一，D 触发器/寄存器

1. 门控时钟

作者在手册中说只要按照标准代码风格编写寄存器，综合工具就会自动为寄存器添加门控时钟单元，但是作者并未提及标准代码风格是什么样的，门控时钟的具体实现又是怎样的。

a. 标准代码风格

要点是明确写入使能信号，有且仅当写入使能置为高电平或者置为低电平时，且 always 内条件满足时，寄存器才可改变。一个典型的错误是，写入使能置为高电平和低电平时，寄存器都会改变，高电平时写入，低电平时清零。正确的做法是要么低电平有效要么高电平有效，但是一般习惯是高电平有效。因此我的理解是只要明确写入有效信号，则可自动添加门控时钟。

b. 门控时钟的实现

通过查阅资料，门控时钟一般有两种实现方法。第一种是使用锁存器，实现电路如图 1 所示。但该方法有两个不足之处，一是如果在电路中，除法器与与门相隔很远，到达锁存器的时钟与到达与门的时钟有较大的延迟差别，则仍会出现毛刺。二是如果在电路中，时钟使能信号距离锁存器很近，可能会不满足锁存器的建立时间，会造成锁存器输出出现亚稳态。但是一般会把门控时钟和寄存器制作为标准单元，提前设置好相对位置，就一般不会出现上述两点问题，verilog 综合器好像也采用的这种方法，它相对于第二种使用寄存器的方法能节省不少面积。

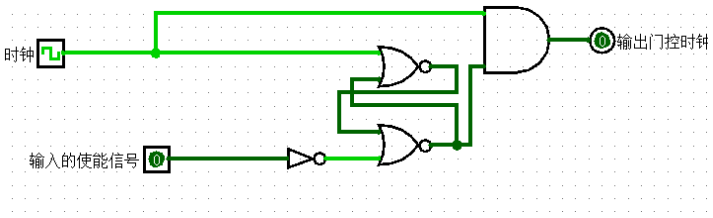

图 1 中间的两个 nor 和非门组成了 D 锁存器

第二种是使用 D 除法器，实现电路如图 2 所示，该方法很难出现毛刺，但是占用面积大。

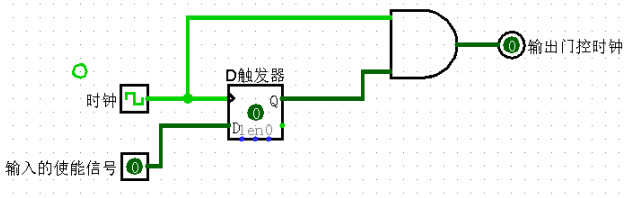

图 2

如果简单的将 CLK 和使能信号与上组成门控时钟是不可行的，会产生毛刺。

2. 异步复位同步释放

E203 中整体使用的是异步复位，但是在 PC 生成单元中却发现了一个例外，大致过程是一个 D 触发器在复位信号到来时置为 1，然后将该 D 触发器的内容在下一个 clk 到来后写入另外一个寄存器，之后再用该寄存器的值做为判断依据，选择用来更新 PC 的值，使得 PC 寄存器复位为 top 指定的复位 PC。作

者未对此进行任何说明，对我产生了不少困惑，但是看了异步复位同步释放后感觉两者很是相似。异步复位信号首先维持的时间不够长，无法做为 PC 的选择信号，可能 PC 还未生成，该信号就降为低电平了，其次异步复位信号的到来时机无法确定，可能在 PC 生成之后，再其次异步复位信号在时钟边沿附近容易产生亚稳态，导致 PC 更新出错。而异步复位同步释放刚好解决了以上问题，若异步复位产生亚稳态也只会影响第一个 D 触发器，到第二个 D 触发器时已经是下一个 clk 了，并且复位信号也会锁存在第二个触发器中知道 PC 更新完毕。

在此过程中作者的设计也很巧妙，使用的全是例化的 DFF，没有使用自己定义的，减少的测试的负担。第一个 DFF 使用的是不带写入使能且复位默认值为 1 的 DFF，将输入端口接常数 0，只要复位信号到来，该 DFF 即输出 1，下一个 clk 到来，因为无输入使能且输入端口为常数 0，因此该 DFF 会自动清零。该 DFF 后接的 DFF 使用的是作者常用于存储控制信号的 DFF 的编码风格，即拥有 set、clr、ena、nxt 和 r 端口，并且每次复位信号到来都会清零，这样相当于即使之前有 reset 未执行完也会被完全冲刷掉。

3. 使用的小技巧

作者作品的代码中用作控制的寄存器均遵循一个固定的风格，我觉得很值得学习，但是作者在手册中未提及，因为 E203 中用作控制的寄存器均是带写入使能和异步复位的寄存器，因此以下将以其为例子展开。按照作者的风格，寄存器以及控制信号应按如下的样子定义：

```
wire xxx_r;//寄存器当前存储的值
wire xxx_set = (置位控制信号，将置位判断逻辑写在这里)
wire xxx_clr = (清零控制信号，将清零判断逻辑写在这里)
wire xxx_ena = xxx _set | xxx _clr;//写入使能信号
wire xxx_nxt = xxx_set | (~xxx _clr);//即将写入至寄存器内
sirv_gnrl_dfflr #(1) xxx _dfflr ( xxx _ena , xxx _nxt , xxx _r , clk , rst_n );
```

置位信号和清零信号均会写使能，这样就无需定义的新的带清零信号的寄存器，可以与例化寄存器保持一致，另外也不会导致寄存器写使能模糊不清，使得综合工具无法自动添加门控时钟。若不涉及控制的寄存器，一般选用不带清零信号的寄存器，可以无谓的信号跳变，以减少功耗。

二，通信协议

作者在 E203 中一共用了两种通信协议，第一种是 ICB 协议，作者在手册中有详细说明，第二种是一种半互锁的握手方式。

1. 我对 ICB 的理解

我认为 ICB 实质上是一种带有两个半互锁通信通道的协议，用于两个需要相互交互的模块，比如 A 模块需要向 B 模块传输数据，并且这些数据还需要 B 模块处理并且向 A 模块反馈错误码和相关数据，总之是需要保持 A 向 B 发送数据、B 处理数据和 B 向 A 反馈数据三个操作顺序执行，常见于访存模块与其他模块之间的通信。

2. 半互锁的握手方式

这种半互锁的握手方式作者并未做任何说明，但是却是控制芯片时序的主要协议，icb 也是基于此产生的。icb 是控制一种交互式的通讯，而半互锁协议则是单向推进的协议。

3. 以 load 指令在 ex 级别的执行为例，具体说明两种协议

以 load 指令为例，其执行顺序为：

- if 取出指令后通过半互锁的方式与 disp 单元通讯
- if 与 disp 通讯的同时，相关数据也被传输到了 decoder 单元，解码后之间被接入至 disp 单元中，因为 decoder 是纯组合逻辑，因此无需通讯协议
- disp 通过半互锁的方式与 alu 总控通讯
- alu 总控通过半互锁的方式与 lsuagu 通讯
- alu 总控与 lsuagu 通讯的同时，lsuagu 还向 alu 共享数据通路生成加法操请求，执行加法操作，由于 alu 数据通路属于组合逻辑，因此也无需通过协议通讯
- 运算完成后，lsuagu 通过 icb 协议与 lsu_ctrl 单元通讯，cmd 通道送去访存地址
- 当 lsuagu 与 lsu_ctrl 的 cmd 通道握手后，lsu_ctrl 也通过 icb 协议与 dtcm_ctrl 通讯，请求读取数据，cmd 通道送去地址，rsp 通道返回读取的数据
- lsu_ctrl 读取到数据后，即通过与 lsuagu 的 rsp 通道返回数据
- lsuagu 与 lsu_ctrl 的 rsp 通道握手成功后，下个 clk 通过半互锁协议与 longpwbck 单元通讯，传输待写回的数据
- longpwbck 与 lsuage 握手成功后，通过半互锁的通讯协议与 wbck 通讯，传输待写回的数据

从上面可以看到一个非常明显的时序关系，若不需要后一个模块向前一个模块返回数据，则使用半互锁协议，需要返回数据则使用 icb 协议。

三，信号

1. 更节省门的书写方式

```
assign   A = B & C & (D?E:F);
assign   G = B & C & (H?I :J);
```

以上信号应按如下写法修改，才能最大限度的节省门

```
assign mid= B & C;
assign   A = mid & (D?E:F);
assign   G = mid & (H?I :J);
```

与第一种写法比较可以节省一个门，在芯片的设计中该技巧非常常用，尤其是解码器部分。

2. 门控信号

通过在某个信号前与上一个控制信号，那么该控制信号就可以控制此信号的改变，当控制信号为 1 时该信号才可改变，达到必要时才改变的效果，以减少信号的翻转，从而减少功耗。以公共数据通路中加减法通路为例，信号 adder_add 和 adder_sub 可表示是否进行加减法操作，也就是该信号不置位则不需要加减法通路，因此可在加减法通路的输入前与上该信号，如

```
adder_addsub = adder_add | adder_sub;
adder_in1 = {`E203_ALU_ADDER_WIDTH{adder_addsub}} & (adder_op1);
adder_in2   =   {`E203_ALU_ADDER_WIDTH{adder_addsub}}   &   (adder_sub   ?
(~adder_op2) : adder_op2);
```

可控制加减法通路的开关，在不用时则关闭，减少信号翻转，以面积(增加门)换取功耗的减少。但是有时候是不值当的，比如简单的逻辑与、或和非操作，功耗不高，用门控信号则有些不值。