd12,6'bx},{6'bx,6'bx,6'd1
4,6'bx},{6'bx,6'bx,6'd15,6'bx}:da_o[4] = da_2_i[2];

{6'bx,6'bx,6'bx,6'd9},{6'bx,6'bx,6'bx,6'd12},{6'bx,6'bx,6'bx
,6'd14},{6'bx,6'bx,6'bx,6'd15}:da_o[4] = da_2_i[3];
        default:da_o[4] = 32'bx;
    endcase

    casex ({addr_1_i[0],addr_1_i[1],addr_1_i[2],addr_1_i[3]})

{6'd18,6'bx,6'bx,6'bx},{6'd19,6'bx,6'bx,6'bx},{6'd21,6'bx,6'
bx,6'bx},{6'd24,6'bx,6'bx,6'bx}:da_o[5] = da_1_i[0];

{6'bx,6'd18,6'bx,6'bx},{6'bx,6'd19,6'bx,6'bx},{6'bx,6'd21,6'
bx,6'bx},{6'bx,6'd24,6'bx,6'bx}:da_o[5] = da_1_i[1];

{6'bx,6'bx,6'd18,6'bx},{6'bx,6'bx,6'd19,6'bx},{6'bx,6'bx,6'd
21,6'bx},{6'bx,6'bx,6'd24,6'bx}:da_o[5] = da_1_i[2];

{6'bx,6'bx,6'bx,6'd18},{6'bx,6'bx,6'bx,6'd19},{6'bx,6'bx,6'b
x,6'd21},{6'bx,6'bx,6'bx,6'd24}:da_o[5] = da_1_i[3];
        default:da_o[5] = 32'bx;
    endcase

    casex ({addr_2_i[0],addr_2_i[1],addr_2_i[2],addr_2_i[3]})

{6'd17,6'bx,6'bx,6'bx},{6'd20,6'bx,6'bx,6'bx},{6'd22,6'bx,6'
bx,6'bx},{6'd23,6'bx,6'bx,6'bx}:da_o[6] = da_2_i[0];

{6'bx,6'd17,6'bx,6'bx},{6'bx,6'd20,6'bx,6'bx},{6'bx,6'd22,6'
bx,6'bx},{6'bx,6'd23,6'bx,6'bx}:da_o[6] = da_2_i[1];

{6'bx,6'bx,6'd17,6'bx},{6'bx,6'bx,6'd20,6'bx},{6'bx,6'bx,6'd
22,6'bx},{6'bx,6'bx,6'd23,6'bx}:da_o[6] = da_2_i[2];

{6'bx,6'bx,6'bx,6'd17},{6'bx,6'bx,6'bx,6'd20},{6'bx,6'bx,6'b
x,6'd22},{6'bx,6'bx,6'bx,6'd23}:da_o[6] = da_2_i[3];
        default: da_o[6] = 32'bx;
    endcase
    end
endmodule

module output_mux(
    input clk,
    input rst_n,
    input [5:0]addr__i[3:0],
    input [31:0]d1_i,
    input [31:0]d2_i,
    input [31:0]d3_i,
    input [31:0]d4_i,
    output reg [31:0]d_o[3:0]
  );
  genvar i;
    reg [5:0]addr__i_r[3:0];
  for (i=0;i<=3;i=i+1)
  begin
always @(posedge clk or negedge rst_n)
    if(!rst_n)addr__i_r[i]<=0;
    else addr__i_r[i]<=addr__i[i];
end
  generate
    for(i=0;i<=3;i=i+1)
      begin
      always@(*)
      begin
        if(i<=1)
          begin
            casex (addr__i_r[i])
              0:d_o[i] = 0;
              1,2,3,4,5,6,7,8: d_o[i] = d1_i;
              9,10,11,12,13,14,15,16: d_o[i] = d2_i;
              17,18,19,20,21,22,23,24: d_o[i] = d3_i;
              default: d_o[i] = 32'bx;
            endcase
```

```
        end
      else
        begin
          casex (addr__i_r[i])
            0:d_o[i] = 0;
            9,10,11,12,13,14,15,16: d_o[i] = d2_i;
            17,18,19,20,21,22,23,24: d_o[i] = d3_i;
            25,26,27,28,29,30,31,32: d_o[i] = d4_i;
            default: d_o[i] = 32'bx;
          endcase
        end
    end
    end
  endgenerate

endmodule
```