设计方案

CP0 协处理器

端口定义

名称	类型	位宽	方向	含义
clk	wire	1	I	时钟信号
reset	wire	1	I	同步复位信号
enable	wire	1	I	CP0 中寄存器写使能
CP0_addr	wire	4:0	I	要写入的寄存器地址（12 或 14）
CP0_in	wire	31:0	I	CP0 写入数据
VPC	wire	31:0	I	受害 PC，陷入内核时传递给 CP0
BD_in	wire	1	I	是否为延迟槽指令
ExcCode_in	wire	4:0	I	内部异常代码
HWInt	wire	5:0	I	外部中断信号
EXL_clr	wire	1	I	SR_EXL 复位信号，返回现场
CP0_out	wire	31:0	O	CP0 读出数据
EPC_out	wire	31:0	O	EPC 读出数据
Req	wire	1	O	陷入内核请求

MFC0 与 MTC0

MFC0 和 MTC0 专为 CP0 内置的寄存器使用。MTC0 只允许写入 SR 和 EPC 两个寄存器，故其 rd 为 13 或 15 时保持原始值不变即可。

主模块中，要将 CP0 的输出 `CP0_output` 与 E 级来的 `E_exact_outputA`（ALU 或 MDU）合流，形成 `M_exact_outputA` 给到 W 级。

综上需要在 MController 传出 `CP0_enable` 和 `CP0_output_sel` 两个信号。

四个寄存器

SR

`SR[15:10]: SR_IM` 每一位表示每个外部中断的开关，该位为 1 时，允许处理这个外部中断。
`SR[1]: SR_EXL` 置为 0 时，允许处理外部中断和内部异常，否则一律拒绝。
`SR[0]: SR_IE` 置为 1 时，允许处理内部异常，否则一律拒绝。

注意 SR_EXL 和 SR_IE 值对应的效果相反

Cause

`Cause[31]: Cause_BD` 置为 1 时，表示陷入内核时的这条指令位于延迟槽中。
`Cause[15:10]: Cause_IP` 每一位表示一种外部设备，该位为 1 时，表示该设备触发了外部中断。
`Cause[6:2]: Cause_ExcCode` 内部异常的异常码。

EPC

记录陷入内核时的指令地址。当陷入内核时的指令是延迟槽指令时，则记录其上一条指令的地址，否则就是这一条的地址。

由于 CP0 在 M 级，则宏观 PC 位于 M 级，EPC 记录的也是 M 级的情况。

PRId

CPU 标识码，只读。

内部异常与外部中断

类型	ExcCode
外部中断 - [x]	0
PC 对齐错误 - [x]	4
PC 超出 `0x3000 ~ 0x6ffc` - [x]	4
`lw lh` 地址对齐错误 - [x]	4
`lh lb` 取 Timer 寄存器 - [x]	4
load 类计算地址时溢出 - [x]	4
load 类地址超出范围 - [x]	4
`sw sh` 地址对齐错误 - [x]	5

类型	ExcCode
`sh` `sb` 存 Timer 寄存器 - [x]	5
store 类计算地址时溢出 - [x]	5
store 类向计时器的 Count 寄存器存值 - [x]	5
store 类地址超出范围 - [x]	5
`syscall` - [x]	8
未知指令 - [x]	10
算术溢出 - [x]	12

F 级

PC 未对齐或超范围，异常码为 `5'b00100`，替换错误指令为 `nop`。

D 级

D Controller 识别指令种类时，如果检测到未知指令或者 `syscall`，输出对应的异常码，否则保持 `5'b00000`。

E 级

ALU 中可能发生的异常：取地址未对齐，地址计算溢出，地址计算超范围，算数运算溢出。运算溢出的判断方法采用指令表的。从 ALU 得到 E 级的 `ExcCode`，与前面合流得到最终的 `ExcCode` 流给 M 级 CP0。**合流时，倘若上一个流水级的 `ExcCode != 5'b00000`，则采用上一级的，因为需要先处理早发生的异常**。

M 级

将最终的 `ExcCode` 接到 CP0 中。

陷入内核

1. 首先，VPC 是宏观 PC。
2. 如果 D 级指令是跳转类，那么 F 级指令在延迟槽中。因此从 F 级开始判断 BD 值，并一直流水到 M 级传入 CP0。
3. 停止异常指令及其之后的所有指令的执行，并将指令地址跳转到 `0x00004180`：
    i. CP0 的 `Req` 信号控制 NPC 的计算，优先级最高。当 `Req` 为 1，NPC 结果为 `0x00004180`，并在下一周期给到 PC。

- 如果阻塞发生，同时需要跳转 `0x00004180`，则会因为阻塞导致 PC_en 关闭，无法把 `0x00004180` 写入 PC。因此需要调整 PC_en 在阻塞且没陷入内核的时候才能关闭。（陷入内核优先级最高）

ii. 清空所有流水级寄存器的数据，将 PC 换成 `0x00004180`，将 BD 换成 0。

iii. 阻止指令产生效果：Req 为 1 则关闭 DM 写使能，禁止 CP0 的寄存器写入。

返回现场

ERET

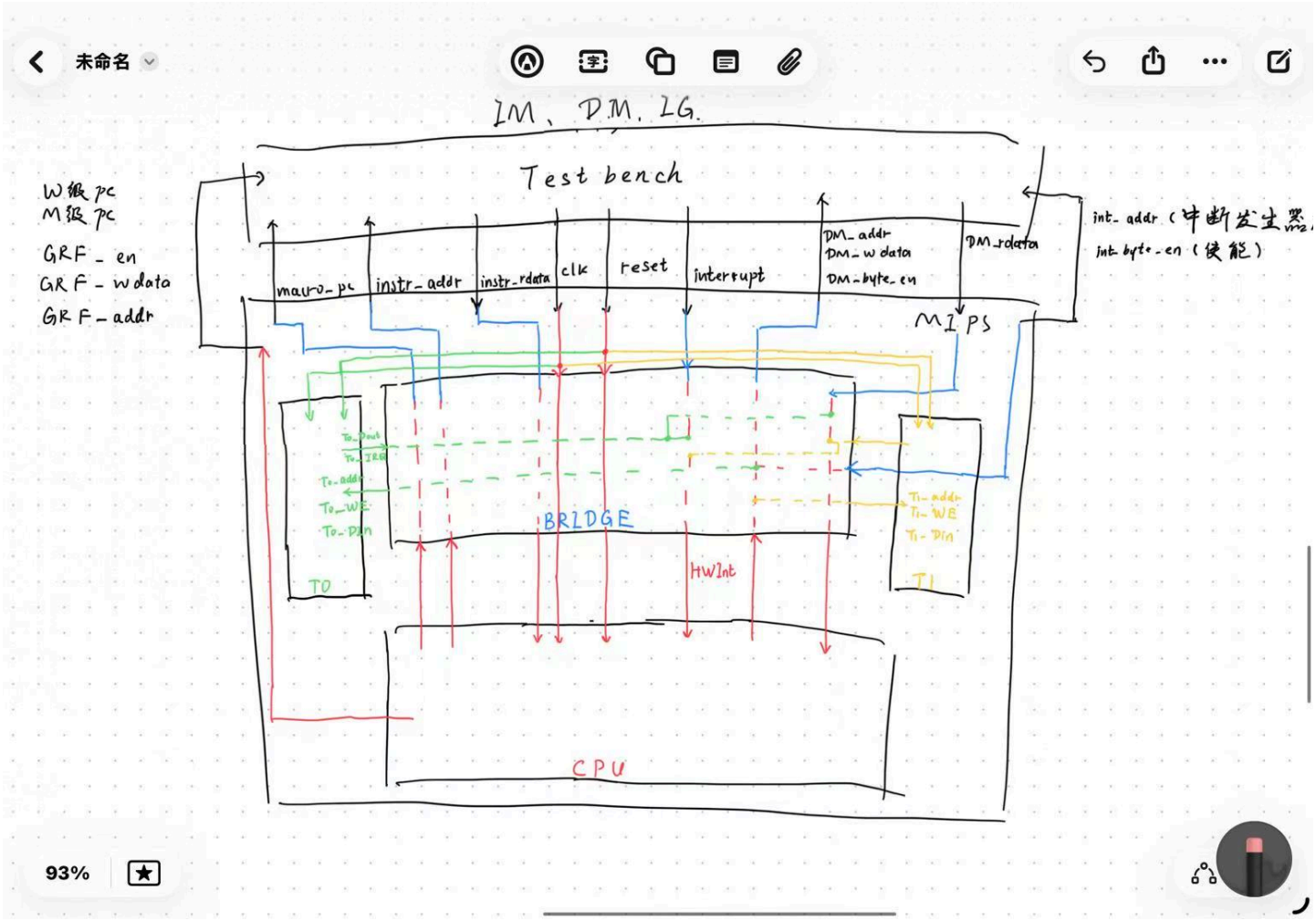
`eret` 用于从异常处理程序回到 EPC。这个跳转指令没有延迟槽，其行为模式类似于 `j`，实现方法如果 D 级是 `eret`，则在 F 级将 PC 替换成 EPC。**要注意这样必须设置 `eret` 的 NPC_sel 为顺序执行！这样就跳过了延迟槽。**

另外，`eret` 流到 M 级时，需要传给 CP0 的 EXLClr，表明异常处理完了，接下来可以处理新的异常。

再另外，如果 `eret` 前一个或两个周期是写入 EPC 寄存器的 `mtc0` 指令，那么 `eret` 无法在 F 级替换出正确的 PC 值，因此直接暴力阻塞即可。

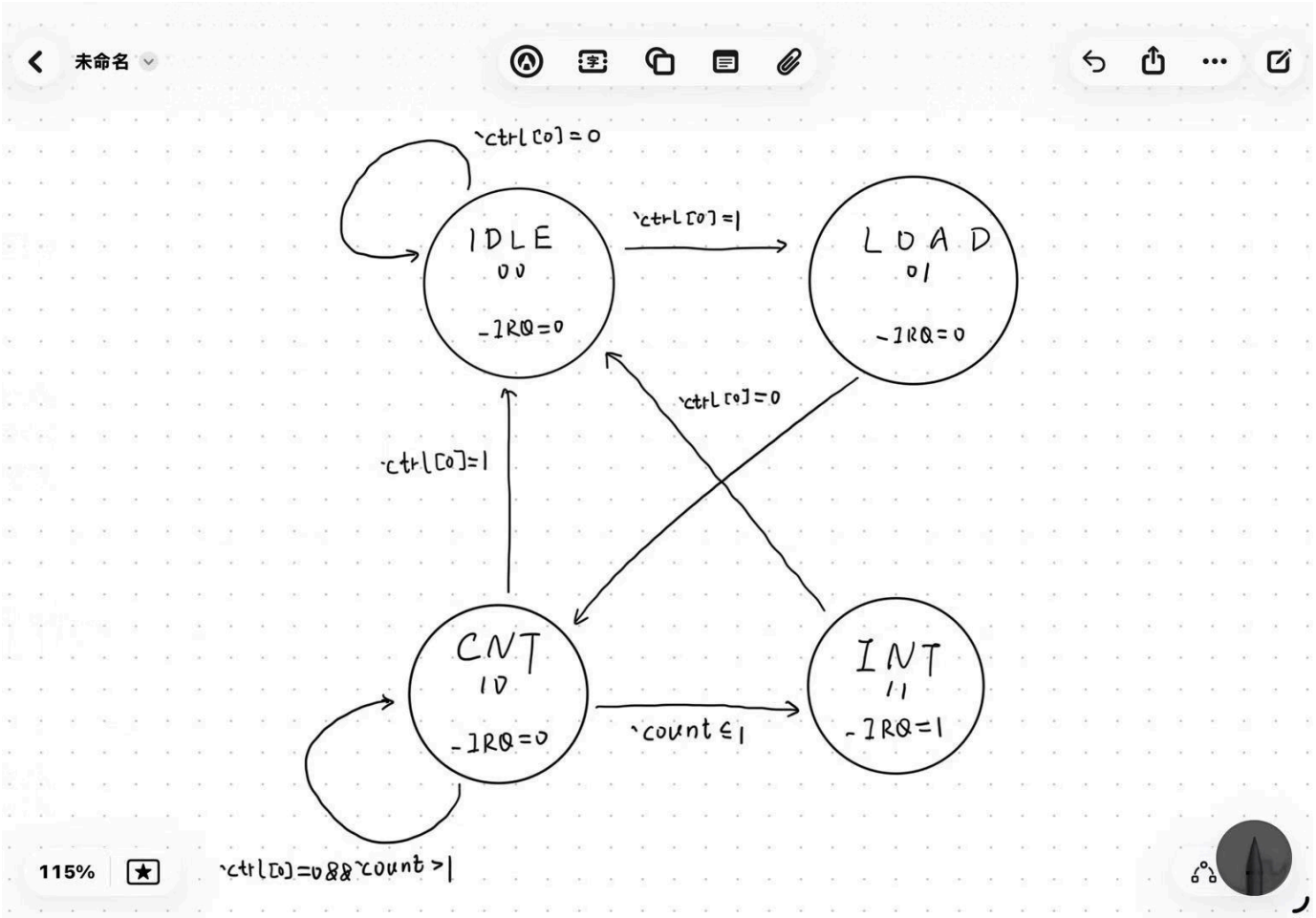
外部设备和顶层通信

顶层模块 MIPS 中会实例化三个模块，分别是 CPU，两个 Timer，以及系统桥 Bridge。需要在 MIPS 中定义好各个模块之间的连线，以及和 Testbench 的连线。

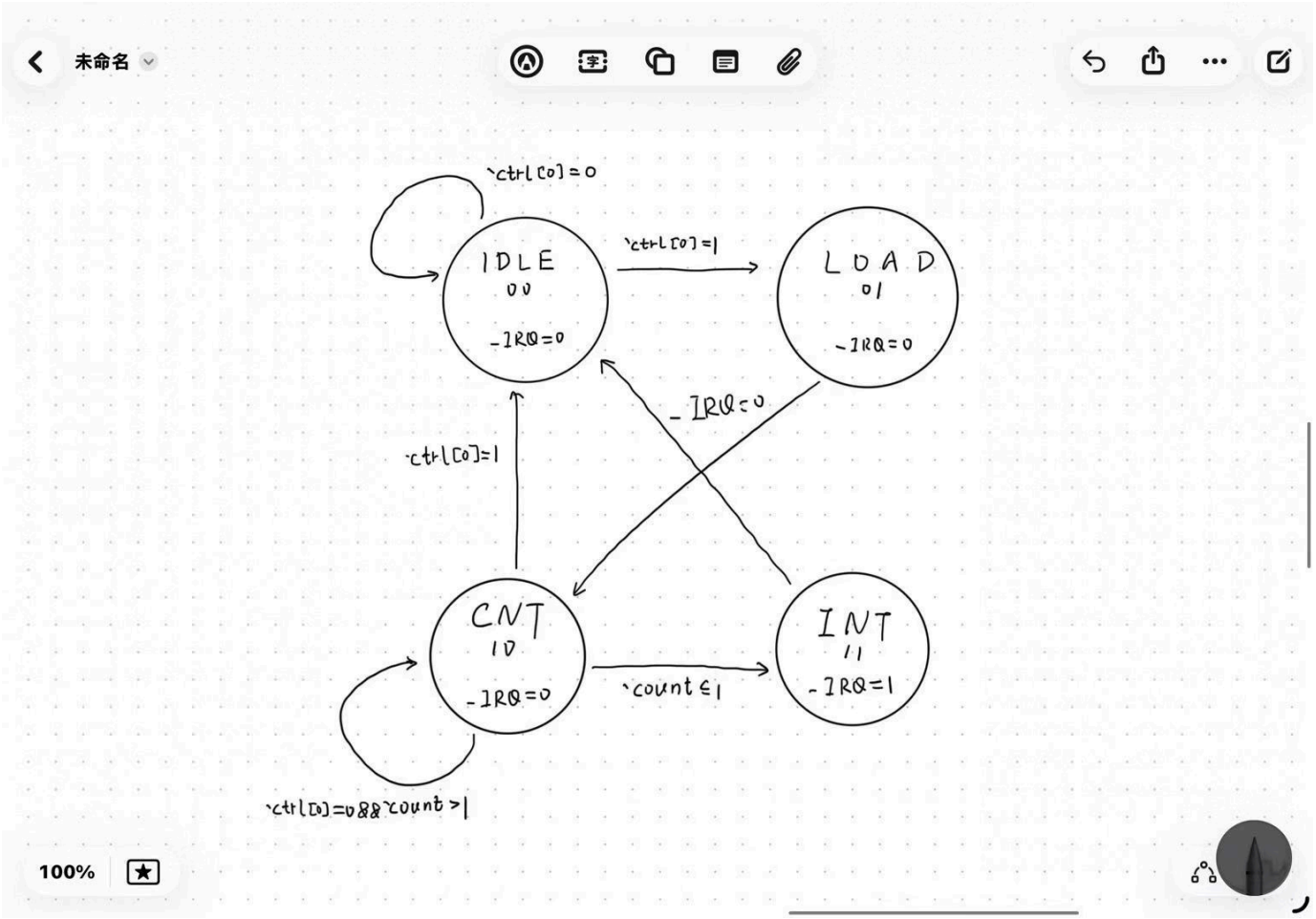


思考题

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？
   答： 键盘 → (中断请求信号 + 键盘对应的输入码<ASCII码>) → 南桥 → 北桥 → CPU处理器 → 内存 → 硬盘。
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）
   答： 首先一个固定的地址可以做到更快的访问，提高 CPU 处理中断异常的效率。并且不需要考虑用户自定义入口地址的有效性检查，减少成本。如果是用户自定义地址，也可以实现中断异常处理功能，但是有可能地址无效。
3. 为何与外设通信需要 Bridge？
   答： 外设种类很多，对于不同的中断，需要预处理相关的信号，再统一给到 CPU 进行处理。因此需要一个模块进行中转。
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。
   答：



`ctrl[2:1] == 2'b00` 时，倒计时结束后，产生持续的 `_IRQ` 信号。



`ctrl[2:1] == 2'b01` 时，倒计时结束后，将 `_IRQ` 归零，即只激活一周期的信号。

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？

   答： 此时往 EPC 保存的地址是 `0x00000000`， `BD` 为 0， 导致处理完异常返回原地址时找不到原来的指令。因此清空时应该保留 PC 和 BD。

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

   答： `jalr` 先将 PC + 8 写入 `$31` （转发），再跳转。故实际上无法成功跳转到 `$31` 原来的值。

测试

ERET 测试

```
ori $t0, 0x00003000
mtc0, $t0, $14
eret
```

中断异常测试

```
ori $1,$1,0x1c01
mtc0 $1,$12

# pc地址未对齐
ori $2,$2,0x300a
jr $2
add $2,$2,$2
# pc地址超范围
jr $2
ori $3,$3,0x0003

# lw、lh 没有字对齐
lw $2,0($3)
lh $2,1($0)
# lh、lb 取Timer寄存器的值
ori $4,$4,0x7f00
lh $5,0($4)
lb $5,20($4)
# 计算地址加法溢出
lui $6,65535
ori $6,$6,65535
lw $7,1($6)
# 取数地址超出范围
ori $7,0x7f0c
lw $7,0($7)

# sw、sh没有字对齐
sw $2,0($3)
sh $2,1($0)
# sh、sb取Timer寄存器的值
sw $5,0($4)#应该没错
sh $5,0($4)
sb $5,20($4)
# 计算地址加法溢出
lui $6,65535
ori $6,$6,65535
sw $7,1($6)
# 向计时器Count寄存器存值
sw $7,-4($7)
# 存数地址超出范围
sw $7,100($7)

# syscall 测试
```

```
syscall

# 未知指令
nor $2,$3,$4

# 算术溢出
addi $1,$0,1
sub $8,$0,$1
add $9,$8,$6 # 不应溢出
sub $9,$6,$8 # 溢出
add $9,$6,$7 # 溢出
sub $9,$0,$6
addi $9,$9,-100 # 溢出


end:
beq $0,$0,end # 死循环
nop

# 异常处理程序
.ktext 0x4180
mfc0 $k0,$12
mfc0 $k0,$13
mfc0 $k0,$14
addi $k0,$k0,4
mtc0 $k0,$14
eret
add $2,$2,$2 # 应当没有延迟槽
```