

双刃剑之一展开

在我们讨论的四种上古神器（包括：重定时、展开、折叠和脉动）中，展开和折叠是相生相克的。这两件神器一伸一缩，代表两种相反的做法，但各有其用武之地。这一帖推出的上古神器是：双刃剑之一展开。

那么展开是什么意思呢？看看课本上的定义：**展开用于产生一个新的程序来描述原有程序的多次迭代。更具体的说，以展开因子 J (unfolding factor J) 展开一个 DSP 程序，就会产生一个以原程序连续迭代 J 次的新程序。展开也被称为“环路展开”。**试图直接去理解这个定义很困难。我猜测这是因为看问题的角度不正确而致，往往直到看完了整章才恍然大悟。我们不急着进入正题，而是先来谈谈 DSP 程序到底是个什么，可以怎么来看待 DSP 程序。

个人见解：

DSP 程序本质上是对一系列的输入数据进行某种形式的处理（如滤波），然后按一定的次序输出结果数据。可由软件或者硬件来进行这种处理，但不论何种实现形式，都会涉及到数据的存储和计算。如果是软件实现，数据是存储在用户定义的变量中；如果是硬件实现，数据存储在延时单元中。DSP 程序具有特定的功能，对于设计者而言，只要能保证 DSP 的功能不变，可以对 DSP 的具体实现进行各种调整，以满足实际应用的指标。

综上所述，DSP 程序的实现可以看成是对输入数据进行存储和计算的过程，期间还会产生中间数据，也需要临时的存储。打个比方，制衣厂将面料做成漂亮的衣服，面料就是输入数据，衣服就是输出数据，要将面料做成衣服需要经过多道工序，中间的半成品需要临时仓库来存储；同样是制衣厂，完成相同功能，但是工厂的部门配置、员工组织和仓库等却可以因地制宜。我们讨论的其他三件神器，都是可以看做一个“工厂”，在保证功能不变的前提下，可以对内部的数据存储和计算流程进行重新的安排，以便达到各个性能指标之间的折中。

下面的讨论分三节：

1. 展开的概念和做法，读完这一节，就可以对任何一个没有开关（多路选择器）的 DFG 进行 J (J 为任意正整数) 阶展开。
2. 展开的性质。使用展开技术，必然要熟知其性质才能得心应手，对于每一个展开的性质会给出详细的证明，并分析其物理意义。
3. 展开的应用。从展开的对象可分：直路展开、环路展开和开关展开；从展开的用途可分：缩短采样周期和并行处理。

讲解：第一节、展开的概念和做法

首先，来看一个软件程序中循环展开的例子。如图 1 示，一个数组对应元素相加的例子。如果采用串行执行，那么需要循环进行 $N=2000000000000000$ 次加法，可想而知这么多次的计算是需要“一定时间”的。要是现有一个双核平台可以使用，那么如何充分利用双核资源来提高计算速度呢？注意到循环中每次迭代都是互不相关的，可以并行计算，有两种计算形式：1) 分块计算，将数据分为连续的两块，第一部分下标从 1 到 $N/2$ ，第二部分下标从 $N/2+1$ 到 N ，并分别在两个 CPU 上同时进行计算；2) 交叉计算，将奇序号的元素分配到 CPU1 计算，而偶序号的元素分配到 CPU2 计算，看起来就是数据交叉的分配到不同 CPU。

1

示例：软件上的循环展开

```
/*一维矩阵加法*/
```

```
N = 200000000000000000;
```

```
for (i=1;i<=N;i++)
```

```
    c[i] = a[i]+b[i];
```

```
/*映射到多个不同CPU，就可以并发/并行执行*/
```

```
c[1] = a[1]+b[1];
```

```
c[2] = a[2]+b[2];
```

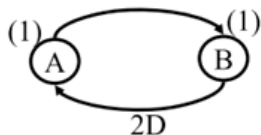
```
...
```

```
c[N] = a[N]+b[N];
```

从这个简单的例子中，可以大致了解到展开的基本思路：采用多份“相同的”资源来同时进行计算。我们所要讨论的展开就是基于“交叉数据分配”原则的并行处理方法。下面来看看一个简单的单环路 DFG 的例子，如图 2。

2

展开 == 并行执行



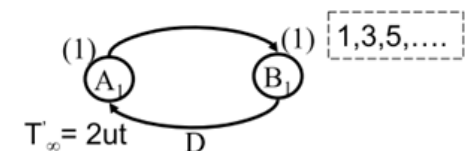
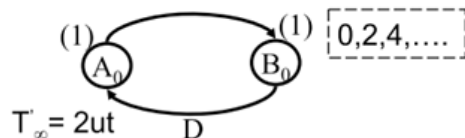
$$A_0 \rightarrow B_0 \Rightarrow A_2 \rightarrow B_2 \Rightarrow A_4 \rightarrow B_4 \Rightarrow \dots$$

$$A_1 \rightarrow B_1 \Rightarrow A_3 \rightarrow B_3 \Rightarrow A_5 \rightarrow B_5 \Rightarrow \dots$$

2 nodes & 2 edges

$$T_{\infty} = (1+1)/2 = 1ut$$

2-unfolded



4 nodes & 4 edges

$$T_{\infty} = 2/1 = 2ut$$

在一个J阶展开的系统中，每一个输入都是J倍降速的，例如对 $x(k \cdot J + m)$ 延时一个单位，得 $x((k-1) \cdot J + m) = x(k \cdot J + m - J)$

首先来挖掘一下这个迭代环路所蕴含的并行性，将迭代的过程画出来，如图 2 左图，A 和 B 的奇数列和偶数列是互不相关的，可以同时进行计算。对左图的 DFG 进行二阶展开，这里暂不管是怎么展开的（稍后会详细讨论），先来看看 2 阶展开前后 DFG 有什么变化：

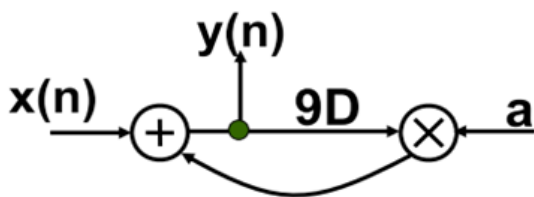
1. 计算节点的个数变为原来的 2 倍。左图的每一个节点对应右图的两个节点，如 $A \rightarrow \{A0, A1\}$ ， $B \rightarrow \{B0, B1\}$ 。
2. 边的数目也是原来的 2 倍，2 条变 4 条。
3. 延时的数目不变，都是 2 个延时。
4. 迭代边界，左图是 1u.t.，右图是 2u.t.。
5. 附加题：采样边界如何变化？左图一次迭代计算一个样本，右图可以计算两个，所以左右图的采样边界相等，即 $1u.t./1=2u.t./2$ 。

细心的同志也许会有疑问：展开后的 DFG，不论是迭代边界还是采样边界，都不比原来的小，而且要命的是展开还消耗了更多的计算节点和边（边对应连线资源），怎么说展开就是并行执行，展开速度就快了呢？

这里要纠正一种错误的“认识”，所谓的迭代边界或者是采样边界，也就是“边界”表征的是迭代 DSP 程序所能达到的最快的运行速度和最大的吞吐率。迭代边界决定系统最高运行频率；采样边界决定了系统最大吞吐率（两者是有区别的，这在前面的章节讨论过）。对于任意的一个迭代 DFG，能不能达到这个极限速度呢？一般是不行的，虽然理论上该 DFG 可以达到这个极限，但是由于实现结构的各种限制，导致实际的迭代周期和采样周期都“远远”大于其相应的边界值。摆在眼前的问题是，如何变换系统架构，使其达到极限速度？展开，展开就可以消除一些制约系统运行速度的限制，使得新的 DFG 运行速度逼近或者完全达到这个极限——这点真的很诱人，有了展开就能将系统的速度和吞吐率提到理论上的极限。

3

展开的例子

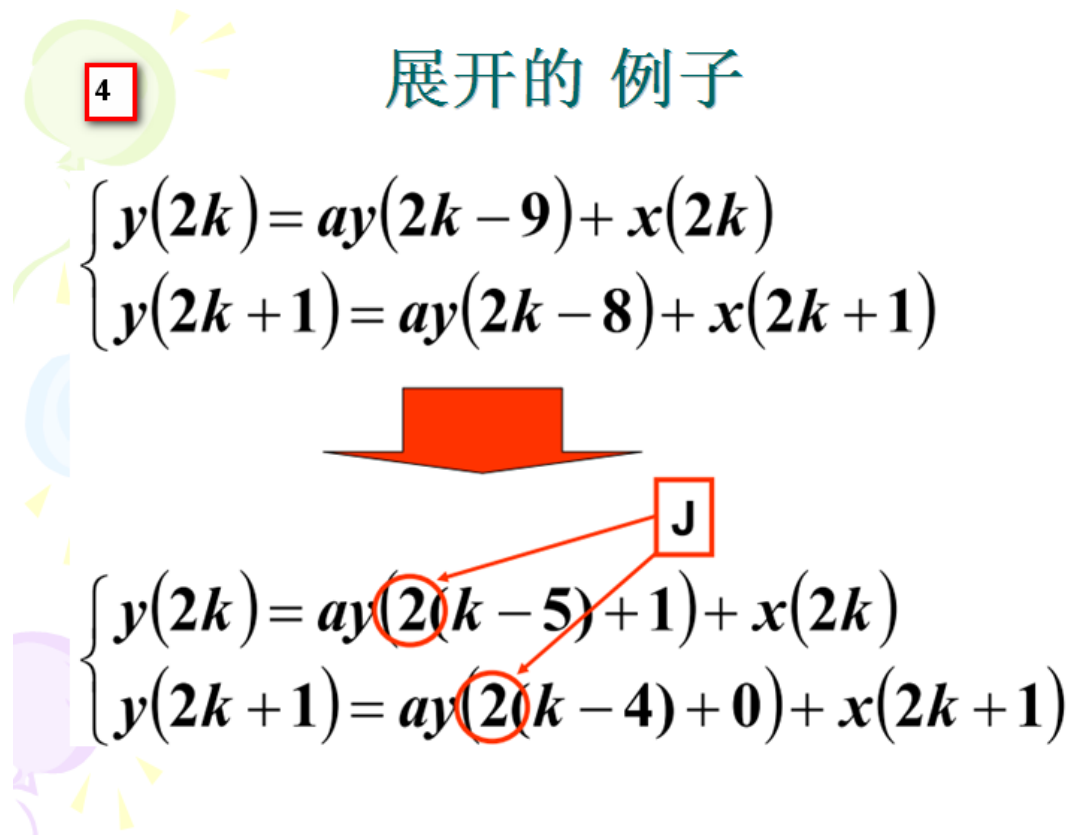


$$y(n) = ay(n-9) + x(n)$$

Unfolding J=2, 2-times parallel \Rightarrow

$$\begin{cases} y(2k) = ay(2k-9) + x(2k) \\ y(2k+1) = ay(2k-8) + x(2k+1) \end{cases}$$

再来看看课本上的 IIR 例子（英文版 P119/中文版 P89），如图 3，采用数据交叉分配法对其进行二阶展开，将输入的奇数列和偶数列分配到 2 份计算硬件上。回忆第三章并行处理的做法，可以直接用到这个问题中，分别令 $n=2k$ 和 $2k+1$ （ k 为非负整数），代入原始迭代公式得到奇数列和偶数列两个新的迭代公式，如图 4，



4

展开的例子

$$\begin{cases} y(2k) = ay(2k-9) + x(2k) \\ y(2k+1) = ay(2k-8) + x(2k+1) \end{cases}$$

↓

J

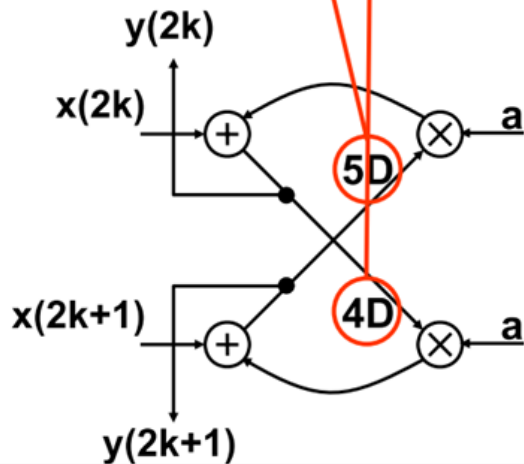
$$\begin{cases} y(2k) = ay(2(k-5) + 1) + x(2k) \\ y(2k+1) = ay(2(k-4) + 0) + x(2k+1) \end{cases}$$

虽然两个新的迭代公式并不是完全独立的，比如偶数列 $y(2k)$ 的计算会用到奇数列 $y(2k-9)$ 这个输出，反过来奇数列 $y(2k+1)$ 也会用到偶数列的 $y(2k-8)$ 这个输出。虽然如此，并不影响对其进行 2 阶展开，根据新的迭代公式画出 DFG 如图 5。

5

展开的例子

$$\begin{cases} y(2k) = ay(2(k-5)+1) + x(2k) \\ y(2k+1) = ay(2(k-4)+0) + x(2k+1) \end{cases}$$



值得注意：在画出新迭代公式的 DFG 之前，先对其进行一些修改，将变量序号表示为

$$J \times k + m, m = 0, 1, \dots, J-1$$

也就是修改为如下形式

$$\begin{aligned} y(2k-9) &= y(2(k-5)+1) \\ y(2k-8) &= y(2(k-4)+0) \end{aligned}$$

其中 $J=2$ 。

采用与第三章相同的做法，从迭代公式入手可以导出一个 DFG 的 J 阶展开结构，但是这个方法还不够强大，也不够直观。某些情况下，我们只能对 DFG 直接进行操作，而没有明显的迭代公式可用或者迭代公式非常复杂。下面将要介绍的展开做法是直接基于 DFG 的，只要能将一个 DSP 抽象成 DFG，就可以对其进行任意阶的展开。

数学符号定义—1

$\lfloor x \rfloor$ is the floor of x , largest integer $\leq x$

$\lceil x \rceil$ is the ceiling of x , smallest integer $\geq x$

$a \% b$ remainder after a/b

首先定义几个数学符号，向下取整 $\text{floor}(x)$ ，向上取整 $\text{ceil}(x)$ ，取余数 $a \% b$ 。重点来了，两步展开算法。DFG 的 J 阶展开，也就是采用 J 倍的复制硬件同时进行计算做法如下

1. 节点的 J 倍复制。对于原始 DFG 中任意一个节点 U ，在新的 DFG 中对应 J 个新的节点，分别记为 U_0, U_1, \dots, U_{J-1} 。
2. 边的连接和边延时的确定。假设原始 DFG 存在一条边 e 如下，边延时（也就是边的权值）为 w

$$U \xrightarrow{w} V$$

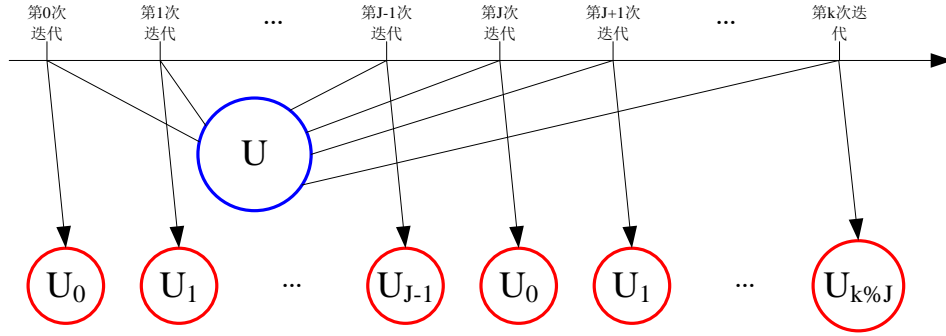
经过第一个步骤后，节点 U 和 V 均被 J 倍复制为对应的 U_0, U_1, \dots, U_{J-1} 和 V_0, V_1, \dots, V_{J-1} ，这些新节点之间将建立起对应原始 DFG 中 e 边的 J 条新边，关系如下

$$U_i \xrightarrow{\lfloor (i+w)/J \rfloor} V_{(i+w)\%J}$$

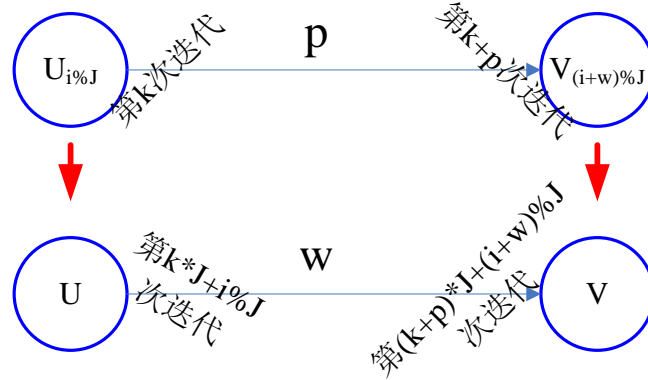
也就是说，新的节点 U_i 和 $V_{(i+w)\%J}$ 之间将形成一条权值为 $\lfloor (i+w)/J \rfloor$ 的新边，其中 $i=0, 1, \dots, J-1$ 。

展开的步骤就这两个，非常简单，相信只要细心去操作，总能正确的对一个 DFG 进行 J 阶展开。但是，为什么这样展开就能保证 DFG 的功能不变呢？课本（英文版 P123/中文版 P92）上有一个证明，思路如下：

J 阶展开，就是将原始 DFG 中任一节点 U 的 J 次连续迭代“交叉分配”到 J 个对应的新节点上同时进行一次迭代，如下图示



原始 DFG 中节点 U 的 J 次迭代，只需由新 DFG 的 J 个相应节点 U_i ($i=0,1,\dots,J-1$) 各进行一次迭代即可完成，原先串行执行 J 次的计算现在由 J 个节点同时执行一次。对于原始 DFG 中的任一条边 $U \xrightarrow{w} V$ ，存在迭代间关系为， U 节点的第 i 次迭代的输出作为 V 节点第 $i+w$ 次迭代的输入，在新的 DFG 中，必需维持这种数据依赖关系不变方可保证功能的正确。从上面的讲解可知， U 节点的第 i 次迭代被分配到 $U_{\{i\%J\}}$ 节点执行，而 V 节点的第 $i+w$ 次迭代被分配到 $V_{\{(i+w)\%J\}}$ 节点执行。假设 $U_{\{i\%J\}}$ 第 k 次迭代输出经过 p 个延时，作为 $V_{\{(i+w)\%J\}}$ 的第 $k+p$ 次迭代输入就能保证新 DFG 功能与原始 DFG 一致，那么肯定有



$$k * J + i\%J + w = (k + p) * J + (i + w)\%J$$

化简公式，有

$$p * J = w + i\%J - (i + w)\%J$$

当 $i=0,1,\dots,J-1$ 时，可化简为

$$p = \lfloor (w + i) / J \rfloor$$

也就是新边的延时 p 由如上公式确定。至此，证明完毕。

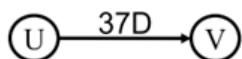
小结：展开是构造并行结构的方法之一，其基本思想就是将原始 DFG 中任一个节点的 J 次迭代交叉分配到新的相应 J 个节点上同时执行。展开算法只包括两个步骤：1) 节点的 J 倍复制；2) 边的连接和边延时的确定。边的连接和边的延时确定，是基于新旧 DFG 的节点间数据依赖关系的一致性。

题外话：这一节的收获，我觉得不是记住展开算法的两个步骤，因为记住这两个步骤是极其容易的；而把这两个步骤背后的意义弄明白才是关键。在边的连接和边的延时确定的证明中，我们是以一种调度的观点来看待 DSP 系统的，这种数据调度的观点在以后的章节中还会频繁使用，故而请大家仔细去体会。

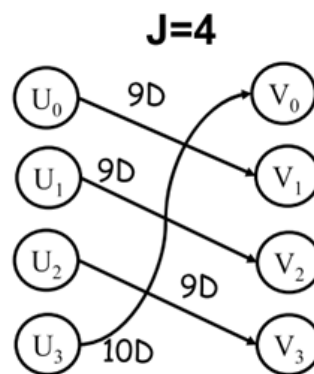
前一节介绍了展开的具体操作方法，只有两个步骤，非常的简洁，如下图所示。

7 展开的算法：J阶展开两步法

- For each node U in the original DFG, draw J nodes $U_0, U_1, U_2, \dots, U_{J-1}$



$$\left\lfloor \frac{(i+w)}{J} \right\rfloor = \left\lfloor \frac{(i+37)}{4} \right\rfloor = \begin{cases} 9, & i = 0, 1, 2 \\ 10, & i = 3 \end{cases}$$



- For each edge $U \rightarrow V$ with w delays in the original DFG, draw the J edges $U_i \rightarrow V_{(i+w)\%J}$ with $\left\lfloor \frac{(i+w)}{J} \right\rfloor$ delays for $i = 0, 1, \dots, J-1$

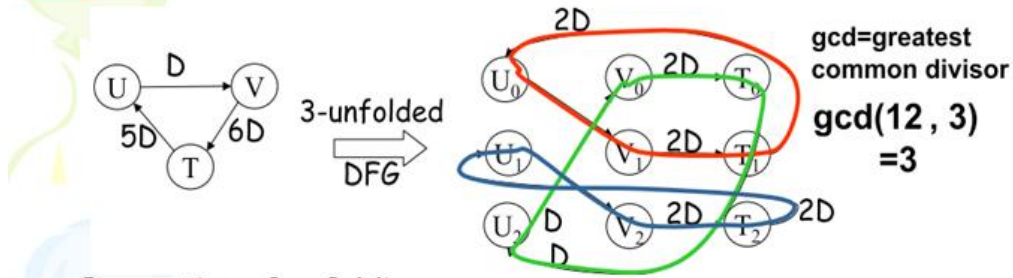
使用展开算法很容易将一个 DFG 进行任意阶展开。通过前一节的例子，我们也大致了解到展开前后 DFG 的区别，对于 J 阶展开：

1. 展开后 DFG 节点数量为展开前 DFG 节点数量的 J 倍。
2. 展开后 DFG 边数量为展开前 DFG 边数量的 J 倍。
3. 展开前后 DFG 的总延时数目不变。
4. 展开后 DFG 迭代边界为展开前 DFG 迭代边界的 J 倍，但采样边界不变(保持相等)。

以上这些认识都是通过观察得出，没有严格证明。这一节，将严格的把展开的最常用的性质列出并证明，同时也尽可能的去思索这些性质的物理意义，看看在硬件上到底对应怎么一回事。

讲解：第二节、展开的性质

展开的属性



Properties of unfolding :

- Unfolding preserves the number of delays in a DFG.

This can be stated as follows:

$$\lfloor w/J \rfloor + \lfloor (w+1)/J \rfloor + \dots + \lfloor (w+J-1)/J \rfloor = w$$

- J -unfolding of a loop l with w_l delays in the original DFG leads to $\gcd(w_l, J)$ loops in the unfolded DFG, and each of these $\gcd(w_l, J)$ loops contains $w_l / \gcd(w_l, J)$ delays and $J / \gcd(w_l, J)$ copies of each node that appears in l .
- Unfolding a DFG with iteration bound T_∞ results in a J -unfolded DFG with iteration bound JT_∞ .

性质一 展开算法保持 DFG 中的延时数目不变。

DFG 的延时数目，对应系统所使用的寄存器数量。展开算法的目的是构造多路并行的 DFG，以便提高系统的运行吞吐率。直观地想， J 路并行的系统，所消耗的资源（实现面积）应该是原系统的 J 倍，但其实不尽然，虽然节点和边的数量变为原来 J 倍，但延时数目，也就是寄存器数目却保持不变。

回顾展开算法的第二个步骤，原始 DFG 中 U 到 V 延时为 w 的一条边展开后得到 J 条对应边，且新边的延时为 $\lfloor (i+w)/J \rfloor$ ，其中 $i=0,1,\dots,J-1$ 。如此说来，只要新形成的 J 条边总延时等于原始边延时 w ，就说明展开不改变 DFG 的延时数目。对应延时为 w 的边的 J 条新边延时总和为

$$\lfloor w/J \rfloor + \lfloor (w+1)/J \rfloor + \dots + \lfloor (w+J-1)/J \rfloor$$

根据 $\lfloor (w+i)/J \rfloor, i=0,1,\dots,J-1$ 的大小，化简去掉 $\lfloor \cdot \rfloor$ 符号即可得证。

证明：令

$$w = nJ + m$$

其中

$$n = \lfloor w/J \rfloor, m = w \% J$$

显然 $n = 0, 1, \dots$ 且 $m = 0, 1, \dots, J-1$ 。将 w 公式带入

$$\lfloor w/J \rfloor + \lfloor (w+1)/J \rfloor + \dots + \lfloor (w+J-1)/J \rfloor$$

得

$$nJ + \lfloor m/J \rfloor + \dots + \lfloor (m+J-1)/J \rfloor$$

以下需证

$$nJ + \lfloor m/J \rfloor + \dots + \lfloor (m+J-1)/J \rfloor = w$$

1. 当 $m=0$ 时，显然成立。

2. 当 $1 \leq m \leq J-1$ 时，

因为 $\lfloor m/J \rfloor = 0$ ，又因为 $J \leq m+J-1 \leq 2J$ ，所以 $\lfloor (m+J-1)/J \rfloor = 1$ 。也就是说式子

$$\lfloor m/J \rfloor + \dots + \lfloor (m+J-1)/J \rfloor$$

前一部分项为 0，后一部分项为 1。那么那些项为 1 呢？如下示

$$\begin{array}{ccccccc} \lfloor m/J \rfloor & + \dots & \lfloor (m+k)/J \rfloor & + & \lfloor (m+k+1)/J \rfloor & \dots + & \lfloor (m+J-1)/J \rfloor \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ 0 & & 0 & & 1 & & 1 \end{array}$$

假设从第 $k+1$ 项开始为 1，则有 $m+k+1=J$ ，也就是 $k+1=J-m$ 。因为从 $k+1$ 到 $J-1$ 项均为 1，所以上式共有 $(J-1)-(k+1)+1=(J-1)-(J-m)+1=J-1-J+m+1=m$ ，即

$$\lfloor m/J \rfloor + \dots + \lfloor (m+J-1)/J \rfloor = m$$

综上所述，有

$$\lfloor w/J \rfloor + \lfloor (w+1)/J \rfloor + \dots + \lfloor (w+J-1)/J \rfloor = nJ + m = w$$

成立。

性质二 原始 DFG 中延时为 w_l 的环路 l 的 J 阶展开，得到展开后的 DFG 中的 $\gcd(w_l, J)$ 个环路， $\gcd(w_l, J)$ 个环路中的每个环路包含了 $w_l/\gcd(w_l, J)$ 个延时以及环路 l 中出现的每个节点的 $J/\gcd(w_l, J)$ 个拷贝。

这个性质的证明比较饶人，有兴趣的同志可以阅读书本上的证明，这里不去讲解。直观的理解，性质二告诉我们三个内容：

1. 一个环路展开会得到若干个环路，也就是说环路展开后得到的仍然是环路。
2. 原始环路的延时会平分到新的环路中。
3. 对于任一个原始节点 U 的新复制得到的 J 个节点也会平分到新的环路中。

对一个环路进行展开，可能得到一个新的环路，也可能得到多个 (>1) 新的环路，新环路的数量跟实际的待展开环路的总延时以及展开阶数有关。规定 $\gcd(w_l, J)$ 表示 w_l 和 J 的最大公约数。性质二定量地指出，延时为 w_l 的环路， J 阶展开会得到 $\gcd(w_l, J)$ 个新的环路，而且每个新环路分配所得的延时数目为 $w_l/\gcd(w_l, J)$ ，对于原始任一个节点 U 的 J 份复制也会平分到 $\gcd(w_l, J)$ 个环路中，也就是每个环路获得 U 的 $J/\gcd(w_l, J)$ 个拷贝。

性质三 展开一个迭代边界为 T_{\inf} 的 DFG，会得到一个迭代边界为 $J \cdot T_{\inf}$ 的 J 阶展开 DFG。

证明：假设原始 DFG 的迭代边界为

$$T_{\infty} = \max_l \left\{ \frac{t_l}{w_l} \right\}$$

根据性质二， J 阶展开所得新环路的计算时间为 $t_l \cdot J/\gcd(w_l, J)$ ，延时数目为 $w_l/\gcd(w_l, J)$ ，则新环路迭代边界为

$$T'_{\infty} = \max_l \left\{ \frac{t_l \cdot J/\gcd(w_l, J)}{w_l/\gcd(w_l, J)} \right\} = J \cdot \max_l \left\{ \frac{t_l}{w_l} \right\} = J \cdot T_{\infty}$$

注意，虽然展开后系统的迭代边界扩大 J 倍，但是由于新系统可以同时处理 J 个样本点，所以采样边界为 $J \cdot T_{\inf}/J = T_{\inf}$ ，也就是说展开不改变系统的极限吞吐率。

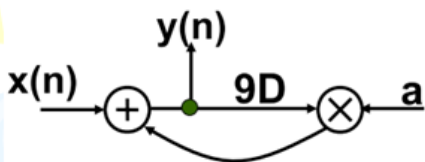
展开后的迭代边界，和采样边界

9

$$T_A=3, T_M=6$$

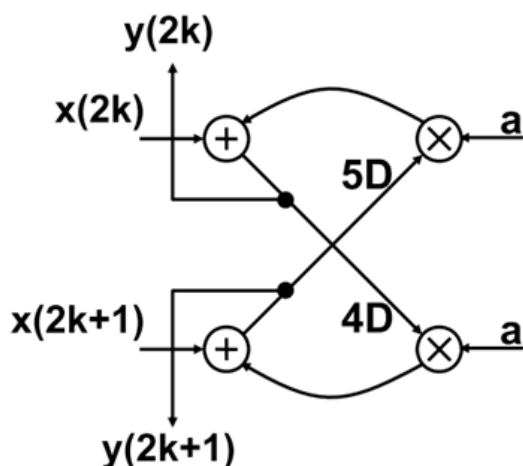
$$\gcd(9, 2) = 1 \Rightarrow 1 \text{ loop}$$

$$T_\infty = 18/9 = 2$$



$$T_\infty = 9/9 = 1$$

But we
process
2 samples

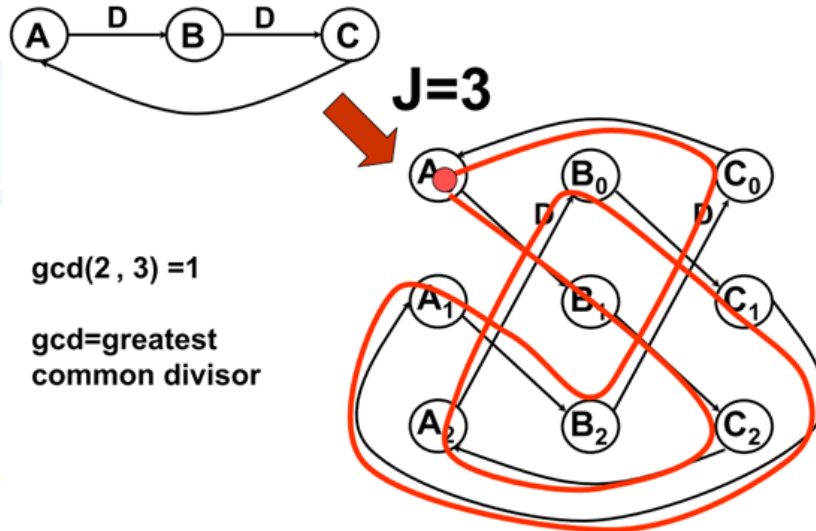


展开可能会导致更长的关键路径。容易犯的一个错误就是认为 J 阶展开就是原始系统的 J 倍加速，其实不尽然，而且也不见得展开阶数越大系统就越快。如果用系统实际吞吐率来衡量系统速度，那么展开阶数 J 与吞吐率一般不成正比。使用展开时，必须综合考虑展开所消耗的资源数量和所带来的系统加速，选择恰当的折中方案，此外不论采用多少阶的展开，系统的速度都不可能突破采样边界的限制。

为了弄清楚展开对关键路径长度的影响，看如下例子，

展开与关键路径

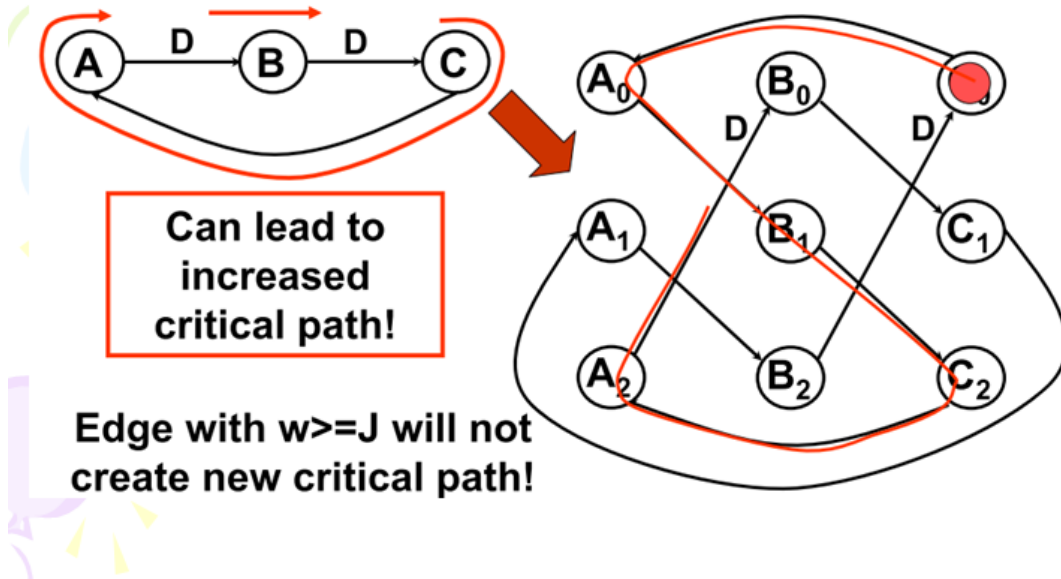
展开可能会导致关键路径的增大，问：
什么情况下，展开才不导致关键路径增大？



假设节点 A、B 和 C 的计算时间均为 $1u.t.$ ，展开前 DFG（左图）关键路径为 C-A，长度为 $2u.t.$ ，3 阶展开后 DFG（右图）关键路径为 $C_0-A_0-B_1-C_2-A_2$ ，长度为 $5u.t.$ 。很多场合关心的问题是：如何确定展开阶数才不会导致关键路径长度的增加呢？这里先给出答案，当 $w \geq J$ ，也就是展开阶数 J 不大于路径延时 w 时，将不会导致更长的关键路径出现。留个思考题：为什么 $w \geq J$ 就能保证不会出现更长的关键路径，当 $w < J$ 就一定会出现更长的关键路径吗？[提示，仔细体会展开步骤二中，旧路径延时 w 是如何在其对应的 J 条新边中进行分配的]

展开与关键路径

If edge with $w < J \Leftrightarrow (J-w)$ paths with zero delay and w paths with 1 delay



在重定时一章讲过，通过重定时可以改变系统的关键路径长度。由于展开可能导致更长的关键路径，结合重定时就有可能改善这一点。以下的性质和推论将有助于进一步理解展开对关键路径的作用以及展开和重定时之间的联系。

性质四 考虑原始 DFG 中延时为 w 的路径，当 $w < J$ 时，该路径的 J 阶展开将得到 $J-w$ 个无延时路径和 w 个延时为 1 的路径。

证明：

1. 当 $w=0$ 时，显然所有复制得到的 J 个路径延时均为 0，性质四成立。
2. 当 $J > w > 0$ 时， $\lfloor (w+J-1)/J \rfloor = 1$ 。 J 个路径的延时如下示，其中前一部分延时为 0，后一部分延时为 1，假设从 $\lfloor (w+k+1)/J \rfloor$ 开始及之后的项为 1，那么共有 $(J-1)-(k+1)+1$ 项为 1，因为 $w+k+1=J$ ，即 $k+1=J-w$ ，带入上式可知共有 $(J-1)-(J-w)+1=w$ 项为 1，同时也说明有 $J-w$ 项为 0。

$$\begin{array}{ccccccc}
 \lfloor w/J \rfloor & , \dots & \lfloor (w+k)/J \rfloor & , & \lfloor (w+k+1)/J \rfloor & \dots, & \lfloor (w+J-1)/J \rfloor \\
 \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 0 & & 0 & & 1 & & 1
 \end{array}$$

综上所述，性质四成立。

推论一 原始 DFG 中包含 J 个或更多延时的任何路径都能生成 J 个路径，其每个路径具有 1 个或更多延时。因此，原始 DFG 中具有 J 个或更多延时的路径不能生成一条在 J 阶展开 DFG 中的关键路径。

根据性质四，容易理解推论一。值得注意，关键路径定义为非零延时的最长路径，如果一条

边具有 1 个以上延时就不能是关键路径中的边，性质四给出了原始 DFG 中延时为 w 的边展开后得到新边中，延时为 0 的新边数目为 $J-w$ 条，如果不想这些边成为关键路径中的边，就必须限制 $J-w \leq 0$ ，也就是 $J \leq w$ 。

性质五 任何通过重定时 J 阶展开 DFG 得到的 G_J 所能得到的可行时钟周期，都可以通过直接对原始 DFG 的 G 重定时和以 J 为展开因子的展开来得到。

性质五指明了，先展开再重定时和先重定时再展开存在一一对应关系，如果使用先展开后重定时得到一个新的 DFG，那么这个 DFG 肯定可以通过某种重定时再展开来得到。具体证明就省略了，有兴趣的同志可以看书本。

通过前两节的学习，可以掌握如何对一个没有开关的 DFG 进行 J 阶展开。因为前面讲到展开的阶数和系统吞吐量往往不是正比关系，那么什么时候需要对 DFG 进行展开，而且展开阶数应该是多少才合理呢？这些都是实际应用前必须考虑的问题。这一节的内容不像前两节那么“死板”，蕴含很多乐趣。下面将通过一些例子来讲解如何使用展开，希望大家能够融会贯通，举一反三。从例子中也能进一步理解迭代边界和采样边界等概念。

讲解：第三节、展开的应用

12

展开的应用

采样周期的缩短

1. DFG 中存在某个节点计算时间大于 T_∞
2. 迭代边界 T_∞ 不是整数
3. 以上两者的混合，即最长的节点计算时间比 T_∞ 大，且 T_∞ 不是整数

并行处理

1. 字并行处理
2. 位并行处理（开关支路的展开）

这节主要练习展开的两个典型应用：采样周期的缩短和并行处理。值得注意的是，展开不等同于并行处理，它只是构造并行电路的其中一种方法。从迭代边界的理论中可知，一个 DFG 的实际运行速度往往没有达到其理论上可以到达的极限速度，系统的极限速度用迭代边界（理论最小时钟周期）或者采样边界（理论最大吞吐量）来衡量。一个给定的 DFG，不论

怎么改造，速度都不可能突破边界速度，所以我们不是无止境的追求提高速度，而是考虑“怎么改造 DFG，使其实际运行速度接近或等于边界速度”。

还记得本章一开始说过，通过展开，可以将一个 DFG 的实际运行速度提高到理论的极限速度，也就是以迭代边界作为实际运行周期，从而达到最大吞吐率。

那为什么说展开可以实现这一点呢？ 原来，原始 DFG 不能以边界速度运行，往往是因为

存在以下 2 个方面的限制因素（当然了，也许还存在其他限制因素，这里只讨论展开可以解决的限制因素）。如图 12 所示，制约系统不能以边界速度运行的 2 个因素是：

1. DFG 中存在某些节点（一个以上）计算时间大于迭代边界 T_{inf} 。
2. 迭代边界 T_{inf} 不是整数时间单位（本书中，假设任意节点的计算时间都是整数个单位时间）。

第一个限制因素容易理解，如果 DFG 中，存在一个节点的计算时间大于 T_{inf} ，那么该 DFG 不可能以迭代周期来运行，否则这个节点可能不能在一个周期内完成计算任务。由展开性质三[展开一个迭代边界为 T_{inf} 的 DFG，会得到一个迭代边界为 $J \cdot T_{inf}$ 的 J 阶展开 DFG]，通过展开增大新 DFG 的迭代边界，就可以去除这个限制。如果新 DFG 能以迭代边界为周期运行，虽然新 DFG 迭代边界比原始 DFG 迭代边界大，但是新 DFG 可以同时处理多个（J 个）样本，采样边界不变，也就是说新 DFG 吞吐率达到理论最大，这就是我们想要的。补充一句，新旧 DFG 的采样边界是相等的，采样边界决定系统极限吞吐率，所以不论是新 DFG 还是旧 DFG，只要以迭代边界为实际运行周期，不管迭代边界大小如何，都能达到极限吞吐率。

特定功能的 DFG，不论是否展开，其采样边界都是一个定值，也就是说理论最大吞吐率是一个定值。怎么知道一个 DFG 是否已经达到极限吞吐率呢？只需看 系统实际运行周期是否等于迭代周期。虽然 J 阶展开后 DFG 比原始 DFG 迭代边界增大，但只要新 DFG 是以其迭代边界（ $J \cdot T_{inf}$ ）为周期运行，就可以保证系统是以理论最大吞吐率 在工作。

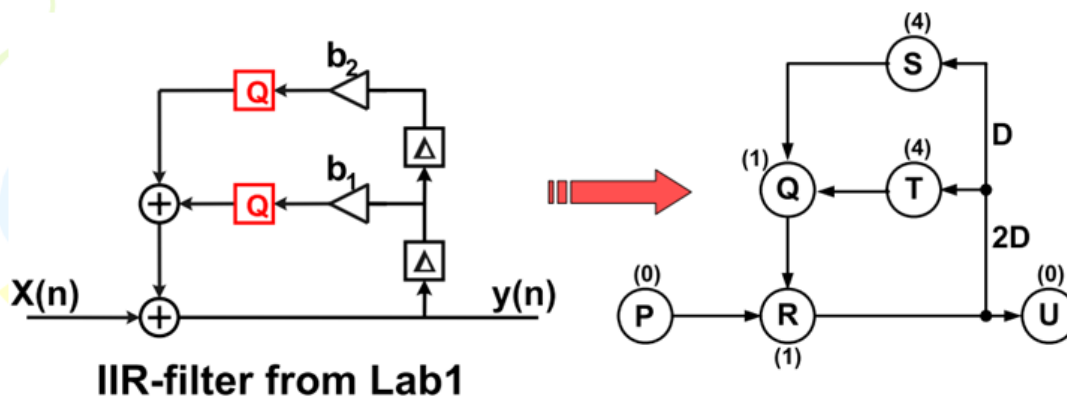
思考题：展开会消耗额外的资源，针对第一个限制因素，展开前后，以什么为代价换取了什么好处？

第二个限制因素中，首先假设 DFG 中任意节点的计算时间都是整数个单位时间。如果迭代边界不是整数，也就是说以带小数的单位时间为周期驱动系统，这个小数部分的单位时间，对任何一个节点来说都是不足以完成任何任务。因此，实际的运行时钟为整数个单位时间才“可能”是最合理的。

先看看第一种情况的例子，如图 13 所示，一个 IIR 滤波器，抽象出 DFG 如图 13 右图示，节点旁边小括号内数字为节点计算时间。

13

采样周期的缩短: case 1

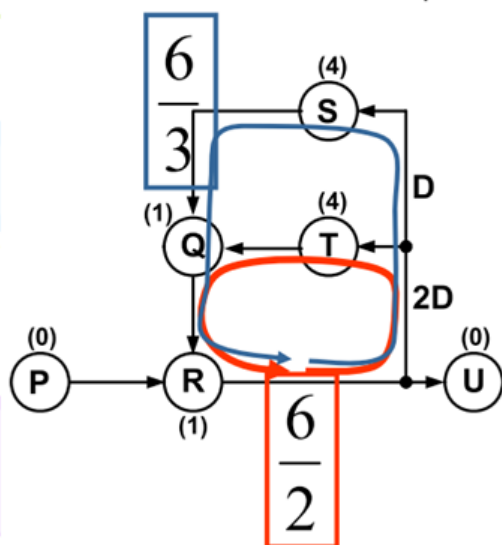


如图 14 所示，原始 DFG 存在两个环路，环路边界分别为 2u.t.和 3u.t.，最糟糕的环路边界也就是迭代边界为 3u.t.，而 DFG 中存在节点 S 和节点 T，计算时间为 4u.t.，所以不论如何重定时，该 DFG 都不能以小于 4u.t.的时间为周期来运行。对图 14 的 DFG 进行重定时，可以将关键路径长度缩小到 4u.t.，即使如此，该 DFG 也没到达理论最高吞吐量（3u.t.处理一个样本）。

14

采样周期的缩短: case 1

The original DFG cannot have sample period equal to the iteration bound because a node computation time is more than iteration bound



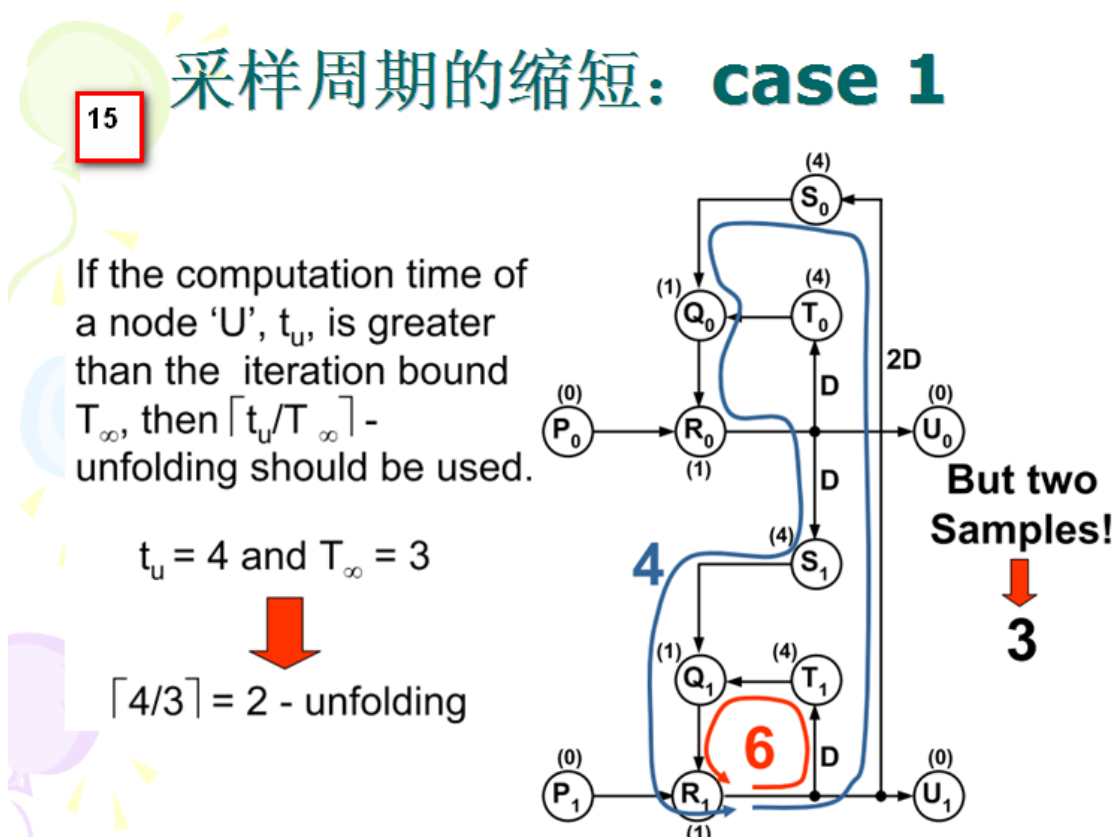
$$T_{\infty} = \max_{l \in L} \left\{ \frac{t_l}{w_l} \right\}$$

$$= \max_{l \in L} \left\{ \frac{6}{3}, \frac{6}{2} \right\} = 3$$

<4, max node time

怎么办呢？ 使用展开就能消除这个限制。比如对原始 DFG 进行 2 阶展开，则所得新 DFG 迭代边界变为 $2 * T_{inf} = 2 * 3 = 6u.t. > 4u.t.$ 。如图 15 所示，2 阶展开后的 DFG，最糟糕的环路用红色路线标识，迭代边界为 $6u.t.$ 。在新 DFG，所有节点的计算时间都小于迭代边界，也就是第一个限制因素已经解除。虽然新 DFG 迭代边界增大到 $6u.t.$ ，但是由于一个周期内可以处理两个样本，所以等价采样边界为 $3u.t.$ ，也就是每 $3u.t.$ 处理一个样本。对于原始 DFG，由于最快只能以 $4u.t.$ 为运行周期，所以等价于每 $4u.t.$ 处理一个样本，而在新 DFG 中却是每 $3u.t.$ 处理一个样本，新 DFG 吞吐率要大于原始 DFG 吞吐率。

正如一开始所说，展开可以解除制约系统达到最大吞吐率的因素。这个例子中只存在第一种限制因素，使用展开解除这个因素，就能使新 DFG 以迭代边界为实际运行周期运行，所以达到理论上的最大吞吐率。

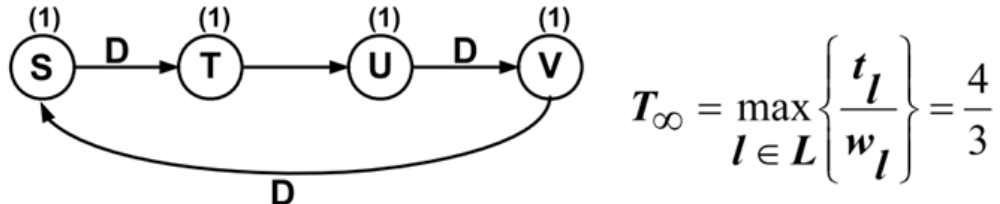


提醒一下爱浪费的同志，请以尽可能小的展开阶数实现解除限制因素的目标。如果 J 阶展开就能解除制约，那么就没有必要使用大于 J 阶的展开，要知道展开阶数越高消耗的资源就越多，但是系统吞吐率是不可能再提高的，这就是“边界”意义。因此，为了解除第一个限制因素，只需进行 $\lceil t_u / T_{inf} \rceil$ 阶展开即可，其中 t_u 为最耗时的节点计算时间， T_{inf} 为原始 DFG 的迭代边界， $\lceil \cdot \rceil$ 为向上取整。

接着是第二种限制因素：迭代边界不是整数个单位时间。如图 16 所示环路的迭代边界为 $4/3 = 1.33333$ 。本着节约的美德，要以最小的展开阶数展开该环路，以解除第二个限制因素，显然，3 阶展开即可。[思考题：为什么说 3 阶展开是解除第二个限制因素的最小展开阶数？提示，展开性质三]

采样周期的缩短: case 2

The original DFG cannot have sample period equal to the iteration bound because the iteration bound is not an integer



If a critical loop bound is of the form t_l/w_l where t_l and w_l are mutually co-prime, then w_l -unfolding should be used.

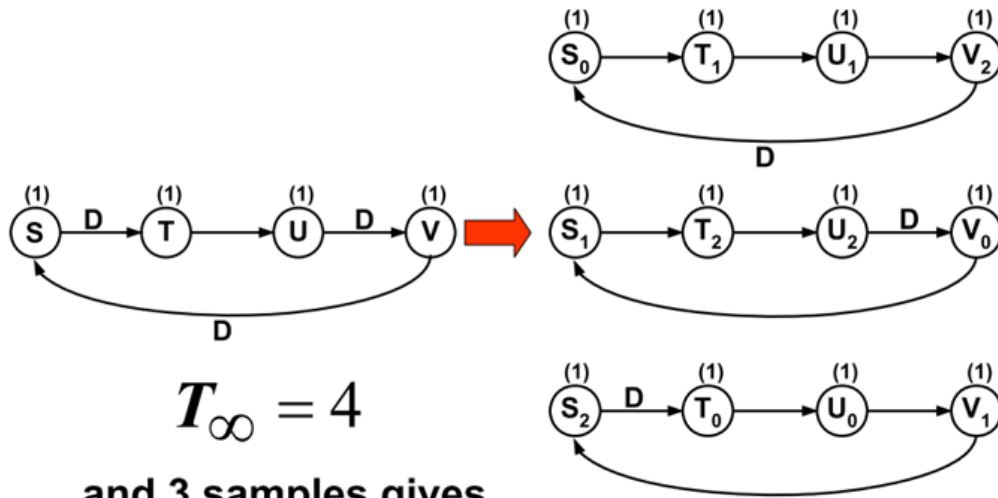


Unfolding of 3

3 阶展开后所得 DFG 如图 17 所示。这个例子涉及到环路的展开, 还记得展开性质二不? [性质二 原始 DFG 中延时为 w_l 的环路 l 的 J 阶展开, 得到展开后的 DFG 中的 $\gcd(w_l, J)$ 个环路, $\gcd(w_l, J)$ 个环路中的每个环路包含了 $w_l/\gcd(w_l, J)$ 个延时以及环路 l 中出现的每个节点的 $J/\gcd(w_l, J)$ 个拷贝。] 下面, 先来验证性质二, 原始环路延时总数为 3, 展开阶数为 3, 则应该得到 $\gcd(w_l, J) = \gcd(3, 3) = 3$ 个新的环路, 新环路的每个环路均分得延时数目为 $w_l/\gcd(w_l, J) = 3/3 = 1$ 个, 每个环路包含任一原始节点的 $J/\gcd(w_l, J) = 3/3 = 1$ 分拷贝。图 17 的结果完全符合性质二的结论。

17

采样周期的缩短: case 2



and 3 samples gives
minimum sample period $4/3$

新的三个环路迭代边界均为 $3 \cdot T_{\text{inf}} = 3 \cdot 4/3 = 4 \text{u.t.}$ 。实际运行周期也可取 4u.t. ，所以新 DFG 的吞吐率可以到达理论最大值，等价于每 $4/3 \text{u.t.}$ 处理一个样本。

有时，DFG 中会同时出现上面所提的两种限制因素。同样的道理，本着节约的美德，应该以最小的展开阶数展开，以同时解除这两个限制因素。

18

采样周期的缩短: case 3 = case 1 + case 2

When the longest node computation time $t_{U,\max}$ is greater than T_∞ , and T_∞ is not an integer

- The minimum unfolding factor **J** that allows the **iteration period** to equal the **iteration bound** can be determined by the following equation

$$J \cdot T_\infty \geq t_{U,\max}$$

- Ex: Assume $T_\infty = 4/3$ and $t_{U,\max} = 6$

- Sol:

$$J \cdot \frac{4}{3} \geq 6 \Rightarrow J = 6$$

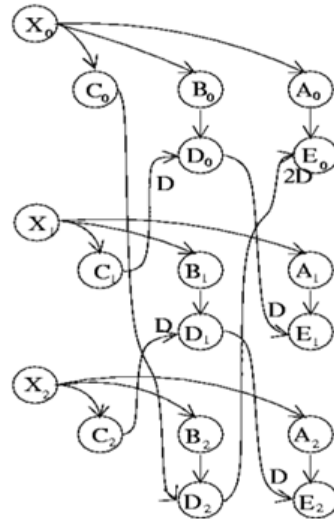
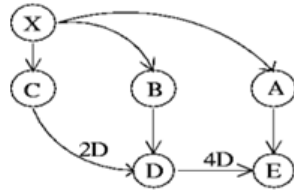
思考题：如何计算解除以上所说三种限制因素的最小展开阶数？

下面再来讨论展开与并行处理的关系。前面简单说到，展开是构造并行处理电路的方法之一，其实不难理解，假设原始 DFG 一个周期只处理一个样本，则 J 阶展开的 DFG 一个周期可以并行处理 J 个样本。这里值得注意的是，此周期非彼周期，展开后的关键路径往往比原关键路径长，以至于展开后 DFG 的运行周期要比原始 DFG 运行周期大，虽然如此，展开还是有可能进一步提高系统实际吞吐率。

字级并行处理，指的是以字为处理单位进行展开。比如图 19 中，假设输入 x 就是一个 w 位的字，在字的层次对图 19 左图 DFG 进行 J 阶展开 (J=3)，得到一个字级 3 路并行处理 DFG 如右图示。[这个比较简单，大家看看书上例子即可]

并行处理：字级并行处理

Another FIR-filter, $J=3$

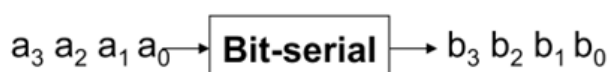
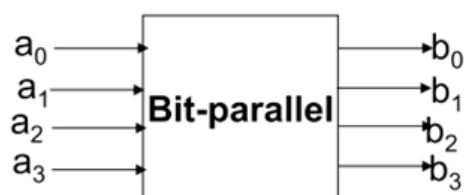


下面的位级并行处理例子，目的是表明如何利用展开变换，由位串行架构导出位并行和数字串行架构。

原始 DFG 进行的是“位串行”处理，一个时钟周期只处理一位。假设字长为 w 位，那么处理一个字就需要 w 个时钟周期。如果要进行位并行处理，也就是一个时钟周期处理一个 w 位的字，只需“直接”进行 w 阶展开即可[这里不再多说]。

20

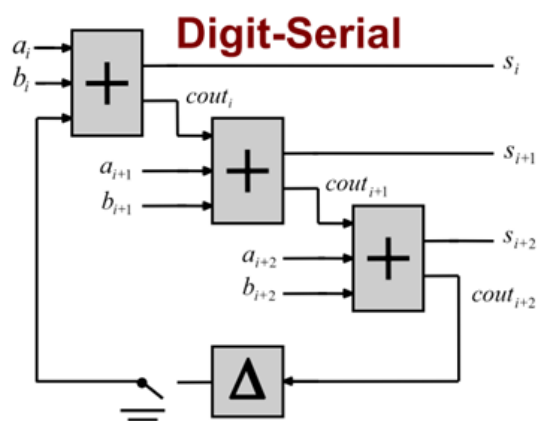
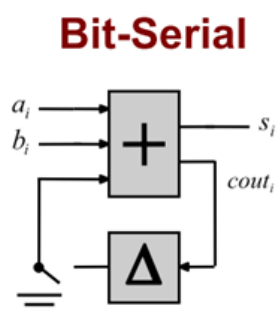
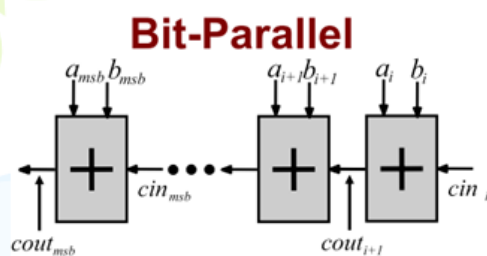
并行处理：位级并行处理



数字串行处理，其实是位串行处理和位并行处理的折中，其实就是阶数小于 w 的展开，比如要求一个时钟周期处理 N 位（ N 称为数字尺寸），一个 w 位的字就需要 w/N 个周期进行处理。

21

并行处理：数字串行处理



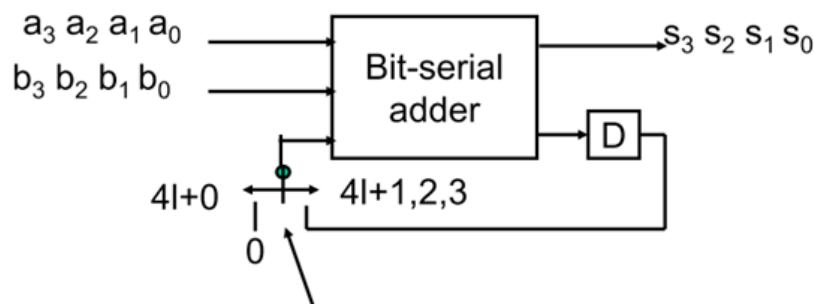
以上所说没什么特别的地方，就是正常的展开而已啊！其实位级展开真正的问题是，开关之路如何展开？以位串行加法器的展开为例，如图 22 所示。

22

新问题来了！

位串行加法器中的开关如何展开

Bit-serial can be seen as a time-multiplexed architecture, in this example on addition (i.e. 1 iteration) takes 4cc.



Switch for carry signal

How to unfold switches?

如图 22 所示，一个 4 位的串行加法器，输入 A 和 B，以及输出 S 为 4 位的字，系统从低位（LSB）一位一位往高位（MSB）处理，如果低位产生进位，先保持到寄存器 D 中，下一周期加到相邻的高位上。图中大方框内的 Bit-serial adder 其实就是一个全加器，两个输入端和一个进位输入端，一个输出端和一个进位输出端。开始计算，输入 a_0 和 b_0 ，进位输入为 0，输出 s_0 ，并将进位输出保持到 D 中；接下来，输入 a_1 和 b_1 ，进位输入为 D 中保持的值，输出为 s_1 ，且进位输出又保持到 D 中；等等。全加器的进位输入端在每一个 $4 \cdot l + 0$ 周期总是输入 0 值，而在 $4 \cdot l + 1, +2, +3$ 周期输入寄存器 D 中的值。对该位串行 DFG 进行展开就能得到位并行或者数字串行系统，但是前面我们并未讨论如何展开一个带开关的边，应该怎么办呢？以下内容就是教会您如何展开开关支路。

开关支路的展开

The following assumptions are made when unfolding an edge $U \rightarrow V$ containing a switch :

- The wordlength W is a multiple of the unfolding factor J , i.e. $W = W'J$.
- All edges into and out of the switch have no delays.

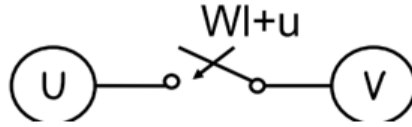
With the above two assumptions an edge $U \rightarrow V$ can be unfolded as follows :

- Write the switching instance as

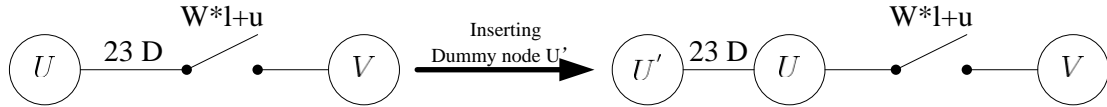
$$Wl + u = J(W'l + \lfloor u/J \rfloor) + (u \% J)$$

- Draw an edge from the node $U_{u \% J} \Leftrightarrow V_{u \% J}$,

which is switched at time instance $(W'l + \lfloor u/J \rfloor)$.



如图 23 示的开关支路，其中 $Wl+u$ 表明在第 $Wl+u$ 次迭代开关闭合，其他次迭代开关断开， W 为字长， $l \in \mathbb{Z}^+ \cup \{0\}$ ， $u=0,1,\dots,W-1$ 。如果开关支路包含延时，可以插入一个哑巴节点从而改造成没有延时的开关支路，如下图所示， U' 和 V 之间的支路就符合通用形式。



对此开关支路进行 J 阶展开，类似一般的展开步骤，先进行 U 和 V 的 J 倍复制，得到 U_0, U_1, \dots, U_{J-1} 和 V_0, V_1, \dots, V_{J-1} ；接下来要建立 U_i 和 V_i 之间的连接关系，分析如下：

原始开关支路的意义是，在第 $Wl+u$ 次迭代，开关闭合，其他次迭代开关断开。现在要弄清楚的是第 $Wl+u$ 次迭代到底调度到那个新节点上运行，那个节点应该在对应时刻进行闭合。假设调度到第 i 节点上运行，也就是说第 i 个节点所执行的迭代 $J \cdot k + i$ 等于 $Wl+u$ ，即有如下关系

$$kJ + i \sim Wl + u \Rightarrow k = \lfloor (Wl + u) / J \rfloor \wedge i = (Wl + u) \% J$$

也就是说，新 DFG 中 U_i 和 V_i 在第 k 次迭代相连，其他次迭代断开。课本上讨论了一种特殊的情况，即 限制字长 W 为 J 的整数倍。假设 $W=W'J$ ， W' 为整数，带入上式并化简，有

$$k = \lfloor (Wl + u) / J \rfloor = \lfloor (W'J \cdot l + u) / J \rfloor = \lfloor W'l + u/J \rfloor = W'l + \lfloor u/J \rfloor$$

$$i = (Wl + u) \% J = (W'J \cdot l + u) \% J = u \% J$$

完整表述为：原始 DFG 中节点对 $U-V$ 间在第 $Wl+u$

次迭代闭合的开关支路，对应于新 DFG 中，节点对 U_i -- V_i 在第 k 次迭代闭合的开关支路。

当满足如下两个基本假设：

1. 字长 W 是展开因子 J 的倍数，即 $W=W'J$ 。
2. 所有进出开关的边都不包含延时。

基于以上两个假设，开关展开的步骤完整描述为：

1. 将闭合时刻写为

$$Wl + u = J \left(W'l + \lfloor u/J \rfloor \right) + (u \% J)$$

2. 展开后的图中，从节点 $U_{\{u\%J\}}$ 到节点 $V_{\{u\%J\}}$ 画一条不带延时的边，它在 $W'l + \lfloor u/J \rfloor$ 时刻闭合。

下面看一个例子，如图 24 的开关支路，进行 3 阶展开。注意字长 $W=9$ ，恰为展开阶数 $J=3$ 的 3 倍，所以 $W'=W/J=9/3=3$ 。根据开关展开步骤 1，可将闭合时刻 $9*1+1$ 和 $9*1+5$ 表示为规定形式，如图 24 底部公式。

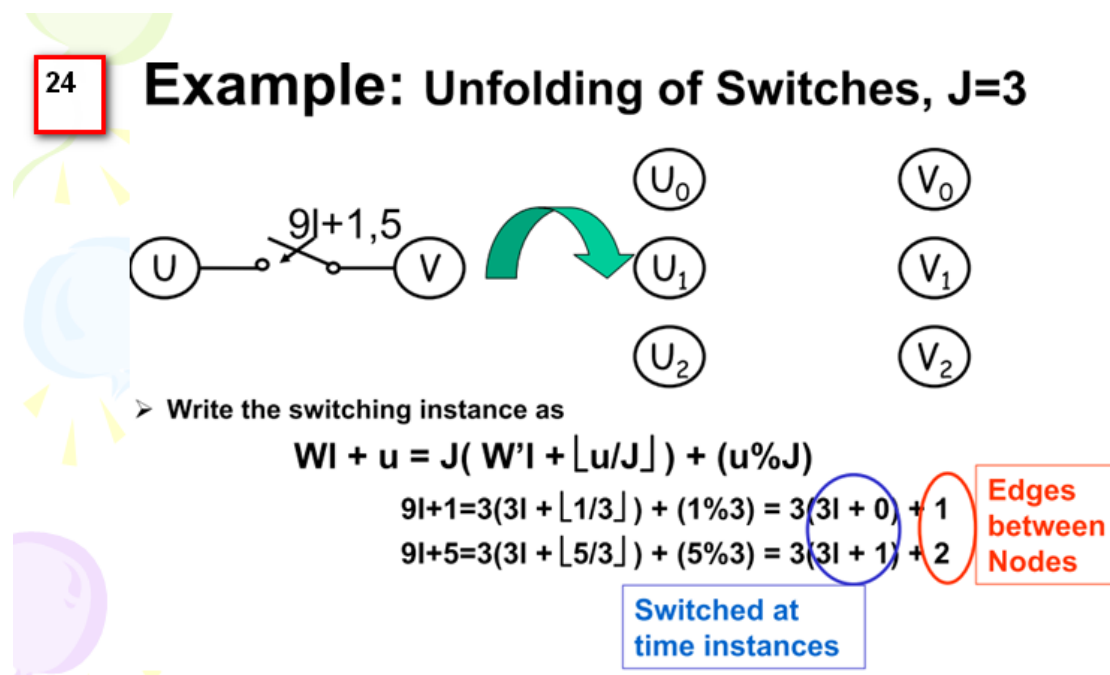
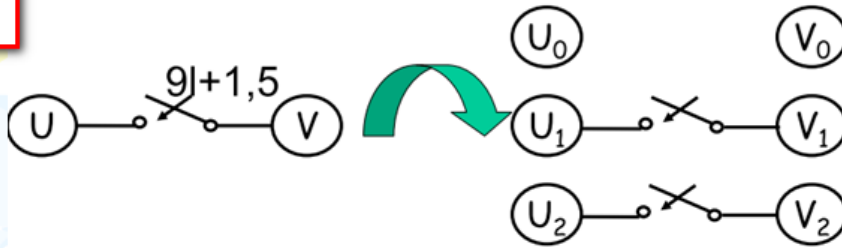


图 24 底部公式表明，新 DFG 的节点对 U_1 -- V_1 和 U_2 -- V_2 存在互联关系，其他节点对不存在连接关系，如图 25 所示。

Example: Unfolding of Switches, J=3

25



➤ Write the switching instance as

$$Wl + u = J(W'l + \lfloor u/J \rfloor) + (u \% J)$$

$$9l+1=3(3l + \lfloor 1/3 \rfloor) + (1 \% 3) = 3(3l + 0) + 1$$

$$9l+5=3(3l + \lfloor 5/3 \rfloor) + (5 \% 3) = 3(3l + 1) + 2$$

Edges
between
Nodes

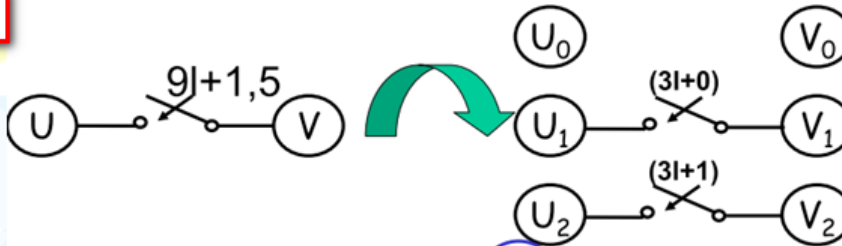
➤ Draw an edge from the node $U_{u \% J} \Rightarrow V_{u \% J}$, i.e.

$$U_1 \Rightarrow V_1 \text{ and } U_2 \Rightarrow V_2$$

以上所确定的两组节点对分别在 $3l+0$ 和 $3l+1$ 时刻闭合，如图 26 所示。

Example: Unfolding of Switches, J=3

26



$$9l+1=3(3l + \lfloor 1/3 \rfloor) + (1 \% 3) = 3(3l + 0) + 1$$

$$9l+5=3(3l + \lfloor 5/3 \rfloor) + (5 \% 3) = 3(3l + 1) + 2$$

Switched at
time instances

switched at time instance $(W'l + \lfloor u/J \rfloor)$, i.e.

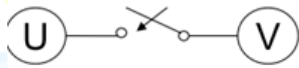
$$U_1 \Rightarrow V_1 \text{ at } (3l+0) \text{ and } U_2 \Rightarrow V_2 \text{ at } (3l+1)$$

练习：请根据开关展开步骤，3 阶展开图 27 所示开关支路。答案如图 28 所示。

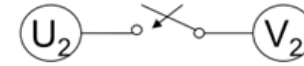
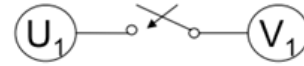
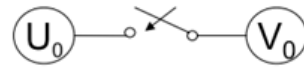
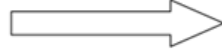
27

多开关展开：示例

12/ + 1, 7, 9, 11



Unfolding by 3



$$Wl + u = J(W'l + \lfloor u/J \rfloor) + (u \% J)$$

To unfold the DFG by $J=3$, the switching instances are as follows

$$12/ + 1 = 3(4/ + 0) + 1$$

$$12/ + 7 = 3(4/ + 2) + 1$$

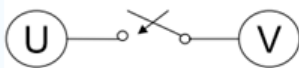
$$12/ + 9 = 3(4/ + 3) + 0$$

$$12/ + 11 = 3(4/ + 3) + 2$$

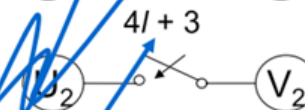
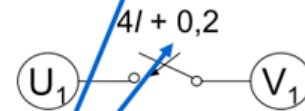
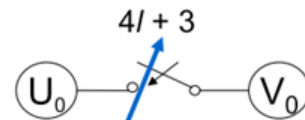
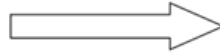
28

多开关展开：示例

12/ + 1, 7, 9, 11



Unfolding by 3



$$Wl + u = J(W'l + \lfloor u/J \rfloor) + (u \% J)$$

Switched at time instances

$$12/ + 1 = 3(4/ + 0) + 1$$

$$12/ + 7 = 3(4/ + 2) + 1$$

$$12/ + 9 = 3(4/ + 3) + 0$$

$$12/ + 11 = 3(4/ + 3) + 2$$

至此，本章的主要内容就讲完了。不论是延时支路的展开还是开关支路的展开，操作都非常简单，相信结合例子，大家都能正确的将一个 DFG 进行正确展开。但是仅仅知道如何展开一个 DFG 并不够，真正理解展开的意义更为重要。这一章令我兴奋的是，从计算调度的角度可以推导出展开的算法及其性质，也就是说展开其实就是一种计算调度方法。不知这样理

解是否恰当？在以后的两章中，还会使用计算调度的思想来进行折叠和脉动的学习。

29

你来想想：思考题！

If Wordlength is not a multiple of J

- **determine $lcm\{W, J\}$, lcm = least common multiple**
- **replace switching instance $Wl+u$ with L/W instances**
 $Ll+u+wW, \text{ for } w=0..L/W-1$
i.e. the switchingperiodicity has been changed
from W to L
- **perform the unfolding as previously**
- **identify the correspondence between original instances and expanded instances**