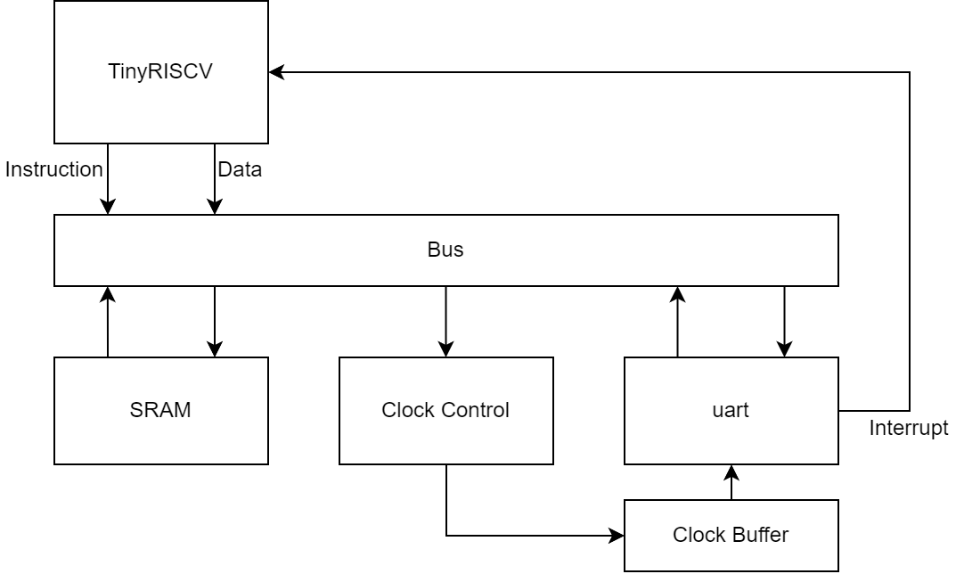
# 基于TinyRISCV的soc

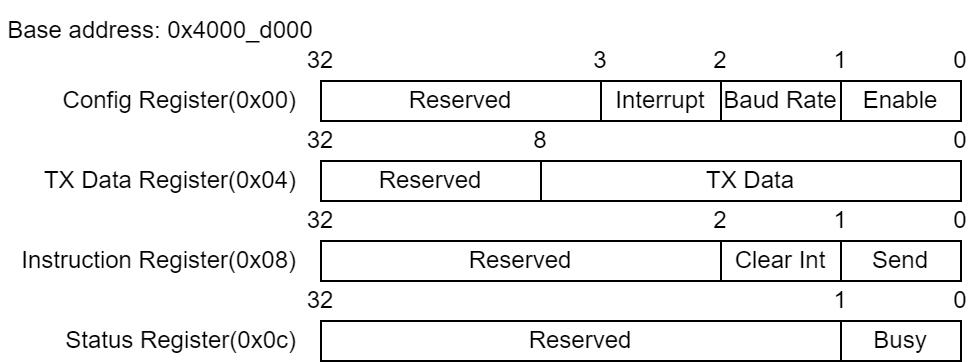
本项目使用开源的TinyRISCV核心设计附带ram，uart，clock/reset control的soc。并实现uart TX中断。



TinyRISCV核心和其附带的rib总线并没有地址和数据的寄存机制，且master并不会接收类似grant和ready等信号。当master发出请求后，读操作会再当前clock cycle直接获得数据。写操作会再下一个clock cycle写入对应的位置。并不适合使用AHB/APB总线。

核心含有2个需要接到ROM/RAM的接口，分别获取Instruction数据和普通RAM数据。这两个接口分别对应总线的Master1和Master0。其中Master0的优先级会高于Master1，并且再Master0获取到总线所有权后，会通过hold\_flag信号通知核心并进行stall处理。

## Uart



Instruction Register用于接收Master发来的指令。用于开始TX发送和清除Interrupt Flag。 当uart处于发送状态时Busy信号为高电平。

uart 由两个FSM进行控制。其中Uart\_state用于控制大的状态，transfer\_state用于控制发送。

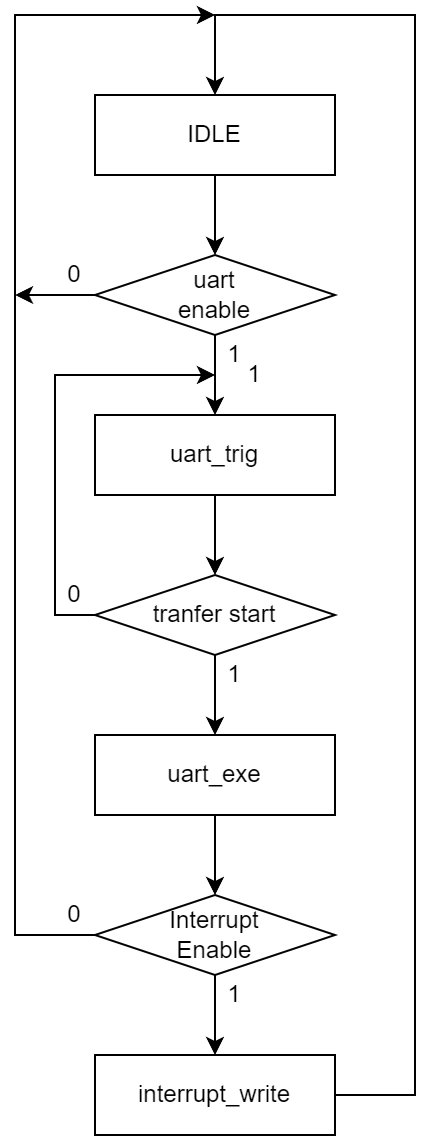
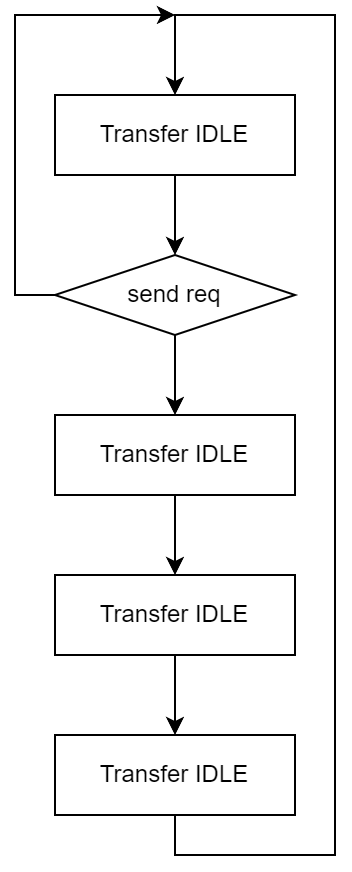
 

Figure: Uart状态机（左），发送状态机（右）

## 总线

TinyRISCV核心使用了设计者自行定义的rib总线。此总线相对于riscv常用的OBI和Arm常用的AHB总线，缺少了对时序的控制。对于OBI和AHB总线，当master发起request后，所有的read和write操作都应该在下一个clock cycle进行，或者根据slave的状态在更加靠后的clock cycle进行，当完成时slave应通过ready通知总线。而rib总线的read会在reqest发起的clock cycle进行，而write则更AHB和OBI相同。

同时rib总线并没有grant信号，无法对多个master进行精确的控制。在默认状态instruciton interface会一直占用总线，直到更高等级的data interface请求总线的使用，此时总线的所有权会直接交给data interface。由于没有grant信号来对每个master进行单独的暂停，其使用hold\_flag直接对instruction fetch进行stall来暂停instruction interface的读写。当核心的data接口当hold\_flag为1，核心的instruction fetch和decode进入stall状态，此时只有控制data interface的execution stage可以进行读写，用于防止总线的错误占用。

## 中断

TinyRISCV核心有8个中断接口。当接口的任意一位为1是，中断被触发。