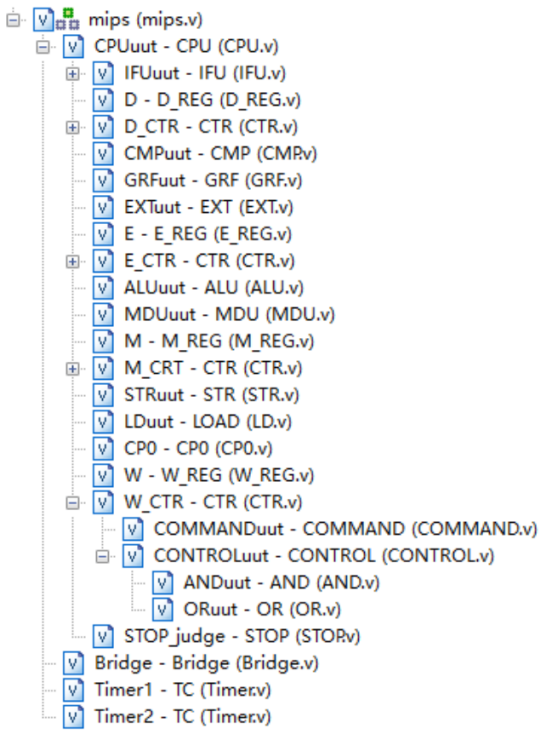
设计文档

设计草稿

ISA包括 add、addi、sub、and、andi、or、ori、slt、sltu、lw、lh、lb、sw、sh、sb、mult、multu、div、divu、mthi、mtlo、mfhi、mflo、beq、bne、lui、jal、jr、nop、syscall、mfc0、mtc0、eret

- 新加入了syscall、mfc0、mtc0、eret
- Verilog文件的结构如下：



新模块

CP0

端口说明

端口名称	方向	大小	说明
clk	input	[0:0]	时钟信号
res	input	[0:0]	同步复位
CP0_WE	input	[0:0]	写入SR或EPC的写使能
EXL_clr	input	[0:0]	SR的EXL置零
BD	input	[0:0]	当前指令是否处于延迟槽中
A1	input	[4:0]	读取的寄存器
A2	input	[4:0]	写入的寄存器
CP0_in	input	[31:0]	写入寄存器A1的值
EPC_in	input	[31:0]	输入的受害PC
exc_in	input	[4:0]	exception的种类
HWInt	input	[5:0]	外界向CPU输入的中断
CP0_out	output	[31:0]	读出寄存器A1的值
EPC_out	output	[31:0]	寄存器EPC的值
response	output	[0:0]	是否响应中断
Req	output	[0:0]	进入处理程序信号

内部逻辑

读出/写入寄存器

```
assign CP0_out = (A1 == 12) ? SR:
                 (A1 == 13) ? Cause:
                 (A1 == 14) ? EPC:
                 (A1 == 15) ? PrID:0;
```

```
if (CP0_WE == 1)begin
    if (A2 == 12) begin
        SR <= CP0_in;
        //$display("%d: Status <= %h", $time, CP0_in);//check
    end
    if (A2 == 14) begin
        EPC <= CP0_in;
        //$display("%d: EPC <= %h", $time, CP0_in);//check
    end
end
```

中断/异常的判断

```
assign exc = (|exc_in) & !`EXL;
assign Int = (|(HWInt & `IM)) & !`EXL & `IE;
assign Req = exc | Int;
```

中断/异常的处理

```
if (Req) begin // int|exc
            `ExcCode <= Int ? 5'b0 : exc_in;//Int comes first!!!
            `EXL <= 1'b1;
            `BD <= BD;
            EPC <= EPC_out;
        end
```

Bridge

端口说明

端口名称	方向	大小	说明
m_data_addr	output	[31:0]	DM_addr
m_data_wdata	output	[31:0]	DM_wdata
m_data_byteen	output	[3:0]	DM_byteen
m_data_rdata	input	[31:0]	DM_rdata
m_data_addr_tmp	input	[31:0]	CPU_addr
m_data_wdata_tmp	input	[31:0]	CPU_wdata
m_data_byteen_tmp	input	[3:0]	CPU_byteen
m_data_rdata_tmp	output	[31:0]	CPU_rdata
T1_addr	output	[31:0]	/
T1_in	output	[31:0]	/
T1_WE	output	[0:0]	/
T1_out	input	[31:0]	/
T2_addr	output	[31:0]	/
T2_in	output	[31:0]	/
T2_WE	output	[0:0]	/
T2_out	input	[31:0]	/

内部逻辑

选择rdata

```
assign m_data_rdata_tmp = ((m_data_addr_tmp >= 16'h7f00) & (m_data_addr_tmp <=
16'h7f0b)) ? T1_out :
                          ((m_data_addr_tmp >= 16'h7f10) & (m_data_addr_tmp <=
16'h7f1b)) ? T2_out :
                            m_data_rdata;
```

思考题

1. 请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的?

- 鼠标和键盘的输入信号都会转化为不同的系统中断信号。CPU根据中断信号的值可以执行对应的汇编指令，这样就达到了相应鼠标和键盘的目的。

2. 请思考为什么我们的 CPU 处理中断异常必须是已经指定好的地址？如果你的 CPU 支持用户自定义入口地址，即处理中断异常的程序由用户提供，其还能提供我们所希望的功能吗？如果可以，请说明这样可能会出现什么问题？否则举例说明。（假设用户提供的中断处理程序合法）

   - 个人认为可以实现，将CPU中Req高电平时要跳转到地址改为由用户输入的地址即可。

     ```
     assign F_NPC = (Req) ? PC_Req_input : ...
     ```

   - 这样的设计可能会导致意想不到的跳转结果（跳转到了正常程序内），需要加入一些跳转约定；且可能降低CPU的适用性

3. 为何与外设通信需要 Bridge?

   - 外设种类多样，加入Bridge可以让CPU在不需要了解外设种类的前提下根据预设的地址获取正确的数据；且其扩展性好，新加外设在系统桥进行添加即可，无需单独为其设计一套读写逻辑。

4. 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态移图。

   - 相同：都从初值寄存器中获取初数值；两种模式都受控制寄存器的控制

   - 不同：模式0在会一直提供中断信号，直至控制寄存器中的中断屏蔽位被设置为0；模式1只会提供一周期的中断信号，然后自动再次赋初值开始计数。

5. 倘若中断信号流入的时候，在检测宏观 PC 的一级如果是一条空泡（你的 CPU 该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在 P7 中，清空流水线产生的空泡指令应该保留原指令的哪些信息?

   - 会保存错误的EPC值

   - E级寄存器要保存**D_PC**、**D_BD**的值（由于宏观PC使用的是M_PC，因此不可保存E_PC、E_B [即保持PC、BD的值不变]，否则会出现在阻塞时外界给出中断信号后ERET跳回后重复执行某条指令的BUG！！！）

6. 为什么 `jalr` 指令为什么不能写成 `jalr $31, $31`?

   英文版的MIPS说明如下：

> Register specifiers rs and rd must not be equal, because such an instruction does not have the same effect when reex ecuted. The result of executing such an instruction is UNPREDICTABLE