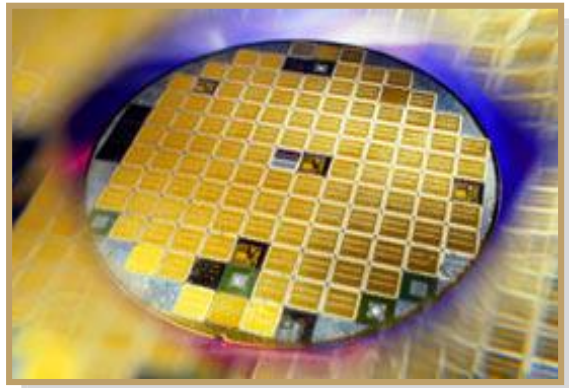
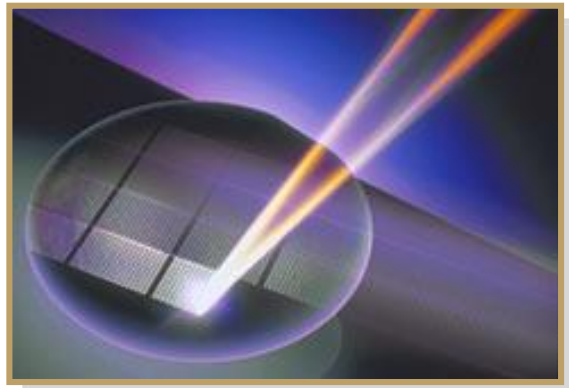
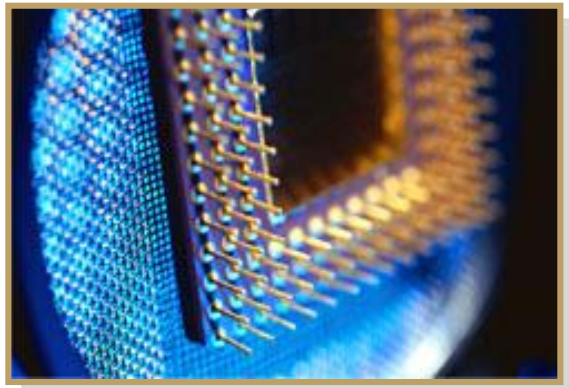
第四章：重定时技术

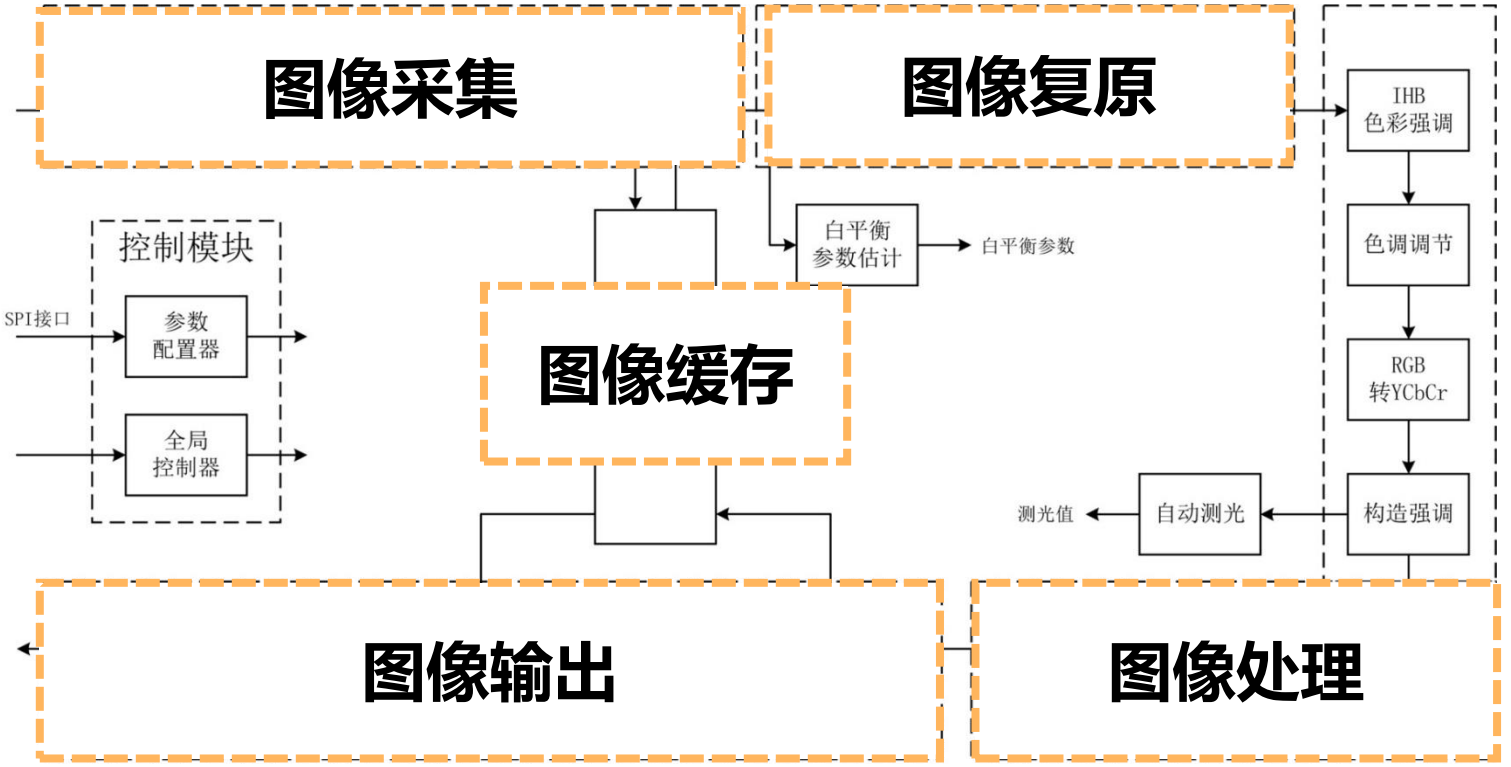




图像处理系统

监控摄像头

相机

医疗内窥镜

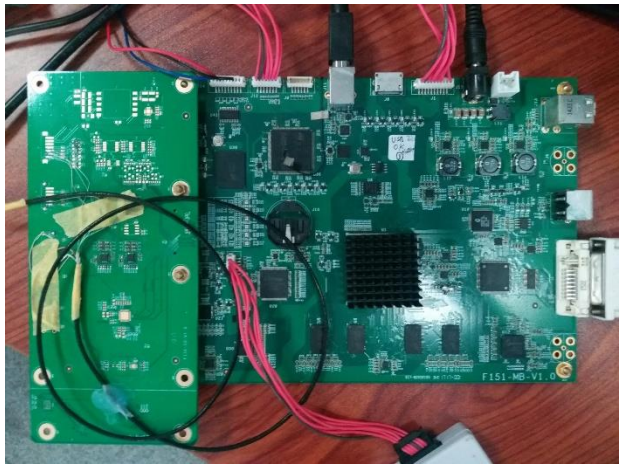


图像信号处理电路

电路工作频率低带来的问题

- 图像信号处理电路目标工作频率150MHz
- FPGA电路设计完成后，图像处理单元工作频率145MHz

如何将频率从145MHz提升至150MHz?

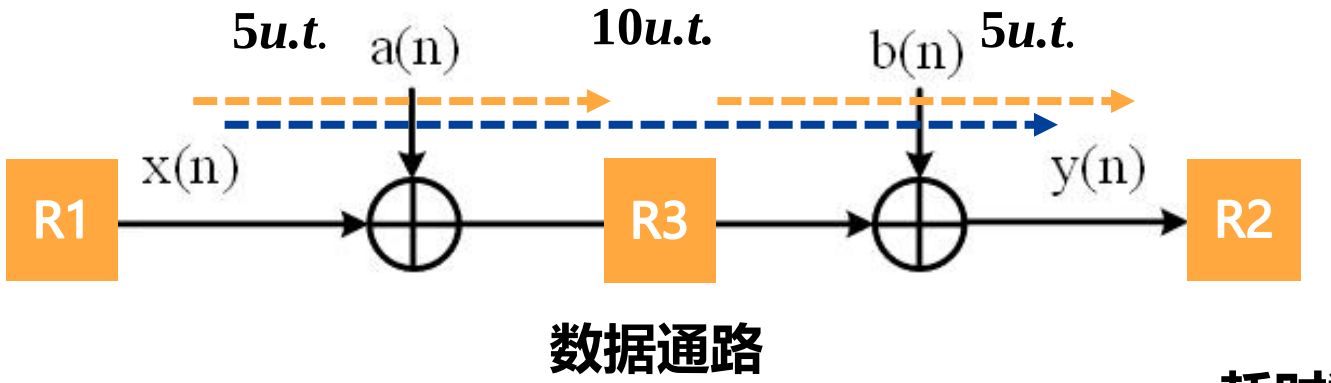


FPGA图像处理电路

如何提高电路工作频率

流水线技术： 提升时钟频率

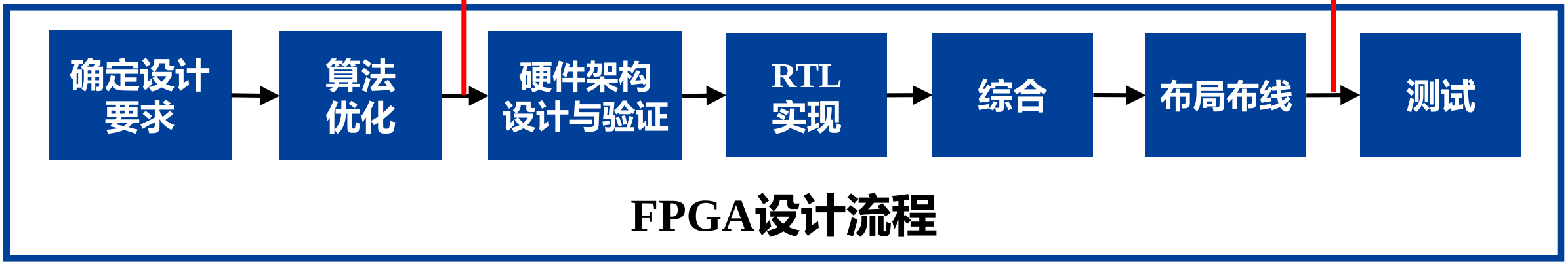
有没有能显著缩短设计周期方法?



数据通路

重定时

耗时数周

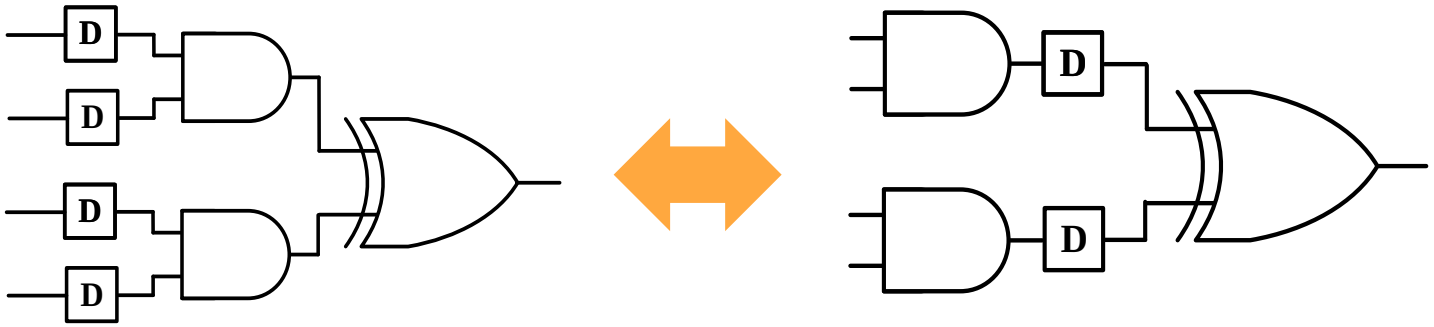


FPGA设计流程

目录

01. 重定时的基本概念——定义

重定时(Retiming)： **是一种变换技术，在不改变系统的输入输出特性的前提下，改变电路延迟元件的配置**



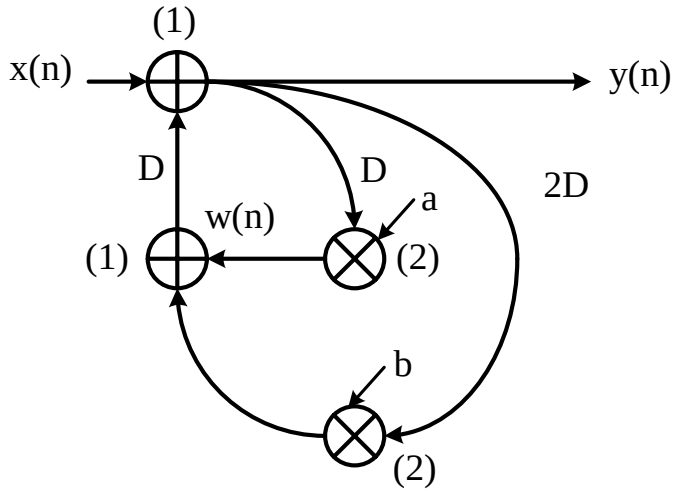
延时单元可以在所有输出与所有输入之间移动

- **减少关键路径**
- **减少寄存器数量**

01. 重定时的基本概念——功能不变

原始DFG

关键路径：1 乘1加



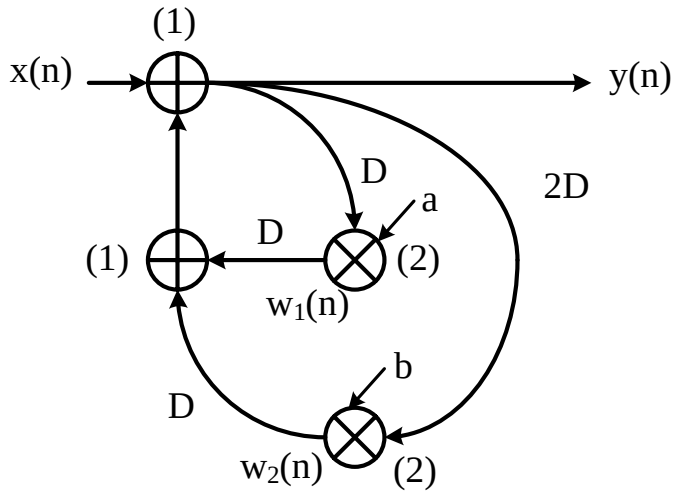
$$w(n) = ay(n-1) + by(n-2)$$

$$y(n) = w(n-1)$$

$$= ay(n-2) + by(n-3) + x(n)$$

一种重定时DFG

关键路径：2加



$$w_1(n) = ay(n-1)$$

$$w_2(n) = by(n-2)$$

$$y(n) = w_1(n-1) + w_2(n-1) + x(n)$$

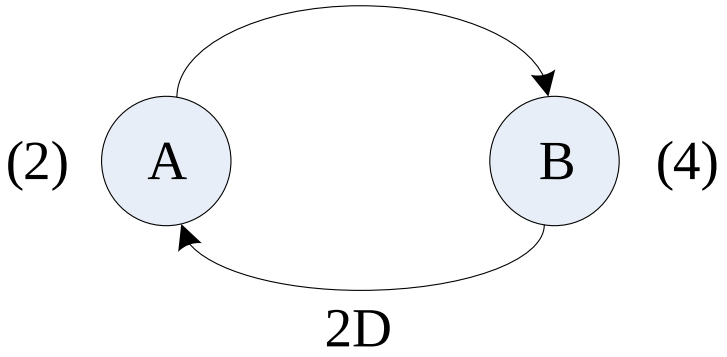
$$= ay(n-2) + by(n-3) + x(n)$$

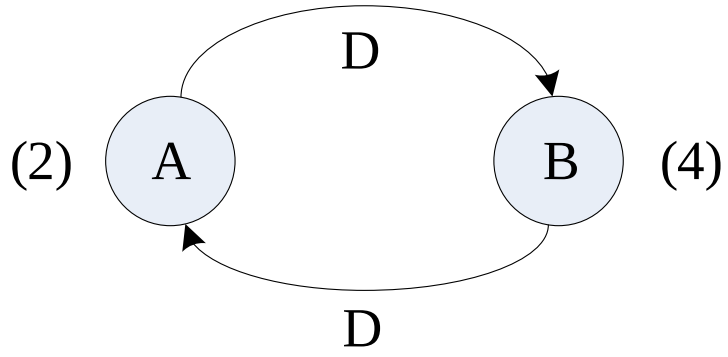
01. 重定时的基本概念——注意点

重定时并不改变：

- 环路中延迟
- 迭代边界

…但是关键路径！





迭代边界=6/2=3

关键路径=6

迭代边界 =6/2=3

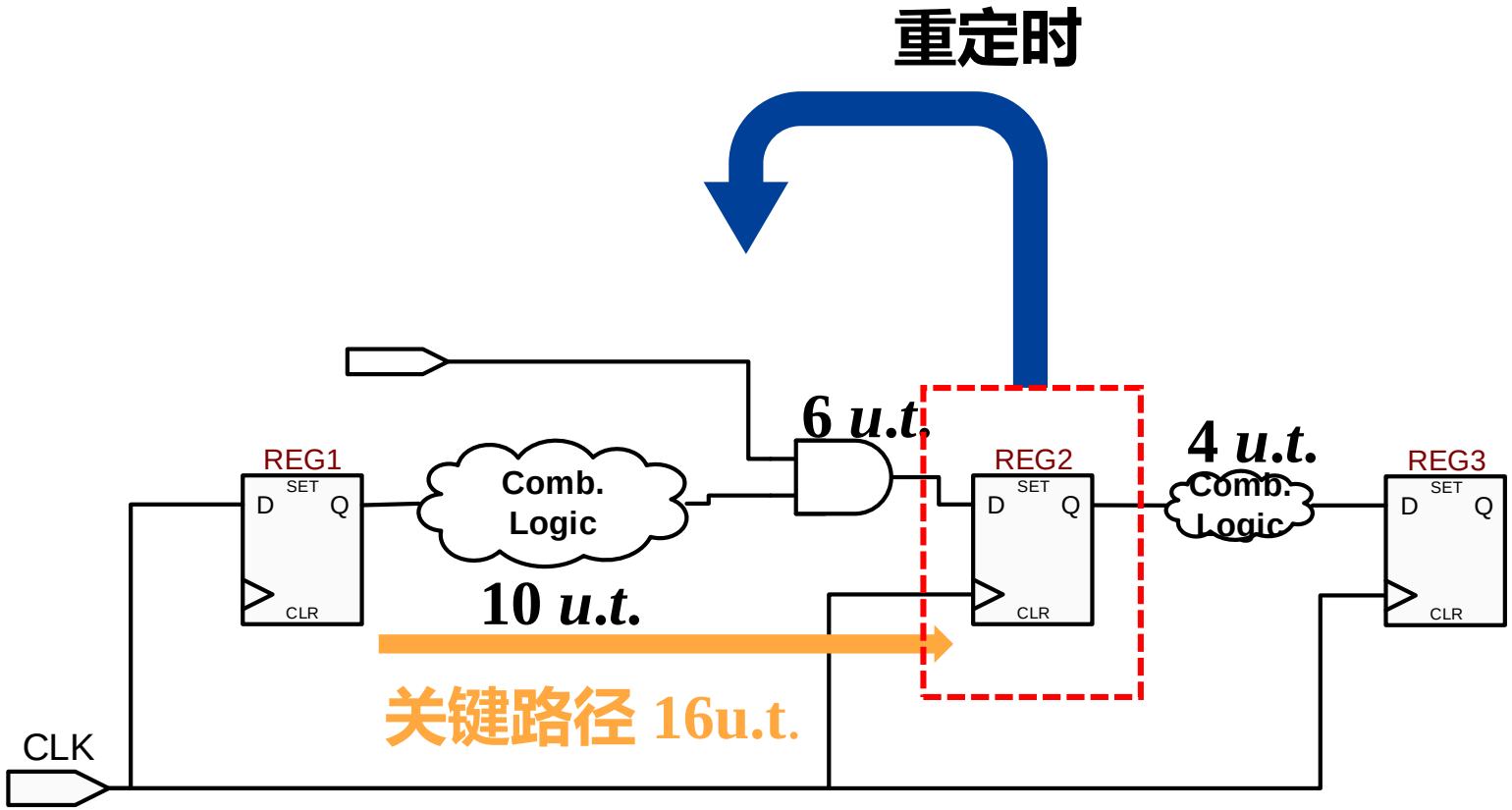
关键路径=4

01. 重定时的基本概念——示例

重定时前

关键路径： 16*u.t.*

寄存器　： 3



01. 重定时的基本概念——示例

重定时前

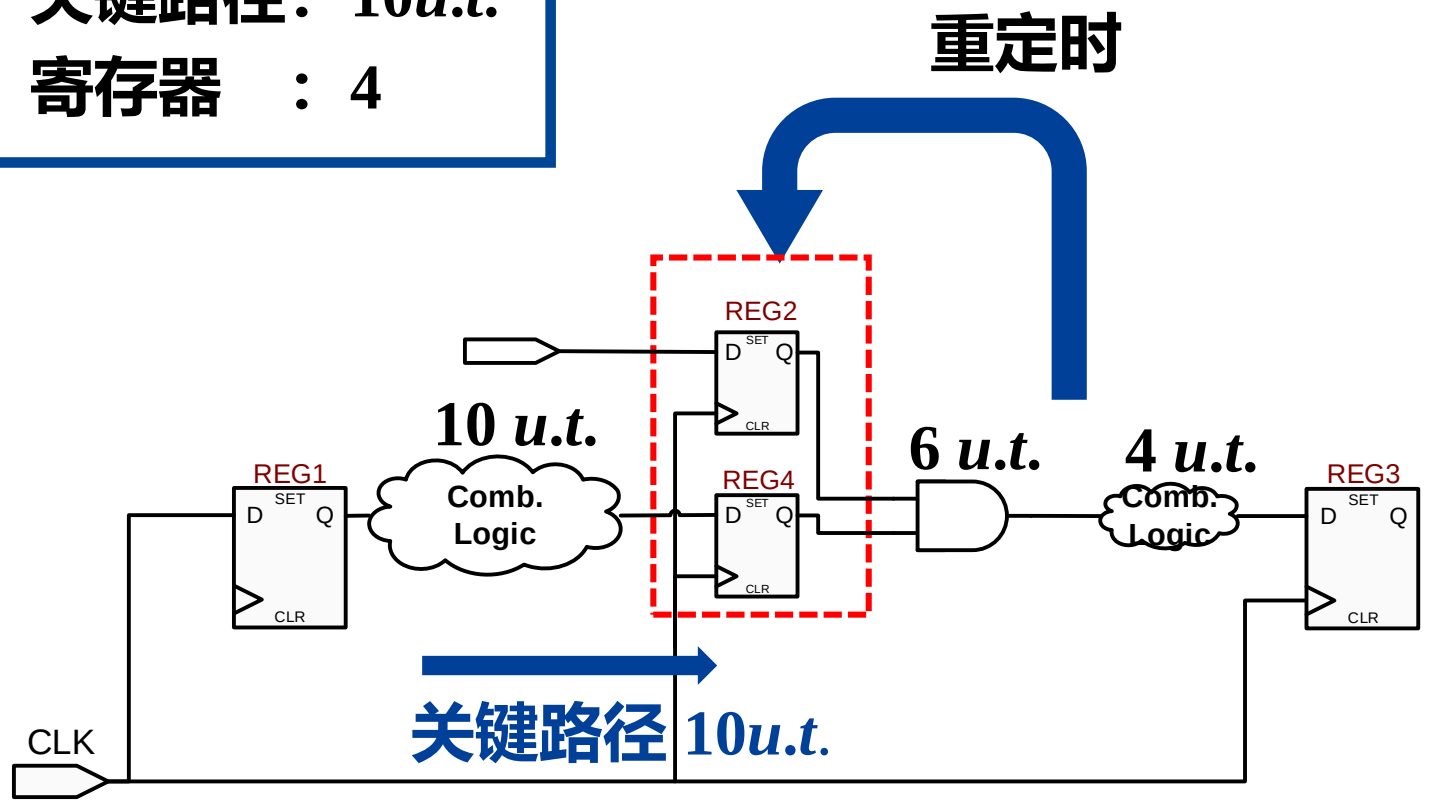
关键路径：16*u.t.*

寄存器　：3

重定时后

关键路径：10*u.t.*

寄存器　：4

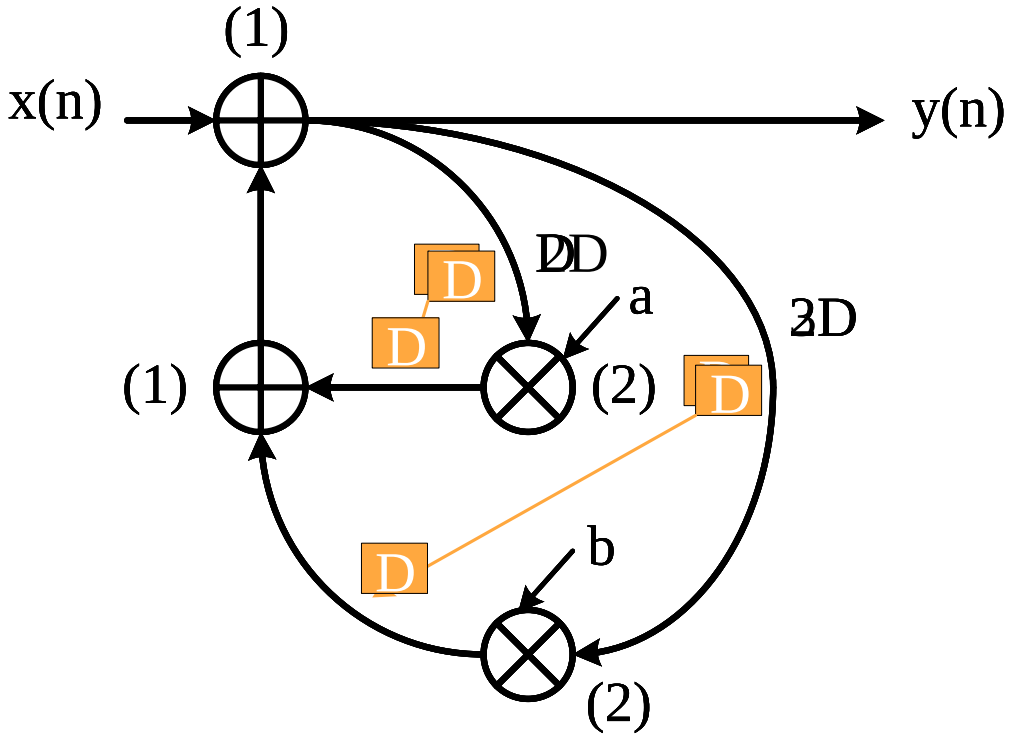


重定时

- 时钟频率提高
- 改变寄存器数目

01. 重定时的基本概念——两种目的

1. 加快速度：



关键路径

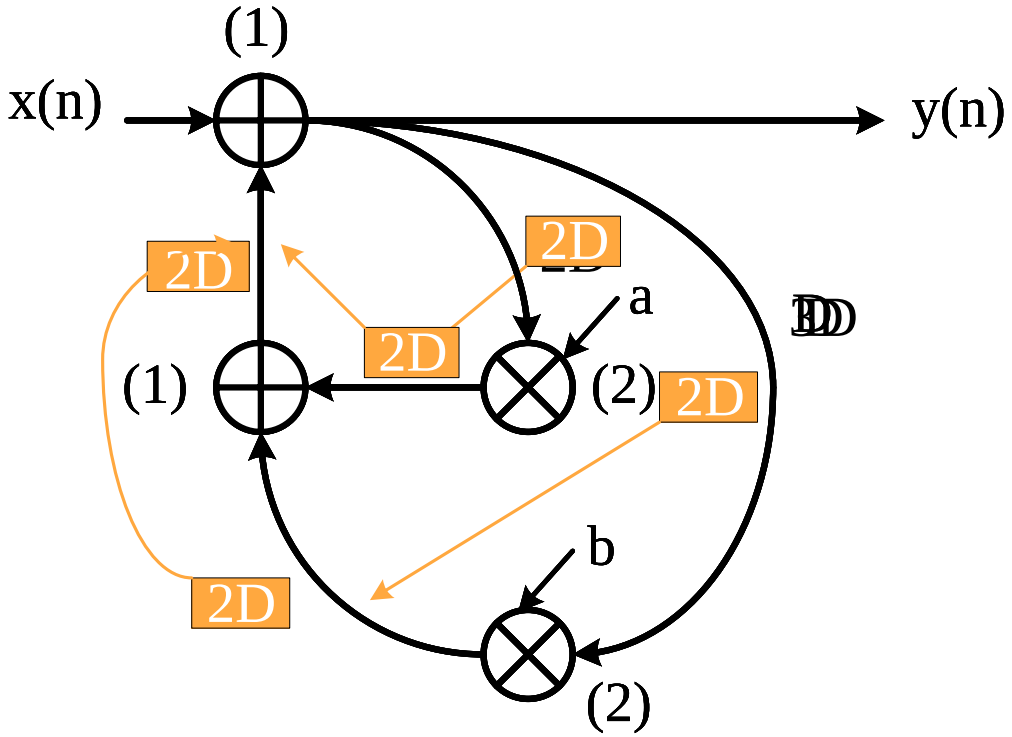
$$T_{critical} = 2T_A + T_M$$

重定时后关键路径

$$T_{critical} = 2T_A$$

01. 重定时的基本概念——两种目的

2. 减小面积：



寄存器个数

延迟单元：5个

重定时后寄存器个数

延迟单元：3个

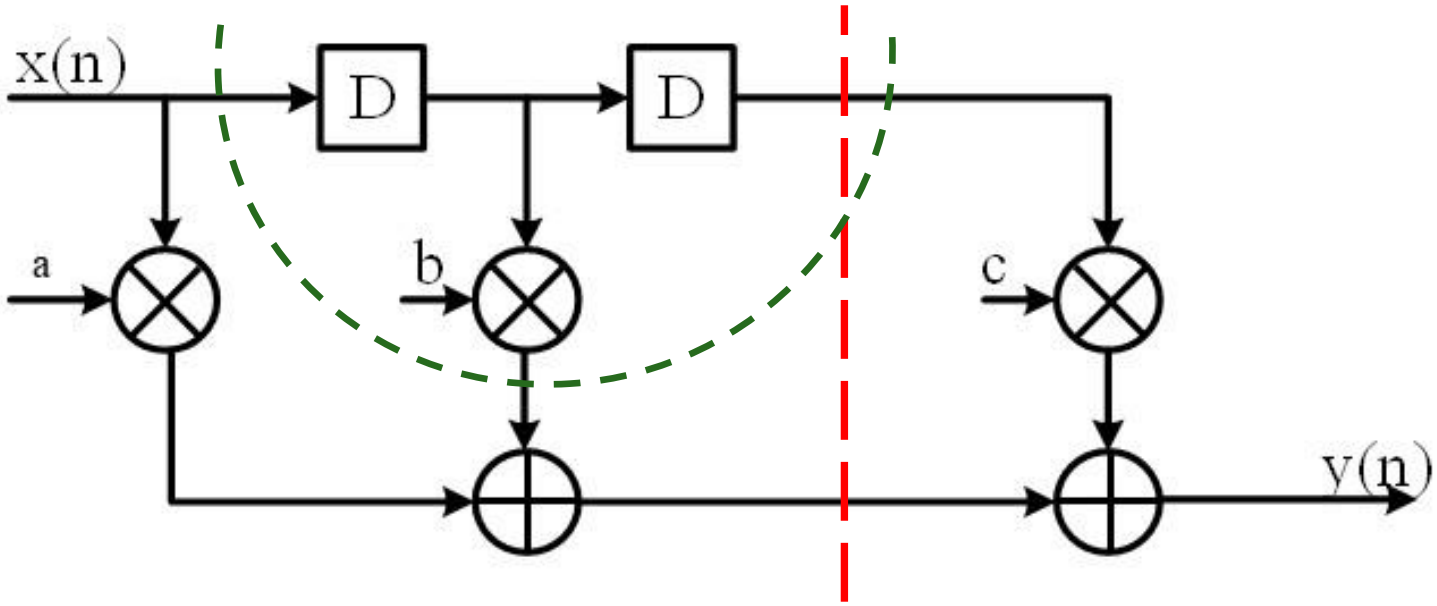
目录

02. 重定时方法

- 最基本的重定时是割集重定时
- 割集重定时方法：
  - 在一个方向的边上增加延时
  - 在另外方向的边上减少同样的延时

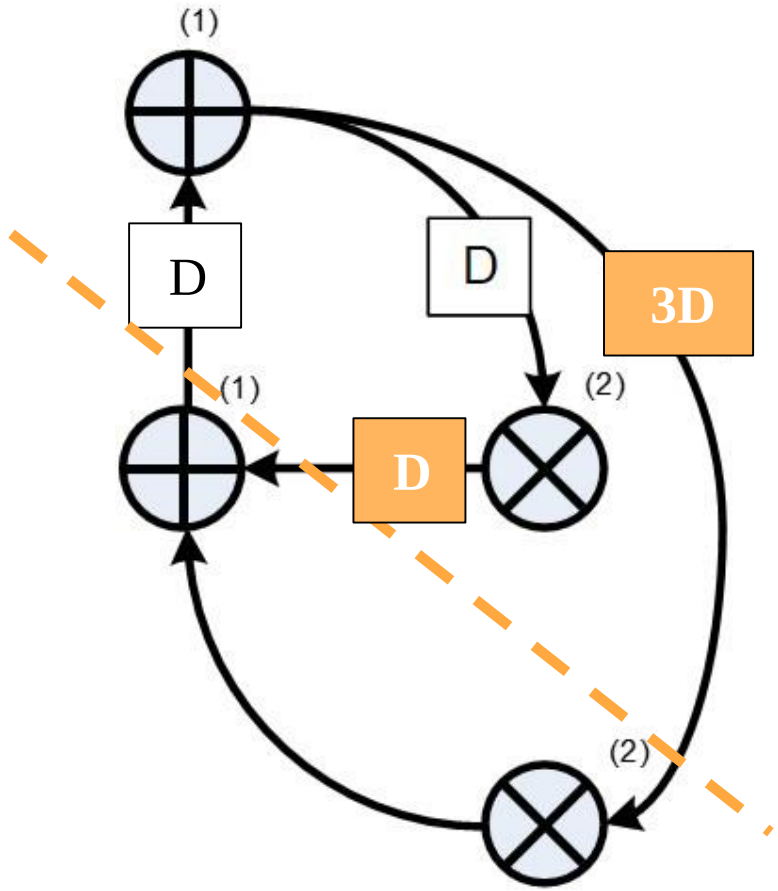
割集

图的一组边，若从图中移走这些边，则图被拆分为互不相连的两个子图或孤立顶点



02. 重定时方法——割集重定时

以IIR滤波器为例



重定时前

关键路径：$T_M + T_A = 3u.t.$

延迟单元：4

在一个方向的边上增加延时

在另外方向的边上减少同样延时

重定时后

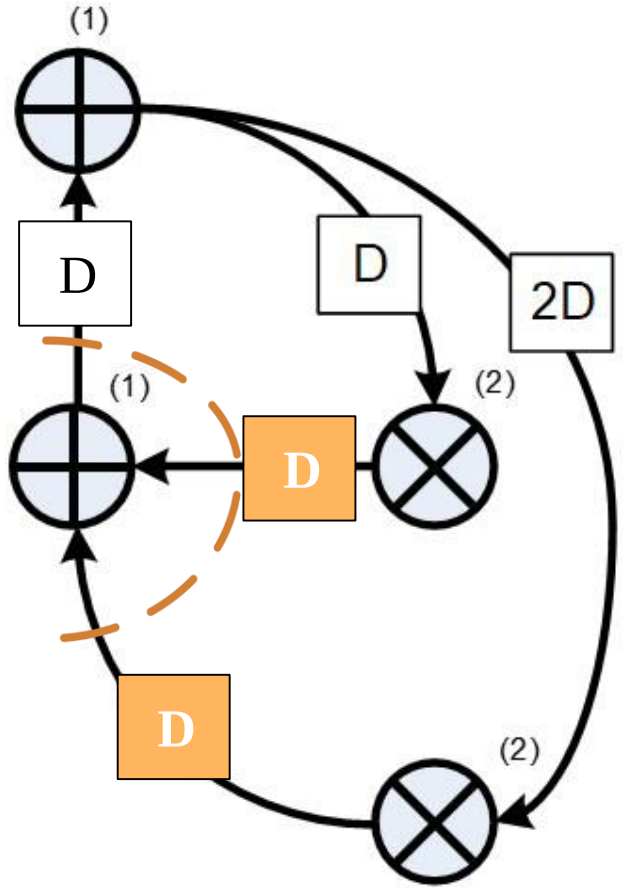
关键路径：$T_M + 2T_A = 4u.t.$

延迟单元：5

02. 重定时方法——割集重定时特例（1）

节点重定时

一个子图仅为一个节点



重定时前

关键路径：$T_M + T_A = 3u.t.$

延迟单元：4

在一个方向的边上增加延时

在另外方向的边上减少同样延时

重定时后

关键路径：$T_M + T_A = 2u.t.$

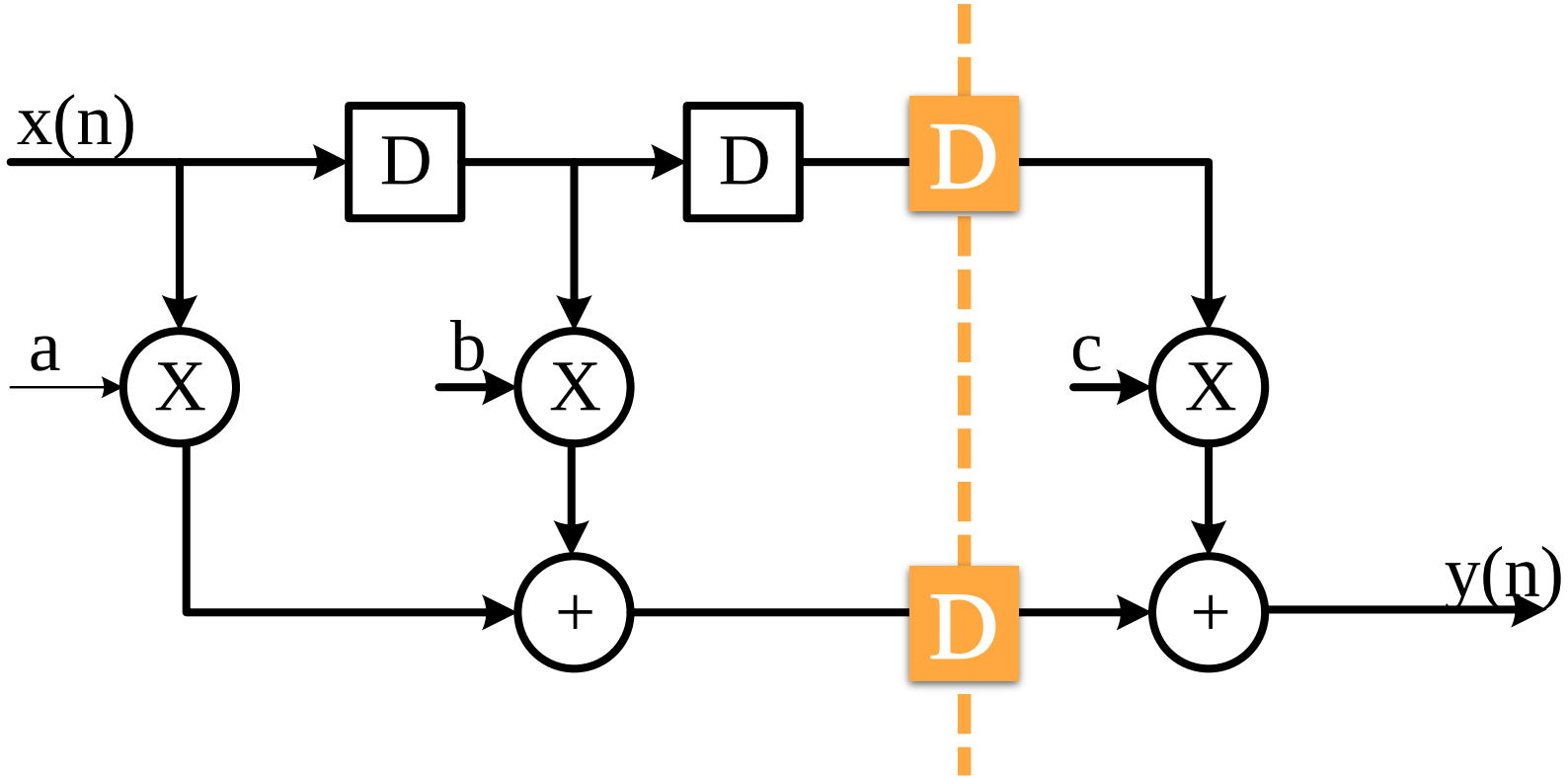
延迟单元：5

02. 重定时方法——割集重定时特例（2）

流水线：针对无环路系统

- 前馈割集的边都是同向的，则都加k延迟，无反向边

划分的割集为前馈割集



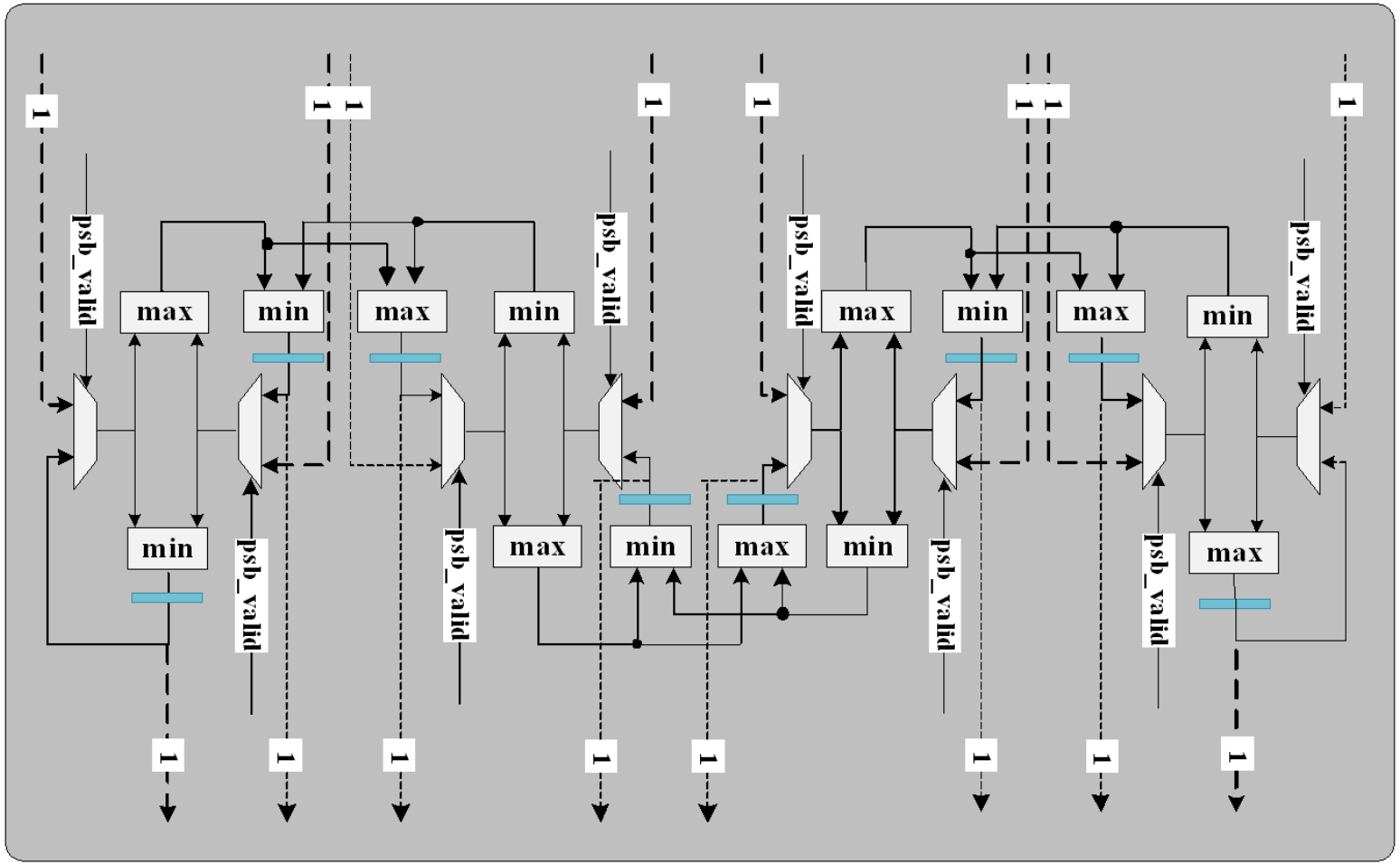
关键路径为T_M+2T_A
2个延时单元

关键路径为T_M+T_A
4个延时单元

目录

03. 重定时的数学求解

简单电路可以手工实现重定时，缩短关键路径



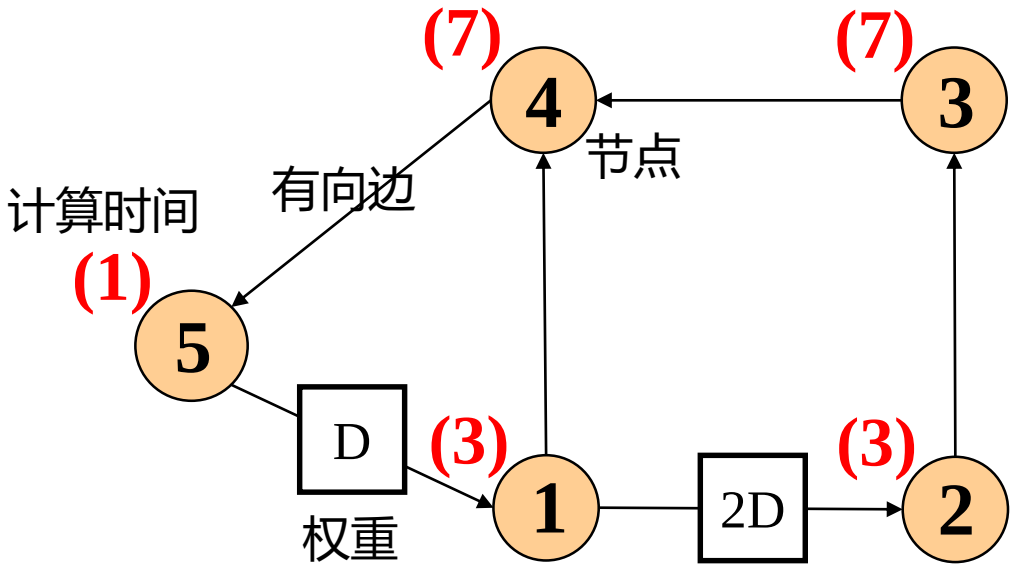
并行排序模块硬件结构

如何对复杂电路重定时?

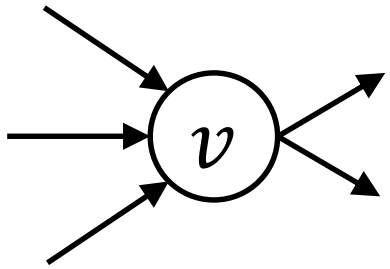
03. 重定时的数学求解——数学定义（1）

用有向图 G 表示电路：

- **节　点**：表示算法中功能的执行，包含计算时间（数字）
- **有向边**：表示节点间通信关系
- **权　重**：表示有向边的寄存器数



重定时值$r(v)$：



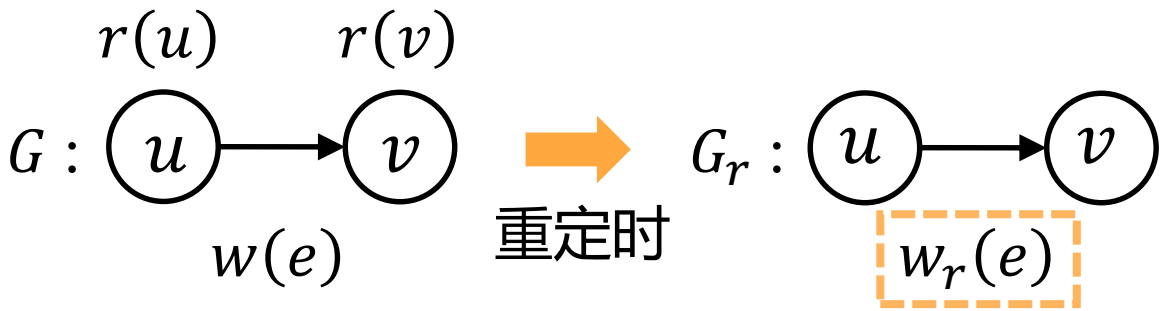
节点v的重定时值$r(v)$表示重定时操作

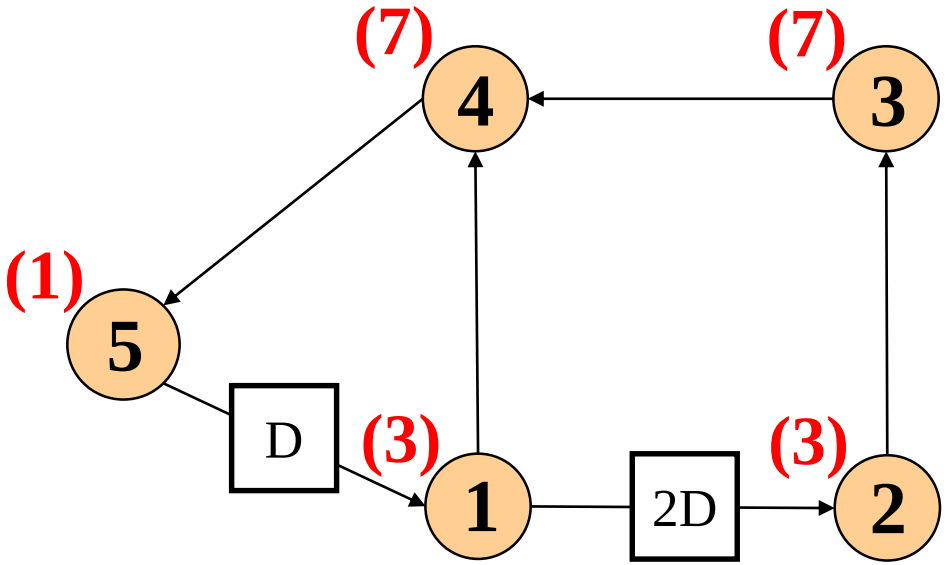
$$\begin{cases} r(v)=0 & \text{不操作} \\ r(v)>0 & \text{节点的每条输入边增加}r(v)\text{延时，同时节点的每条输出边减少}r(v)\text{延时} \\ r(v)<0 & \text{节点的每条输入边减少}r(v)\text{延时，同时节点的每条输出边增加}r(v)\text{延时} \end{cases}$$

03. 重定时的数学求解——数学定义（2）

重定时：

通过调整每个节点v的重定时值$r(v)$，
改变权重，将电路G映射到G_r





重定时方程：

用来确定节点u到v的边$u \to v$重定时后的权重

$$w_r(e) = w(e) + r(v) - r(u)$$

重定时后每条边的延时不小于0，即$w_r(e) \geq 0$

03. 重定时的数学求解——性质

重定时性质：可由重定时方程导出

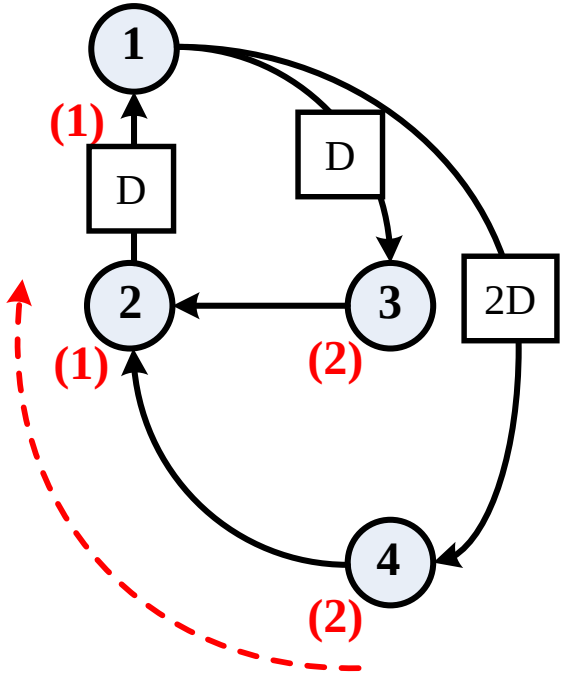
- 对于重定时的路径 $p= V_0 \rightarrow V_1 \rightarrow \ldots \rightarrow V_k$，其权重由下式计算

$$w_r(p) = w(p) + r(V_k) - r(V_0)$$

只与路径起始、终止节点的重定时值相关

- 重定时不改变环路中的总延迟数： 因为环路的 $V_k=V_0$
- 重定时不改变DFG的迭代边界 T_∞：$T_\infty=T_L/W_L$，因为环路的运行时间和延迟数都不变化
- 所有节点重定时值$r(V)$都增加常数值 j， 重定时映射 $G \rightarrow G_r$ 不变

03. 重定时的数学求解——IIR滤波器求解



关键路径3*u.t.*

计算机求解重定时值[1]

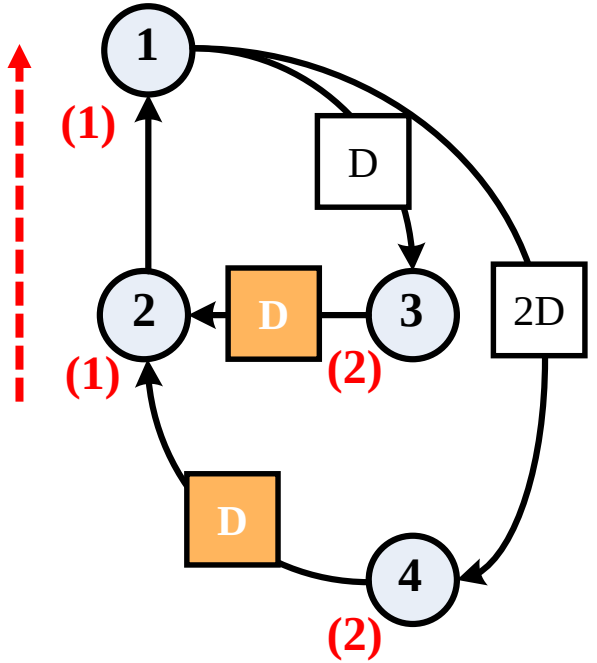
$r(1) = 0$

$r(2) = 1$

$r(3) = 0$

$r(4) = 0$

重定时



关键路径2*u.t.*

初始权重：

$1 \to 3 = 1,\ 1 \to 4 = 2,$

$2 \to 1 = 1,\ 3 \to 2 = 0,$

$4 \to 2 = 0$

$$w_r(e) = w(e) + r(v) - r(u)$$

$$w_r(3 \to 2) = w(3 \to 2) + r(2) - r(3) = 0 + 1 - 0 = 1$$

重定时后权重：

$1 \to 3 = 1,\ 1 \to 4 = 2,$

$2 \to 1 = 0,\ 3 \to 2 = 1,$

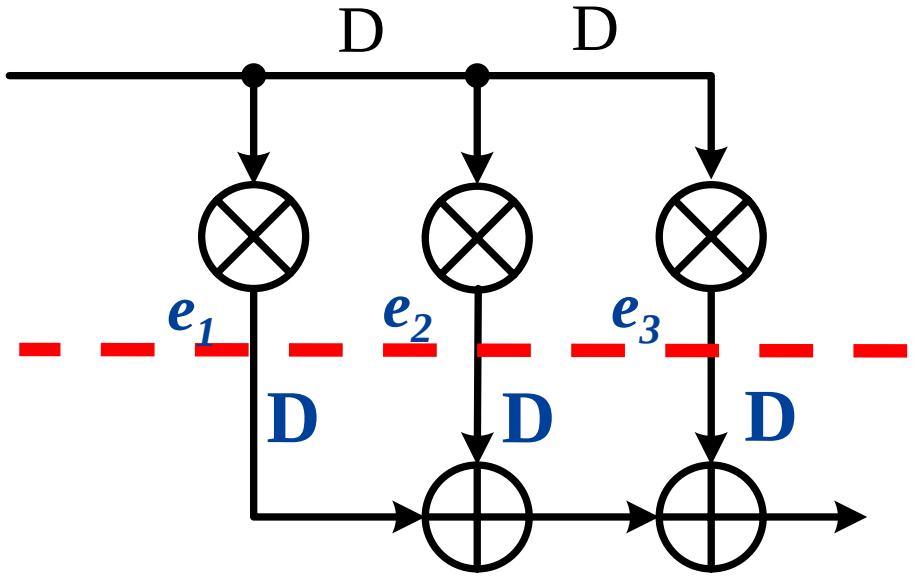
$4 \to 2 = 1$

[1] A. Goldberg, “An efficient implementation of a scaling minimum-cost flow algorithm”, J. Algorithms 22, 1997, pp. 1-29.

03. 重定时的数学求解——FIR滤波器求解

流水线：

$G_1=0$



$G_2=1$

3阶FIR直接形式

前馈割集与重定时

G_1任一节点赋值r(U)=0

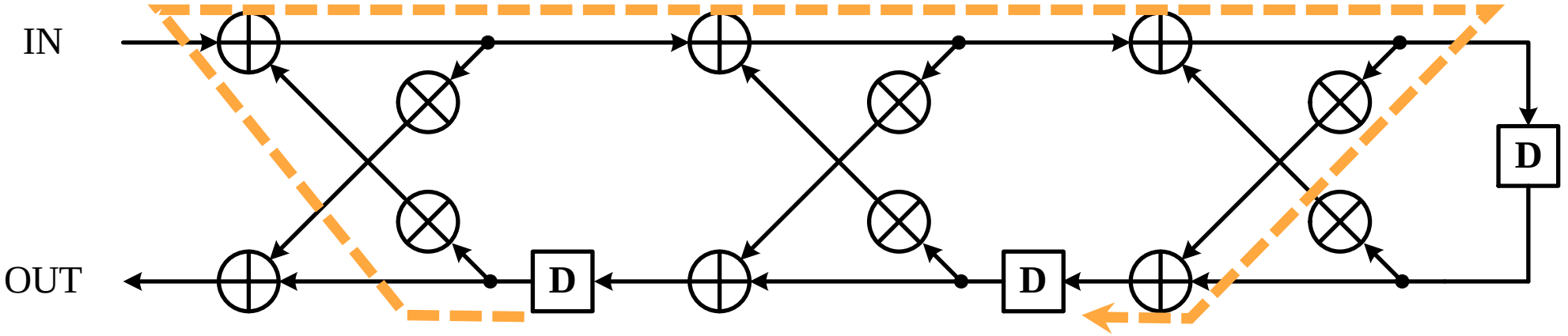
G_2任一节点赋值r(V)=1

重定时后的结构

目录

04. 重定时应用 —— 3阶格型滤波器

关键路径：

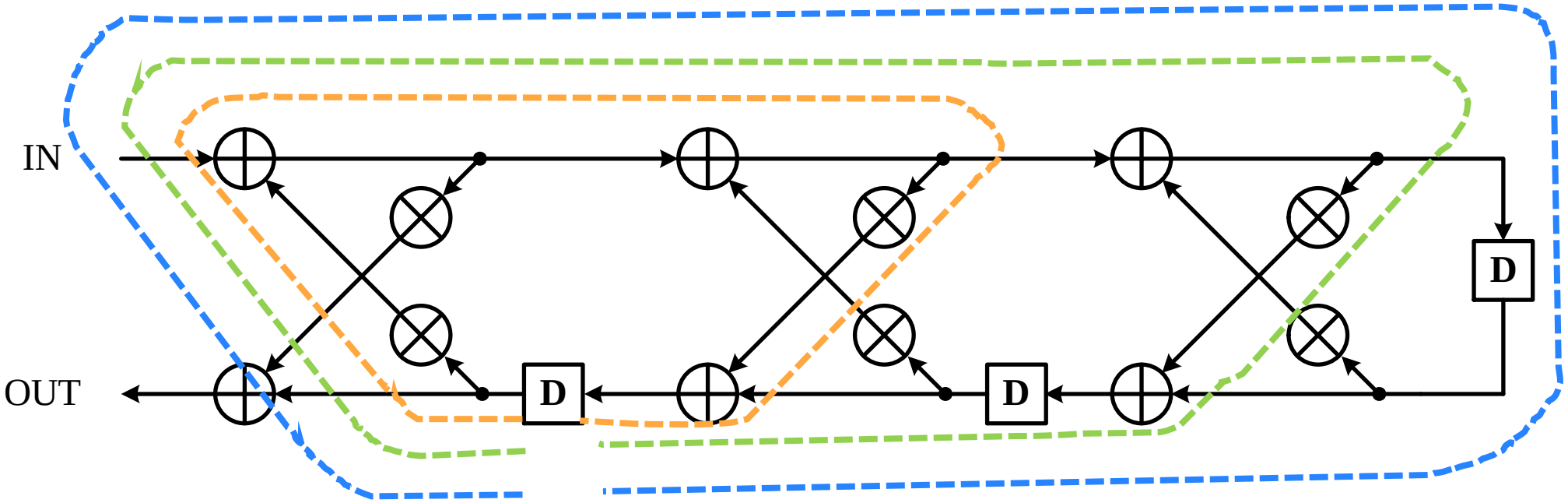


关键路径= $(N+1)T_A + 2T_M$

N = 滤波器阶数

04. 重定时应用 —— 3阶格型滤波器

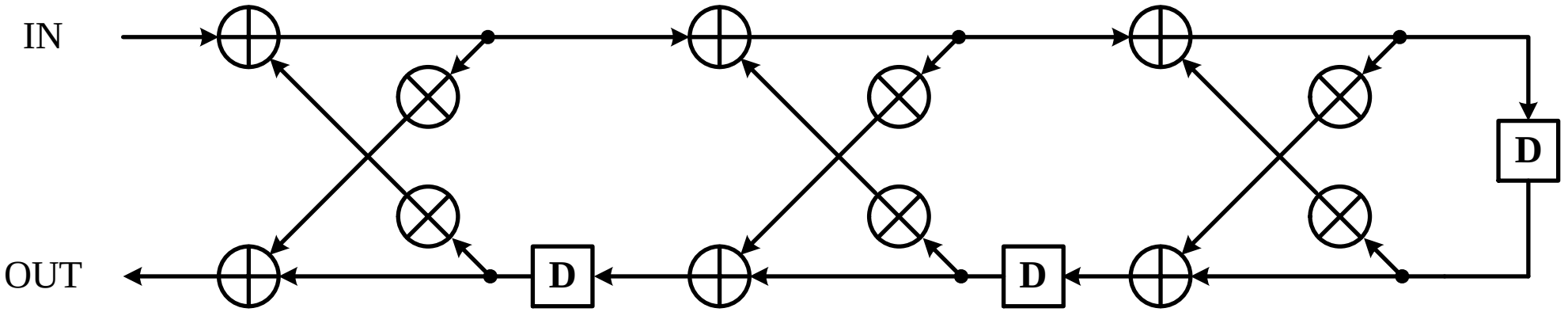
环路边界：



$$T_{L1} = \frac{3T_A + 2T_M}{1} \qquad T_{L2} = \frac{5T_A + 2T_M}{2} \qquad T_{L3} = \frac{5T_A + T_M}{3}$$

04. 重定时应用 —— 3阶格型滤波器

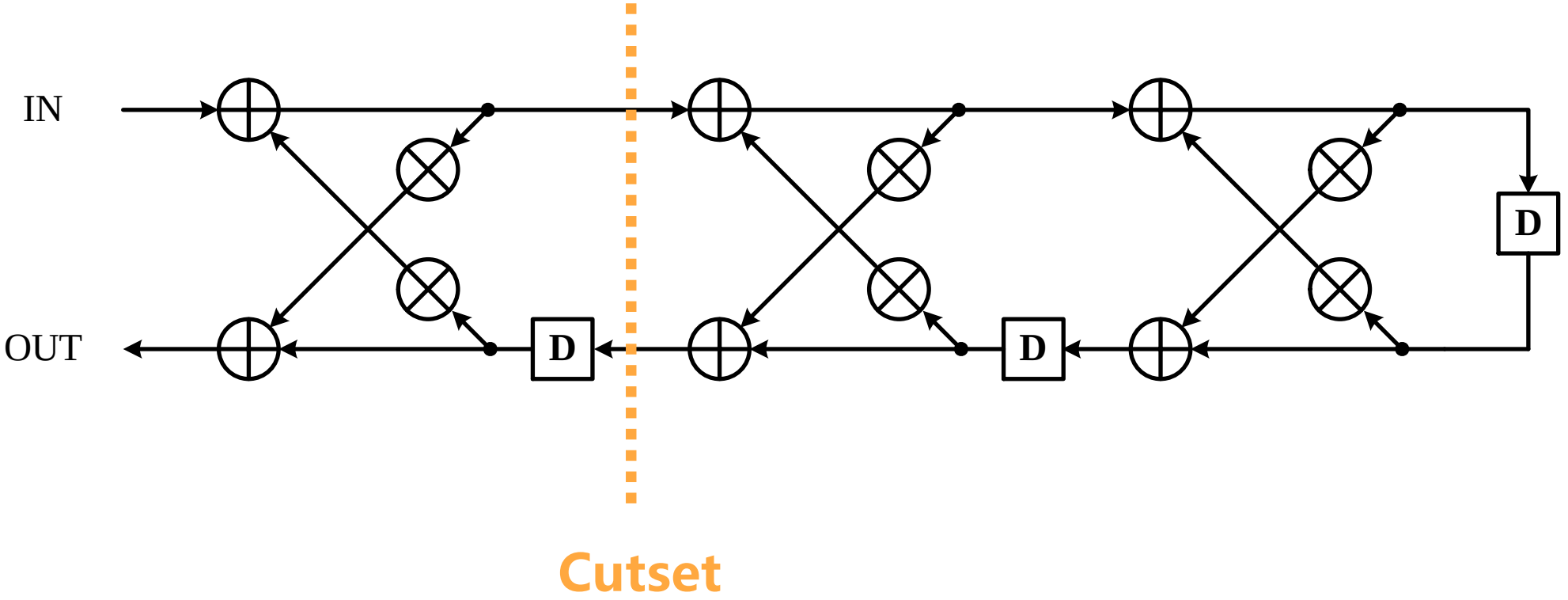
迭代边界：



$$T_\infty = \max\{T_{L1}, T_{L2}, T_{L3}, \ldots\} = \frac{3T_A + 2T_M}{1}$$

04. 重定时应用 —— 3阶格型滤波器

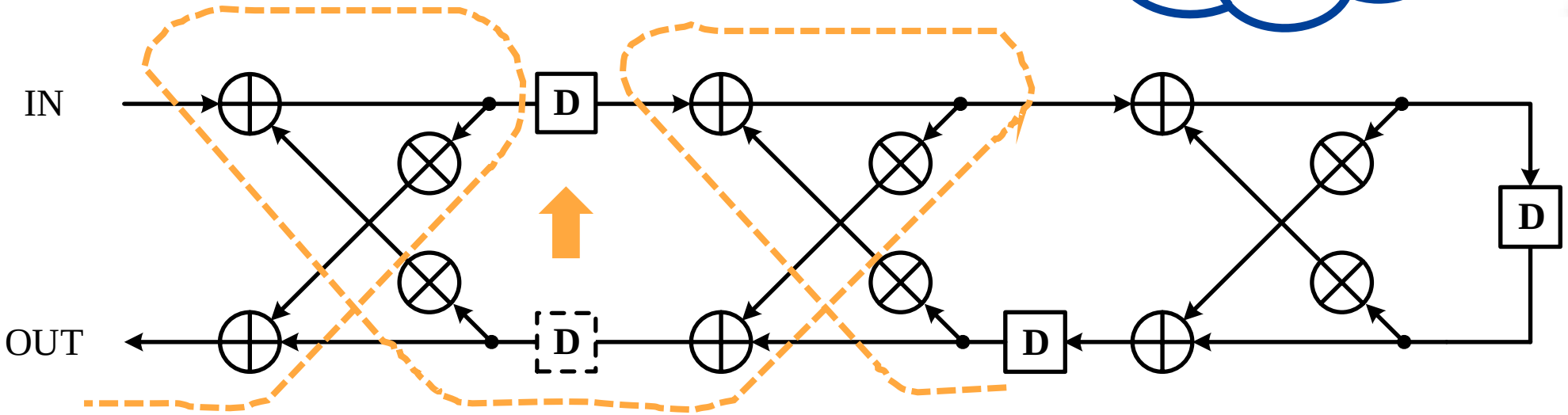
割集重定时：



04. 重定时应用 —— 3阶格型滤波器

割集重定时：

更好的手段?



关键路径

$$T_{critical} = 4T_A + 4T_M$$

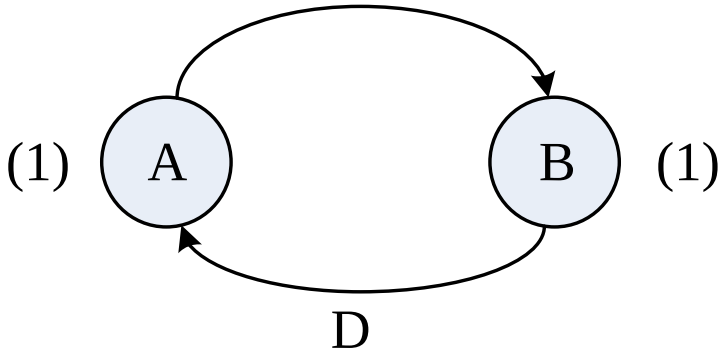
原有关键路径

重定时后关键路径结果更糟糕！

04. 重定时应用 —— k倍降速（k-slow）

用kD取代D：k = 2

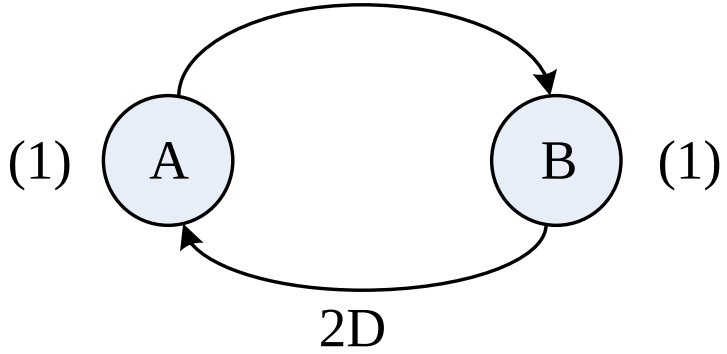
- 是T_{clk}不变的降速
- 隔1时钟输入1样点，奇数时钟插入空操作
- 硬件利用率50%
- 时钟周期不变，T_{clk}=2u.t.，迭代周期加倍，T_{iter}=4u.t.



Time	
0	A0→B0
2	A1→B1
4	A2→B2

T_{clk}=2u.t.
T_{iter}=2u.t.

2-slow



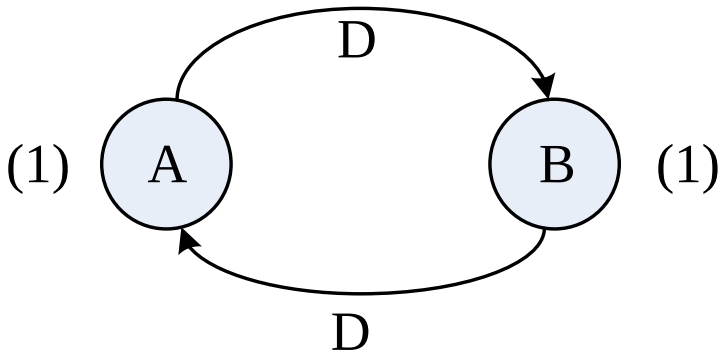
Time	
0	A0→B0
2	
4	A1→B1
6	
8	A2→B2

T_{clk}=2u.t.
T_{iter}=2x2u.t.=4u.t.

04. 重定时应用 —— k倍降速（k-slow）

割集重定时：

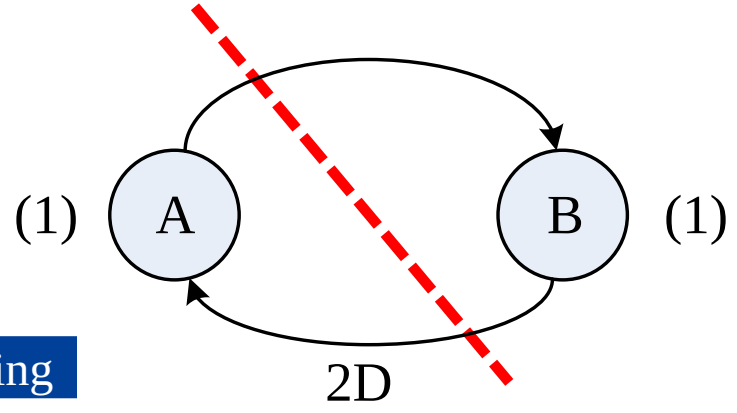
- Tclk缩小一倍，则采样率提高一倍
- 可交替进行奇偶两组独立的迭代，则硬件可以完全利用



T_{clk}=1u.t.
T_{iter}=2x1u.t.=2u.t.

Time	
0	A0→B1
1	B1→A2 A1→B2
2	A2→B3 B2→A3
3	B3→A4 A3→B4
4	A4→B5 B4→A5

Retiming



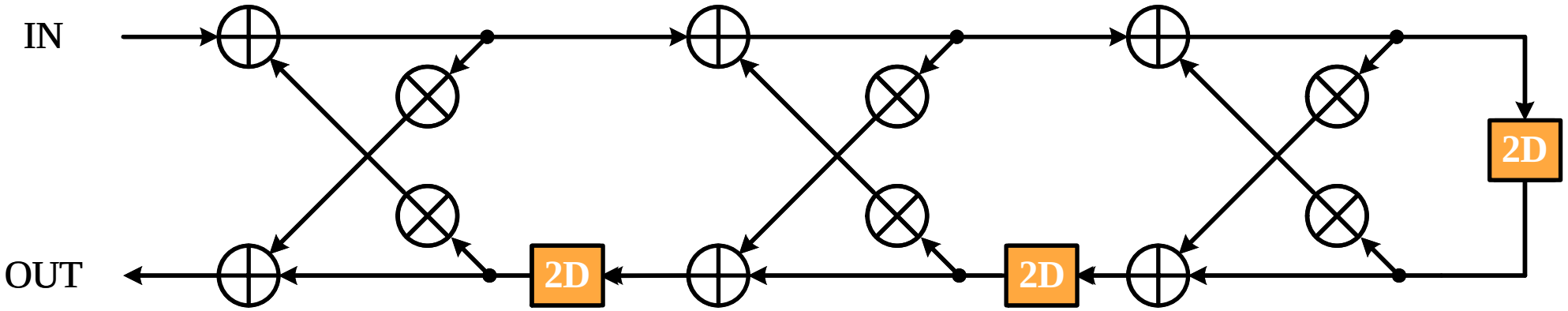
T_{clk}=2u.t.
T_{iter}=2x2u.t.=4u.t.

Time	
0	A0→B0
2	
4	A1→B1
6	
8	A2→B2

04. 重定时应用 —— 3阶格型滤波器

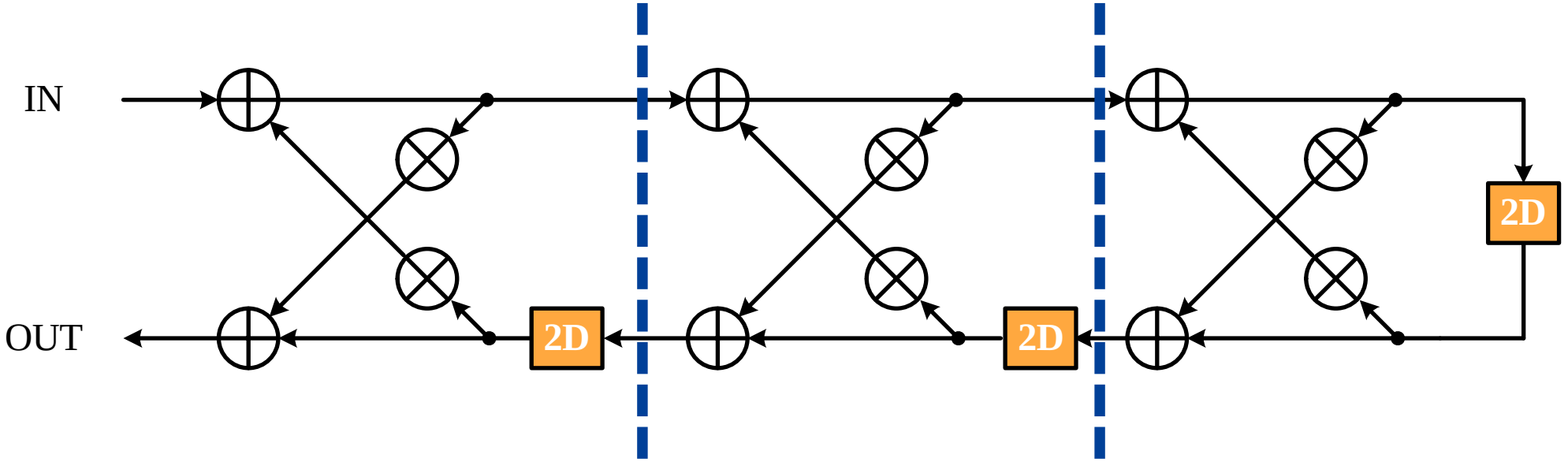
2倍降速（2-slow）重定时：

- 先2倍降速：样点间隔输入x_0, -, x_1, -, x_2, -, x_3, -, …



2倍降速（2-slow）重定时：

- 割集重定时



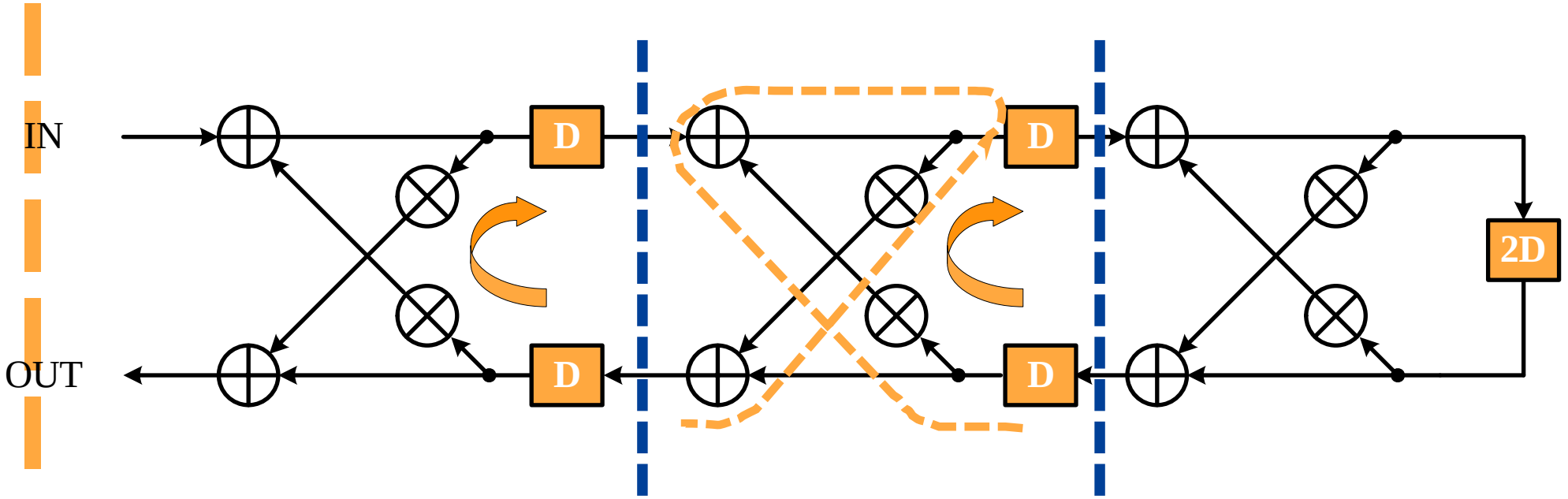
Cutsets

04. 重定时应用 —— 3阶格型滤波器

2倍降速（2-slow）重定时：

重定时后变好

- 关键路径：重定时前：$4T_A+4T_M$　重定时后：$2T_A+2T_M$

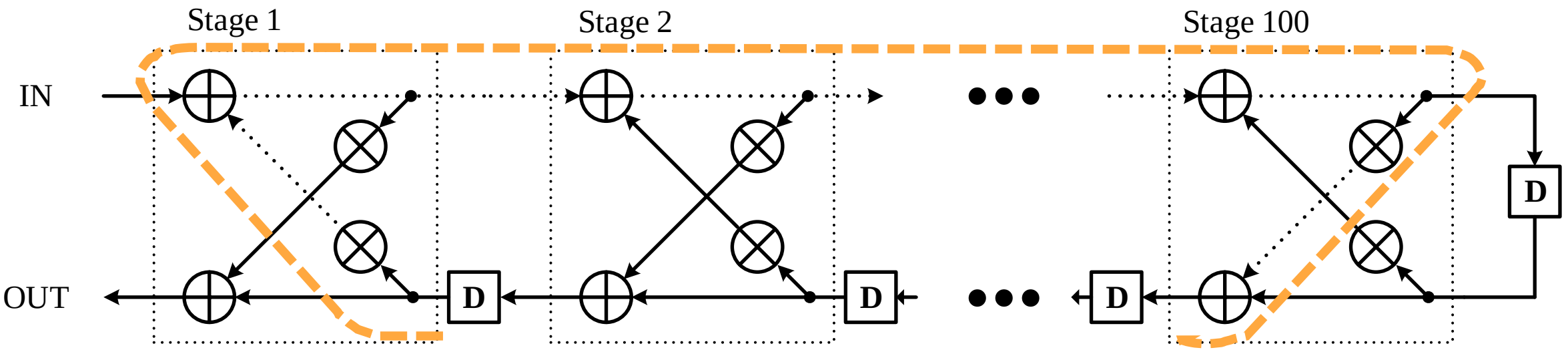


- 割集重定时可以多次使用，以便取得优化的性能

04. 重定时应用 —— 100阶格型滤波器

关键路径?

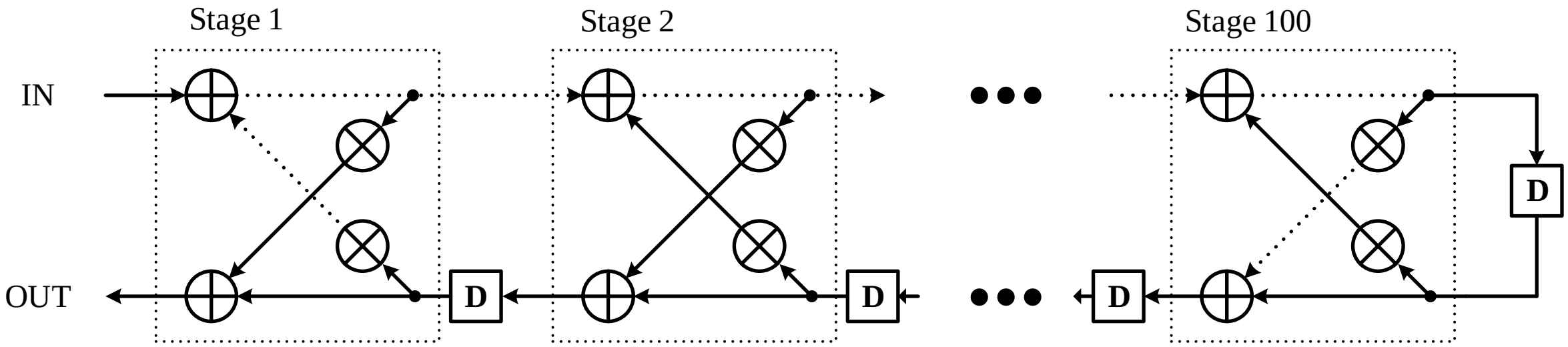
$101T_A+2T_M$



04. 重定时应用 —— 100阶格型滤波器

2 倍降速（2-slow）重定时:

- 最小采样周期为105u.t.的100级格型滤波器

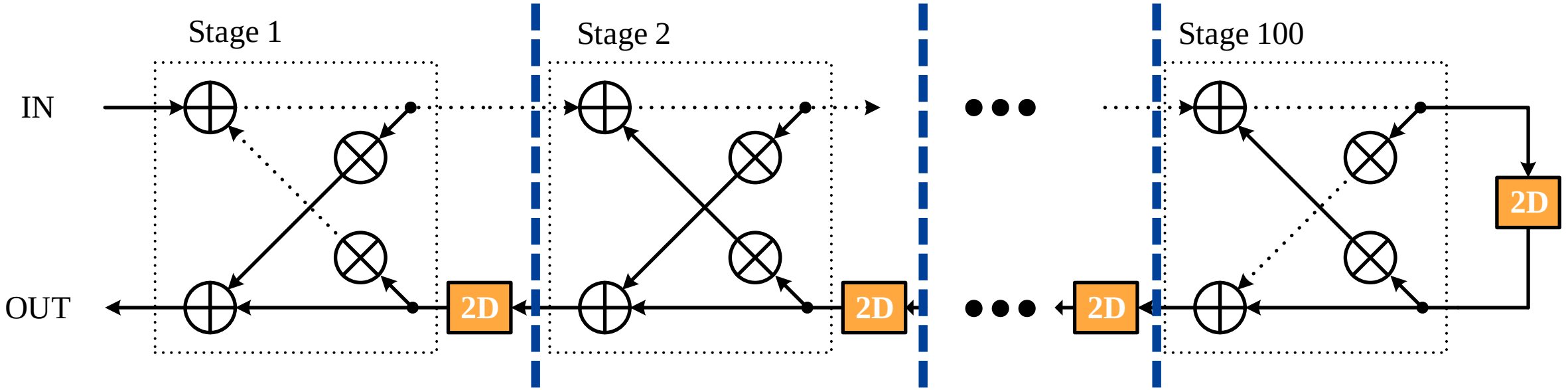


加法器：1u.t.；乘法器：2u.t.

04. 重定时应用 —— 100阶格型滤波器

2 倍降速（2-slow）重定时:

- 电路的2倍降速版本

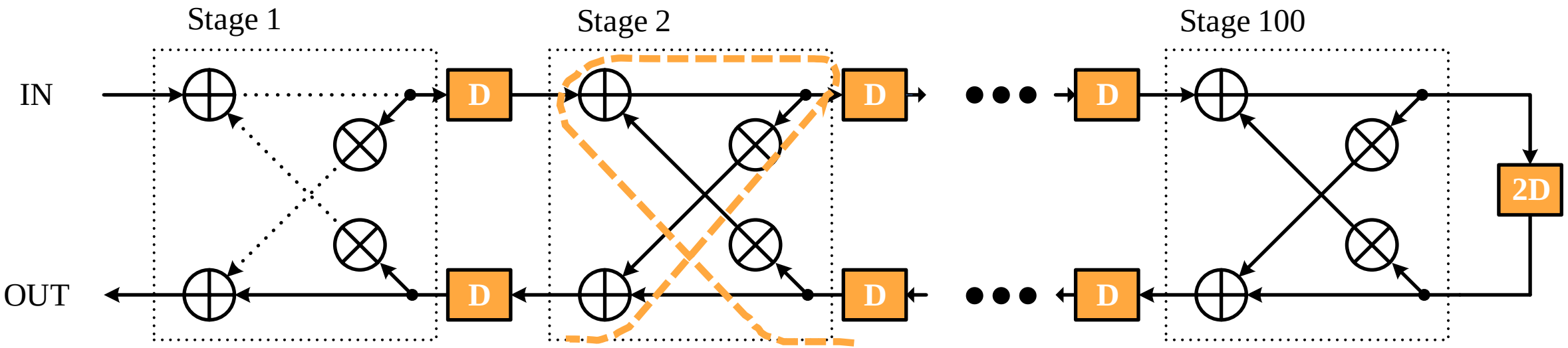


加法器：1u.t.；乘法器：2u.t.

04. 重定时应用 —— 100阶格型滤波器

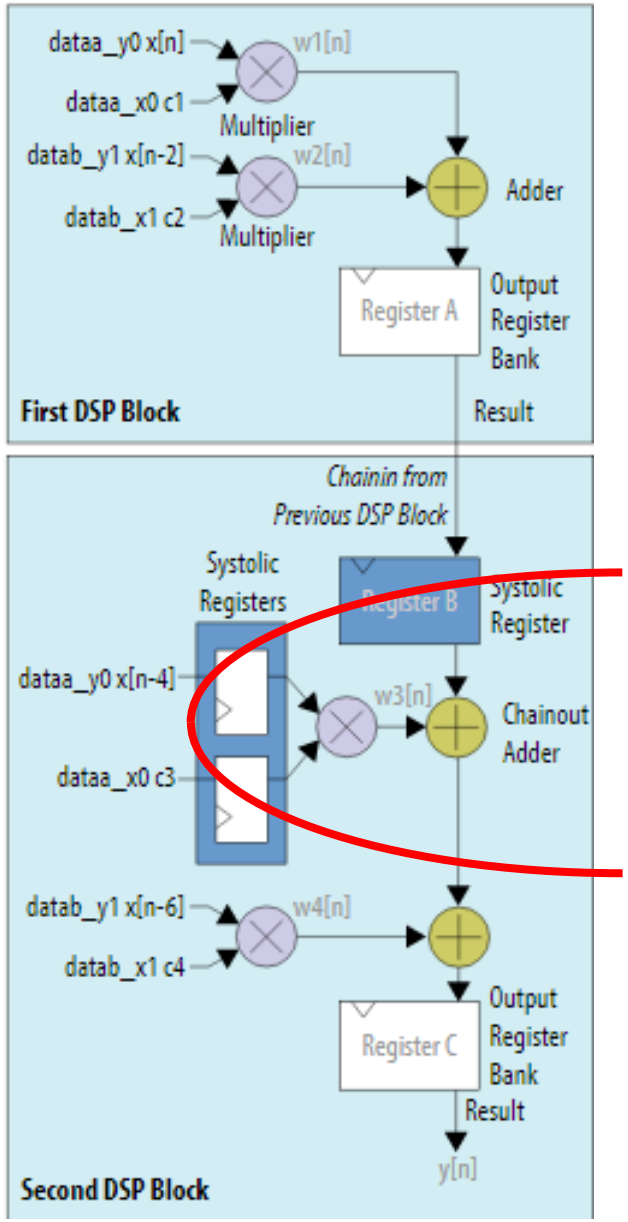
2倍降速（2-slow）重定时：

- 关键路径为6u.t.的2倍降速电路的重定时版本，最小采样周期为12u.t.



加法器：1u.t.；乘法器：2u.t.

04. 重定时应用 —— DSP的优化

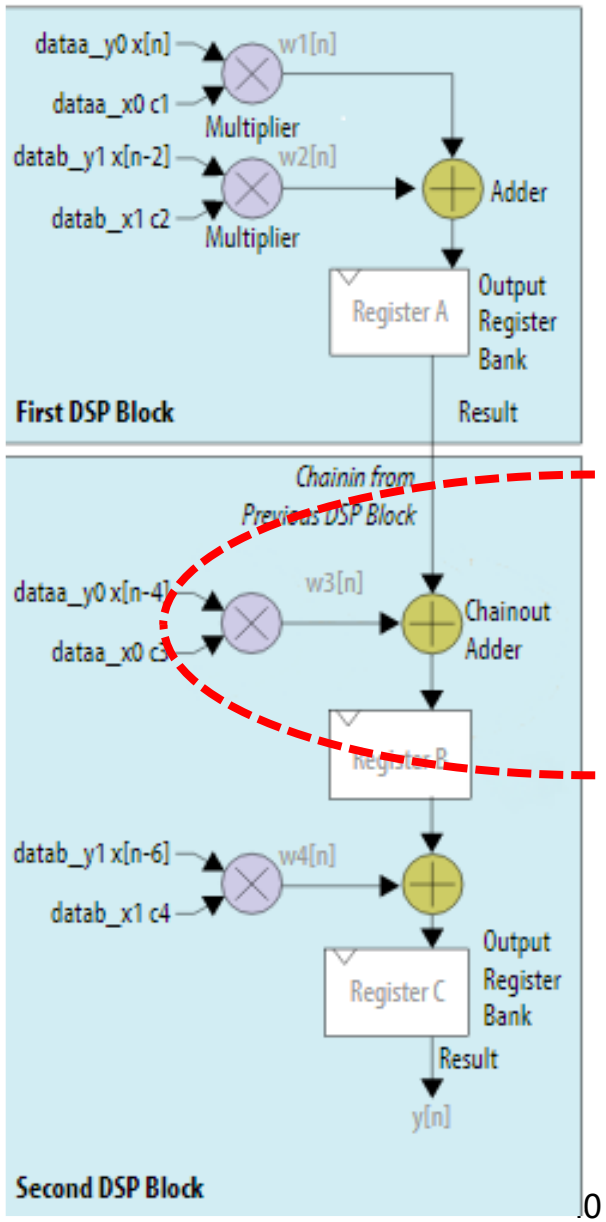


采用重定时前

时钟频率为226MHz

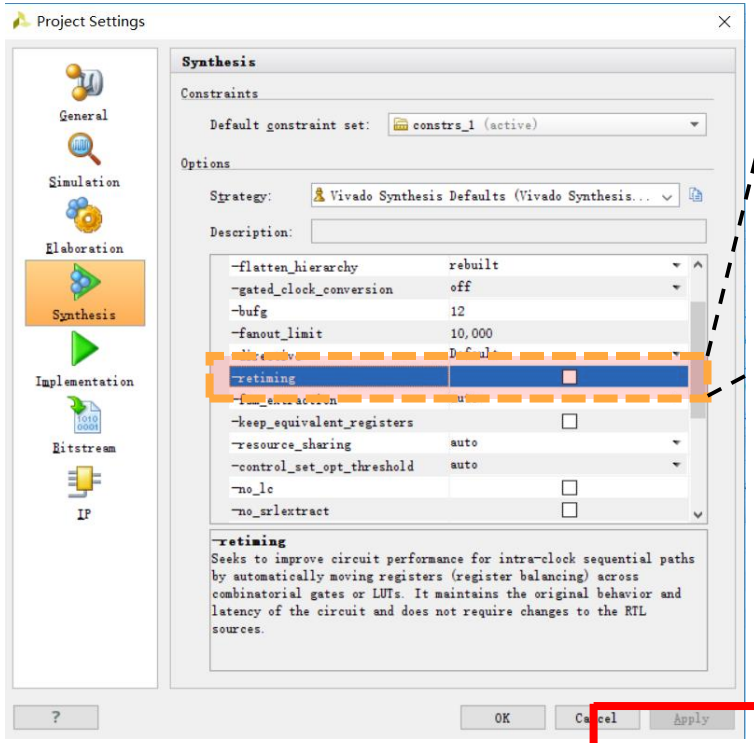
采用重定时后

时钟频率达到280MHz



04. 重定时应用 —— 使用工具进行重定时

工具：Xilinx Vivado 2016.3



-retiming* √

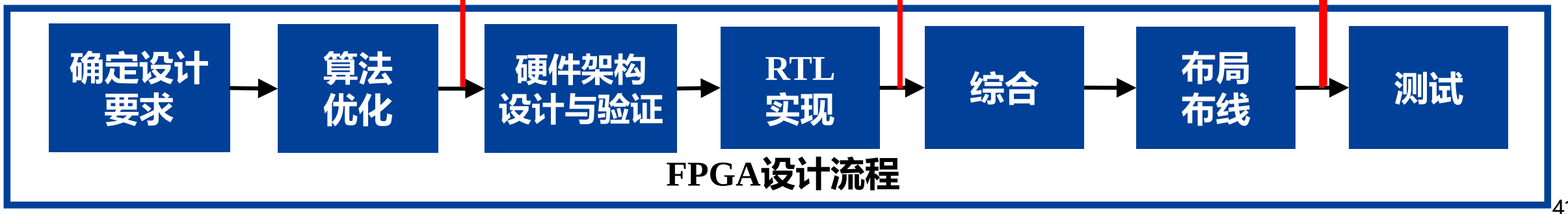
频率由145MHz

提升至151.2MHz

重定时前图像

耗时数周

耗时一天



FPGA设计流程

目录

05. 总结

重定时技术

- 在保持系统功能不变的前提下，改变系统延时数目和分布的方法
- 可以不改变电路结构而提高频率，缩短设计周期

重定时方法

- 割集重定时：
  - 在一个方向的边上增加延时，在另外方向的边上减少同样的延时
  - k倍降速（k-slow）技术

重定时的重要性质

- 不改变环路中的总延迟数
- 不改变DFG的迭代边界T_∞

阅读与思考

割集重定时中常用的技术是c倍降速：在重定时前把每个寄存器换为一组c个寄存器，方便后续重定时优化。阅读文献后思考：如何进行c倍降速?

Post-Placement C-slow Retiming for the Xilinx Virtex FPGA

Nicholas Weaver* UC Berkeley Berkeley, CA

Yury Markovskiy UC Berkeley Berkeley, CA

Yatish Patel UC Berkeley Berkeley, CA

John Wawrzynek UC Berkeley Berkeley, CA

ABSTRACT

C-slow retiming is a process of automatically increasing the throughput of a design by enabling fine grained pipelining of problems with feedback loops. This transfor-

1. Introduction

Leiserson's retiming algorithm[7] offers a polynomial time algorithm to optimize the clock period on arbitrary synchronous circuits without changing circuit semantics.

N. Weaver, Y. Markovskiy, Y. Patel, J. Wawrzynek. “Postplacement C-slow retiming for the Xilinx-Virtex FPGA”. Eleventh ACM International Symposium on Field Programmable Gate Arrays, 2003: 185-194.

谢谢！