电子科大 Tony 工作总结

作者：Tony　整理：Spike@BeLLsTuDio

序

很早之前就想对这几个月工作经历写的东西，一是作为自己的总结，二是自己也很想将自己这段时间的一些经历和大家分享一下，希望对初学者而言能使得他们能少走一些弯路。只是公司里的事情很多，最近经常加班，所以一直拖到现在。

能来到这家公司应该是一种缘份--缘起 NIOS。当初三月份 altera 来我们学校建立 SOPC 实验室的时候自己还不知道 NIOS 是什么东西，只是想在 altera 的 FAE 讲完 NIOS 后多问他几个时序约束的问题，然后拷一份 PPT 回去。但是想不到因为那一份 NIOS 的培训资料，我认识了 edacn 上的 cawan，他给我讲了很多 NIOS 的东西，之后是丁哥在 SOC 版帖了位 NIOS 大赛的通知，然后我和队友就去报了名，并去川大参加了 NIOS 的培训，认识了峻龙的 FAE----也是我现在的 boss。在这里要谢谢 cawan、丁哥、和我一起参加 NIOS 竞赛的队友刘科以及我的 BOSS，是他们让我有了这一段的经历。

在公司里的几个月，做的项目其实不多，但是收获还是有一些，我觉得收获最大的是设计理念的改变，这也是我这段时间最想总结的，我会在后面逐渐阐述。

时序是设计出来的

我的 boss 有在华为及峻龙工作的背景，自然就给我们讲了一些华为及 altera 做逻辑的一些东西，而我们的项目规范，也基本上是按华为的那一套去做。在工作这几个月中，给我感触最深的是华为的那句话：时序是设计出来的，不是仿出来的，更不是凑出来的。

在我们公司，每一个项目都有很严格的评审，只有评审通过了，才能做下一步的工作。以做逻辑为例，并不是一上来就开始写代码，而是要先写总体设计方案和逻辑详细设计方案，要等这些方案评审通过，认为可行了，才能进行编码，一般来说这部分工作所占的时间要远大于编码的时间。

总体方案主要是涉及模块划分，一级模块和二级模块的接口信号和时序（我们要求把接口信号的时序波形描述出来）以及将来如何测试设计。在这一级方案中，要保证在今后的设

计中时序要收敛到一级模块(最后是在二级模块中)。什么意思呢？我们在做详细设计的时候，对于一些信号的时序肯定会做一些调整的,但是这种时序的调整最多只能波及到本一级模块，而不能影响到整个设计。记得以前在学校做设计的时候，由于不懂得设计时序，经常因为有一处信号的时序不满足，结果不得不将其它模块信号的时序也改一下，搞得人很郁闷。

在逻辑详细设计方案这一级的时候，我们已经将各级模块的接口时序都设计出来了，各级模块内部是怎么实现的也基本上确定下来了。

由于做到这一点，在编码的时候自然就很快了，最重要的是这样做后可以让设计会一直处于可控的状态，不会因为某一处的错误引起整个设计从头进行。

做逻辑的难点在于系统结构设计和仿真验证

刚去公司的时候 BOSS 就和我讲，做逻辑的难点不在于 RTL 级代码的设计，而在于系统结构设计和仿真验证方面。目前国内对可综合的设计强调的比较多，而对系统结构设计和仿真验证方面似乎还没有什么资料，这或许也从一个侧面反映了国内目前的设计水平还比较低下吧。

以前在学校的时候，总是觉得将 RTL 级代码做好就行了，仿真验证只是形式而已，所以对 HDL 的行为描述方面的语法不屑一顾，对 testbench 也一直不愿意去学--因为觉得画波形图方便；对于系统结构设计更是一点都不懂了。

到了公司接触了些东西才发现完全不是这样。

其实在国外，花在仿真验证上的时间和人力大概是花在 RTL 级代码上的两倍，现在仿真验证才是百万门级芯片设计的关键路径。仿真验证的难点主要在于怎么建模才能完全和准确地去验证设计的正确性（主要是提高代码覆盖），在这过程中，验证速度也是很重要的。

验证说白了也就是怎么产生足够覆盖率的激励源，然后怎么去检测错误。我个人认为，在仿真验证中，最基本就是要做到验证的自动化。这也是为什么我们要写 testbench 的原因。在我现在的一个设计中，每次跑仿真都要一个小时左右（这其实算小设计）。由于画波形图无法做到验证自动化，如果用通过画波形图来仿真的话，一是画波形会画死（特别是对于算法复杂的、输入呈统计分布的设计），二是看波形图要看死，三是检错率几乎为零。

那么怎么做到自动化呢？我个人的水平还很有限，只能简单地谈下 BFM（bus function model，总线功能模型）。

以做一个 MAC 的 core 为例（背板是 PCI 总线），那么我们需要一个 MAC_BFM 和 PCI_BFM 及 PCI_BM（PCI behavior model）。MAC_BFM 的主要功能是产生以太网帧(激励源)，随机的长

度和帧头，内容也是随机的,在发送的同时也将其复制一份到 PCI_BM 中；PCI_BFM 的功能则是仿 PCI 总线的行为，比如被测收到了一个正确帧后会向 PCI 总线发送一个请求，PCI_BFM 则会去响应它,并将数据收进来;PCI_BM 的主要功能是将 MAC_BFM 发送出来的东西与 PCI_BFM 接收到的东西做比较，由于它具有了 MAC_BFM 的发送信息和 PCI_BFM 的接收信息，只要设计合理，它总是可以自动地、完全地去测试被测是否工作正常，从而实现自动检测。

华为在仿真验证方面估计在国内来说是做的比较好的，他们已建立起了比较好的验证平台，大部分与通信有关的 BFM 都做好了，听我朋友说，现在他们只需要将被测放在测试平台中，并配置好参数，就可以自动地检测被测功能的正确与否。

在功能仿真做完后，由于我们做在是 FPGA 的设计，在设计时已经基本保证 RTL 级代码在综合结果和功能仿真结果的一致性，只要综合布局布线后的静态时序报告没有违反时序约束的警告，就可以下到板子上去调试了。事实上，在华为中兴，他们做 FPGA 的设计时也是不做时序仿真的，因为做时序仿真很花时间，且效果也不见得比看静态时序分析报告好。

当然了，如果是 ASIC 的设计话，它们的仿真验证的工作量要大一些，在涉及到多时钟域的设计时，一般还是做后仿的。不过在做后仿之前，也一般会先用形式验证工具和通过静态时序分序报告去查看有没有违反设计要求的地方，这样做了之后，后仿的工作量可以小很多。

在 HDL 语言方面，国内语言很多人都在争论 VHDL 和 verilog 哪个好，其实我个人认为这并没有多大的意义，外面的大公司基本上都是用 verilog 在做 RTL 级的代码，所以还是建议大家尽量学 verilog。在仿真方面，由于 VHDL 在行为级建模方面弱于 verilog，用 VHDL 做仿真模型的很少，当然也不是说 verilog 就好，其实 verilog 在复杂的行为级建模方面的能力也是有限的，比如目前它还不支持数组。在一些复杂的算法设计中，需要高级语言做抽象才能描述出行为级模型。在国外，仿真建模很多都是用 System C 和 E 语言，用 verilog 的都算是很落后的了，国内华为的验证平台好像是用 System C 写。

在系统结构设计方面，由于我做的设计还不够大，还谈不上什么经验，只是觉得必须要具备一些计算机系统结构的知识才行。划分的首要依据是功能，之后是选择合适的总线结构、存储结构和处理器架构，通过系统结构划分要使各部分功能模块清晰，易于实现。这一部分我想过段时间有一点体会了再和大家分享，就先不误导大家了。

规范很重要

工作过的朋友肯定知道，公司里是很强调规范的，特别是对于大的设计（无论软件还是硬件），不按照规范走几乎是不可实现的。逻辑设计也是这样：如果不按规范做的话，过一个

月后调试时发现有错，回头再看自己写的代码，估计很多信号功能都忘了，更不要说检错了；如果一个项目做了一半一个人走了，接班的估计得从头开始设计；如果需要在原来的版本基础上增加新功能，很可能也得从头来过，很难做到设计的可重用性。

在逻辑方面，我觉得比较重要的规范有这些：

1.设计必须文档化。

要将设计思路，详细实现等写入文档，然后经过严格评审通过后才能进行下一步的工作。这样做乍看起来很花时间，但是从整个项目过程来看，绝对要比一上来就写代码要节约时间，且这种做法可以使项目处于可控、可实现的状态。

2.代码规范。

a.设计要参数化。比如一开始的设计时钟周期是 30ns，复位周期是 5 个时钟周期，我们可以这么写：

```
parameter  CLK_PERIOD = 30;
parameter  RST_MUL_TIME = 5;
parameter  RST_TIME = RST_MUL_TIME * CLK_PERIOD;
...
rst_n = 1'b0;
# RST_TIME rst_n = 1'b1;
...
# CLK_PERIOD/2 clk <= ~clk;
```

如果在另一个设计中的时钟是 40ns，复位周期不变，我们只需对 CLK_PERIOD 进行重新例化就行了，从而使得代码更加易于重用。

b.信号命名要规范化。

1) 信号名一律小写，参数用大写。

2) 对于低电平有效的信号结尾要用_n 标记，如 rst_n。

3) 端口信号排列要统一，一个信号只占一行，最好按输入输出及从哪个模块来到哪个模块去的关系排列，这样在后期仿真验证找错时后　方便很多。如：

```
module a(
        //input
        clk,
        rst_n,    //globle signal
```

```
    wren,
    rden,
    avalon_din,  //related to avalon bus
    sdi,         //related to serial port input
    //output
    data_ready,
    avalon_dout, //related to avalon bus
    ...
   );
```

4) 一个模块尽量只用一个时钟，这里的一个模块是指一个 module 或者是一个 entity。在多时钟域的设计中涉及到跨时钟域的设计中最好有专门一个模块做时钟域的隔离。这样做可以让综合器综合出更优的结果。

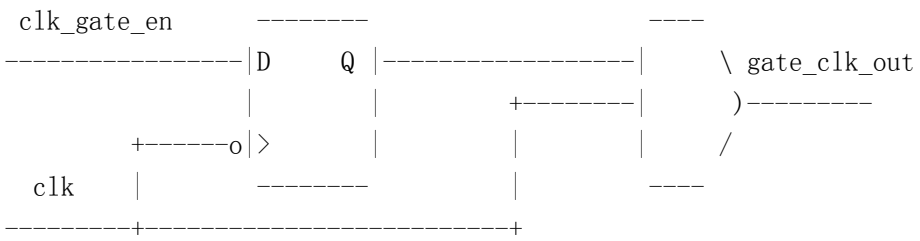
5) 尽量在底层模块上做逻辑，在高层尽量做例化，顶层模块只能做例化，禁止出现任何胶连逻辑（glue logic），哪怕仅仅是对某个信号取反。理由同上。

6) 在 FPGA 的设计上禁止用纯组合逻辑产生 latch，带 D 触发器的 latch 的是允许的，比如配置寄存器就是这种类型。

7) 一般来说，进入 FPGA 的信号必须先同步，以提高系统工作频率（板级）。

8) 所有模块的输出都要寄存器化，以提高工作频率，这对设计做到时序收敛也是极有好处的。

9) 除非是低功耗设计，不然不要用门控时钟--这会增加设计的不稳定性，在要用到门控时钟的地方，也要将门控信号用时钟的下降沿 打一拍再输出与时钟相与。



10)禁止用计数器分频后的信号做其它模块的时钟，而要用改成时钟使能的方式，否则这种时钟满天飞的方式对设计的可靠性极为不利，也大大增加了静态时序分析的复杂性。如 FPGA 的输入时钟是 25M 的，现在系统内部要通过 RS232 与 PC 通信，要以 rs232_1xclk 的速率发送数据。

不要这样做：

```
always (posedge rs232_1xclk or negedge rst_n)
begin
    ...
end
```

而要这样做：

```
always (posedge clk_25m or negedge rst_n)
begin
    ...
    else if ( rs232_1xclk == 1'b1 )
    ...
end
```

11)状态机要写成 3 段式的(这是最标准的写法)，即

```
...
always @(posedge clk or negedge rst_n)
...
    current_state <= next_state;
...
always @ (current_state ...)
...
case(current_state)
      ...
      s1:
         if ...
           next_state = s2;
      ...
...
always @(posedge clk or negedge rst_n)
...
    else
       a <= 1'b0;
```

```
c <= 1'b0;
c <= 1'b0;              //赋默认值
 case(current_state)
     s1:
         a <= 1'b0;   //由于上面赋了默认值，这里就不用再对 b、c
赋值了
     s2:
         b <= 1'b1;
     s3:
         c <= 1'b1;
     default:
     ...
...
```

3.ALTERA 参考设计准则

1) Ensure Clock, Preset, and Clear configurations are free of glitches.

2) Never use Clocks consisting of more than one level of combinatorial logic.

3) Carefully calculate setup times and hold times for multi-Clock systems.

4) Synchronize signals between flipflops in multi-Clock systems whenthe setup and hold time requirements cannot be met.

5) Ensure that Preset and Clear signals do not contain race conditions.

6) Ensure that no other internal race conditions exist.

7) Register all glitch-sensitive outputs.

8) Synchronize all asynchronous inputs.

9) Never rely on delay chains for pin-to-pin or internal delays.

10)Do not rely on Power-On Reset. Use a master Reset pin to clear all flipflops.

11)Remove any stuck states from state machines or synchronous logic.

其它方面的规范一时没有想到，想到了再写，也欢迎大家补充。

如何提高电路工作频率

对于设计者来说，我们当然希望我们设计的电路的工作频率（在这里如无特别说明，工作频率指 FPGA 片内的工作频率）尽量高。我们也经常听说用资源换速度，用流水的方式可以提高工作频率，这确实是一个很重要的方法，今天我想进一步去分析该如何提高电路的工作频率。

我们先来分析下是什么影响了电路的工作频率。

我们电路的工作频率主要与寄存器到寄存器之间的信号传播时延及 clock skew 有关。在 FPGA 内部如果时钟走长线的话，clock skew 很小，基本上可以忽略，在这里为了简单起见，我们只考虑信号的传播时延的因素。

信号的传播时延包括寄存器的开关时延、走线时延、经过组合逻辑的时延（这样划分或许不是很准确，不过对分析问题来说应该是没有可以的），要提高电路的工作频率，我们就要在这三个时延中做文章，使其尽可能的小。

我们先来看开关时延，这个时延是由器件物理特性决定的，我们没有办法去改变，所以我们只能通过改变走线方式和减少组合逻辑的方法来提高工作频率。

1.通过改变走线的方式减少时延。

以 altera 的器件为例，我们在 quartus 里面的 timing closure floorplan 可以看到有很多条条块块，我们可以将条条块块按行和按列分，每一个条块代表 1 个 LAB，每个 LAB 里有 8 个或者是 10 个 LE。它们的走线时延的关系如下：同一个 LAB 中（最快）< 同列或者同行 < 不同行且不同列。

我们通过给综合器加适当的约束（不可贪心，一般以加 5%裕量较为合适，比如电路工作在 100Mhz，则加约束加到 105Mhz 就可以了，贪心效果反而不好，且极大增加综合时间）可以将相关的逻辑在布线时尽量布的靠近一点，从而减少走线的时延。（注：约束的实现不完全是通过改进布局布线方式去提高工作频率，还有其它的改进措施）

2.通过减少组合逻辑的减少时延。

上面我们讲了可以通过加约束来提高工作频率，但是我们在做设计之初可万万不可将提高工作频率的美好愿望寄托在加约束上，我们要通过合理的设计去避免出现大的组合逻辑，从而提高电路的工作频率，这才能增强设计的可移植性，才可以使得我们的设计在移植到另一同等速度级别的芯片时还能使用。

我们知道，目前大部分 FPGA 都基于 4 输入 LUT 的，如果一个输出对应的判断条件大于四输入的话就要由多个 LUT 级联才能完成，这样就引入一级组合逻辑时延，我们要减少组合逻

辑，无非就是要输入条件尽可能的少，，这样就可以级联的 LUT 更少，从而减少了组合逻辑引起的时延。

我们平时听说的流水就是一种通过切割大的组合逻辑(在其中插入一级或多级 D 触发器，从而使寄存器与寄存器之间的组合逻辑减少）来提高工作频率的方法。比如一个 32 位的计数器，该计数器的进位链很长，必然会降低工作频率，我们可以将其分割成 4 位和 8 位的计数，每当 4 位的计数器计到 15 后触发一次 8 位的计数器，这样就实现了计数器的切割，也提高了工作频率。

在状态机中，一般也要将大的计数器移到状态机外，因为计数器这东西一般是经常是大于 4 输入的，如果再和其它条件一起做为状态的跳变判据的话，必然会增加 LUT 的级联，从而增大组合逻辑。以一个 6 输入的计数器为例，我们原希望当计数器计到 111100 后状态跳变，现在我们将计数器放到状态机外，当计数器计到 111011 后产生个 enable 信号去触发状态跳变，这样就将组合逻辑减少了。

上面说的都是可以通过流水的方式切割组合逻辑的情况，但是有些情况下我们是很难去切割组合逻辑的，在这些情况下我们又该怎么做呢？

状态机就是这么一个例子，我们不能通过往状态译码组合逻辑中加入流水。如果我们的设计中有一个几十个状态的状态机，它的状态译码逻辑将非常之巨大，毫无疑问，这极有可能是设计中的关键路径。那我们该怎么做呢？还是老思路，减少组合逻辑。我们可以对状态的输出进行分析，对它们进行重新分类，并根据这个重新定义成一组组小状态机，通过对输入进行选择(case 语句)并去触发相应的小状态机，从而实现了将大的状态机切割成小的状态机。在 ATA6 的规范中（硬盘的标准），输入的命令大概有 20 十种，每一个命令又对应很多种状态，如果用一个大的状态机（状态套状态）去做那是不可想象的，我们可以通过 case 语句去对命令进行译码，并触发相应的状态机，这样做下来这一个模块的频率就可以跑得比较高了。

总结：提高工作频率的本质就是要减少寄存器到寄存器的时延，最有效的方法就是避免出现大的组合逻辑，也就是要尽量去满足四输入的条件，减少 LUT 级联的数量。我们可以通过加约束、流水、切割状态的方法提高工作频率。