**中国科学技术大学计算机学院**

**《数字电路实验》报告**



实验题目：\_\_\_\_综合实验\_\_\_\_

学生姓名：\_\_\_\_\_\_徐奥\_\_\_\_\_\_

学生学号：\_\_\_PB20061343\_\_\_

完成日期：2021年12月19日

计算机实验教学中心制

2020年09月

【实验题目】

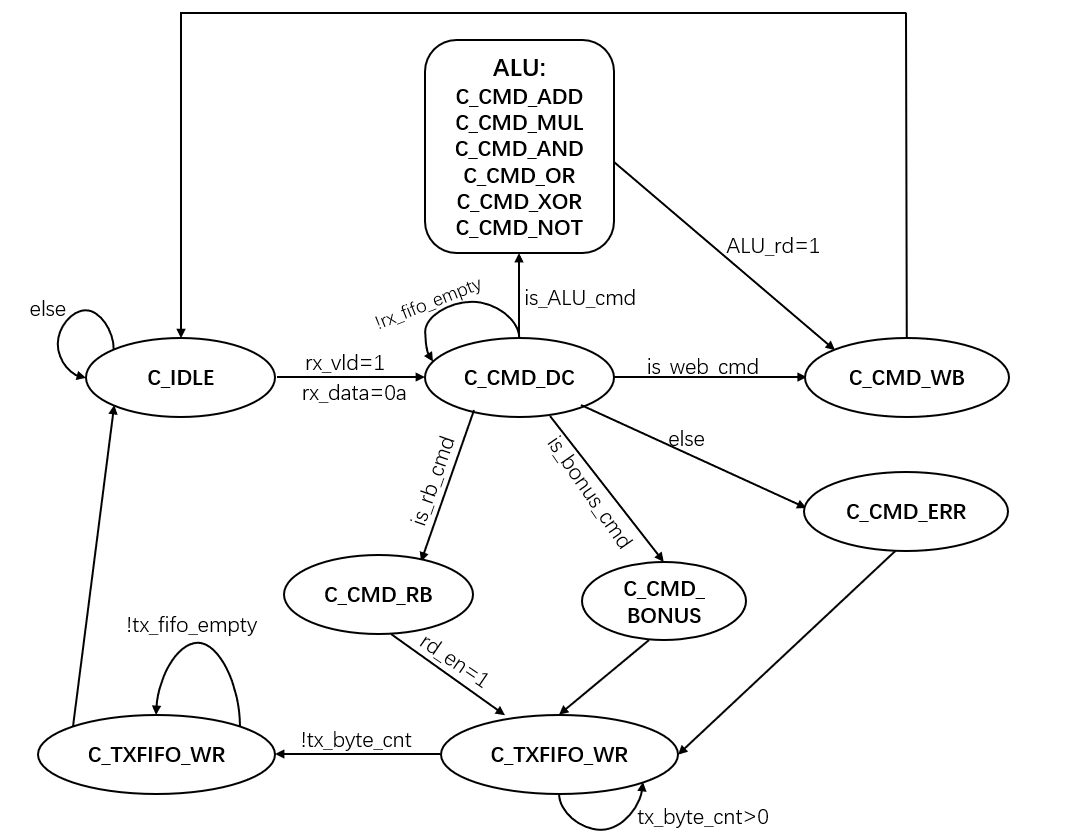
1. 在FPGAOL平台上，利用串口终端等外设，实现简单的Shell功能，例如：在串口协议基础上，实现一个读写命令解析功能，功能电路接收以ASCII码格式发来的命令，并根据命令类型做出合适的响应。
2. 在Logisim中或者在FPGA开发板上实现逻辑电路，通过LED点阵实现汉字的循环显示。要求至少循环显示十个汉字，汉字内容及机内码的形式保存在ROM中，控制电路顺序读取数据，完成机内码到区位码的转换，通过查询字库，获取16\*16的像素数据，最终显示在LED点阵上。

【实验练习】

**一、在FPGAOL平台上实现Shell功能**

**1. 建立状态转换图**

本程序会在以下状态中循环。初始状态为C\_IDLE，当读取到来自Shell的指令时，进入C\_CMD\_DC状态进行指令解码，根据不同指令进入对应的状态，执行完毕后回到初始状态。状态转换图如下：



**2. 命令格式说明**

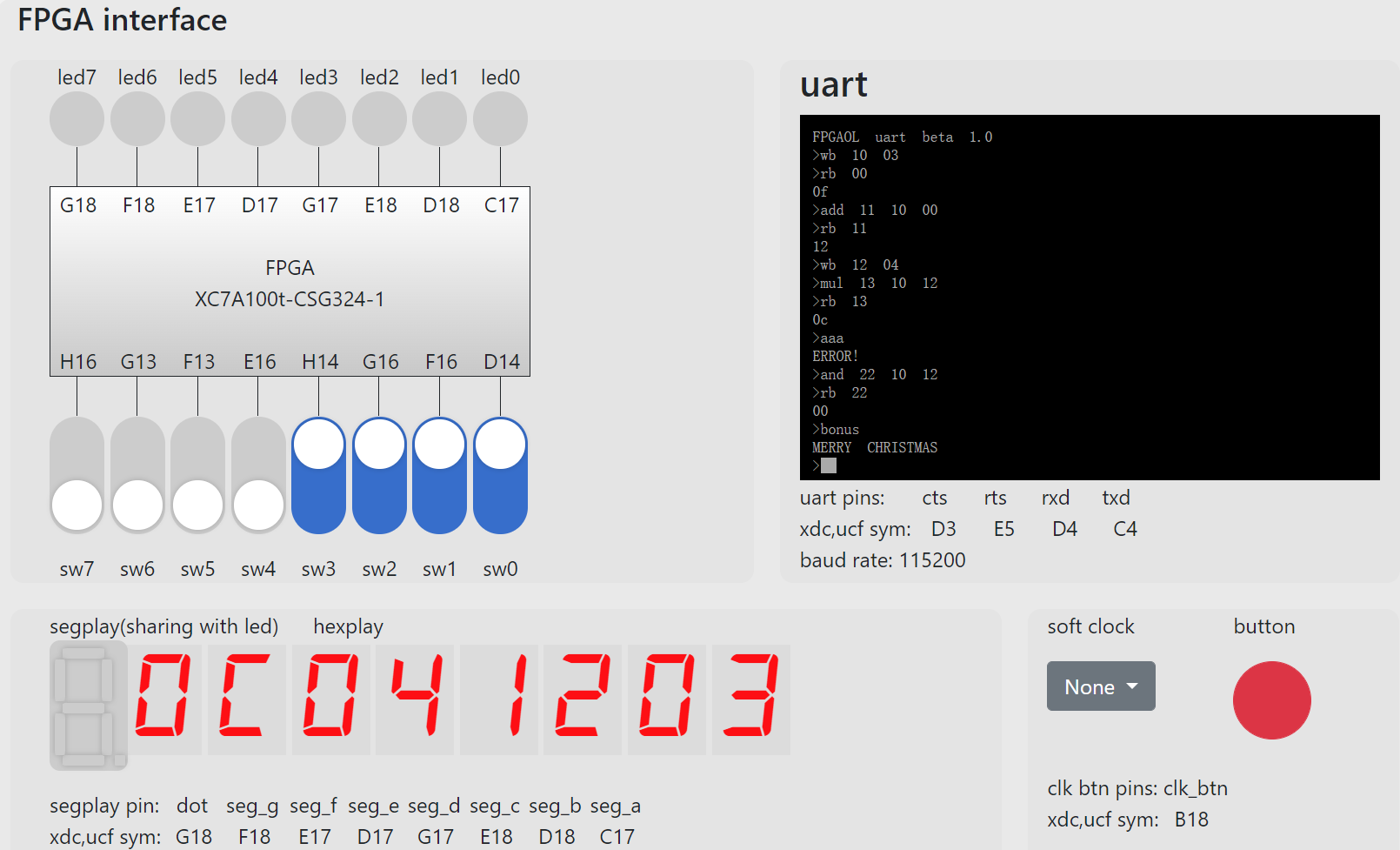
|  |  |  |  |
| --- | --- | --- | --- |
| 命令功能 | 格式 | 举例 | 说明 |
| 写字节 | wb [addr] [data] | wb 11 1a | 向11地址写入字节1a |
| 读字节 | rb [addr] | rb 10 | 从10地址读取一个字节，并以ASCII码格式显示在串口终端 |
| 加运算 | add [addr] [addr] [addr] | add 12 11 10 | 将地址11和10的值相加，结果存储在地址12中 |
| 乘运算 | mul [addr] [addr] [addr] | mul 12 11 10 | 将地址11和10的值相乘，结果存储在地址12中 |
| 与运算 | and [addr] [addr] [addr] | and 12 11 10 | 将地址11和10的值相与，结果存储在地址12中 |
| 或运算 | or [addr] [addr] [addr] | or 12 11 10 | 将地址11和10的值相或，结果存储在地址12中 |
| 非运算 | not [addr] [addr] | not 12 11 | 将地址11的值取非，结果存储到地址12中 |
| 异或运算 | xor [addr] [addr] [addr] | xor 12 11 10 | 将地址11和10的值相异或，结果存储在地址12中 |
| 彩蛋 | bonus | bonus | 在串口终端上显示  ”MERRY CHRISTMAS” |
| 其他 |  |  | 无效命令，串口终端打印”ERROR!”字样 |

说明：本程序对命令格式有严格要求，命令、 地址、数据之间有且仅有一个空格，且只有在输入回车后，当前一行命令才会被读入。

1. 地址空间分配

|  |  |  |  |
| --- | --- | --- | --- |
| 读地址空间 | | 写地址空间 | |
| 00 | 8个拨动开关所表示的字节数据 | 00 | 8个LED所对应的字节数据 |
| 10 | 数码管1~0位所表示的字节数据 | 10 | 数码管1~0位所表示的字节数据 |
| 11 | 数码管3~2位所表示的字节数据 | 11 | 数码管3~2位所表示的字节数据 |
| 12 | 数码管5~4位所表示的字节数据 | 12 | 数码管5~4位所表示的字节数据 |
| 13 | 数码管7~6位所表示的字节数据 | 13 | 数码管7~6位所表示的字节数据 |
| 20~2f | 内部存储空间 | 20~2f | 内部存储空间 |

说明：20至2f共计16个地址空间，每个对应的内容空间为8bits

1. 实测演示截图
2. 扩展部分
3. 扩展了指令个数，引入了加、乘、与、或、非、异或操作
4. 扩展了内存个数，由原来的5个扩展为21个
5. 命令实现过程

**（1）读入并解析来自Shell的命令：**

通过模块rx将来自串口的数据进行转化，约定数据收发频率为115200，运用分频计数器对串口接收信号进行采样，保存到rx\_data,每读完8bits数据，置读取完成的标志位rx\_vld为1.

rx\_vld作为rx\_fifo写入的使能标志，当一次8bits数据读入后，调用fifo模块将该数据存储到rx\_fifo中。

当来自串口的信号读取到换行符时，说明来自Shell的一行命令输入完毕，此时状态机进入命令解码状态。从rx\_fifo中读取数据并且存入rx\_byte\_buff，更新命令标志变量is\_add\_cmd，is\_wb\_cmd等.

**（2）wb命令：**

若已将rx\_fifo中的数据读取完毕，且is\_wb\_cmd标志变量为1，则进入写命令状态。根据rx\_byte\_buff更新wr\_addr和wr\_data，前者为要写入数据的目标地址，后者为要写入的数据内容。根据wr\_addr，更新对应的地址数据。

**（3）rb指令**

若已将rx\_fifo中的数据读取完毕，且is\_rb\_cmd为1，则进入读命令状态。根据rx\_byte\_buff更新rd\_addr，即要读取数据的地址。然后根据rd\_addr到对应地址读取数据，存储到rd\_data.

之后进入C\_TXFIFO\_WR和C\_TXFIFO\_WAIT状态，将要显示到串口终端的数据写入tx\_fifo，并逐个tx\_fifo转化为tx\_data，再将tx\_data通过调用tx模块转化为输出到串口的数据。每当tx模块转化完其接收到的数据，则置tx\_fifo的读使能有效，即再从tx\_fifo中读取数据送入tx模块。

tx模块，实现将读入的8bits数据按照数据收发频率115200转化为输出数据，输出到串口终端。

**（4）add命令**

若已将rx\_fifo中的数据读取完毕，且is\_add\_cmd标志位为1，那么进入加命令状态。根据rx\_byte\_buff更新ALU\_addr\_0, ALU\_addr\_1, ALU\_addr\_2，第一个为结果存储的目标地址，后两个为进行加法运算的操作数地址。

根据ALU\_addr\_1和ALU\_addr\_2获取对应位置的操作数，存储到ALU\_operand\_1和ALU\_operand\_2，调用8bits加法器，求得加法结果，存入ALU\_result。

用ALU\_addr\_0和ALU\_result分别更新wr\_addr和wr\_data，进入写命令状态，根据wr\_addr，用wr\_data更新目标地址的值。

**（5）mul命令**

若已将rx\_fifo中的数据读取完毕，且is\_mul\_cmd标志位为1，那么进入乘命令状态。根据rx\_byte\_buff更新ALU\_addr\_0, ALU\_addr\_1, ALU\_addr\_2，第一个为结果存储的目标地址，后两个为进行乘法运算的操作数地址。

根据ALU\_addr\_1和ALU\_addr\_2获取对应位置的操作数，存储到ALU\_operand\_1和ALU\_operand\_2，调用8bits乘法器，求得乘法结果，存入ALU\_result。

用ALU\_addr\_0和ALU\_result分别更新wr\_addr和wr\_data，进入写命令状态，根据wr\_addr，用wr\_data更新目标地址的值。

**（6）逻辑运算命令**

本程序实现的逻辑运算包括与、或、非、异或，所有逻辑运算执行过程类似，只是在部分细节上存在微小差异，故归到一类来说明。

若已将rx\_fifo中的数据读取完毕，且根据更新后的逻辑运算命令标志位，判定为逻辑运算命令，则进入对应的逻辑运算命令执行状态。

根据rx\_byte\_buff更新ALU\_addr\_0, ALU\_addr\_1, ALU\_addr\_2，第一个为结果存储的目标地址，后两个为进行逻辑运算的操作数地址。

根据ALU\_addr\_1和ALU\_addr\_2获取对应位置的操作数，存储到ALU\_operand\_1和ALU\_operand\_2，进行逻辑运算，将结果存储到ALU\_result，用ALU\_addr\_0和ALU\_result分别更新wr\_addr和wr\_data，进入写命令状态，根据wr\_addr，用wr\_data更新目标地址的值。

注：非运算只有一个操作数，故非运算在上述执行过程中只会获得一个操作数地址，相应的，只会获得一个有效操作数。

**（7）彩蛋**

若已将rx\_fifo中的数据读取完毕，且彩蛋命令标志位为1，则进入彩蛋命令处理状态。

置tx\_byte\_buff为 ”MERRY CHRISTMAS\n”, 进入C\_TXFIFO\_WR和C\_TXFIFO\_WAIT状态，将要tx\_byte\_buff写入tx\_fifo，并逐个tx\_fifo转化为tx\_data，再将tx\_data通过调用tx模块转化为输出到串口的数据。

**（8）命令不合法，输出REEOR信息**

若已将rx\_fifo中的数据读取完毕，且更新后的各命令标志位均为零，即读取到的命令非法，则进入错误命令处理状态。

置tx\_byte\_buff为 ”ERROR!\n”, 进入C\_TXFIFO\_WR和C\_TXFIFO\_WAIT状态，将要tx\_byte\_buff写入tx\_fifo，并逐个tx\_fifo转化为tx\_data，再将tx\_data通过调用tx模块转化为输出到串口的数据。

**（9）将部分地址对应的数据显示到LED或数码管上**

将对应的数据轮换存储到hexplay\_data，利用刷新，显示到FPGA上。

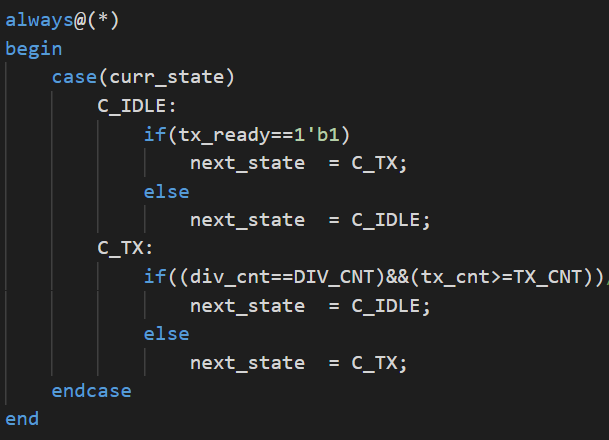
1. 关键代码演示说明

**（1）tx模块**

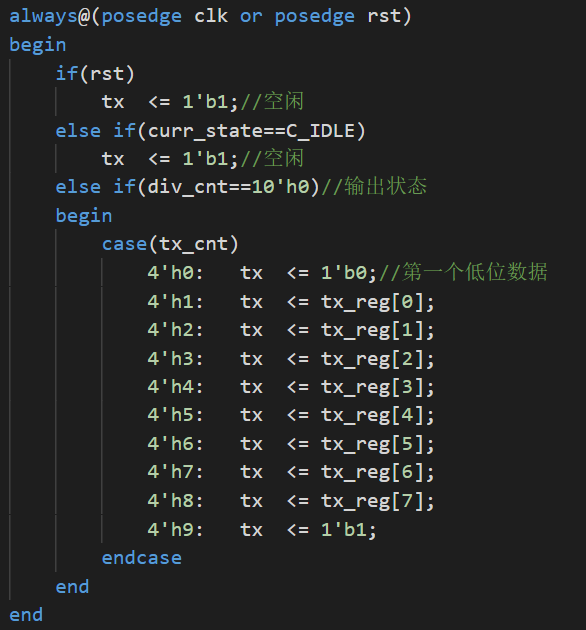
此模块要实现的功能为将8bits input数据转化为串口数据，即一个信号会持续868个时钟周期，再加上开头一位起始低位，结尾一位停止高位。

建立状态机：共包含两个状态，C\_IDLE和C\_TX，前者为空闲状态，后者为输出状态，即将tx\_data的每一位转化为868时钟周期的输出信号tx。当tx\_fifo非空时，状态机进入输出状态。当数据输出数目达到10\*868后，说明当前8bits数目输出完毕，进入空闲状态。

div\_cnt记录当前bit的数据已经输出了多少位，tx\_cnt记录当前输出是第几位数据。更新次态的代码为：



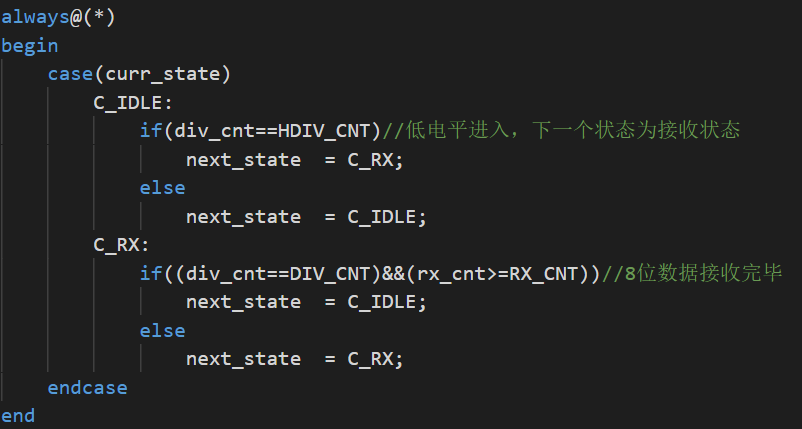
转化输出信号tx的代码为：



**（2）rx模块**

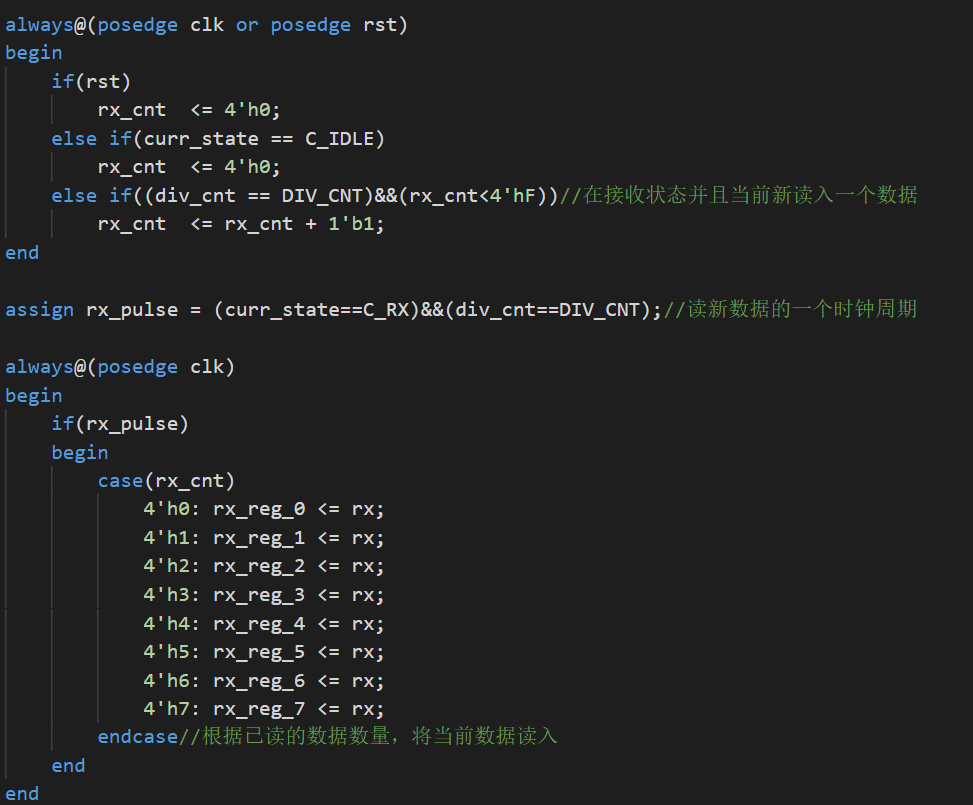
本模块实现了串口数据的输入，并将输入的数据转化为8bits的rx\_data。与tx模块实现过程类似，只不过数据传输方向相反。

状态机有两个状态，空闲状态和输入状态。由于数据采样是在868bits信号的中间，所以当第一个868时钟周期的低位开始信号输入一半时，状态由空闲装填转化为输入状态。更新次态的代码如下：



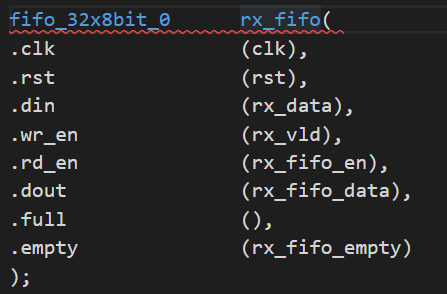
数据采样代码如下：





**（3）rx\_fifo**

fifo这个ip核实现了先入先出的数据存储和读取，rx\_fifo实现了将原本868时钟周期才读入1bit的输入数据暂存，然后在获得完整的来自Shell的一行指令后，再依次输出，实现了对时钟不同步的输入信号的同步化处理。

 它的调用代码为：

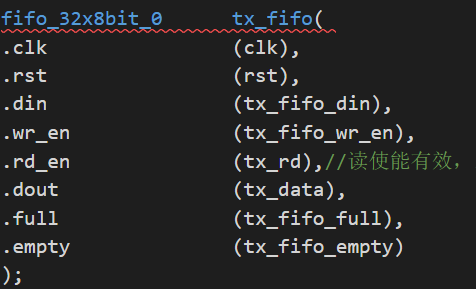
向rx\_fifo写入数据时：写使能为rx\_vld，写使能为1当且仅当rx模块刚读取完8bit数据。

从tx\_fifo读取数据：读使能为rx\_fifo\_en，在命令解码状态会置这个标志位为1

**（4）tx\_fifo**

tx\_fifo实现了将要输出的数据，根据约定好的串口数据收发频率进行转化输出，将每一位要输出的数据转化为868时钟周期的信号。

调用代码为：

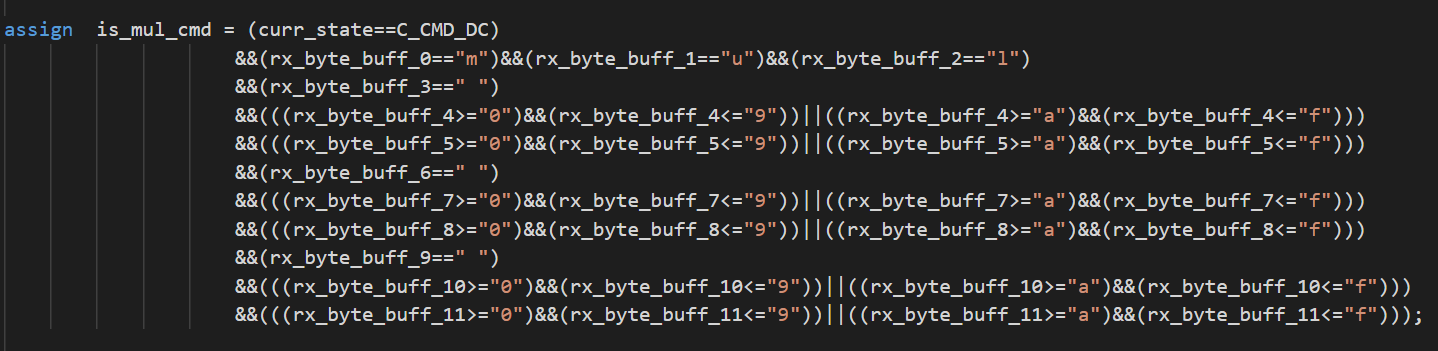


写入数据：将要输入的数据赋值给tx\_fifo\_din，写使能为tx\_fifo\_wr\_en，在C\_TXFIFO\_WR状态会置此标志位为1.

读取数据：读使能为tx\_rd，这个标志位为1当且仅当tx\_fifo非空且tx模块输出完上一个8bits数据。

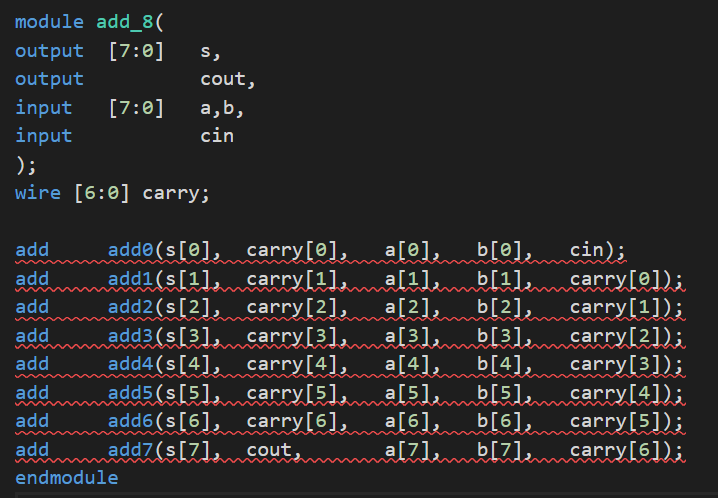
**（5）命令解码状态C\_CMD\_DC**

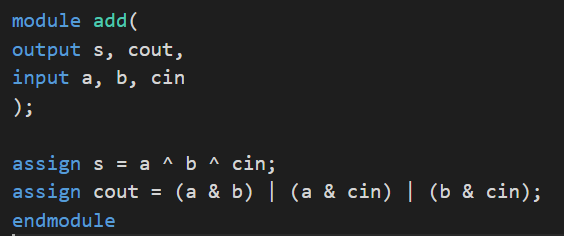
将来自rx\_fifo的数据存储到rx\_byte\_buff，根据rx\_byte\_buff更新命令标志位，例如判断是否为乘法命令的标志位：（其他标志位基本类似）

之后会根据各种标志位进行状态转移。

**（6）加法命令**

根据处理好的两个操作数，调用加法模块，计算结果。8bits加法器代码为：

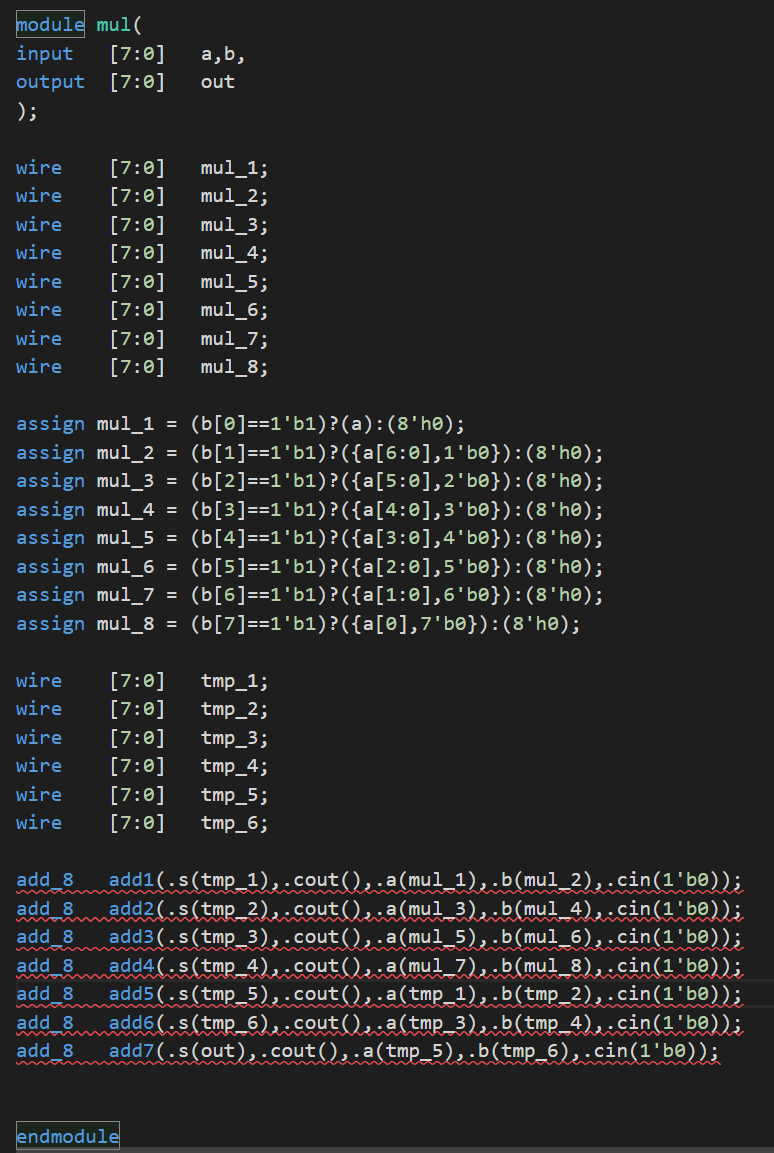




**（7）乘法命令**

8bits乘法器的实现是运用了列竖式的思想，进行二进制数相乘，判断乘数的每个bit是否是1，若是，则将被乘数移位后加入结果.

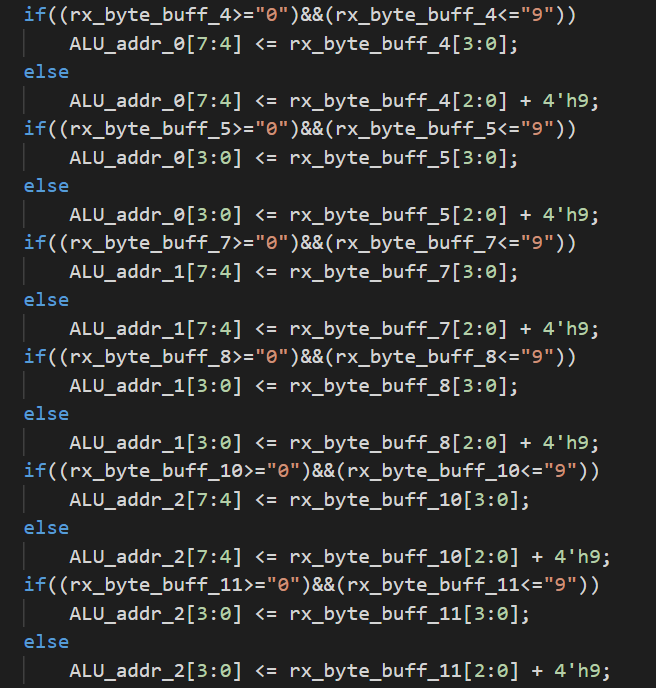
代码如下：



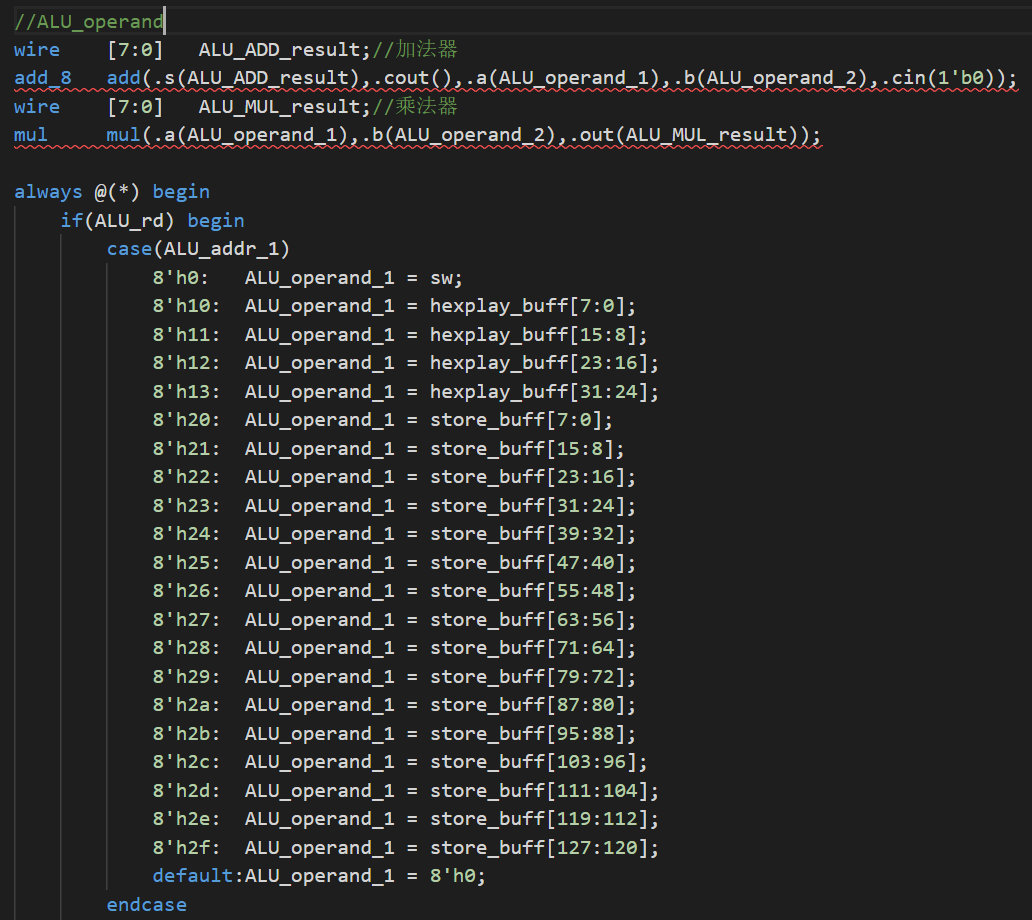
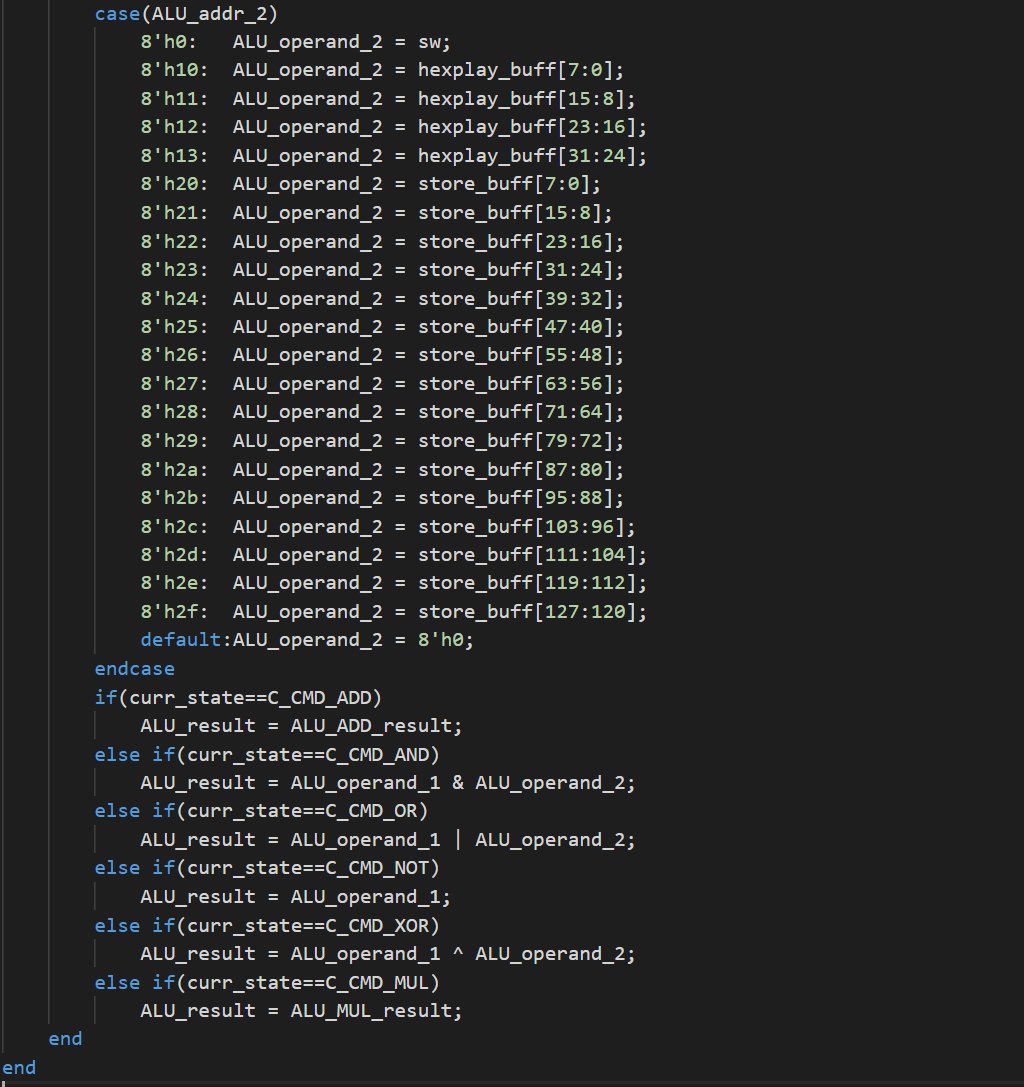
**（8）ALU**

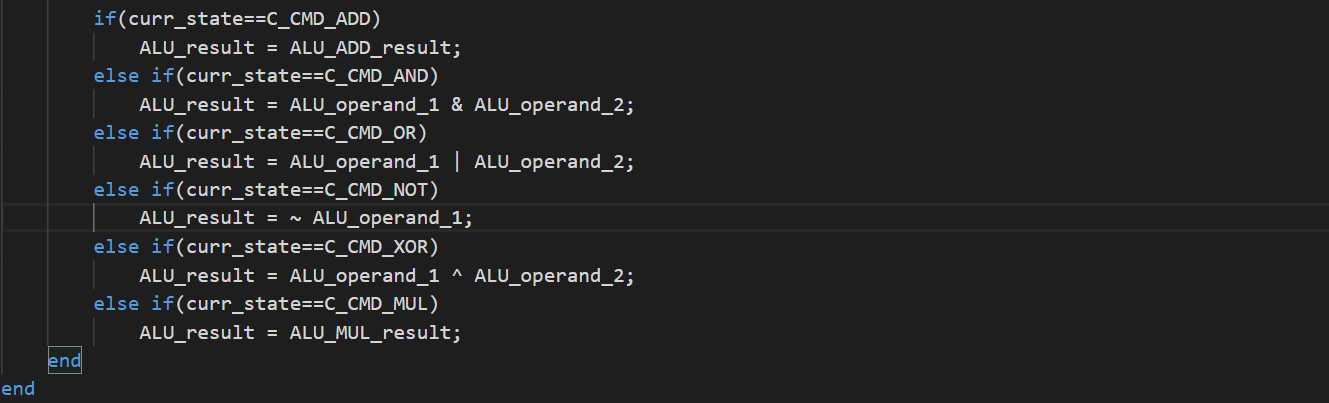
ALU包含加、乘、与、或、非、异或。

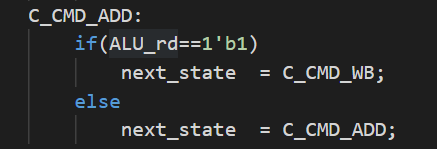
1.根据rx\_byte\_buff获得操作数地址和结果地址的代码如下：（以加法指令为例，其他指令如此类似，在具体细节上存在细微差别）



2.根据操作数，求结果的代码如下：



3.比较重要的一点是，ALU运算过程需要两个时钟周期，也就是是说，所有进行ALU运算的状态都需要两个时钟周期。第一个时钟周期完成根据rx\_byte\_buff更新ALU\_addr，第二个时钟周期内根据ALU\_addr取出相应的操作数，并做运算。第二个时钟周期内也存在先后顺序，即先获得操作数，再进行运算，这个先后顺序用阻塞赋值实现。两个时钟周期依靠标志位ALU\_rd实现，当且仅当已经进入ALU运算的状态后，置ALU\_rd为1。这一部分的次态设置如下（以加法为例）：



也即在进入加法运算状态的第一个时钟周期内，因ALU\_rd此时为0，故次态仍为加法状态。经过第一个时钟周期，ALU\_rd被置为1，所以在第二个时钟周期时，次态就为写操作状态。

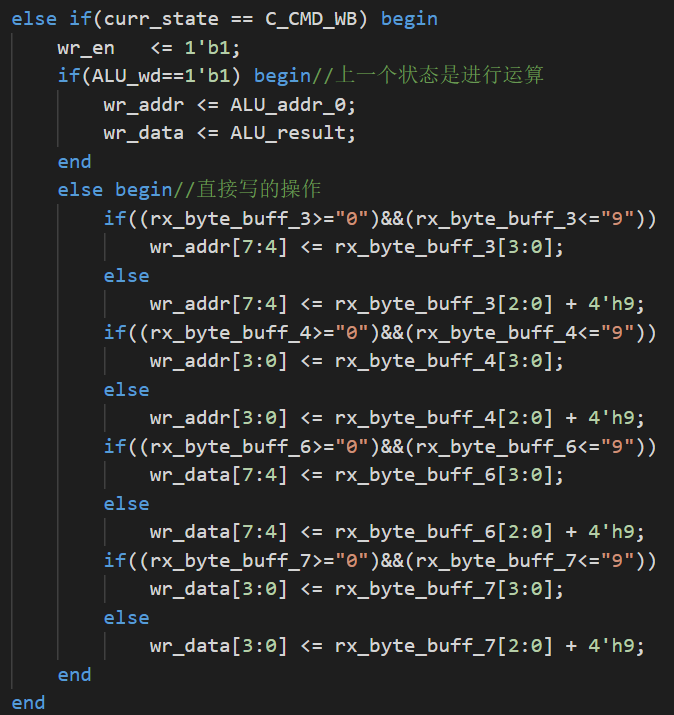
**（9）写命令**

写命令的实行过程包括：更新要写入的目标地址wr\_addr，获得要写入的数据；根据wr\_addr将wr\_data写入相应的地址。

需要特殊处理的是，在扩展了ALU指令后，进入到写操作状态有两种情况。一是来自Shell的命令就是写命令，二是来自Shell的命令是运算指令，运算过程执行完后，需要将运算结果存储到目标地址时，也会进入写操作状态。

为区分这两个不同情况，设置标志位ALU\_wd，ALU\_wd为1，表示上一个状态是运算状态，此时要写入的数据存储在ALU\_result中，要写入的目标地址存储在ALU\_addr\_0中。ALU\_wd=0，表示当前命令就是单纯的写命令，要写入的目标地址和数据均来自Shell，现存储在rx\_byte\_buff中。

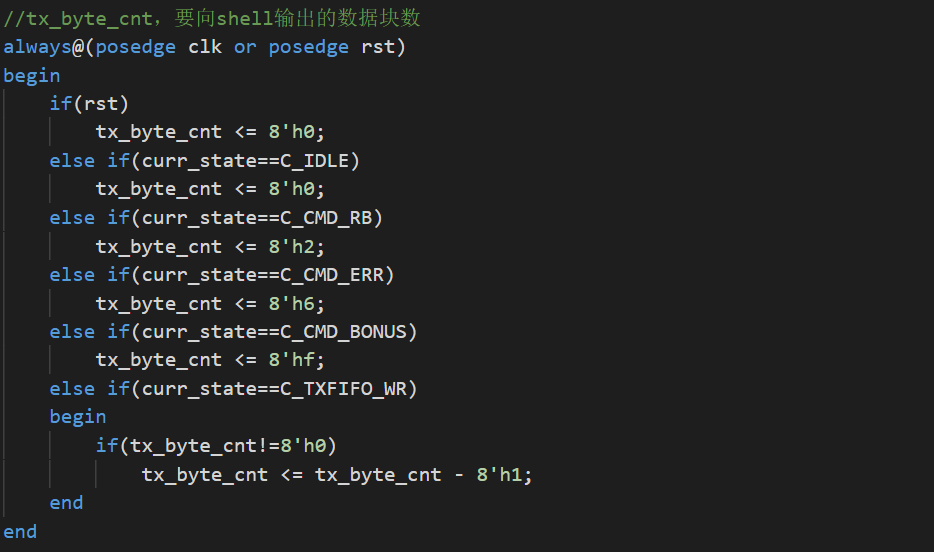
所以在写操作状态中，更新wr\_addr和wr\_data的操作如下：

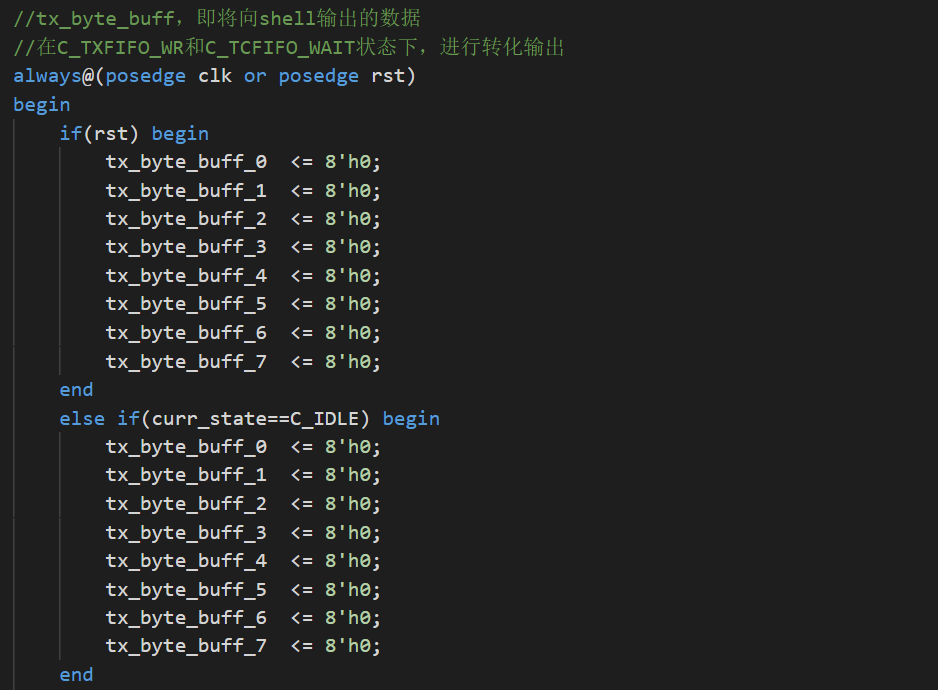


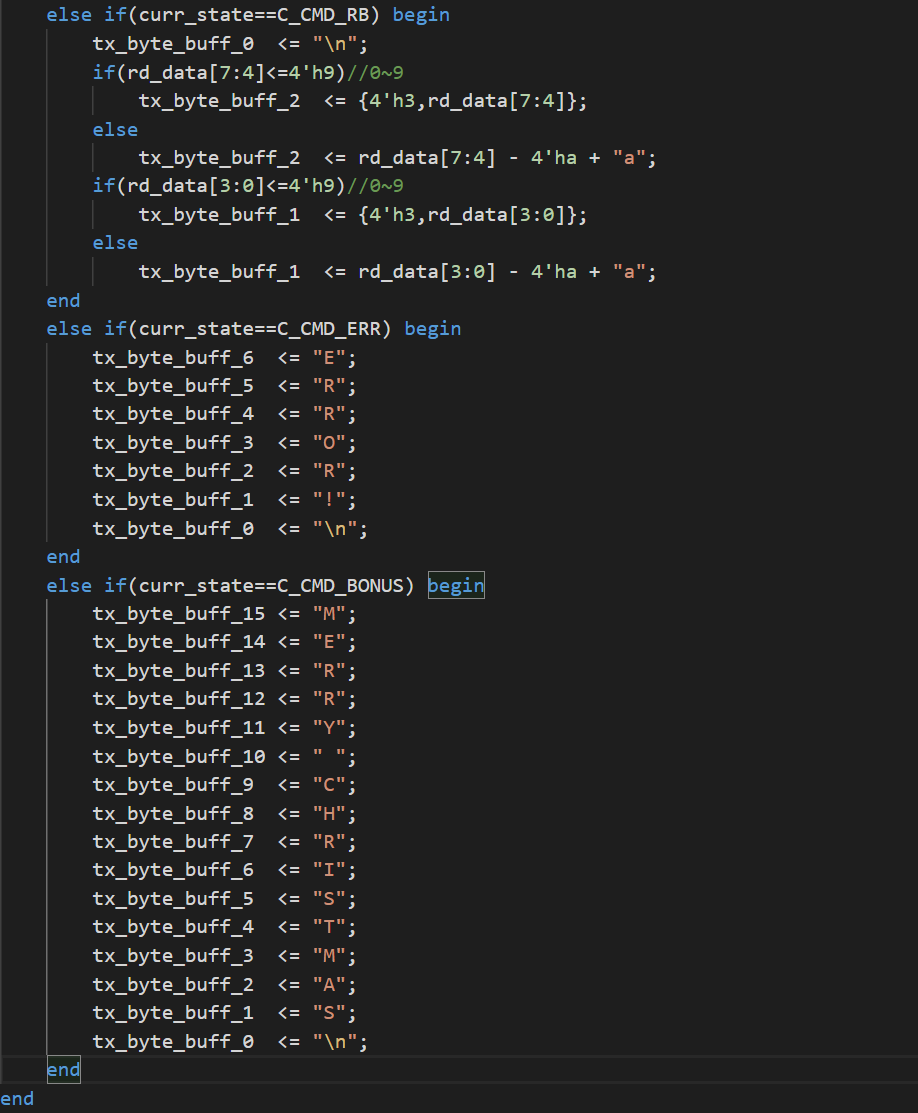
**（10）为向Shell输出做准备**

要输出的数据个数存储到tx\_byte\_cnt中，要输出的数据存储到tx\_byte\_buff中。这个过程在进入C\_TXFIFO\_WR之前完成。

代码如下：







进入C\_TXFIFO\_WR状态后，tx\_byte\_cnt每个时钟周期减一，实现将tx\_byte\_buff逐个写入tx\_fifo中。每写完一个就会开始输出，将输出送入tx模块转化为串口支持的数据。

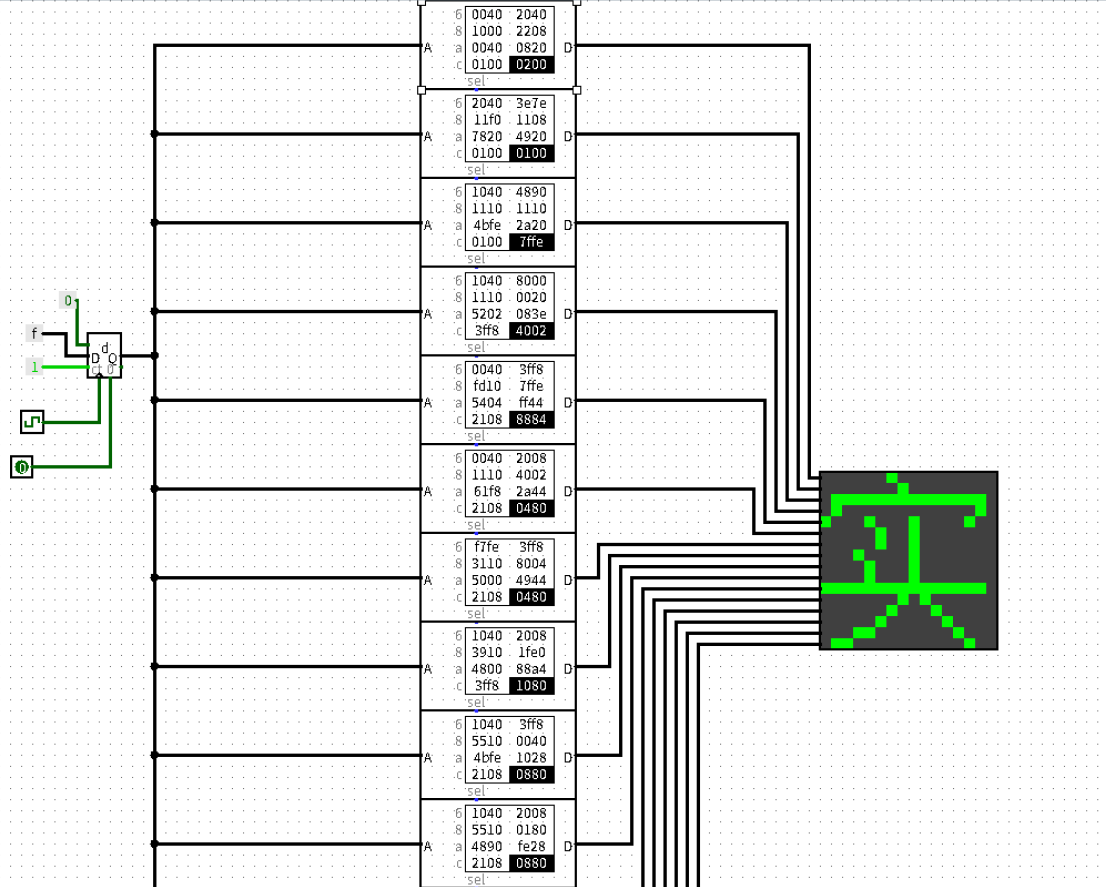
最后在tx\_byte\_cnt=0但tx\_fifo中的数据仍没有全部输出时，状态进入C\_TXFIFO\_WAIT状态，将tx\_fifo剩下的数据全部输出。

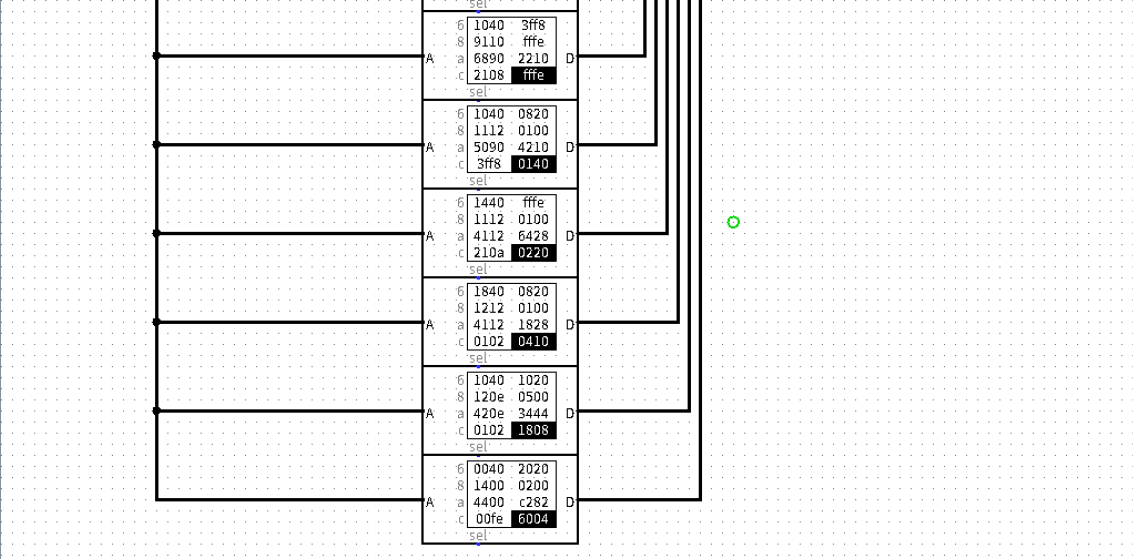
**二、在Logisim中通过LED点阵实现汉字的循环显示**

要在16\*16的LED点阵中显示一个汉字，需要16根16bits线作为输入。而要实现循环输出16个汉字，则需要每个16bits的线可以循环输出16个16bits数据。故使用ROM存储16\*16bits的数据，共需要16个ROM。而汉字循环过程用计数器实现。

在汉字转换成16\*16点阵的在线转换器上将要显示的汉字转化成16进制数，然后每一行存储到不同的ROM中。

引入计数器，进行0~15循环计数，实现每个ROM的16个数据的循环输出。如下图：





向计数器引入时钟信号，在合适的频率下，可实现速度适中的汉字循环显示效果。

【总结与思考】

1.本次实验，学习到了如何通过串口实现与FPGA实现数据交互，如何在没有同步时钟信号的情况下，实现数据的采样。

2.学习了FIFO IP核的使用，它存储的数据的特点是先入先出，实现了无法同步处理的数据的暂存。

3.学习了在较大规模的Verilog程序中，状态机的使用，本次实验的核心实现逻辑在于状态机的设计与实现，根据不同条件实现状态的转移，在不同状态内完成相应的操作，实现了将一个大的任务拆分成容易实现的子任务。

4.在本次实验，对Verilog语言的使用进行了大量的联系，对于Verilog与计算机高级语言的区别有了更深入的认识，其中最重要的一点就是它的不同Always块是并行执行的，代码书写的先后顺序并不影响执行的并行性。如果需要引入先后顺序，则需要加入标志变量或者拆分为不同的状态。

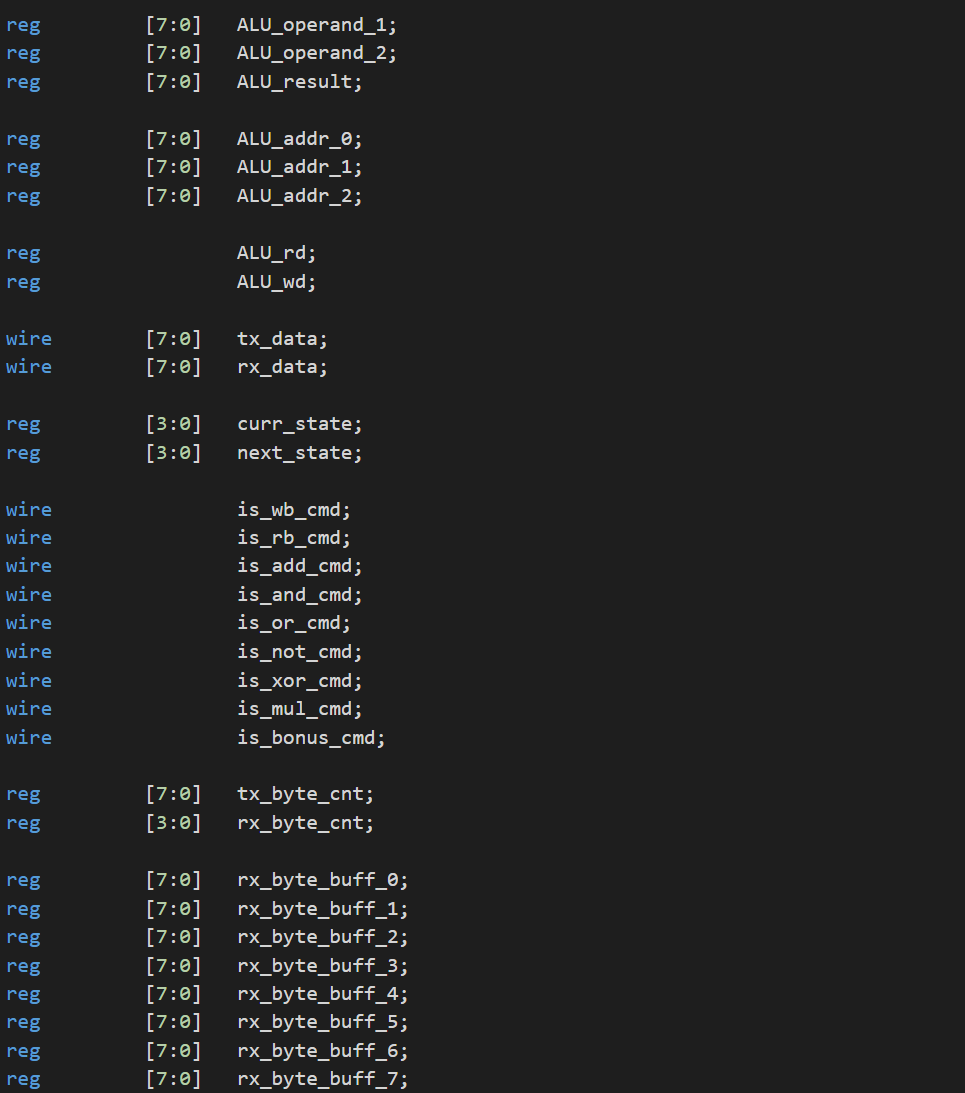
5.本次实验在任务量和代码量都远超过去每一次实验，难度也比较大，作为综合实验也是具有一定的挑战性。

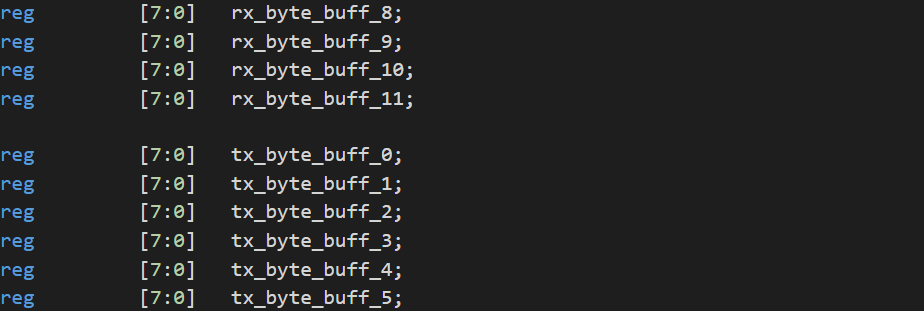
6.建议在今后的课程中，加入对IP核使用的介绍，并且讲解一下综合实验。

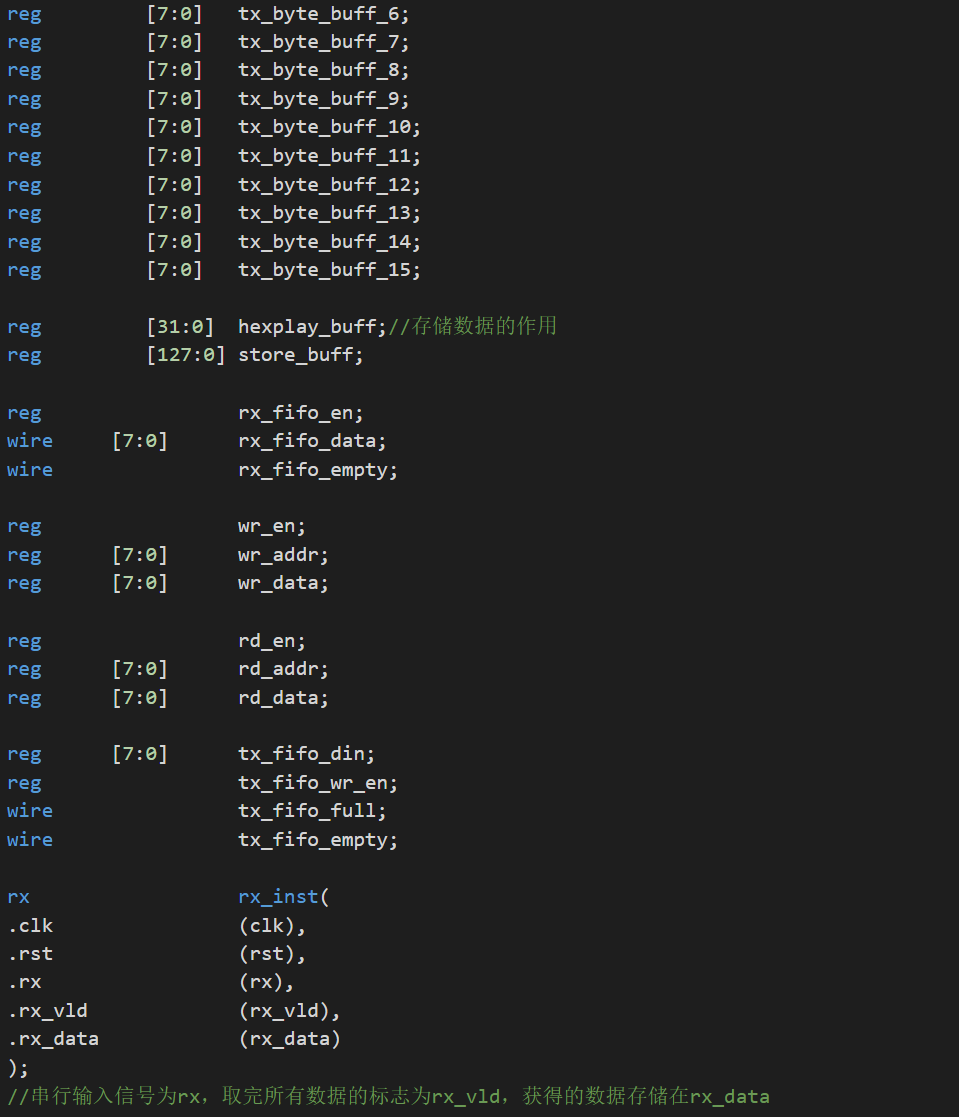
【附录：完整代码】

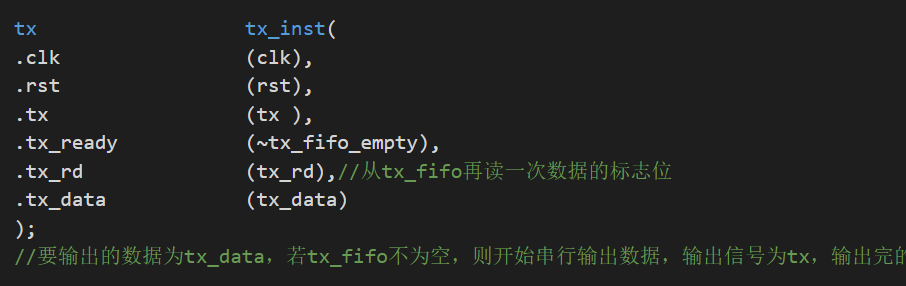
**lab10\_v.v**

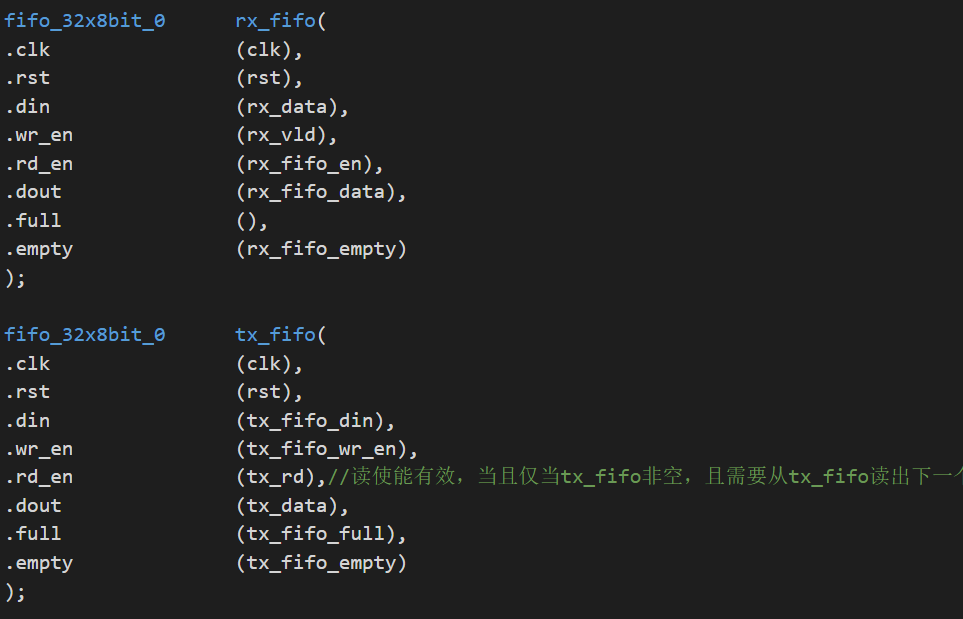


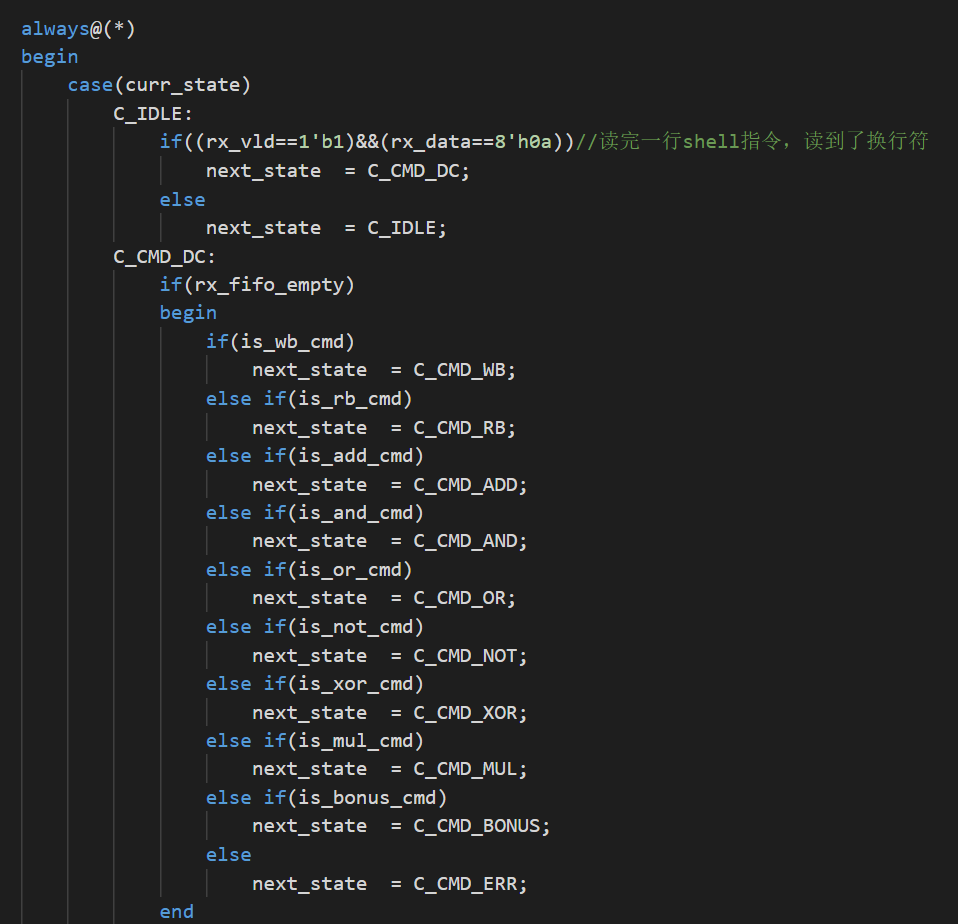




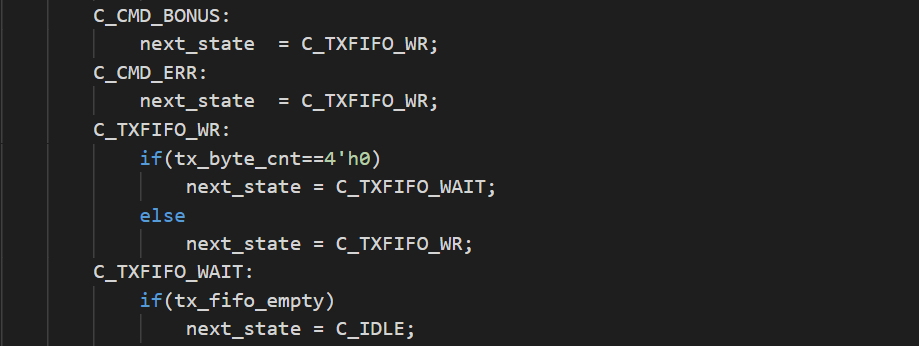


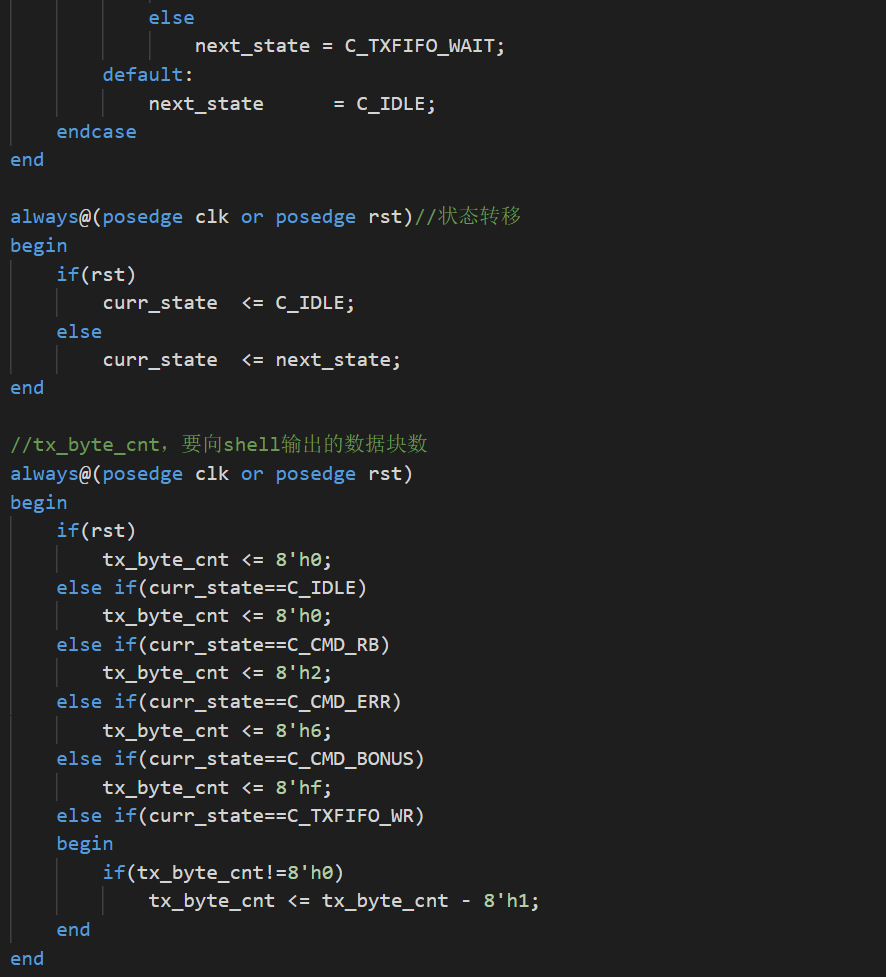


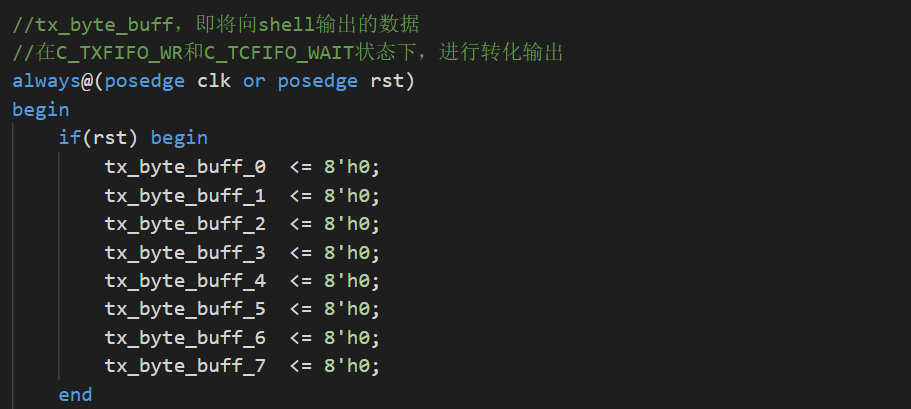


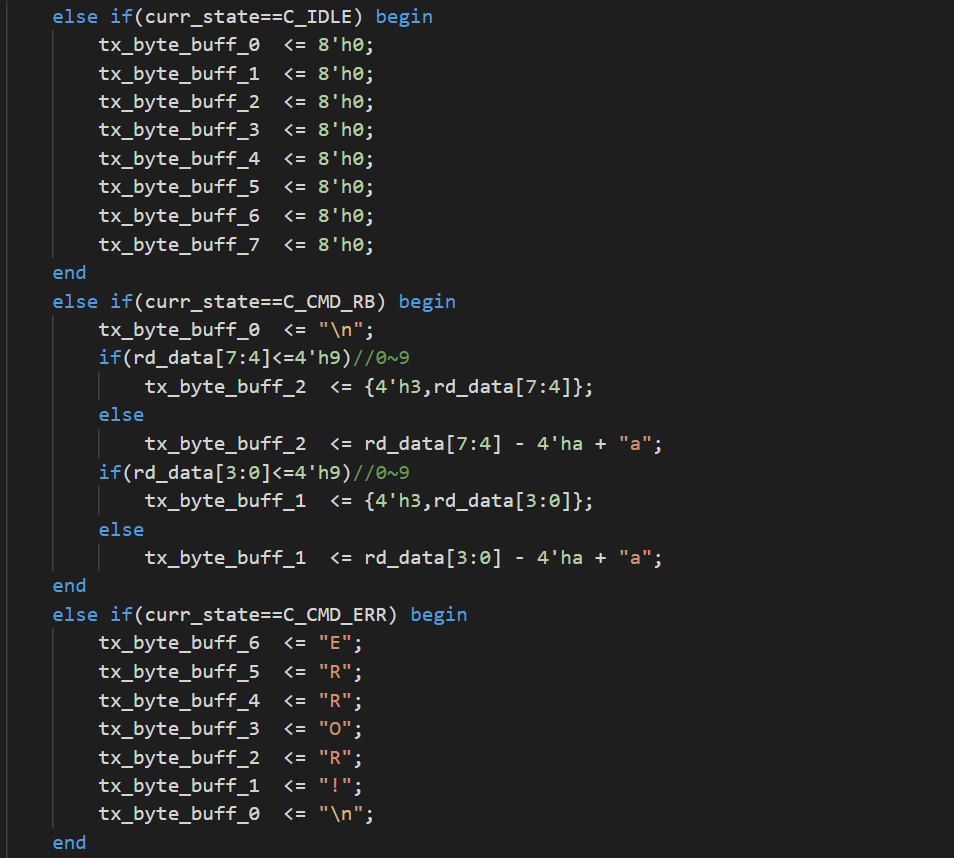


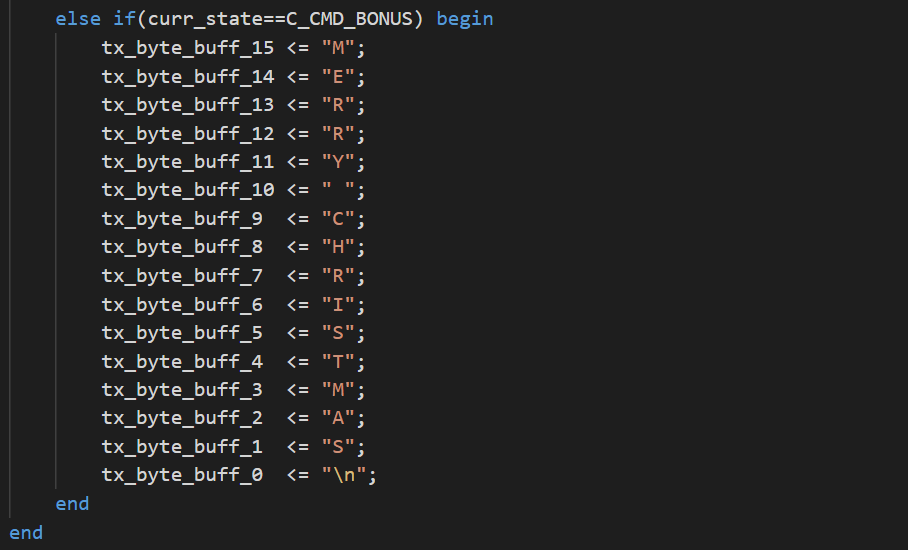


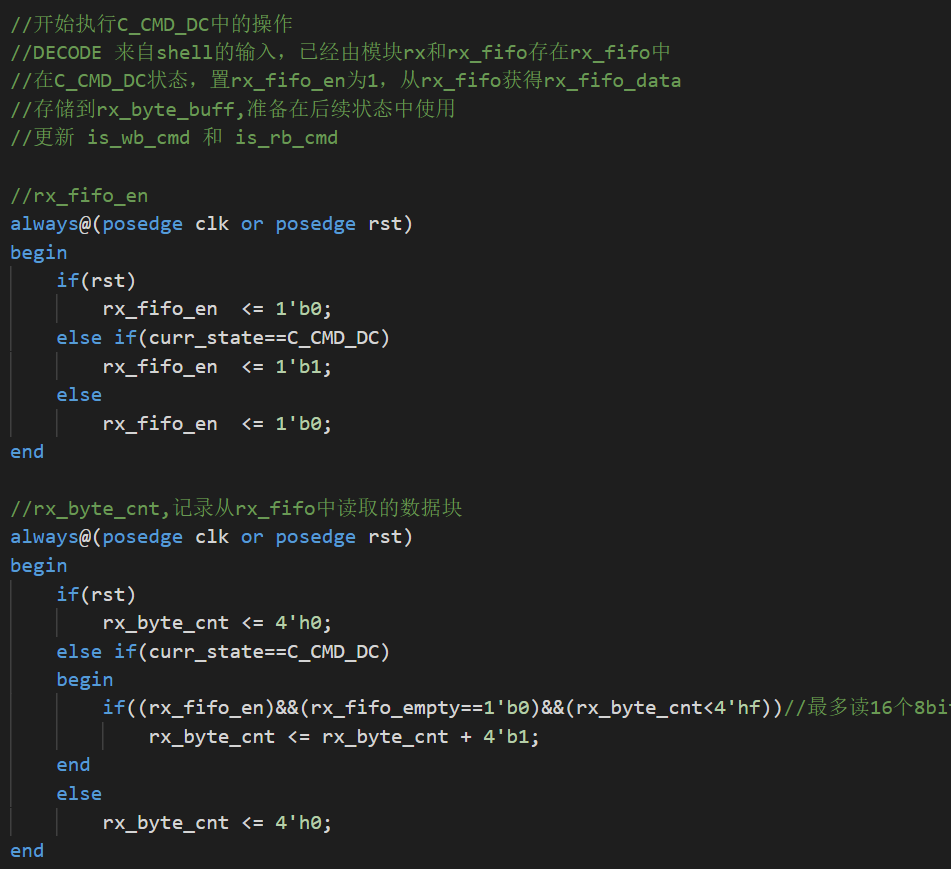


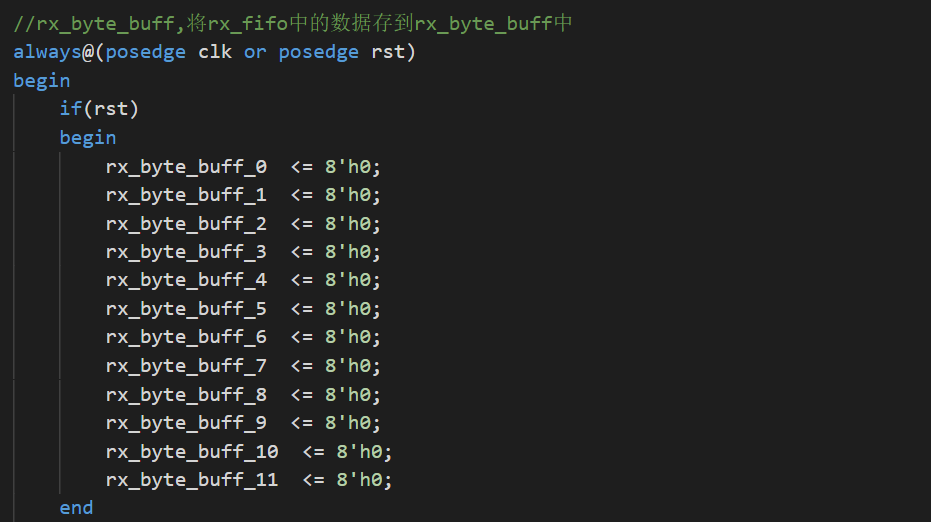


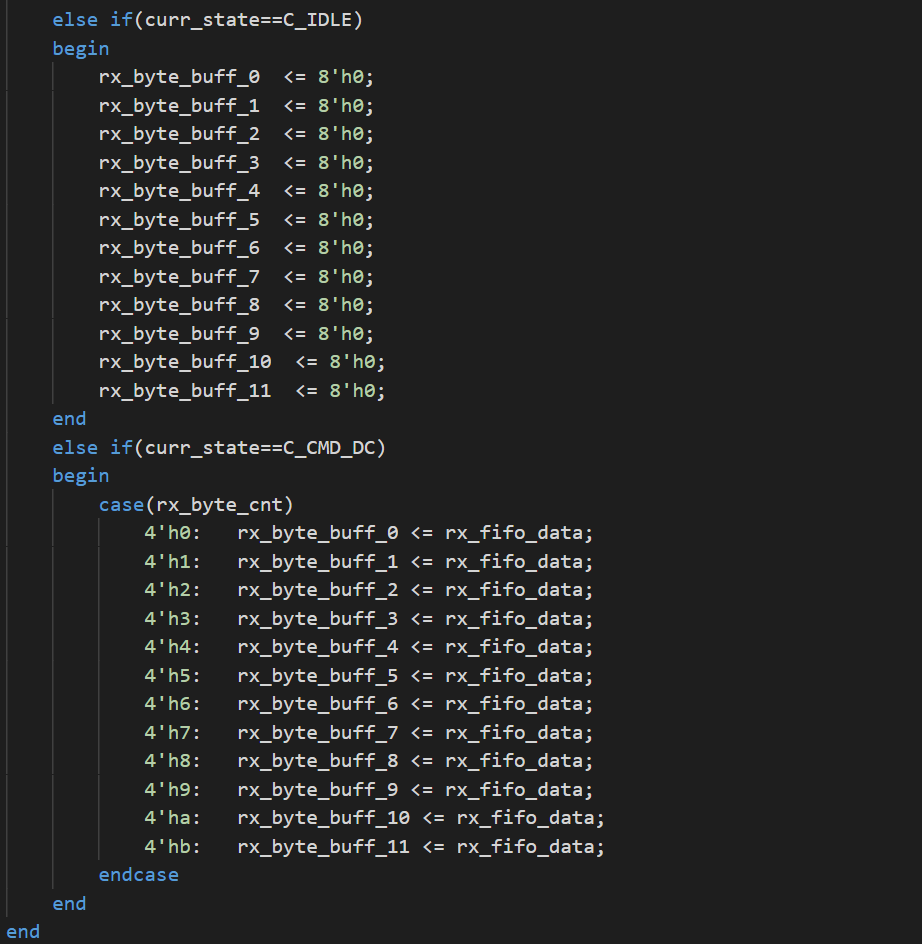




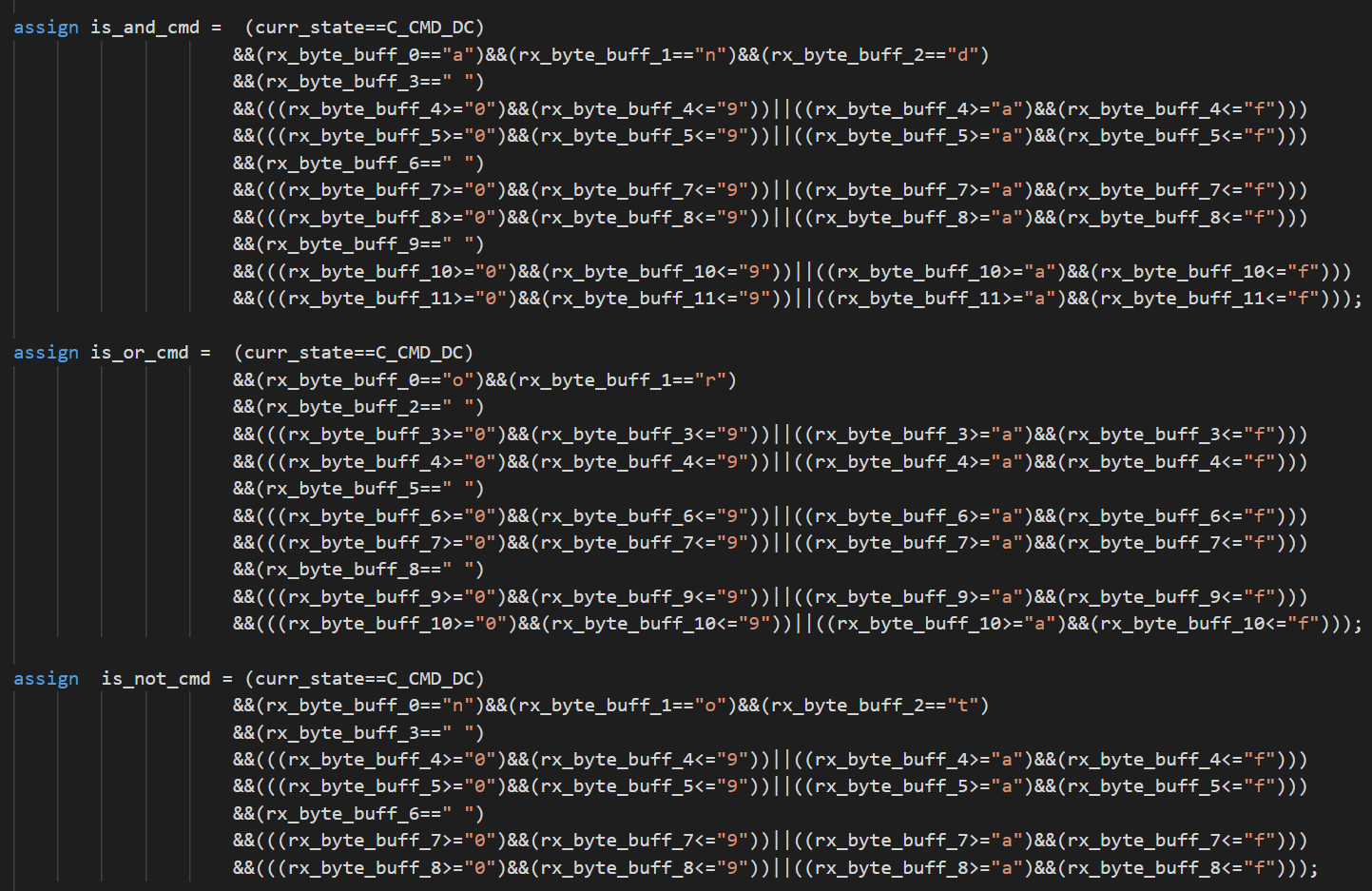


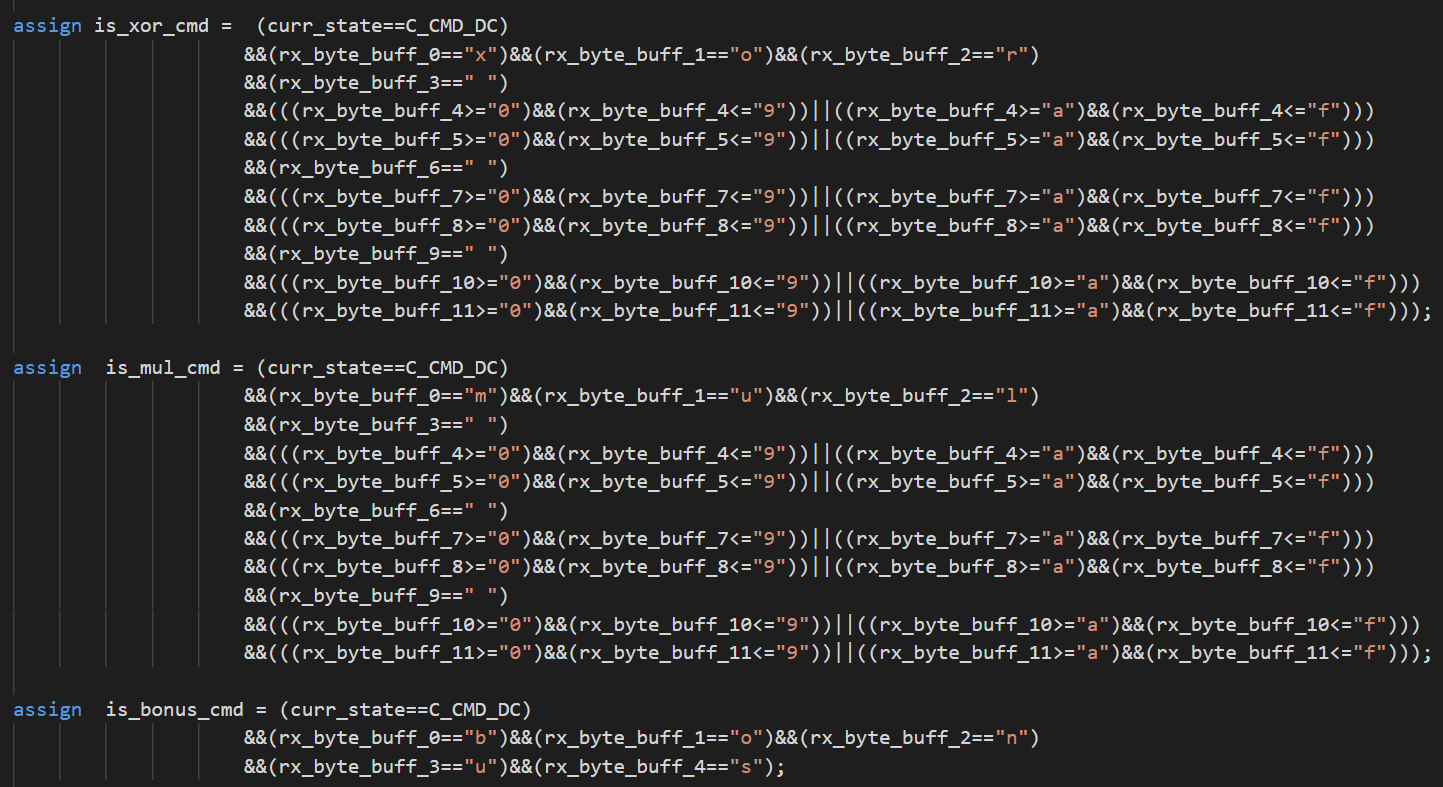




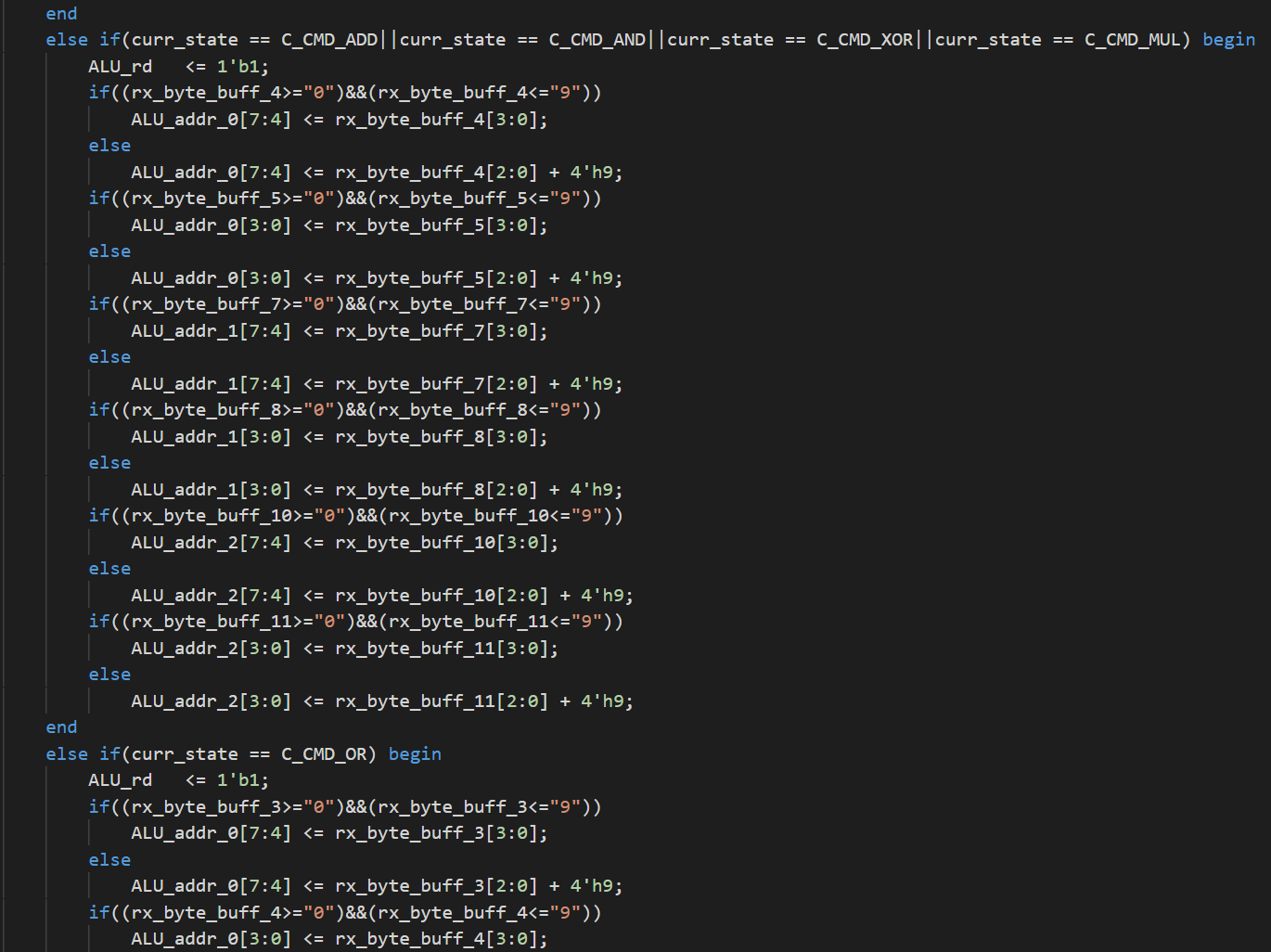


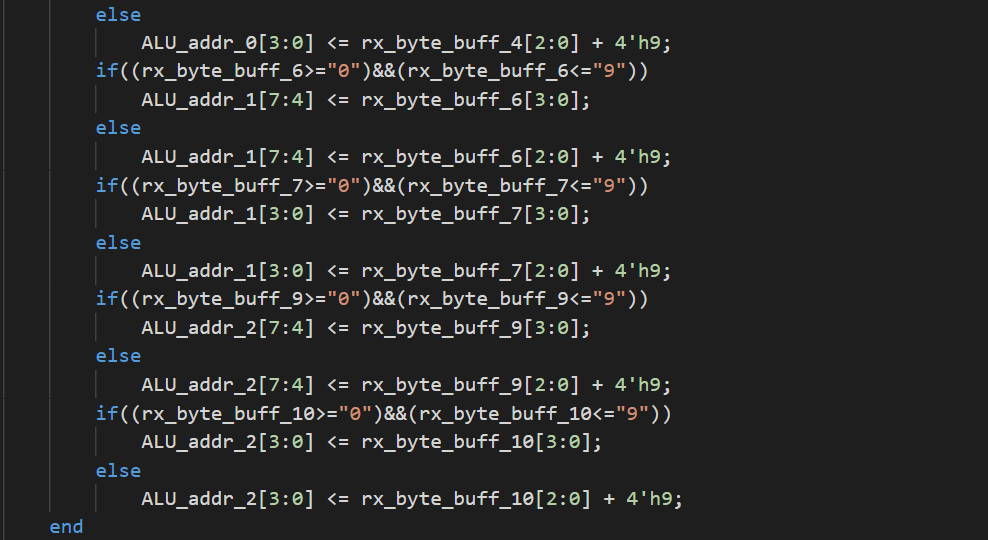


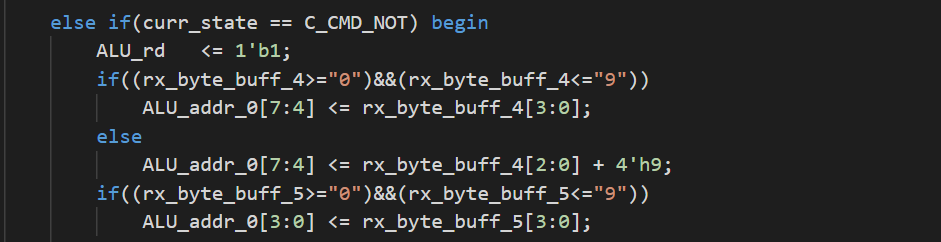


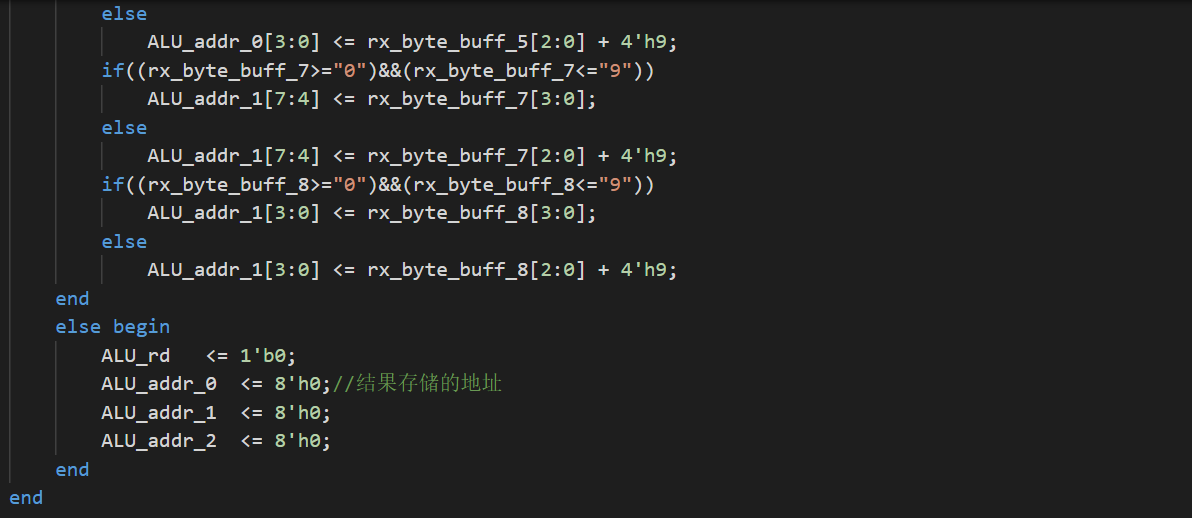


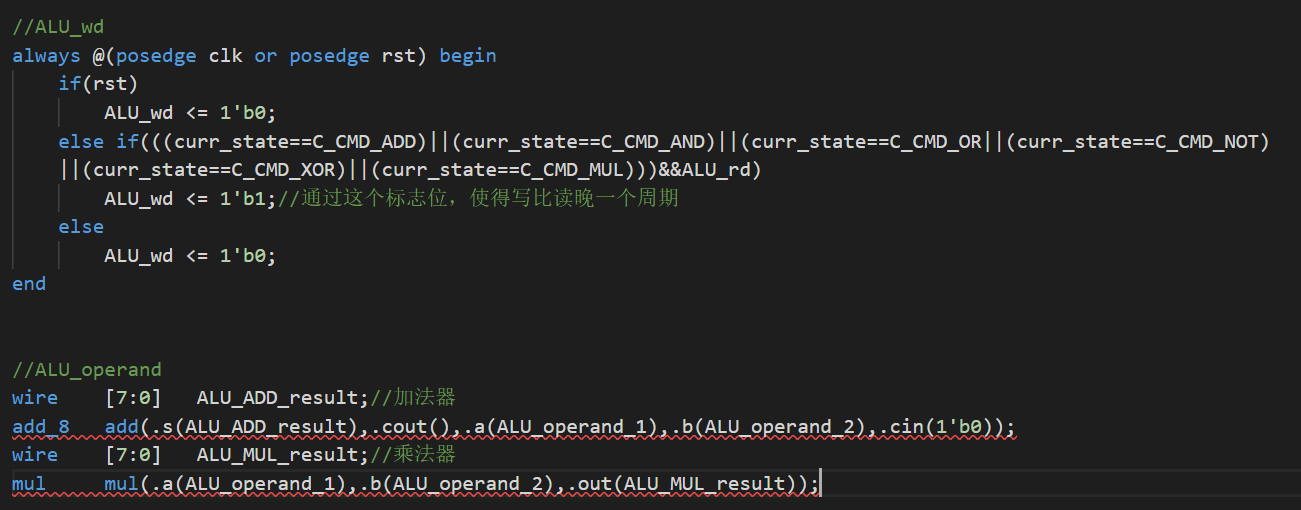


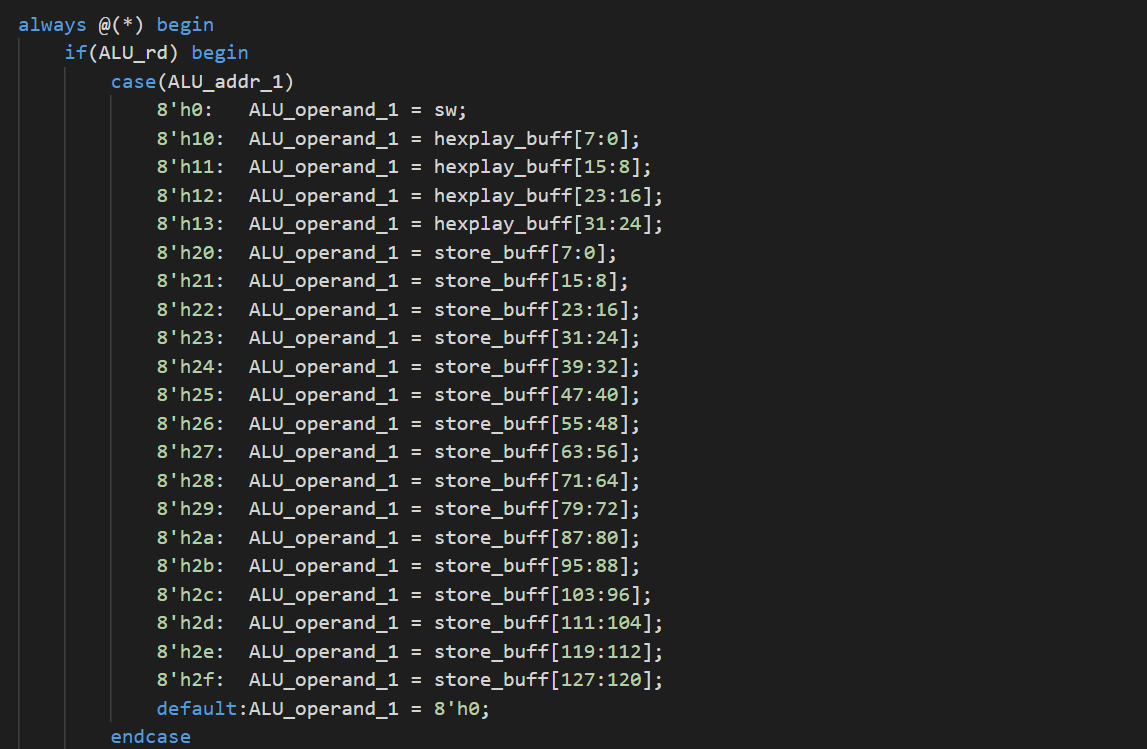


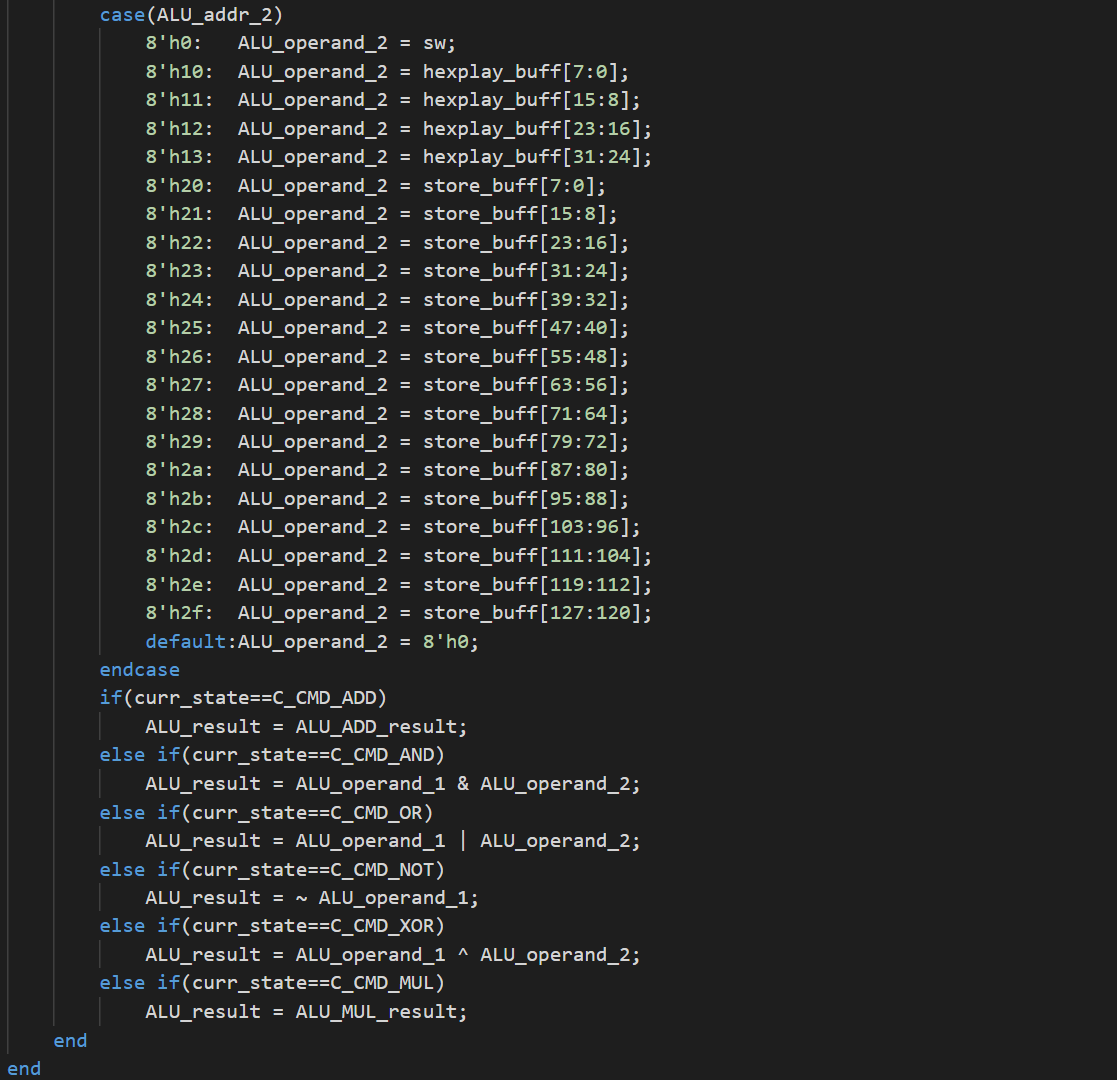


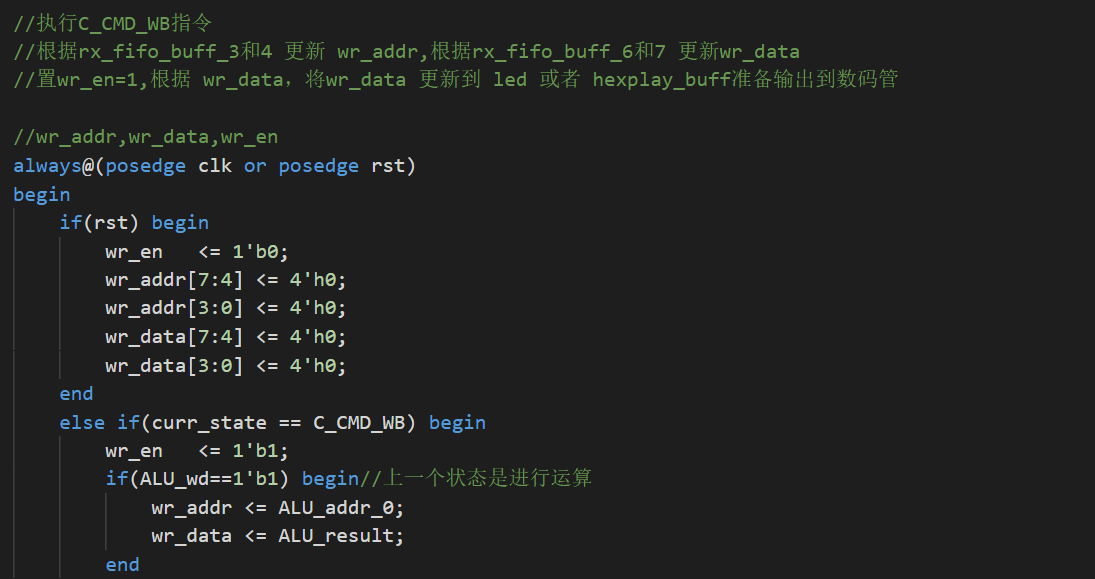


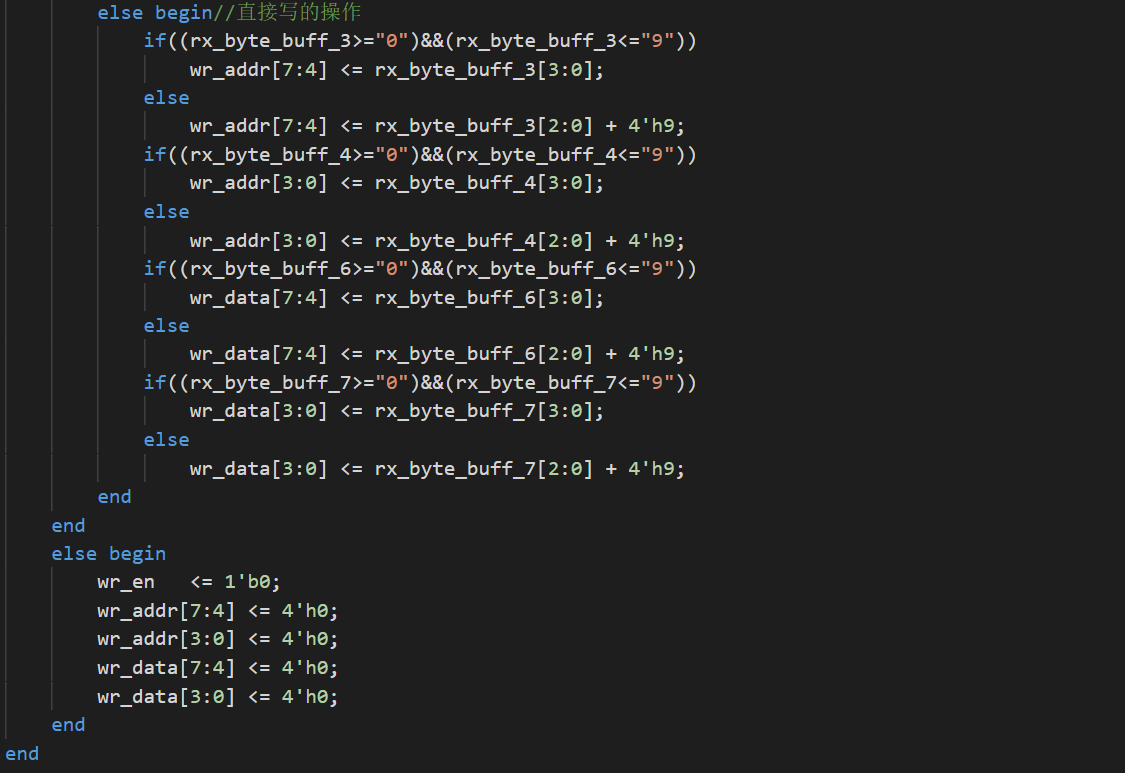




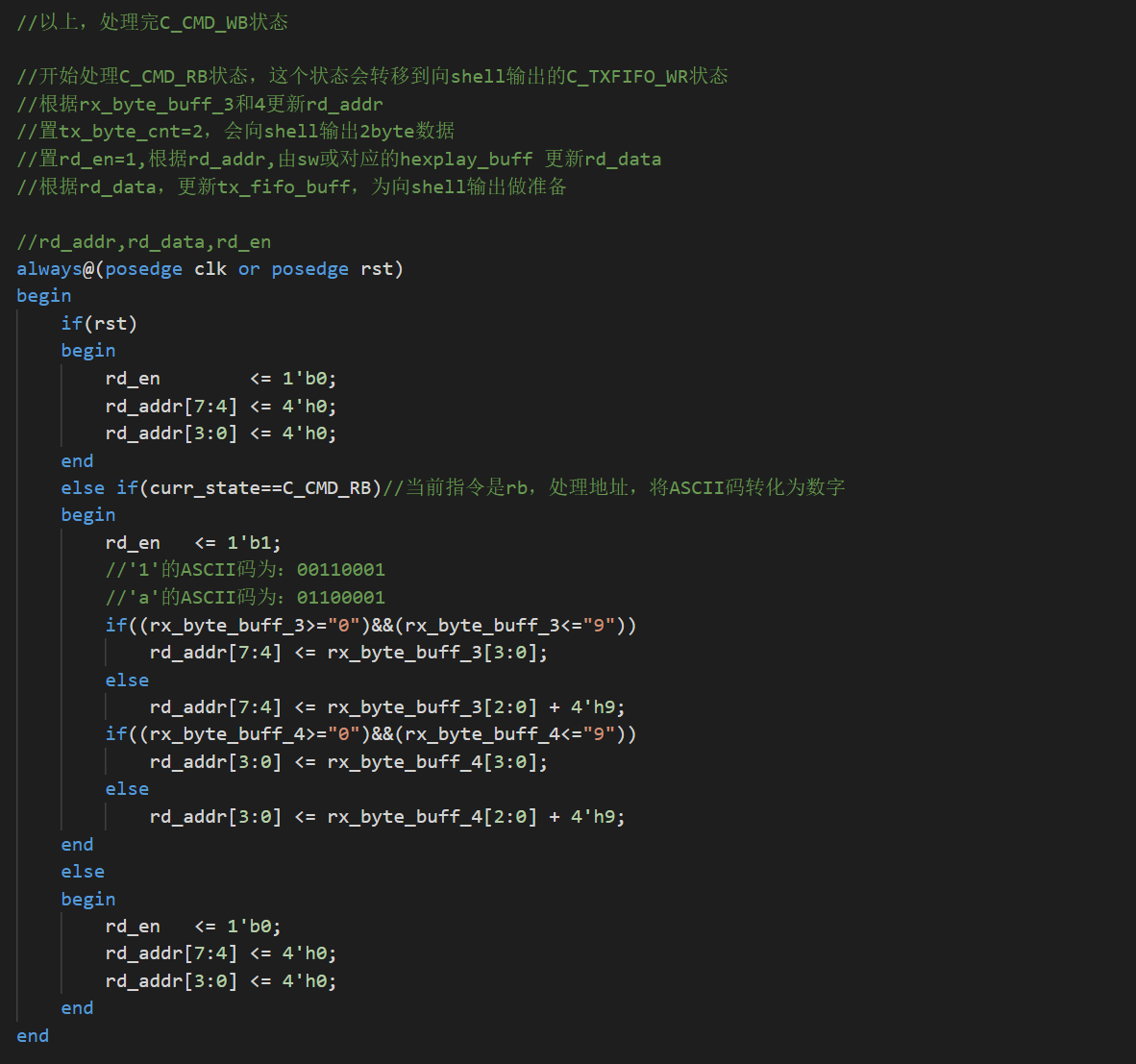


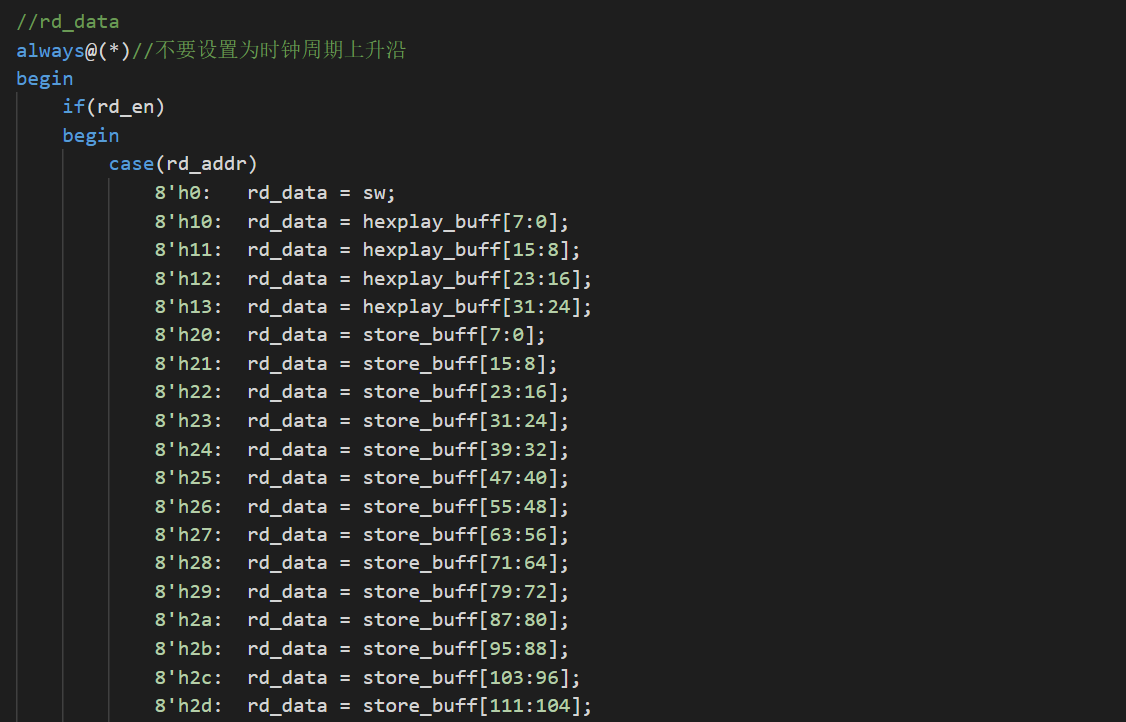


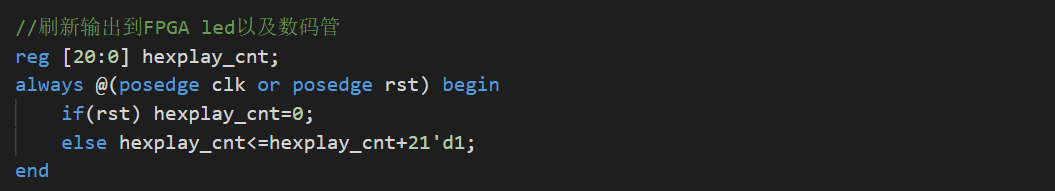
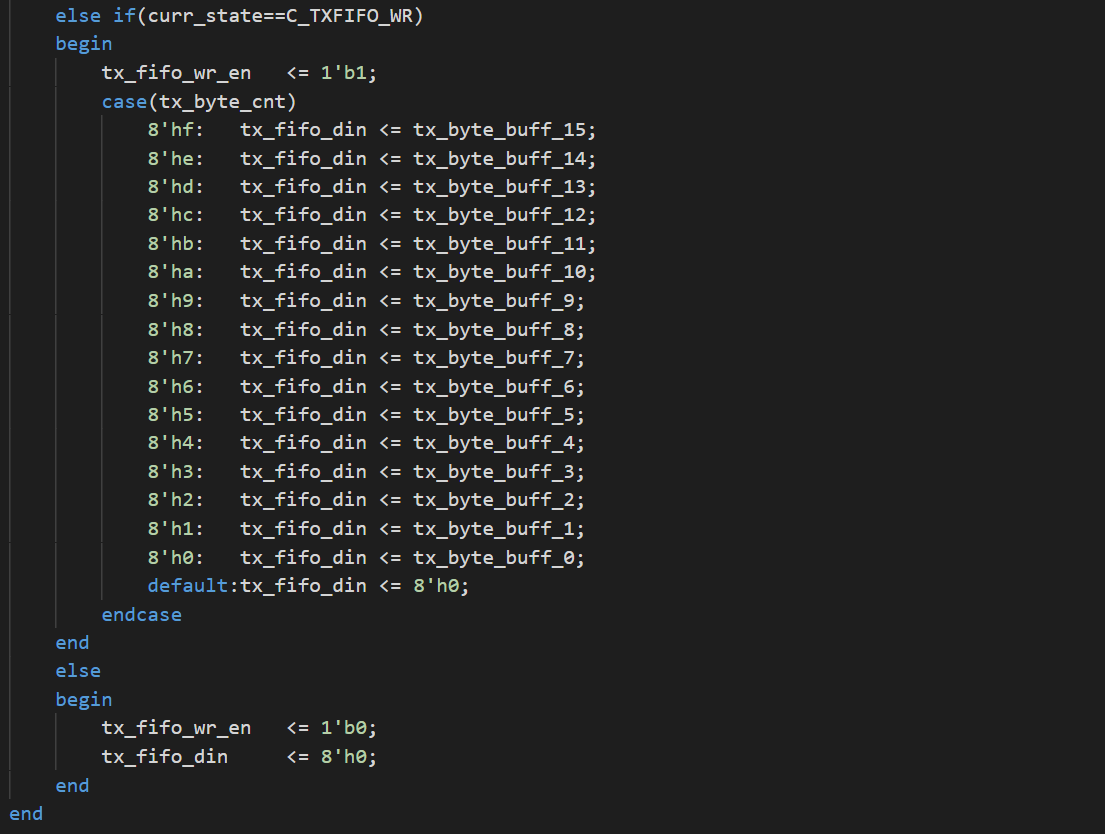
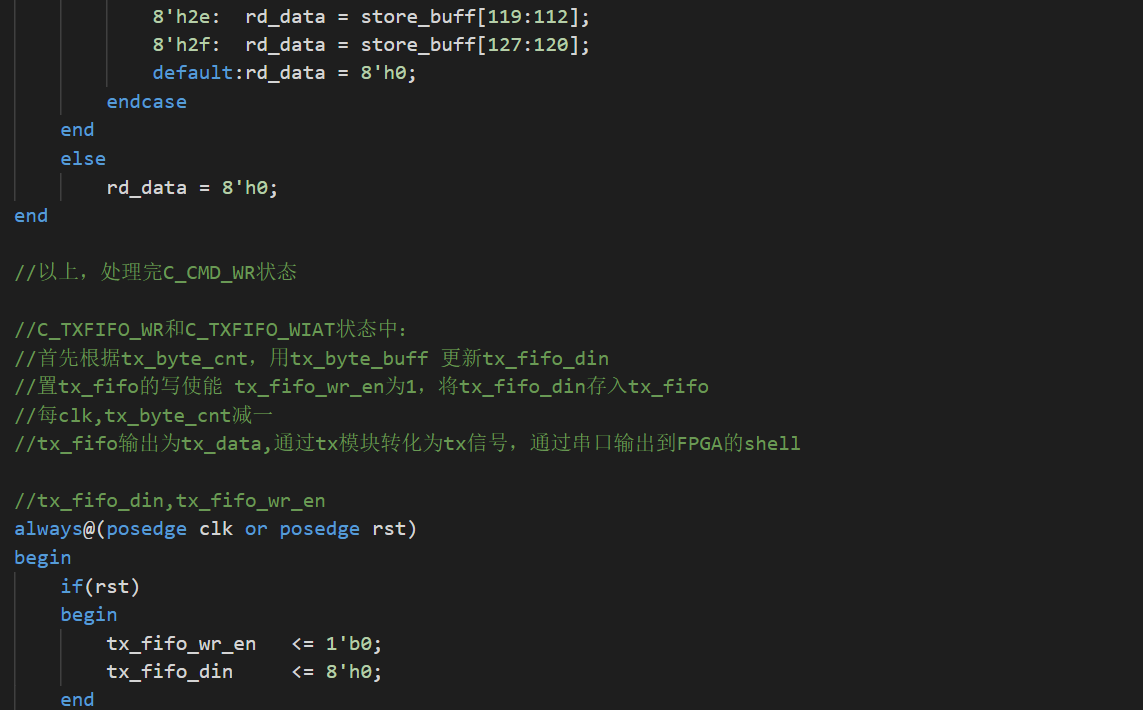


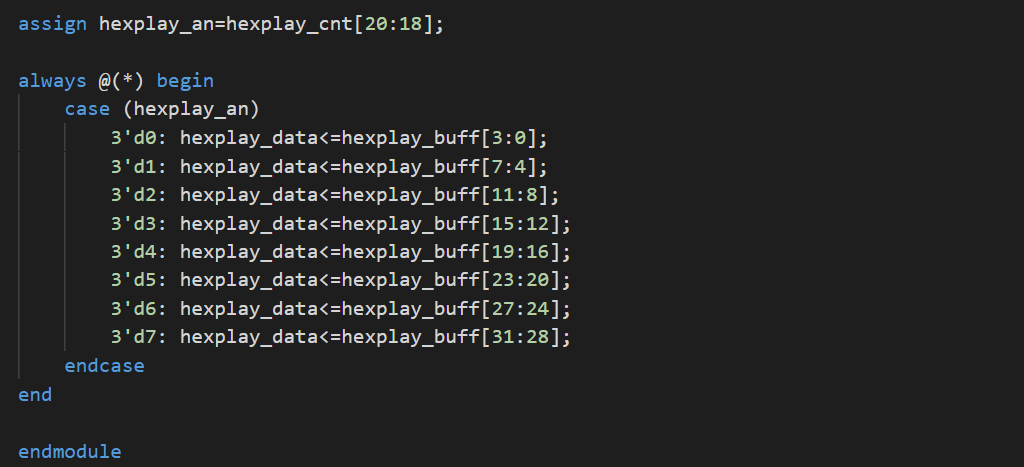




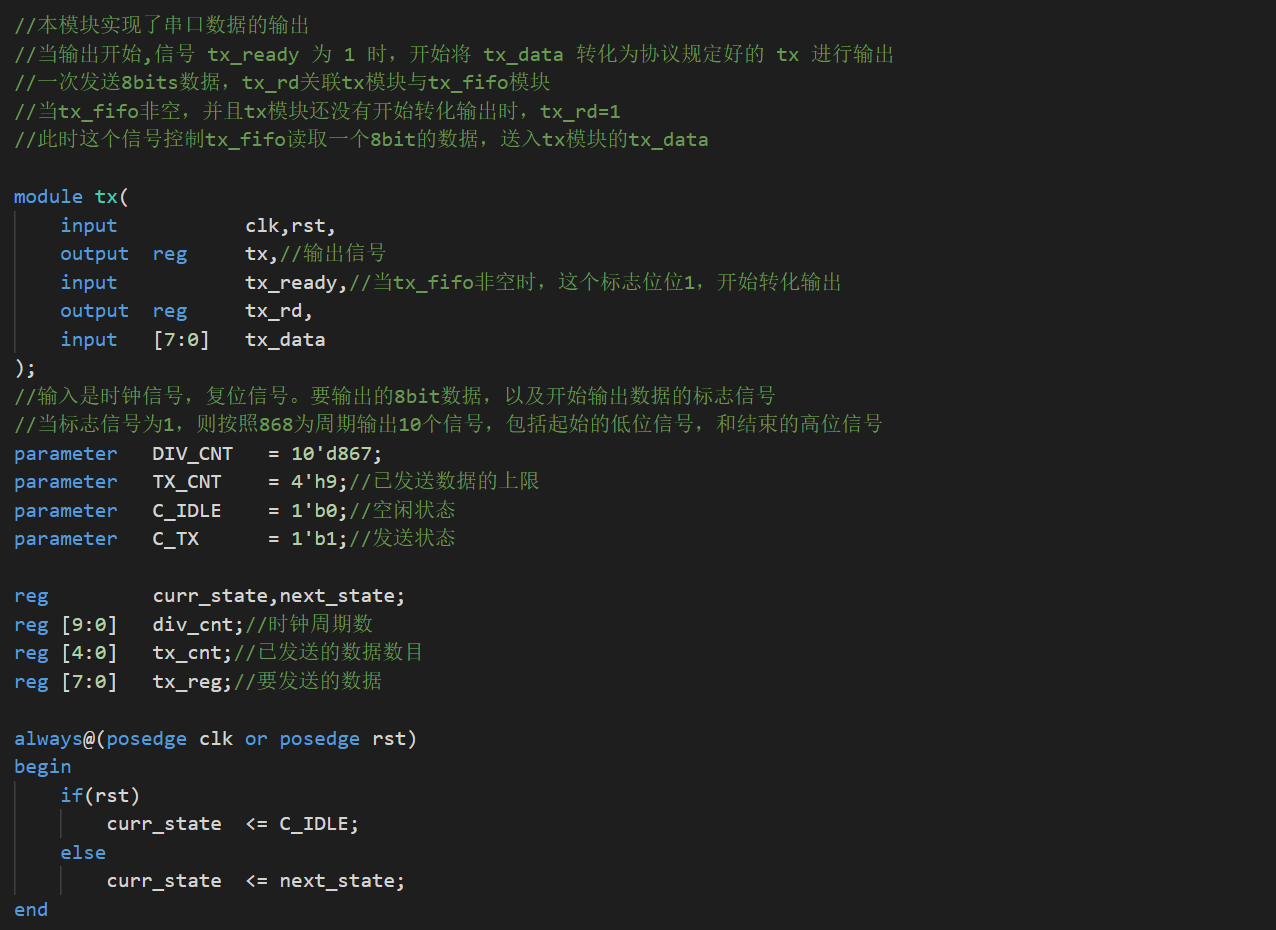


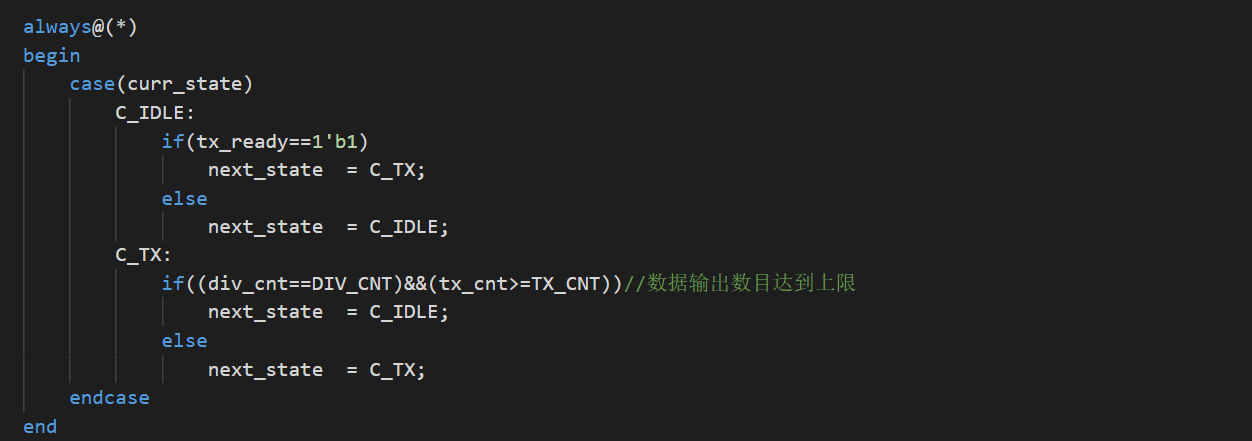


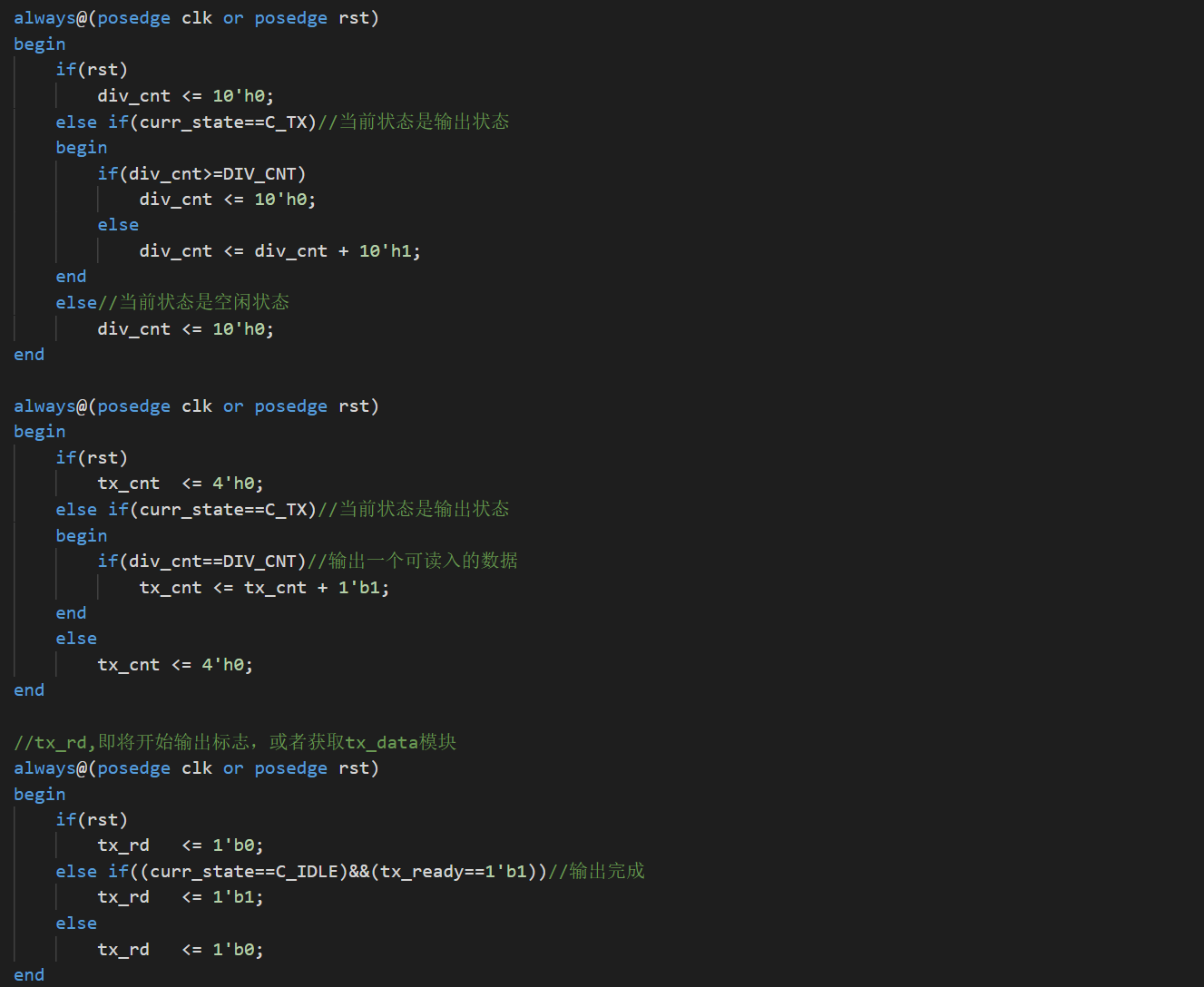


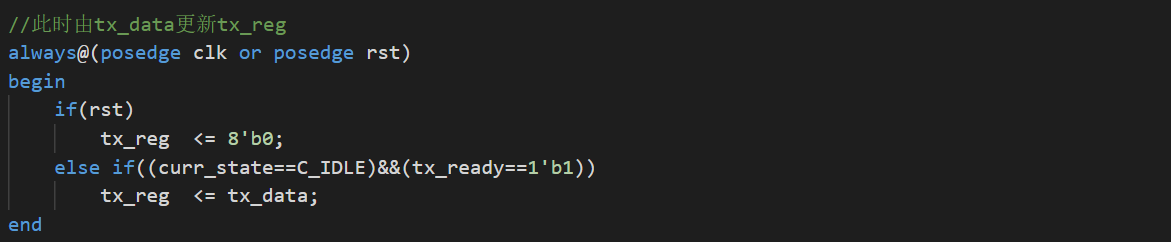


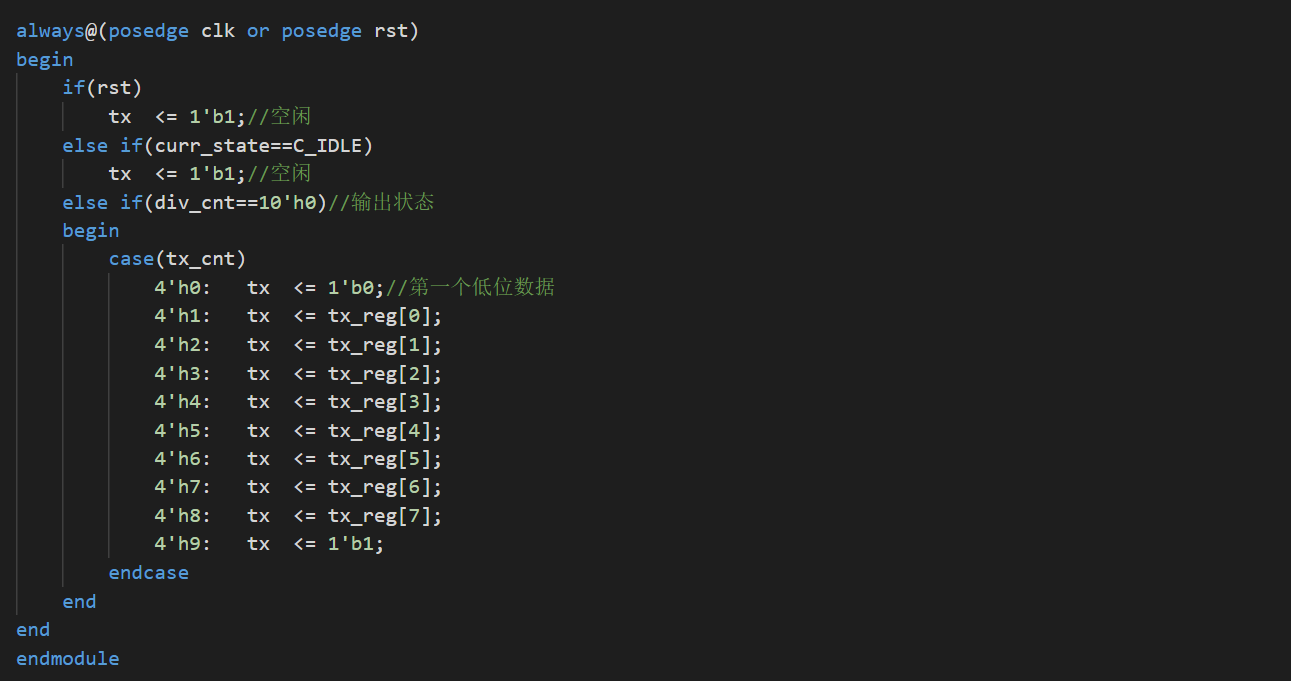
**tx.v**



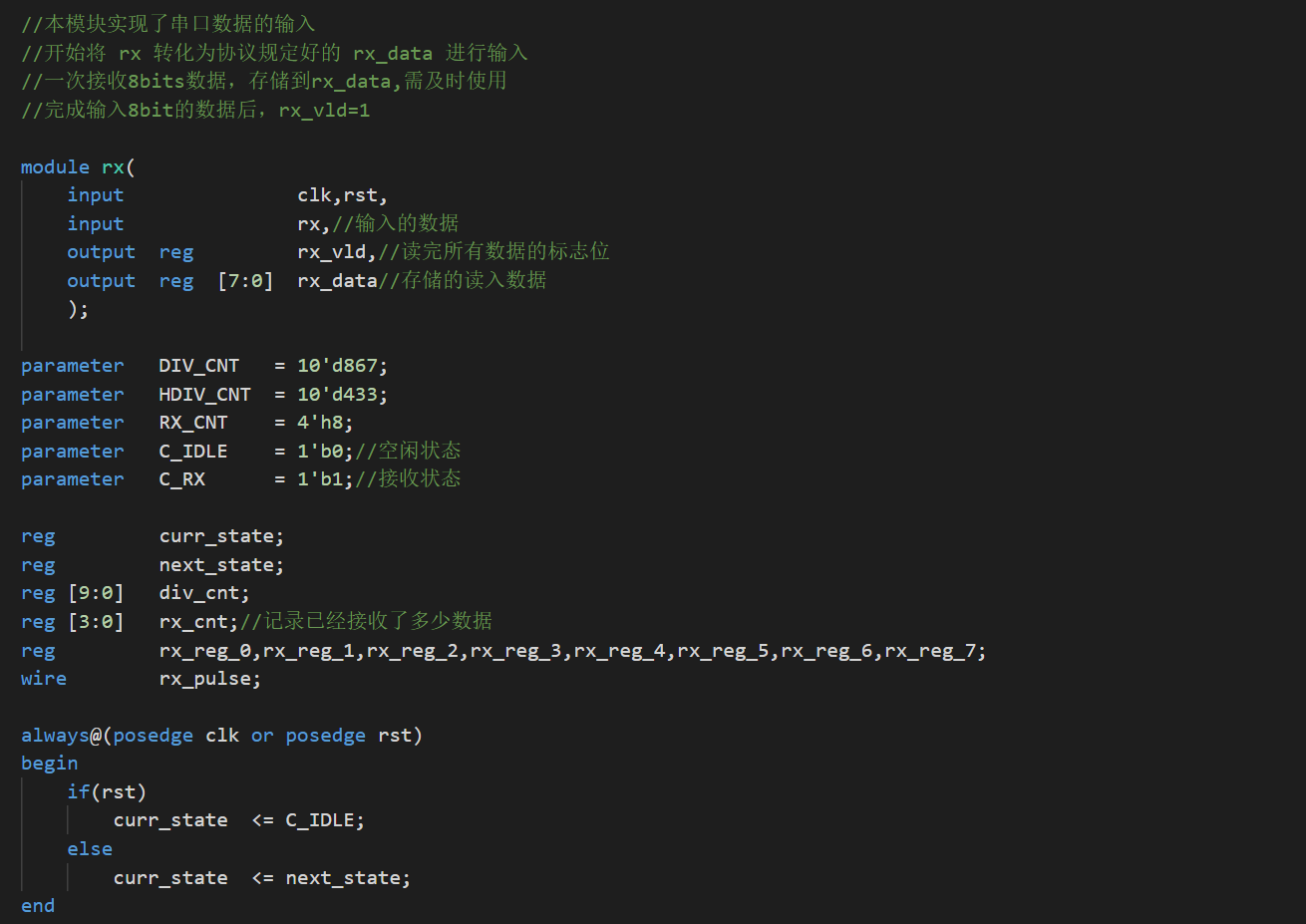


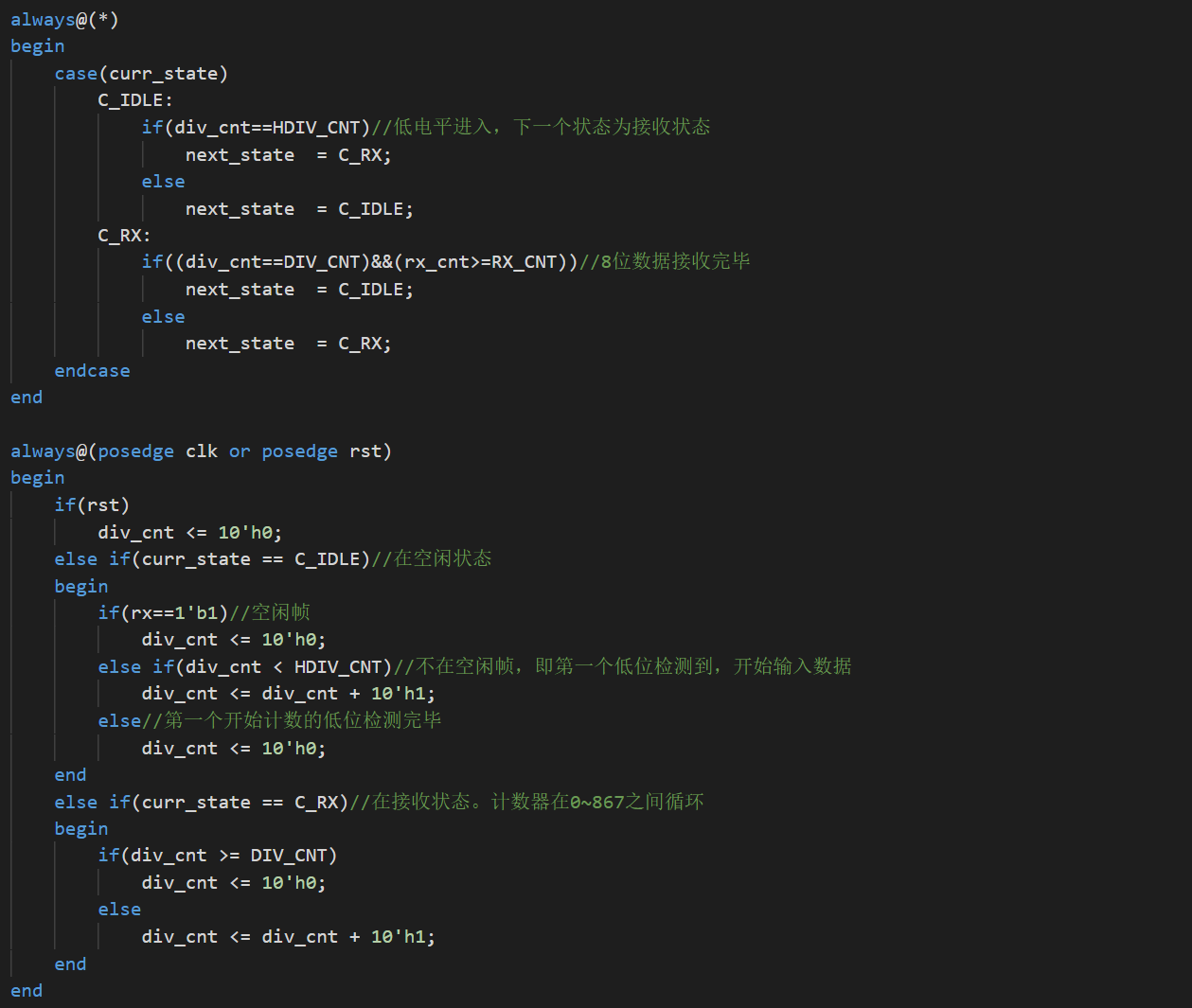


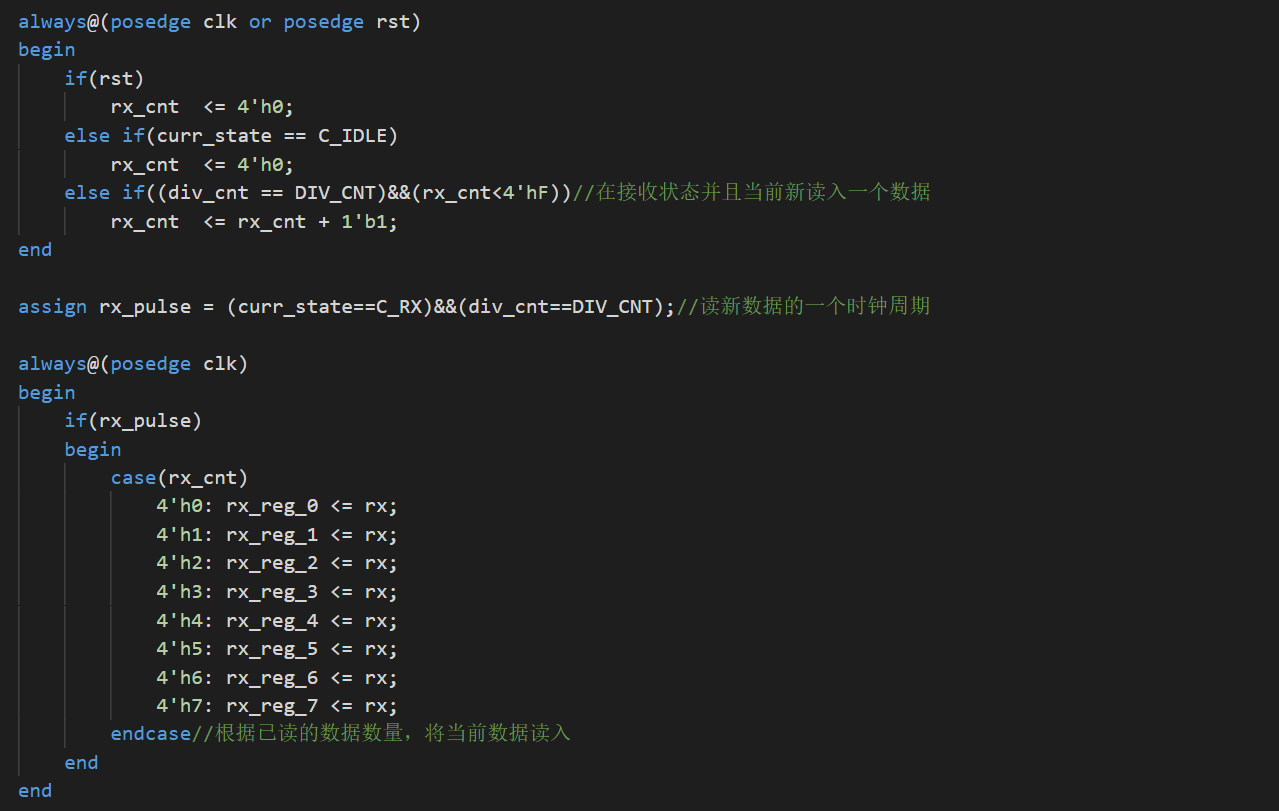


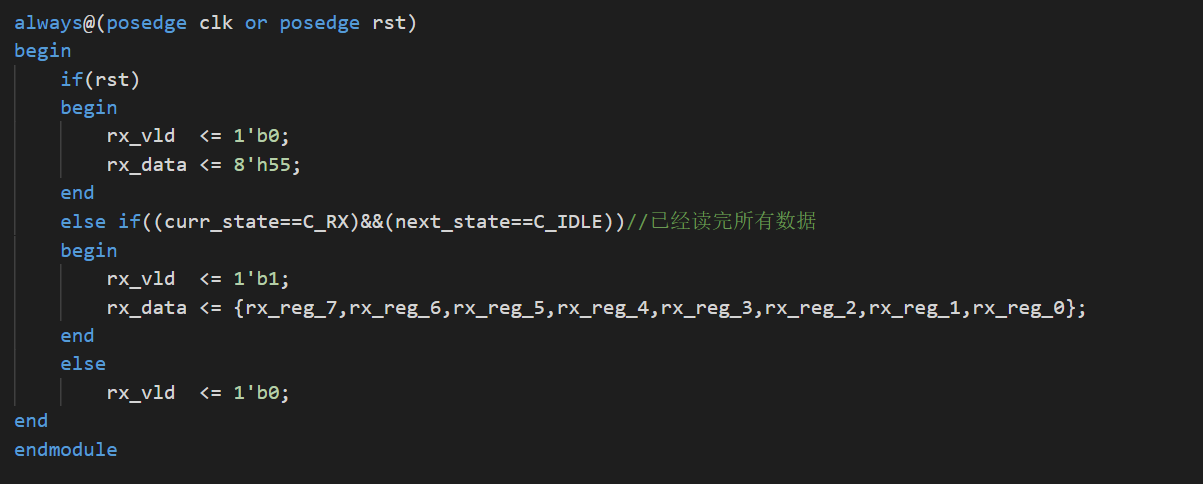


**rx.v**

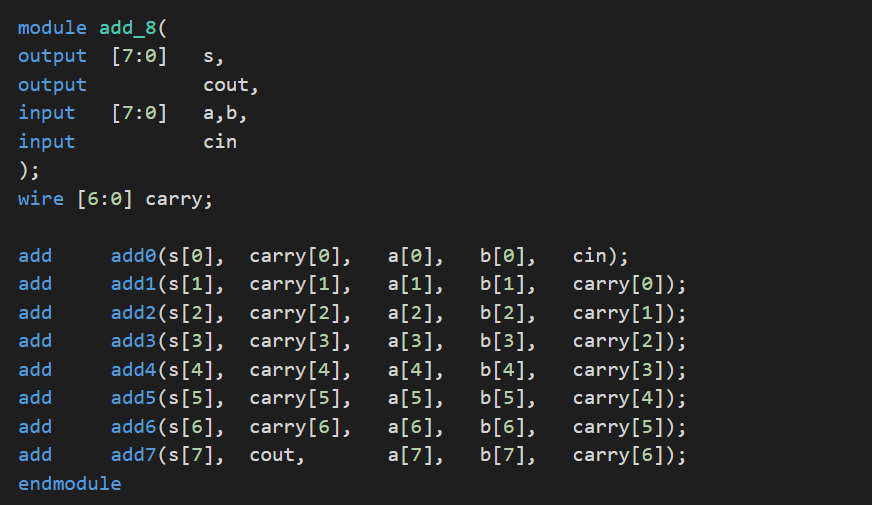




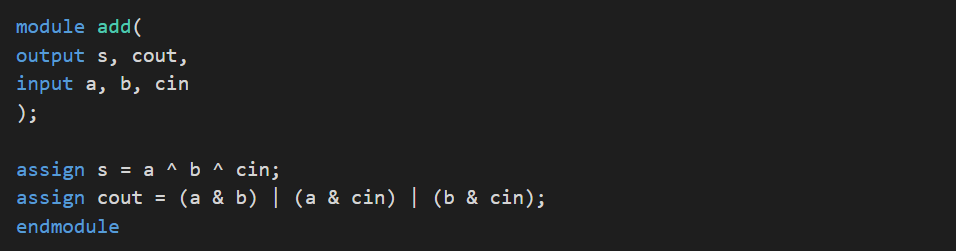




**add\_8.v**



**add.v**



**mul.v**

