电子科技大学电子科学与工程学院

实 验 报 告

(2022 - 2023 - 1)

课程名称 IC 综合实验 2（IC 设计）

实验名称 设计仿真

指导老师 王忆文

学生姓名 曾立伟，高德睿，蒋熠凡，易健驰

学生学号 2020340105007，2020340102017，

2020340102009，2020340107022

电子科技大学电子科学与工程学院

电 子 科 技 大 学

实 验 报 告

实验地点：清水河校区芯火基地 B 区 519

实验时间：2022.11 – 2022.12

报告目录

一、实验室名称：清水河校区芯火基地 B 区 519

二、实验项目名称：除法器&处理器

三、实验学时：100 学时

四、实验原理：请附页

五、实验目的：请附页

六、实验内容：请附页

七、实验器材（设备、元器件）：请附页

八、实验步骤：请附页

九、实验数据及结果分析：请附页

十、实验结论：请附页

十一、总结及心得体会：请附页

十二、对本实验过程及方法、手段的改进建议：请附页

实验报告成绩：_____

目录

一、 实验室名称 7
二、 实验项目名称 7
三、 实验学时 7
四、 实验原理 7
1. SRT除法 7
(1) SRT迭代式的推导 7
(2) 商数字集及冗余度 8
(3) 遏制条件 9
(4) 优化后的商选择过程 10
(5) on the fly 在线舍入算法 11
2. $RISCV$ 12
(1) 基本整数子集的程序员模型 12
(2) 基本指令格式 13
(3) 整数计算指令 13
a) 整数寄存器-立即数指令 13
b) 整数寄存器-寄存器操作 14
(4) 控制转移指令 14
a) 无条件跳转 14
b) 条件分支 15
(5) Load 和 store 指令 15
五、 实验目的 15
六、 实验内容 16
七、 实验器材（设备、元器件） 16
八、 实验步骤 16
1. 编写除法器并仿真验证 16
(1) 顶层控制 16
(2) 无符号除法结构框图 16
(3) 子模块介绍 17
a) 预处理模块 17
b) 有符号除法拓展 18
c) 选商模块 18
(4) 测试与优化 19
a) 两种常见的双操作数加法器的优化原理 19
A. 超前进位加法器CLA（$Carry\ Lookahead\ Adder$） 19
B. 进位选择加法器（$Carry-select\ Adder$） 20
b) 对于关键路径的延时的优化 20
c) 对于计算正确率的测试与优化 22
2. 使用$Verilog$编写代码（实现部分指令的代码因面积限制被注释） 25
(1) $FRAG\_adder$ 25
a) 参数 25
b) 端口 25
c) 作用 25

(2) $FRAG\_ALU\_ctrl$ 25
a) 参数 25
b) 端口 25
c) 作用 26
(3) $FRAG\_ALU$ 26
a) 参数 26
b) 端口 26
c) 作用 26
(4) $FRAG\_ctrl$ 27
a) 参数 27
b) 端口 27
c) 作用 28
(5) $FRAG\_data\_mem$ 28
a) 参数 28
b) 端口 28
c) 作用 28
(6) $FRAG\_forward$ 28
a) 参数 29
b) 端口 29
c) 作用 29
(7) $FRAG\_hazard$ 30
a) 参数 30
b) 端口 30
c) 作用 31
(8) $FRAG\_imm\_gen$ 31
a) 参数 31
b) 端口 31
c) 作用 31
(9) $FRAG\_inst\_mem$ 31
a) 参数 31
b) 端口 31
c) 作用 31
(10) $FRAG\_JorB\_ctrl$ 31
a) 参数 32
b) 端口 32
c) 作用 32
(11) $FRAG\_mux\_2$ 32
a) 参数 32
b) 端口 32
c) 作用 32
(12) $FRAG\_mux\_3$ 32
a) 参数 32
b) 端口 32
c) 作用 32

(13) $FRAG\_pipline$ 33
a) 参数 33
b) 端口 33
c) 作用 33
(14) $FRAG\_register\_file$ 33
a) 参数 33
b) 端口 33
c) 作用 34
(15) $FRAG\_UART$ 34
a) 参数 34
b) 端口 34
c) 作用 34
(16) TOP(MODULE不单独介绍) 34
3. 使用VCS进行仿真 38
4. 使用$FPGA$进行验证 39
九、 实验数据及结果分析 43
1. 除法器的优化 43
a) 对于关键路径的延时的优化 43
b) 对于计算正确率的测试与优化 45
2. 处理器数据记录 48
十、 实验结论 48
十一、 总结及心得体会 48
十二、 对本实验过程及方法、手段的改进建议 48
十三、 代码 48
1. $FRAG\_adder.v$ 48
2. $FRAG\_ALU\_ctrl.v$ 48
3. $FRAG\_ALU.v$ 51
4. $FRAG\_ctrl.v$ 54
5. $FRAG\_data\_mem.v$ 56
6. $FRAG\_forward.v$ 65
7. $FRAG\_hazard.v$ 66
8. $FRAG\_imm\_gen.v$ 67
9. $FRAG\_inst\_mem.v$ 68
10. $FRAG\_JorB\_ctrl.v$ 69
11. $FRAG\_mux\_2.v$ 69
12. $FRAG\_mux\_3.v$ 70
13. $FRAG\_pipline.v$ 70
14. $FRAG\_register\_file.v$ 71
15. $FRAG\_UART.v$ 72
16. $MODULE\_ex\_mem.v$ 77
17. $MODULE\_ex.v$ 80
18. $MODULE\_id\_ex.v$ 82
19. $MODULE\_id.v$ 87
20. $MODULE\_if\_id.v$ 88

21. $MODULE\_if.v$ 90
22. $MODULE\_mem\_wb.v$ 92
23. $MODULE\_mem.v$ 93
24. $MODULE\_wb.v$ 95
25. $on\_the\_fly\_conversion.v$. 95
26. $pre\_processing.v$ 97
27. $r8\_qds.v$. 103
28. $r8\_table.v$ 105
29. $radix4\_table.v$ 107
30. $srt\_8\_div.v$ 110
31. $TOP.v$ 116
32. $auto\_tst.v$ 126
33. $TB.v$ 128

一、 实验室名称

清水河校区芯火基地B区519。

二、 实验项目名称

参照$RISCV$指令集设计带有除法指令的32位处理器，实现了I指令集中除$FENCE$，$ECALL$，$EBREAK$外的37条指令（实现部分指令的代码因面积限制被注释，注释后共14条，涉及各个类型）和M指令集中的4条除法指令，其中除法指令使用基于$Radix8-SRT$算法的，可完成有符号数和无符号数除法的32位除法器实现。该处理器具有5级流水线结构，能有效防止数据冒险和控制冒险。同时使用$UART$完成指令，数据的接收和数据的发送。

三、 实验学时

100学时。

四、 实验原理

1. SRT除法

(1) SRT迭代式的推导

将除法操作定义如下：

$$x = q \cdot d + rem$$

式中，$|rem| < |d|$, x代表被除数，d代表除数，q代表商，rem代表余数。

在使用数字递归算法（$Digit\ Recurrence\ Algorithms$）进行除法操作时迭代$n$次，每次迭代中产生基$r$的商，其中商的最高位先产生。对于整数除法，它应该等于被除数x的位数。为了理解方便，我们在进行后续推导以前先将被除数和除数移位至最高有效位恰好在小数点的右边一位，将其视为初始的x和d。这个操作是自然的，因为除法运算是从高位到低位得出结果。将第j次迭代后的商值记为$q[j]$（第j次及其以前迭代产生的商值进行叠加之后的结果）：

$$q[j] = \sum_{i=1}^{j} q_i r^{-i}$$

其中，q_i为第i次迭代时产生的商值，那么最后的商就是第n次迭代后的商：

$$q[n] = \sum_{i=1}^{n} q_i r^{-i}$$

无论对于恢复余数、不恢复余数或是SRT算法，都是先通过部分被除数与除数进行选商，然后用被除数减去商和除数的积，最后再左移1位，重复上述步骤。因此定义部分余数$w[j]$为：

$$w[j] = r^j(x - dq[j])$$

为了推导相邻部分余数的关系，我们将上式续写：

$$w[j+1] = r^{j+1}(x - dq[j+1])$$

可以得到：

$$w[j+1] = rw[j] - dq_{j+1}$$

其中初始值$w[0] = x$，并且$w[j]$必须满足下式：

$$-d < w[j] < d$$

在手算除法中，商的选择是根据余数和除数自己来选择的，其具有唯一性，

计算机在每次迭代时计算的商值也是唯一的，同时在计算每次迭代的商值时需要先移位，扩大其值后进行选商，那么商q_{j+1}的选择就需要一个含有移位后余数$rw[j]$和除数d的函数来决定，称为商的选择函数。而这个选择函数必须保证$w[j+1]$的有界性，假设其可以表示为：

$$q_{j+1} = SEL(rw[j], d)$$

下面以手算除法为基础，阐述下SRT算法的步骤。

1) 设迭代次数为j，将其计算得到的余数$w[j]$进行移位，得到移位后的余数$rw[j]$。
2) 根据余数$rw[j]$和除数d，通过商的选择函数来选择本次的商q_{j+1} 。
3) 将本次迭代产生的商q_{j+1}和除数d相乘得到$q_{j+1}d$。
4) 用移位后的余数$rw[j]$减去商和除数的乘积$q_{j+1}d$，得到下一次循环的余数$w[j+1]$，以此类推。

在本次$IC2$综合实验中，由于工艺和面积的限制，我们不可能采用全精度的除数和余数来进行选商，这就要求采用部分余数以及部分除数来生成查找表，用以减小电路的复杂度，延迟以及面积，也就是说采用截断后的部分余数和部分除数来进行选商。而截断同样会产生一系列的问题，例如商选择集的改变，误差的增加等等。如此，上面的SRT算法步骤第二步将会进行改变，改为通过截断后的部分余数$rw[j]$和部分除数d，通过商的选择函数来选择本次的商q_{j+1} 。而其余步骤由于不涉及查找表部分，同时需要保留全精度的数据来进行计算，保证数据的正确性，故其余部分不改变，采用全精度的除数和余数。

按照上面所说的流程，可以发现基的大小和最终迭代次数n成反比。SRT算法的基越大，每次迭代时移位的位数越大，生成的结果位宽越宽，那么迭代次数自然就会变少。但是，基数增大，导致截断后部分余数和部分除数增大，同时商选择函数选择的商值也会增大，这样就造成其变得更复杂，延迟也就随之增大。所以，基的大小需要综合考虑（包括延迟，面积，功耗）。本次综合实验选用基的大小为8，也就是$SRT8$算法。

(2) **商数字集及冗余度**

未截断的SRT算法的商选择数字集为$\{0,1,\dots,r-1\}$，但是如上面所说，不能采用全精度的除数和余数来进行选商，而是需要进行截断，而截断后采用部分余数和部分除数来查找表会造成一定的问题，商的结果可能会不准确。例如，被除数为6.1，除数为2.3，未截断时计算机选择的商为2。但是对于采用部分余数和部分除数来说，假设其会以6作为部分余数，2作为部分除数，这是选择的商就为3，造成了计算机计算错误。针对于上面的计算错误，一般就有两种做法，第一种做法是在选商之后的减法操作中，如果发现结果为负数，那么就恢复部分余数，改选择的商为2后重新进行计算，但是这样会多延迟一拍，严重降低了计算机性能。另一种是允许部分余数是负数的情况，继续向下进行计算，这种方法会使得原来的商数字集不能选择出对应的商。因此，需要改变商数字集来符合这种方法。

新的方法允许部分余数是负数，从而造成新的商数字集也有负数的商值，同时由于部分余数的正负对称性，因此，新的商数字集也应该是对称的，故新的商数字集由一个对称的带符号连续整数组成。设a为其中最大的整数，那么此数字集可以如下式表示：

$$q_{j+1} = \{-a, -a+1, \dots\dots, -1, 0, 1, \dots\dots, a-1, a\}$$

只有新的商数字集的个数比以前的商数字集大，才能保证计算机在截断

后产生错误结果还能继续向下进行计算，也就是说其必须是大于r个（SRT的基）的连续整数集合，因此可以得出：

$$a \geq \frac{r}{2}$$

按照新的商数字集来说，其会针对于一个确定部分余数和部分除数，商的选择不一定唯一，就拿上面的例子来说，其商既可以选择2，也可以选择3，这个重叠区域的大小可以用下面的冗余因子ρ来决定，同时其还可以表示商数字集的冗余度：

$$\rho = \frac{a}{r-1}, \rho > \frac{1}{2}$$

由于a最大可取到$a = r - 1$，这时$\rho = 1$，其称为最大冗余。对于相同的基r来说，如果商数字集中a的值增大，那么其冗余度就会变大，商数字选择函数就会更容易实现，也就是说查找表所需要截断的位宽就会变小。查找表产生的延迟就越小，但是与此同时，除数和每次得到的可能的商的乘积$q_{j+1}d$就会越复杂，其延迟就越高。

在本次设计中采用的是基8 SRT算法，选取冗余商集为$\{-6, \ldots\ldots, -1, 0, +1, \ldots\ldots 6\}$，但当商集$a$变大时，实现除数和商的乘积较为复杂，不同于$d$和$-d$，$2d$和$-2d$实现起来简单。（$-d$仅是将其取反，$2d$是将其左移一位，$-2d$也是对$2d$进行取反，而例如$3d$就需要实现$d$和$2d$的加法，这会产生一个加法器。）因此我们查阅资料，对选商函数进行了优化。我们参考了$Algorithm_for_high_speed_shared_radix_8_division_and_radix_8_square_root$这篇论文，将商选择函数分解成两部分，减小了商选择部分的资源占用，如下两式所示：

$$w_x[j+1] = rw[j] - dq_{x_{j+1}}, with\ q_{x_{j+1}} = -4, 0, +4$$
$$w[j+1] = w_x[j+1] - dq_{y_{j+1}}, with\ q_{y_{j+1}} = -2, -1, 0, +1, +2$$

具体的商选择过程将在(4)节介绍。第二步的商集为$\{-2, -1, 0, 1, 2\}$，冗余度$\rho = 2/3 > 1/2$。

(3) **遏制条件**

通过(2)节的分析可以看出可以看出，商选择函数有一定的要求，必须使得下一次的余数有界，其绝对值至少不能大于除数，这个条件称为遏制条件。

设第j次迭代后，余数$w[j]$的上边界和下边界分别为$A[j]$和$B[j]$，即：

$$\forall j, A[j] \leq w[j] \leq B[j]$$

需要注意的是：其上下边界不是q_{j+1}为某一个值时的上下边界，而是对于所有的q_{j+1}，取其上界和下界。

设第$j+1$次迭代时通过查找表查到的选商为$q_{j+1} = k$，这时移位后余数$rw[j]$的选择区间为$[L_k(j), U_k(j)]$。因此：

$$L_k(j) \leq rw[j] \leq U_k(j) \Rightarrow A_{j+1} \leq w[j+1] = rw[j] - k \cdot d \leq B_{j+1} \quad (a)$$

从上式，可以得到：

$$A_{j+1} = L_{k_1}(j) - k_1 \cdot d, B_{j+1} = U_{k2}(j) - k_2 \cdot d \quad (b)$$

由于$A[j]$和$B[j]$，其代表所有q_{j+1}的余数上界和下界，而$L_k(j)$仅针对于$q_{j+1} = k$时，其移位后余数的边界，故上式的商值选取必定不同，为了区分起见，分别选用k_1和k_2对应$A[j]$和$B[j]$的不同商值。

在除法中，当部分余数越大时，其下次选择的商值越大。故可以认为在第$j+1$次迭代时，$A[j]$和$B[j]$分别是$q_{j+1} = a$时移位后余数的上界和$q_{j+1} = -a$时

移位后余数的下界，也就是说$k_1 = a$，$k_2 = -a$，即：

$$L_{-a}(j) = rA_j, U_a(j) = rA_j \tag{c}$$

将(c)式代入(b), 得到：

$$A_{j+1} = rA_j - a \cdot d, B_{j+1} = rB_j - (-a) \cdot d$$

由于等式右边的a，d与迭代次数j无关，故$A[j]$和$B[j]$与j无关，可以分别用A和B表示，同时$L_k(j)$与$U_k(j)$也与j无关，可以改为L_k与U_k。上式变为：

$$A = rA - a \cdot d, B = rB - (-a) \cdot d$$

解上面的方程可以得到：

$$A = \frac{a}{r-1}d = \rho \cdot d, B = \frac{-a}{r-1}d = -\rho \cdot d \tag{d}$$

当$\rho = 1$时，边界上下界分别为d和$-d$，这是在除法中不允许的。因此冗余度的范围为：$\frac{1}{2} < \rho < 1$

将式(d)带入到式(a)后可得：

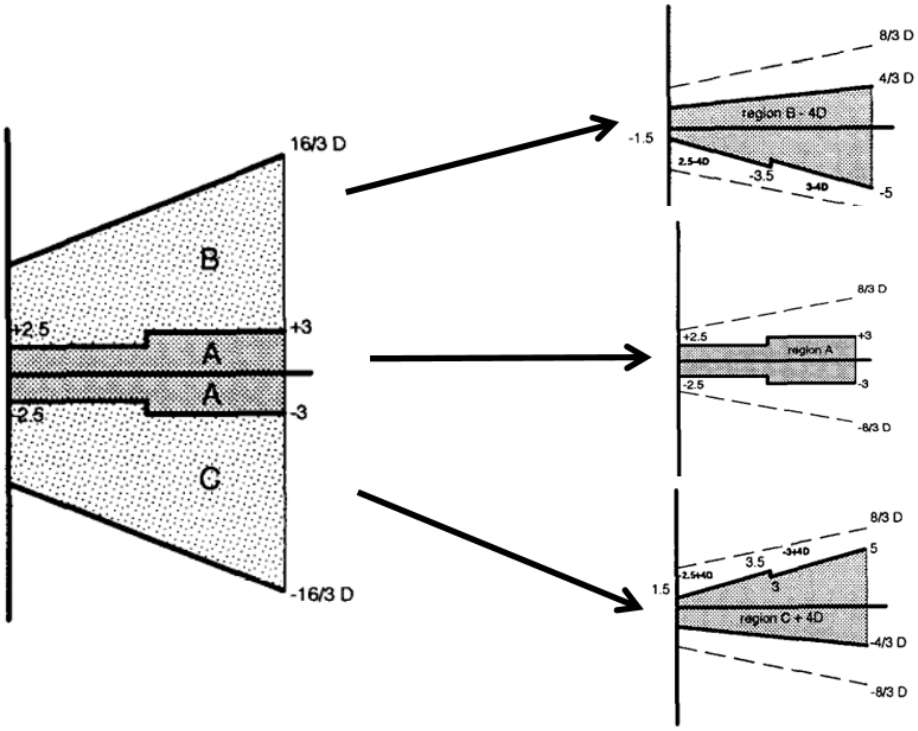
$$U_k = (\rho + k)d, L_k = (-\rho + k)d \tag{*}$$

$(*)$式就是*SRT*除法的遏制条件。

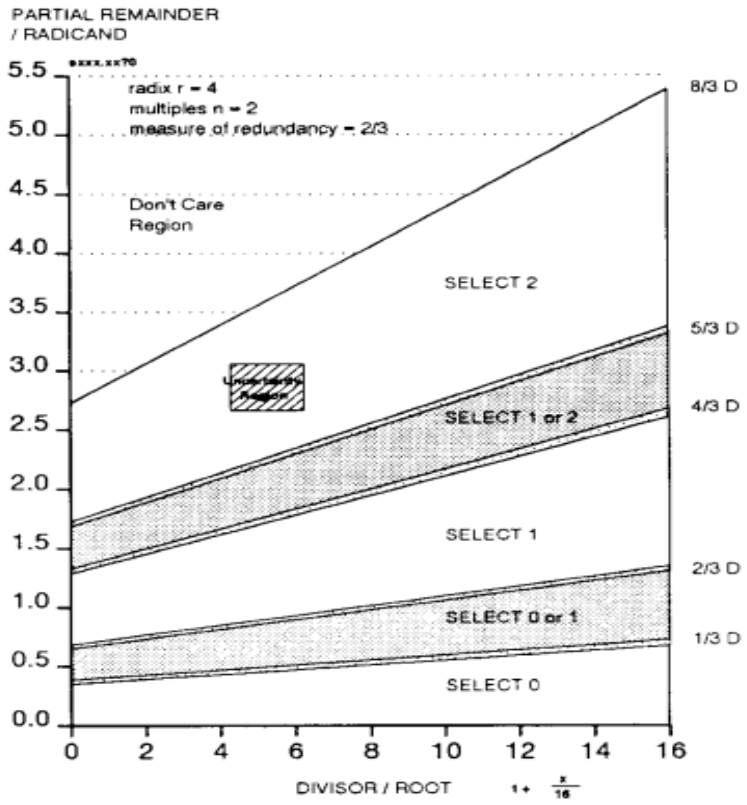
(4) **优化后的商选择过程**

如(2)节所述，我们将每次迭代分解为两步，相较于传统基8选商的复杂真值表大大减少资源占用，同时相较于基4又有更高的迭代速度。

首先通过选第一次商将基8的$rw[j] - d$图转换到基4可以操作的范围内；图中B区域也即$rw[j]$过大，选取$q_{x_{j+1}} = 4$来让B区域转移到斜率位于$(-8/3,8/3)$范围；C 区域操作类似，选取$q_{x_{j+1}} = -4$；A区域不需要进行操作，选取$q_{x_{j+1}} = 0$。



经过第一次选商之后，部分余数被增大或是减小到更小的范围内（冗余度为$2/3$的基4除法），以方便我们进行第二次选商。此时商集为$\{-2,-1,0,1,2\}$，商和除数的积易于表示，做出第二步的$rw[j] - d$图如下：



$rw[j]-d$可以直观地反映出选商过程。例如：$q=1$时，$U_k=(\rho+k)d=5/3d, L_k=(-\rho+k)d=1/3d$，当坐标$(rw[j],d)$落入这个范围时，该次迭代过程的选商结果为1。

(5) on the fly 在线舍入算法

对于最终的商而言，其是对每次循环得到的商值进行叠加而得到的，可以在每次迭代时得到一定的商值后，用加法器将上一次的运算结果和这一次产生的值进行相加后得到这次迭代后的结果。但是这样会增加一个加法器，会大大增加这部分的延迟，同样这部分在关键路径之内，故如何减小这部分的延迟也是提高此次设计性能的一部分。而提高这部分的算法就是在线舍入算法。每次迭代选出一个商之后，输入到$on\ the\ fly$恢复模块，在下一次时钟沿到来前计算出该位恢复后的结果。

本质上我们的目的是将冗余商集$\{-6,-5\ldots\ldots,0,\ldots\ldots5,6\}$恢复成非冗余商集$\{0,1,2\ldots\ldots,7\}$，需要考虑的问题是对于冗余商集中负数的恢复。核心的思路是：**对于出现负数结果的某一位，为了将此位结果恢复为正，需要回到上一位（权重更大的一位）结果，使其减一**。

设相邻两位的$srt-8$除法结果分别为k_{m+1}，k_m，高位k_{m+1}是经过恢复算法之后的结果，为正。假设k_m为负值，下面我们通过计算来验证之前的恢复算法正确性。

设k_m对应的权重为r^m，r 为基，此两位对应的数值应该为$(k_m+rk_{m+1})r^m$。为了将负值k_m恢复成正值，我们将高位结果k_{m+1}减去1，相当于用更大的权重来将负数补充成正值。为了使此两位对应数值结果不变，新的r^m位对应的除法结果应该为：

$$k'_m=(k_m+rk_{m+1})-r(k_{m+1}-1)=k_m+r=r-|k_m|$$

在代码实现中，我们考虑用A，B两个寄存器实现$on\ the\ fly$算法。A寄存

器存放正确的结果，B寄存器存放每一位相较于A寄存器减1的结果；一旦遇到负数需要恢复的情况，便调用B寄存器中上一位的值，同时当前位进行修正，结果为$r-|q_{j+1}|$（假设进行恢复前的输入为商q_{j+1}），可以将两个寄存器的归纳式列出如下：

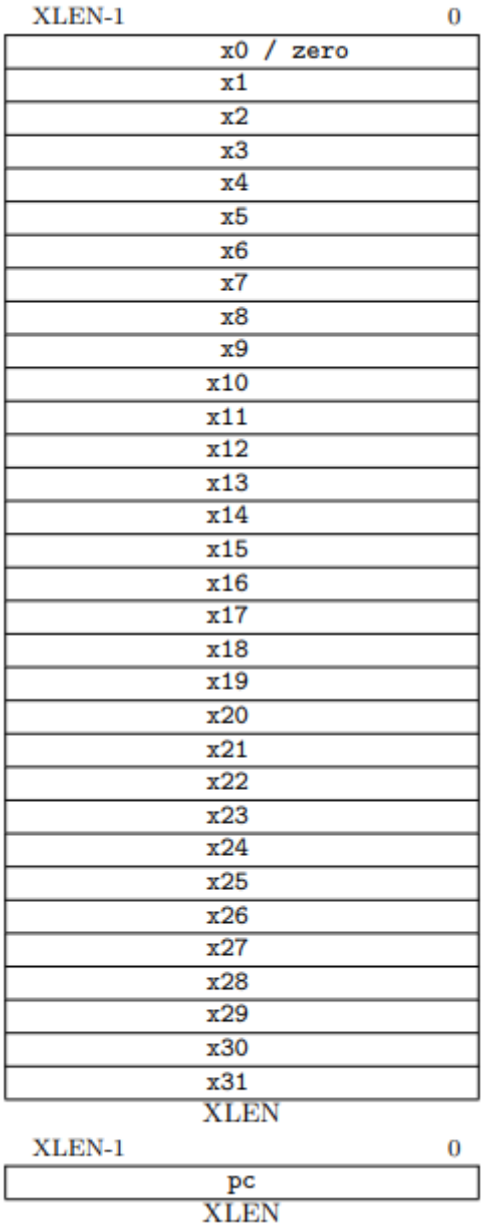
$$A[j+1]=\begin{cases}(A[j],q_{j+1}) & \cdots\cdots\cdots q_{j+1}\geq 0\\ (B[j],(r-|q_{j+1}|)) & \cdots\cdots\cdots q_{j+1}\leq 0\end{cases}$$

$$B[j+1]=\begin{cases}(A[j],(q_{j+1}-1)) & \cdots\cdots\cdots q_{j+1}\geq 0\\ (B[j],(r-|q_{j+1}|-1)) & \cdots\cdots\cdots q_{j+1}\leq 0\end{cases}$$

2. *RISCV*

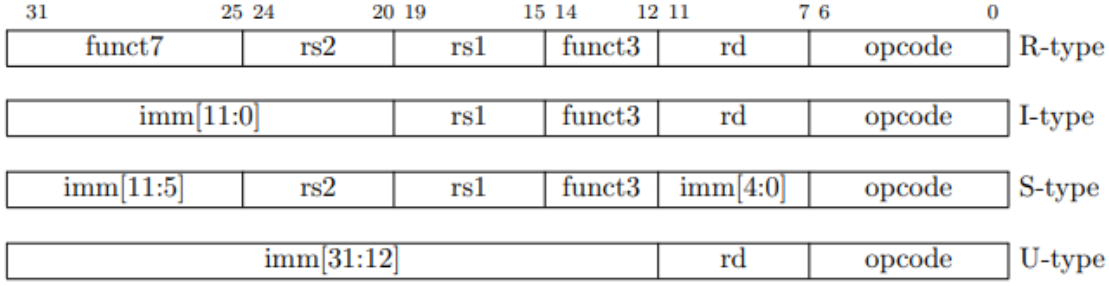
(1) 基本整数子集的程序员模型

寄存器$x0$是硬件连线的常数0。没有硬件连线的子程序返回地址连接寄存器，但是在一个过程调用中，标准软件调用约定使用寄存器$x1$来保存返回地址。对于$RV32$，其x寄存器是32位宽度的。还有一个额外的用户可见寄存器：程序计数器pc保存了当前指令的地址。



(2) 基本指令格式

在基本ISA中，有四种核心指令格式（$R/I/S/U$），如图所示。所有的指令都是固定32位长度的，并且在存储器中必须在4字节边界对齐。

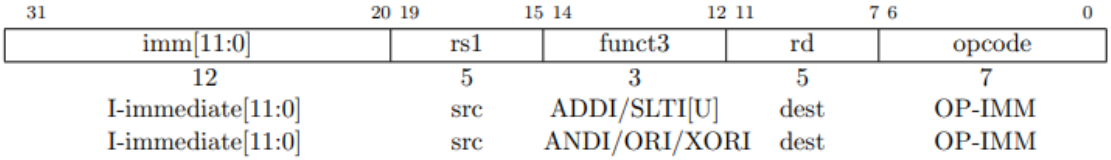


在所有格式中，$RISC-V\ ISA$将源寄存器（$rs1$和$rs2$）和目标寄存器（rd）固定在同样的位置，以简化指令译码。在指令中，立即数被打包，朝着最左边可用位的方向，并且是分配好的，以减少硬件复杂度。特别地，所有立即数的符号位总是在指令的第31位，以加速符号扩展电路。

(3) 整数计算指令

绝大多数整数计算指令对保存在整数寄存器中的$XLEN$位值进行操作。整数计算指令要么使用I类格式编码为寄存器-立即数操作，要么使用R类格式编码为寄存器-寄存器操作。对于寄存器-立即数指令和寄存器-寄存器指令，其目标都是寄存器rd。没有整数计算指令产生算术异常。

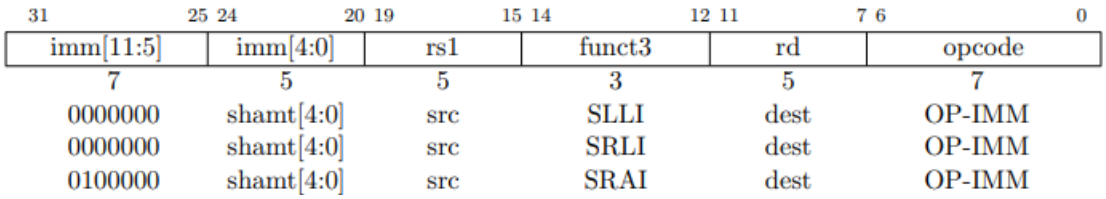
a) 整数寄存器-立即数指令



$ADDI$将符号扩展的12位立即数加到寄存器$rs1$上。算术溢出被忽略，而结果就是运算结果的低$XLEN$位。

$SLTI$（$set\ less\ than\ immediate$）将数值1放到寄存器$rd$中，如果寄存器$rs1$小于符号扩展的立即数（比较时，两者都作为有符号数），否则将0写入rd。$SLTIU$与之相似，但是将两者作为无符号数进行比较（也就是说，立即数被首先符号扩展为$XLEN$位，然后被作为一个无符号数）。注意，$SLTIU\ rd,rs1,1$将设置rd为1，如果$rs1$等于0，否则将rd设置为0。

$ANDI$、ORI、$XORI$是逻辑操作，在寄存器$rs1$和符号扩展的12位立即数上执行按位AND、OR、XOR操作，并把结果写入rd。注意，$XORI\ rd,rs1,-1$在$rs1$上执行一个按位取反操作。



被移位常数次，被编码为I类格式的特例。被移位的操作数放在$rs1$中，移位的次数被编码到I立即数字段的低5位。右移类型被编码到I立即数的一位高位。$SLLI$是逻辑左移（0被移入低位）；$SRLI$是逻辑右移（0被移入高位）；$SRAI$是算术右移（原来的符号位被复制到空出的高位中）。

31	12 11	7 6	0
imm[31:12]	rd	opcode	
20	5	7	
U-immediate[31:12]	dest	LUI	
U-immediate[31:12]	dest	AUIPC	

LUI（*load upper immediate*）用于构建32位常数，并使用*U*类格式。*LUI*将*U*立即数放到目标寄存器*rd*的高20位，将*rd*的低12位填0。

AUIPC（*add upper immediate to pc*）用于构建*pc*相对地址，并使用*U*类格式。*AUIPC*从20位*U*立即数构建一个32位偏移量，将其低12位填0，然后将这个偏移量加到*pc*上，最后将结果写入寄存器*rd*。

b) 整数寄存器-寄存器操作

*RV32I*定义了几种算术*R*类操作。所有操作都是读取*rs*1和*rs*2寄存器作为源操作数，并把结果写入到寄存器*rd*中。*funct*7和*funct*3字段选择了操作的类型。

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

*ADD*和*SUB*分别执行加法和减法。溢出被忽略，并且结果的低*XLEN*位被写入目标寄存器*rd*。*SLT*和*SLTU*分别执行符号数和无符号数的比较，如果$rs1 < rs2$，则将1写入*rd*，否则写入0。注意，*SLTU rd*,*x*0,*rs*2，如果*rs*2不等于0，则把1写入*rd*，否则将0写入*rd*。*AND*、*OR*、*XOR*执行按位逻辑操作。*SLL*、*SRL*、*SRA*分别执行逻辑左移、逻辑右移、算术右移，被移位的操作数是寄存器*rs*1，移位次数是寄存器*rs*2的低5位。

(4) **控制转移指令**

*RV32I*提供了两类控制转移指令：无条件跳转和条件分支。

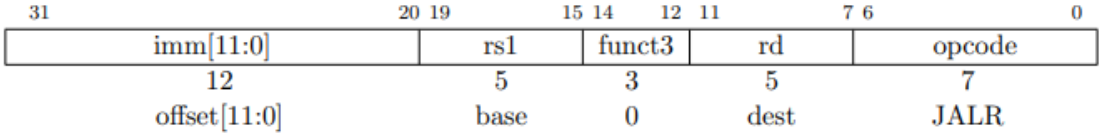
a) 无条件跳转

跳转并连接（*JAL*）指令使用了*UJ*类格式，此处*J*立即数编码了一个2的倍数的有符号偏移量。这个偏移量被符号扩展，加到*pc*上，形成跳转目标地址，跳转范围因此达到$\pm 1MB$。*JAL*将跳转指令后面指令的地址（$pc+4$）保存到寄存器*rd*中。标准软件调用约定使用*x*1来作为返回地址寄存器。

普通的无条件跳转指令（汇编语言伪指令*J*）被编码为$rd = x0$的*JAL*指令。

31	30 21	20	19 12	11 7	6 0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1	10	1	8	5	7
offset[20:1]				dest	JAL

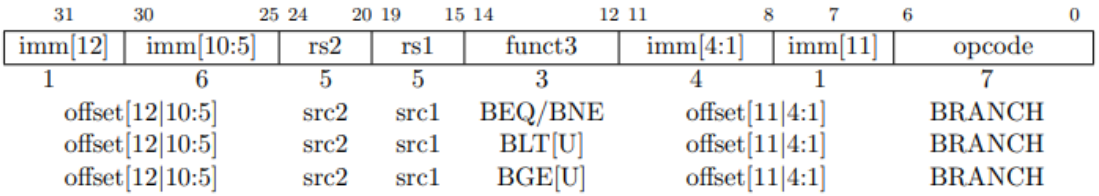
间接跳转指令*JALR*（*jump and link register*）使用*I*类编码。通过将12位有符号*I*类立即数加上*rs*1，然后将结果的最低位设置为0，作为目标地址。跳转指令后面指令的地址（$pc+4$）保存到寄存器*rd*中。如果不需要结果，则可以把*x*0作为目标寄存器。



31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	0	dest	JALR

b) 条件分支

所有分支指令使用SB类指令格式。12位B立即数编码了以2字节倍数的有符号偏移量，并被加到当前pc上，生成目标地址。条件分支范围是$\pm 4KB$。

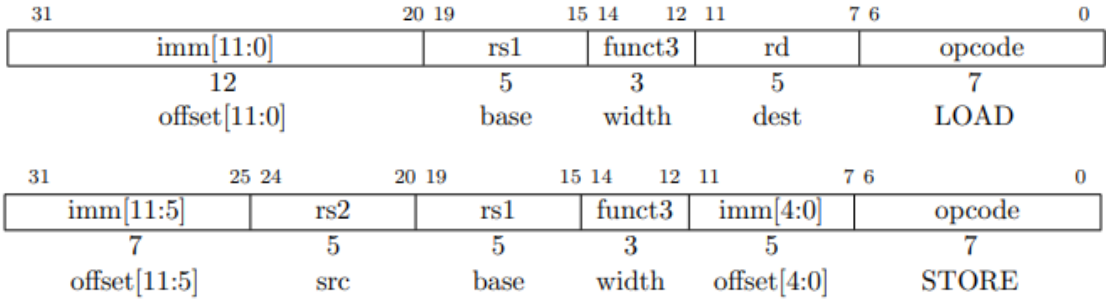


31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12|10:5]		src2	src1	BEQ/BNE	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BLT[U]	offset[11|4:1]		BRANCH
offset[12|10:5]		src2	src1	BGE[U]	offset[11|4:1]		BRANCH

分支指令比较两个寄存器。BEQ和BNE将跳转，如果$rs1$和$rs2$相等或者不相等。BLT和$BLTU$将跳转，如果$rs1$小于$rs2$，分别使用有符号数和无符号数进行比较。BGE和$BGEU$将跳转，如果$rs1$大于等于$rs2$，分别使用有符号数和无符号数进行比较。注意，BGT、$BGTU$、BLE和$BLEU$可以通过将BLT、$BLTU$、BGE、$BGEU$的操作数对调来实现。

(5) Load 和 store 指令

$RV32I$是一个$load-store$体系结构，也就是说，只有$load$和$store$指令可以访问存储器，而算术指令只在CPU寄存器上进行操作运算。



31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	width	dest	LOAD

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	width	offset[4:0]	STORE

$Load$和$store$指令在寄存器和存储器之间传输数值。$Load$指令编码为I类格式，而$store$指令编码为S类格式。有效字节地址是通过将寄存器$rs1$与符号扩展的12位偏移量相加而获得的。$Load$指令将存储器中的一个值复制到寄存器rd中。$Store$指令将寄存器$rs2$中的值复制到存储器中。

LW指令将一个32位数值从存储器复制到rd中。LH指令从存储器中读取一个16位数值，然后将其进行符号扩展到32位，再保存到rd中。LHU指令存储器中读取一个16位数值，然后将其进行零扩展到32位，再保存到rd中。对于8位数值，LB和LBU指令的定义与前面类似。SW、SH、SB指令分别将从$rs2$低位开始的32位、16位、8位数值保存到存储器中。

五、 实验目的

参照$RISCV$指令集设计带有除法指令的32位处理器，实现了I指令集中除$FENCE$，$ECALL$，$EBREAK$外的37条指令（实现部分指令的代码因面积限制被注释，注释后共14条，涉及各个类型）和M指令集中的4条除法指令，其中除法指令使用基于$Radix8-SRT$算法的，可完成有符号数和无符号数除法的32位除法器实现。该处理器具有5级流水线结构，能有效防止数

据冒险和控制冒险。同时使用$UART$完成指令，数据的接收和数据的发送。

六、 实验内容

1. 编写除法器并仿真验证。
2. 使用$Verilog$编写代码。
3. 使用VCS进行仿真。
4. 使用$FPGA$进行验证。

七、 实验器材（设备、元器件）

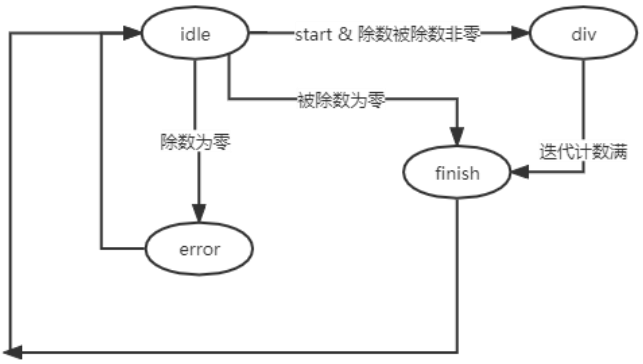
$Vivado$，VCS。

八、 实验步骤

1. 编写除法器并仿真验证

(1) 顶层控制

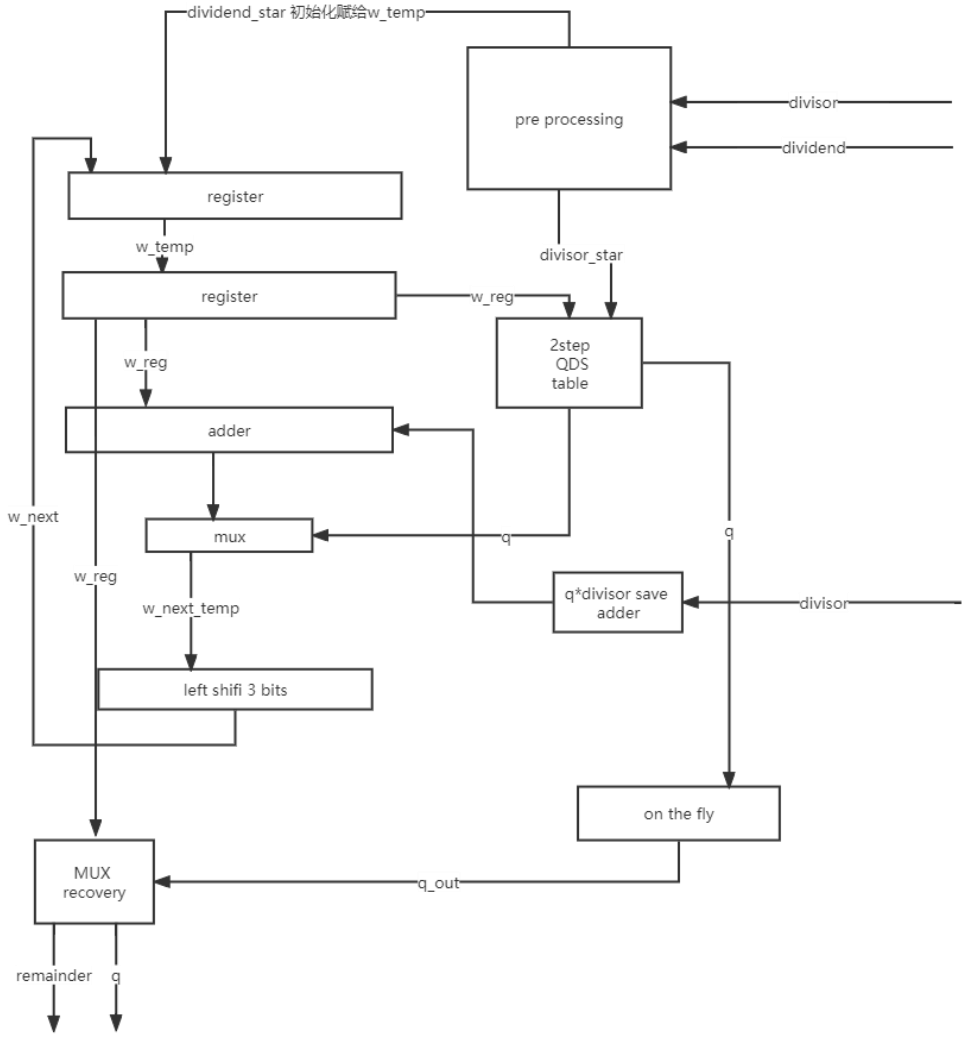
为了确保和$RISCV$控制模块正常交互，引入了一个简单的状态机用来控制除法器运算，如下图所示：



(2) 无符号除法结构框图

无符号除法整体的原理在前面已经进行了详细阐述，在此给出运算流程图。

主要模块有：预处理、qds 商选择查找表、部分余数迭代、on the fly 恢复算法，关键信号的流向已经在图中给出。



(3) 子模块介绍

a) 预处理模块

对于32bit的除法器来说，除数的有效位数限制了商的有效位数，同样限制了$srt8$的有效迭代次数，因此我们可以通过预处理进行除数移位以免去不必要的迭代，需要明确的是，对于srt算法来说，免去迭代即意味着提前对被除数（即初始的部分余数）进行$radix$阶数的提高（也同时意味着商选择为0，部分余数不变，对于第一次进入运算操作来说，是没有任何影响的）对于基8来说，即相当于移位3个二进制位，因此一共最少可有11种情况，为了规避极限情况可能的数据冲突，我们选择分为12种情况，对应迭代运算次数为0~11次。除两边极限情况以外，当除数为

$$32'b0000\_0000\_0000\_0000\_0000\_0000\_0000\_001x$$

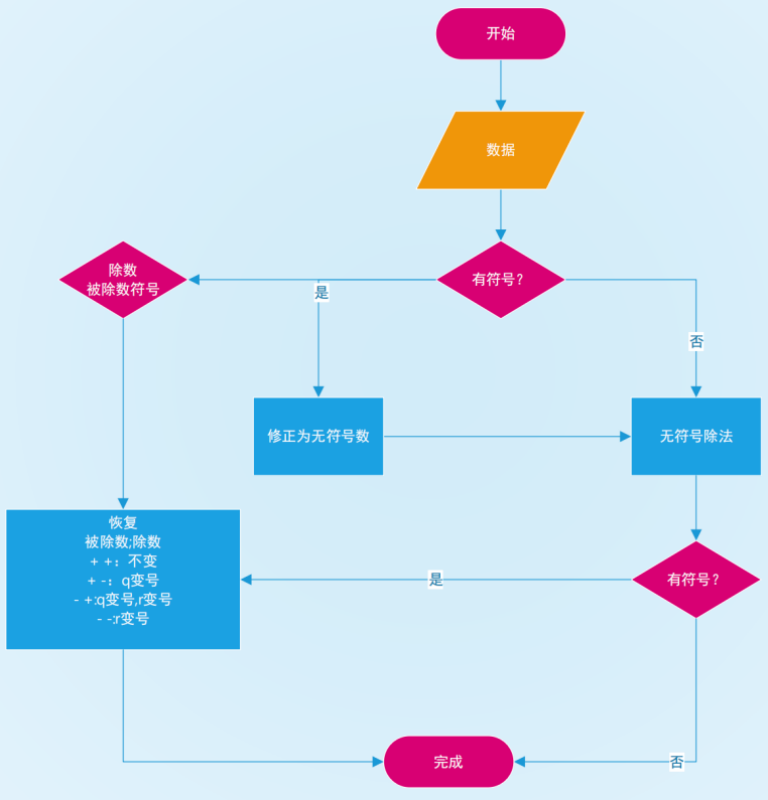
$$\sim 32'b01xx\_xxxx\_xxxx\_xxxx\_xxxx\_xxxx\_xxxx\_xxxx$$

时，分为10种初始的部分余数移位情况。除数则如前所说，正常只需要确保送入计算时和初始的部分余数的相对阶数差为整数即可恢复，被除数如前移位，则除数只需要确保把有效位移动到首位即可，此时便可通过移位对最终余数进行恢复。模块输出的迭代次数控制顶层状态机的div状态运行次数，恢复值控制最终的余数恢复，需要注意的是，为了减小延时，此模块为纯组合逻辑，因此输入的变量通过$start$信号进行锁存，确保相

关控制信号稳定。

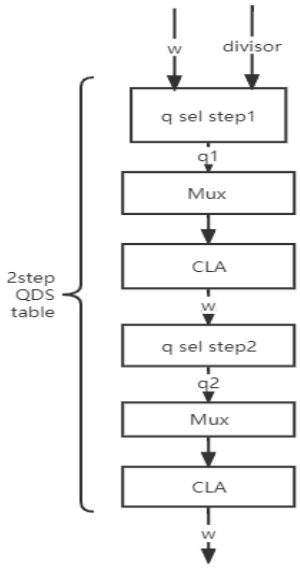
b) 有符号除法拓展

由于在状态机控制里，除法器的正常运行需要保证预移位处理模块输出结果正常，因此有一个维持一个时钟周期的初始化时间，对于预处理模块而言，其关键路径延迟是远远小于这个时钟的（事实上，通过仿真软件的模拟，这个时钟仅仅由高位宽加法器所决定），因此我们在此阶段引入有符号转化处理是不会进一步增加总的运算延迟的。结束阶段也是如此。有符号数转化无符号数以及相应的修正流程如下图所示：



c) 选商模块

作为SRT算法的核心，选商模块的基本结构框图如下，作为需要反复仿真优化的核心组合逻辑，两个$q\ sel$模块是重中之重。



(4) 测试与优化

a) 两种常见的双操作数加法器的优化原理

计算机算术发展到今天已经有几百年的历史了，作为基础算法的加法算法有了很多不同的方法，而加法器同样是SRT除法的基础，同时也是提高算法的速度和面积的一个关键部分。因此加法器设计是除法设计中关键性的一环，其算法的好坏决定了除法性能的优劣。

加法器的算法有很多，但是其总体分为两种，一个是双操作数加法器，一个是多操作数加法器。而一般的多操作数为了减小延迟，其均会通过算法，将其转化为两个操作数相加，然后使用双操作加法器来完成加法。先介绍双操作数的典型加法器。

A. 超前进位加法器CLA（$Carry\ Lookahead\ Adder$）

超前进位加法器是对普通的全加器进行改良而设计成的并行加法器，主要对普通全加器串联时互相进位产生的延迟进行了改良。

假设二进制加法器两个输入的第i位分别为A_i和B_i，其进位输入为C_i。那么就很容易得到超前进位加法器的本位和S_i与进位输出C_{i+1}：

$$\begin{cases} S_i = A_i \oplus B_i \oplus C_i \\ C_{i+1} = A_iB_i + A_iC_i + B_iC_i = A_iB_i + (A_i \oplus B_i)C_i \end{cases} \tag{1}$$

令$p_i = A_i \oplus B_i, g_i = A_iB_i$，那么上式可以写为：

$$\begin{cases} S_i = p_i \oplus C_i \\ C_{i+1} = g_i + p_iC_i \end{cases} \tag{2}$$

下面将式(2)进行一级一级的展开，来说明超前进位加法器的并行处理思想：

$$\begin{cases} C_1 = g_0 + p_0C_0 \\ C_2 = g_1 + p_1C_1 = g_1 + p_1g_0 + p_1p_0C_0 \\ C_3 = g_2 + p_2C_2 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0C_0 \end{cases} \tag{3}$$

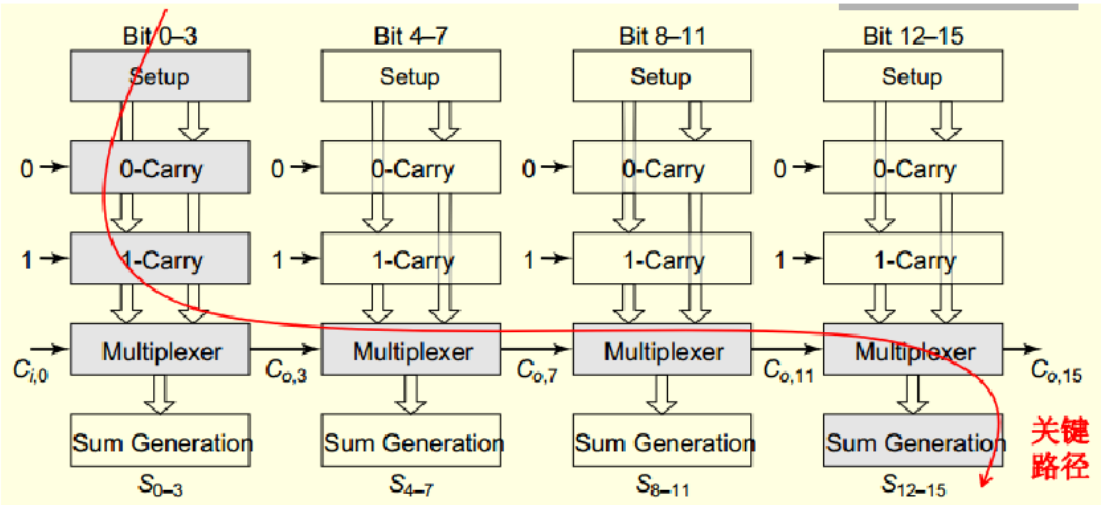
从上面的计算中可以看出，超前进位加法器有三步运算：第一步，计算每一位的g_i和p_i。第二步，根据式(3)，通过每一位的g_i和p_i计算出每一位的进位C_i。第三步，根据式(2)计算出最终每一位的本位和以及最高位的进位C_{i+1}。

从式(3)中可以看出，第一步和第三步很简单，但是第二步对于位宽比较大的数据来说，其延迟比较大，同时计算也比较复杂。

B. 进位选择加法器（$Carry-select\ Adder$）

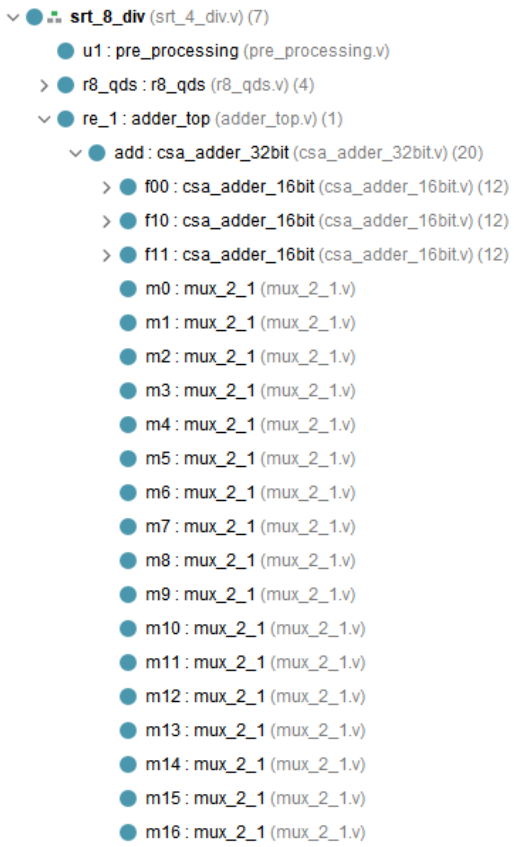
进位选择加法器是针对于位宽比较大的两个数进行相加，通过采用多个位宽小的加法器（一般选用全加器），构成一个位宽较大的加法器，来减小延迟。这是由于随着位宽的增大，CLA加法器的复杂度和产生的延迟近乎于指数的增加。采用多个小位宽可以很大程度上减小其延迟，增加速度。

由于CLA第二步复杂度很高，那么对于位宽大的加法器来说，假设其有N位位宽，以M位位宽的加法为一级，将N位加法器切割成N/M个M位的加法器。对于高位的$N/M-1$个加法器来说，开始其输入进位并没有被算出，但是对于二进制算术来说，其输入进位不是“0”，就是“1”。提前计算好输入进位为“0”和“1”的结果，等到输入进位进来，就可以直接选择器结果，而不需要重新计算。下图为以4个$4bit$加法器来构成16bit加法器的进位选择加法器。



b) 对于关键路径的延时的优化

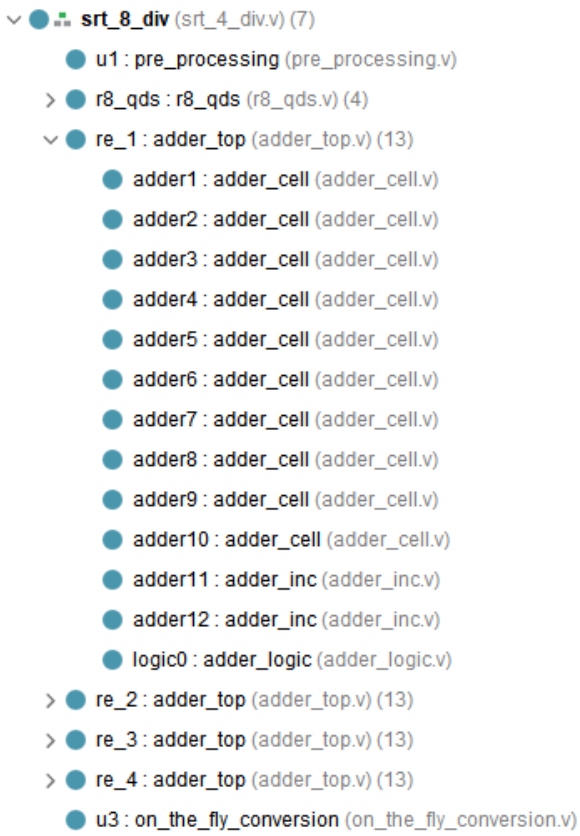
对于SRT除法中的加法单元，分别采用CLA、CSA、普通全加器进行比较。可以看到使用CSA的SRT8的大致结构如下图，对于内部加法器的实际算法不在此赘述。



经过$Vivado$的时序报告如下图，逻辑延迟已经用红色方框标出。由于$FPGA$布线和ICC自动布线的差异性，我们仅参考逻辑延迟。

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clo
Constrained Paths (1)														
CLK_50M (10)														
Path 11	5.634	19	18	43	divisor_reg_reg[5]/C	u3/qm_reg_reg[20]/D	14.231	3.948	10.283	27.7	72.3	20.000	CLK_50M	CLK_50M
Path 12	5.647	19	18	43	divisor_reg_reg[5]/C	u3/qm_reg_reg[21]/D	14.220	3.948	10.272	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 13	5.665	19	17	72	divisor_reg_reg[5]/C	w_reg_reg[8]/D	14.304	4.409	9.895	30.8	69.2	20.000	CLK_50M	CLK_50M
Path 14	5.665	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[28]/D	14.201	3.948	10.253	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 15	5.667	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[25]/D	14.197	3.948	10.249	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 16	5.673	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[24]/D	14.192	3.948	10.244	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 17	5.681	19	17	72	divisor_reg_reg[5]/C	w_reg_reg[9]/D	14.288	4.409	9.879	30.9	69.1	20.000	CLK_50M	CLK_50M
Path 18	5.702	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[23]/D	14.205	3.948	10.257	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 19	5.717	19	18	43	divisor_reg_reg[5]/C	u3/qm_reg_reg[18]/D	14.189	3.948	10.241	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 20	5.724	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[30]/D	14.141	3.948	10.193	27.9	72.1	20.000	CLK_50M	CLK_50M

之后我们又将CSA换为了CLA，结构如下：



可以看到，逻辑延迟明显下降。

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Excep
Constrained Paths (1)															
CLK_50M (10)															
Path 11	5.406	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[5]/D	14.321	3.777	10.544	26.4	73.6	20.000	CLK_50M	CLK_50M	
Path 12	5.480	20	18	82	divisor_reg_reg[8]/C	w_reg_reg[4]/D	14.291	3.646	10.645	25.5	74.5	20.000	CLK_50M	CLK_50M	
Path 13	5.498	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[7]/D	14.226	3.777	10.449	26.5	73.5	20.000	CLK_50M	CLK_50M	
Path 14	5.533	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[9]/D	14.195	3.777	10.418	26.6	73.4	20.000	CLK_50M	CLK_50M	
Path 15	5.584	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[16]/D	14.140	3.777	10.363	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 16	5.587	19	18	44	divisor_reg_reg[8]/C	u3/qm_reg_reg[27]/D	14.367	3.508	10.859	24.4	75.6	20.000	CLK_50M	CLK_50M	
Path 17	5.604	19	18	55	divisor_reg_reg[8]/C	u3/q_reg_reg[11]/D	14.350	3.508	10.842	24.4	75.6	20.000	CLK_50M	CLK_50M	
Path 18	5.613	20	18	82	divisor_reg_reg[8]/C	w_reg_reg[1]/D	14.116	3.646	10.470	25.8	74.2	20.000	CLK_50M	CLK_50M	
Path 19	5.620	19	18	59	divisor_reg_reg[8]/C	u3/q_reg_reg[27]/D	14.332	3.508	10.824	24.5	75.5	20.000	CLK_50M	CLK_50M	
Path 20	5.622	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[10]/D	14.105	3.777	10.328	26.8	73.2	20.000	CLK_50M	CLK_50M	

然而，尽管如此，无论是在$Vivado$还是LVS，采用逻辑综合软件自动综合出来的加法器，逻辑延迟反而最小，并且在DC中显示面积也相对较小，我们最终并未来得及查明理论和实际产生如此偏差结果的原因，但最后只得不加优化，采用最普通的加法。

$Vivado$关于使用普通加法器的除法器的时序报告如下：

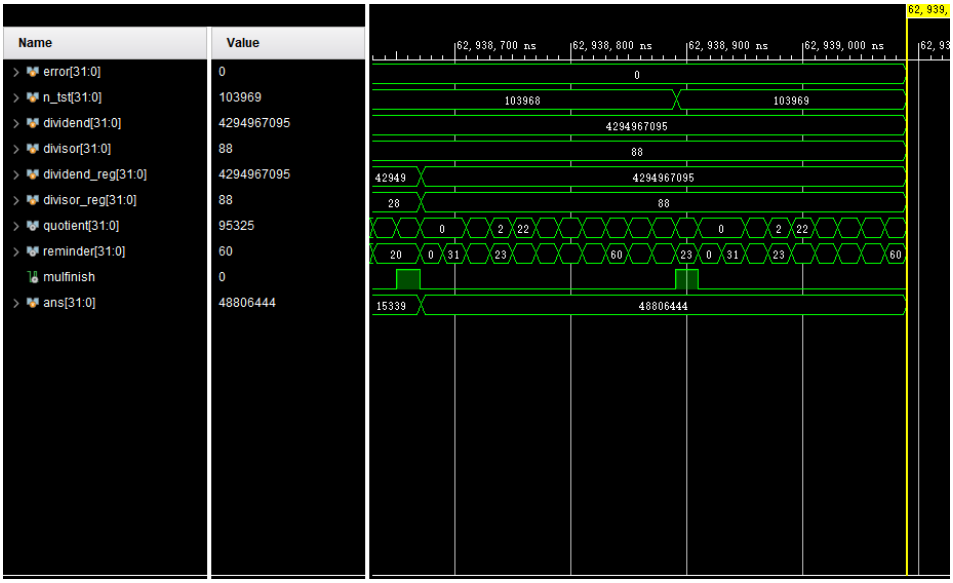
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exceptio
Constrained Paths (1)															
CLK_50M (10)															
Path 11	7.520	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[6]/D	12.502	3.270	9.232	26.2	73.8	20.000	CLK_50M	CLK_50M	
Path 12	7.704	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[18]/D	12.322	3.285	9.037	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 13	7.731	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[7]/D	12.292	3.285	9.007	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 14	7.767	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[11]/D	12.255	3.300	8.955	26.9	73.1	20.000	CLK_50M	CLK_50M	
Path 15	7.791	18	17	46	divisor_reg_reg[33]/C	w_reg_reg[21]/D	12.234	3.266	8.968	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 16	7.809	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[14]/D	12.215	3.285	8.930	26.9	73.1	20.000	CLK_50M	CLK_50M	
Path 17	7.813	18	17	46	divisor_reg_reg[33]/C	w_reg_reg[3]/D	12.251	2.997	9.254	24.5	75.5	20.000	CLK_50M	CLK_50M	
Path 18	7.826	19	18	81	divisor_reg_reg[33]/C	w_reg_reg[37]/D	11.923	3.409	8.514	28.6	71.4	20.000	CLK_50M	CLK_50M	
Path 19	7.830	19	17	81	divisor_reg_reg[33]/C	w_reg_reg[23]/D	12.197	3.417	8.780	28.0	72.0	20.000	CLK_50M	CLK_50M	
Path 20	7.833	19	17	81	divisor_reg_reg[33]/C	w_reg_reg[20]/D	12.193	3.417	8.776	28.0	72.0	20.000	CLK_50M	CLK_50M	

c) 对于计算正确率的测试与优化

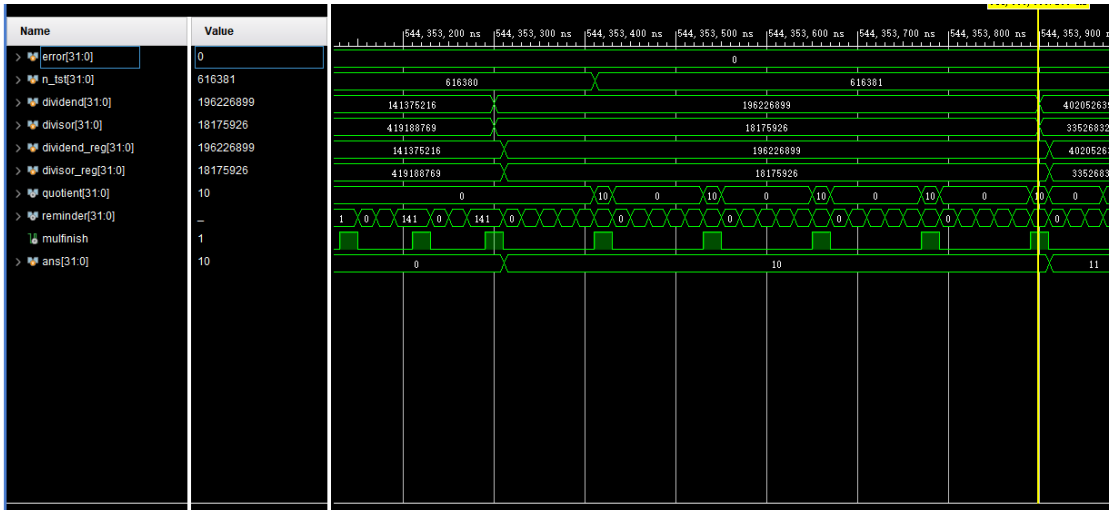
通过reg 32位测无符号，$integer$测有符号，定时更新测试数据寄存器，更新测试结果；通过在报告窗口打印出错时的$q1$和$q2$的值，方便修改优化qds表。

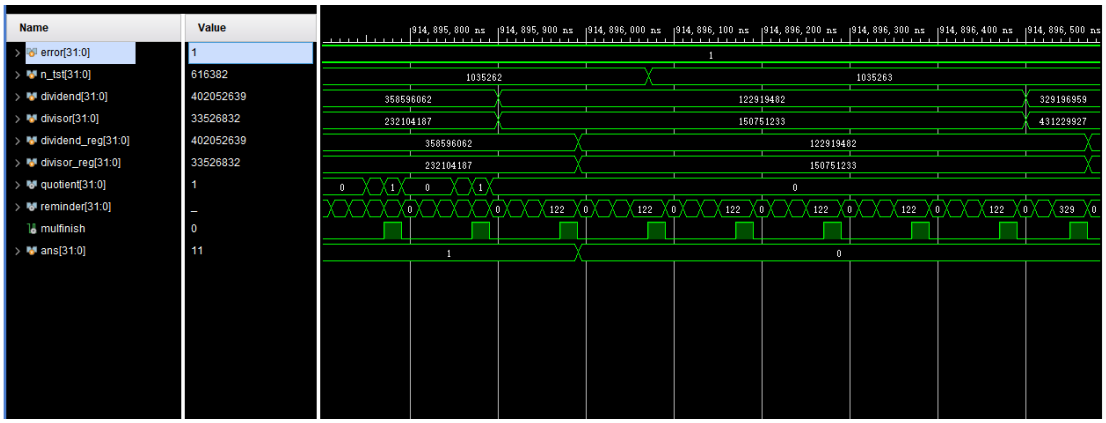
测试组	$dividend$	$divisor$	测试数量	正确率
A	$(\{\$random\}\%255) + 32'hffffff00$	$\{\$random\}\%255$	103,969	100%
B	$\{\$random\}\%4294967295$	$\{\$random\}\%4294967295$	1,668,484	99.99994%
C	$\{\$random\}\%4294967295$	$(\{\$random\}\%63) + 32'd960$	100,666	100%
D	$\{\$random\}\%4294967295$	$\{\$random\}\%4294967295$	2,141,102	100%

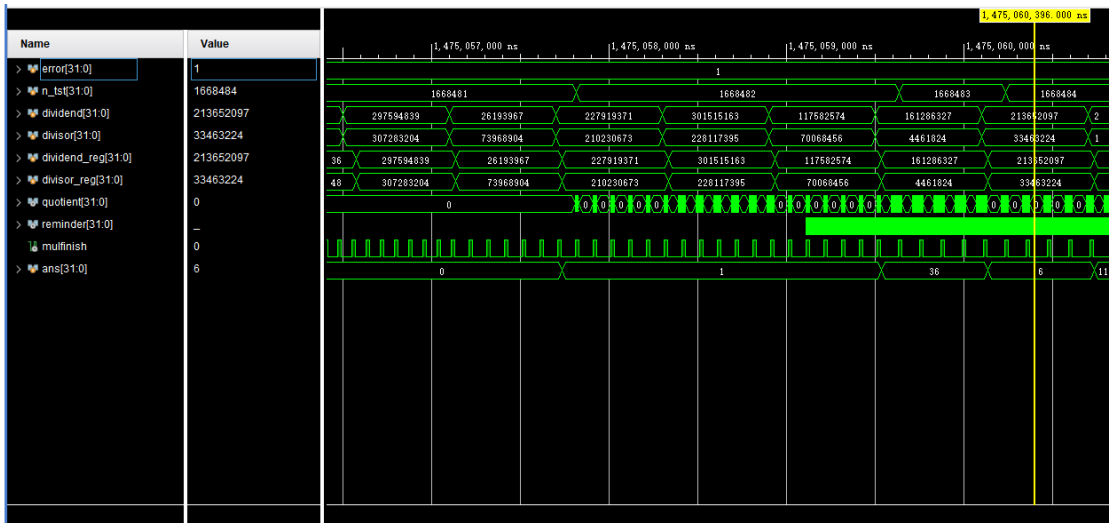
测试组A：测量除法器在被除数极大，除数极小情况下的正确率。



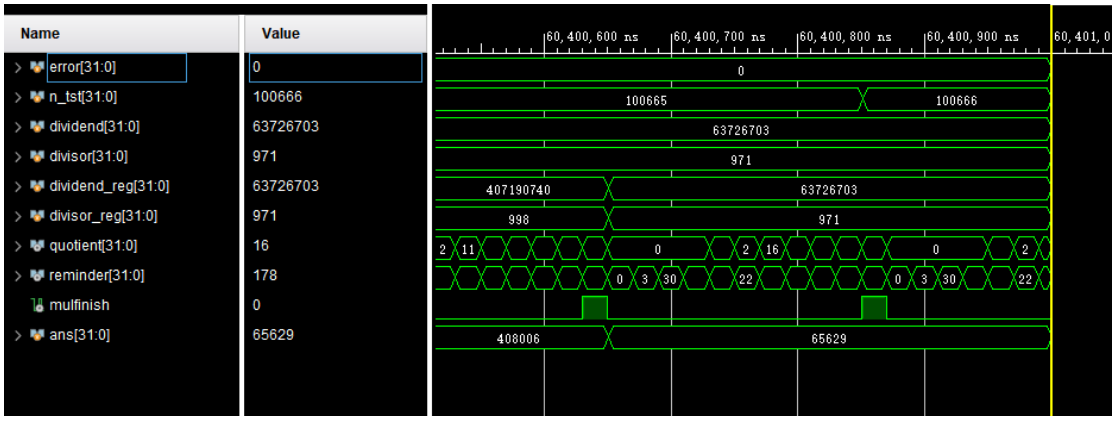
测试组B：测量除法器在完全随机的被除数与除数的情况下无符号运算的正确率。



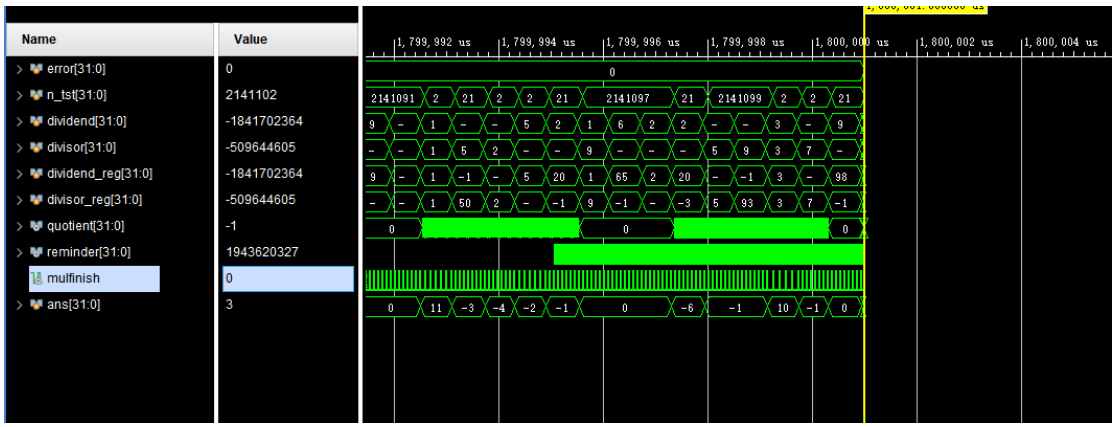




测试组C：测试被除数在确保首位连续4个1的情况下的正确率，之所以单独测这种情况是因为，对于SRT算法的qds选择来说，除数越大，则pd图对应的曲线的相对斜率越大，也因此对于qds选择的位宽精度要求就越大，在优化qds表提升正确率的大部分时间里，都需要在这种激励条件下进行测试。

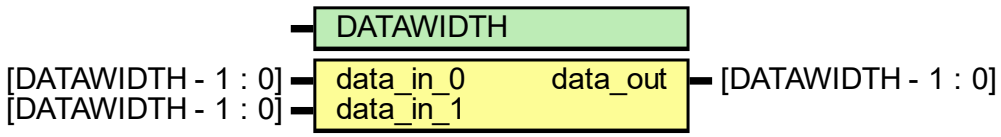


测试组D：测量除法器在完全随机的被除数与除数的情况下有符号运算的正确率。



2. 使用$Verilog$编写代码（实现部分指令的代码因面积限制被注释）

（1）$FRAG\_adder$



a） 参数

$DATAWIDTH$：输入数据和输出数据的数据宽度。

b） 端口

$data\_in\_0$：第1个输入数据。

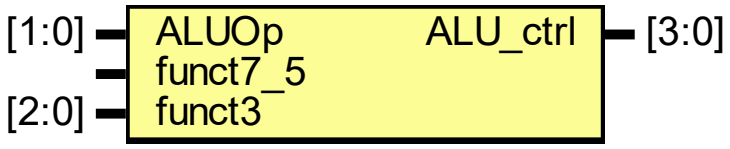
$data\_in\_1$：第2个输入数据。

$data\_out$：输出数据。

c） 作用

实现数据宽度为$DATAWIDTH$的2个数据的加法，且无进位输入与进位输出。

（2）$FRAG\_ALU\_ctrl$



a） 参数

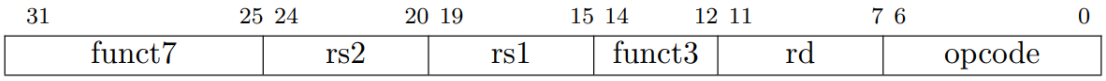
无。

b） 端口

$ALUOp$：$FRAG_ctrl$产生的ALU操作信号。

- 当$ALUOp = 2'b00$时，$FRAG_ALU$将执行加法操作。
- 当$ALUOp = 2'b01$时，$FRAG_ALU$将执行减法操作。
- 当$ALUOp = 2'b10$时，$FRAG_ALU$执行的操作将依据$funct7_5$和$funct3$确定。
- 当$ALUOp = 2'b11$时，$FRAG_ALU$将执行除法操作。

$funct7\_5$：指令内容中$funct7$的第5位。如图所示。



$funct3$：指令内容中的$funct3$。

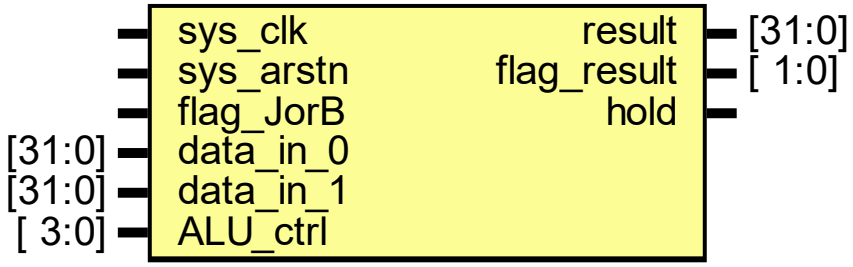
ALU_ctrl：ALU控制信号。

- 当$ALU_ctrl = 4'b0010$时，$FRAG_ALU$将执行加法操作。
- 当$ALU_ctrl = 4'b0110$时，$FRAG_ALU$将执行减法操作。
- 当$ALU_ctrl = 4'b1100$时，$FRAG_ALU$将执行有符号除法操作。
- 当$ALU_ctrl = 4'b1101$时，$FRAG_ALU$将执行无符号除法操作。
- 当$ALU_ctrl = 4'b1110$时，$FRAG_ALU$将执行有符号求余操作。
- 当$ALU_ctrl = 4'b1111$时，$FRAG_ALU$将执行无符号求余操作。

c） 作用

实现ALU控制信号的产生。

(3) $FRAG\_ALU$



a） 参数

无。

b） 端口

sys_clk：时钟信号。

sys_arstn：异步复位信号，低电平有效。

$flag\_JorB$：跳转或分支发生信号。

- 表示跳转或分支是否发生。

$data\_in\_0$：第1个输入数据。

$data\_in\_1$：第2个输入数据。

ALU_ctrl：ALU控制信号（前面已进行介绍，这里将不再赘述）。

$result$：操作结果。

$flag\_result$：操作结果标志位。

- $flag_result$的第1位表示$result = 0$是否成立。
- $flag_result$的第0位表示$result < 0$是否成立。

$hold$：暂停信号。

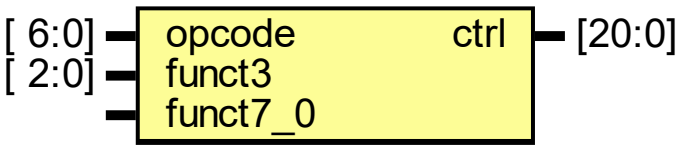
- 若$hold = 1'b1$，对$MODULE_if$，$MODULE_if_id$和$MODULE_id_ex$暂停，对$MODULE_ex_mem$冲刷，表示$FRAG_ALU$正在执行执行多周期除法和求余操作。

c） 作用

实现2个数据的加法、减法，除法和求余操作，且输出结果标志位。若执行多周期除法和求余操作，将产生信号对$MODULE_if$，

$MODULE_if_id$和$MODULE_id_ex$暂停，对$MODULE_ex_mem$冲刷。

(4) $FRAG\_ctrl$

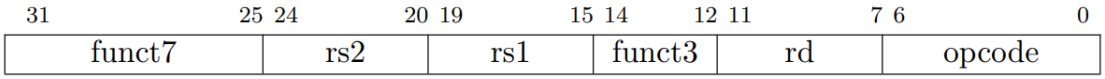


a) 参数

无。

b) 端口

$opcode$：指令内容中的$opcode$。如图所示。



$funct3$：指令内容中的$funct3$。

$funct7\_0$：指令内容中$funct7$的第0位。

$ctrl$：控制信号。

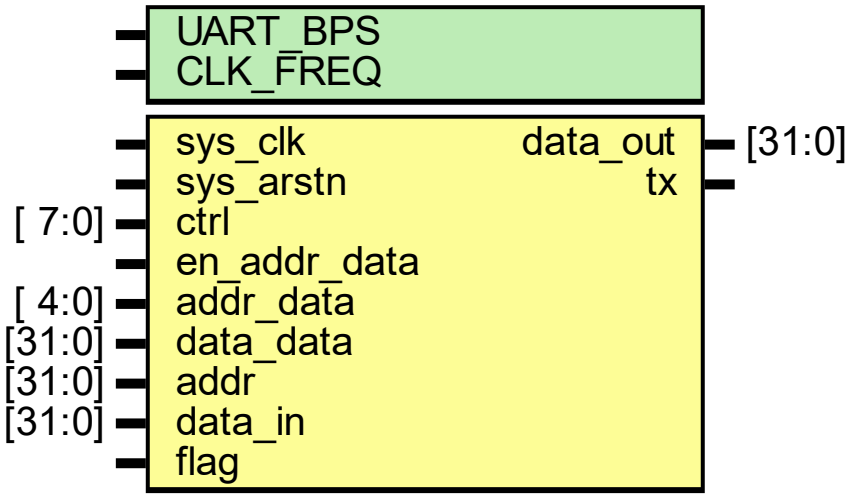
- $ctrl$的第20~19位为$ALUOp$（前面已进行介绍，这里将不再赘述）。
- $ctrl$的第18~17位为$ALUSrc$。
  - $ALUSrc$的第1位表示$FRAG_ALU$第1个输入数据的来源，若为$1'b0$，表示第1个输入数据是寄存器$Rs1$的数据，若为$1'b1$，表示第1个输入数据是$32'h0$。
  - $ALUSrc$的第0位表示$FRAG_ALU$第2个输入数据的来源，若为$1'b0$，表示第2个输入数据是寄存器$Rs2$的数据，若为$1'b1$，表示第2个输入数据是立即数。
- $ctrl$的第16~15位为$JumpBranch$。
  - $JumpBranch$的第1位表示指令是否为跳转指令。
  - $JumpBranch$的第0位表示指令是否为分支指令。
- $ctrl$的第14~12位为$BranchType$。
  - 表示分支指令的具体类型。
- $ctrl$的第11位为$MemRead$。
  - 表示指令是否为加载指令。
- $ctrl$的第10~8位为$LoadType$。
  - 表示加载指令的具体类型。
- $ctrl$的第7位为$MemWrite$。
  - 表示指令是否为贮存指令。
- $ctrl$的第6~4位为$StoreType$。
  - 表示贮存指令的具体类型。
- $ctrl$的第3~2位为$Inst$。
  - 若$Inst = 2'b00$，表示指令为其他指令。
  - 若$Inst = 2'b01$，表示指令为$AUIPC$。
  - 若$Inst = 2'b10$，表示指令为JAL。
  - 若$Inst = 2'b11$，表示指令为$JALR$。

- $ctrl$的第1位为$RegWrite$。
  - 表示是否需要对寄存器进行写操作。
- $ctrl$的第0位为$MemtoReg$。
  - 表示写入寄存器的数据来源是否为数据存储器。

c） 作用

实现控制信号的产生。

(5) $FRAG\_data\_mem$



a） 参数

$UART\_BPS$：$UART$的波特率。

CLK_FREQ：时钟频率。

b） 端口

sys_clk：时钟信号。

sys_arstn：异步复位信号，低电平有效。

$ctrl$：$FRAG_ctrl$输出的$ctrl$第11~4位，即$MemRead$、$LoadType$，$MemWrite$和$StoreType$（前面已进行介绍，这里将不再赘述）。

en_addr_data：数据初始化的使能信号。

$addr\_data$：数据初始化的地址。

$data\_data$：数据初始化的数据。

$addr$：数据地址。

$data\_in$：对数据存储器执行写操作的数据。

$flag$：运行结束信号。

- 若16条指令执行完毕，该信号将变为高电平，用于启动$UART$的数据发送端，发送数据存储器的全部数据。

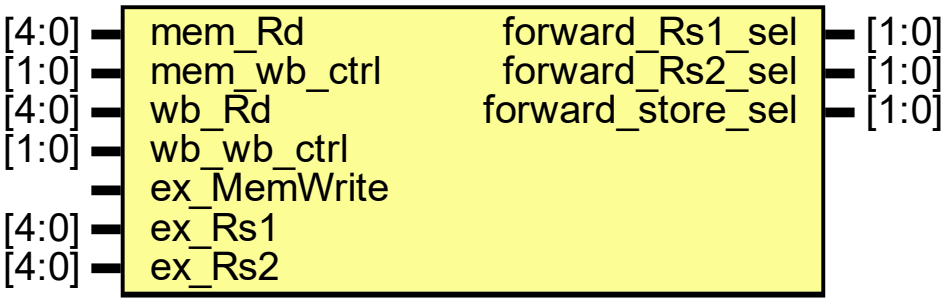
$data\_out$：对数据存储器执行读操作的数据。

tx：$UART$的数据发送端。

c） 作用

实现4个深度为8，宽度为8的数据存储器行为。若16条指令执行完毕，将启动$UART$的数据发送端，发送数据存储器的全部数据，共32字节。

(6) $FRAG\_forward$



a) 参数

无。

b) 端口

mem_Rd：访存阶段的Rd地址。

mem_wb_ctrl：访存阶段的$FRAG_ctrl$输出的$ctrl$第1~0位，即$RegWrite$和$MemtoReg$（前面已进行介绍，这里将不再赘述）。

wb_Rd：回写阶段的Rd地址。

wb_wb_ctrl：回写阶段的$FRAG_ctrl$输出的$ctrl$第1~0位，即$RegWrite$和$MemtoReg$（前面已进行介绍，这里将不再赘述）。

$ex\_MemWrite$：执行阶段的$FRAG_ctrl$输出的$ctrl$第7位，即$MemWrite$（前面已进行介绍，这里将不再赘述）。

ex_Rs1：执行阶段的$Rs1$地址。

ex_Rs2：执行阶段的$Rs2$地址。

$forward\_Rs1\_sel$：$Rs1$数据的选择信号。

- 若$forward_Rs1_sel = 2'b00$，表示$Rs1$数据为执行阶段的数据。
- 若$forward_Rs1_sel = 2'b01$，表示$Rs1$数据为访存阶段的数据。
- 若$forward_Rs1_sel = 2'b10$，表示$Rs1$数据为回写阶段的数据。

$forward\_Rs2\_sel$：$Rs2$数据的选择信号。

- 若$forward_Rs2_sel = 2'b00$，表示$Rs2$数据为执行阶段的数据。
- 若$forward_Rs2_sel = 2'b01$，表示$Rs2$数据为访存阶段的数据。
- 若$forward_Rs2_sel = 2'b10$，表示$Rs2$数据为回写阶段的数据。

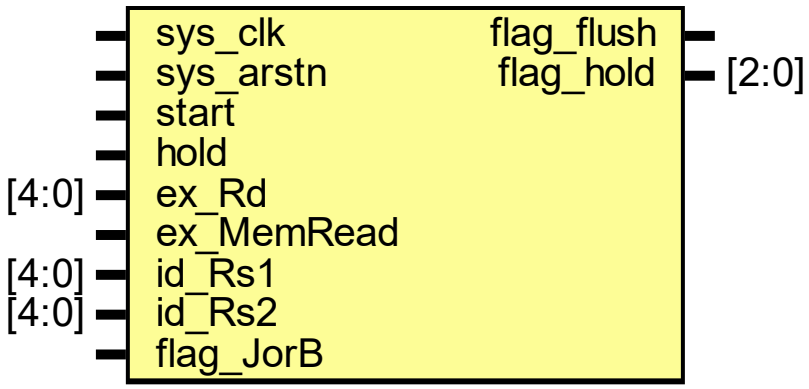
$forward\_store\_sel$：贮存数据的选择信号。

- 若$forward_store_sel = 2'b00$，表示贮存数据为执行阶段的数据。
- 若$forward_store_sel = 2'b01$，表示贮存数据为访存阶段的数据。
- 若$forward_store_sel = 2'b10$，表示贮存数据为回写阶段的数据。

c) 作用

通过产生数据选择信号，实现访存和回写阶段数据的前递，解决数据冒险。

(7) *FRAG_hazard*



a) 参数

无。

b) 端口

sys_clk：时钟信号。

sys_arstn：异步复位信号，低电平有效。

start：初始化完成的信号。

- 表示初始化是否完成。

hold：*FRAG_ALU*产生的*hold*信号（前面已进行介绍，这里将不再赘述）。

ex_Rd：执行阶段的*Rd*地址。

ex_MemRead：执行阶段的*FRAG_ctrl*输出的*ctrl*第11位，即*MemRead*（前面已进行介绍，这里将不再赘述）。

id_Rs1：译码阶段的*Rs1*地址。

id_Rs2：译码阶段的*Rs2*地址。

flag_JorB：跳转或分支发生信号（前面已进行介绍，这里将不再赘述）。

flag_flush：冲刷流水线信号。

- 若*flag_flush* = 1′*b*1，对*MODULE_if_id*，*MODULE_id_ex*和*MODULE_ex_mem*冲刷，表示跳转或分支发生。

flag_hold：暂停流水线信号。

- 若*flag_hold*的第2位为1′*b*1，对*MODULE_if*、*MODULE_if_id*、*MODULE_id_ex*，*MODULE_ex_mem*和*MODULE_mem_wb*暂停，表示初始化尚未完成。
- 若*flag_hold*的第1位为1′*b*1，对*MODULE_if*，*MODULE_if_id*和*MODULE_id_ex*暂停，对*MODULE_ex_mem*冲刷，表示*FRAG_ALU*正在执行多周期除法和求余操作。
- 若*flag_hold*的第0位为1′*b*1，对*MODULE_if*和*MODULE_if_id*暂停，对*MODULE_id_ex*冲刷，表示加载指令的下一条指令需要使用加载指令写入寄存器的数据。

c） 作用

通过暂停和冲刷，解决控制冒险。

(8) *FRAG_imm_gen*

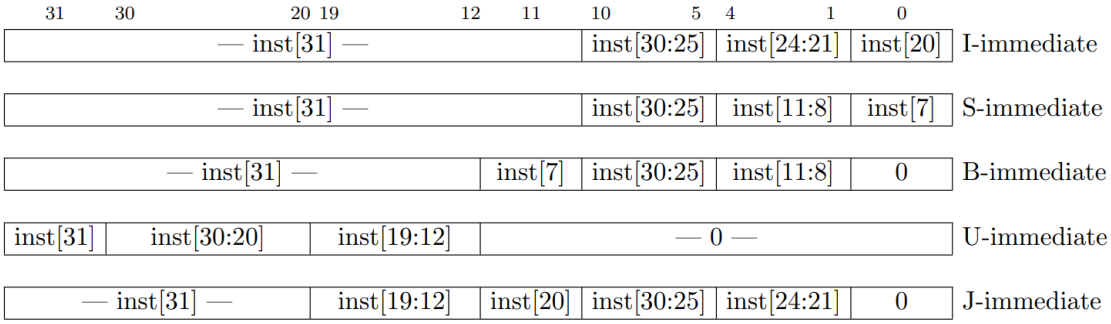
a） 参数

无。

b） 端口

inst_data：指令内容。

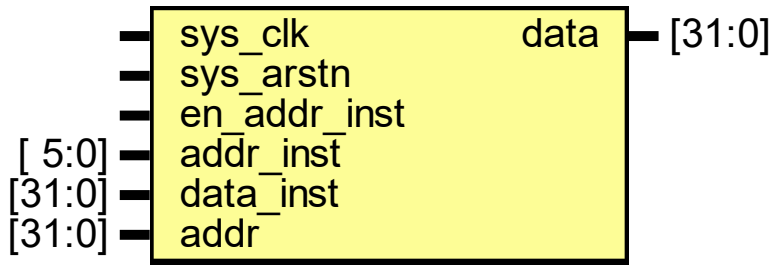
imm：立即数。如图所示。



c） 作用

实现立即数的产生。

(9) *FRAG_inst_mem*



a） 参数

无。

b） 端口

sys_clk：时钟信号。

sys_arstn：异步复位信号，低电平有效。

en_addr_inst：指令初始化的使能信号。

addr_inst：指令初始化的地址。

data_inst：指令初始化的数据。

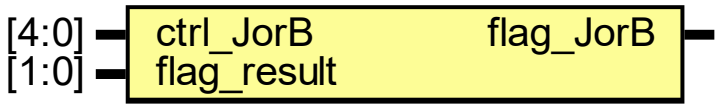
addr：指令地址。

data：指令内容。

c） 作用

实现深度为16，宽度为32的指令存储器行为。

(10) *FRAG_JorB_ctrl*



a) 参数

无。

b) 端口

$ctrl\_JorB$：$ctrl$的第16~12位，即$JumpBranch$和$BranchType$（前面已进行介绍，这里将不再赘述）。

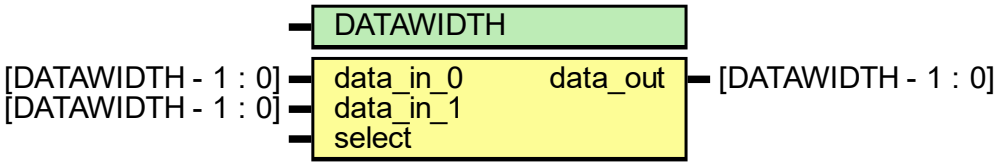
$flag\_result$：操作结果标志位（前面已进行介绍，这里将不再赘述）。

$flag\_JorB$：跳转或分支发生信号（前面已进行介绍，这里将不再赘述）。

c) 作用

实现跳转或分支发生信号的产生。

(11) $FRAG\_mux\_2$



a) 参数

$DATAWIDTH$：输入数据和输出数据的数据宽度。

b) 端口

$data\_in\_0$：第1个输入数据。

$data\_in\_1$：第2个输入数据。

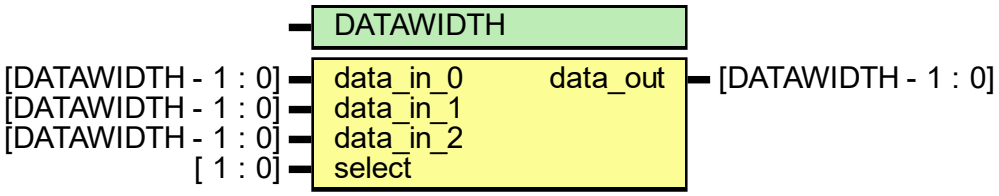
$select$：选择信号。

$data\_out$：输出数据。

c) 作用

实现数据宽度为$DATAWIDTH$的2个数据的选择。

(12) $FRAG\_mux\_3$



a) 参数

$DATAWIDTH$：输入数据和输出数据的数据宽度。

b) 端口

$data\_in\_0$：第1个输入数据。

$data\_in\_1$：第2个输入数据。

$data\_in\_2$：第3个输入数据。

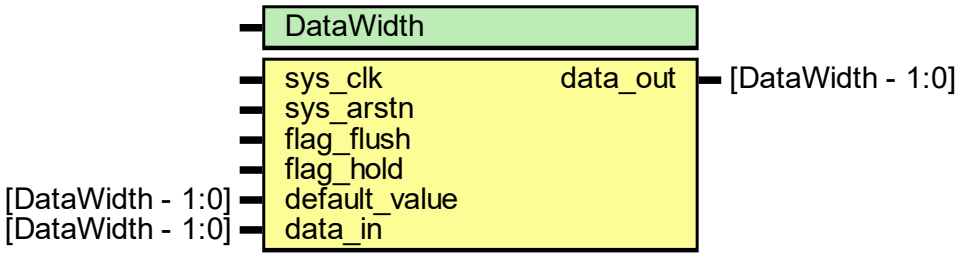
$select$：选择信号。

$data\_out$：输出数据。

c) 作用

实现数据宽度为$DATAWIDTH$的3个数据的选择。

(13) $FRAG\_pipline$



a) 参数

$DATAWIDTH$：输入数据和输出数据的数据宽度。

b) 端口

sys_clk：时钟信号。

sys_arstn：异步复位信号，低电平有效。

$flag\_flush$：冲刷流水线信号。

- 表示该流水线是否需要冲刷。

$flag\_hold$：暂停流水线信号。

- 表示该流水线是否需要暂停。

$default\_value$：冲刷流水线时的输出数据。

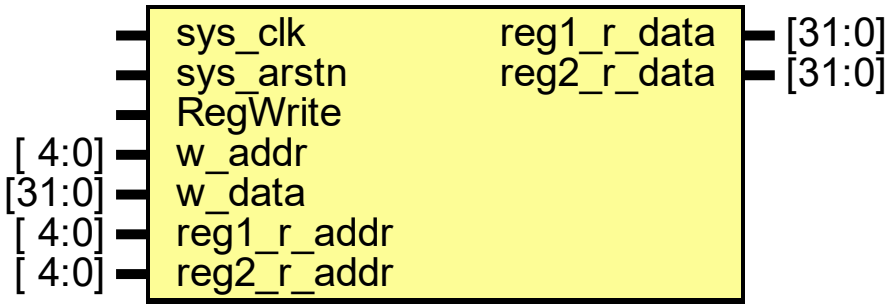
$data\_in$：输入数据。

$data\_out$：输出数据。

c) 作用

实现数据宽度为$DATAWIDTH$的数据的流水线行为。

(14) $FRAG\_register\_file$



a) 参数

无。

b) 端口

sys_clk：时钟信号。

sys_arstn：异步复位信号，低电平有效。

$RegWrite$：$FRAG_ctrl$输出的$ctrl$的第1位，即$RegWrite$（前面已进行介绍，这里将不再赘述）。

w_addr：对寄存器执行写操作的Rd地址。

w_data：对寄存器执行写操作的Rd数据。

$reg1\_r\_addr$：对寄存器执行读操作的$Rs1$地址。

$reg2\_r\_addr$：对寄存器执行读操作的$Rs2$地址。

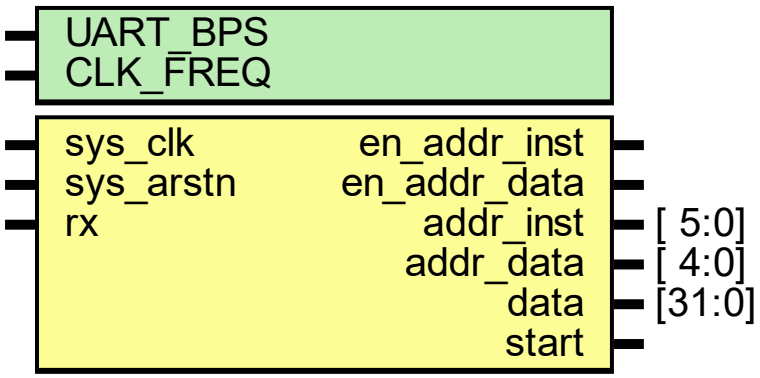
$reg1\_r\_data$：对寄存器执行读操作的$Rs1$数据。

$reg2\_r\_data$：对寄存器执行读操作的$Rs2$数据。

c） 作用

实现深度为20，宽度为32的寄存器组行为，其中寄存器组为写优先。

(15) $FRAG\_UART$



a） 参数

$UART\_BPS$：$UART$的波特率。

CLK_FREQ：时钟频率。

b） 端口

sys_clk：时钟信号。

sys_arstn：异步复位信号，低电平有效。

rx：$UART$的数据接收端。

en_addr_inst：指令初始化的使能信号。

en_addr_data：数据初始化的使能信号。

$addr\_inst$：指令初始化的地址。

$addr\_data$：数据初始化的地址。

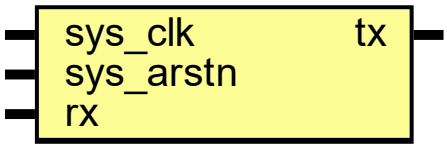
$data$：初始化的数据。

$start$：初始化完成的信号（前面已进行介绍，这里将不再赘述）。

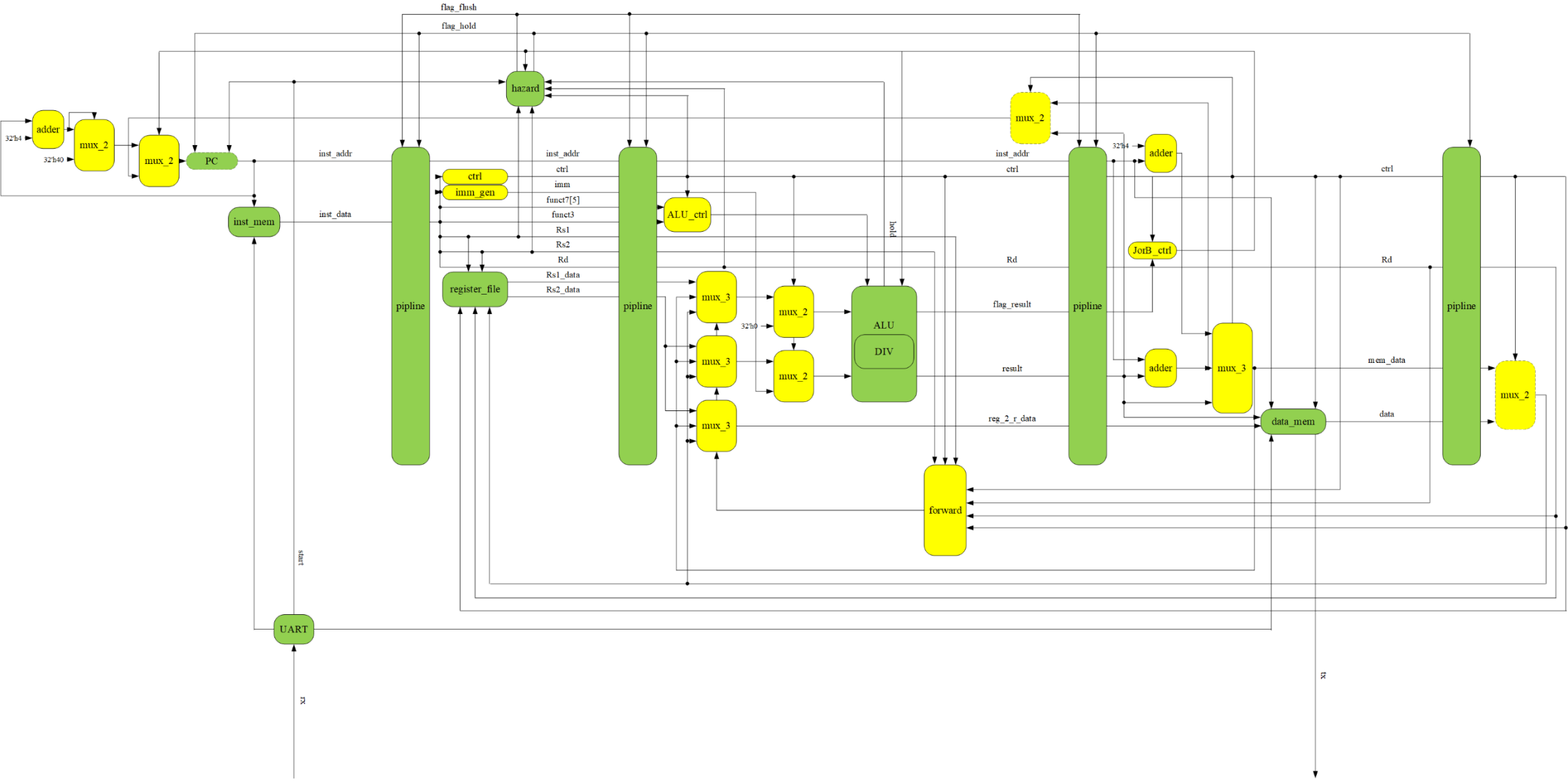
c） 作用

接收96字节，即16条数据宽度为32的指令和8条数据宽度为32的数据，之后，初始化指令存储器和数据存储器并产生初始化完成信号。

(16) TOP（$MODULE$不单独介绍）



整体框图如图所示。绿色部分表示时序逻辑，黄色部分表示组合逻辑。


flag_flush
flag_hold
hazard
adder
32'h4
mux_2
32'h40
mux_2
PC
inst_addr
inst_mem
inst_data
pipline
ctrl
imm_gen
register_file
inst_addr
ctrl
imm
funct7[5]
funct3
Rs1
Rs2
Rd
Rs1_data
Rs2_data
pipline
ALU_ctrl
mux_3
mux_3
mux_3
32'h0
mux_2
mux_2
ALU
DIV
hold
forward
mux_2
inst_addr
ctrl
Rd
flag_result
result
reg_2_r_data
pipline
32'h4
adder
JorB_ctrl
adder
mux_3
data_mem
ctrl
Rd
mem_data
data
pipline
mux_2
uart
UART
tx
rx

具体工作流程如下：

1) $UART$通过rx从上位机接收96字节，即16条数据宽度为32的指令和8条数据宽度为32的数据，初始化指令存储器和数据存储器，随后将$start$变为高电平，表示初始化完成。期间，流水线始终暂停，指令地址始终为$32'hffffffff$（指令存储器为全译码方式且其深度为16，故指令内容始终为$32'h0$，不会产生任何影响）。
2) 在取指阶段，PC寄存器通过流水线暂停信号$flag_hold$和跳转或分支发生信号$flag_JorB$输出当前时刻的指令地址，指令存储器接收指令地址并发送指令内容，指令地址和指令内容进入下一阶段。当前时刻的指令地址与$32'h4$相加得到下一时刻的指令地址。若所有指令执行完毕，指令地址将保持$32'h40$（指令存储器为全译码方式且其深度为16，故指令内容始终为$32'h0$，不会产生任何影响）。
3) 在译码阶段，指令内容被发送到各个模块，得到控制信号、立即数、寄存器地址，寄存器数据等，并将其传入下一阶段。
4) 在执行阶段，$ALUOp$进入ALU_ctrl得到ALU控制信号，$ALUSrc$选择进入ALU的两个数据，ALU输出运算结果和运算结果标志位，若执行多周期除法和求余操作，将产生信号对$MODULE_if$，$MODULE_if_id$和$MODULE_id_ex$暂停，对$MODULE_ex_mem$冲刷，以保证多周期除法和求余操作的正确执行。运算结果，运算结果标志位等进入下一阶段。
5) 在访存阶段，对于跳转或分支指令，$JorB_ctrl$根据控制信号和从执行阶段传入的运算结果标志位产生跳转或分支发生信号$flag_JorB$，跳转或分支地址由控制信号选择访存阶段运算结果得到；对于加载和贮存指令，$data_mem$内部产生片选信号，对4个深度为8，宽度为8的数据存储器进行读写操作，地址为从执行阶段传入的运算结果；对于$AUIPC$，JAL和$JALR$指令，控制信号选择访存阶段运算结果得到将传入回写阶段的数据。访存阶段运算结果，对数据存储器执行读操作得到的数据等进入下一阶段。
6) 在回写阶段，控制信号选择得到回写数据，控制信号，回写地址和回写数据进入寄存器组进行回写（寄存器组是写优先）。
7) 若访存阶段的指令地址$32'h40$，即所有指令执行完毕，$data_mem$通过tx发送数据存储器的全部数据，共32字节。

数据冒险和控制冒险是两个关键问题。

对于数据冒险，如图所示，$sub\ x2,x1,x3$将执行阶段的运算结果写入$x2$，但在这之前，$and\ x12,x2,x5$需要在执行阶段使用$x2$的数据。若不进行处理，将使得$and\ x12,x2,x5$在执行阶段使用的$x2$数据为$sub\ x2,x1,x3$执行之前的$x2$数据，这将导致错误发生。若将流水线暂停，以等待$sub\ x2,x1,x3$执行，将降低运行速度。

```
sub x2, x1, x3   // Register z2 written by sub
and x12, x2, x5   // 1st operand(x2) depends on sub
or  x13, x6, x2   // 2nd operand(x2) depends on sub
add x14, x2, x2   // 1st(x2) & 2nd(x2) depend on sub
sd   x15, 100(x2)  // Base (x2) depends on sub
```

因此，$FRAG_forward.v$中的进程根据控制信号对执行阶段的$Rs1$地址和$Rs2$地址与访存阶段的Rd地址和回写阶段的Rd地址进行了比对，产生选择信号。选择信号进入执行阶段的多路复用器中，对传入执行阶段的数据，访存阶段的运算结果和回写阶段的会写数据进行选择，得到正确的寄存器数据。这样就解决了数据冒险。

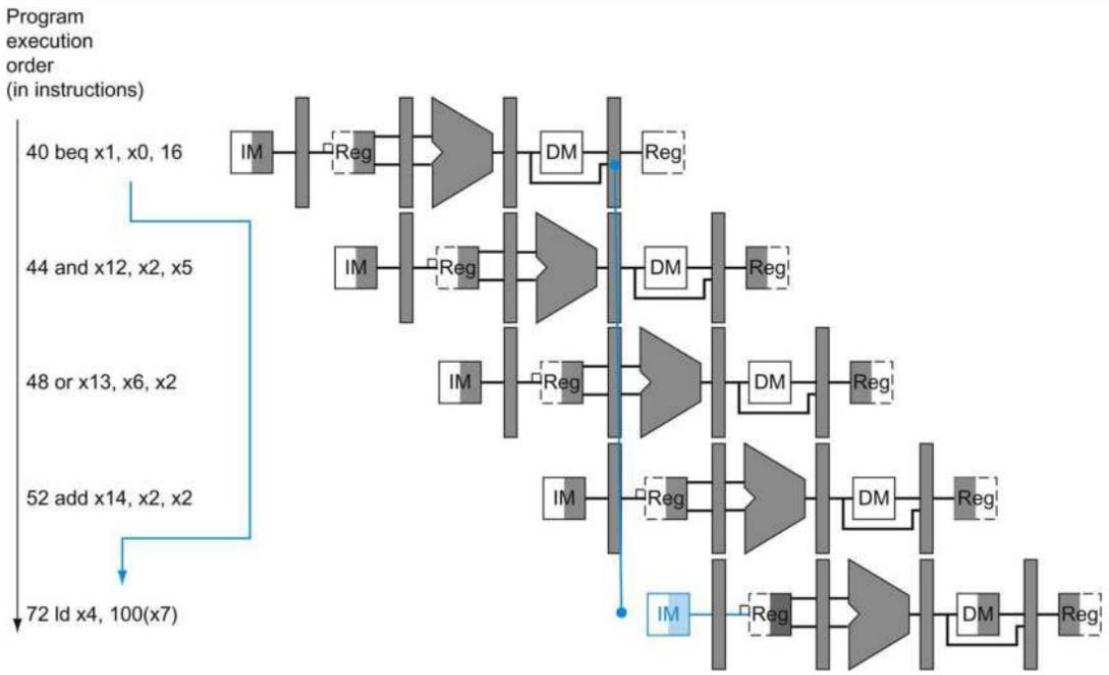
```
    always @(*) begin
        if((mem_wb_ctrl[1] == 1'b1) & (mem_Rd != 5'b0) & (mem_Rd == ex_Rs1)) begin
            forward_Rs1_sel = 2'b01;
        end else if((wb_wb_ctrl[1] == 1'b1) & (wb_Rd != 5'b0) & (wb_Rd == ex_Rs1)) begin
            forward_Rs1_sel = 2'b10;
        end else begin
            forward_Rs1_sel = 2'b00;
        end
    end

    always @(*) begin
        if((mem_wb_ctrl[1] == 1'b1) & (mem_Rd != 5'b0) & (mem_Rd == ex_Rs2)) begin
            forward_Rs2_sel = 2'b01;
        end else if((wb_wb_ctrl[1] == 1'b1) & (wb_Rd != 5'b0) & (wb_Rd == ex_Rs2)) begin
            forward_Rs2_sel = 2'b10;
        end else begin
            forward_Rs2_sel = 2'b00;
        end
    end

    always @(*) begin
        if((mem_wb_ctrl[1] == 1'b1) & (ex_MemWrite == 1'b1) & (mem_Rd != 5'b0) & (mem_Rd ==
ex_Rs2)) begin
            forward_store_sel = 2'b01;
        end else if((wb_wb_ctrl[1] == 1'b1) & (ex_MemWrite == 1'b1) & (wb_Rd != 5'b0) & (wb_Rd ==
ex_Rs2)) begin
            forward_store_sel = 2'b10;
        end else begin
            forward_store_sel = 2'b00;
        end
    end
```

对于控制冒险，共有以下3种情况：

1) 如图所示，在执行$beq\ x1,x0,x16$时，在判断跳转或分支是否发生前，处理器将执行可能不应被执行的指令。因此，处理器遇到跳转或分支指令时，默认跳转或分支不发生，若发生，$MODULE_if$将根据跳转或分支发生信号$flag_JorB$和跳转或分支地址得到指令地址，$MODULE\_if\_id$，$MODULE_id_ex$和$MODULE_ex_mem$将被冲刷，以取消不应被执行的指令。



2) 在执行多周期除法和求余操作时，对$MODULE_if$，$MODULE_if_id$和$MODULE_id_ex$暂停，对$MODULE_ex_mem$冲刷，以保证多周期除法和求余操作正确执行。

3) 若加载指令的下一条指令需要使用加载指令写入寄存器的数据，需要对$MODULE_if$和$MODULE_if_id$暂停，对$MODULE_id_ex$冲刷，以保证下一条指令的正确执行。

3. 使用VCS进行仿真

在$TB.v$中，先通过rx向处理器发送96字节，即初始化指令存储器和数据存储器，所有指令执行完毕后，再通过tx接收32字节，即接收数据存储器的所有数据，最后与预期结果进行比对，若数据存储器内的数据与预期结果一致，则测试成功，否则，测试失败。

$TB.v$包括以下7种情况：

(1) 指令：$addi\ x1,x1,1$，$addi\ x1,x1,1$，$addi\ x1,x1,1$，$sw\ x1,0(x0)$。
数据：0，0，0，0，0，0，0，0。
预期结果：3，0，0，0，0，0，0，0。

(2) 指令：$addi\ x1,x1,1$，$addi\ x1,x1,1$，$jal\ x2,1000$，$addi\ x1,x1,1$，$sw\ x1,0(x0)$，$sw\ x2,4(x0)$。
数据：0，0，0，0，0，0，0，0。
预期结果：2，12，0，0，0，0，0，0。

(3) 指令：$addi\ x1,x1,1$，$sw\ x1,0(x0)$，$lw\ x1,0(x0)$，$addi\ x1,x1,1$，$sw\ x1,0(x0)$。
数据：0，0，0，0，0，0，0，0。
预期结果：2，0，0，0，0，0，0，0。

(4) 指令：$lw\ x1,0(x0)$，$lw\ x2,4(x0)$，$remu\ x3,x2,x1$，$sw\ x3,8(x0)$，$remu\ x4,x1,x2$，$sw\ x4,12(x0)$。
数据：69，10，0，0，0，0，0，0。
其预期结果为69，10，10，9，0，0，0，0。

(5) 指令：$lw\ x1,0(x0)$，$lw\ x2,4(x0)$，$rem\ x3,x2,x1$，$sw\ x3,8(x0)$，$rem\ x4,x1,x2$，$sw\ x4,12(x0)$。

数据：69，10，0，0，0，0，0，0。

其预期结果为69，10，10，9，0，0，0，0。

(6) 指令：$lw\ x1,0(x0)$，$lw\ x2,4(x0)$，$divu\ x3,x2,x1$，$sw\ x3,8(x0)$，$divu\ x4,x1,x2$，$sw\ x4,12(x0)$。

数据：69，10，0，0，0，0，0，0。

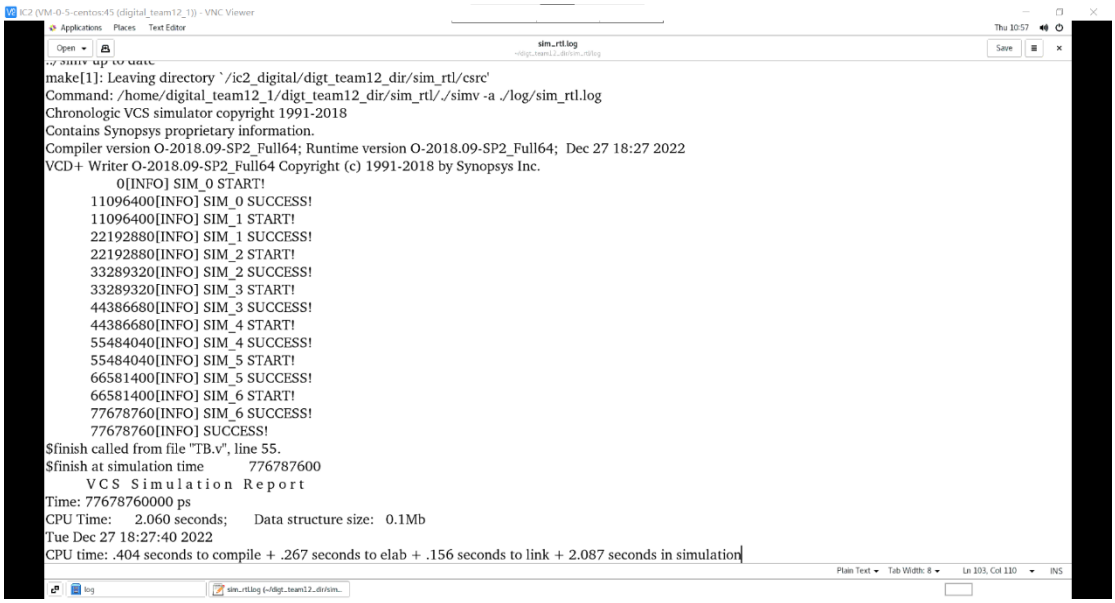
其预期结果为69，10，0，6，0，0，0，0。

(7) 指令：$lw\ x1,0(x0)$，$lw\ x2,4(x0)$，$div\ x3,x2,x1$，$sw\ x3,8(x0)$，$div\ x4,x1,x2$，$sw\ x4,12(x0)$。

数据：69，10，0，0，0，0，0，0。

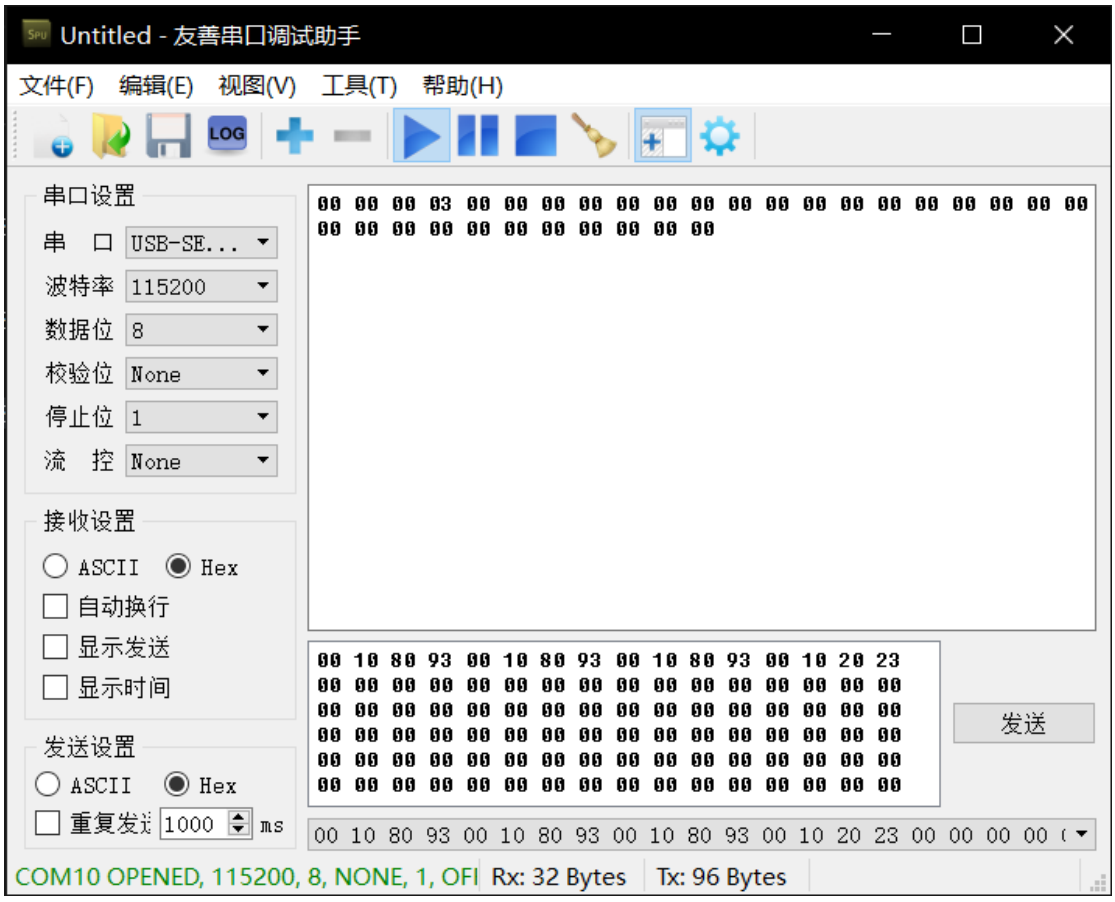
其预期结果为69，10，0，6，0，0，0，0。

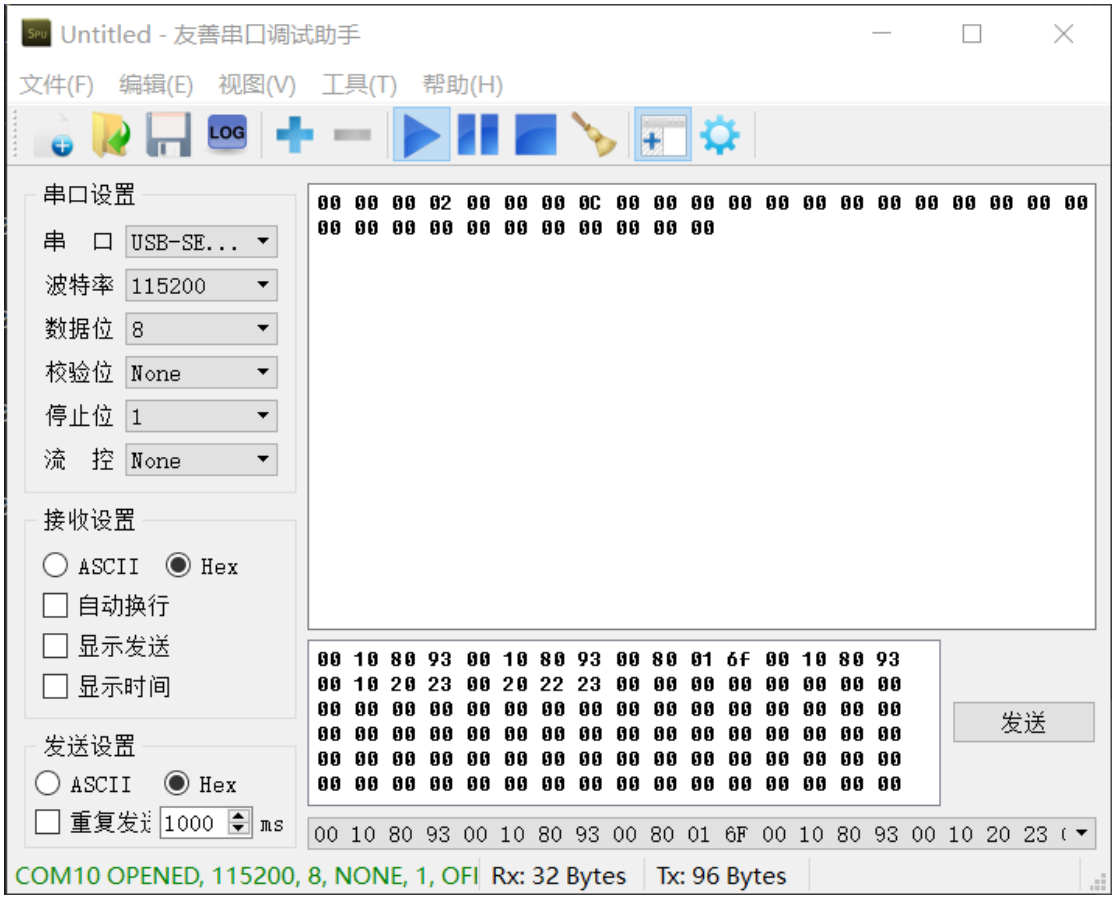
测试结果如图所示。

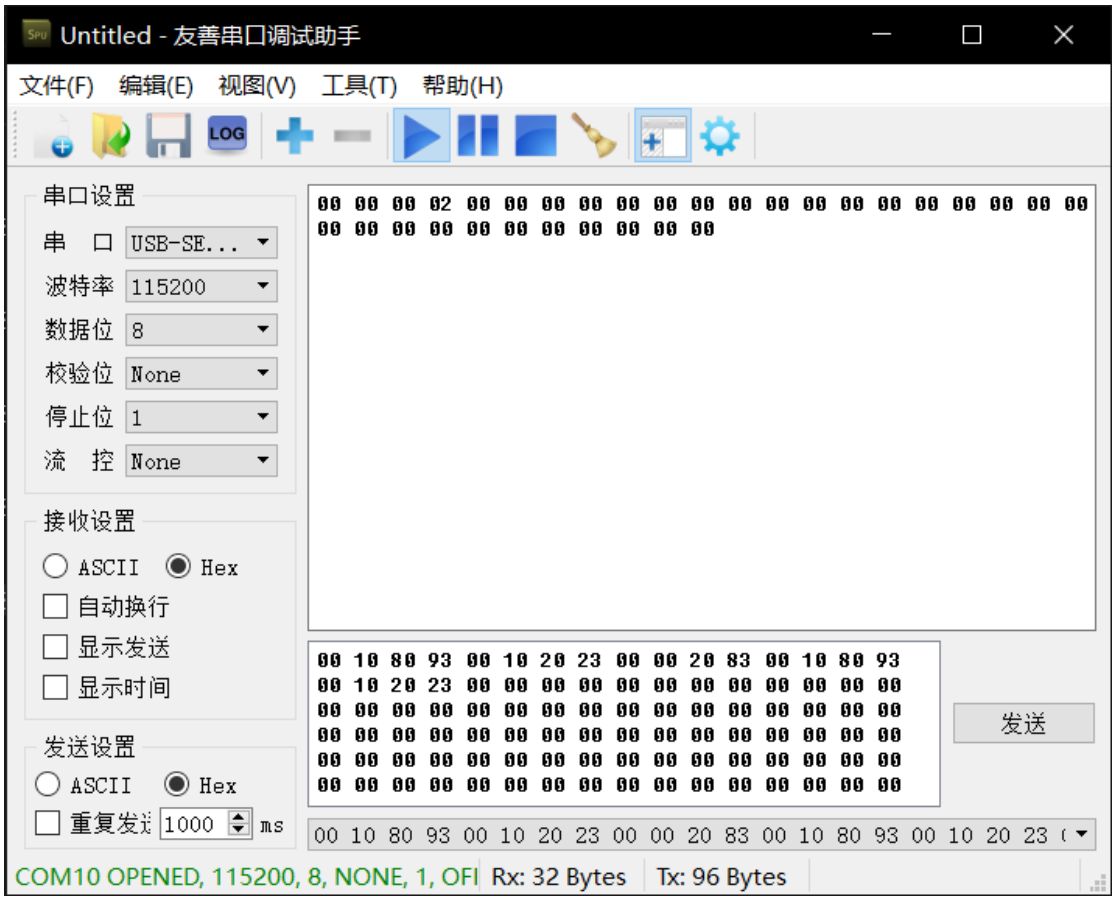


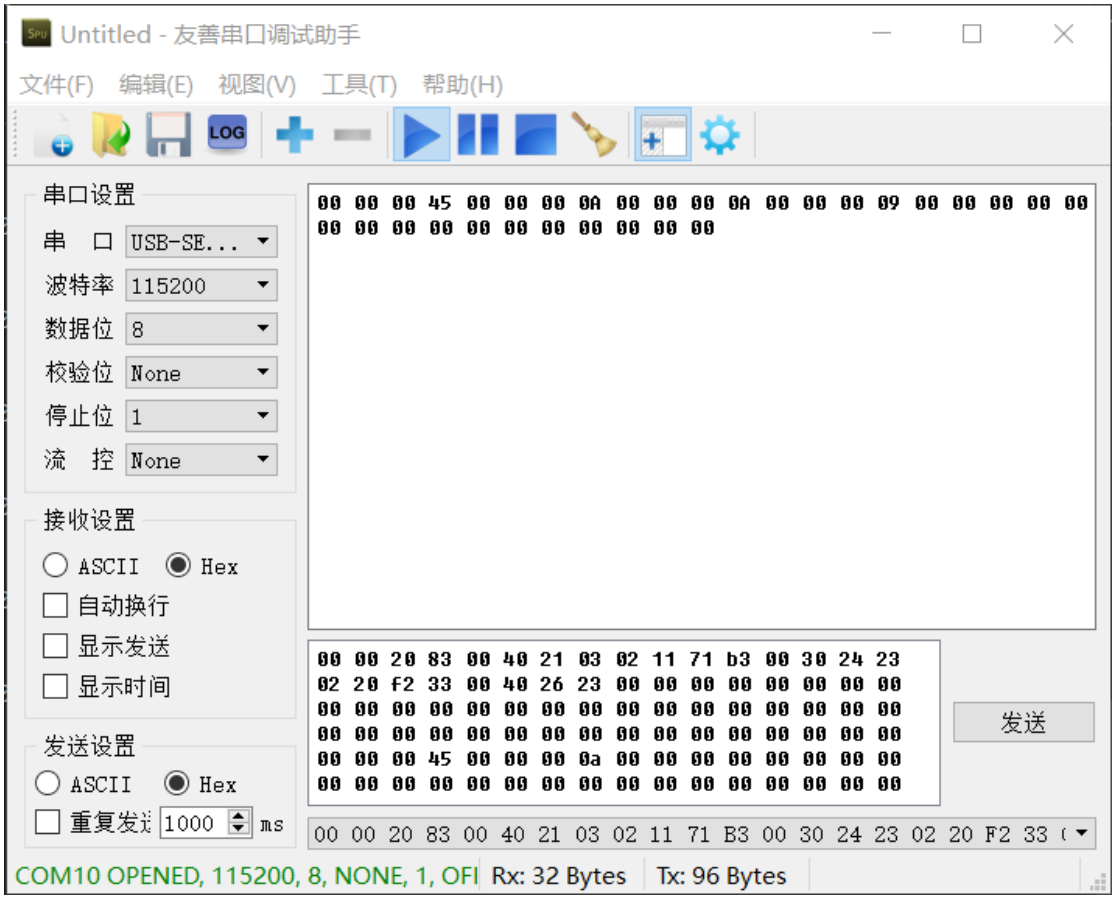
4. 使用$FPGA$进行验证

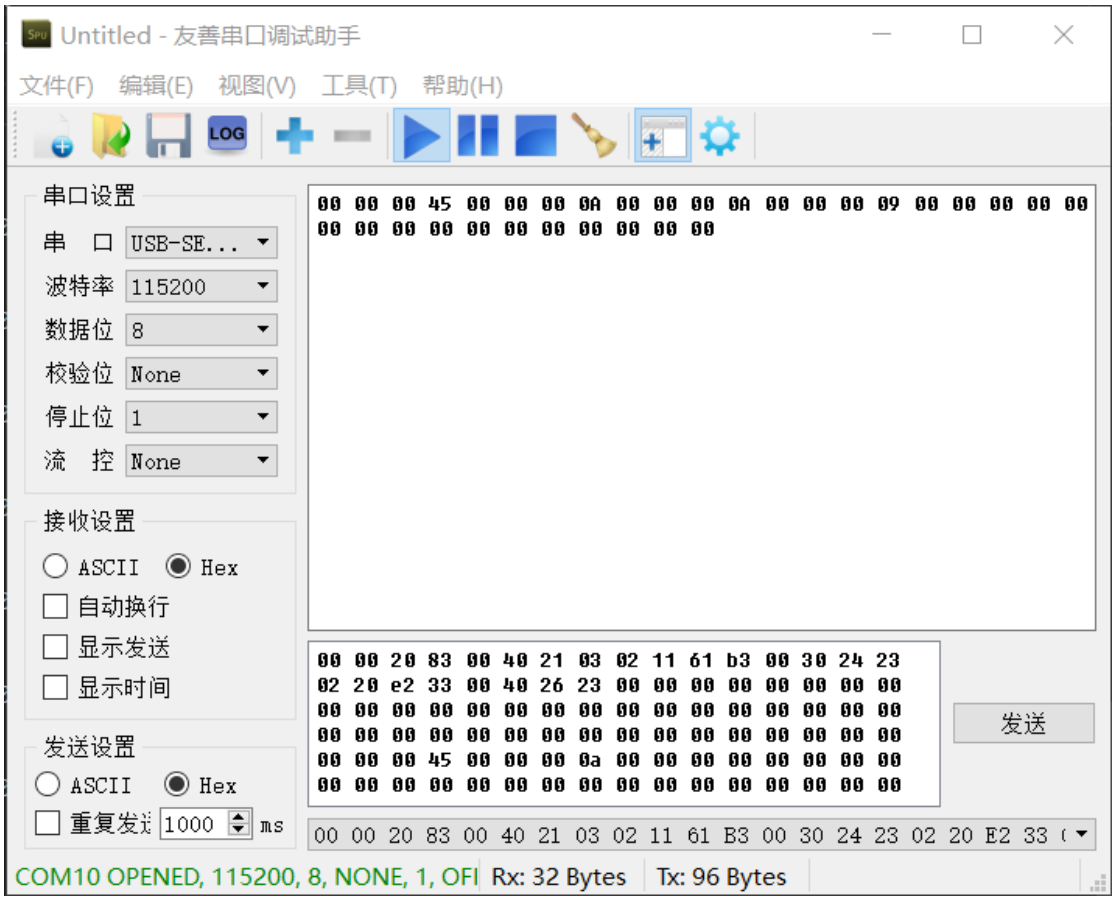
使用$FPGA$对$TB.v$中的7种情况进行了测试，测试结果如图所示。观察到7种情况全部通过。

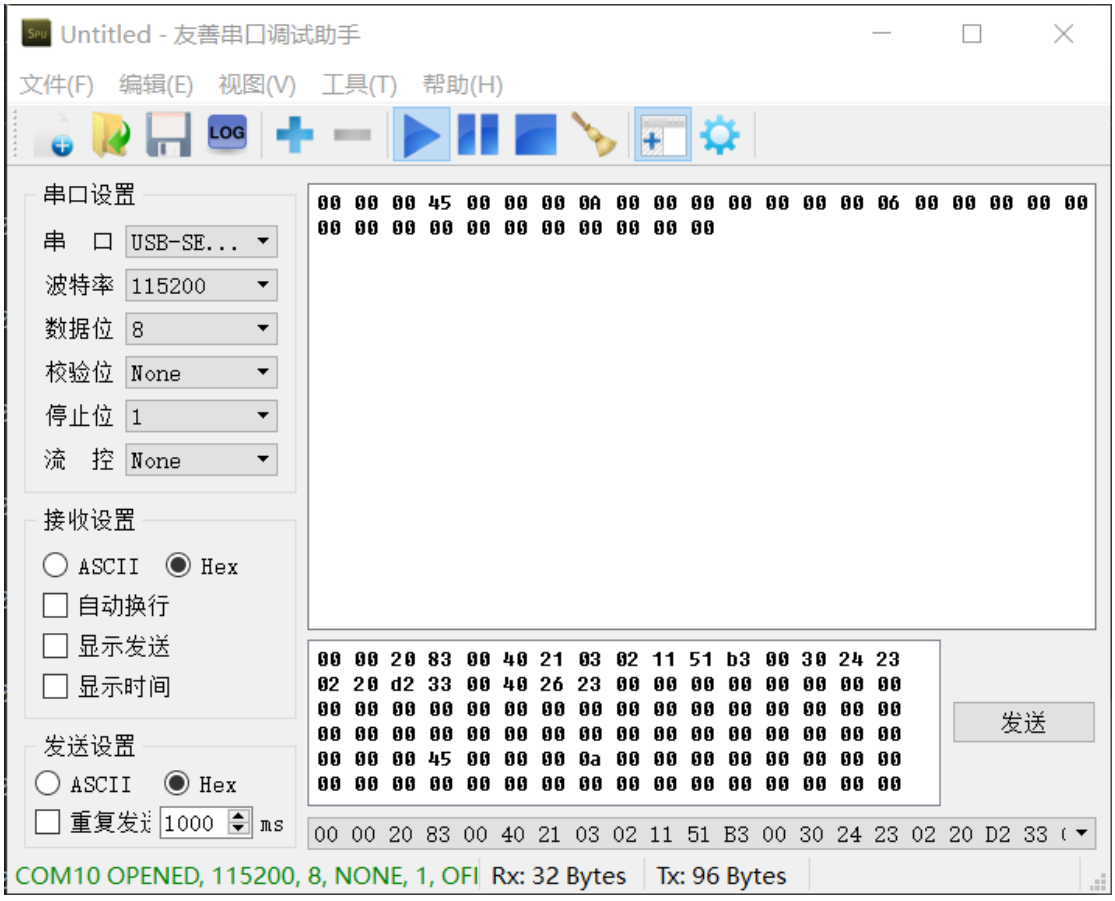

Untitled - 友善串口调试助手
文件(F) 编辑(E) 视图(V) 工具(T) 帮助(H)
LOG
串口设置
串 口 USB-SE...
波特率 115200
数据位 8
校验位 None
停止位 1
流 控 None
接收设置
ASCII Hex
自动换行
显示发送
显示时间
发送设置
ASCII Hex
重复发送 1000 ms
00 00 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
00 10 80 93 00 10 80 93 00 10 80 93 00 10 20 23
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
发送
00 10 80 93 00 10 80 93 00 10 80 93 00 10 20 23 00 00 00 00
COM10 OPENED, 115200, 8, NONE, 1, OFI
Rx: 32 Bytes
Tx: 96 Bytes

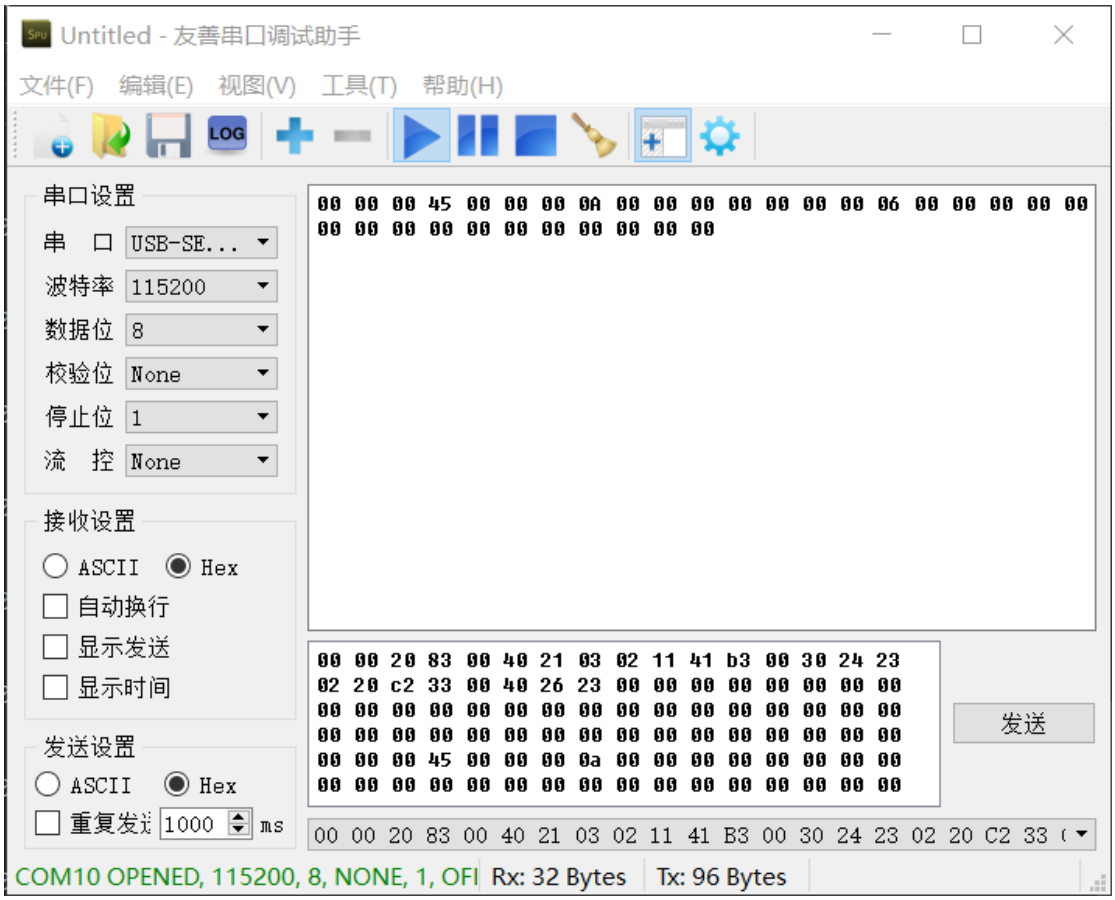

Untitled - 友善串口调试助手
文件(F) 编辑(E) 视图(V) 工具(T) 帮助(H)
LOG
串口设置
串 口 USB-SE...
波特率 115200
数据位 8
校验位 None
停止位 1
流 控 None
接收设置
ASCII Hex
自动换行
显示发送
显示时间
发送设置
ASCII Hex
重复发送 1000 ms
00 00 00 02 00 00 00 0C 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
00 10 80 93 00 10 80 93 00 80 01 6F 00 10 80 93
00 10 20 23 00 20 22 23 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
发送
00 10 80 93 00 10 80 93 00 80 01 6F 00 10 80 93 00 10 20 23
COM10 OPENED, 115200, 8, NONE, 1, OFI
Rx: 32 Bytes
Tx: 96 Bytes


Untitled - 友善串口调试助手
文件(F) 编辑(E) 视图(V) 工具(T) 帮助(H)
LOG
串口设置
串 口 USB-SE...
波特率 115200
数据位 8
校验位 None
停止位 1
流 控 None
接收设置
ASCII Hex
自动换行
显示发送
显示时间
发送设置
ASCII Hex
重复发送 1000 ms
00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
00 10 80 93 00 10 20 23 00 00 20 83 00 10 80 93
00 10 20 23 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
发送
00 10 80 93 00 10 20 23 00 00 20 83 00 10 80 93 00 10 20 23
COM10 OPENED, 115200, 8, NONE, 1, OFI
Rx: 32 Bytes
Tx: 96 Bytes


Untitled - 友善串口调试助手
文件(F) 编辑(E) 视图(V) 工具(T) 帮助(H)
LOG
串口设置
串 口 USB-SE...
波特率 115200
数据位 8
校验位 None
停止位 1
流 控 None
接收设置
ASCII Hex
自动换行
显示发送
显示时间
发送设置
ASCII Hex
重复发送 1000 ms
00 00 00 45 00 00 00 0A 00 00 00 0A 00 00 00 09 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
00 00 20 83 00 40 21 03 02 11 71 b3 00 30 24 23
02 20 f2 33 00 40 26 23 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 45 00 00 00 0a 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
发送
00 00 20 83 00 40 21 03 02 11 71 B3 00 30 24 23 02 20 F2 33
COM10 OPENED, 115200, 8, NONE, 1, OFI
Rx: 32 Bytes
Tx: 96 Bytes


Untitled - 友善串口调试助手
文件(F) 编辑(E) 视图(V) 工具(T) 帮助(H)
LOG
串口设置
串 口 USB-SE...
波特率 115200
数据位 8
校验位 None
停止位 1
流 控 None
接收设置
ASCII Hex
自动换行
显示发送
显示时间
发送设置
ASCII Hex
重复发送 1000 ms
00 00 00 45 00 00 00 0A 00 00 00 0A 00 00 00 09 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
00 00 20 83 00 40 21 03 02 11 61 b3 00 30 24 23
02 20 e2 33 00 40 26 23 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 45 00 00 00 0a 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
发送
00 00 20 83 00 40 21 03 02 11 61 B3 00 30 24 23 02 20 E2 33
COM10 OPENED, 115200, 8, NONE, 1, OFI
Rx: 32 Bytes
Tx: 96 Bytes


Untitled - 友善串口调试助手
文件(F) 编辑(E) 视图(V) 工具(T) 帮助(H)
LOG
串口设置
串 口 USB-SE...
波特率 115200
数据位 8
校验位 None
停止位 1
流 控 None
接收设置
ASCII Hex
自动换行
显示发送
显示时间
发送设置
ASCII Hex
重复发送 1000 ms
00 00 00 45 00 00 00 0A 00 00 00 00 00 00 00 06 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00
00 00 20 83 00 40 21 03 02 11 51 b3 00 30 24 23
02 20 d2 33 00 40 26 23 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 45 00 00 00 0a 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
发送
00 00 20 83 00 40 21 03 02 11 51 B3 00 30 24 23 02 20 D2 33
COM10 OPENED, 115200, 8, NONE, 1, OFI
Rx: 32 Bytes
Tx: 96 Bytes

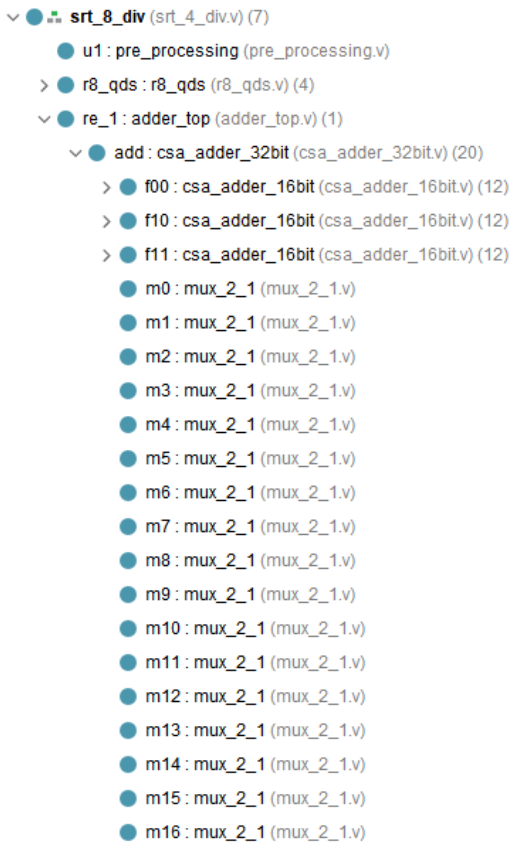


九、 实验数据及结果分析

1. 除法器的优化

a) 对于关键路径的延时的优化

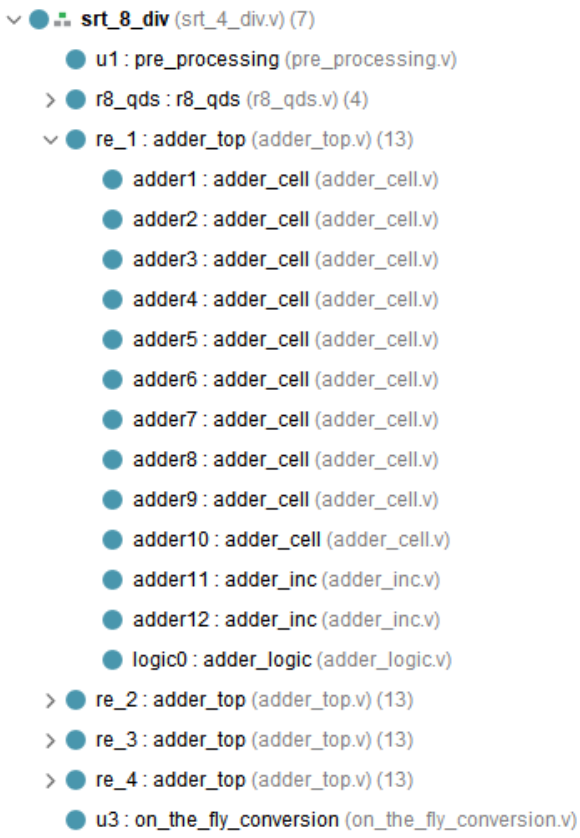
对于SRT除法中的加法单元，分别采用CLA、CSA、普通全加器进行比较。可以看到使用CSA的$SRT8$的大致结构如下图，对于内部加法器的实际算法不在此赘述。



经过$Vivado$的时序报告如下图，逻辑延迟已经用红色方框标出。由于$FPGA$布线和ICC自动布线的差异性，我们仅参考逻辑延迟。

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Cl
Constrained Paths (1)														
CLK_50M (10)														
Path 11	5.634	19	18	43	divisor_reg_reg[5]/C	u3/qm_reg_reg[20]/D	14.231	3.948	10.283	27.7	72.3	20.000	CLK_50M	CLK_50M
Path 12	5.647	19	18	43	divisor_reg_reg[5]/C	u3/qm_reg_reg[21]/D	14.220	3.948	10.272	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 13	5.665	19	17	72	divisor_reg_reg[5]/C	w_reg_reg[8]/D	14.304	4.409	9.895	30.8	69.2	20.000	CLK_50M	CLK_50M
Path 14	5.665	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[28]/D	14.201	3.948	10.253	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 15	5.667	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[25]/D	14.197	3.948	10.249	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 16	5.673	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[24]/D	14.192	3.948	10.244	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 17	5.681	19	17	72	divisor_reg_reg[5]/C	w_reg_reg[9]/D	14.288	4.409	9.879	30.9	69.1	20.000	CLK_50M	CLK_50M
Path 18	5.702	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[23]/D	14.205	3.948	10.257	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 19	5.717	19	18	43	divisor_reg_reg[5]/C	u3/qm_reg_reg[18]/D	14.189	3.948	10.241	27.8	72.2	20.000	CLK_50M	CLK_50M
Path 20	5.724	19	18	43	divisor_reg_reg[5]/C	u3/q_reg_reg[30]/D	14.141	3.948	10.193	27.9	72.1	20.000	CLK_50M	CLK_50M

之后我们又将CSA换为了CLA，结构如下：



可以看到，逻辑延迟明显下降。

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Excep
Constrained Paths (1)															
CLK_50M (10)															
Path 11	5.406	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[5]/D	14.321	3.777	10.544	26.4	73.6	20.000	CLK_50M	CLK_50M	
Path 12	5.480	20	18	82	divisor_reg_reg[8]/C	w_reg_reg[4]/D	14.291	3.646	10.645	25.5	74.5	20.000	CLK_50M	CLK_50M	
Path 13	5.498	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[7]/D	14.226	3.777	10.449	26.5	73.5	20.000	CLK_50M	CLK_50M	
Path 14	5.533	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[9]/D	14.195	3.777	10.418	26.6	73.4	20.000	CLK_50M	CLK_50M	
Path 15	5.584	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[16]/D	14.140	3.777	10.363	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 16	5.587	19	18	44	divisor_reg_reg[8]/C	u3/qm_reg_reg[27]/D	14.367	3.508	10.859	24.4	75.6	20.000	CLK_50M	CLK_50M	
Path 17	5.604	19	18	55	divisor_reg_reg[8]/C	u3/q_reg_reg[11]/D	14.350	3.508	10.842	24.4	75.6	20.000	CLK_50M	CLK_50M	
Path 18	5.613	20	18	82	divisor_reg_reg[8]/C	w_reg_reg[1]/D	14.116	3.646	10.470	25.8	74.2	20.000	CLK_50M	CLK_50M	
Path 19	5.620	19	18	59	divisor_reg_reg[8]/C	u3/q_reg_reg[27]/D	14.332	3.508	10.824	24.5	75.5	20.000	CLK_50M	CLK_50M	
Path 20	5.622	19	18	44	divisor_reg_reg[8]/C	w_reg_reg[10]/D	14.105	3.777	10.328	26.8	73.2	20.000	CLK_50M	CLK_50M	

然而，尽管如此，无论是在$Vivado$还是LVS，采用逻辑综合软件自动综合出来的加法器，逻辑延迟反而最小，并且在DC中显示面积也相对较小，我们最终并未来得及查明理论和实际产生如此偏差结果的原因，但最后只得不加优化，采用最普通的加法。

$Vivado$关于使用普通加法器的除法器的时序报告如下：

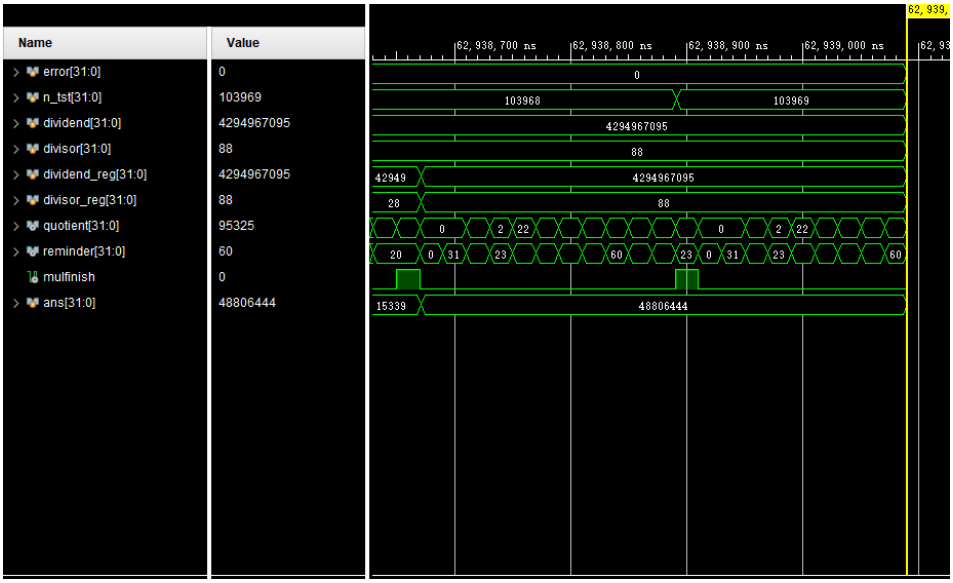
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %	Net %	Requirement	Source Clock	Destination Clock	Exceptio
Constrained Paths (1)															
CLK_50M (10)															
Path 11	7.520	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[6]/D	12.502	3.270	9.232	26.2	73.8	20.000	CLK_50M	CLK_50M	
Path 12	7.704	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[18]/D	12.322	3.285	9.037	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 13	7.731	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[7]/D	12.292	3.285	9.007	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 14	7.767	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[11]/D	12.255	3.300	8.955	26.9	73.1	20.000	CLK_50M	CLK_50M	
Path 15	7.791	18	17	46	divisor_reg_reg[33]/C	w_reg_reg[21]/D	12.234	3.266	8.968	26.7	73.3	20.000	CLK_50M	CLK_50M	
Path 16	7.809	18	16	86	divisor_reg_reg[33]/C	w_reg_reg[14]/D	12.215	3.285	8.930	26.9	73.1	20.000	CLK_50M	CLK_50M	
Path 17	7.813	18	17	46	divisor_reg_reg[33]/C	w_reg_reg[3]/D	12.251	2.997	9.254	24.5	75.5	20.000	CLK_50M	CLK_50M	
Path 18	7.826	19	18	81	divisor_reg_reg[33]/C	w_reg_reg[37]/D	11.923	3.409	8.514	28.6	71.4	20.000	CLK_50M	CLK_50M	
Path 19	7.830	19	17	81	divisor_reg_reg[33]/C	w_reg_reg[23]/D	12.197	3.417	8.780	28.0	72.0	20.000	CLK_50M	CLK_50M	
Path 20	7.833	19	17	81	divisor_reg_reg[33]/C	w_reg_reg[20]/D	12.193	3.417	8.776	28.0	72.0	20.000	CLK_50M	CLK_50M	

b) 对于计算正确率的测试与优化

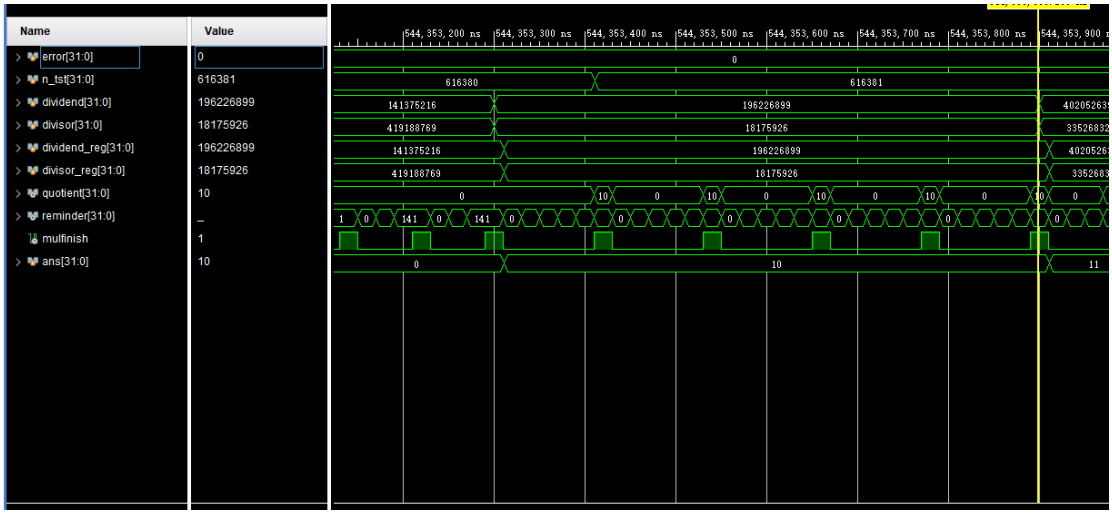
通过reg 32位测无符号，$integer$测有符号，定时更新测试数据寄存器，更新测试结果；通过在报告窗口打印出错时的$q1$和$q2$的值，方便修改优化qds表。

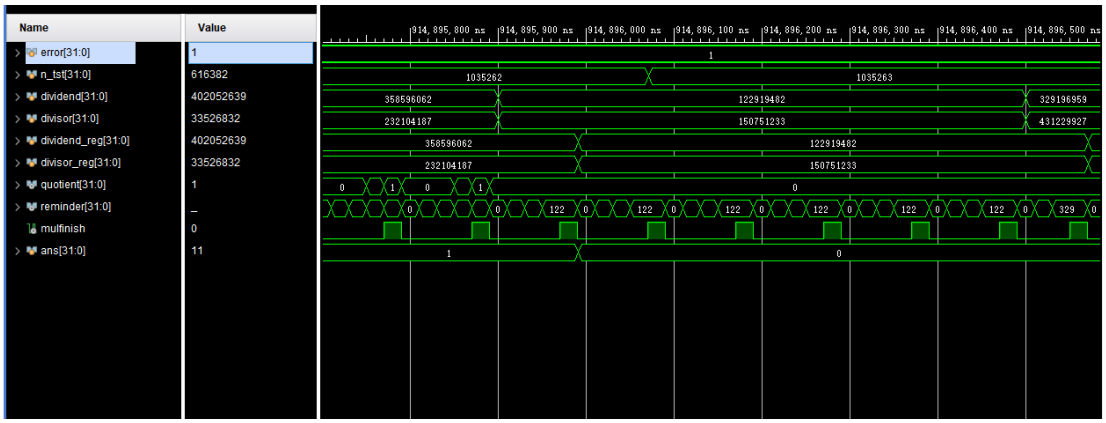
测试组	$dividend$	$divisor$	测试数量	正确率
A	$(\{\$random\}\%255) + 32'hffffff00$	$\{\$random\}\%255$	103,969	100%
B	$\{\$random\}\%4294967295$	$\{\$random\}\%4294967295$	1,668,484	99.99994%
C	$\{\$random\}\%4294967295$	$(\{\$random\}\%63) + 32'd960$	100,666	100%
D	$\{\$random\}\%4294967295$	$\{\$random\}\%4294967295$	2,141,102	100%

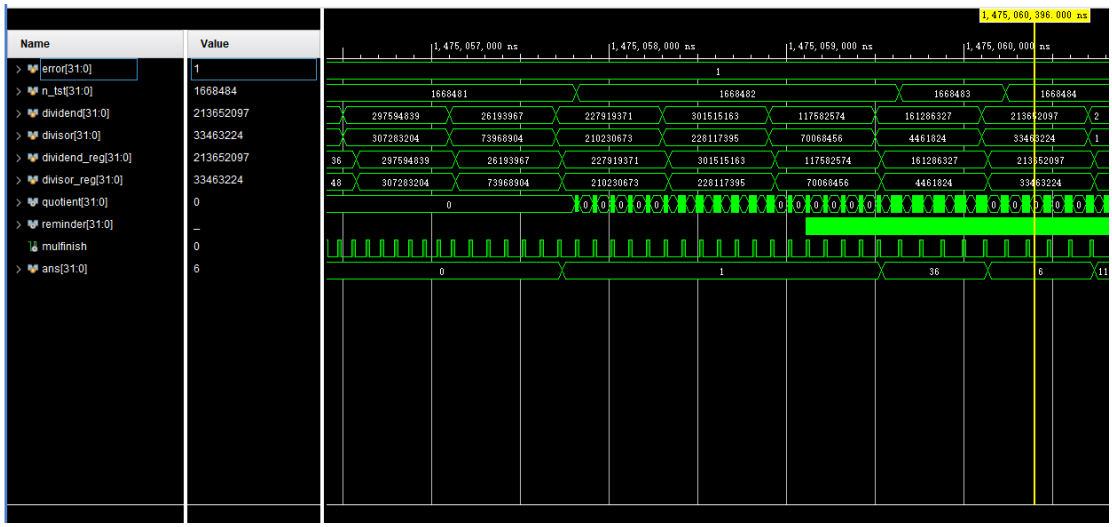
测试组A：测量除法器在被除数极大，除数极小情况下的正确率。



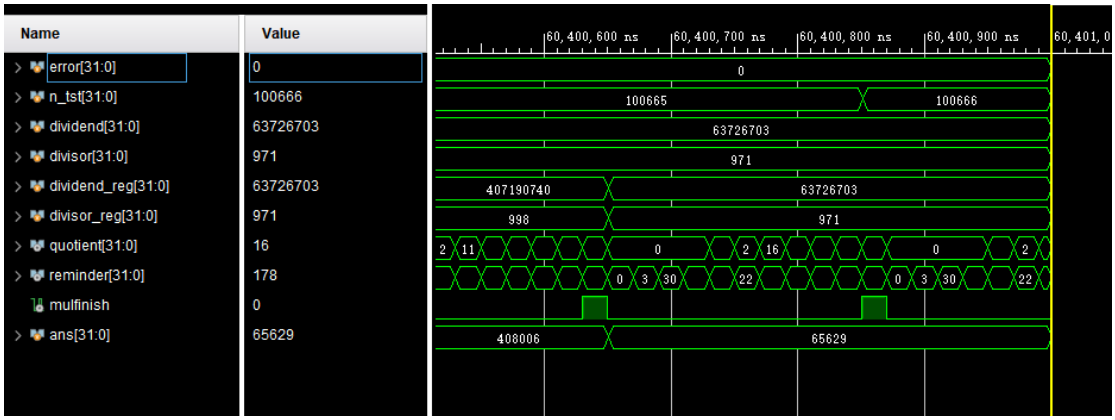
测试组B：测量除法器在完全随机的被除数与除数的情况下无符号运算的正确率。



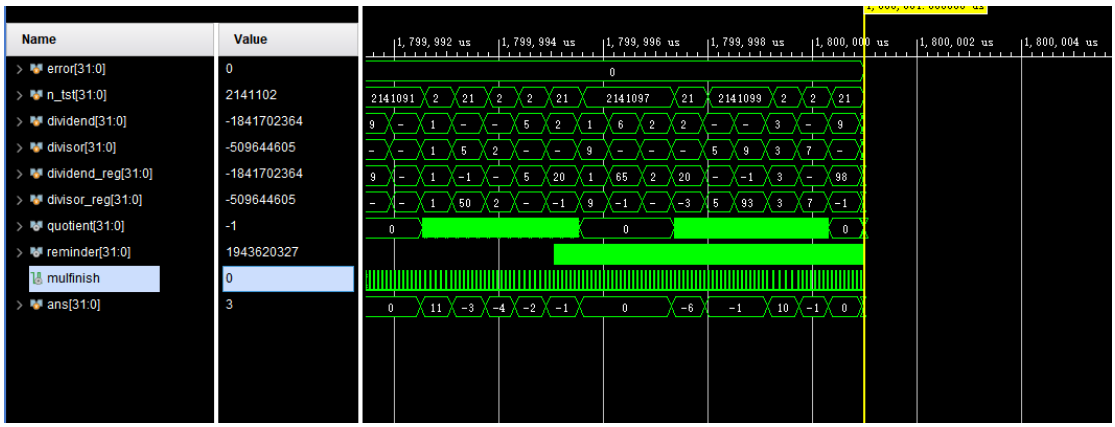




测试组C：测试被除数在确保首位连续4个1的情况下的正确率，之所以单独测这种情况是因为，对于SRT算法的qds选择来说，除数越大，则pd图对应的曲线的相对斜率越大，也因此对于qds选择的位宽精度要求就越大，在优化qds表提升正确率的大部分时间里，都需要在这种激励条件下进行测试。



测试组D：测量除法器在完全随机的被除数与除数的情况下有符号运算的正确率。



2. 处理器数据记录

处理器频率为25MHz，$UART$波特率为115200。

十、 实验结论

经过仿真，优化和测试，可认为实验目的已成功达到。

十一、 总结及心得体会

1. 熟悉了$Vivado$，VCS的使用方法。
2. 增强了团队沟通协作能力。

十二、 对本实验过程及方法、手段的改进建议

无。

十三、 代码

1. $FRAG\_adder.v$

```
module FRAG_adder #(
    parameter DATAWIDTH = 32
)(
    input  [DATAWIDTH - 1 : 0]  data_in_0   ,
    input  [DATAWIDTH - 1 : 0]  data_in_1   ,
    output [DATAWIDTH - 1 : 0]  data_out
);

    assign data_out = data_in_0 + data_in_1;

endmodule
```

2. $FRAG\_ALU\_ctrl.v$

```
module FRAG_ALU_ctrl (
    input       [1:0]   ALUOp   ,
    input               funct7_5,
    input       [2:0]   funct3  ,
    output  reg [3:0]   ALU_ctrl
);
```

```
localparam OP_ADD       =   2'b00   ;
localparam OP_SUB       =   2'b01   ;
localparam OP_TBD       =   2'b10   ;
// localparam OP_CMP       =   2'b11   ;
localparam OP_DIV       =   2'b11   ;

localparam ADD_SUB      =   3'b000  ;
// localparam SLL          =   3'b001  ;
// localparam SLT          =   3'b010  ;
// localparam SLTU         =   3'b011  ;
// localparam XOR          =   3'b100  ;
// localparam SRL_SRA      =   3'b101  ;
// localparam OR           =   3'b110  ;
// localparam AND          =   3'b111  ;

localparam BEQ          =   3'b000  ;
localparam BNE          =   3'b001  ;
// localparam BLT          =   3'b100  ;
// localparam BGE          =   3'b101  ;
localparam BLTU         =   3'b110  ;
localparam BGEU         =   3'b111  ;

localparam DIV          =   3'b100  ;
localparam DIVU         =   3'b101  ;
localparam REM          =   3'b110  ;
localparam REMU         =   3'b111  ;

// localparam CTRL_AND     =   4'b0000 ;
// localparam CTRL_OR      =   4'b0001 ;
localparam CTRL_ADD     =   4'b0010 ;
localparam CTRL_SUB     =   4'b0110 ;
// localparam CTRL_XOR     =   4'b0011 ;
// localparam CTRL_SLL     =   4'b0100 ;
// localparam CTRL_SRL     =   4'b0101 ;
// localparam CTRL_SRA     =   4'b0111 ;
// localparam CTRL_CMPS    =   4'b1000 ;
// localparam CTRL_CMPU    =   4'b1001 ;
// localparam CTRL_SLT     =   4'b1010 ;
// localparam CTRL_SLTU    =   4'b1011 ;
localparam CTRL_DIV     =   4'b1100 ;
localparam CTRL_DIVU    =   4'b1101 ;
localparam CTRL_REM     =   4'b1110 ;
localparam CTRL_REMU    =   4'b1111 ;
```

```
    always @(*) begin
        case(ALUOp)
            OP_ADD  : ALU_ctrl = CTRL_ADD;
            OP_SUB  : ALU_ctrl = CTRL_SUB;
            OP_TBD  : ALU_ctrl = (funct7_5 == 1'b0) ? CTRL_ADD :
CTRL_SUB   ;
            // OP_TBD  : case(funct3)
            //              ADD_SUB   : ALU_ctrl = (funct7_5 == 1'b0) ?
CTRL_ADD : CTRL_SUB   ;
                        //   SLL       : ALU_ctrl =
CTRL_SLL                                ;
                        //   SLT       : ALU_ctrl =
CTRL_SLT                                ;
                        //   SLTU      : ALU_ctrl =
CTRL_SLTU                               ;
                        //   XOR       : ALU_ctrl =
CTRL_XOR                                ;
                        //   SRL_SRA   : ALU_ctrl = (funct7_5 == 1'b0) ?
CTRL_SRL : CTRL_SRA   ;
                        //   OR        : ALU_ctrl =
CTRL_OR                                 ;
                        //   AND       : ALU_ctrl =
CTRL_AND                                ;
                    //       default   : ALU_ctrl =
4'b0000                                 ;
                    //   endcase
            // OP_CMP  : case(funct3)
            //              BEQ       : ALU_ctrl = CTRL_CMPS  ;
            //              BNE       : ALU_ctrl = CTRL_CMPS  ;
            //              BLT       : ALU_ctrl = CTRL_CMPS  ;
            //              BGE       : ALU_ctrl = CTRL_CMPS  ;
            //              BLTU      : ALU_ctrl = CTRL_CMPU  ;
            //              BGEU      : ALU_ctrl = CTRL_CMPU  ;
            //              default   : ALU_ctrl = 4'b0000    ;
            //           endcase
            OP_DIV  : case(funct3)
                        DIV       : ALU_ctrl =
CTRL_DIV                                ;
                        DIVU      : ALU_ctrl =
CTRL_DIVU                               ;
                        REM       : ALU_ctrl =
CTRL_REM                                ;
```

```
                        REMU      : ALU_ctrl =
CTRL_REMU                                    ;
                        default   : ALU_ctrl =
4'b0000                                      ;
                    endcase
            default : ALU_ctrl = 4'b0000 ;
        endcase
    end

endmodule
```

3. $FRAG\_ALU.v$

```
module FRAG_ALU (
    input             sys_clk      ,
    input             sys_arstn    ,
    input             flag_JorB    ,
    input       [31:0] data_in_0    ,
    input       [31:0] data_in_1    ,
    input       [ 3:0] ALU_ctrl     ,
    output      [31:0] result       ,
    output      [ 1:0] flag_result  ,
    output             hold
);

    wire        eq          ;
    wire        lg          ;
    reg  [31:0] result_temp ;
    wire        start       ;
    reg         start_delay1;
    wire        sign        ;
    reg         hold_temp   ;
    wire        finish      ;
    reg  [31:0] result_div  ;
    wire        sys_rst_n   ;
    wire        div_flag    ;
    wire        start_div   ;
    wire [31:0] q           ;
    wire [31:0] r           ;

    // localparam CTRL_AND     =   4'b0000 ;
    // localparam CTRL_OR      =   4'b0001 ;
    localparam CTRL_ADD     =   4'b0010 ;
    localparam CTRL_SUB     =   4'b0110 ;
    // localparam CTRL_XOR     =   4'b0011 ;
    // localparam CTRL_SLL     =   4'b0100 ;
```

```
    // localparam CTRL_SRL     =   4'b0101 ;
    // localparam CTRL_SRA     =   4'b0111 ;
    // localparam CTRL_CMPS    =   4'b1000 ;
    // localparam CTRL_CMPU    =   4'b1001 ;
    // localparam CTRL_SLT     =   4'b1010 ;
    // localparam CTRL_SLTU    =   4'b1011 ;
    localparam CTRL_DIV     =   4'b1100 ;
    localparam CTRL_DIVU    =   4'b1101 ;
    localparam CTRL_REM     =   4'b1110 ;
    localparam CTRL_REMU    =   4'b1111 ;

    assign sys_rst_n    = (~flag_JorB) &
sys_arstn
                                    ;
    assign div_flag     = (data_in_1 != 32'h0) & ((ALU_ctrl == CTRL_DIV)
| (ALU_ctrl == CTRL_DIVU) | (ALU_ctrl == CTRL_REM) | (ALU_ctrl ==
CTRL_REMU))  ;
    assign start_div    = start &
(~start_delay1)
                                          ;
    assign hold         = (hold_temp | start) &
(~finish)
                             ;
    assign start        = (div_flag == 1'b1) ? 1'b1 :
1'b0
                      ;
    assign sign         =
~ALU_ctrl[0]
                                               ;
    assign eq           = (result_temp ==
32'h0)
                                  ;
    assign lg           =
result_temp[31]
                                                ;
    assign flag_result  = {eq,
lg}
                                           ;
    assign result       = (finish == 1'b1) ? result_div :
result_temp
                   ;
    // assign result = ((ALU_ctrl == CTRL_SLT) || (ALU_ctrl ==
CTRL_SLTU)) ? {31'b0, lg} : result_temp;
```

```
    always @(*) begin
        case(ALU_ctrl)
            // CTRL_AND    :   result_temp = data_in_0 &
data_in_1                   ;
            // CTRL_OR     :   result_temp = data_in_0 |
data_in_1                   ;
            CTRL_ADD    :   result_temp = data_in_0 +
data_in_1                   ;
            CTRL_SUB    :   result_temp = data_in_0 -
data_in_1                   ;
            // CTRL_XOR    :   result_temp = data_in_0 ^
data_in_1                   ;
            // CTRL_SLL    :   result_temp = data_in_0 <<
data_in_1[4:0]            ;
            // CTRL_SRL    :   result_temp = $signed(data_in_0) <<
data_in_1[4:0]   ;
            // CTRL_SRA    :   result_temp = $signed(data_in_0) >>
data_in_1[4:0]   ;
            // CTRL_CMPS   :   result_temp = $signed(data_in_0) -
$signed(data_in_1);
            // CTRL_CMPU   :   result_temp = data_in_0 -
data_in_1                   ;
            // CTRL_SLT    :   result_temp = $signed(data_in_0) -
$signed(data_in_1);
            // CTRL_SLTU   :   result_temp = data_in_0 -
data_in_1                   ;
            CTRL_DIV    :   result_temp =
32'hffffffff                            ;
            CTRL_DIVU   :   result_temp =
32'hffffffff                            ;
            CTRL_REM    :   result_temp =
data_in_0                               ;
            CTRL_REMU   :   result_temp =
data_in_0                               ;
            default     :   result_temp = data_in_0 +
data_in_1                   ;
        endcase
    end

    always @(*) begin
        if(finish == 1'b1) begin
            case(ALU_ctrl)
                CTRL_DIV    :   result_div = q      ;
                CTRL_DIVU   :   result_div = q      ;
```

```
                CTRL_REM    :   result_div = r      ;
                CTRL_REMU   :   result_div = r      ;
                default     :   result_div = 32'h0  ;
            endcase
        end else begin
            result_div = 32'h0;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            hold_temp <= 1'b0;
        end else if(sys_rst_n == 1'b0) begin
            hold_temp <= 1'b0;
        end else if(start_div == 1'b1) begin
            hold_temp <= 1'b1;
        end else if(finish == 1'b1) begin
            hold_temp <= 1'b0;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            start_delay1 <= 1'b0;
        end else begin
            start_delay1 <= start;
        end
    end

    SRT_8_div SRT_8_div_inst(
        .clk            (sys_clk    )    ,
        .rst_n          (sys_rst_n  )    ,
        .start          (start_div  )    ,
        .dividend_i     (data_in_0  )    ,
        .divisor_i      (data_in_1  )    ,
        .sign_define    (sign       )    ,
        .quotient_o     (q          )    ,
        .reminder_o     (r          )    ,
        .mulfinish      (finish     )
    );

endmodule
```

4. *FRAG_ctrl.v*

```
module FRAG_ctrl (
```

```
    input       [ 6:0]  opcode  ,
    input       [ 2:0]  funct3  ,
    input               funct7_0,
    output  reg [20:0]  ctrl
);

    localparam R_M      = 7'b0110011;
    localparam I        = 7'b0010011;
    localparam L        = 7'b0000011;
    localparam S        = 7'b0100011;
    localparam JAL      = 7'b1101111;
    localparam JALR     = 7'b1100111;
    localparam B        = 7'b1100011;
    localparam LUI      = 7'b0110111;
    localparam AUIPC    = 7'b0010111;

    // reg [1:0]   ALUOp       ;
    // reg [1:0]   ALUSrc      ;
    // reg [1:0]   JumpBranch  ;
    // reg [2:0]   BranchType  ;
    // reg         MemRead     ;
    // reg [2:0]   LoadType    ;
    // reg         MemWrite    ;
    // reg [2:0]   StoreType   ;
    // reg [1:0]   Inst        ;
    // reg         RegWrite    ;
    // reg         MemtoReg    ;

    always @(*) begin
        case(opcode)
            R_M    : ctrl = (funct7_0 == 1'b0) ?
                            {2'b10, 2'b00, 2'b00, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b00, 2'b10} :
                            {2'b11, 2'b00, 2'b00, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b00, 2'b10};
            I      : ctrl = {2'b10, 2'b01, 2'b00, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b00, 2'b10};
            L      : ctrl = {2'b00, 2'b01, 2'b00, 3'b000, 1'b1, funct3,
1'b0, 3'b000, 2'b00, 2'b11};
            S      : ctrl = {2'b00, 2'b01, 2'b00, 3'b000, 1'b0, 3'b000,
1'b1, funct3, 2'b00, 2'b00};
            JAL    : ctrl = {2'b00, 2'b11, 2'b10, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b10, 2'b10};
```

```
            JALR   : ctrl = {2'b00, 2'b01, 2'b10, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b11, 2'b10};
            B      : ctrl = {2'b01, 2'b00, 2'b01, funct3, 1'b0, 3'b000,
1'b0, 3'b000, 2'b00, 2'b00};
            LUI    : ctrl = {2'b00, 2'b11, 2'b00, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b00, 2'b10};
            AUIPC  : ctrl = {2'b00, 2'b11, 2'b00, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b01, 2'b10};
            default: ctrl = {2'b00, 2'b00, 2'b00, 3'b000, 1'b0, 3'b000,
1'b0, 3'b000, 2'b00, 2'b00};
        endcase
    end

endmodule
```

5. *FRAG_data_mem.v*

```
module FRAG_data_mem #(
    parameter   UART_BPS    =   115200  ,
    parameter   CLK_FREQ    =   25000000
)(
    input               sys_clk     ,
    input               sys_arstn   ,
    input       [ 7:0]  ctrl        ,
    input               en_addr_data,
    input       [ 4:0]  addr_data   ,
    input       [31:0]  data_data   ,
    input       [31:0]  addr        ,
    input       [31:0]  data_in     ,
    input               flag        ,
    output  reg [31:0]  data_out    ,
    output  reg         tx
);

    // localparam LB  = 3'b000;
    // localparam LH  = 3'b001;
    localparam LW  = 3'b010;
    // localparam LBU = 3'b100;
    // localparam LHU = 3'b101;

    // localparam SB  = 3'b000;
    // localparam SH  = 3'b001;
    localparam SW  = 3'b010;

    localparam  BAUD_CNT_MAX = CLK_FREQ / UART_BPS;
```

```
    reg     [ 0:3]  cs                      ;
    reg     [ 7:0]  ctrl_delay1             ;
    reg     [31:0]  data_in_temp            ;
    reg     [ 7:0]  mem_0           [0:7]   ;
    reg     [ 7:0]  mem_1           [0:7]   ;
    reg     [ 7:0]  mem_2           [0:7]   ;
    reg     [ 7:0]  mem_3           [0:7]   ;
    reg     [ 7:0]  data_out_temp_0         ;
    reg     [ 7:0]  data_out_temp_1         ;
    reg     [ 7:0]  data_out_temp_2         ;
    reg     [ 7:0]  data_out_temp_3         ;
    reg     [ 8:0]  baud_cnt                ;
    reg             bit_flag                ;
    reg     [ 3:0]  bit_cnt                 ;
    reg             work_en                 ;
    wire            pi_flag                 ;
    reg     [ 7:0]  pi_data                 ;
    reg     [ 4:0]  byte_cnt                ;
    reg             byte_cnt_delay1         ;
    reg             byte_cnt_delay0         ;
    reg             stop                    ;

    integer i_0;
    integer i_1;
    integer i_2;
    integer i_3;

    assign pi_flag = ((byte_cnt == 5'b11111) & (byte_cnt_delay1 == 1'b0)
& (byte_cnt_delay0 == 1'b0)) ? 1'b0 : (byte_cnt_delay0 ^ byte_cnt[0]);

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            stop <= 1'b0;
        end else if((bit_flag == 1'b1) & (bit_cnt == 4'd9) & (byte_cnt
== 5'b11111)) begin
            stop <= 1'b1;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            byte_cnt <= 5'b11111;
        end else if(stop == 1'b1) begin
            byte_cnt <= byte_cnt;
```

```
    end else if((work_en == 1'b1) | (pi_flag == 1'b1)) begin
        byte_cnt <= byte_cnt;
    end else if((flag == 1'b1) | (byte_cnt != 5'b11111)) begin
        byte_cnt <= byte_cnt + 8'b1;
    end
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        byte_cnt_delay0 <= 1'b0;
    end else begin
        byte_cnt_delay0 <= byte_cnt[0];
    end
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        byte_cnt_delay1 <= 1'b0;
    end else begin
        byte_cnt_delay1 <= byte_cnt[1];
    end
end

always @(*) begin
    case(byte_cnt[1:0])
        2'b00   :   pi_data <= mem_0[byte_cnt[4:2]] ;
        2'b01   :   pi_data <= mem_1[byte_cnt[4:2]] ;
        2'b10   :   pi_data <= mem_2[byte_cnt[4:2]] ;
        2'b11   :   pi_data <= mem_3[byte_cnt[4:2]] ;
        default :   pi_data <= 8'b0                 ;
    endcase
end

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        work_en <= 1'b0;
    else    if(pi_flag == 1'b1)
        work_en <= 1'b1;
    else    if((bit_flag == 1'b1) && (bit_cnt == 4'd9))
        work_en <= 1'b0;

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        baud_cnt <= 9'b0;
```

```
    else    if((baud_cnt == BAUD_CNT_MAX - 1) || (work_en == 1'b0))
        baud_cnt <= 9'b0;
    else    if(work_en == 1'b1)
        baud_cnt <= baud_cnt + 1'b1;

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        bit_flag <= 1'b0;
    else    if(baud_cnt == 9'd1)
        bit_flag <= 1'b1;
    else
        bit_flag <= 1'b0;

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        bit_cnt <= 4'b0;
    else    if((bit_flag == 1'b1) && (bit_cnt == 4'd10))
        bit_cnt <= 4'b0;
    else    if((bit_flag == 1'b1) && (work_en == 1'b1))
        bit_cnt <= bit_cnt + 1'b1;

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        tx <= 1'b1;
    else    if(bit_flag == 1'b1)
        case(bit_cnt)
            0       : tx <= 1'b0;
            1       : tx <= pi_data[0];
            2       : tx <= pi_data[1];
            3       : tx <= pi_data[2];
            4       : tx <= pi_data[3];
            5       : tx <= pi_data[4];
            6       : tx <= pi_data[5];
            7       : tx <= pi_data[6];
            8       : tx <= pi_data[7];
            9       : tx <= 1'b1;
            10      : tx <= 1'b1;
            default : tx <= 1'b1;
        endcase

always @(*) begin
    if(ctrl[7] == 1'b1) begin
        case(ctrl[6:4])
            // LB      :   case(addr[1:0])
```

```
        //                  2'b00   :   cs = 4'b1000;
        //                  2'b01   :   cs = 4'b0100;
        //                  2'b10   :   cs = 4'b0010;
        //                  2'b11   :   cs = 4'b0001;
        //                  default :   cs = 4'b0000;
        //              endcase
        // LH       :   case(addr[1])
        //                  1'b0    :   cs = 4'b1100;
        //                  1'b1    :   cs = 4'b0011;
        //                  default :   cs = 4'b0000;
        //              endcase
        LW          :                   cs = 4'b1111;
        // LBU      :   case(addr[1:0])
        //                  2'b00   :   cs = 4'b1000;
        //                  2'b01   :   cs = 4'b0100;
        //                  2'b10   :   cs = 4'b0010;
        //                  2'b11   :   cs = 4'b0001;
        //                  default :   cs = 4'b0000;
        //              endcase
        // LHU      :   case(addr[1])
        //                  1'b0    :   cs = 4'b1100;
        //                  1'b1    :   cs = 4'b0011;
        //                  default :   cs = 4'b0000;
        //              endcase
        default :                   cs = 4'b0000;
    endcase
end else if(ctrl[3] == 1'b1) begin
    case(ctrl[2:0])
        // SB       :   case(addr[1:0])
        //                  2'b00   :   cs = 4'b1000;
        //                  2'b01   :   cs = 4'b0100;
        //                  2'b10   :   cs = 4'b0010;
        //                  2'b11   :   cs = 4'b0001;
        //                  default :   cs = 4'b0000;
        //              endcase
        // SH       :   case(addr[1])
        //                  1'b0    :   cs = 4'b1100;
        //                  1'b1    :   cs = 4'b0011;
        //                  default :   cs = 4'b0000;
        //              endcase
        SW          :                   cs = 4'b1111;
        default :                   cs = 4'b0000;
    endcase
end else begin
```

```
            cs = 4'b0000;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            ctrl_delay1 <= 8'b0;
        end else begin
            ctrl_delay1 <= ctrl;
        end
    end

    always @(*) begin
        if(ctrl[3] == 1'b1) begin
            case(ctrl[2:0])
                // SB      :   case(addr[1:0])
                //                 2'b00   :   data_in_temp =
{data_in[7:0], 24'b0}        ;
                //                 2'b01   :   data_in_temp = {8'b0,
data_in[7:0], 16'b0}  ;
                //                 2'b10   :   data_in_temp = {16'b0,
data_in[7:0], 8'b0}  ;
                //                 2'b11   :   data_in_temp = {24'b0,
data_in[7:0]}        ;
                //                 default :   data_in_temp =
32'h0                        ;
                //             endcase
                // SH      :   case(addr[1])
                //                 1'b0    :   data_in_temp =
{data_in[15:0], 16'b0}       ;
                //                 1'b1    :   data_in_temp = {16'b0,
data_in[15:0]}       ;
                //                 default :   data_in_temp =
32'h0                        ;
                //             endcase
                SW      :                   data_in_temp =
data_in                      ;
                default :                   data_in_temp =
32'h0                        ;
            endcase
        end else begin
            data_in_temp = 32'h0;
        end
    end
```

```
always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        for(i_0 = 0; i_0 <= 7; i_0 = i_0 + 1) begin
            mem_0[i_0] <= 8'h0;
        end
    end else if(en_addr_data == 1'b1) begin
        mem_0[addr_data[4:2]] <= data_data[31:24];
    end else if((cs[0] == 1'b1) && (ctrl[3] == 1'b1))begin
        mem_0[addr[4:2]] <= data_in_temp[31:24];
    end
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        for(i_1 = 0; i_1 <= 7; i_1 = i_1 + 1) begin
            mem_1[i_1] <= 8'h0;
        end
    end else if(en_addr_data == 1'b1) begin
        mem_1[addr_data[4:2]] <= data_data[23:16];
    end else if((cs[1] == 1'b1) && (ctrl[3] == 1'b1))begin
        mem_1[addr[4:2]] <= data_in_temp[23:16];
    end
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        for(i_2 = 0; i_2 <= 7; i_2 = i_2 + 1) begin
            mem_2[i_2] <= 8'h0;
        end
    end else if(en_addr_data == 1'b1) begin
        mem_2[addr_data[4:2]] <= data_data[15:8];
    end else if((cs[2] == 1'b1) && (ctrl[3] == 1'b1))begin
        mem_2[addr[4:2]] <= data_in_temp[15:8];
    end
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        for(i_3 = 0; i_3 <= 7; i_3 = i_3 + 1) begin
            mem_3[i_3] <= 8'h0;
        end
    end else if(en_addr_data == 1'b1) begin
        mem_3[addr_data[4:2]] <= data_data[7:0];
```

```
        end else if((cs[3] == 1'b1) && (ctrl[3] == 1'b1))begin
            mem_3[addr[4:2]] <= data_in_temp[7:0];
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            data_out_temp_0 <= 8'h0;
        end else if((cs[0] == 1'b1) & (ctrl[7] == 1'b1))begin
            data_out_temp_0 <= mem_0[addr[4:2]];
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            data_out_temp_1 <= 8'h0;
        end else if((cs[1] == 1'b1) & (ctrl[7] == 1'b1))begin
            data_out_temp_1 <= mem_1[addr[4:2]];
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            data_out_temp_2 <= 8'h0;
        end else if((cs[2] == 1'b1)  & (ctrl[7] == 1'b1))begin
            data_out_temp_2 <= mem_2[addr[4:2]];
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            data_out_temp_3 <= 8'h0;
        end else if((cs[3] == 1'b1) & (ctrl[7] == 1'b1))begin
            data_out_temp_3 <= mem_3[addr[4:2]];
        end
    end

    always @(*) begin
        if(ctrl_delay1[7] == 1'b1) begin
            case(ctrl_delay1[6:4])
                // LB      :   case(addr[1:0])
                //                 2'b00   :   data_out =
{{24{data_out_temp_0[7]}}, data_out_temp_0}                         ;
```

```
                //                2'b01   :   data_out =
{{24{data_out_temp_1[7]}}, data_out_temp_1}                          ;
                //                2'b10   :   data_out =
{{24{data_out_temp_2[7]}}, data_out_temp_2}                          ;
                //                2'b11   :   data_out =
{{24{data_out_temp_3[7]}}, data_out_temp_3}                          ;
                //                default :   data_out =
32'h0                                                               ;
                //            endcase
                // LH      :   case(addr[1])
                //                1'b0    :   data_out =
{{16{data_out_temp_0[7]}}, data_out_temp_0, data_out_temp_1}         ;
                //                1'b1    :   data_out =
{{16{data_out_temp_2[7]}}, data_out_temp_2, data_out_temp_3}         ;
                //                default :   data_out =
32'h0                                                               ;
                //            endcase
                LW      :                   data_out = {data_out_temp_0,
data_out_temp_1, data_out_temp_2, data_out_temp_3} ;
                // LBU     :   case(addr[1:0])
                //                2'b00   :   data_out = {{24{1'b0}},
data_out_temp_0}                                    ;
                //                2'b01   :   data_out = {{24{1'b0}},
data_out_temp_1}                                    ;
                //                2'b10   :   data_out = {{24{1'b0}},
data_out_temp_2}                                    ;
                //                2'b11   :   data_out = {{24{1'b0}},
data_out_temp_3}                                    ;
                //                default :   data_out =
32'h0                                                               ;
                //            endcase
                // LHU     :   case(addr[1])
                //                1'b0    :   data_out = {{24{1'b0}},
data_out_temp_0, data_out_temp_1}                    ;
                //                1'b1    :   data_out = {{24{1'b0}},
data_out_temp_2, data_out_temp_3}                    ;
                //                default :   data_out =
32'h0                                                               ;
                //            endcase
                default :                   data_out =
32'h0                                                               ;
            endcase
        end else begin
            data_out = 32'h0;
```

```
        end
    end

endmodule
```

6. $FRAG\_forward.v$

```
module FRAG_forward (
    input       [4:0]   mem_Rd              ,
    input       [1:0]   mem_wb_ctrl         ,
    input       [4:0]   wb_Rd               ,
    input       [1:0]   wb_wb_ctrl          ,
    input               ex_MemWrite         ,
    input       [4:0]   ex_Rs1              ,
    input       [4:0]   ex_Rs2              ,
    output  reg [1:0]   forward_Rs1_sel     ,
    output  reg [1:0]   forward_Rs2_sel     ,
    output  reg [1:0]   forward_store_sel
);

    always @(*) begin
        if((mem_wb_ctrl[1] == 1'b1) & (mem_Rd != 5'b0) & (mem_Rd ==
ex_Rs1)) begin
            forward_Rs1_sel = 2'b01;
        end else if((wb_wb_ctrl[1] == 1'b1) & (wb_Rd != 5'b0) & (wb_Rd
== ex_Rs1)) begin
            forward_Rs1_sel = 2'b10;
        end else begin
            forward_Rs1_sel = 2'b00;
        end
    end

    always @(*) begin
        if((mem_wb_ctrl[1] == 1'b1) & (mem_Rd != 5'b0) & (mem_Rd ==
ex_Rs2)) begin
            forward_Rs2_sel = 2'b01;
        end else if((wb_wb_ctrl[1] == 1'b1) & (wb_Rd != 5'b0) & (wb_Rd
== ex_Rs2)) begin
            forward_Rs2_sel = 2'b10;
        end else begin
            forward_Rs2_sel = 2'b00;
        end
    end

    always @(*) begin
```

```
        if((mem_wb_ctrl[1] == 1'b1) & (ex_MemWrite == 1'b1) & (mem_Rd !=
5'b0) & (mem_Rd == ex_Rs2)) begin
            forward_store_sel = 2'b01;
        end else if((wb_wb_ctrl[1] == 1'b1) & (ex_MemWrite == 1'b1) &
(wb_Rd != 5'b0) & (wb_Rd == ex_Rs2)) begin
            forward_store_sel = 2'b10;
        end else begin
            forward_store_sel = 2'b00;
        end
    end

endmodule
```

7. *FRAG_hazard.v*

```
module FRAG_hazard (
    input               sys_clk      ,
    input               sys_arstn    ,
    input               start        ,
    input               hold         ,
    input       [4:0]   ex_Rd        ,
    input               ex_MemRead   ,
    input       [4:0]   id_Rs1       ,
    input       [4:0]   id_Rs2       ,
    input               flag_JorB    ,
    output              flag_flush   ,
    output      [2:0]   flag_hold
);

    reg  hold_0_temp        ;
    reg  hold_0_temp_delay1 ;
    wire hold_0             ;
    wire hold_1             ;
    wire hold_2             ;

    assign flag_flush   = flag_JorB                          ;
    assign flag_hold    = {hold_2, hold_1, hold_0}            ;
    assign hold_0       = hold_0_temp & (~hold_0_temp_delay1)  ;
    assign hold_1       = hold                               ;
    assign hold_2       = ~start                             ;

    always @(*) begin
        if((ex_MemRead == 1'b1) & (ex_Rd != 5'b0) & ((ex_Rd == id_Rs1) |
(ex_Rd == id_Rs2))) begin
            hold_0_temp = 1'b1;
        end else begin
```

```
            hold_0_temp = 1'b0;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            hold_0_temp_delay1 <= 1'b0;
        end else begin
            hold_0_temp_delay1 <= hold_0_temp;
        end
    end

endmodule
```

8. *FRAG_imm_gen.v*

```
module FRAG_imm_gen (
    input       [31: 0]  inst_data,
    output reg  [31: 0]  imm
);

    localparam R_M      = 7'b0110011;
    localparam I        = 7'b0010011;
    localparam L        = 7'b0000011;
    localparam S        = 7'b0100011;
    localparam JAL      = 7'b1101111;
    localparam JALR     = 7'b1100111;
    localparam B        = 7'b1100011;
    localparam LUI      = 7'b0110111;
    localparam AUIPC    = 7'b0010111;


    always @(*) begin
        case(inst_data[6:0])
            R_M    : imm =
32'h0
  ;
            I      : imm = {{20{inst_data[31]}},
inst_data[31:20]}                                   ;
            L      : imm = {{20{inst_data[31]}},
inst_data[31:20]}                                   ;
            S      : imm = {{20{inst_data[31]}}, inst_data[31:25],
inst_data[11:7]}                        ;
            JAL    : imm = {{12{inst_data[31]}}, inst_data[19:12],
inst_data[20], inst_data[30:21], 1'b0};
```

```
            JALR   : imm = {{20{inst_data[31]}},
inst_data[31:20]}                                       ;
            B      : imm = {{20{inst_data[31]}}, inst_data[7],
inst_data[30:25], inst_data[11:8], 1'b0}  ;
            LUI    : imm = {inst_data[31:12],
12'h0}                                                  ;
            AUIPC  : imm = {inst_data[31:12],
12'h0}                                                  ;
            default: imm =
32'h0
  ;
        endcase
    end

endmodule
```

9. *FRAG_inst_mem.v*

```
module FRAG_inst_mem (
    input               sys_clk     ,
    input               sys_arstn   ,
    input               en_addr_inst,
    input       [ 5:0]  addr_inst   ,
    input       [31:0]  data_inst   ,
    input       [31:0]  addr        ,
    output reg  [31:0]  data
);

    reg [31:0] mem[0:15];

    integer i;

    always @(posedge sys_clk, negedge sys_arstn) begin
        if (sys_arstn == 1'b0) begin
            for(i = 0; i <= 15; i = i + 1) begin
                mem[i] <= 32'h0;
            end
        end else if(en_addr_inst == 1'b1) begin
            mem[addr_inst[5:2]] <= data_inst;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if (sys_arstn == 1'b0) begin
            data <= 32'h0;
        end else if (addr[31:6] == 26'b0) begin
```

```
            data <= mem[addr[5:2]];
        end else begin
            data <= 32'h0;
        end
    end

endmodule
```

10. $FRAG\_JorB\_ctrl.v$

```
module FRAG_JorB_ctrl (
    input       [4:0]   ctrl_JorB   ,
    input       [1:0]   flag_result ,
    output  reg         flag_JorB
);

    localparam BEQ      =   3'b000  ;
    localparam BNE      =   3'b001  ;
    // localparam BLT      =   3'b100  ;
    // localparam BGE      =   3'b101  ;
    localparam BLTU     =   3'b110  ;
    localparam BGEU     =   3'b111  ;

    always @(*) begin
        if(ctrl_JorB[4] == 1'b1) begin
            flag_JorB = 1'b1;
        end else if(ctrl_JorB[3] == 1'b1) begin
            case(ctrl_JorB[2:0])
                BEQ    : flag_JorB =  flag_result[1];
                BNE    : flag_JorB = ~flag_result[1];
        //         BLT    : flag_JorB =  flag_result[0];
        //         BGE    : flag_JorB = ~flag_result[0];
                BLTU   : flag_JorB =  flag_result[0];
                BGEU   : flag_JorB = ~flag_result[0];
                default: flag_JorB =  flag_result[0];
            endcase
        end else begin
            flag_JorB = 1'b0;
        end
    end

endmodule
```

11. $FRAG\_mux\_2.v$

```
module FRAG_mux_2 #(
    parameter DATAWIDTH = 32
)(
```

```
    input  [DATAWIDTH - 1 : 0]  data_in_0   ,
    input  [DATAWIDTH - 1 : 0]  data_in_1   ,
    input                       select      ,
    output [DATAWIDTH - 1 : 0]  data_out
);

    assign data_out = (select == 1'b0) ? data_in_0 : data_in_1;

endmodule
```

12. *FRAG_mux_3.v*

```
module FRAG_mux_3 #(
    parameter DATAWIDTH = 32
)(
    input  [DATAWIDTH - 1 : 0]  data_in_0   ,
    input  [DATAWIDTH - 1 : 0]  data_in_1   ,
    input  [DATAWIDTH - 1 : 0]  data_in_2   ,
    input  [            1 : 0]  select      ,
    output [DATAWIDTH - 1 : 0]  data_out
);

    assign data_out =   (select[1] == 1'b0) ?
                       ((select[0] == 1'b0) ? data_in_0 : data_in_1) :
                       data_in_2;

endmodule
```

13. *FRAG_pipline.v*

```
module FRAG_pipline #(
    parameter DataWidth = 32
) (
    input                       sys_clk         ,
    input                       sys_arstn       ,
    input                       flag_flush      ,
    input                       flag_hold       ,
    input   [DataWidth - 1:0]   default_value   ,
    input   [DataWidth - 1:0]   data_in         ,
    output  [DataWidth - 1:0]   data_out
);

    reg [DataWidth - 1:0] data_temp;

    assign data_out = data_temp;

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
```

```
            data_temp <= default_value;
        end else if(flag_flush == 1'b1) begin
            data_temp <= default_value;
        end else if(flag_hold == 1'b1) begin
            data_temp <= data_temp;
        end else begin
            data_temp <= data_in;
        end
    end

endmodule
```

14. *FRAG_register_file.v*

```
module FRAG_register_file (
    input                   sys_clk         ,
    input                   sys_arstn       ,
    input                   RegWrite        ,
    input       [ 4:0]      w_addr          ,
    input       [31:0]      w_data          ,
    input       [ 4:0]      reg1_r_addr     ,
    input       [ 4:0]      reg2_r_addr     ,
    output  reg [31:0]      reg1_r_data     ,
    output  reg [31:0]      reg2_r_data
);

    reg [31: 0] regs[0:19];

    integer i;

    always @(posedge sys_clk, negedge sys_arstn) begin
        if (sys_arstn == 1'b0) begin
            for(i = 0; i <= 19; i = i + 1) begin
                regs[i] <= 32'h0;
            end
        end else if ((RegWrite == 1'b1) && (w_addr != 32'h0)) begin
            regs[w_addr] <= w_data;
        end
    end

    always @(*) begin
        if (reg1_r_addr == 32'h0) begin
            reg1_r_data = 32'h0;
        end else if ((RegWrite == 1'b1) && (reg1_r_addr == w_addr))
begin
            reg1_r_data = w_data;
```

```
        end else begin
            reg1_r_data = regs[reg1_r_addr];
        end
    end

    always @(*) begin
        if (reg2_r_addr == 32'h0) begin
            reg2_r_data = 32'h0;
        end else if ((RegWrite == 1'b1) && (reg2_r_addr == w_addr))
begin
            reg2_r_data = w_data;
        end else begin
            reg2_r_data = regs[reg2_r_addr];
        end
    end

endmodule
```

15. $FRAG\_UART.v$

```
module FRAG_UART #(
    parameter   UART_BPS    =   115200  ,
    parameter   CLK_FREQ    =   25000000
)(
    input               sys_clk     ,
    input               sys_arstn   ,
    input               rx          ,
    output              en_addr_inst,
    output              en_addr_data,
    output reg  [ 5:0]  addr_inst   ,
    output reg  [ 4:0]  addr_data   ,
    output      [31:0]  data        ,
    output reg          start

);

    localparam  BAUD_CNT_MAX = CLK_FREQ / UART_BPS;

    reg         rx_reg1             ;
    reg         rx_reg2             ;
    reg         rx_reg3             ;
    reg         start_nedge         ;
    reg         work_en             ;
    reg [ 9:0]  baud_cnt            ;
    reg         bit_flag            ;
    reg [ 3:0]  bit_cnt             ;
```

```
    reg [ 7:0]  rx_data             ;
    reg         rx_flag             ;
    reg [ 7:0]  po_data             ;
    reg         po_flag             ;
    reg [31:0]  data_temp           ;
    reg         po_flag_delay1      ;
    reg         po_flag_delay2      ;
    reg         en_addr_inst_temp1  ;
    reg         en_addr_inst_temp2  ;
    reg         en_addr_data_temp1  ;
    reg         en_addr_data_temp2  ;

    assign data         = ((en_addr_inst == 1'b1) | (en_addr_data ==
1'b1)) ? data_temp : 32'h0 ;
    assign en_addr_inst = en_addr_inst_temp1 &
en_addr_inst_temp2                              ;
    assign en_addr_data = en_addr_data_temp1 &
en_addr_data_temp2                              ;

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            en_addr_inst_temp1 <= 1'b0;
        end else if((addr_inst[1:0] == 2'b10) & (po_flag == 1'b1)) begin
            en_addr_inst_temp1 <= 1'b1;
        end else begin
            en_addr_inst_temp1 <= 1'b0;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            en_addr_data_temp1 <= 1'b0;
        end else if((addr_data[1:0] == 2'b10) & (po_flag == 1'b1)) begin
            en_addr_data_temp1 <= 1'b1;
        end else begin
            en_addr_data_temp1 <= 1'b0;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            en_addr_inst_temp2 <= 1'b1;
        end else if((addr_inst == 6'b111111) & (po_flag_delay2 == 1'b1))
begin
```

```
            en_addr_inst_temp2 <= 1'b0;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            en_addr_data_temp2 <= 1'b0;
        end else if(start == 1'b1) begin
            en_addr_data_temp2 <= 1'b0;
        end else if((addr_data == 5'b11111) & (po_flag_delay2 == 1'b1))
begin
            en_addr_data_temp2 <= 1'b0;
        end else if(en_addr_inst_temp2 == 1'b0) begin
            en_addr_data_temp2 <= 1'b1;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            start <= 1'b0;
        end else if((addr_inst == 6'b111111) & (addr_data == 5'b11111) &
(en_addr_inst_temp2 == 1'b0) & (en_addr_data_temp2 == 1'b1) &
(po_flag_delay2 == 1'b1))begin
            start <= 1'b1;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            addr_inst <= 6'b111111;
        end else if((en_addr_inst_temp2 == 1'b1) & (po_flag == 1'b1))
begin
            addr_inst <= addr_inst + 6'b1;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            addr_data <= 5'b11111;
        end else if((en_addr_data_temp2 == 1'b1) & (po_flag == 1'b1))
begin
            addr_data <= addr_data + 5'b1;
        end
    end
```

```
always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        data_temp <= 32'b0;
    end else if(po_flag == 1'b1) begin
        data_temp <= {data_temp[23:0], po_data};
    end
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        po_flag_delay1 <= 1'b0;
    end else begin
        po_flag_delay1 <= po_flag;
    end
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        po_flag_delay2 <= 1'b0;
    end else begin
        po_flag_delay2 <= po_flag_delay1;
    end
end

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        rx_reg1 <= 1'b1;
    else
        rx_reg1 <= rx;

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        rx_reg2 <= 1'b1;
    else
        rx_reg2 <= rx_reg1;

always@(posedge sys_clk or negedge sys_arstn)
    if(sys_arstn == 1'b0)
        rx_reg3 <= 1'b1;
    else
        rx_reg3 <= rx_reg2;

always@(posedge sys_clk or negedge sys_arstn)
```

```
        if(sys_arstn == 1'b0)
            start_nedge <= 1'b0;
        else    if((~rx_reg2) && (rx_reg3))
            start_nedge <= 1'b1;
        else
            start_nedge <= 1'b0;

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            work_en <= 1'b0;
        else    if(start_nedge == 1'b1)
            work_en <= 1'b1;
        else    if((bit_cnt == 4'd8) && (bit_flag == 1'b1))
            work_en <= 1'b0;

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            baud_cnt <= 9'b0;
        else    if((baud_cnt == BAUD_CNT_MAX - 1) || (work_en == 1'b0))
            baud_cnt <= 9'b0;
        else    if(work_en == 1'b1)
            baud_cnt <= baud_cnt + 1'b1;

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            bit_flag <= 1'b0;
        else    if(baud_cnt == BAUD_CNT_MAX/2 - 1)
            bit_flag <= 1'b1;
        else
            bit_flag <= 1'b0;

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            bit_cnt <= 4'b0;
        else    if((bit_cnt == 4'd8) && (bit_flag == 1'b1))
            bit_cnt <= 4'b0;
        else    if(bit_flag ==1'b1)
            bit_cnt <= bit_cnt + 1'b1;

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            rx_data <= 8'b0;
        else    if((bit_cnt >= 4'd1)&&(bit_cnt <= 4'd8)&&(bit_flag ==
1'b1))
```

```
            rx_data <= {rx_reg3, rx_data[7:1]};

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            rx_flag <= 1'b0;
        else    if((bit_cnt == 4'd8) && (bit_flag == 1'b1))
            rx_flag <= 1'b1;
        else
            rx_flag <= 1'b0;

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            po_data <= 8'b0;
        else    if(rx_flag == 1'b1)
            po_data <= rx_data;

    always@(posedge sys_clk or negedge sys_arstn)
        if(sys_arstn == 1'b0)
            po_flag <= 1'b0;
        else
            po_flag <= rx_flag;

endmodule
```

16. *MODULE_ex_mem.v*

```
module MODULE_ex_mem (
    input           sys_clk         ,
    input           sys_arstn       ,
    input           flag_flush      ,
    input   [ 2:0]  flag_hold       ,
    input   [14:0]  mem_ctrl_i      ,
    input   [ 1:0]  flag_result_i   ,
    input   [31:0]  inst_addr_i     ,
    input   [31:0]  result_i        ,
    input   [31:0]  reg2_r_data_i   ,
    input   [ 1:0]  wb_ctrl_i       ,
    input   [ 4:0]  mem_Rd_i        ,
    output  [14:0]  mem_ctrl_o      ,
    output  [ 1:0]  flag_result_o   ,
    output  [31:0]  inst_addr_o     ,
    output  [31:0]  result_o        ,
    output  [31:0]  reg2_r_data_o   ,
    output  [ 1:0]  wb_ctrl_o       ,
    output  [ 4:0]  mem_Rd_o
);
```

```
    wire final_flush;
    wire final_hold;

    assign final_flush  = (flag_hold[1] == 1'b1) | (flag_flush ==
1'b1) ;
    assign final_hold   = (flag_hold[2] == 1'b1)                        ;

    FRAG_pipline #(
        15
    ) pipline_mem_ctrl(
        .sys_clk      (sys_clk          ),
        .sys_arstn    (sys_arstn        ),
        .flag_flush   (final_flush      ),
        .flag_hold    (final_hold       ),
        .default_value(15'b0            ),
        .data_in      (mem_ctrl_i       ),
        .data_out     (mem_ctrl_o       )
    );

    FRAG_pipline #(
        2
    ) pipline_flag_result(
        .sys_clk      (sys_clk          ),
        .sys_arstn    (sys_arstn        ),
        .flag_flush   (final_flush      ),
        .flag_hold    (final_hold       ),
        .default_value(2'b0             ),
        .data_in      (flag_result_i    ),
        .data_out     (flag_result_o    )
    );

    FRAG_pipline #(
        32
    ) pipline_inst_addr(
        .sys_clk      (sys_clk          ),
        .sys_arstn    (sys_arstn        ),
        .flag_flush   (final_flush      ),
        .flag_hold    (final_hold       ),
        .default_value(32'b0            ),
        .data_in      (inst_addr_i      ),
        .data_out     (inst_addr_o      )
    );
```

```
FRAG_pipline #(
    32
) pipline_result(
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .flag_flush   (final_flush      ),
    .flag_hold    (final_hold       ),
    .default_value(32'b0            ),
    .data_in      (result_i         ),
    .data_out     (result_o         )
);

FRAG_pipline #(
    32
) pipline_reg2_r_data(
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .flag_flush   (final_flush      ),
    .flag_hold    (final_hold       ),
    .default_value(32'b0            ),
    .data_in      (reg2_r_data_i    ),
    .data_out     (reg2_r_data_o    )
);

FRAG_pipline #(
    2
) pipline_wb_ctrl(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(2'b0         ),
    .data_in      (wb_ctrl_i    ),
    .data_out     (wb_ctrl_o    )
);

FRAG_pipline #(
    5
) pipline_mem_Rd(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(5'b0         ),
```

```
        .data_in      (mem_Rd_i     ),
        .data_out     (mem_Rd_o     )
    );

endmodule
```

17. $MODULE\_ex.v$

```
module MODULE_ex(
    input           sys_clk         ,
    input           sys_arstn       ,
    input           flag_JorB_i     ,
    input           funct7_5_i      ,
    input   [ 2:0]  funct3_i        ,
    input   [ 3:0]  ex_ctrl_i       ,
    input   [31:0]  reg1_r_data_i   ,
    input   [31:0]  reg2_r_data_i   ,
    input   [31:0]  imm_i           ,
    input   [31:0]  mem_data_i      ,
    input   [31:0]  wb_data_i       ,
    input   [ 4:0]  mem_Rd_i        ,
    input   [ 1:0]  mem_wb_ctrl_i   ,
    input   [ 4:0]  wb_Rd_i         ,
    input   [ 1:0]  wb_wb_ctrl_i    ,
    input   [ 4:0]  ex_Rs1_i        ,
    input   [ 4:0]  ex_Rs2_i        ,
    input   [31:0]  inst_addr_i     ,
    input   [14:0]  mem_ctrl_i      ,
    input   [ 1:0]  wb_ctrl_i       ,
    input   [ 4:0]  ex_Rd_i         ,
    output  [ 4:0]  ex_Rd_o         ,
    output  [14:0]  mem_ctrl_o      ,
    output  [ 1:0]  wb_ctrl_o       ,
    output  [31:0]  reg2_r_data_o   ,
    output  [31:0]  inst_addr_o     ,
    output  [31:0]  result_o        ,
    output  [ 1:0]  flag_result_o   ,
    output          hold_o
);

    wire [ 3:0] ALU_ctrl            ;
    wire [31:0] ALU_data_in_1       ;
    wire [31:0] ALU_data_in_2       ;
    wire [31:0] reg1_r_data         ;
    wire [31:0] reg2_r_data_1       ;
    wire [31:0] reg2_r_data_2       ;
```

```
wire [ 1:0] forward_Rs1_sel     ;
wire [ 1:0] forward_Rs2_sel     ;
wire [ 1:0] forward_store_sel   ;

assign ex_Rd_o          = ex_Rd_i       ;
assign mem_ctrl_o       = mem_ctrl_i    ;
assign wb_ctrl_o        = wb_ctrl_i     ;
assign reg2_r_data_o    = reg2_r_data_2 ;
assign inst_addr_o      = inst_addr_i   ;

FRAG_ALU_ctrl FRAG_ALU_ctrl_ex(
    .ALUOp   (ex_ctrl_i[3:2]),
    .funct7_5(funct7_5_i    ),
    .funct3  (funct3_i      ),
    .ALU_ctrl(ALU_ctrl      )
);

FRAG_ALU FRAG_ALU_ex(
    .sys_clk    (sys_clk      ),
    .sys_arstn  (sys_arstn    ),
    .flag_JorB  (flag_JorB_i  ),
    .data_in_0  (ALU_data_in_1),
    .data_in_1  (ALU_data_in_2),
    .ALU_ctrl   (ALU_ctrl     ),
    .result     (result_o     ),
    .flag_result(flag_result_o),
    .hold       (hold_o       )
);

FRAG_mux_2 FRAG_mux_2_Rs1_ex(
    .data_in_0(reg1_r_data  ),
    .data_in_1(32'h0        ),
    .select   (ex_ctrl_i[1] ),
    .data_out (ALU_data_in_1)
);

FRAG_mux_2 FRAG_mux_2_Rs2_ex(
    .data_in_0(reg2_r_data_1),
    .data_in_1(imm_i        ),
    .select   (ex_ctrl_i[0] ),
    .data_out (ALU_data_in_2)
);

FRAG_mux_3 FRAG_mux_3_Rs1_ex(
```

```
        .data_in_0(reg1_r_data_i    ),
        .data_in_1(mem_data_i       ),
        .data_in_2(wb_data_i        ),
        .select   (forward_Rs1_sel  ),
        .data_out (reg1_r_data      )
    );

    FRAG_mux_3 FRAG_mux_3_Rs2_ex(
        .data_in_0(reg2_r_data_i    ),
        .data_in_1(mem_data_i       ),
        .data_in_2(wb_data_i        ),
        .select   (forward_Rs2_sel  ),
        .data_out (reg2_r_data_1    )
    );

    FRAG_mux_3 FRAG_mux_3_store_ex(
        .data_in_0(reg2_r_data_1    ),
        .data_in_1(mem_data_i       ),
        .data_in_2(wb_data_i        ),
        .select   (forward_store_sel),
        .data_out (reg2_r_data_2    )
    );

    FRAG_forward FRAG_forward_ex(
        .mem_Rd             (mem_Rd_i           ),
        .mem_wb_ctrl        (mem_wb_ctrl_i      ),
        .wb_Rd              (wb_Rd_i            ),
        .wb_wb_ctrl         (wb_wb_ctrl_i       ),
        .ex_MemWrite        (mem_ctrl_i[5]      ),
        .ex_Rs1             (ex_Rs1_i           ),
        .ex_Rs2             (ex_Rs2_i           ),
        .forward_Rs1_sel    (forward_Rs1_sel    ),
        .forward_Rs2_sel    (forward_Rs2_sel    ),
        .forward_store_sel  (forward_store_sel  )
    );

endmodule
```

18. *MODULE_id_ex.v*

```
module MODULE_id_ex (
    input           sys_clk         ,
    input           sys_arstn       ,
    input           flag_flush      ,
    input   [ 2:0]  flag_hold       ,
    input           funct7_5_i      ,
```

```
    input   [ 2:0]  funct3_i        ,
    input   [ 3:0]  ex_ctrl_i       ,
    input   [31:0]  reg1_r_data_i   ,
    input   [31:0]  reg2_r_data_i   ,
    input   [31:0]  imm_i           ,
    input   [ 4:0]  ex_Rs1_i        ,
    input   [ 4:0]  ex_Rs2_i        ,
    input   [31:0]  inst_addr_i     ,
    input   [14:0]  mem_ctrl_i      ,
    input   [ 1:0]  wb_ctrl_i       ,
    input   [ 4:0]  ex_Rd_i         ,
    output          funct7_5_o      ,
    output  [ 2:0]  funct3_o        ,
    output  [ 3:0]  ex_ctrl_o       ,
    output  [31:0]  reg1_r_data_o   ,
    output  [31:0]  reg2_r_data_o   ,
    output  [31:0]  imm_o           ,
    output  [ 4:0]  ex_Rs1_o        ,
    output  [ 4:0]  ex_Rs2_o        ,
    output  [31:0]  inst_addr_o     ,
    output  [14:0]  mem_ctrl_o      ,
    output  [ 1:0]  wb_ctrl_o       ,
    output  [ 4:0]  ex_Rd_o
);

    wire final_flush;
    wire final_hold;

    assign final_flush  = (flag_hold[0] == 1'b1) | (flag_flush ==
1'b1)     ;
    assign final_hold   = (flag_hold[2] == 1'b1) | (flag_hold[1] ==
1'b1)   ;

    FRAG_pipline #(
        1
    ) pipline_funct7_5(
        .sys_clk      (sys_clk      ),
        .sys_arstn    (sys_arstn    ),
        .flag_flush   (final_flush  ),
        .flag_hold    (final_hold   ),
        .default_value(1'b0         ),
        .data_in      (funct7_5_i   ),
        .data_out     (funct7_5_o   )
    );
```

```
FRAG_pipline #(
    3
) pipline_funct3(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(3'b0         ),
    .data_in      (funct3_i     ),
    .data_out     (funct3_o     )
);

FRAG_pipline #(
    4
) pipline_ex_ctrl(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(4'b0001      ),
    .data_in      (ex_ctrl_i    ),
    .data_out     (ex_ctrl_o    )
);

FRAG_pipline #(
    32
) pipline_reg1_r_data(
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .flag_flush   (final_flush      ),
    .flag_hold    (final_hold       ),
    .default_value(32'b0            ),
    .data_in      (reg1_r_data_i    ),
    .data_out     (reg1_r_data_o    )
);

FRAG_pipline #(
    32
) pipline_reg2_r_data(
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .flag_flush   (final_flush      ),
    .flag_hold    (final_hold       ),
```

```
    .default_value(32'b0            ),
    .data_in      (reg2_r_data_i    ),
    .data_out     (reg2_r_data_o    )
);

FRAG_pipline #(
    32
) pipline_imm(
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .flag_flush   (final_flush      ),
    .flag_hold    (final_hold       ),
    .default_value(32'b0            ),
    .data_in      (imm_i            ),
    .data_out     (imm_o            )
);

FRAG_pipline #(
    5
) pipline_ex_Rs1(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(5'b0         ),
    .data_in      (ex_Rs1_i     ),
    .data_out     (ex_Rs1_o     )
);

FRAG_pipline #(
    5
) pipline_ex_Rs2(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(5'b0         ),
    .data_in      (ex_Rs2_i     ),
    .data_out     (ex_Rs2_o     )
);

FRAG_pipline #(
    32
) pipline_inst_addr(
```

```
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .flag_flush   (final_flush      ),
    .flag_hold    (final_hold       ),
    .default_value(32'b0            ),
    .data_in      (inst_addr_i      ),
    .data_out     (inst_addr_o      )
);

FRAG_pipline #(
    15
) pipline_mem_ctrl(
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .flag_flush   (final_flush      ),
    .flag_hold    (final_hold       ),
    .default_value(15'b0            ),
    .data_in      (mem_ctrl_i       ),
    .data_out     (mem_ctrl_o       )
);

FRAG_pipline #(
    2
) pipline_wb_ctrl(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(2'b10        ),
    .data_in      (wb_ctrl_i    ),
    .data_out     (wb_ctrl_o    )
);

FRAG_pipline #(
    5
) pipline_ex_Rd(
    .sys_clk      (sys_clk      ),
    .sys_arstn    (sys_arstn    ),
    .flag_flush   (final_flush  ),
    .flag_hold    (final_hold   ),
    .default_value(5'b0         ),
    .data_in      (ex_Rd_i      ),
    .data_out     (ex_Rd_o      )
);
```

```
endmodule
```

19. *MODULE_id.v*

```
module MODULE_id (
    input           sys_clk         ,
    input           sys_arstn       ,
    input           start_i         ,
    input           hold_i          ,
    input   [31:0]  inst_addr_i     ,
    input   [31:0]  inst_data_i     ,
    input           RegWrite_i      ,
    input   [ 4:0]  rd_i            ,
    input   [31:0]  w_data_i        ,
    input   [ 4:0]  ex_Rd_i         ,
    input           ex_MemRead_i    ,
    input           flag_JorB_i     ,
    output          flag_flush_o    ,
    output  [ 2:0]  flag_hold_o     ,
    output  [20:0]  ctrl_o          ,
    output          funct7_5_o      ,
    output  [ 2:0]  funct3_o        ,
    output  [31:0]  reg1_r_data_o   ,
    output  [31:0]  reg2_r_data_o   ,
    output  [31:0]  imm_o           ,
    output  [ 4:0]  Rs1_o           ,
    output  [ 4:0]  Rs2_o           ,
    output  [31:0]  inst_addr_o     ,
    output  [ 4:0]  Rd_o
);

    assign funct7_5_o   = inst_data_i[   30];
    assign funct3_o     = inst_data_i[14:12];
    assign Rs1_o        = inst_data_i[19:15];
    assign Rs2_o        = inst_data_i[24:20];
    assign Rd_o         = inst_data_i[11: 7];
    assign inst_addr_o  = inst_addr_i       ;

    FRAG_ctrl FRAG_ctrl_id(
        .opcode     (inst_data_i[6:0]    ),
        .funct3     (inst_data_i[14:12]  ),
        .funct7_0   (inst_data_i[25]     ),
        .ctrl       (ctrl_o             )
    );
```

```
    FRAG_register_file FRAG_register_file_id(
        .sys_clk    (sys_clk            ),
        .sys_arstn  (sys_arstn          ),
        .RegWrite   (RegWrite_i         ),
        .w_addr     (rd_i               ),
        .w_data     (w_data_i           ),
        .reg1_r_addr(inst_data_i[19:15]  ),
        .reg2_r_addr(inst_data_i[24:20]  ),
        .reg1_r_data(reg1_r_data_o      ),
        .reg2_r_data(reg2_r_data_o      )
    );

    FRAG_imm_gen FRAG_imm_gen_id(
        .inst_data(inst_data_i  ),
        .imm      (imm_o        )
    );

    FRAG_hazard FRAG_hazard_id(
        .sys_clk        (sys_clk            ),
        .sys_arstn      (sys_arstn          ),
        .start          (start_i            ),
        .hold           (hold_i             ),
        .ex_Rd          (ex_Rd_i            ),
        .ex_MemRead     (ex_MemRead_i       ),
        .id_Rs1         (inst_data_i[19:15]  ),
        .id_Rs2         (inst_data_i[24:20]  ),
        .flag_JorB      (flag_JorB_i        ),
        .flag_flush     (flag_flush_o       ),
        .flag_hold      (flag_hold_o        )
    );

endmodule
```

20. *MODULE_if_id.v*

```
module MODULE_if_id (
    input           sys_clk     ,
    input           sys_arstn   ,
    input           flag_flush  ,
    input   [ 2:0]  flag_hold   ,
    input   [31:0]  inst_data_i ,
    input   [31:0]  inst_addr_i ,
    output  [31:0]  inst_data_o ,
    output  [31:0]  inst_addr_o
);
```

```
    localparam NOP = 32'b00000000000000000000000000010011;

    wire        final_hold          ;
    reg         final_hold_delay1   ;
    reg [31:0]  inst_data_temp      ;

    assign final_hold   =   (flag_hold != 3'b000);
    assign inst_data_o  =   (sys_arstn == 1'b0) ? 32'h0 :
                            (flag_flush == 1'b1) ? NOP :
                            ((final_hold == 1'b1) & (final_hold_delay1 ==
1'b0)) ? inst_data_i :
                            (final_hold_delay1 == 1'b1) ?
inst_data_temp :
                            inst_data_i;

    FRAG_pipline pipline_inst_addr(
        .sys_clk      (sys_clk      )  ,
        .sys_arstn    (sys_arstn    )  ,
        .flag_flush   (flag_flush   )  ,
        .flag_hold    (final_hold   )  ,
        .default_value(32'b0        )  ,
        .data_in      (inst_addr_i  )  ,
        .data_out     (inst_addr_o  )
    );

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            final_hold_delay1 <= 32'h0;
        end else begin
            final_hold_delay1 <= final_hold;
        end
    end

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            inst_data_temp <= 32'h0;
        end else if(final_hold_delay1 == 1'b1) begin
            inst_data_temp <= inst_data_temp;
        end else begin
            inst_data_temp <= inst_data_i;
        end
    end

endmodule
```

21. $MODULE\_if.v$

```
module MODULE_if (
    input           sys_clk             ,
    input           sys_arstn           ,
    input           flag_JorB_i         ,
    input  [31:0]   inst_addr_JorB_i    ,
    input  [ 2:0]   flag_hold_i         ,
    input           rx_i                ,
    output [31:0]   inst_addr_o         ,
    output [31:0]   inst_data_o         ,
    output          en_addr_data_o      ,
    output [ 4:0]   addr_data_o         ,
    output [31:0]   data_o              ,
    output          start_o
);

    wire    [31:0]  inst_addr_in        ;
    reg     [31:0]  inst_addr_temp      ;
    wire    [31:0]  inst_addr_out       ;
    wire    [31:0]  inst_addr_add4      ;
    wire    [31:0]  inst_addr_add4_judge;
    wire            final_hold          ;
    reg             start_delay1        ;
    wire            flag_start          ;
    wire            en_addr_inst        ;
    wire    [ 5:0]  addr_inst           ;
    wire    [31:0]  data_out            ;

    assign inst_addr_o          = inst_addr_out             ;
    assign inst_addr_out        = inst_addr_temp            ;
    assign final_hold           = |flag_hold_i              ;
    assign flag_start           = start_o & (~start_delay1) ;
    assign data_o               = data_out                  ;

    always @(posedge sys_clk, negedge sys_arstn) begin
        if(sys_arstn == 1'b0) begin
            inst_addr_temp <= 32'hffffffff;
        end else if(flag_start == 1'b1) begin
            inst_addr_temp <= 32'h0;
        end else if(final_hold == 1'b1) begin
            inst_addr_temp <= inst_addr_temp;
        end else begin
            inst_addr_temp <= inst_addr_in;
        end
```

```
end

always @(posedge sys_clk, negedge sys_arstn) begin
    if(sys_arstn == 1'b0) begin
        start_delay1 <= 1'b0;
    end else begin
        start_delay1 <= start_o;
    end
end

FRAG_mux_2 FRAG_mux_2_0_if(
    .data_in_0(inst_addr_add4       ),
    .data_in_1(32'h40               ),
    .select   (inst_addr_add4[6]    ),
    .data_out (inst_addr_add4_judge )
);

FRAG_mux_2 FRAG_mux_2_1_if(
    .data_in_0(inst_addr_add4_judge ),
    .data_in_1(inst_addr_JorB_i     ),
    .select   (flag_JorB_i          ),
    .data_out (inst_addr_in         )
);

FRAG_adder FRAG_adder_if(
    .data_in_0(inst_addr_out    ),
    .data_in_1(32'h4            ),
    .data_out (inst_addr_add4   )
);

FRAG_inst_mem FRAG_inst_mem_if(
    .sys_clk        (sys_clk            ),
    .sys_arstn      (sys_arstn          ),
    .en_addr_inst   (en_addr_inst       ),
    .addr_inst      (addr_inst          ),
    .data_inst      (data_out           ),
    .addr           (inst_addr_out      ),
    .data           (inst_data_o        )
);

FRAG_UART FRAG_UART_if(
    .sys_clk     (sys_clk       ),
    .sys_arstn   (sys_arstn     ),
    .rx          (rx_i          ),
```

```
        .en_addr_inst(en_addr_inst  ),
        .en_addr_data(en_addr_data_o),
        .addr_inst   (addr_inst     ),
        .addr_data   (addr_data_o   ),
        .data        (data_out      ),
        .start       (start_o       )
    );

endmodule
```

22. *MODULE_mem_wb.v*

```
module MODULE_mem_wb (
    input           sys_clk        ,
    input           sys_arstn      ,
    input   [ 2:0]  flag_hold      ,
    input   [ 1:0]  wb_ctrl_i      ,
    input   [31:0]  wb_data_i      ,
    input   [31:0]  data_i         ,
    input   [ 4:0]  wb_Rd_i        ,
    output  [ 1:0]  wb_ctrl_o      ,
    output  [31:0]  wb_data_o      ,
    output  [31:0]  data_o         ,
    output  [ 4:0]  wb_Rd_o
);

    wire final_hold;

    assign final_hold   = (flag_hold[2] == 1'b1);
    assign data_o       = data_i                ;

    FRAG_pipline #(
        2
    ) pipline_wb_ctrl(
        .sys_clk      (sys_clk      ),
        .sys_arstn    (sys_arstn    ),
        .flag_flush   (1'b0         ),
        .flag_hold    (final_hold   ),
        .default_value(2'b0         ),
        .data_in      (wb_ctrl_i    ),
        .data_out     (wb_ctrl_o    )
    );

    FRAG_pipline #(
        32
    ) pipline_wb_data(
```

```
        .sys_clk      (sys_clk          ),
        .sys_arstn    (sys_arstn        ),
        .flag_flush   (1'b0             ),
        .flag_hold    (final_hold       ),
        .default_value(32'b0            ),
        .data_in      (wb_data_i        ),
        .data_out     (wb_data_o        )
    );

    FRAG_pipline #(
        5
    ) pipline_wb_Rd(
        .sys_clk      (sys_clk      ),
        .sys_arstn    (sys_arstn    ),
        .flag_flush   (1'b0         ),
        .flag_hold    (final_hold   ),
        .default_value(5'b0         ),
        .data_in      (wb_Rd_i      ),
        .data_out     (wb_Rd_o      )
    );

endmodule
```

23. *MODULE_mem.v*

```
module MODULE_mem(
    input           sys_clk         ,
    input           sys_arstn       ,
    input   [14:0]  mem_ctrl_i      ,
    input   [ 1:0]  flag_result_i   ,
    input   [31:0]  inst_addr_i     ,
    input   [31:0]  result_i        ,
    input   [31:0]  reg2_r_data_i   ,
    input   [ 1:0]  wb_ctrl_i       ,
    input   [ 4:0]  mem_Rd_i        ,
    input           en_addr_data_i  ,
    input   [ 4:0]  addr_data_i     ,
    input   [31:0]  data_data_i     ,
    output  [ 4:0]  mem_Rd_o        ,
    output  [ 1:0]  wb_ctrl_o       ,
    output          flag_JorB_o     ,
    output  [31:0]  inst_addr_JorB_o,
    output  [31:0]  data_o          ,
    output  [31:0]  mem_data_o      ,
    output          tx_o
);
```

```
    wire [31:0] inst_addr_addimm    ;
    wire [31:0] inst_addr_add4      ;
    wire [31:0] inst_addr_JorB      ;
    wire [ 1:0] sel                 ;
    wire        flag                ;

    assign mem_Rd_o         =
mem_Rd_i                                                    ;
    assign wb_ctrl_o        =
wb_ctrl_i                                                   ;
    assign inst_addr_JorB_o =
inst_addr_JorB                                              ;
    assign inst_addr_JorB   = (mem_ctrl_i[1:0] == 2'b11) ?
{result_i[31:1], 1'b0} : inst_addr_addimm;
    assign sel              = (mem_ctrl_i[1:0] == 2'b11) ? 2'b10 :
mem_ctrl_i                      ;
    assign flag             = (inst_addr_i ==
32'h40)                                         ;

    FRAG_adder FRAG_adder_addimm_mem(
        .data_in_0(inst_addr_i       ),
        .data_in_1(result_i          ),
        .data_out (inst_addr_addimm  )
    );

    FRAG_adder FRAG_adder_add4_mem(
        .data_in_0(inst_addr_i       ),
        .data_in_1(32'h4             ),
        .data_out (inst_addr_add4    )
    );

    FRAG_mux_3 FRAG_mux_3_mem(
        .data_in_0(result_i        ),
        .data_in_1(inst_addr_addimm),
        .data_in_2(inst_addr_add4  ),
        .select   (sel             ),
        .data_out (mem_data_o      )
    );

    FRAG_JorB_ctrl FRAG_JorB_ctrl_mem(
        .ctrl_JorB  (mem_ctrl_i[14:10]  ),
        .flag_result(flag_result_i      ),
        .flag_JorB  (flag_JorB_o        )
```

```
    );

    FRAG_data_mem FRAG_data_mem_mem(
        .sys_clk        (sys_clk        ),
        .sys_arstn      (sys_arstn      ),
        .ctrl           (mem_ctrl_i[9:2]),
        .en_addr_data   (en_addr_data_i ),
        .addr_data      (addr_data_i    ),
        .data_data      (data_data_i    ),
        .addr           (result_i       ),
        .data_in        (reg2_r_data_i  ),
        .flag           (flag           ),
        .data_out       (data_o         ),
        .tx             (tx_o           )
    );

endmodule
```

24. *MODULE_wb.v*

```
module MODULE_wb(
    input   [ 1:0]  wb_ctrl_i       ,
    input   [31:0]  wb_data_i       ,
    input   [31:0]  data_i          ,
    input   [ 4:0]  wb_Rd_i         ,
    output  [ 4:0]  wb_Rd_o         ,
    output          RegWrite_o      ,
    output  [31:0]  w_data_o
);
    assign wb_Rd_o      = wb_Rd_i                                   ;
    assign RegWrite_o   = wb_ctrl_i[1]                              ;
    assign w_data_o     = (wb_ctrl_i[0] == 1'b0) ? wb_data_i :
data_i   ;

endmodule
```

25. *on_the_fly_conversion.v*

```
module on_the_fly_conversion (
    input clk,
    input rst_n,

    input [3:0] q_in,
    input [1:0] state_in,

    output [31:0] q_out

);
```

```
  reg [31:0] qm_reg;
  reg [31:0] q_reg;

  wire q_in_0110 = (q_in == 4'b0110);
  wire q_in_0101 = (q_in == 4'b0101);
  wire q_in_0100 = (q_in == 4'b0100);
  wire q_in_0011 = (q_in == 4'b0011);
  wire q_in_0010 = (q_in == 4'b0010);
  wire q_in_0001 = (q_in == 4'b0001);
  wire q_in_0000 = (q_in[2:0] == 3'b000);
  wire q_in_1110 = (q_in == 4'b1110);
  wire q_in_1101 = (q_in == 4'b1101);
  wire q_in_1100 = (q_in == 4'b1100);
  wire q_in_1011 = (q_in == 4'b1011);
  wire q_in_1010 = (q_in == 4'b1010);
  wire q_in_1001 = (q_in == 4'b1001);

  wire active = (state_in == 2'b01);

  wire[31:0] q_next  =(active & q_in_0110) ? {q_reg[28:0] , 3'b110 } :
 (active & q_in_0101) ? {q_reg[28:0] , 3'b101 } :
(active & q_in_0100) ? {q_reg[28:0] , 3'b100 } :
(active & q_in_0011) ? {q_reg[28:0] , 3'b011 } :
                  (active & q_in_0010) ? {q_reg[28:0] , 3'b010 } :
            (active & q_in_0001) ? {q_reg[28:0] , 3'b001 } :
              (active & q_in_0000) ? {q_reg[28:0] , 3'b000 } :
                (active & q_in_1001) ? {qm_reg[28:0] , 3'b111 } :
                  (active & q_in_1010) ? {qm_reg[28:0] , 3'b110 } :
                  (active & q_in_1011) ? {qm_reg[28:0] , 3'b101 } :
                  (active & q_in_1100) ? {qm_reg[28:0] , 3'b100 } :
                  (active & q_in_1101) ? {qm_reg[28:0] , 3'b011 } :
                  (active & q_in_1110) ? {qm_reg[28:0] , 3'b010 } :
32'b0;
  wire[31:0] qm_next  =(active & q_in_0110) ? {q_reg[28:0] , 3'b101 } :
 (active & q_in_0101) ? {q_reg[28:0] , 3'b100 } :
(active & q_in_0100) ? {q_reg[28:0] , 3'b011 } :
(active & q_in_0011) ? {q_reg[28:0] , 3'b010 } :
                  (active & q_in_0010) ? {q_reg[28:0] , 3'b001 } :
            (active & q_in_0001) ? {q_reg[28:0] , 3'b000 } :
              (active & q_in_0000) ? {qm_reg[28:0] , 3'b111 } :
                (active & q_in_1001) ? {qm_reg[28:0] , 3'b110 } :
                  (active & q_in_1010) ? {qm_reg[28:0] , 3'b101 } :
                  (active & q_in_1011) ? {qm_reg[28:0] , 3'b100 } :
```

```
                (active & q_in_1100) ? {qm_reg[28:0] , 3'b011 } :
                (active & q_in_1101) ? {qm_reg[28:0] , 3'b010 } :
                (active & q_in_1110) ? {qm_reg[28:0] , 3'b001 } :
32'b0;


  //2bit shift reg
  always @(posedge clk, negedge rst_n) begin
    if (!rst_n) begin
      qm_reg <= 0;
      q_reg  <= 0;

    end else begin
      qm_reg <= qm_next;
      q_reg  <= q_next;

    end
  end

  assign q_out = q_reg;
endmodule
```

26. *pre_processing.v*

```
module pre_processing #(
    parameter DW = 32,
    parameter V  = 2,
    parameter K  = 2
) (

    input start,

    input [DW-1:0] dividend,
    input [DW-1:0] divisor,


    output [DW/2-1:0] iterations,
    output [  DW+2:0] divisor_star,
    output [  DW+5:0] dividend_star,
    output [DW/2-1:0] recovery

);

  reg [  DW+1:0] divisor_temp;
  reg [  DW+4:0] dividend_temp;
  reg [DW/2-1:0] recovery_temp;
  reg [DW/2-1:0] iterations_temp;
```

```
always @(*) begin
  if (start) begin
    casex (divisor)
      32'b1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
        divisor_temp    = {divisor, 1'b0, 1'b0};
        dividend_temp   = {2'b0, dividend, 1'b0, 1'b0, 1'b0};

        iterations_temp = 1;
        recovery_temp   = 31;
      end

      32'b01xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
        divisor_temp    = {divisor[30:0], 1'b0, 1'b0, 1'b0};
        dividend_temp   = {4'b0, dividend, 1'b0};  //>>1

        iterations_temp = 2;
        recovery_temp   = 30;
      end
      32'b001x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
        divisor_temp    = {divisor[29:0], 2'b00, 1'b0, 1'b0};
        dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

        iterations_temp = 2;
        recovery_temp   = 29;
      end
      32'b0001_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
        divisor_temp    = {divisor[28:0], 3'b000, 1'b0, 1'b0};
        dividend_temp   = {2'b0, dividend, 1'b0, 1'b0, 1'b0};

        iterations_temp = 2;
        recovery_temp   = 28;
      end
      32'b0000_1xxx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
        divisor_temp    = {divisor[27:0], 4'b0000, 1'b0, 1'b0};
        dividend_temp   = {4'b0, dividend, 1'b0};

        iterations_temp = 3;
        recovery_temp   = 27;
      end
      32'b0000_01xx_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
        divisor_temp    = {divisor[26:0], 5'b00000, 1'b0, 1'b0};
        dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};
```

```
  iterations_temp = 3;
  recovery_temp   = 26;
end
32'b0000_001x_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[25:0], 6'b000000, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 3;
  recovery_temp   = 25;
end
32'b0000_0001_xxxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[24:0], 7'b0000000, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 4;
  recovery_temp   = 24;
end
32'b0000_0000_1xxx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[23:0], 8'b00000000, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 4;
  recovery_temp   = 23;
end
32'b0000_0000_01xx_xxxx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[22:0], 9'b000000000, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 4;
  recovery_temp   = 22;
end
32'b0000_0000_001x_xxxx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[21:0], 10'b0, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 5;
  recovery_temp   = 21;
end
32'b0000_0000_0001_xxxx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[20:0], 11'b0, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 5;
```

```
  recovery_temp   = 20;
end
32'b0000_0000_0000_1xxx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[19:0], 12'b0, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 5;
  recovery_temp   = 19;
end
32'b0000_0000_0000_01xx_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[18:0], 13'b0, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 6;
  recovery_temp   = 18;
end
32'b0000_0000_0000_001x_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[17:0], 14'b0, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 6;
  recovery_temp   = 17;
end
32'b0000_0000_0000_0001_xxxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[16:0], 15'b0, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 6;
  recovery_temp   = 16;
end
32'b0000_0000_0000_0000_1xxx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[15:0], 16'b0, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 7;
  recovery_temp   = 15;
end
32'b0000_0000_0000_0000_01xx_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[14:0], 17'b0, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 7;
  recovery_temp   = 14;
end
```

```
32'b0000_0000_0000_0000_001x_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[13:0], 18'b0, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 7;
  recovery_temp   = 13;
end
32'b0000_0000_0000_0000_0001_xxxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[12:0], 19'b0, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 8;
  recovery_temp   = 12;
end
32'b0000_0000_0000_0000_0000_1xxx_xxxx_xxxx: begin
  divisor_temp    = {divisor[11:0], 20'b0, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 8;
  recovery_temp   = 11;
end
32'b0000_0000_0000_0000_0000_01xx_xxxx_xxxx: begin
  divisor_temp    = {divisor[10:0], 21'b0, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 8;
  recovery_temp   = 10;
end
32'b0000_0000_0000_0000_0000_001x_xxxx_xxxx: begin
  divisor_temp    = {divisor[9:0], 22'b0, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 9;
  recovery_temp   = 9;
end
32'b0000_0000_0000_0000_0000_0001_xxxx_xxxx: begin
  divisor_temp    = {divisor[8:0], 23'b0, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 9;
  recovery_temp   = 8;
end
32'b0000_0000_0000_0000_0000_0000_1xxx_xxxx: begin
  divisor_temp    = {divisor[7:0], 24'b0, 1'b0, 1'b0};
```

```
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 9;
  recovery_temp   = 7;
end
32'b0000_0000_0000_0000_0000_0000_01xx_xxxx: begin
  divisor_temp    = {divisor[6:0], 25'b0, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 10;
  recovery_temp   = 6;
end
32'b0000_0000_0000_0000_0000_0000_001x_xxxx: begin
  divisor_temp    = {divisor[5:0], 26'b0, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 10;
  recovery_temp   = 5;
end
32'b0000_0000_0000_0000_0000_0000_0001_xxxx: begin
  divisor_temp    = {divisor[4:0], 27'b0, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};

  iterations_temp = 10;
  recovery_temp   = 4;
end
32'b0000_0000_0000_0000_0000_0000_0000_1xxx: begin
  divisor_temp    = {divisor[3:0], 28'b0, 1'b0, 1'b0};
  dividend_temp   = {4'b0, dividend, 1'b0};

  iterations_temp = 11;
  recovery_temp   = 3;
end
32'b0000_0000_0000_0000_0000_0000_0000_01xx: begin
  divisor_temp    = {divisor[2:0], 29'b0, 1'b0, 1'b0};
  dividend_temp   = {3'b0, dividend, 1'b0, 1'b0};

  iterations_temp = 11;
  recovery_temp   = 2;
end
32'b0000_0000_0000_0000_0000_0000_0000_001x: begin
  divisor_temp    = {divisor[1:0], 30'b0, 1'b0, 1'b0};
  dividend_temp   = {2'b0, dividend, 2'b0, 1'b0};
```

```
          iterations_temp = 11;
          recovery_temp   = 1;
        end
        32'b0000_0000_0000_0000_0000_0000_0000_0001: begin
          divisor_temp    = {divisor[0], 31'b0, 1'b0, 1'b0};
          dividend_temp   = {4'b0, dividend, 1'b0};

          iterations_temp = 12;
          recovery_temp   = 0;
        end
        32'b0000_0000_0000_0000_0000_0000_0000_0000: begin
          divisor_temp    = 0;
          dividend_temp   = 0;

          iterations_temp = 0;
          recovery_temp   = 0;
        end
        default: begin
          divisor_temp    = 0;
          dividend_temp   = 0;

          iterations_temp = 0;
          recovery_temp   = 0;
        end
      endcase
    end else begin
      divisor_temp    = 0;
      dividend_temp   = 0;

      iterations_temp = 0;
      recovery_temp   = 0;
    end
  end

  assign dividend_star = {1'b0, dividend_temp};
  assign divisor_star  = {1'b0, divisor_temp};
  assign iterations    = iterations_temp;
  assign recovery      = recovery_temp;

endmodule
```

27. *r8_qds.v*

```
module r8_qds (
    input  [32+5:0] divisor_real,
    input  [   3:0] divisor_index,
```

```
    input  [32+5:0] w_reg,
    output [   2:0] q_table2,
    output [32+5:0] w_reg2,
    output [   3:0] q_table
);

  parameter DW = 32;
  wire [1:0] q_table1;
  wire [3:0] q_table2_temp = q_table2[2] ? {q_table2[2], (~q_table2[1:0]
+ 1)} : q_table2;
  //wire [2:0] q_table2;
  wire [3:0] q_2 = {q_table2_temp[2], q_table2_temp};
  wire [3:0]
q_1=(q_table1==2'b01)?4'd4:(q_table1==2'b10)?4'b1100:(q_table1==2'b00)?
4'd0:4'd0;

  wire [3:0] q_table_com = q_2 + q_1;
  assign q_table = q_table_com[3] ? {q_table_com[3], (~q_table_com[2:0]
+ 1)} : q_table_com;

  wire        [DW+5:0] divisor_4_real = divisor_real << 2;
  wire        [DW+5:0] divisor_4_neg = (~divisor_real + 1'b1) << 2;

  wire signed [   6:0] dividend_index1 = w_reg[DW+5:DW-1];  //

  assign  w_reg2 = (q_table1==2'b01)? divisor_4_neg + w_reg:
 (q_table1==2'b10)? divisor_4_real + w_reg:
 (q_table1==2'b00)?w_reg:
 w_reg;

  wire signed [6:0] dividend_index2 = w_reg2[DW+5:DW-1];
  wire        [2:0] dividend_expand = w_reg2[DW-2:DW-4];
  wire        [1:0] divisor_expand = divisor_real[DW-2:DW-3];

  //to tst
  wire        [2:0] dividend_expand_ex = w_reg2[DW-5:DW-7];
  wire        [1:0] divisor_expand2_ex = divisor_real[DW-4:DW-5];

  r8_table u1 (
      .dividend_index(dividend_index1),
      .divisor_index (divisor_index),

      .q_table1(q_table1)
```

```
  );

  radix4_table u2 (
      .dividend_index    (dividend_index2),
      .divisor_index     (divisor_index),
      .dividend_expand   (dividend_expand),
      .divisor_expand    (divisor_expand),
      .dividend_expand_ex(dividend_expand_ex),
      .divisor_expand_ex (divisor_expand2_ex),
      .q_table           (q_table2)

  );

endmodule
```

28. *r8_table.v*

```
module r8_table (
    input signed [6:0] dividend_index,
    input        [3:0] divisor_index,

    output [1:0] q_table1  //00 ->0;01 ->4;10 ->-4
);




  wire d_1000 = (divisor_index == 4'b1000);
  wire d_1001 = (divisor_index == 4'b1001);
  wire d_1010 = (divisor_index == 4'b1010);
  wire d_1011 = (divisor_index == 4'b1011);
  wire d_1100 = (divisor_index == 4'b1100);
  wire d_1101 = (divisor_index == 4'b1101);
  wire d_1110 = (divisor_index == 4'b1110);
  wire d_1111 = (divisor_index == 4'b1111);




  wire x_ge_20 = (dividend_index >= 20);
  wire x_ge_22 = (dividend_index >= 22);
  wire x_ge_25 = (dividend_index >= 25);
  wire x_ge_27 = (dividend_index >= 27);
  wire x_ge_30 = (dividend_index >= 30);
  wire x_ge_32 = (dividend_index >= 32);
  wire x_ge_35 = (dividend_index >= 35);
  wire x_ge_37 = (dividend_index >= 37);
```

```
wire x_ge_neg20 = (dividend_index >= -20);
wire x_ge_neg22 = (dividend_index >= -22);
wire x_ge_neg25 = (dividend_index >= -25);
wire x_ge_neg27 = (dividend_index >= -27);
wire x_ge_neg30 = (dividend_index >= -30);
wire x_ge_neg32 = (dividend_index >= -32);
wire x_ge_neg35 = (dividend_index >= -35);
wire x_ge_neg37 = (dividend_index >= -37);
wire x_ge_neg45 = (dividend_index >= -45);
wire x_ge_neg40 = (dividend_index >= -40);
wire x_ge_neg39 = (dividend_index >= -39);
wire x_ge_neg38 = (dividend_index >= -38);

wire d_1000_q_4 = (d_1000 & x_ge_20);
wire d_1000_q_0 = (d_1000 & ~x_ge_20 & x_ge_neg20);
wire d_1000_q_neg4 = (d_1000 & ~x_ge_neg20);

wire d_1001_q_4 = (d_1001 & x_ge_22);
wire d_1001_q_0 = (d_1001 & ~x_ge_22 & x_ge_neg22);
wire d_1001_q_neg4 = (d_1001 & ~x_ge_neg22);

wire d_1010_q_4 = (d_1010 & x_ge_25);
wire d_1010_q_0 = (d_1010 & ~x_ge_25 & x_ge_neg25);
wire d_1010_q_neg4 = (d_1010 & ~x_ge_neg25);

wire d_1011_q_4 = (d_1011 & x_ge_27);
wire d_1011_q_0 = (d_1011 & ~x_ge_27 & x_ge_neg27);
wire d_1011_q_neg4 = (d_1011 & ~x_ge_neg27);

wire d_1100_q_4 = (d_1100 & x_ge_30);
wire d_1100_q_0 = (d_1100 & ~x_ge_30 & x_ge_neg30);
wire d_1100_q_neg4 = (d_1100 & ~x_ge_neg30);

wire d_1101_q_4 = (d_1101 & x_ge_32);
wire d_1101_q_0 = (d_1101 & ~x_ge_32 & x_ge_neg32);
wire d_1101_q_neg4 = (d_1101 & ~x_ge_neg32);

wire d_1110_q_4 = (d_1110 & x_ge_35);
wire d_1110_q_0 = (d_1110 & ~x_ge_35 & x_ge_neg35);
wire d_1110_q_neg4 = (d_1110 & ~x_ge_neg35);

wire d_1111_q_4 = (d_1111 & x_ge_37);
```

```
  wire d_1111_q_0 = (d_1111 & ~x_ge_37 & x_ge_neg38);
  wire d_1111_q_neg4 = (d_1111 & ~x_ge_neg38);

  wire q_4 = d_1111_q_4 | d_1110_q_4 | d_1101_q_4 | d_1100_q_4
      | d_1011_q_4 | d_1010_q_4 | d_1001_q_4 | d_1000_q_4;

  wire q_0 = d_1111_q_0 | d_1110_q_0 | d_1101_q_0 | d_1100_q_0
      | d_1011_q_0 | d_1010_q_0 | d_1001_q_0 | d_1000_q_0;

  wire q_neg4 = d_1111_q_neg4 | d_1110_q_neg4 | d_1101_q_neg4 |
d_1100_q_neg4
      | d_1011_q_neg4 | d_1010_q_neg4 | d_1001_q_neg4 | d_1000_q_neg4;

  assign q_table1 = q_4 ? 2'b01 : q_neg4 ? 2'b10 : q_0 ? 2'b00 : 2'b00;


endmodule
```

29. *radix4_table.v*

```
module radix4_table (
    input  signed [6:0] dividend_index,
    input         [3:0] divisor_index,
    input         [2:0] dividend_expand,
    input         [1:0] divisor_expand,
    input         [2:0] dividend_expand_ex,
    input         [1:0] divisor_expand_ex,
    output        [2:0] q_table

);

  wire dividend_index_neg = dividend_index[6];

  wire dividend_index_fix = dividend_index_neg ? ~dividend_index + 1 :
dividend_index;


  wire d_1000 = (divisor_index == 4'b1000);
  wire d_1001 = (divisor_index == 4'b1001);
  wire d_1010 = (divisor_index == 4'b1010);
  wire d_1011 = (divisor_index == 4'b1011);
  wire d_1100 = (divisor_index == 4'b1100);
  wire d_1101 = (divisor_index == 4'b1101);
  wire d_1110 = (divisor_index == 4'b1110);
  wire d_1111 = (divisor_index == 4'b1111);
```

```
wire x_ge_12 = (dividend_index >= 12);
wire x_ge_14 = (dividend_index >= 14);
wire x_ge_15 = (dividend_index >= 15);
wire x_ge_16 = (dividend_index >= 16);
wire x_ge_18 = (dividend_index >= 18);
wire x_ge_20 = (dividend_index >= 20);
wire x_ge_24 = (dividend_index >= 24);
wire x_ge_22 = (dividend_index >= 22);
wire x_ge_21 = (dividend_index >= 21);
wire x_ge_23 = (dividend_index >= 23);
wire x_ge_4 = (dividend_index >= 4);
wire x_ge_6 = (dividend_index >= 6);
wire x_ge_8 = (dividend_index >= 8);

wire x_ge_neg4 = (dividend_index >= -4);
wire x_ge_neg6 = (dividend_index >= -6);
wire x_ge_neg8 = (dividend_index >= -8);

wire x_ge_neg13 = (dividend_index >= -13);
wire x_ge_neg15 = (dividend_index >= -15);
wire x_ge_neg16 = (dividend_index >= -16);
wire x_ge_neg18 = (dividend_index >= -18);
wire x_ge_neg20 = (dividend_index >= -20);
wire x_ge_neg22 = (dividend_index >= -22);
wire x_ge_neg24 = (dividend_index >= -24);
wire x_ge_neg23 = (dividend_index >= -23);
wire x_ge_neg25 = (dividend_index >= -25);


wire d_1000_q_2 = (d_1000 & x_ge_12);
wire d_1000_q_1 = (d_1000 & x_ge_4 & ~x_ge_12);
wire d_1000_q_0 = (d_1000 & ~x_ge_4 & x_ge_neg4);
wire d_1000_q_neg1 = (d_1000 & x_ge_neg13 & ~x_ge_neg4);
wire d_1000_q_neg2 = (d_1000 & ~x_ge_neg13);


wire d_1001_q_2 = (d_1001 & x_ge_14);
wire d_1001_q_1 = (d_1001 & x_ge_4 & ~x_ge_14);
wire d_1001_q_0 = (d_1001 & x_ge_neg6 & ~x_ge_4);
wire d_1001_q_neg1 = (d_1001 & x_ge_neg15 & ~x_ge_neg6);
wire d_1001_q_neg2 = (d_1001 & ~x_ge_neg15);

wire d_1010_q_2 = (d_1010 & x_ge_15);
wire d_1010_q_1 = (d_1010 & x_ge_4 & ~x_ge_15);
wire d_1010_q_0 = (d_1010 & x_ge_neg6 & ~x_ge_4);
```

```
  wire d_1010_q_neg1 = (d_1010 & x_ge_neg16 & ~x_ge_neg6);
  wire d_1010_q_neg2 = (d_1010 & ~x_ge_neg16);

  wire d_1011_q_2 = (d_1011 & x_ge_16);
  wire d_1011_q_1 = (d_1011 & x_ge_4 & ~x_ge_16);
  wire d_1011_q_0 = (d_1011 & x_ge_neg6 & ~x_ge_4);
  wire d_1011_q_neg1 = (d_1011 & x_ge_neg18 & ~x_ge_neg6);
  wire d_1011_q_neg2 = (d_1011 & ~x_ge_neg18);

  wire d_1100_q_2 = (d_1100 & x_ge_18);
  wire d_1100_q_1 = (d_1100 & x_ge_6 & ~x_ge_18);
  wire d_1100_q_0 = (d_1100 & x_ge_neg8 & ~x_ge_6);
  wire d_1100_q_neg1 = (d_1100 & x_ge_neg20 & ~x_ge_neg8);
  wire d_1100_q_neg2 = (d_1100 & ~x_ge_neg20);

  wire d_1101_q_2 = (d_1101 & x_ge_20);
  wire d_1101_q_1 = (d_1101 & x_ge_6 & ~x_ge_20);
  wire d_1101_q_0 = (d_1101 & x_ge_neg8 & ~x_ge_6);
  wire d_1101_q_neg1 = (d_1101 & x_ge_neg20 & ~x_ge_neg8);
  wire d_1101_q_neg2 = (d_1101 & ~x_ge_neg20);

  wire d_1110_q_2 = (d_1110 & x_ge_22);
  wire d_1110_q_1 = (d_1110 & x_ge_8 & ~x_ge_22);
  wire d_1110_q_0 = (d_1110 & x_ge_neg8 & ~x_ge_8);
  wire d_1110_q_neg1 = (d_1110 & x_ge_neg22 & ~x_ge_neg8);
  wire d_1110_q_neg2 = (d_1110 & ~x_ge_neg22);
  //23/15-->2
  wire sel1=(divisor_expand==0)|((dividend_expand >=
2)&(divisor_expand==1))|
((dividend_expand >=4)&(divisor_expand==2))|
((divisor_expand==3)&(
((dividend_expand ==7)&(  ((divisor_expand_ex==1))    |
((divisor_expand_ex==3)&(dividend_expand_ex>=4))
|  ((divisor_expand_ex==2))  |((divisor_expand_ex==0))   )  )|
(((dividend_expand
==6))&(  ((divisor_expand_ex==0))   |   ((dividend_expand_ex >=3)&(divis
or_expand_ex==1))   )       )      ));//|((dividend_expand
==7)&(divisor_expand==3));


  //-24/15-->-1
  wire sel2=((dividend_expand >= 4)&(divisor_expand==2))|(
(   (   ((divisor_expand_ex==3)&({dividend_expand[0],dividend_expand_ex}
>=5)) |
```

```
((divisor_expand_ex==2)&({dividend_expand[0],dividend_expand_ex} >=10))
 )   |   (dividend_expand>=2)   )   &(divisor_expand==3)
);

  wire d_1111_q_2 = sel1 ? (d_1111 & x_ge_23) : (d_1111 & x_ge_24);  //
  wire d_1111_q_1 = sel1 ? (d_1111 & x_ge_8 & ~x_ge_23) : (d_1111 &
x_ge_8 & ~x_ge_24);
  wire d_1111_q_0 = (d_1111 & x_ge_neg8 & ~x_ge_8);
  wire d_1111_q_neg1 = sel2?(d_1111 & x_ge_neg24 & ~x_ge_neg8 ):(d_1111
& x_ge_neg23 & ~x_ge_neg8 ) ;
  wire d_1111_q_neg2 = sel2 ? (d_1111 & ~x_ge_neg24) : (d_1111 &
~x_ge_neg23);


  wire q_2 = d_1111_q_2 | d_1110_q_2 | d_1101_q_2 | d_1100_q_2
      | d_1011_q_2 | d_1010_q_2 | d_1001_q_2 | d_1000_q_2;

  wire q_1 = d_1111_q_1 | d_1110_q_1 | d_1101_q_1 | d_1100_q_1
      | d_1011_q_1 | d_1010_q_1 | d_1001_q_1 | d_1000_q_1;

  wire q_0 = d_1111_q_0 | d_1110_q_0 | d_1101_q_0 | d_1100_q_0
      | d_1011_q_0 | d_1010_q_0 | d_1001_q_0 | d_1000_q_0;

  wire q_neg1 = d_1111_q_neg1 | d_1110_q_neg1 | d_1101_q_neg1 |
d_1100_q_neg1
      | d_1011_q_neg1 | d_1010_q_neg1 | d_1001_q_neg1 | d_1000_q_neg1;

  wire q_neg2 = d_1111_q_neg2 | d_1110_q_neg2 | d_1101_q_neg2 |
d_1100_q_neg2
      | d_1011_q_neg2 | d_1010_q_neg2 | d_1001_q_neg2 | d_1000_q_neg2;

  assign q_table = {
    (q_neg2 | q_neg1), ((q_2 | q_neg2) ? 2'b10 : (q_1 | q_neg1) ?
2'b01 : q_0 ? 2'b00 : 2'b00)
  };



endmodule
```

30. *srt_8_div.v*

```
module SRT_8_div #(
    parameter DW = 32
) (
    input clk,
    input rst_n,
```

```
    input start,

    input [DW-1:0] dividend_i,
    input [DW-1:0] divisor_i,
    input          sign_define,

    output [DW-1:0] quotient_o,
    output [DW-1:0] reminder_o,
    output          mulfinish
);
  wire [DW-1:0] quotient;
  wire [DW-1:0] reminder;


  wire [DW-
1:0]   dividend=sign_define?(dividend_i[31]?(~dividend_i+1):dividend_i):
dividend_i ;
  wire [DW-1:0]     divisor=
sign_define?(divisor_i[31]?(~divisor_i+1):divisor_i):divisor_i ;

  wire [1:0] sign_flag = sign_define ? {dividend_i[31], divisor_i[31]} :
2'b00;


  wire [DW/2-1:0] iterations;
  wire [DW+2:0] divisor_star;
  wire [DW+5:0] dividend_star;
  wire [DW/2-1:0] recovery;



  pre_processing u1 (

      .start(start),

      .dividend(dividend),
      .divisor (divisor),


      .iterations   (iterations),
      .divisor_star (divisor_star),
      .dividend_star(dividend_star),
      .recovery     (recovery)
```

```
  );

  wire [DW+5:0] w_0_4 = dividend_star[DW+5:0];

  localparam idle = 2'b00, div = 2'b01, finish = 2'b10, error = 2'b11;
  reg [1:0] state, state_next;
  reg [DW/4:0] counter, counter_next;
  reg [DW+5:0] w_reg, w_temp;
  reg [DW+5:0] divisor_reg, divisor_temp;
  reg [DW/2-1:0] iterations_temp, iterations_reg, recovery_temp,
recovery_reg;

  wire                                [   1:0] state_in = state;

  wire                                [DW+5:0] divisor_real =
divisor_reg;
  wire                                [DW+5:0] divisor_2_real =
divisor_real << 1;
  wire                                [DW+5:0] divisor_neg =
~divisor_real + 1'b1;
  wire                                [DW+5:0] divisor_2_neg =
divisor_neg << 1;



  wire dividend_eq_0_t = (dividend == 0);
  //wire divisor_eq_0_t = (divisor == 0);

  wire signed                          [   6:0] dividend_index =
w_reg[DW+5:DW-1];
  wire                                [   3:0] divisor_index =
divisor_reg[DW+2:DW-1];
  wire                                [   2:0]
              q_table;
  wire                                [   3:0]
              q_in;
  wire                                [32+5:0]
              w_reg2;
  r8_qds r8_qds (
      .divisor_real (divisor_real),
      .divisor_index(divisor_index),
      .w_reg        (w_reg),
      .q_table2     (q_table),
```

```
    .w_reg2       (w_reg2),
    .q_table      (q_in)

);



  wire[DW+5:0] w_next_temp = (q_table == 3'b001) ? divisor_neg +
w_reg2 :
               (q_table == 3'b010) ? w_reg2 + divisor_2_neg :
                 (q_table == 3'b101) ? w_reg2 + divisor_real :
                   (q_table == 3'b110) ? w_reg2 + divisor_2_real :
                     (q_table ==3'b000) ?  w_reg2 : 12'b0 ;

  wire [DW+5:0] w_next = w_next_temp << 3;


  //商生?


  wire [DW-1:0] q_out_1;

  on_the_fly_conversion u3 (
    .clk  (clk),
    .rst_n(rst_n),

    .q_in    (q_in),
    .state_in(state_in),

    .q_out(q_out_1)


  );

  always @(posedge clk, negedge rst_n) begin
    if (!rst_n) begin
      state          <= idle;
      divisor_reg    <= 0;
      counter        <= 0;
      w_reg          <= 0;
      iterations_reg <= 0;
      recovery_reg   <= 0;

    end else begin
      state          <= state_next;
```

```
    divisor_reg    <= divisor_temp;
    counter        <= counter_next;
    w_reg          <= w_temp;
    iterations_reg <= iterations_temp;
    recovery_reg   <= recovery_temp;


  end
end


always @(*) begin
  case (state)
    idle: begin
      if (start & ~dividend_eq_0_t )//& ~divisor_eq_0_t)
    begin
        state_next      = div;
        divisor_temp    = {2'b0, divisor_star, 1'b0};
        w_temp          = w_0_4;  //there
        counter_next    = 0;
        iterations_temp = iterations;
        recovery_temp   = recovery;

      end else if (start & dividend_eq_0_t) begin
        state_next      = finish;
        divisor_temp    = 0;
        w_temp          = 0;
        counter_next    = 0;
        iterations_temp = 0;
        recovery_temp   = 0;
      end else
    if (start )//& divisor_eq_0_t)
    begin
        state_next      = error;
        divisor_temp    = 0;
        w_temp          = 0;
        counter_next    = 0;
        iterations_temp = 0;
        recovery_temp   = 0;
      end else begin
        state_next      = idle;
        divisor_temp    = 0;
        w_temp          = 0;
        counter_next    = 0;
        iterations_temp = 0;
```

```
        recovery_temp   = 0;


      end
    end

    div: begin
      if (counter != iterations_reg - 1) begin
        state_next      = div;
        w_temp          = w_next;
        divisor_temp    = divisor_reg;
        counter_next    = counter + 1'b1;
        iterations_temp = iterations_reg;
        recovery_temp   = recovery_reg;

      end else begin
        state_next      = finish;
        w_temp          = w_next_temp;
        divisor_temp    = divisor_reg;
        counter_next    = 0;
        iterations_temp = iterations_reg;
        recovery_temp   = recovery_reg;
      end

    end

    finish: begin
      state_next      = idle;
      divisor_temp    = 0;
      counter_next    = 0;
      w_temp          = 0;
      iterations_temp = 0;
      recovery_temp   = 0;
    end
    error: begin
      state_next      = idle;
      divisor_temp    = 0;
      counter_next    = 0;
      w_temp          = 0;
      iterations_temp = 0;
      recovery_temp   = 0;
    end
  endcase
end
```

```
  wire w_reg_unsign = (w_reg[DW+3] == 1);
  wire [DW+5:0] w_reg_fix = w_reg_unsign ? w_reg + divisor_real : w_reg;
  wire [DW-1:0] q_out_fix = w_reg_unsign ? q_out_1 - 1 : q_out_1;
  wire diverror;
  wire [DW+32:0] reminder_temp = ({27'b0, w_reg_fix} << (recovery_reg -
1));
  assign reminder   = reminder_temp[DW+32:DW+1];
  assign quotient   = q_out_fix;
  assign reminder_o = sign_flag[1] ? (~reminder + 1) : reminder;
  assign quotient_o = (sign_flag[1] != sign_flag[0]) ? (~quotient + 1) :
quotient;
  assign diverror   = (state == error);
  assign mulfinish  = (state == finish);

endmodule
```

31. $TOP.v$

```
module TOP (
    input   sys_clk  ,
    input   sys_arstn,
    input   rx       ,
    output  tx
);

    wire        if_flag_JorB_i          ;
    wire [31:0] if_inst_addr_JorB_i     ;
    wire [ 2:0] if_flag_hold_i          ;
    wire [31:0] if_inst_addr_o          ;
    wire [31:0] if_inst_data_o          ;
    wire        if_en_addr_data_o       ;
    wire [ 4:0] if_addr_data_o          ;
    wire [31:0] if_data_o               ;
    wire        if_start_o              ;

    wire        if_id_flag_flush        ;
    wire [ 2:0] if_id_flag_hold         ;
    wire [31:0] if_id_inst_data_i       ;
    wire [31:0] if_id_inst_addr_i       ;
    wire [31:0] if_id_inst_data_o       ;
    wire [31:0] if_id_inst_addr_o       ;

    wire        id_start_i              ;
    wire        id_hold_i               ;
    wire [31:0] id_inst_addr_i          ;
    wire [31:0] id_inst_data_i          ;
```

```
wire        id_RegWrite_i          ;
wire [ 4:0] id_rd_i                ;
wire [31:0] id_w_data_i            ;
wire [ 4:0] id_ex_Rd_i             ;
wire        id_ex_MemRead_i        ;
wire        id_flag_JorB_i         ;
wire        id_flag_flush_o        ;
wire [ 2:0] id_flag_hold_o         ;
wire [20:0] id_ctrl_o              ;
wire        id_funct7_5_o          ;
wire [ 2:0] id_funct3_o            ;
wire [31:0] id_reg1_r_data_o       ;
wire [31:0] id_reg2_r_data_o       ;
wire [31:0] id_imm_o               ;
wire [ 4:0] id_Rs1_o               ;
wire [ 4:0] id_Rs2_o               ;
wire [31:0] id_inst_addr_o         ;
wire [ 4:0] id_Rd_o                ;

wire        id_ex_flag_flush       ;
wire [ 2:0] id_ex_flag_hold        ;
wire        id_ex_funct7_5_i       ;
wire [ 2:0] id_ex_funct3_i         ;
wire [ 3:0] id_ex_ex_ctrl_i        ;
wire [31:0] id_ex_reg1_r_data_i    ;
wire [31:0] id_ex_reg2_r_data_i    ;
wire [31:0] id_ex_imm_i            ;
wire [ 4:0] id_ex_ex_Rs1_i         ;
wire [ 4:0] id_ex_ex_Rs2_i         ;
wire [31:0] id_ex_inst_addr_i      ;
wire [14:0] id_ex_mem_ctrl_i       ;
wire [ 1:0] id_ex_wb_ctrl_i        ;
wire [ 4:0] id_ex_ex_Rd_i          ;
wire        id_ex_funct7_5_o       ;
wire [ 2:0] id_ex_funct3_o         ;
wire [ 3:0] id_ex_ex_ctrl_o        ;
wire [31:0] id_ex_reg1_r_data_o    ;
wire [31:0] id_ex_reg2_r_data_o    ;
wire [31:0] id_ex_imm_o            ;
wire [ 4:0] id_ex_ex_Rs1_o         ;
wire [ 4:0] id_ex_ex_Rs2_o         ;
wire [31:0] id_ex_inst_addr_o      ;
wire [14:0] id_ex_mem_ctrl_o       ;
wire [ 1:0] id_ex_wb_ctrl_o        ;
```

```
wire [ 4:0] id_ex_ex_Rd_o          ;

wire        ex_flag_JorB_i         ;
wire        ex_funct7_5_i          ;
wire [ 2:0] ex_funct3_i            ;
wire [ 3:0] ex_ex_ctrl_i           ;
wire [31:0] ex_reg1_r_data_i       ;
wire [31:0] ex_reg2_r_data_i       ;
wire [31:0] ex_imm_i               ;
wire [31:0] ex_mem_data_i          ;
wire [31:0] ex_wb_data_i           ;
wire [ 4:0] ex_mem_Rd_i            ;
wire [ 1:0] ex_mem_wb_ctrl_i_ex    ;
wire [ 4:0] ex_wb_Rd_i             ;
wire [ 1:0] ex_wb_wb_ctrl_i        ;
wire [ 4:0] ex_ex_Rs1_i            ;
wire [ 4:0] ex_ex_Rs2_i            ;
wire [31:0] ex_inst_addr_i         ;
wire [14:0] ex_mem_ctrl_i          ;
wire [ 1:0] ex_wb_ctrl_i           ;
wire [ 4:0] ex_ex_Rd_i             ;
wire [ 4:0] ex_ex_Rd_o             ;
wire [14:0] ex_mem_ctrl_o          ;
wire [ 1:0] ex_wb_ctrl_o           ;
wire [31:0] ex_reg2_r_data_o       ;
wire [31:0] ex_inst_addr_o         ;
wire [31:0] ex_result_o            ;
wire [ 1:0] ex_flag_result_o       ;
wire        ex_hold_o              ;

wire        ex_mem_flag_flush      ;
wire [ 2:0] ex_mem_flag_hold       ;
wire [14:0] ex_mem_mem_ctrl_i      ;
wire [ 1:0] ex_mem_flag_result_i   ;
wire [31:0] ex_mem_inst_addr_i     ;
wire [31:0] ex_mem_result_i        ;
wire [31:0] ex_mem_reg2_r_data_i   ;
wire [ 1:0] ex_mem_wb_ctrl_i_ex_mem ;
wire [ 4:0] ex_mem_mem_Rd_i        ;
wire [14:0] ex_mem_mem_ctrl_o      ;
wire [ 1:0] ex_mem_flag_result_o   ;
wire [31:0] ex_mem_inst_addr_o     ;
wire [31:0] ex_mem_result_o        ;
wire [31:0] ex_mem_reg2_r_data_o   ;
```

```
wire [ 1:0] ex_mem_wb_ctrl_o        ;
wire [ 4:0] ex_mem_mem_Rd_o         ;

wire [14:0] mem_mem_ctrl_i          ;
wire [ 1:0] mem_flag_result_i       ;
wire [31:0] mem_inst_addr_i         ;
wire [31:0] mem_result_i            ;
wire [31:0] mem_reg2_r_data_i       ;
wire [ 1:0] mem_wb_ctrl_i           ;
wire [ 4:0] mem_mem_Rd_i            ;
wire        mem_en_addr_data_i      ;
wire [ 4:0] mem_addr_data_i         ;
wire [31:0] mem_data_data_i         ;
wire [ 4:0] mem_mem_Rd_o            ;
wire [ 1:0] mem_wb_ctrl_o           ;
wire        mem_flag_JorB_o         ;
wire [31:0] mem_inst_addr_JorB_o    ;
wire [31:0] mem_data_o              ;
wire [31:0] mem_mem_data_o          ;

wire [ 2:0] mem_wb_flag_hold        ;
wire [ 1:0] mem_wb_wb_ctrl_i        ;
wire [31:0] mem_wb_wb_data_i        ;
wire [31:0] mem_wb_data_i           ;
wire [ 4:0] mem_wb_wb_Rd_i          ;
wire [ 1:0] mem_wb_wb_ctrl_o        ;
wire [31:0] mem_wb_wb_data_o        ;
wire [31:0] mem_wb_data_o           ;
wire [ 4:0] mem_wb_wb_Rd_o          ;

wire [ 1:0] wb_wb_ctrl_i            ;
wire [31:0] wb_wb_data_i            ;
wire [31:0] wb_data_i               ;
wire [ 4:0] wb_wb_Rd_i              ;
wire [ 4:0] wb_wb_Rd_o              ;
wire        wb_RegWrite_o           ;
wire [31:0] wb_w_data_o             ;

assign if_flag_JorB_i          = mem_flag_JorB_o         ;
assign if_inst_addr_JorB_i     = mem_inst_addr_JorB_o    ;
assign if_flag_hold_i          = id_flag_hold_o          ;

assign if_id_flag_flush        = id_flag_flush_o         ;
assign if_id_flag_hold         = id_flag_hold_o          ;
```

```
assign if_id_inst_data_i      = if_inst_data_o         ;
assign if_id_inst_addr_i      = if_inst_addr_o         ;

assign id_start_i             = if_start_o             ;
assign id_hold_i              = ex_hold_o              ;
assign id_inst_addr_i         = if_id_inst_addr_o      ;
assign id_inst_data_i         = if_id_inst_data_o      ;
assign id_RegWrite_i          = wb_RegWrite_o          ;
assign id_rd_i                = wb_wb_Rd_o             ;
assign id_w_data_i            = wb_w_data_o            ;
assign id_ex_Rd_i             = ex_ex_Rd_o             ;
assign id_ex_MemRead_i        = ex_mem_ctrl_o[9]       ;
assign id_flag_JorB_i         = mem_flag_JorB_o        ;

assign id_ex_flag_flush       = id_flag_flush_o        ;
assign id_ex_flag_hold        = id_flag_hold_o         ;
assign id_ex_funct7_5_i       = id_funct7_5_o          ;
assign id_ex_funct3_i         = id_funct3_o            ;
assign id_ex_ex_ctrl_i        = id_ctrl_o[20:17]       ;
assign id_ex_reg1_r_data_i    = id_reg1_r_data_o       ;
assign id_ex_reg2_r_data_i    = id_reg2_r_data_o       ;
assign id_ex_imm_i            = id_imm_o               ;
assign id_ex_ex_Rs1_i         = id_Rs1_o               ;
assign id_ex_ex_Rs2_i         = id_Rs2_o               ;
assign id_ex_inst_addr_i      = id_inst_addr_o         ;
assign id_ex_mem_ctrl_i       = id_ctrl_o[16:2]        ;
assign id_ex_wb_ctrl_i        = id_ctrl_o[1:0]         ;
assign id_ex_ex_Rd_i          = id_Rd_o                ;

assign ex_flag_JorB_i         = mem_flag_JorB_o        ;
assign ex_funct7_5_i          = id_ex_funct7_5_o       ;
assign ex_funct3_i            = id_ex_funct3_o         ;
assign ex_ex_ctrl_i           = id_ex_ex_ctrl_o        ;
assign ex_reg1_r_data_i       = id_ex_reg1_r_data_o    ;
assign ex_reg2_r_data_i       = id_ex_reg2_r_data_o    ;
assign ex_imm_i               = id_ex_imm_o            ;
assign ex_mem_data_i          = mem_mem_data_o         ;
assign ex_wb_data_i           = wb_w_data_o            ;
assign ex_mem_Rd_i            = ex_mem_mem_Rd_o        ;
assign ex_mem_wb_ctrl_i_ex    = ex_mem_wb_ctrl_o       ;
assign ex_wb_Rd_i             = mem_wb_wb_Rd_o         ;
assign ex_wb_wb_ctrl_i        = mem_wb_wb_ctrl_o       ;
assign ex_ex_Rs1_i            = id_ex_ex_Rs1_o         ;
assign ex_ex_Rs2_i            = id_ex_ex_Rs2_o         ;
```

```
assign ex_inst_addr_i          = id_ex_inst_addr_o       ;
assign ex_mem_ctrl_i           = id_ex_mem_ctrl_o        ;
assign ex_wb_ctrl_i            = id_ex_wb_ctrl_o         ;
assign ex_ex_Rd_i              = id_ex_ex_Rd_o           ;

assign ex_mem_flag_flush       = id_flag_flush_o         ;
assign ex_mem_flag_hold        = id_flag_hold_o          ;
assign ex_mem_mem_ctrl_i       = ex_mem_ctrl_o           ;
assign ex_mem_flag_result_i    = ex_flag_result_o        ;
assign ex_mem_inst_addr_i      = ex_inst_addr_o          ;
assign ex_mem_result_i         = ex_result_o             ;
assign ex_mem_reg2_r_data_i    = ex_reg2_r_data_o        ;
assign ex_mem_wb_ctrl_i_ex_mem = ex_wb_ctrl_o            ;
assign ex_mem_mem_Rd_i         = ex_ex_Rd_o              ;

assign mem_mem_ctrl_i          = ex_mem_mem_ctrl_o       ;
assign mem_flag_result_i       = ex_mem_flag_result_o    ;
assign mem_inst_addr_i         = ex_mem_inst_addr_o      ;
assign mem_result_i            = ex_mem_result_o         ;
assign mem_reg2_r_data_i       = ex_mem_reg2_r_data_o    ;
assign mem_wb_ctrl_i           = ex_mem_wb_ctrl_o        ;
assign mem_mem_Rd_i            = ex_mem_mem_Rd_o         ;
assign mem_en_addr_data_i      = if_en_addr_data_o       ;
assign mem_addr_data_i         = if_addr_data_o          ;
assign mem_data_data_i         = if_data_o               ;

assign mem_wb_flag_hold        = id_flag_hold_o          ;
assign mem_wb_wb_ctrl_i        = mem_wb_ctrl_o           ;
assign mem_wb_wb_data_i        = mem_mem_data_o          ;
assign mem_wb_data_i           = mem_data_o              ;
assign mem_wb_wb_Rd_i          = mem_mem_Rd_o            ;

assign wb_wb_ctrl_i            = mem_wb_wb_ctrl_o        ;
assign wb_wb_data_i            = mem_wb_wb_data_o        ;
assign wb_data_i               = mem_wb_data_o           ;
assign wb_wb_Rd_i              = mem_wb_wb_Rd_o          ;

MODULE_if if_inst(
    .sys_clk         (sys_clk              ),
    .sys_arstn       (sys_arstn            ),
    .flag_JorB_i     (if_flag_JorB_i       ),
    .inst_addr_JorB_i(if_inst_addr_JorB_i  ),
    .flag_hold_i     (if_flag_hold_i       ),
    .rx_i            (rx                   ),
```

```
    .inst_addr_o     (if_inst_addr_o       ),
    .inst_data_o     (if_inst_data_o       ),
    .en_addr_data_o  (if_en_addr_data_o    ),
    .addr_data_o     (if_addr_data_o       ),
    .data_o          (if_data_o            ),
    .start_o         (if_start_o           )
);

MODULE_if_id if_id_inst(
    .sys_clk    (sys_clk            ),
    .sys_arstn  (sys_arstn          ),
    .flag_flush (if_id_flag_flush   ),
    .flag_hold  (if_id_flag_hold    ),
    .inst_data_i(if_id_inst_data_i  ),
    .inst_addr_i(if_id_inst_addr_i  ),
    .inst_data_o(if_id_inst_data_o  ),
    .inst_addr_o(if_id_inst_addr_o  )
);

MODULE_id id_inst(
    .sys_clk      (sys_clk          ),
    .sys_arstn    (sys_arstn        ),
    .start_i      (id_start_i       ),
    .hold_i       (id_hold_i        ),
    .inst_addr_i  (id_inst_addr_i   ),
    .inst_data_i  (id_inst_data_i   ),
    .RegWrite_i   (id_RegWrite_i    ),
    .rd_i         (id_rd_i          ),
    .w_data_i     (id_w_data_i      ),
    .ex_Rd_i      (id_ex_Rd_i       ),
    .ex_MemRead_i (id_ex_MemRead_i  ),
    .flag_JorB_i  (id_flag_JorB_i   ),
    .flag_flush_o (id_flag_flush_o  ),
    .flag_hold_o  (id_flag_hold_o   ),
    .ctrl_o       (id_ctrl_o        ),
    .funct7_5_o   (id_funct7_5_o    ),
    .funct3_o     (id_funct3_o      ),
    .reg1_r_data_o(id_reg1_r_data_o ),
    .reg2_r_data_o(id_reg2_r_data_o ),
    .imm_o        (id_imm_o         ),
    .Rs1_o        (id_Rs1_o         ),
    .Rs2_o        (id_Rs2_o         ),
    .inst_addr_o  (id_inst_addr_o   ),
    .Rd_o         (id_Rd_o          )
```

```
);

MODULE_id_ex id_ex_inst(
    .sys_clk      (sys_clk             ),
    .sys_arstn    (sys_arstn           ),
    .flag_flush   (id_ex_flag_flush    ),
    .flag_hold    (id_ex_flag_hold     ),
    .funct7_5_i   (id_ex_funct7_5_i    ),
    .funct3_i     (id_ex_funct3_i      ),
    .ex_ctrl_i    (id_ex_ex_ctrl_i     ),
    .reg1_r_data_i(id_ex_reg1_r_data_i ),
    .reg2_r_data_i(id_ex_reg2_r_data_i ),
    .imm_i        (id_ex_imm_i         ),
    .ex_Rs1_i     (id_ex_ex_Rs1_i      ),
    .ex_Rs2_i     (id_ex_ex_Rs2_i      ),
    .inst_addr_i  (id_ex_inst_addr_i   ),
    .mem_ctrl_i   (id_ex_mem_ctrl_i    ),
    .wb_ctrl_i    (id_ex_wb_ctrl_i     ),
    .ex_Rd_i      (id_ex_ex_Rd_i       ),
    .funct7_5_o   (id_ex_funct7_5_o    ),
    .funct3_o     (id_ex_funct3_o      ),
    .ex_ctrl_o    (id_ex_ex_ctrl_o     ),
    .reg1_r_data_o(id_ex_reg1_r_data_o ),
    .reg2_r_data_o(id_ex_reg2_r_data_o ),
    .imm_o        (id_ex_imm_o         ),
    .ex_Rs1_o     (id_ex_ex_Rs1_o      ),
    .ex_Rs2_o     (id_ex_ex_Rs2_o      ),
    .inst_addr_o  (id_ex_inst_addr_o   ),
    .mem_ctrl_o   (id_ex_mem_ctrl_o    ),
    .wb_ctrl_o    (id_ex_wb_ctrl_o     ),
    .ex_Rd_o      (id_ex_ex_Rd_o       )
);

MODULE_ex ex_inst(
    .sys_clk      (sys_clk             ),
    .sys_arstn    (sys_arstn           ),
    .flag_JorB_i  (ex_flag_JorB_i      ),
    .funct7_5_i   (ex_funct7_5_i       ),
    .funct3_i     (ex_funct3_i         ),
    .ex_ctrl_i    (ex_ex_ctrl_i        ),
    .reg1_r_data_i(ex_reg1_r_data_i    ),
    .reg2_r_data_i(ex_reg2_r_data_i    ),
    .imm_i        (ex_imm_i            ),
    .mem_data_i   (ex_mem_data_i       ),
```

```
    .wb_data_i     (ex_wb_data_i        ),
    .mem_Rd_i      (ex_mem_Rd_i         ),
    .mem_wb_ctrl_i (ex_mem_wb_ctrl_i_ex ),
    .wb_Rd_i       (ex_wb_Rd_i          ),
    .wb_wb_ctrl_i  (ex_wb_wb_ctrl_i     ),
    .ex_Rs1_i      (ex_ex_Rs1_i         ),
    .ex_Rs2_i      (ex_ex_Rs2_i         ),
    .inst_addr_i   (ex_inst_addr_i      ),
    .mem_ctrl_i    (ex_mem_ctrl_i       ),
    .wb_ctrl_i     (ex_wb_ctrl_i        ),
    .ex_Rd_i       (ex_ex_Rd_i          ),
    .ex_Rd_o       (ex_ex_Rd_o          ),
    .mem_ctrl_o    (ex_mem_ctrl_o       ),
    .wb_ctrl_o     (ex_wb_ctrl_o        ),
    .reg2_r_data_o (ex_reg2_r_data_o    ),
    .inst_addr_o   (ex_inst_addr_o      ),
    .result_o      (ex_result_o         ),
    .flag_result_o (ex_flag_result_o    ),
    .hold_o        (ex_hold_o           )
);

MODULE_ex_mem ex_mem_inst(
    .sys_clk      (sys_clk                    ),
    .sys_arstn    (sys_arstn                  ),
    .flag_flush   (ex_mem_flag_flush          ),
    .flag_hold    (ex_mem_flag_hold           ),
    .mem_ctrl_i   (ex_mem_mem_ctrl_i          ),
    .flag_result_i(ex_mem_flag_result_i       ),
    .inst_addr_i  (ex_mem_inst_addr_i         ),
    .result_i     (ex_mem_result_i            ),
    .reg2_r_data_i(ex_mem_reg2_r_data_i       ),
    .wb_ctrl_i    (ex_mem_wb_ctrl_i_ex_mem    ),
    .mem_Rd_i     (ex_mem_mem_Rd_i            ),
    .mem_ctrl_o   (ex_mem_mem_ctrl_o          ),
    .flag_result_o(ex_mem_flag_result_o       ),
    .inst_addr_o  (ex_mem_inst_addr_o         ),
    .result_o     (ex_mem_result_o            ),
    .reg2_r_data_o(ex_mem_reg2_r_data_o       ),
    .wb_ctrl_o    (ex_mem_wb_ctrl_o           ),
    .mem_Rd_o     (ex_mem_mem_Rd_o            )
);

MODULE_mem mem_inst(
    .sys_clk        (sys_clk             ),
```

```
    .sys_arstn       (sys_arstn            ),
    .mem_ctrl_i      (mem_mem_ctrl_i       ),
    .flag_result_i   (mem_flag_result_i    ),
    .inst_addr_i     (mem_inst_addr_i      ),
    .result_i        (mem_result_i         ),
    .reg2_r_data_i   (mem_reg2_r_data_i    ),
    .wb_ctrl_i       (mem_wb_ctrl_i        ),
    .mem_Rd_i        (mem_mem_Rd_i         ),
    .en_addr_data_i  (mem_en_addr_data_i   ),
    .addr_data_i     (mem_addr_data_i      ),
    .data_data_i     (mem_data_data_i      ),
    .mem_Rd_o        (mem_mem_Rd_o         ),
    .wb_ctrl_o       (mem_wb_ctrl_o        ),
    .flag_JorB_o     (mem_flag_JorB_o      ),
    .inst_addr_JorB_o(mem_inst_addr_JorB_o ),
    .data_o          (mem_data_o           ),
    .mem_data_o      (mem_mem_data_o       ),
    .tx_o            (tx                   )
);

MODULE_mem_wb mem_wb_inst(
    .sys_clk         (sys_clk                ),
    .sys_arstn       (sys_arstn              ),
    .flag_hold       (mem_wb_flag_hold       ),
    .wb_ctrl_i       (mem_wb_wb_ctrl_i       ),
    .wb_data_i       (mem_wb_wb_data_i       ),
    .data_i          (mem_wb_data_i          ),
    .wb_Rd_i         (mem_wb_wb_Rd_i         ),
    .wb_ctrl_o       (mem_wb_wb_ctrl_o       ),
    .wb_data_o       (mem_wb_wb_data_o       ),
    .data_o          (mem_wb_data_o          ),
    .wb_Rd_o         (mem_wb_wb_Rd_o         )
);

MODULE_wb wb_inst(
    .wb_ctrl_i       (wb_wb_ctrl_i       ),
    .wb_data_i       (wb_wb_data_i       ),
    .data_i          (wb_data_i          ),
    .wb_Rd_i         (wb_wb_Rd_i         ),
    .wb_Rd_o         (wb_wb_Rd_o         ),
    .RegWrite_o      (wb_RegWrite_o      ),
    .w_data_o        (wb_w_data_o        )
);
```

```
endmodule
```

32. *auto_tst.v*

```
module auto_tst ();

  reg  [31:0] dividend;
  reg  [31:0] divisor;

  reg  [31:0] dividend_reg;
  reg  [31:0] divisor_reg;

  wire [31:0] quotient;
  wire [31:0] reminder;
  wire        mulfinish;
  wire        diverror;
  wire [ 3:0] q_2_0;
  wire [ 3:0] q_1_0;
  wire [ 3:0] q_table_com_0;

  reg clock, clock2;
  reg rst;
  reg q_print;
  integer D, d, ans, r, error, n_tst, ans_reg, e;
  initial begin
    clock    = 0;
    rst      = 0;
    clock2   = 0;
    dividend = 0;
    divisor  = 0;
    error    = 0;
    n_tst    = 0;
    q_print  = 0;
    #200 rst = 1;

  end

  always #10 clock = ~clock;

  always #300 clock2 = ~clock2;

  always @(posedge clock2) begin
    dividend = {$random} % 4294967295;  //({$random}%255)
+32'hffffff00;//{$random}%494967295;
    divisor  = {$random} %
4294967295;  //{$random}%255;//({$random}%63)+32'd960;//;
```

```
  end

  always @(negedge mulfinish) begin
    #1 dividend_reg = dividend;
    divisor_reg = divisor;
    D           = dividend_reg;
    d           = divisor_reg;
    ans_reg     = ans;
    ans         = D / d;
  end

  /*
always@(negedge clock) begin
if(q_print)begin
$display("%b           %b           %b",q_table_com_0,q_1_0,q_2_0);
end
//else $display("-------------------------");
end

always@(posedge q_print)begin
$display("-------------------------");
$display("%d           %d           %d",D,d,ans,quotient);
$display("-------------------------");
end
*/

  always @(posedge mulfinish) begin
    #1
      if (ans != ans_reg) begin
        n_tst = n_tst + 1;
        if (ans != quotient) begin
          error = error + 1;
          $display("%d           %d           %d           %d", D, d,
ans, quotient);
          //e=( ($itor(ans) -$itor(quotient)))/$itor( ans );
          //$display("%0d           %0d           %f",error,n_tst,$itor
(( ($itor(ans) -$itor(quotient)))/$itor( ans )));
        end
      end
  end

  srt_8_div srt_8_div (
      .clk        (clock),
```

```
      .rst_n      (rst),
      .start      (1'd1),
      .sign_define(1'd1),
      .dividend_i (dividend),
      .divisor_i  (divisor),
      .quotient_o (quotient),
      .reminder_o (reminder),
      .mulfinish  (mulfinish)

  );

endmodule
```

33. $TB.v$

```
`define SIM_RTL

module TB ();

  parameter UART_BPS = 115200;
  parameter CLK_FREQ = 50000000;
  parameter CONSTANT = 1000000000;
  localparam PERIOD = CONSTANT / CLK_FREQ;
  localparam BAUD_CNT_MAX = CLK_FREQ / UART_BPS;

`ifdef SIM_RTL

  reg  sys_clk;
  reg  sys_arstn;
  reg  rx;
  wire tx;

  TOP TOP_inst (
      .sys_clk  (sys_clk),
      .sys_arstn(sys_arstn),
      .rx       (rx),
      .tx       (tx)
  );

  // initial begin
  //   $vcdpluson;
  //   $vcdplusmemon;
  //   $vcdplusdeltacycleon;
  // end

  initial sys_clk = 1'b1;
```

```
always #(PERIOD / 2) sys_clk = ~sys_clk;

initial rx = 1'b1;

integer flag_0;
integer flag_1;
integer flag_2;
integer flag_3;
integer flag_4;
integer flag_5;
integer flag_6;

initial begin
  sim_0(flag_0);
  sim_1(flag_1);
  sim_2(flag_2);
  sim_3(flag_3);
  sim_4(flag_4);
  sim_5(flag_5);
  sim_6(flag_6);
  if (flag_0 & flag_1 & flag_2 & flag_3 & flag_4 & flag_5 & flag_6)
    $display($time, "[INFO] SUCCESS!");
  $finish;
end

task reset();
  begin
    sys_arstn = 1'b0;
    #(PERIOD * 5 / 4);
    sys_arstn = 1'b1;
  end
endtask

task rx_byte(input [7:0] data);
  integer i_bit;
  begin
    for (i_bit = 0; i_bit < 10; i_bit = i_bit + 1) begin
      case (i_bit)
        0: rx = 1'b0;
        1: rx = data[0];
        2: rx = data[1];
        3: rx = data[2];
        4: rx = data[3];
```

```
          5: rx = data[4];
          6: rx = data[5];
          7: rx = data[6];
          8: rx = data[7];
          9: rx = 1'b1;
        endcase
        #(BAUD_CNT_MAX * PERIOD);
      end
    end
  endtask

  task rx_word(input [31:0] data);
    integer i_byte;
    begin
      for (i_byte = 0; i_byte < 4; i_byte = i_byte + 1) begin
        case (i_byte)
          0: rx_byte(data[31:24]);
          1: rx_byte(data[23:16]);
          2: rx_byte(data[15:8]);
          3: rx_byte(data[7:0]);
        endcase
      end
    end
  endtask

  task test(input [2:0] i, input [31:0] data_0, input [31:0] data_1,
input [31:0] data_2,
            input [31:0] data_3, input [31:0] data_4, input [31:0]
data_5, input [31:0] data_6,
            input [31:0] data_7, output flag);
    reg            temp_0;
    reg            temp_1;
    reg     [ 7:0] temp_byte;
    reg     [31:0] temp_word;
    integer        j_bit;
    integer        j_byte;
    integer        j_word;
    reg     [31:0] result_0;
    reg     [31:0] result_1;
    reg     [31:0] result_2;
    reg     [31:0] result_3;
    reg     [31:0] result_4;
    reg     [31:0] result_5;
    reg     [31:0] result_6;
```

```
    reg     [31:0] result_7;
    integer        finish;
    integer        i_display;
    begin
      finish = 0;
      j_byte = -1;
      j_word = -1;
      for (; finish != 1;) begin
        @(negedge tx);
        #(PERIOD * 3) temp_0 = tx;
        for (j_bit = 0; j_bit < 8; j_bit = j_bit + 1) begin
          #(BAUD_CNT_MAX * PERIOD);
          temp_byte = {tx, temp_byte[7:1]};
        end
        #(BAUD_CNT_MAX * PERIOD) temp_1 = tx;
        if ((temp_0 == 1'b0) & (temp_1 == 1'b1)) begin
          j_byte    = j_byte + 1;
          temp_word = {temp_word[23:0], temp_byte};
          if (j_byte == 3) begin
            j_byte = -1;
            j_word = j_word + 1;
            case (j_word)
              0: result_0 = temp_word;
              1: result_1 = temp_word;
              2: result_2 = temp_word;
              3: result_3 = temp_word;
              4: result_4 = temp_word;
              5: result_5 = temp_word;
              6: result_6 = temp_word;
              7: result_7 = temp_word;
            endcase
            if (j_byte == 3) j_word = -1;
            if (j_word == 7) finish = 1;
          end
        end
      end
      flag = 1;
      for (i_display = 0; i_display < 8; i_display = i_display + 1)
begin
        case (i_display)
          0: begin
            if (result_0 != data_0) begin
              flag = 0;
              $display($time, "\tresult_0 = %d", result_0);
```

```
    $finish;
  end
end
1: begin
  if (result_1 != data_1) begin
    flag = 0;
    $display($time, "\tresult_1 = %d", result_1);
    $finish;
  end
end
2: begin
  if (result_2 != data_2) begin
    flag = 0;
    $display($time, "\tresult_2 = %d", result_2);
    $finish;
  end
end
3: begin
  if (result_3 != data_3) begin
    flag = 0;
    $display($time, "\tresult_3 = %d", result_3);
    $finish;
  end
end
4: begin
  if (result_4 != data_4) begin
    flag = 0;
    $display($time, "\tresult_4 = %d", result_4);
    $finish;
  end
end
5: begin
  if (result_5 != data_5) begin
    flag = 0;
    $display($time, "\tresult_5 = %d", result_5);
    $finish;
  end
end
6: begin
  if (result_6 != data_6) begin
    flag = 0;
    $display($time, "\tresult_6 = %d", result_6);
    $finish;
  end
```

```
        end
        7: begin
          if (result_7 != data_7) begin
            flag = 0;
            $display($time, "\tresult_7 = %d", result_7);
            $finish;
          end
        end
      endcase
    end
    if (flag == 1) $display($time, "[INFO] SIM_%d SUCCESS!", i);
  end
endtask

task sim_0(output flag);
  integer i_0_inst;
  integer i_0_data;
  fork
    begin
      $display($time, "[INFO] SIM_0 START!");
      reset();
      rx_word(32'b00000000000100001000000010010011);
      rx_word(32'b00000000000100001000000010010011);
      rx_word(32'b00000000000100001000000010010011);
      rx_word(32'b00000000000100000010000000100011);
      for (i_0_inst = 0; i_0_inst < 12; i_0_inst = i_0_inst + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
      for (i_0_data = 0; i_0_data < 8; i_0_data = i_0_data + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(0, 3, 0, 0, 0, 0, 0, 0, 0, flag);
  join
endtask

task sim_1(output flag);
  integer i_1_inst;
  integer i_1_data;
  fork
    begin
      $display($time, "[INFO] SIM_1 START!");
      reset();
      rx_word(32'b00000000000100001000000010010011);
```

```
      rx_word(32'b00000000000100001000000010010011);
      rx_word(32'b00000000100000000000000101101111);
      rx_word(32'b00000000000100001000000010010011);
      rx_word(32'b00000000000100000010000000100011);
      rx_word(32'b00000000000100000010001000100011);
      for (i_1_inst = 0; i_1_inst < 10; i_1_inst = i_1_inst + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
      for (i_1_data = 0; i_1_data < 8; i_1_data = i_1_data + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(1, 2, 12, 0, 0, 0, 0, 0, 0, flag);
  join
endtask

task sim_2(output flag);
  integer i_2_inst;
  integer i_2_data;
  fork
    begin
      $display($time, "[INFO] SIM_2 START!");
      reset();
      rx_word(32'b00000000000100001000000010010011);
      rx_word(32'b00000000000100000010000000100011);
      rx_word(32'b00000000000000000010000010000011);
      rx_word(32'b00000000000100001000000010010011);
      rx_word(32'b00000000000100000010000000100011);
      for (i_2_inst = 0; i_2_inst < 11; i_2_inst = i_2_inst + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
      for (i_2_data = 0; i_2_data < 8; i_2_data = i_2_data + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(2, 2, 0, 0, 0, 0, 0, 0, 0, flag);
  join
endtask

task sim_3(output flag);
  integer i_3_inst;
  integer i_3_data;
  fork
    begin
```

```
      $display($time, "[INFO] SIM_3 START!");
      reset();
      rx_word(32'b00000000000000000010000010000011);
      rx_word(32'b00000000010000000010000100000011);
      rx_word(32'b00000010000100010111000110110011);
      rx_word(32'b00000000001100000010010000100011);
      rx_word(32'b00000010001000001111001000110011);
      rx_word(32'b00000000010000000010011000100011);
      for (i_3_inst = 0; i_3_inst < 10; i_3_inst = i_3_inst + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
      rx_word(32'd69);
      rx_word(32'd10);
      for (i_3_data = 0; i_3_data < 6; i_3_data = i_3_data + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(3, 69, 10, 10, 9, 0, 0, 0, 0, flag);
  join
endtask

task sim_4(output flag);
  integer i_4_inst;
  integer i_4_data;
  fork
    begin
      $display($time, "[INFO] SIM_4 START!");
      reset();
      rx_word(32'b00000000000000000010000010000011);
      rx_word(32'b00000000010000000010000100000011);
      rx_word(32'b00000010000100010111000110110011);
      rx_word(32'b00000000001100000010010000100011);
      rx_word(32'b00000010001000001111001000110011);
      rx_word(32'b00000000010000000010011000100011);
      for (i_4_inst = 0; i_4_inst < 10; i_4_inst = i_4_inst + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
      rx_word(32'd69);
      rx_word(32'd10);
      for (i_4_data = 0; i_4_data < 6; i_4_data = i_4_data + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(4, 69, 10, 10, 9, 0, 0, 0, 0, flag);
```

```
  join
endtask

task sim_5(output flag);
  integer i_5_inst;
  integer i_5_data;
  fork
    begin
      $display($time, "[INFO] SIM_5 START!");
      reset();
      rx_word(32'b00000000000000000010000010000011);
      rx_word(32'b00000000010000000010000100000011);
      rx_word(32'b00000010000100010101000110110011);
      rx_word(32'b00000000001100000010010000100011);
      rx_word(32'b00000010001000001101001000110011);
      rx_word(32'b00000000010000000010011000100011);
      for (i_5_inst = 0; i_5_inst < 10; i_5_inst = i_5_inst + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
      rx_word(32'd69);
      rx_word(32'd10);
      for (i_5_data = 0; i_5_data < 6; i_5_data = i_5_data + 1) begin
        rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(5, 69, 10, 0, 6, 0, 0, 0, 0, flag);
  join
endtask

task sim_6(output flag);
  integer i_6_inst;
  integer i_6_data;
  fork
    begin
      $display($time, "[INFO] SIM_6 START!");
      reset();
      rx_word(32'b00000000000000000010000010000011);
      rx_word(32'b00000000010000000010000100000011);
      rx_word(32'b00000010000100010100000110110011);
      rx_word(32'b00000000001100000010010000100011);
      rx_word(32'b00000010001000001100001000110011);
      rx_word(32'b00000000010000000010011000100011);
      for (i_6_inst = 0; i_6_inst < 10; i_6_inst = i_6_inst + 1) begin
        rx_word(32'b00000000000000000000000000000000);
```

```
        end
        rx_word(32'd69);
        rx_word(32'd10);
        for (i_6_data = 0; i_6_data < 6; i_6_data = i_6_data + 1) begin
          rx_word(32'b00000000000000000000000000000000);
        end
      end
      test(6, 69, 10, 0, 6, 0, 0, 0, 0, flag);
    join
  endtask

`endif

`ifdef SIM_GATE

  reg  chip_sys_clk;
  reg  chip_sys_arstn;
  reg  chip_rx;
  wire chip_tx;

  CHIP CHIP_inst (
      .chip_sys_clk  (chip_sys_clk),
      .chip_sys_arstn(chip_sys_arstn),
      .chip_rx       (chip_rx),
      .chip_tx       (chip_tx)
  );

  // initial begin
  //   $vcdpluson;
  //   $vcdplusmemon;
  //   $vcdplusdeltacycleon;
  // end

  initial chip_sys_clk = 1'b1;

  always #(PERIOD / 2) chip_sys_clk = ~chip_sys_clk;

  initial chip_rx = 1'b1;

  integer flag_0;
  integer flag_1;
  integer flag_2;
  integer flag_3;
  integer flag_4;
```

```
integer flag_5;
integer flag_6;

initial begin
  sim_0(flag_0);
  sim_1(flag_1);
  sim_2(flag_2);
  sim_3(flag_3);
  sim_4(flag_4);
  sim_5(flag_5);
  sim_6(flag_6);
  if (flag_0 & flag_1 & flag_2 & flag_3 & flag_4 & flag_5 & flag_6)
    $display($time, "[INFO] SUCCESS!");
  $finish;
end

task reset();
  begin
    chip_sys_arstn = 1'b0;
    #(PERIOD * 5 / 4);
    chip_sys_arstn = 1'b1;
  end
endtask

task chip_rx_byte(input [7:0] data);
  integer i_bit;
  begin
    for (i_bit = 0; i_bit < 10; i_bit = i_bit + 1) begin
      case (i_bit)
        0: chip_rx = 1'b0;
        1: chip_rx = data[0];
        2: chip_rx = data[1];
        3: chip_rx = data[2];
        4: chip_rx = data[3];
        5: chip_rx = data[4];
        6: chip_rx = data[5];
        7: chip_rx = data[6];
        8: chip_rx = data[7];
        9: chip_rx = 1'b1;
      endcase
      #(BAUD_CNT_MAX * PERIOD);
    end
  end
endtask
```

```
  task chip_rx_word(input [31:0] data);
    integer i_byte;
    begin
      for (i_byte = 0; i_byte < 4; i_byte = i_byte + 1) begin
        case (i_byte)
          0: chip_rx_byte(data[31:24]);
          1: chip_rx_byte(data[23:16]);
          2: chip_rx_byte(data[15:8]);
          3: chip_rx_byte(data[7:0]);
        endcase
      end
    end
  endtask

  task test(input [2:0] i, input [31:0] data_0, input [31:0] data_1,
input [31:0] data_2,
            input [31:0] data_3, input [31:0] data_4, input [31:0]
data_5, input [31:0] data_6,
            input [31:0] data_7, output flag);
    reg            temp_0;
    reg            temp_1;
    reg     [ 7:0] temp_byte;
    reg     [31:0] temp_word;
    integer        j_bit;
    integer        j_byte;
    integer        j_word;
    reg     [31:0] result_0;
    reg     [31:0] result_1;
    reg     [31:0] result_2;
    reg     [31:0] result_3;
    reg     [31:0] result_4;
    reg     [31:0] result_5;
    reg     [31:0] result_6;
    reg     [31:0] result_7;
    integer        finish;
    integer        i_display;
    begin
      finish = 0;
      j_byte = -1;
      j_word = -1;
      for (; finish != 1;) begin
        @(negedge chip_tx);
        #(PERIOD * 3) temp_0 = chip_tx;
```

```
        for (j_bit = 0; j_bit < 8; j_bit = j_bit + 1) begin
          #(BAUD_CNT_MAX * PERIOD);
          temp_byte = {chip_tx, temp_byte[7:1]};
        end
        #(BAUD_CNT_MAX * PERIOD) temp_1 = chip_tx;
        if ((temp_0 == 1'b0) & (temp_1 == 1'b1)) begin
          j_byte    = j_byte + 1;
          temp_word = {temp_word[23:0], temp_byte};
          if (j_byte == 3) begin
            j_byte = -1;
            j_word = j_word + 1;
            case (j_word)
              0: result_0 = temp_word;
              1: result_1 = temp_word;
              2: result_2 = temp_word;
              3: result_3 = temp_word;
              4: result_4 = temp_word;
              5: result_5 = temp_word;
              6: result_6 = temp_word;
              7: result_7 = temp_word;
            endcase
            if (j_byte == 3) j_word = -1;
            if (j_word == 7) finish = 1;
          end
        end
      end
      flag = 1;
      for (i_display = 0; i_display < 8; i_display = i_display + 1)
begin
        case (i_display)
          0: begin
            if (result_0 != data_0) begin
              flag = 0;
              $display($time, "\tresult_0 = %d", result_0);
              $finish;
            end
          end
          1: begin
            if (result_1 != data_1) begin
              flag = 0;
              $display($time, "\tresult_1 = %d", result_1);
              $finish;
            end
          end
```

```
    2: begin
      if (result_2 != data_2) begin
        flag = 0;
        $display($time, "\tresult_2 = %d", result_2);
        $finish;
      end
    end
    3: begin
      if (result_3 != data_3) begin
        flag = 0;
        $display($time, "\tresult_3 = %d", result_3);
        $finish;
      end
    end
    4: begin
      if (result_4 != data_4) begin
        flag = 0;
        $display($time, "\tresult_4 = %d", result_4);
        $finish;
      end
    end
    5: begin
      if (result_5 != data_5) begin
        flag = 0;
        $display($time, "\tresult_5 = %d", result_5);
        $finish;
      end
    end
    6: begin
      if (result_6 != data_6) begin
        flag = 0;
        $display($time, "\tresult_6 = %d", result_6);
        $finish;
      end
    end
    7: begin
      if (result_7 != data_7) begin
        flag = 0;
        $display($time, "\tresult_7 = %d", result_7);
        $finish;
      end
    end
  endcase
end
```

```
    if (flag == 1) $display($time, "[INFO] SIM_%d SUCCESS!", i);
  end
endtask

task sim_0(output flag);
  integer i_0_inst;
  integer i_0_data;
  fork
    begin
      $display($time, "[INFO] SIM_0 START!");
      reset();
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000000100000100000000100011);
      for (i_0_inst = 0; i_0_inst < 12; i_0_inst = i_0_inst + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
      for (i_0_data = 0; i_0_data < 8; i_0_data = i_0_data + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(0, 3, 0, 0, 0, 0, 0, 0, 0, flag);
  join
endtask

task sim_1(output flag);
  integer i_1_inst;
  integer i_1_data;
  fork
    begin
      $display($time, "[INFO] SIM_1 START!");
      reset();
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000100000000000000101101111);
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000000100000100000000100011);
      chip_rx_word(32'b00000000001000000100001000100011);
      for (i_1_inst = 0; i_1_inst < 10; i_1_inst = i_1_inst + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
      for (i_1_data = 0; i_1_data < 8; i_1_data = i_1_data + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
```

```
      end
    end
    test(1, 2, 12, 0, 0, 0, 0, 0, 0, flag);
  join
endtask

task sim_2(output flag);
  integer i_2_inst;
  integer i_2_data;
  fork
    begin
      $display($time, "[INFO] SIM_2 START!");
      reset();
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000000100000010000000100011);
      chip_rx_word(32'b00000000000000000010000010000011);
      chip_rx_word(32'b00000000000100001000000010010011);
      chip_rx_word(32'b00000000000100000010000000100011);
      for (i_2_inst = 0; i_2_inst < 11; i_2_inst = i_2_inst + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
      for (i_2_data = 0; i_2_data < 8; i_2_data = i_2_data + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(2, 2, 0, 0, 0, 0, 0, 0, 0, flag);
  join
endtask

task sim_3(output flag);
  integer i_3_inst;
  integer i_3_data;
  fork
    begin
      $display($time, "[INFO] SIM_3 START!");
      reset();
      chip_rx_word(32'b00000000000000000010000010000011);
      chip_rx_word(32'b00000000010000000010000100000011);
      chip_rx_word(32'b00000010000100010111000110110011);
      chip_rx_word(32'b00000000001100000010010000100011);
      chip_rx_word(32'b00000010001000001111001000110011);
      chip_rx_word(32'b00000000010000000010011000100011);
      for (i_3_inst = 0; i_3_inst < 10; i_3_inst = i_3_inst + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
```

```
      end
      chip_rx_word(32'd69);
      chip_rx_word(32'd10);
      for (i_3_data = 0; i_3_data < 6; i_3_data = i_3_data + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(3, 69, 10, 10, 9, 0, 0, 0, 0, flag);
  join
endtask

task sim_4(output flag);
  integer i_4_inst;
  integer i_4_data;
  fork
    begin
      $display($time, "[INFO] SIM_4 START!");
      reset();
      chip_rx_word(32'b00000000000000000010000010000011);
      chip_rx_word(32'b00000000010000000010000100000011);
      chip_rx_word(32'b00000010000100010110000110110011);
      chip_rx_word(32'b00000000001100000010010000100011);
      chip_rx_word(32'b00000010001000001110001000110011);
      chip_rx_word(32'b00000000010000000010011000100011);
      for (i_4_inst = 0; i_4_inst < 10; i_4_inst = i_4_inst + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
      chip_rx_word(32'd69);
      chip_rx_word(32'd10);
      for (i_4_data = 0; i_4_data < 6; i_4_data = i_4_data + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(4, 69, 10, 10, 9, 0, 0, 0, 0, flag);
  join
endtask

task sim_5(output flag);
  integer i_5_inst;
  integer i_5_data;
  fork
    begin
      $display($time, "[INFO] SIM_5 START!");
      reset();
```

```
      chip_rx_word(32'b00000000000000000010000010000011);
      chip_rx_word(32'b00000000010000000010000100000011);
      chip_rx_word(32'b00000010000100010101000110110011);
      chip_rx_word(32'b00000000001100000010010000100011);
      chip_rx_word(32'b00000010001000001101001000110011);
      chip_rx_word(32'b00000000010000000010011000100011);
      for (i_5_inst = 0; i_5_inst < 10; i_5_inst = i_5_inst + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
      chip_rx_word(32'd69);
      chip_rx_word(32'd10);
      for (i_5_data = 0; i_5_data < 6; i_5_data = i_5_data + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(5, 69, 10, 0, 6, 0, 0, 0, 0, flag);
  join
endtask

task sim_6(output flag);
  integer i_6_inst;
  integer i_6_data;
  fork
    begin
      $display($time, "[INFO] SIM_6 START!");
      reset();
      chip_rx_word(32'b00000000000000000010000010000011);
      chip_rx_word(32'b00000000010000000010000100000011);
      chip_rx_word(32'b00000010000100010100000110110011);
      chip_rx_word(32'b00000000001100000010010000100011);
      chip_rx_word(32'b00000010001000001100001000110011);
      chip_rx_word(32'b00000000010000000010011000100011);
      for (i_6_inst = 0; i_6_inst < 10; i_6_inst = i_6_inst + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
      chip_rx_word(32'd69);
      chip_rx_word(32'd10);
      for (i_6_data = 0; i_6_data < 6; i_6_data = i_6_data + 1) begin
        chip_rx_word(32'b00000000000000000000000000000000);
      end
    end
    test(6, 69, 10, 0, 6, 0, 0, 0, 0, flag);
  join
endtask
```

```
`endif

endmodule
```