

第三章、打怪抢宝之流水与并行

喜欢玩魔兽的朋友都知道，游戏的前期，不论是基地、士兵或是英雄都是很弱的，此时英雄主要带领小兵到处去打怪练级，同时抢些小小的宝物。这一章的任务就是打怪抢宝，而流水与并行就是两个小小的宝物。称其为“小小的宝物”是恰当的，因为从第四章到第七章，将会隆重推出四件上古之神器：1) 重定时、2) 展开、3) 折叠和 4) 脉动。这四个神器相对于流水与并行这两个小小宝物那可谓神通广大。实际上，流水就是重定时的特例，而并行又是展开的特例。

题外话：前期不把级练好，那后期更甭想用得起上古神器了，所以这一章大家务必细心体会。

下面采用结合实例讲解的方式，大家看了之后应该学会如何将流水与并行的技术应用到其他的 DSP 程序中去。本章分两个小节：

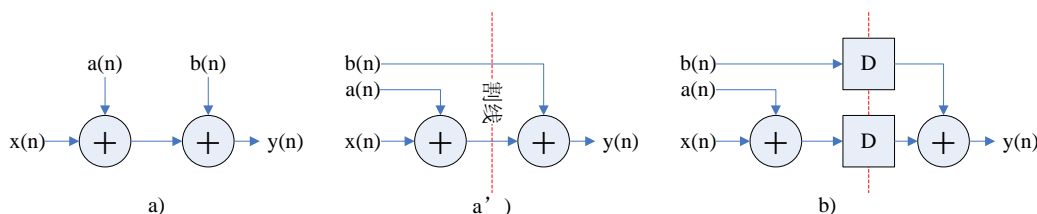
1. 主要介绍如何在 DSP 程序中应用流水与并行技术，同时还给出了流水的严格定义（并行没有进行严格的定义，大家先体会其过程，在展开一章将对并行做进一步的探讨）。
2. 使用一种技术，就应该知道这种技术所能带来的好处和坏处。流水和并行的坏处大家可以发帖子讨论，而好处呢？主要有高速度和低功耗，前者显而易见，不再多说，而后者低功耗将被详细讨论。

讲解：第一节、流水与并行

一、初步的认识

如下图一 a) 的数据通路，迭代方程非常简单，就是 3 个数组对应元素相加，即

$$y(n) = x(n) + a(n) + b(n)$$



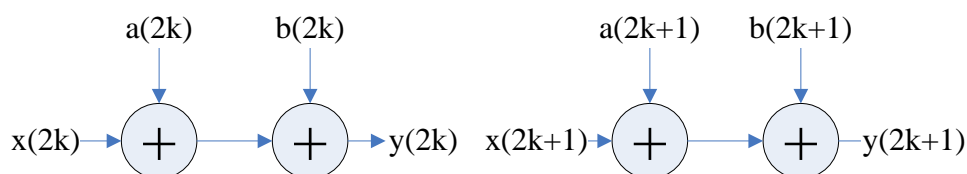
图一、示例，a)数据通路；b)流水线结构

数据通路是一条前向（非递归）路径。从图一 a')可以更清晰地看到前向非递归的特点。对图一中的数据通路，假设加法节点计算时间为 TA ，那么关键路径长度就是 $TA+TA=2*TA$ 。按图一 a')的割线处（割线所隔的两条同向边）插入“流水线寄存器”，也就是延时，就得到二级流水线结构如图一 b)所示。图一 b)二级流水线结构的关键路径长度变为 TA ，比原来的缩短了，这样迭代周期可以缩小，吞吐率也能提高。

对图一 a')进行并行扩展也非常简单，注意观察迭代式子或者图一 a)，节点不存在迭代间的优先关系，也就是说任一次迭代都可以单独进行而与其他次迭代没依赖关系。比如想构造一个 2 阶并行处理系统，那么可以将输入序列 $x(n)$ 、 $a(n)$ 和 $b(n)$ 分为奇偶两列分别由两套相同的硬件电路来算。具体做法是，将 $n=2k$ 和 $n=2k+1$ 带入原迭代式，得两个新的迭代式，如下

$$\begin{aligned} y(2k) &= x(2k) + a(2k) + b(2k) \\ y(2k+1) &= x(2k+1) + a(2k+1) + b(2k+1) \end{aligned}$$

这两个式子完全可以单独计算，根据式子构造的并行硬件如图一 c)所示。



图一、示例，c)并行处理结构

练习：对于这个示例要构造更高阶的并行系统，如何操作？

二、流水线

观察前面的例子，可以看出流水线采用沿着数据通路引入流水线寄存器的方法来缩短有效的关键路径，从而缩短迭代周期（采样周期也会相应缩短），提高系统吞吐率。这里先直接给出流水线的三句口诀（也是流水线的严格定义，见课本）：

- 1、一个架构的速度（或时钟周期）由任意两个寄存器间、或一个输入与一个寄存器间、或一个寄存器与输出间、或输入与输出间路径中最长的路径限定。
- 2、这个最长的路径或“关键路径”可以通过在架构中恰当插入流水线寄存器来缩短。
- 3、流水线寄存器只能按照穿过任一图的前馈割集的方式插入。

为了说明第三点，需要引入两个定义：

- 割集：割集是一个图边的集合，如果从图中移去这些边，图就成为不相连的了。
- 前馈割集：如果数据在割集的所有边上都沿前进方向移动，这个割集就称为前馈割集。

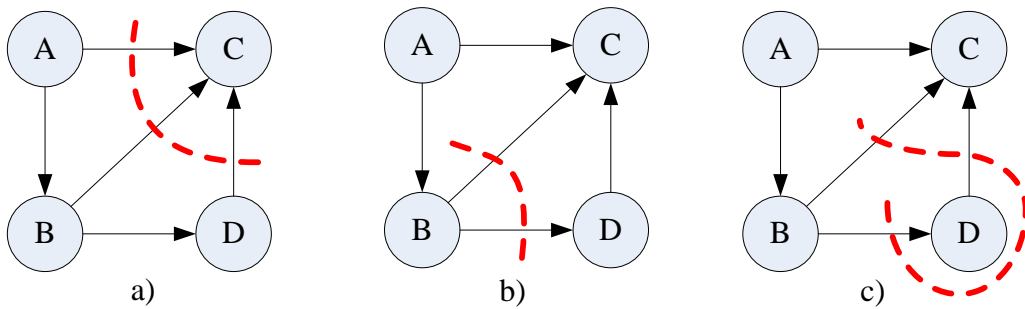
对于 1，等价的说法就是零延时最长路径决定了架构的速度，也就是关键路径。对于 2，在初步认识的示例中，的确是通过在架构中插入流水线寄存器从而缩短了关键路径长度，由原来的 $2 \cdot TA$ 变为 TA ，缩短一半。但是要注意，如果想通过插入流水线寄存器的方法来缩短关键路径，那插入的位置就要恰当，必须保证流水线寄存器出现在关键路径的某些边上。众所周知，关键路径上的边必须是零延时的，如果在关键路径的某些边上插入流水线寄存器，那么这条关键路径将被这些流水线寄存器斩断成若干段，架构中新的关键路径肯定小于等于旧的关键路径。之所以有“等于”，是那么一种情况，如果流水线寄存器不是插入到关键路径的某条边上，那原始关键路径没有被斩断，架构的关键路径长度不变。还有另一种情况，在一个架构中，存在多条等长的关键路径，流水线寄存器只斩断其中一些，还有一些没斩断，那么架构的关键路径长度还是不变。反过来说，流水线寄存器要插入到能斩断所有关键路径的边上才能缩短关键路径长度。

总之，插入流水线就是想斩断原始的所有关键路径，以得到更短的关键路径。那么插入流水线寄存器，是不是指随便在某些边上加入延时就行了呢？不是的，要想保证架构的功能不变，必须按照规定的法则进行流水线寄存器的插入。看第 3 个口诀：流水线寄存器只能按照穿过任一图的前馈割集的方式插入。

-----我们先把割集和前馈割集弄清楚-----

割集是一个连通图的边的集合，如果从图中移去割集中所包含的边，该连通图就会成为“两个”不相连的连通子图。首先割集中的边要足够多，多到只要移去割集中的边，就能把原来的一个连通图变为两个孤立的连通子图。其次，割集中的边要足够少，少到只要少移去一条割集中的边，就得不到不相连的两个连通子图（注：割集的严格定义可以参考图论方面的书籍）。这里要明白一点，割集中的边是充分的但不多余，恰好可使一个连通图变为两个

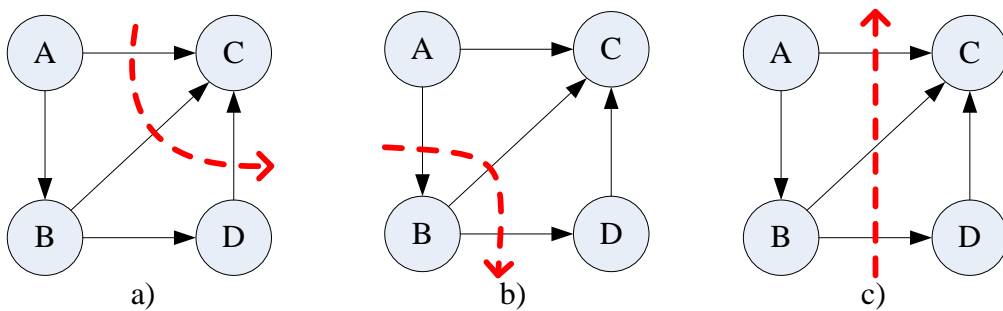
不相通的连通子图，如下图二的几个示例：



图二、割集示例：a)割集；b)不是割集，不充分；c)不是割集，多余了

图二 a)是割集，边不多也不少；b)不是割集，如果多加 $A \rightarrow B$ 的边就是了；c)也不是割集，多了 $B \rightarrow C$ 的边。

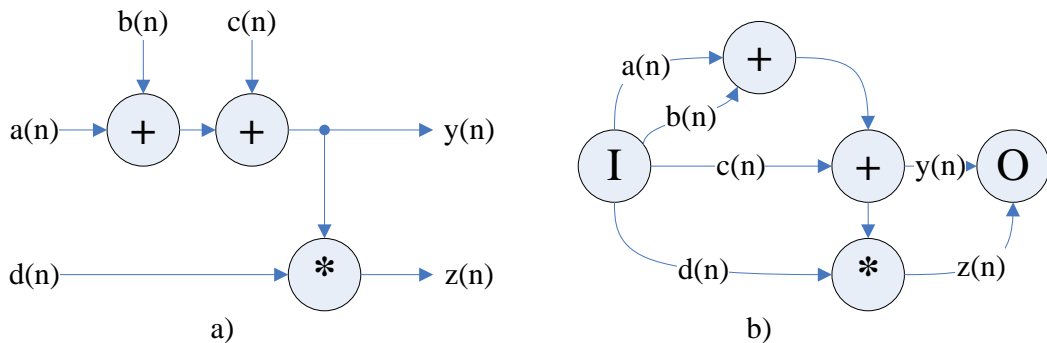
如果指定割线方向，如图三所示，每条割线（红色虚线）都有一个方向。如果割集中的边都在割线的同一侧，那么该割集成为单向割集，比如图三 a)和 c)；反之，割集中的边不全在割线同一侧，则称该割集为双向割集，如图三 b)。课本上说流水线寄存器只能在前馈割集上插入，其实有些不恰当，应该是在单向割集上插入。



图三、割集示例：a)单向割集；b)双向割集；c)单向割集

-----回到流水线讲解-----

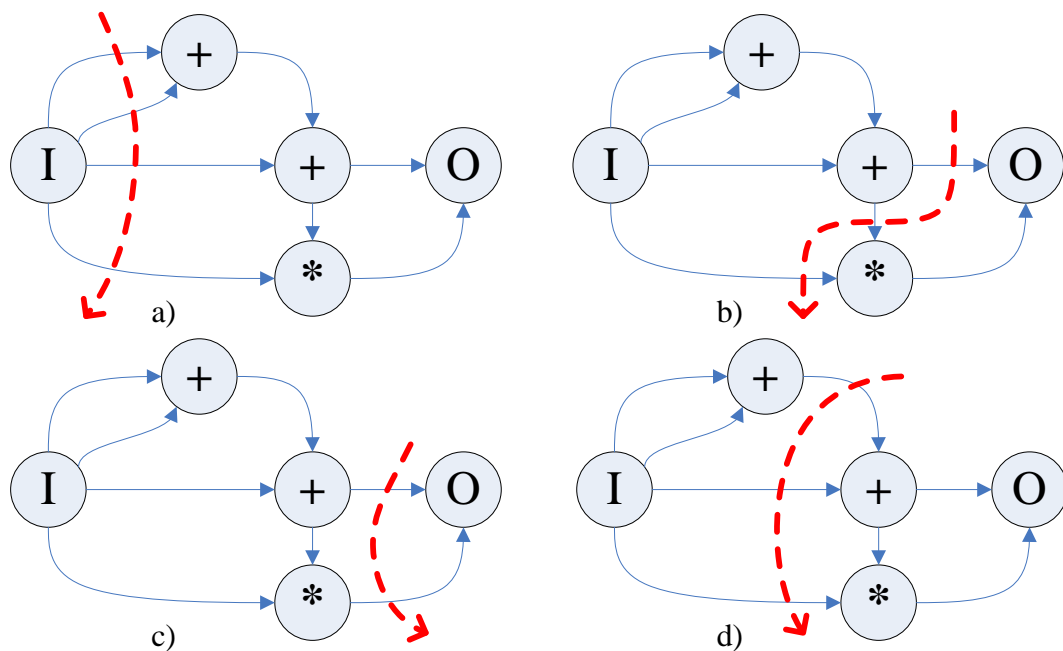
明白了割集和单向割集（也就是课本上的前馈割集，以后我们均认为在单向割集上插入流水线寄存器），此外还有一点需要注意：当 DSP 系统有多个输入端和多个输出端时，应该先把多个输入端折合成一个输入节点，多个输出端折合成一个输出节点，当然了，输入节点和输出节点计算不耗时间，也就是计算时间为 0u.t.。比如图四 a)的 DSP 框图，输入有四个分别为 $a(n)$ 、 $b(n)$ 、 $c(n)$ 和 $d(n)$ ，输出有两个分别为 $y(n)$ 和 $z(n)$ ；图四 b)为其 DFG，所有输入折合为节点 I，所有输出折合为节点 O。



图四、多输入多输出系统及其DFG

下面以图四的 DFG 为例，来练练手，看如何插入流水线寄存器以缩短关键路径。首先假设加法节点计算时间 $TA=1u.t.$ ，乘法节点计算时间为 $TM=2u.t.$ ，那么图四 b) 的 DFG 关键路径

为 $(0) \xrightarrow{I} (1) \xrightarrow{+} (1) \xrightarrow{+} (2) \xrightarrow{*} (0) \xrightarrow{O}$ ，长度为 $TA+TA+TM=4u.t.$ 。列出图四 b) DFG 的一些单向割集如图五所示。

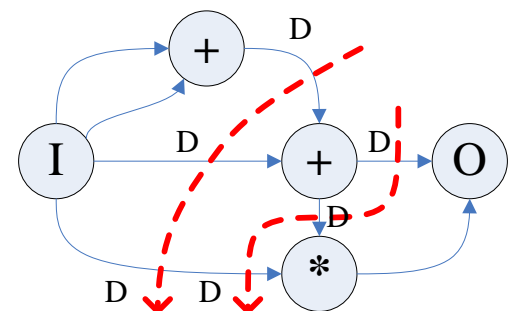


图五、四种可能的单向割集

图五 a) 的割集不算斩断有效关键路径，因为决定关键路径长度的是两个加法节点和一个乘法节点，而 a) 只是将一个计算时间为 $0u.t.$ 的输入节点从关键路径上去掉而已；b) 斩断有效关键路径，新的关键路径为 $I \rightarrow +$ 或者是 $* \rightarrow O$ ，长度均为 $2*TA=TM=2u.t.$ ；c) 和 a) 类似，不算斩断关键路径，仅仅是把计算时间为 $0u.t.$ 的输出节点从关键路径上去掉而已；d) 斩断有效关键路径，新的关键路径为 $+ \rightarrow * \rightarrow O$ ，长度为 $TA+TM=3u.t.$ 。比较上述的四种流水线插入，b) 的插入最有效，可以使得关键路径缩短到 $2u.t.$ ，次之是 d) 的插入，关键路径缩短到 $3u.t.$ ，而 a) 和 c) 属于无效的流水线插入。

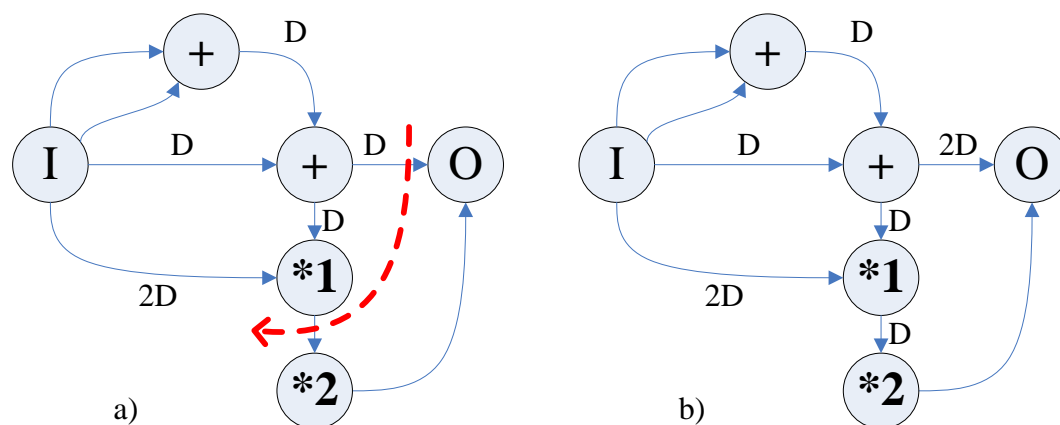
练习：对图五，还能找出其它能作为流水线插入的单向割集吗？

对于图五的示例，如果进行两次流水线插入，也就是先进行 b) 再进行 d)，可得如下结构

构 ，虽然关键路径只剩一条 $* \rightarrow O$ （只进行 b) 的话，就有两条： $I \rightarrow +$ 和 $* \rightarrow O$ ），但是关键路径长度仍然为 $TM=2u.t.$ ，看起来第二次的流水线插入 d)

并没有起到作用，因为它只斩断了 $I++$ 这条关键路径，还剩另一条等长的关键路径 $*O$ 没被斩断，而且 $*O$ 似乎也没办法斩断了。

下面我们要介绍的细粒度流水线将可以解决 $*O$ 没法斩断的问题。仔细分析新的关键路径 $*O$ ，如果乘法节点能拆分，比如拆成两个部分，分别为节点 $*1$ 和节点 $*2$ ，且两个节点计算时间相等都为 $1u.t.$ ，也就是原始乘法节点的一半，如图六 a)，那么可以再次插入一级流水线（红色割线所示），就可以斩断乘法节点，从而得到关键路径长度为 $1u.t.$ 的“终极”细粒度流水架构如图六 b)所示。



图六、a)乘法节点可拆分;b)细粒度流水线架构

细粒度流水线，无非就是假设那些决定架构关键路径的节点是可以拆分的，将其拆分之后再运用流水线寄存器将其斩断，从而进一步缩短关键路径。反过来，我们把节点不可拆分的流水线称为“粗粒度”流水线。

流水线想要用好，除了按正确的方法（单向割集法）进行插入之外，还要注意分析插入的位置，使得插入的流水线寄存器恰好能斩断有效的关键路径，从而得到更短的关键路径，如若不然这种流水线插入就没有意义了。

题外话：课本以 3 阶 FIR 滤波器为例介绍流水线，帖子中我们不用这个例子，留给大家自己去分析。这样流水线一节就有两个示例分析可以帮助你理解和掌握它。大家看书切记盲从，应该与实际结合起来进行思考，又时书上的一些说法是不太恰当，我想主要是由于文字表达的问题，而不是作者的水平有问题。说实在的，只有真正用心的人才能听得懂作者要说的意思。这本书的很多技术都是作者千辛万苦从 N 多文献中总结出来的，疏忽之处在所难免，但如果没有这本书，你想深入掌握这些本事，可能得自己去看好多文章，有时都不知从何下手，不论如何，谢谢这个印度老儿 Keshab K.Parhi。

此外，课本 3.2.1 节数据广播结构不属于流水线内容，但其中所用的转置变换有时是非常有用的，这一小节留个大家自己阅读，只要弄明白如何用 SFG 进行架构的转置操作就行。

这段内容看完，大家至少记住两点：1) 流水线的插入方法，单向割集法；2) 怎样插入流水线才能缩短关键路径。

三、并行处理

在课本的 P51，一上来作者就揭示了流水与并行的对偶关系，即一个 DSP 程序，如果可以划分为一组不相关的计算能够在一个流水线系统中按交替的方式计算，那么它也能够利

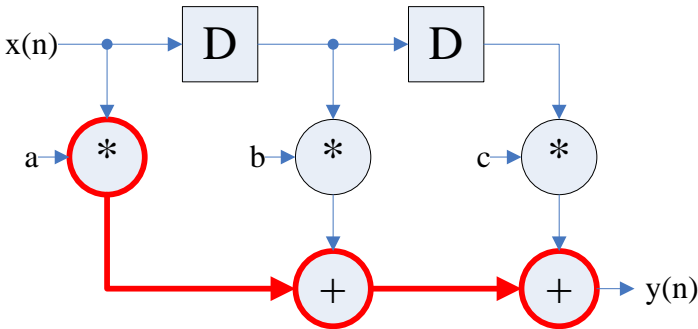
用复制的硬件按并行处理的模式计算。这说明了 DSP 固有的并行性可以通过流水线来挖掘，同时也可以通过并行处理来挖掘。-----这种流水和并行的关系，作为思考题，大家好好琢磨琢磨，心得体会发帖讨论。

这一节所讨论的并行处理，主要通过观察来构造，当然了通过观察肯定很难构造出复杂系统的并行处理架构，之所以在这讨论并行，主要是让大家为第五章的展开开个门，而在第五章将给出一套简洁（2 步法）而强大的构造并行硬件的方法。

示例分析：3 阶 FIR 滤波器的并行处理，迭代公式如下

$$y(n) = ax(n) + bx(n-1) + cx(n-2)$$

直接实现形式如下图



关键路径为*++，长度是 TM+TA+TA。为了将其改造为一个 3 阶并行处理系统（可同时处理 N 份样本点的系统称为 N 阶并行处理系统），可先对其迭代公式进行如下改造，将 $n=3k$ 、 $3k+1$ 和 $3k+2$ 带入原迭代公式，得到如下 3 个新迭代公式

$$\begin{aligned} y(3k) &= ax(3k) + bx(3k-1) + cx(3k-2) \\ y(3k+1) &= ax(3k+1) + bx(3k) + cx(3k-1) \\ y(3k+2) &= ax(3k+2) + bx(3k+1) + cx(3k) \end{aligned}$$

观察以上三个式子，发现只要输入 $x(3k)$ 、 $x(3k+1)$ 和 $x(3k+2)$ 就可以计算出 $y(3k)$ 、 $y(3k+1)$ 和 $y(3k+2)$ ，其他的输入 $x(**)$ 都是之前就已经输入的数据，可由 $x(3k+1)$ 和 $x(3k+2)$ 延时而得。也就是说这三个式子可以并行计算，类似的步骤还可以构造 4 阶并行处理系统或更高阶并行处理系统（你试试）。

并行处理系统也称为块处理系统，因为系统的输入是一块一块的，比如我们所讨论的 3 阶并行处理系统，每次输入都是 3 个数据为一块，即 $x(3k)$ 、 $x(3k+1)$ 和 $x(3k+2)$ ，输出也是 3 个数据为一块，即 $y(3k)$ 、 $y(3k+1)$ 和 $y(3k+2)$ 。块中所包含的数据个数 L 称为块尺寸，在我们系统中，块尺寸 L 为 3。并行处理系统中的一个延时，跟原始串行系统中的延时不太一样，称为块延时，也称 L 级减慢，称为 L-slow。例如示例中的 3 阶并行处理系统，有三个输入端口，每个周期输入情况如下表所示

周期 k	端口 0---x(3k)	端口 1---x(3k+1)	端口 2---x(3k+2)
0	x(0)	x(1)	x(2)
1	x(3)	x(4)	x(5)
2	x(6)	x(7)	x(8)
3	x(9)	x(10)	x(11)
等等			

对于某个输入端，例如端口 2，输入为 $x(3k+2)$ 的子序列，看下图，问，当 $x(8)$ 出现在输入

端时，寄存器（延迟单元）里的数据是什么？

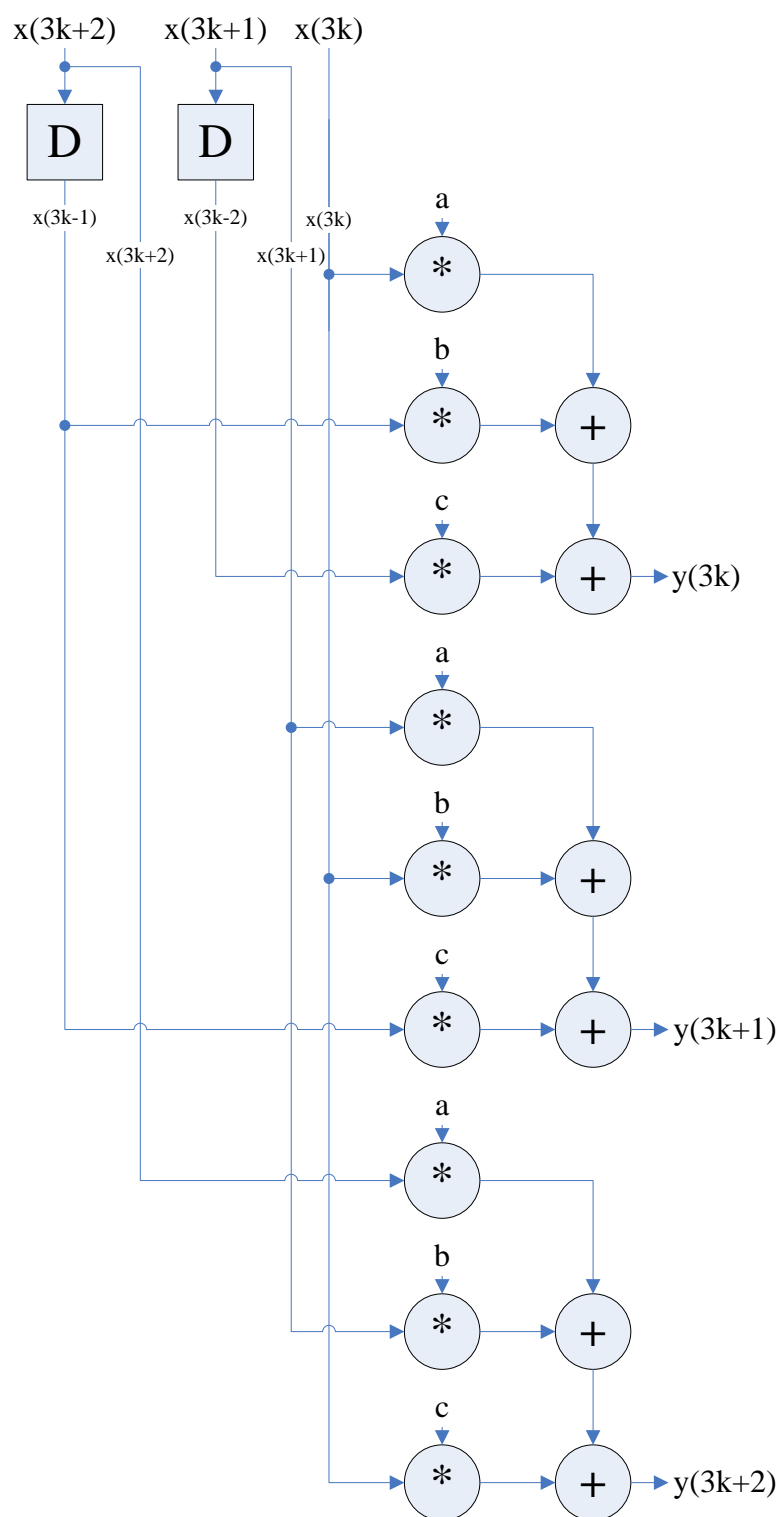


显然延时单元里的值是 $x(5)$ ，因为从输入序列可以知道 $x(8)$ 之前的一个输入就是 $x(5)$ ，在原始的串行系统中，输入序列是 $\dots x(8)x(7)x(6)x(5)\dots$ ， $x(8)$ 延迟一个单位就是 $x(7)$ ，但是块系统中不一样， $x(8)$ 延迟一个单位是 $x(8-3)=x(5)$ ，其中 3 为块尺寸。从代数式看，就是 $x(3k+2)$ 延迟一个单位，得到 $x(3(k-1)+2)=x(3k-1)$ 而不是 $x(3k+2-1)=x(3k+1)$ 。综上所述，对于块尺寸为 L 的并行处理系统，如果某端口输入为 $x(Lk+m)$ ， $m=0,\dots,L-1$ ， k 为非负整数，延时一个单位得到 $x(L(k-1)+m)=x(Lk+m-L)$ 而不是 $x(Lk+m-1)$ 。

明白了块延迟，就可以根据新得到的三个迭代式画出 3 阶 FIR 滤波器的 3 阶并行处理架构如下图：

此外，课本上还讨论了一下块并行处理系统中时钟周期和采样周期的差别。需要注意的是，课本上将迭代周期等价于采样周期，这与课本的其他章节不一致，我们约定，迭代周期就是时钟周期，即 $T_{\text{iter}}=T_{\text{clk}}$ ，根据系统的并行度与迭代周期，可计算出采样周期为 $T_{\text{sample}}=T_{\text{iter}}/L$ ， L 为并行处理系统的块尺寸。

哈哈，终于可以休息了。。大家学完这些知识还需要多体会多练习，时刻想着运用到自己的工作中去，不要让这些“小小宝物”生锈了。



上一节我们以增加系统吞吐率为目的，分别介绍了流水线和并行处理两项技术。流水线技术用于斩断有效关键路径（也就是缩短关键路径），从而提高系统运行的频率，吞吐率也得到相应的提高；并行处理技术通过构造多份“相同”的硬件电路，以同时处理多个输入样本点（提高并行程度），来提高吞吐率。

有时候，设计的目标不是一味的提高系统的吞吐率，而是更加关注于系统的面积或者功耗等

其他指标的改善。比如在移动多媒体终端上，当系统的吞吐率达到预定的指标，就没有必要再往上提升，此时，如何缩小移动多媒体终端的面积以方便携带 或者 降低系统功耗以延长电池的续航时间 才是重点。

使用智能手机的同志都明显感觉到手机电池充电带来的麻烦吧，动不动就要为其充电。功能强大的移动设备的功耗是极其备受关注的，为了降低功耗甚至可以牺牲面积。这一节关于低功耗设计的分析虽然很“浅显”，但是却能让大家对流水和并行在低功耗设计中的功用有所认识。

讲解：低功耗设计的利刃——流水与并行

电路设计说到底就是在各个性能指标上玩折衷，削弱某一个指标以加强另一个指标。为了做到这一点，必须先弄清楚所涉及的各个指标之间的关系。这节的讨论主要涉及面积和功耗两个指标，由于使用了流水与并行技术，使得系统的面积增大了，但是却能带来功耗降低的好处。下面首先从物理电路的角度去分析影响功耗的一些系统因素，直到我们对功耗的来源产生一个认识后，将给出两个 CMOS 电路分析的“简化”公式——延时公式和动态功耗公式。这两个公式将帮我们弄清楚流水与并行用于低功耗设计的窍门。

从《数字电子线路》和《大规模 VLSI 设计》等书籍中，可以了解到，要驱动门电路需要一个工作电压 V_0 ，比如 TTL 电路需要 5v，CMOS 电路需要 3.3v 等。当然了以上这些都是标准的电压，在具体的设计中，你可以使用其他电压如 1.5v，不一定非得是 5v 或 3.3v。另外，为了驱动晶体管或者场效应，电压不能低于某个阈值 V_t ，如晶体管电路阈值约为 0.4~0.7v（PN 节导通电压）。

本质上，信号从低电平切换到高电平可等价为一个电容充电的过程；反过来，信号从高电平切换到低电平可等价电容放电的过程。数字电路中的信号电平切换可等价电容充放电，而电容充放电是需要时间的，可以用充放电的速率来衡量充放电的快慢，那么下面我们先来考察一下什么因素会影响电容充放电的速率。

电容充电公式：

$$U(t) = U(0) * (1 - e^{-t/RC}) \quad (1)$$

电容放电公式：

$$U(t) = U(0) * e^{-t/RC} \quad (2)$$

充放电 速率：

$$\left| \frac{dU(t)}{dt} \right| = \frac{U(0)}{RC} * e^{-t/RC} \quad (3)$$

充电和放电的速率公式相同。仔细观察公式(3)，要想提高充放电速率，有两个可行的方法：1) 增大 $U(0)$ ；2) 减小 C 。增大 $U(0)$ 就是提高电路的驱动电压 V_0 ，减小 C 就是切断有效关键路径，使得充放电电容尽可能小。在实际的系统中，这些电容充放电速率越快，表示电路高低电平间切换就越快，从而电路的运行频率也就可以更高；反之则不然。

有了以上的认识，我们给出一个简化的 CMOS 传播延时公式，公式所反映的内容就是驱动电压 V_0 、充放电电容 C_{charge} 和信号传输延时 T_{pd} 的关系（信号传输延时也就是电平切换时间）

$$T_{pd} = \frac{C_{charge} V_0}{k(V_0 - V_t)^2} \quad (4)$$

其中 C_{charge} 为等价的充放电电容， V_0 为门电路工作电压（电源电压）， V_t 为门电路阈值电压，参数 k 为工艺参数。从公式(4)可以看出， C_{charge} 越小或 V_0 越大，都会使得 T_{pd} 越小，也就是传播越快（电平切换越快），这和前面对电容充放电的分析结论是一致的。

练习：使用流水线提高时钟频率的本质是什么？

答案：流水线斩断了有效关键路径路径，相当于把系统中最长传播延时路径上的 C_{charge} 分成了多份，也就是 C_{charge} 减小了。由于传播延时 T_{pd} 变小了，因此系统可以运行得更快。

到此上面讨论告一段落，下面再来讨论电路中功耗的形成。电路的功耗分为静态功耗和动态功耗，静态功耗我们暂时拿它没办法，这里只讨论动态功耗。动态功耗是系统功耗的主要来源，占绝大部分比重。以 CMOS 电路为例，动态功耗的形成是由于信号高低电平切换的中间过程，会有一小段“短暂”的时间，互补对称 PMOS 和 NMOS 同时导通，形成毛刺电流。该毛刺电流非常大，根据焦耳公式 $Q=I^2 \cdot R \cdot t$ ，大电流产生大热量，对于频率越高的电路（如 CPU），毛刺电流越频繁，故而发热量也极大。衡量 CMOS 动态功耗可以用如下简化公式

$$P = C_{total} V_0^2 f \quad (5)$$

C_{total} 为电路总的等价电容， V_0 为门电路工作电压， f 为电路时钟频率。从公式(5)可以看出，削减功耗的可行方法有：

1. 减小电路总电容 C_{total} 。 C_{total} 与电路资源量相关，减少资源占用可以减少 C_{total} ，有时不容易做到。
2. 减低电路驱动电压 V_0 。
3. 减小电路时钟频率 f 。

此外，

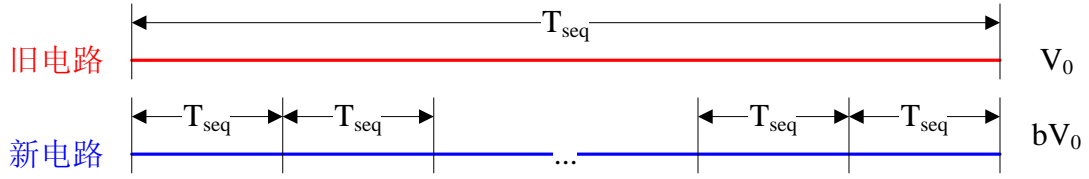
4. 减少信号的状态切换，比如设计状态机，使得相邻的状态只需要尽可能少的位翻转等等。

注： C_{charge} 与 C_{total} 是不同的概念。大家务必弄清楚，前者与关键路径有关，用于衡量信号延时，后者与整个系统有关，用于衡量系统的整体动态功耗。

有了以上的公式(4)和公式(5)，就可以具体来分析流水和并行是怎么样做到降低系统功耗。下面分 3 段来讨论，先是流水低功耗设计，然后并行低功耗设计，最后是流水并行（混合）低功耗设计。

注：关于大量运用流水并行技术进行低功耗设计的项目可以参考 *opencore* 上由 Xuke 大侠发起的 *h.264 解码器设计项目 (nova)*。Xuke 现在 IBM 担任 **Staff R&D Engineer**。

一、用流水线降低功耗



如上图示，单独把旧电路中关键路径（红色线段）拿出来分析，加入 M 级“理想的”流水线之后得到新电路（蓝色线段）。必须注意一点，不论是旧电路还是新电路，都采用相同的时钟频率来运行，很多同志一说到流水线，就会定势思维想到提高系统时钟频率，其实提高时钟频率只是流水的其中一个用途，而这里我们不需要提高系统时钟频率，而是需要降低系统功耗。

前面分析过，流水线技术斩断了原始系统的有效关键路径，也相当于分解了原始系统的有效关键路径充电电容。为了不至于混淆，约定如下符号：

$C_{o-charge}$ ($C_{o-charge}$) 为旧电路关键路径上的等价充放电电容，上图的整段红线所代表的电容；

$C_{n-charge}$ ($C_{n-charge}$) 为新电路关键路径上的等价充放电电容，上图一小段蓝线所代表的电容；

V_0 为旧电路工作电压；

bV_0 为新电路工作电压， b 是大于 0 小于 1 的数（其实从这就可以看出我们要通过降低工作电压的方法来降低功耗）；

P_o 为旧电路功耗；

P_n 为新电路功耗；

C_{total} 新旧电路的等价电容，可以近似认为流水线不改变电路总电容（实际上加入流水线会使 C_{total} 增大一点点）。

由于“理想” M 级流水线的插入，使得 $C_{n-charge} = C_{o-charge}/M$ 。延时公式(4)中的充放电电容变小，而我们又不需要改变传输延时 T_{pd} ，那么肯定能带来工作电压的减小，关系如下式

$$T_{pd} = \frac{C_{n-charge} bV_0}{k(bV_0 - V_t)^2}$$

同时又有

$$T_{pd} = \frac{C_{o-charge} V_0}{k(V_0 - V_t)^2}$$

合并以上两式，并用 $C_{n-charge} = C_{o-charge}/M$ 带入，得到

$$b(V_0 - V_t)^2 = M(bV_0 - V_t)^2$$

这是一个关于 b 的二次方程，容易求出 b 。取合理的 b ，则新电压为 bV_0 。

注：求解二次方程，取合理解就不用多说了吧。这里让我回忆起初中的代数课了，那时老

师就开始讲如何求解二次方程了。

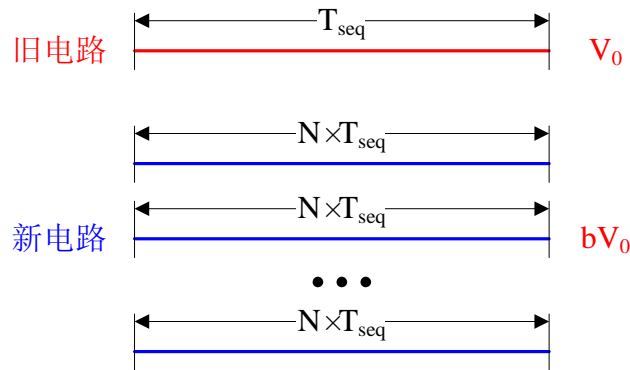
新电路的功耗为

$$P_n = C_{total} (bV_0)^2 f = b^2 C_{total} V_0^2 f = b^2 P_0$$

新电路功耗为旧电路的 b 平方倍，注意到 b 是(0,1)的小数，这个功耗降低是很可观的，用于工作电压降低到原来的 b 倍，导致功耗降低到原来的 b 平方倍。

以上的过程让我们明白，对原始电路的有效关键路径进行“理想的” M 级流水变换之后，在维持系统时钟不变的情况下，系统工作电压可以降低到多少。实际低功耗设计分析要比这里的分析更为复杂，但是本质都是一样的，大家就举一反三吧。

二、用并行处理降低功耗



类似于流水低功耗的讨论，并行处理也能用于低功耗设计。如上图，原始电路的关键路径如红线段所示，运行频率为 T_{seq} ，系统电压为 V_0 ；使用 N 阶并行处理来构造新电路，如蓝色线段所示，由于我们不需要提高系统吞吐率，所以在 N 阶并行处理电路中应该以原始时钟频率的 N 分之一，也就是以 $N \cdot T_{seq}$ 为周期来运行（系统速率变慢，但吞吐率不变）。由于系统速度的减慢，可以使得我们降低工作电压以实现低功耗设计。

按流水中类似的做法，列出公式求解，看看 N 阶并行电路，电压可以降低到什么程度。注意并行处理没有斩断有效关键路径，故而关键路径的充放电电容不变，均为 C_{charge} ，但是系统的总电容为原来的 N 倍，即 $N \cdot C_{total}$ 。看如下过程

$$NT_{seq} = N \frac{C_{charge} V_0}{k(V_0 - V_t)^2} = \frac{C_{charge} bV_0}{k(bV_0 - V_t)^2}$$

化简如上的等式，有

$$N(bV_0 - V_t)^2 = b(V_0 - V_t)^2$$

哈哈，这个结论竟然与流水的结论一模一样，大家看得出来吗？真是异曲同工、异路同归。电路设计就是那么搞笑，博大精深啊。

同样的道理，求解关于 b 的二次方程，得到合理的解。新电路电压为 bV_0 ，新电路功耗为原来的 b 平方倍， b 是(0,1)之间的小数。

三、流水线与并行处理的结合

流水与并行结合的情况就留给大家练练手了。这里不再多说，直接列出分析的公式和结论如下（所有符号按前面的约定）：

$$NT_{seq} = N \frac{C_{o-charge} V_0}{k(V_0 - V_t)^2} = \frac{C_{n-charge} b V_0}{k(b V_0 - V_t)^2} = \frac{\frac{C_{o-charge}}{M} b V_0}{k(b V_0 - V_t)^2}$$

化简，得

$$MN(bV_0 - V_t)^2 = b(V_0 - V_t)^2$$

与前面的结论非常类似，都是关于 b 的二次方程，不同的地方只是最左边的乘数分别为 M 、 N 和 MN 而已。求解得到合理的 b ，则新电路电压为旧电路电压 b 倍，新电路功耗为旧电路功耗的 b 平方倍，再说一次， b 是大于 0 小于 1 的小数。

小结：

这一贴我们主要是“简单”地了解如何使用流水线和并行处理技术进行低功耗设计，说白了就是牺牲电路面积（流水和并行占用了更多的电路资源），换取了较低的工作电压，从而降低系统的功耗。具体的分析过程只涉及到两个公式：公式(4)和公式(5)。为了让大家有个形象的认识，对这两个公式做了一些分析，更为具体的内容请参看相关的其他书籍。

此外，课本上还有一些习题，大家可以自己做做。最后就是强烈推荐大家去看看 Xuke 大侠的 opencore 设计 nova，会让你对低功耗设计的认识更上一层楼。