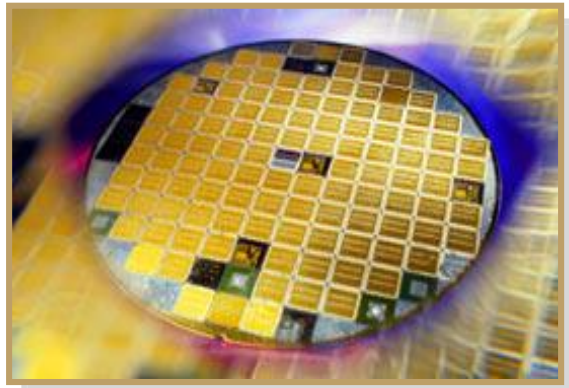
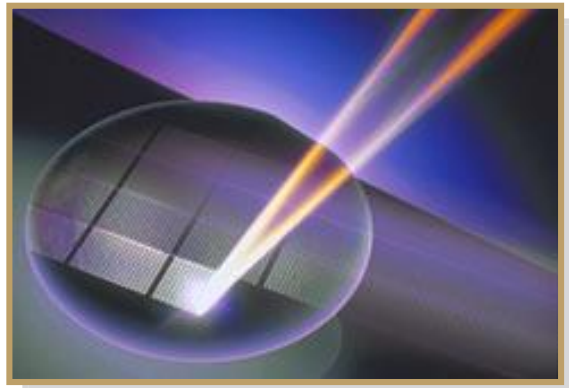
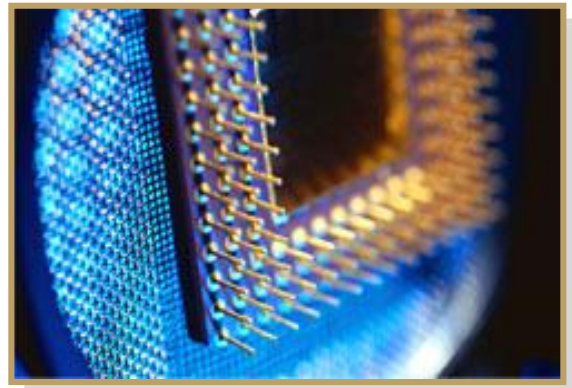
第九章：VLSI设计关键问题——定点和RTL

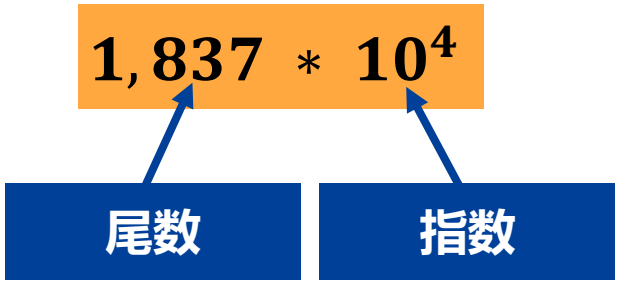




目录

01.有限字长量化——引言

浮点表示

- 数字系统对应十进制表示法
- 相应的二进制标准有很多中，其中一个共同的标准：IEEE 754-1985 (IEC 559)

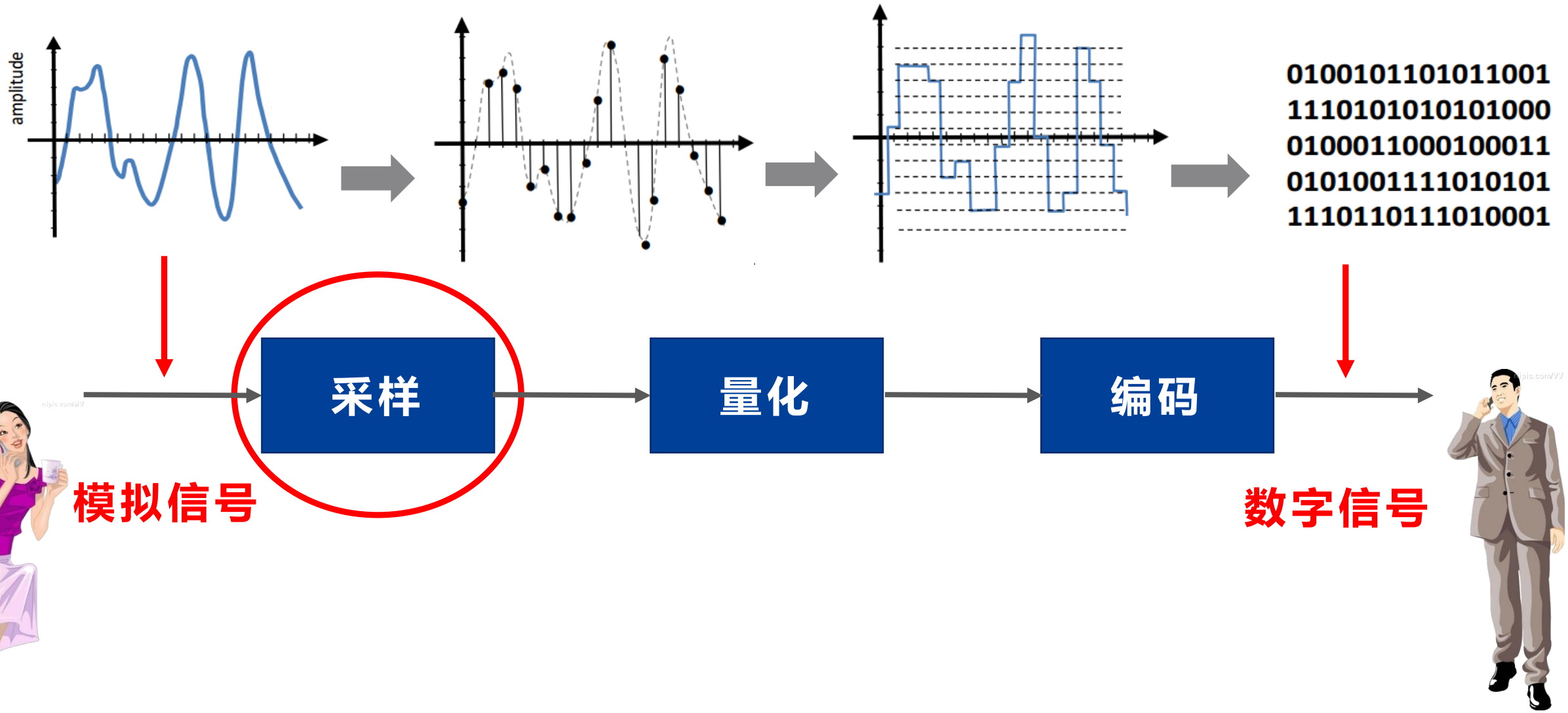


数字系统：存储单元的容量有限

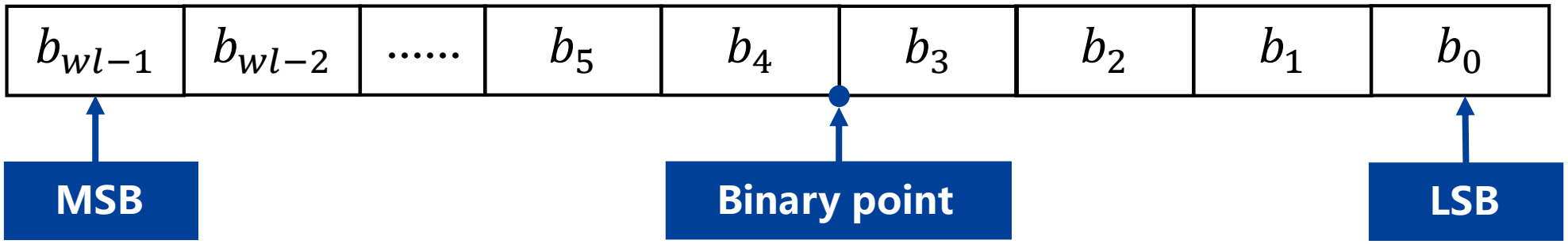
有限字长的影响，主要表现在以下三个方面：

- 输入信号经A/D变换而产生量化误差
- 滤波器的系数量化误差
- 运算误差

01.有限字长量化——量化误差的来源



01.有限字长量化——定点二进制数的表示



其中

- b_i is the i^{th} binary digit.
- wl is the word length in bits.
- b_{wl-1} is the location of the most significant, or highest, bit (MSB).
- b_0 is the location of the least significant, or lowest, bit (LSB).

01.有限字长量化——定点二进制数的表示方法

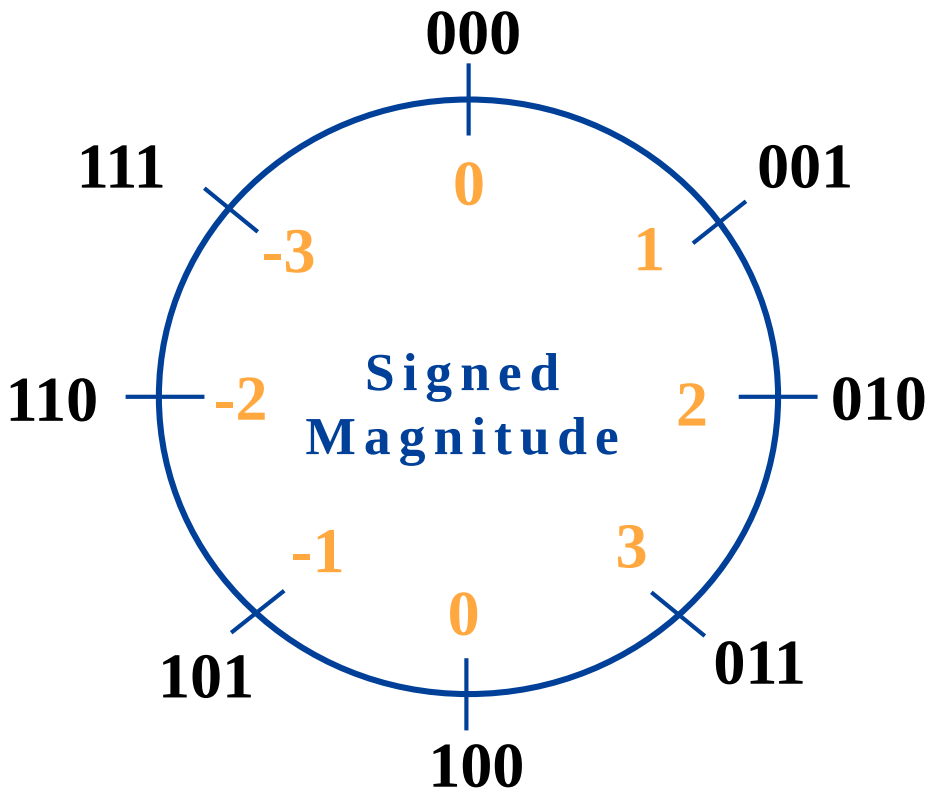
定点二进制数x有原码、反码和补码三种表示形式

若$x = 0.X_1\ X_2 \ldots X_b$，则其原码、反码和补码分别定义为：

- $$[x]_{原}=\begin{cases} x = 0.X_1X_2 \ldots X_b & 0 \le x < 1 \\ -x = 1.X_1X_2 \ldots X_b & -1 \le x < 0 \end{cases}$$
- $$[x]_{反}=\begin{cases} x = 0.X_1X_2 \ldots X_b & 0 \le x < 1 \\ 1 - x = 1.\overline{X_1}\ \overline{X_2} \ldots \overline{X_b} & -1 \le x < 0 \end{cases}$$
- $$[x]_{补}=\begin{cases} x & 0 \le x < 1 \\ 2 + x & -1 \le x < 0 \end{cases}$$

01.有限字长量化——原码 Signed Magnitude

Unsigned numbers with a sign-bit

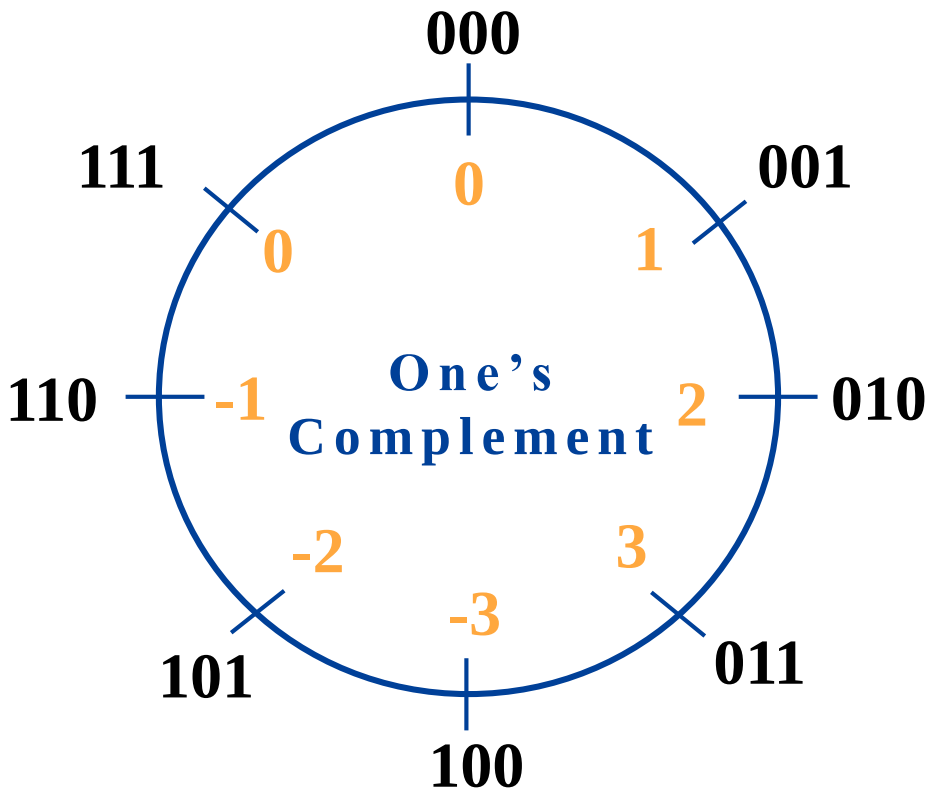


特点

- 两个“0”
- ✓ 低功耗?
- ✓ 容易转化为负数

01.有限字长量化——反码 One's Complement

Signed numbers by inverting （complement)

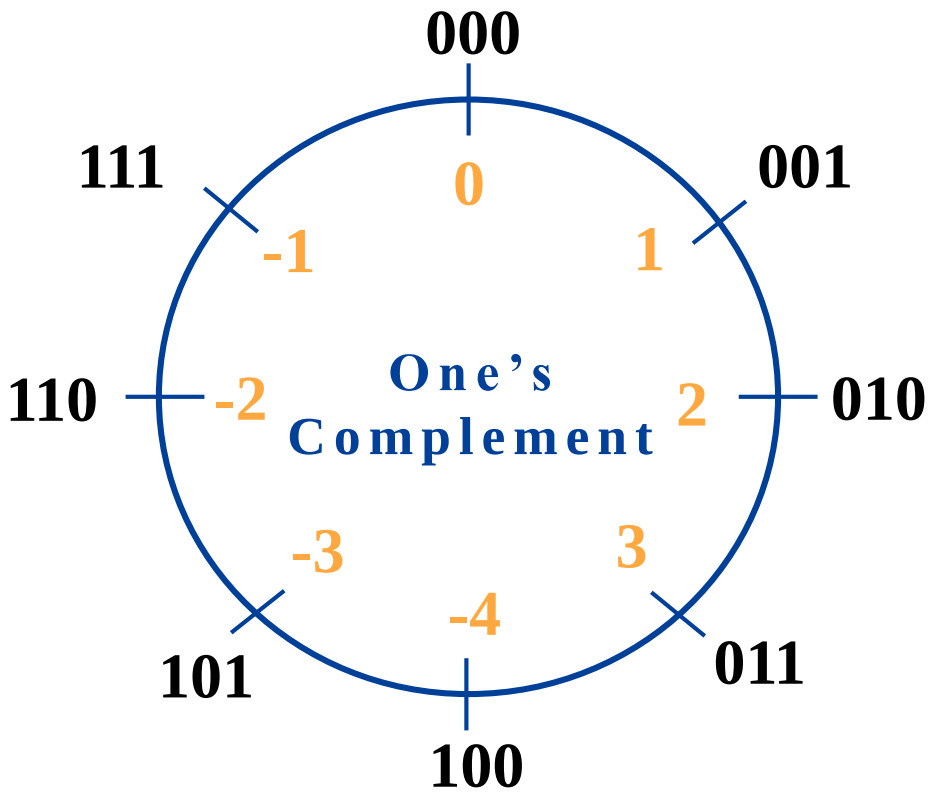


特点

- 两个“0”
- ✓ 容易转化为负数

01.有限字长量化——补码 Two's Complement

Most widely used fixed point numbering system



特点

- ✓ 1个 "0"
- ✓ 容易做加法
- • 不容易转化为负数

01.有限字长量化——定点二进制数的补码表示

1010.0110

$$\Rightarrow -\mathbf{1}*\mathbf{2}^{\mathbf{3}}+\mathbf{0}*\mathbf{2}^{\mathbf{2}}+\mathbf{1}*\mathbf{2}^{\mathbf{1}}+\mathbf{0}*\mathbf{2}^{\mathbf{0}}+\mathbf{0}*\mathbf{2}^{-\mathbf{1}}+\mathbf{1}*\mathbf{2}^{-\mathbf{2}}+\mathbf{1}*\mathbf{2}^{-\mathbf{3}}+0^{*}\,\mathbf{2}^{-4}$$

$$=-8+2+\frac{1}{4}+\frac{1}{8}$$

$01=\ (0+2^{0})\ =1;\quad 11=\ (-2^{1}+2^{0})\ =-1$

加法

```
  010010.1     (18.5)
+  0110.110   (6.75)
-------------------
  011001.010  (25.25)
```

减法

```
  010010.1     (18.5)
-  0110.110   (6.75)
```

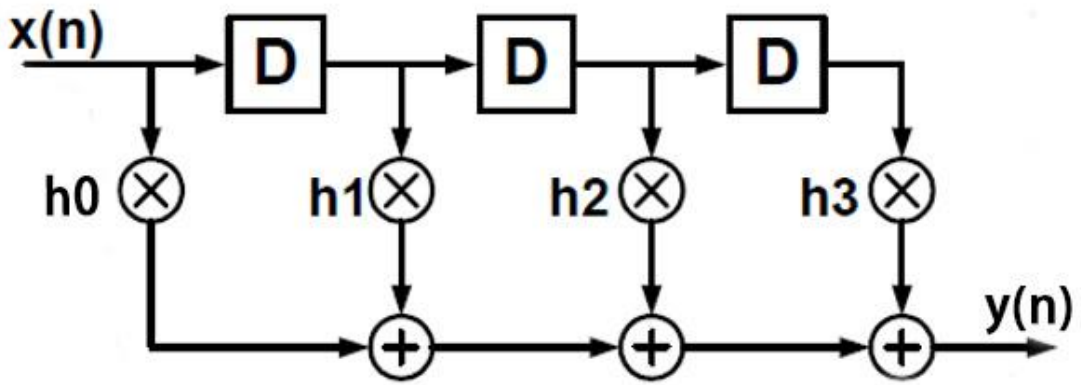
取补码

符号位拓展

```
  010010.1     (18.5)
+ 111001.010   (-6.75)
-------------------
1 001011.110   (11.75)
```

舍弃进位比特

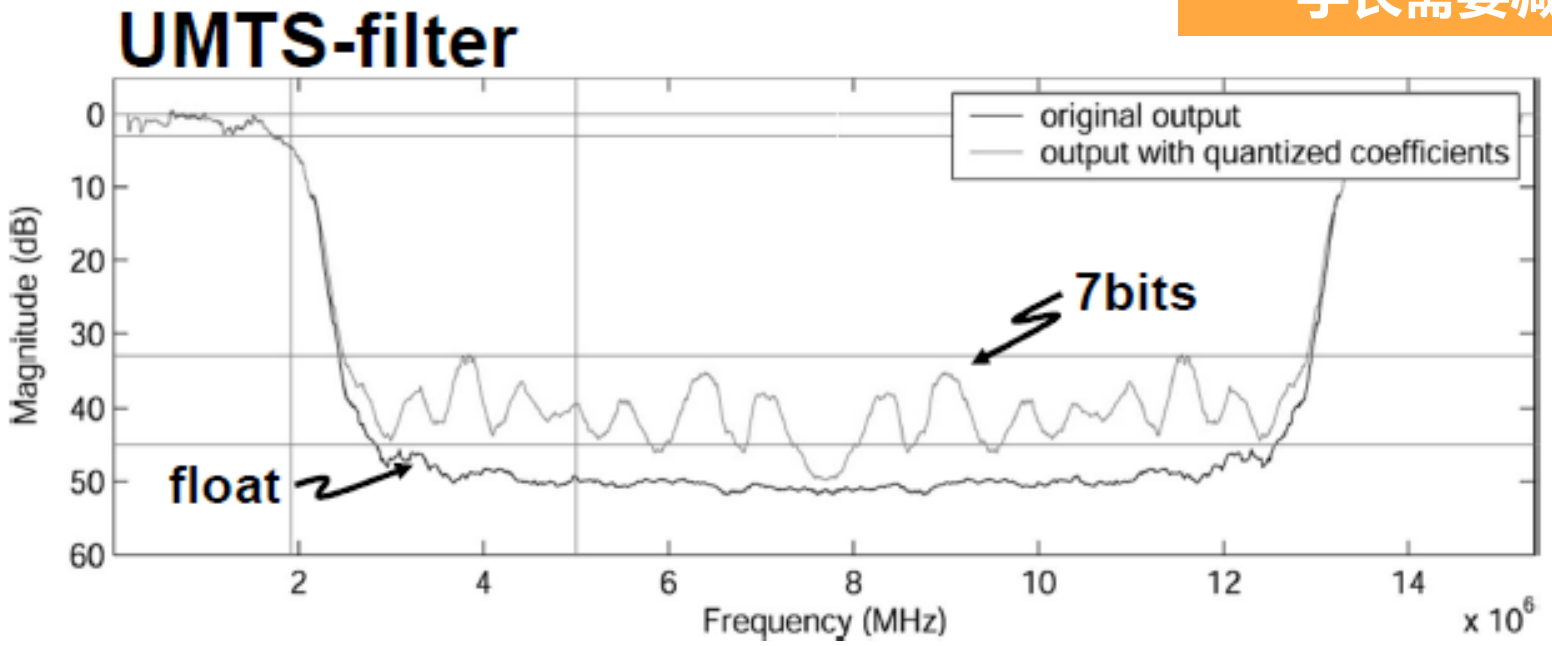
01.有限字长量化——字长



多余的比特带来

- 能量消耗
- 延时增大
- 面积增大

字长需要减少



01.有限字长量化——量化及量化误差

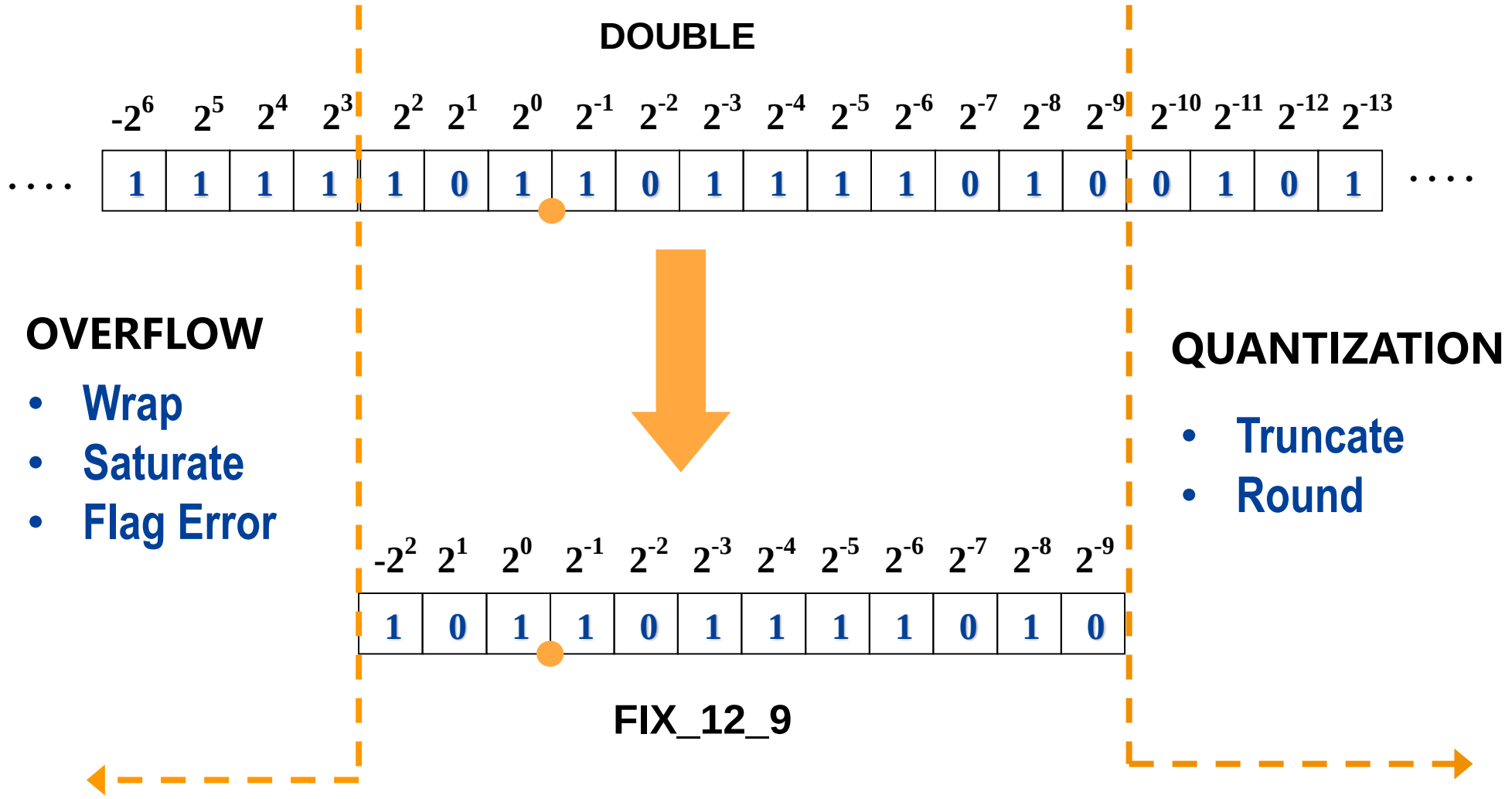
理论上十进制数可用无穷多位二进制数表示

$$x = \beta_0 + \sum_{n=1}^{\infty} \beta_n 2^{-n}$$

符号位　　有效数字位

实际中，只能用有限位近似表示(b+1)位，这种过程称为**量化**。

01.有限字长量化——浮点数到定点数的量化

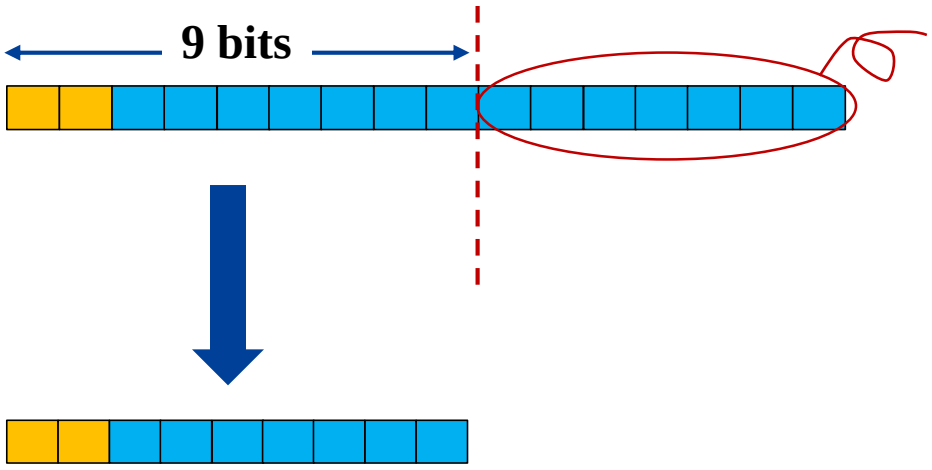


01.有限字长量化——量化

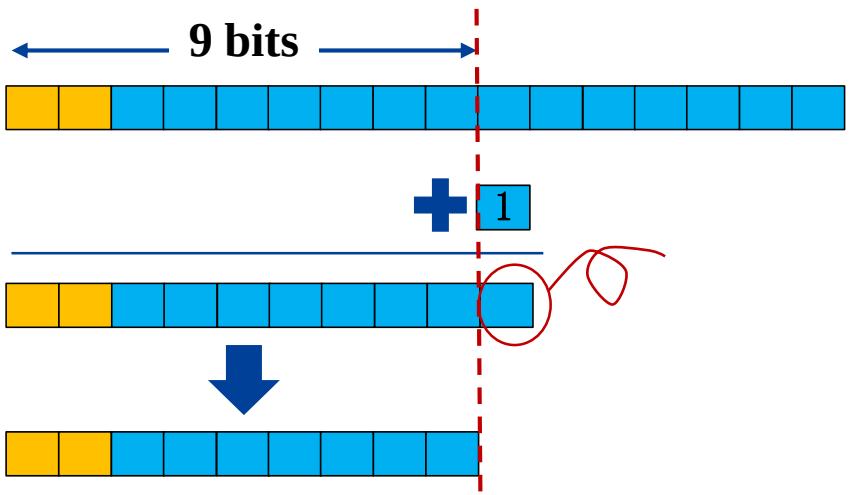
发生在小数位数不足以表达该数值的小数部分时

用户可选择：

- 截尾（Truncate）
- 舍入（Round）：更精确，但需要更多操作



截尾



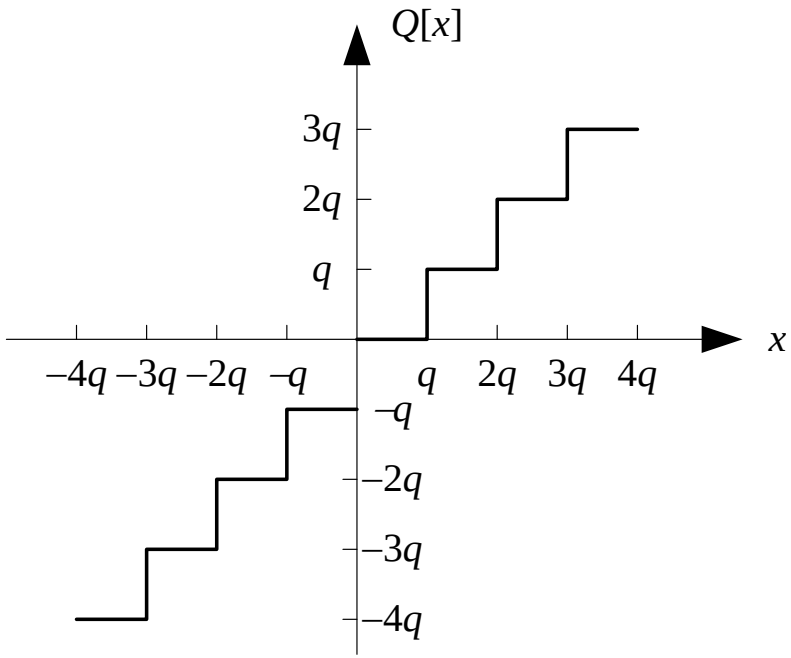
舍入

01.有限字长量化——量化方式

截尾量化

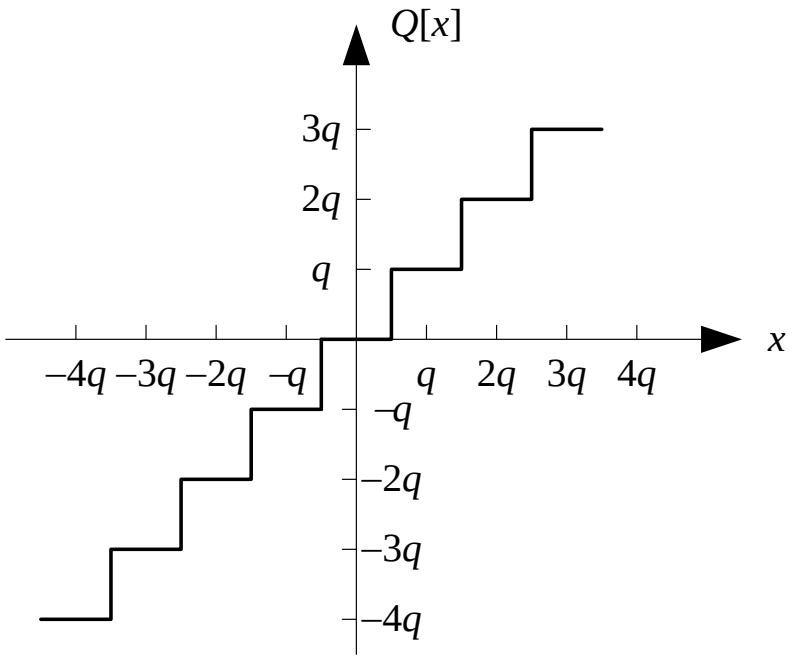
截掉b位之后数据

$$Q[x] = \beta_0 + \sum_{n=1}^{b} \beta_n 2^{-n}$$



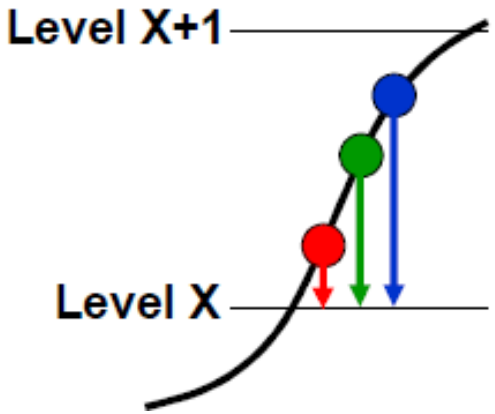
舍入量化

视b+1位后数据的大小决定b位数据的值



01.有限字长量化——截尾和舍入误差

截尾量化



所有数值朝向同一方向进行近似

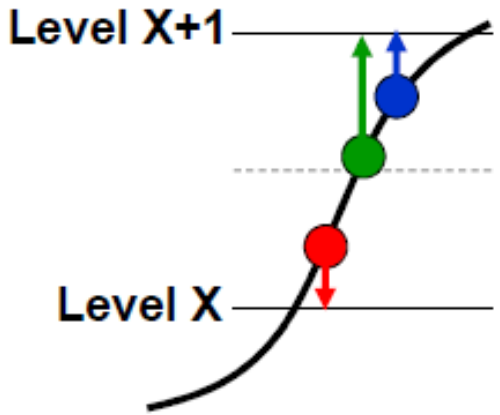
最大误差 = 1LSB

截尾误差： $E_T = Q[x] - x$

正数和补码负数截尾误差范围为：

$-q < E_T \leq 0 \quad q = 2^{-b}$

舍入量化



数值朝上或朝下进行近似

最大误差= 1/2 LSB

舍入误差： $E_R = Q[x] - x$

舍入误差范围为：

$-q/2 \leq E_R \leq q/2$

舍入误差对称分布
截尾误差单极性分布

01.有限字长量化——溢出问题

产生原因：

$$y[k]=\sum_n h[n]x[k-n] \Rightarrow |y[k]|_{\max} \le x_{\max}\sum_n |h[n]|$$

x[k]的最大绝对值

$x_{\max}\sum_n |h[n]|$ 超出了表示范围，就会产生溢出。

避免溢出的方法：

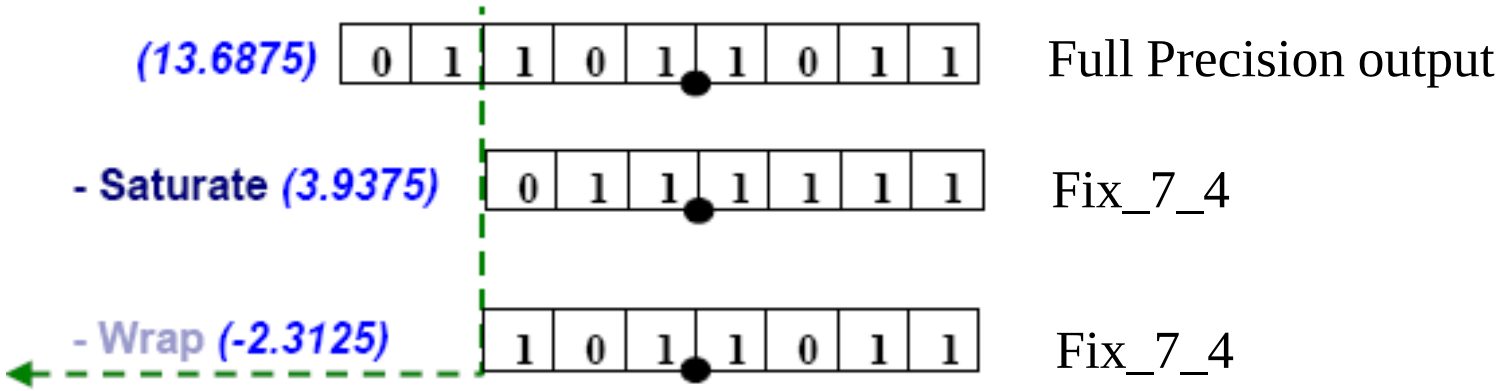
- **适当增加字长**
- **将输入信号乘以小于1的比例因子A，使下式成立**

$$Ax_{\max}\sum_n |h[n]| < 1$$

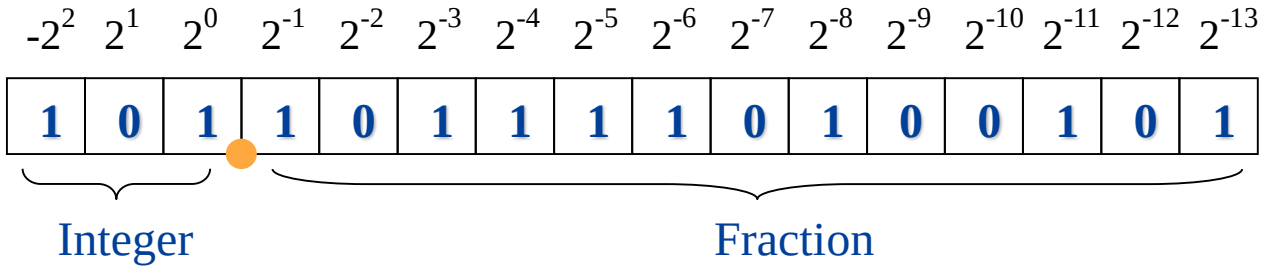
01.有限字长量化——溢出问题

发生溢出时，用户可选择：

- 饱和（Saturate）近似到最大的正（或最大负）值
- 裁剪（Wrap）该值，即丢弃超出定点数中最高位的任何有效位
- 在模拟期间将溢出标记（Flag）为错误



01.有限字长量化——二进制小数运算



Value = -2.261108…

Format = Fix_16_13

Format	Sign_Width_Decimal point from the LSB
Sign	Fix = Signed Value ；UFix = Unsigned value

使用以上方法，转换下面的小数

定义以下二进制小数（补码形式）的format，计算其表达的数值

Format = < Fix_ 12 _ 5 >　　**Value = -28.65625**

01.有限字长量化——问题

表示以下数字，应采用什么Format?

a) Max value: +1
Min value: -1
Quantized to 12 bit data

b) Max value: 0.8
Min value: 0.2
Quantized to 10 bit data

c) Max value: 278
Min value: -138
Quantized to 11 bit data

表示以下计算结果，应采用什么Format?

Operation	Full Precision Output Type
<Fix_12_9> + <Fix_8_3>	
<Fix_8_7> x <Ufix_8_6>	

Operation	Full Precision Output Type
<Fix_12_9> + <Fix_8_3>	**<FIX_15_9>**
<Fix_8_7> x <Ufix_8_6>	**<FIX_16_13>**

目录

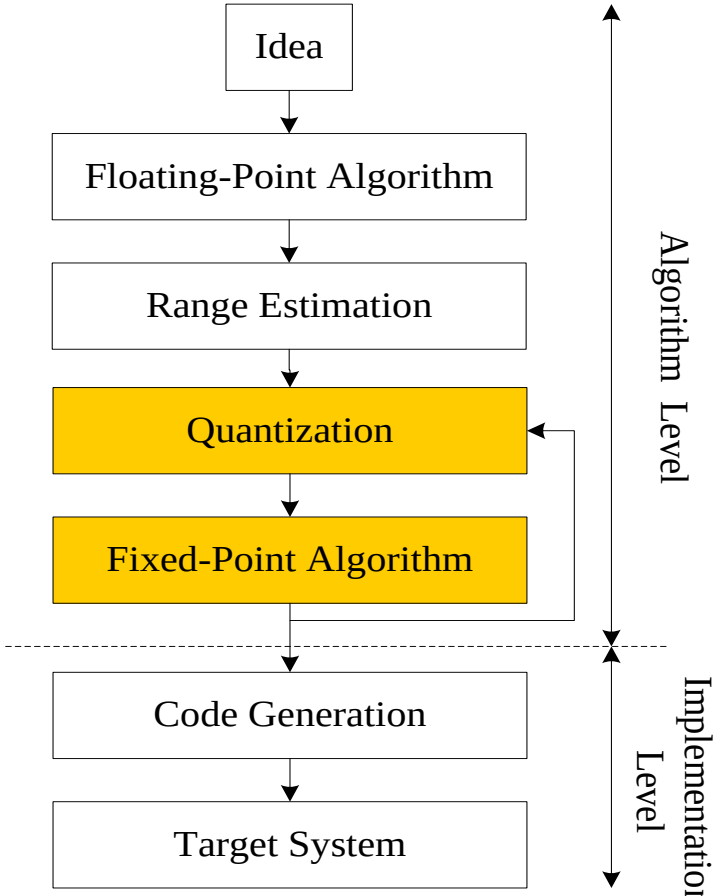
02.浮点转定点量化

数字信号处理算法

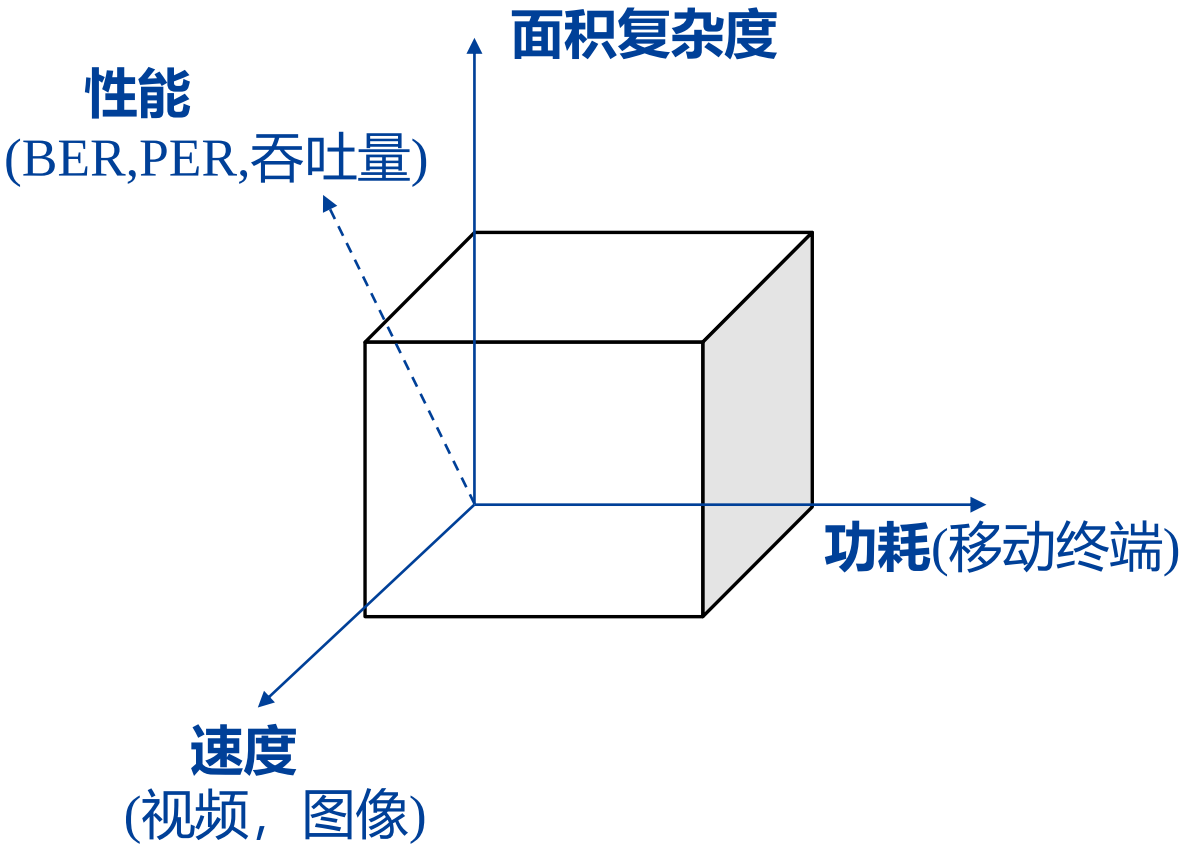
- 通常以浮点数形式开发
- 之后映射到定点，用于数字硬件实现

定点化的数字硬件

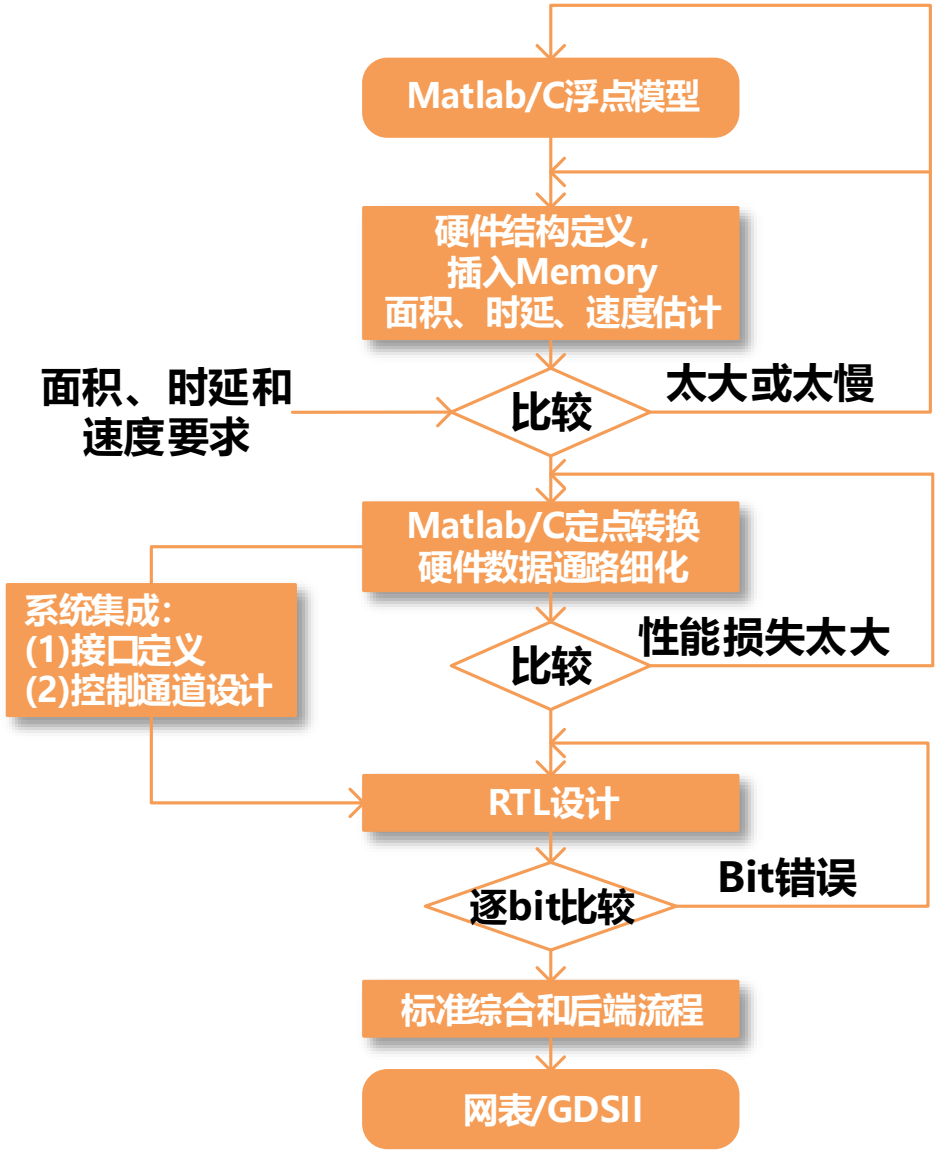
- 面积更小
- 功耗更低
- 单位生产成本更低



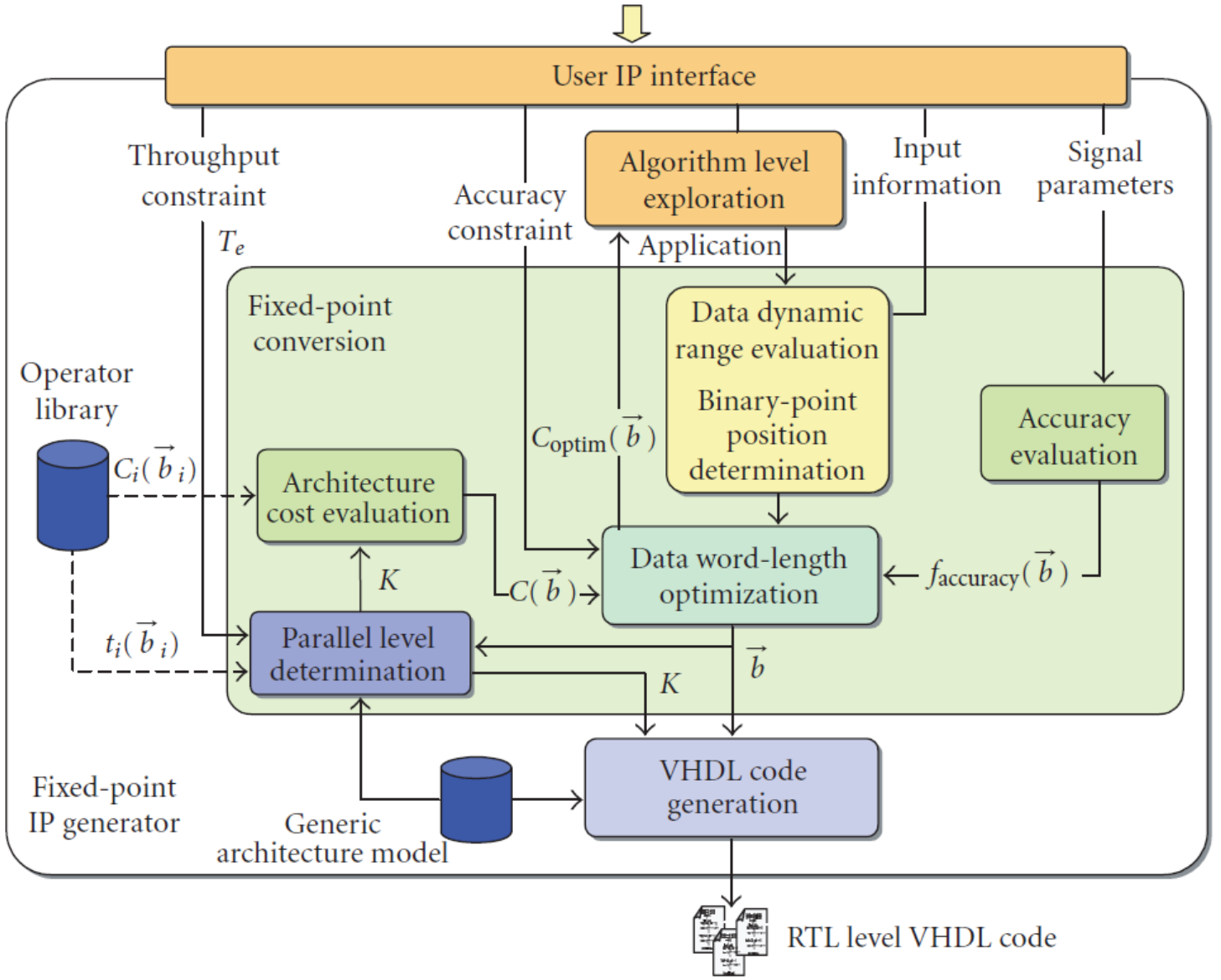
02.浮点转定点量化—— VLSI数字信号处理系统设计流程



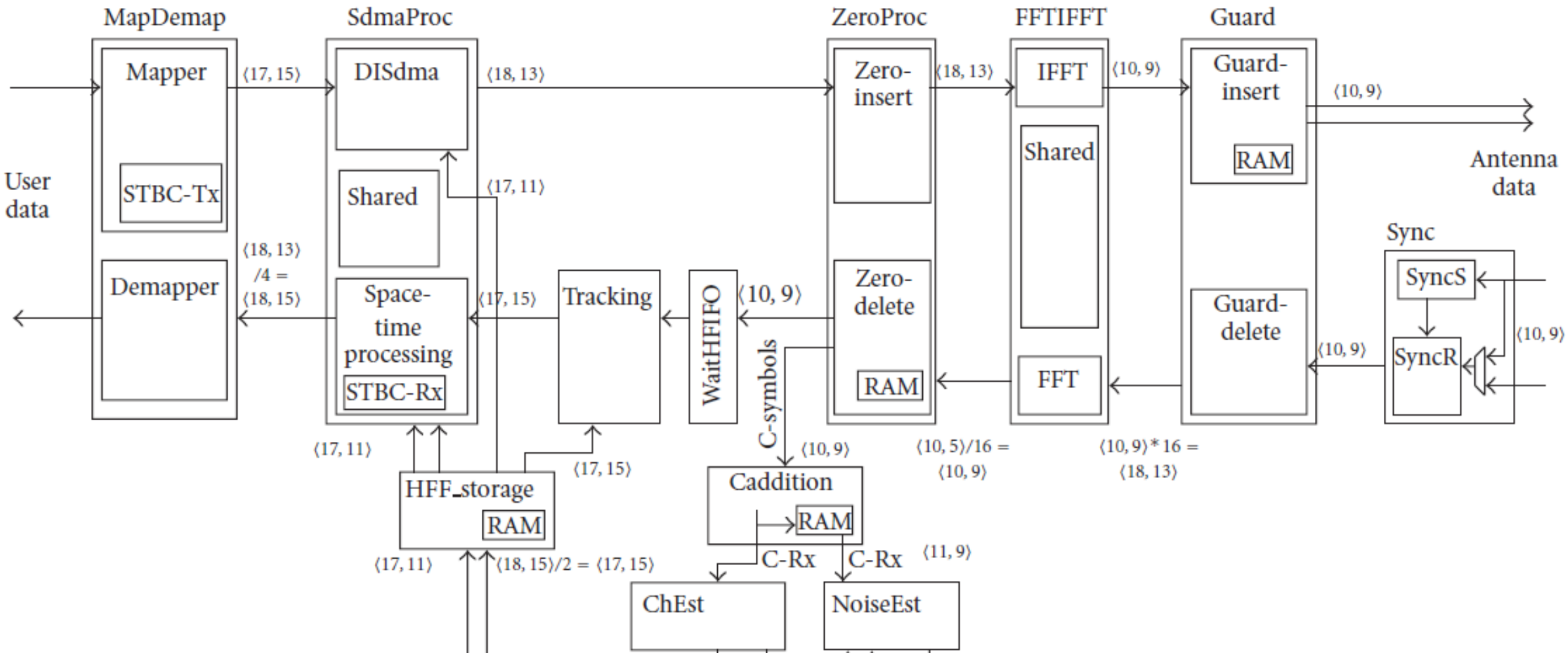
设计空间探索



02.浮点转定点量化——浮点到定点的转化方法



02.浮点转定点量化——量化结构分析



02.浮点转定点量化——最优字长

字长更长

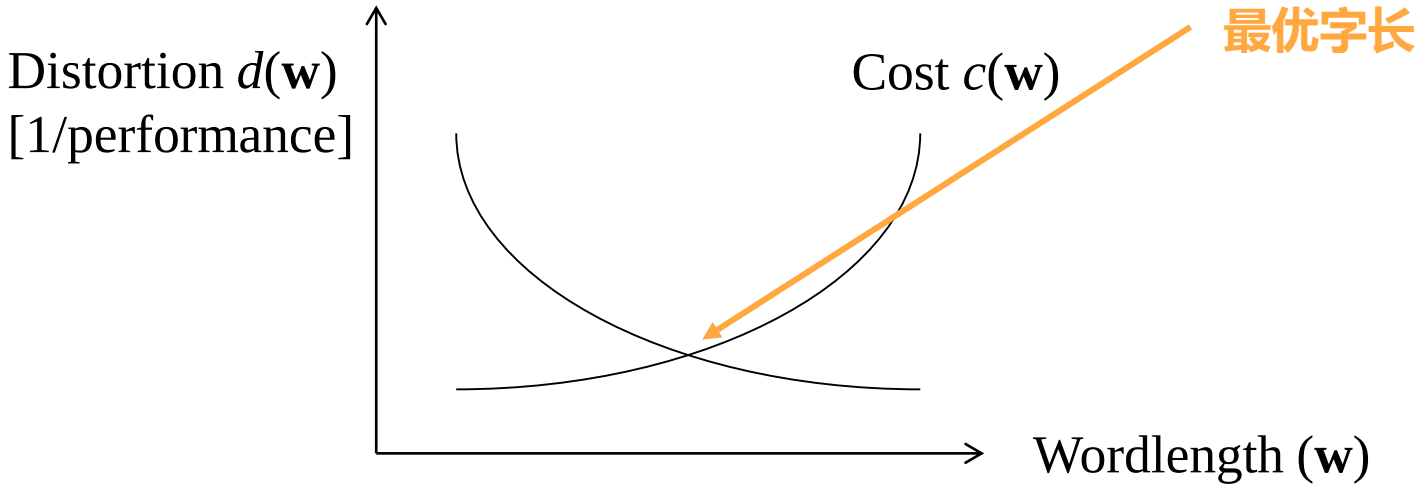
- 可能提升应用程序的性能
- 增大硬件成本

字长更短

- 可能增大量化误差和引起溢出
- 降低硬件成本

最优字长

- 最大化提升应用性能，减少量化误差，降低硬件成本



02.浮点转定点量化——字长优化方法

基于理论分析

- 量化误差模型
- 对于反馈系统，可能发生不稳定性和极循环（Limit cycle）
- 难以开发自适应或非线性系统的分析量化误差模型

基于仿真

- 在观察误差时选择字长
- 重复，直到字长收敛
- 需要很久的仿真时间

目录

03.FIR滤波器设计——数字滤波

数字滤波可以便捷地改变信号的特性

- 常用的滤波器改变信号的频率特性，让一些信号频率通过，阻塞另一些信号频率。通过消除一个或一些频率分量来改变信号的频谱。
  - **低通滤波器(low pass filter)：通低频，阻高频**
  - **高通滤波器(high pass filter)：通高频，阻低频**
  - **带通滤波器(band pass filter)：允许一定频带内的频率通过**
  - **带阻滤波器(band stop filter)：允许一定频带以外的所有频率通过**
- 截至频率(cut-off frequency): 滤波器拐角处的频率

03.FIR滤波器设计——数字滤波的应用

数字滤波：在形形色色的信号中提取所需要的信号,抑制不需要的信号或干扰信号。可应用于：

- **消除信息在传输过程中由于信道不理想所引起的失真**
- **滤除不需要的背景噪声**
- **去除干扰**
- **频带分割， 信号谱的成形**

它广泛地应用于数字通信，雷达，遥感，声纳，语音合成，图象处理，测量与控制，高清晰度电视，多媒体物理学，生物医学，机器人等。

03.FIR滤波器设计——滤波器的实现

$$y[n]=\sum_{k=-\infty}^{\infty}h[k]x[n-k]$$

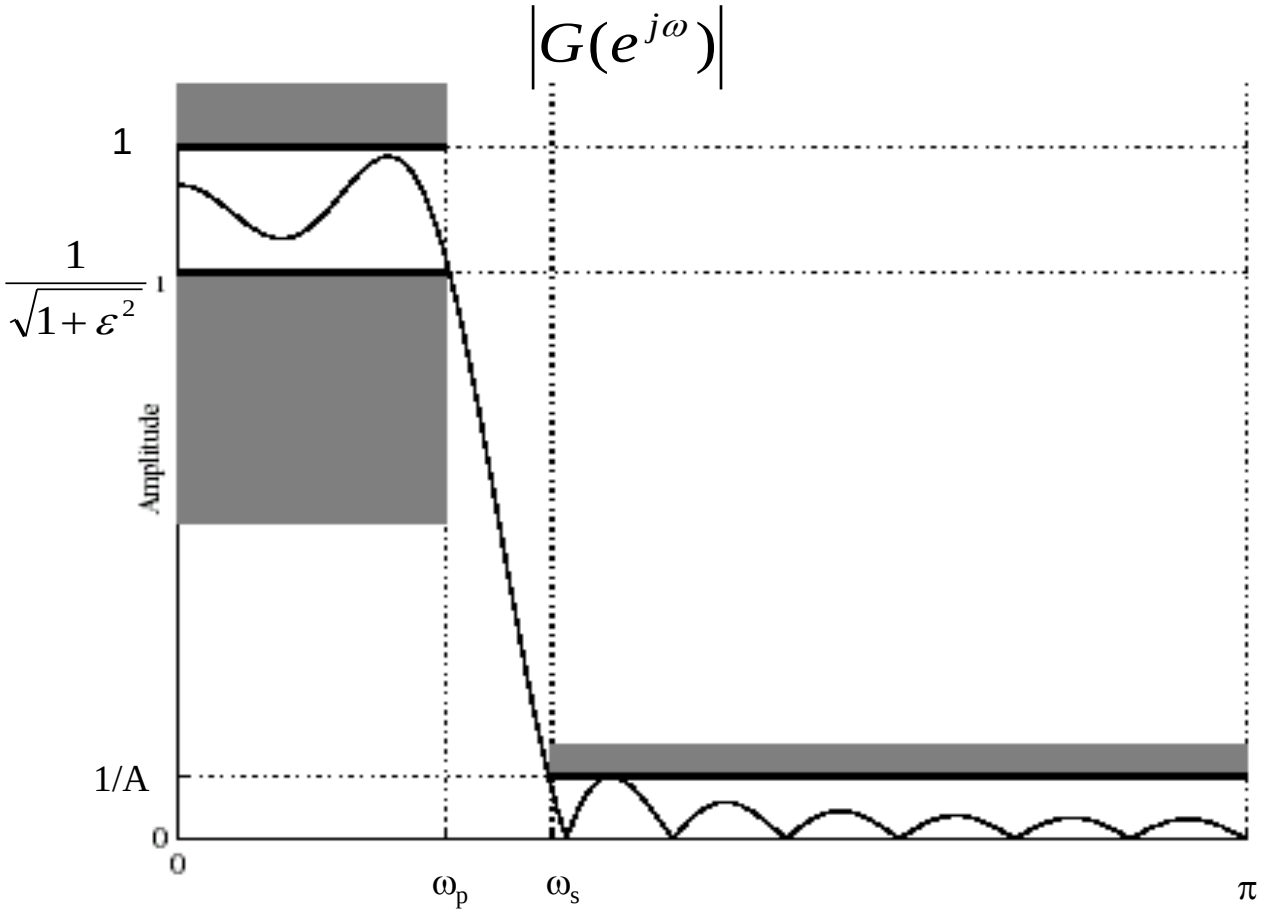
因果的N阶 FIR 滤波器可以表示为： $H(z)=\sum_{k=0}^{N}b_k z^{-k}$ **或** $H(z)=\sum_{k=0}^{N}h[k]z^{-k}$

FIR滤波器总是稳定的

实现滤波器需要考虑的问题：

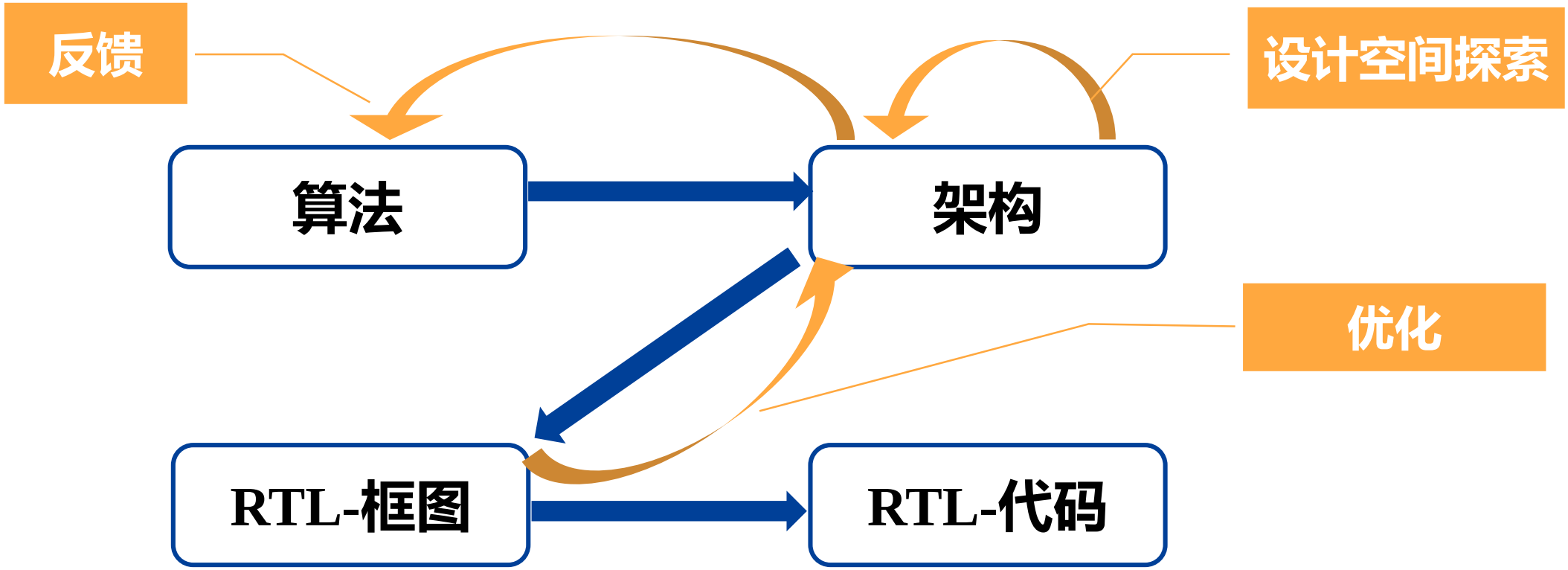
- **软件或硬件**
- **数字系统实现时的有限字长效应**
- **采用合适的结构，使滤波器在有限字长的情况下能提供较好的性能**

03.FIR滤波器设计——低通滤波器指标



归一化的数字低通滤波器幅度响应指标

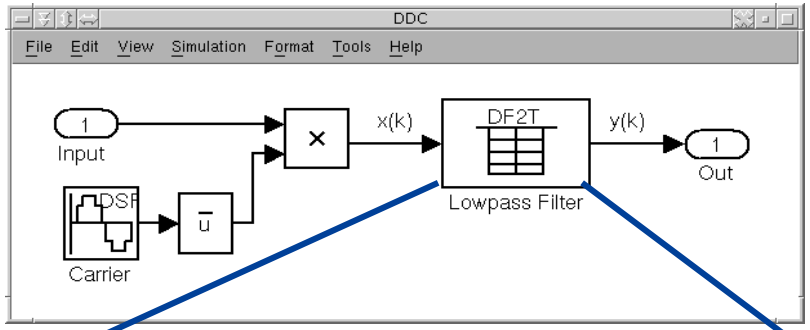
03.FIR滤波器设计—— Where to start?



03.FIR滤波器设计——算法

高层次系统框图：设计内容

- 输入和输出
- 吞吐量
- 算法要求



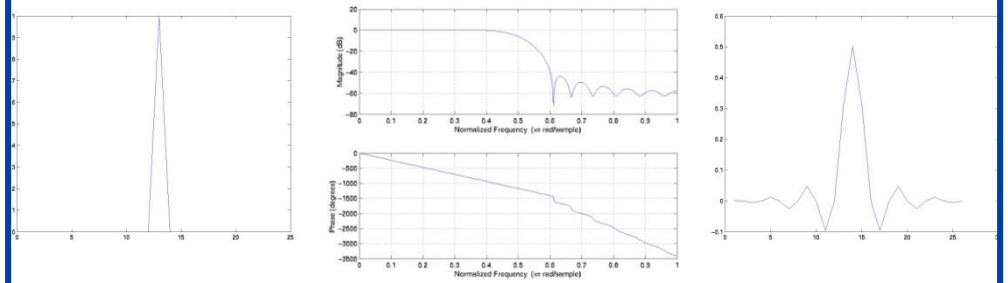
算法描述

1）数学描述

$$y(k)=\sum_{i=0}^{N} b_i x(k-i)$$

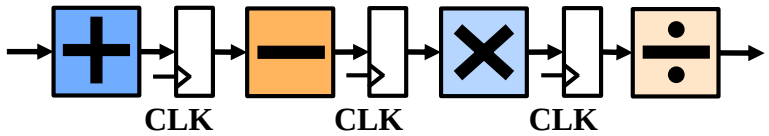
2）性能标准：■准确率 ■优化约束

3）实现限制：■面积 ■速度

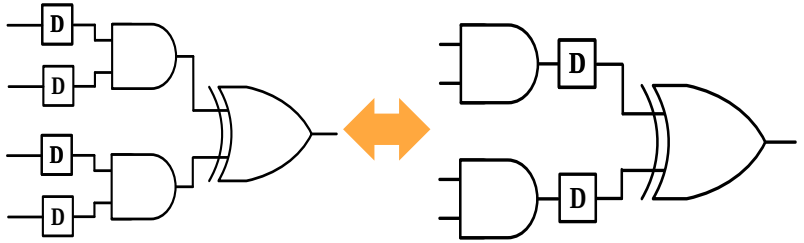


优化方法

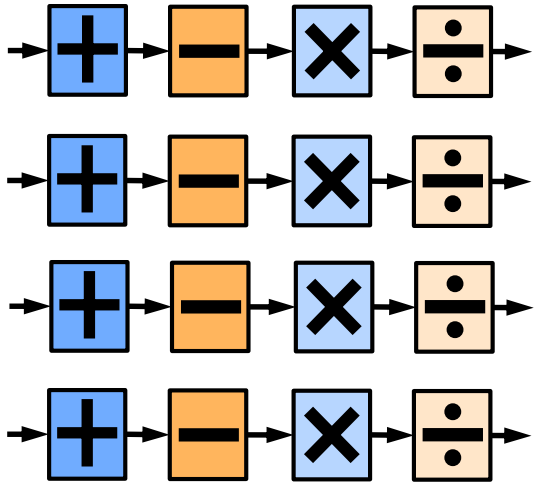
- **为提高数据处理能力**
  - 流水线技术
  - 并行处理
  - 重定时技术
  - 展开



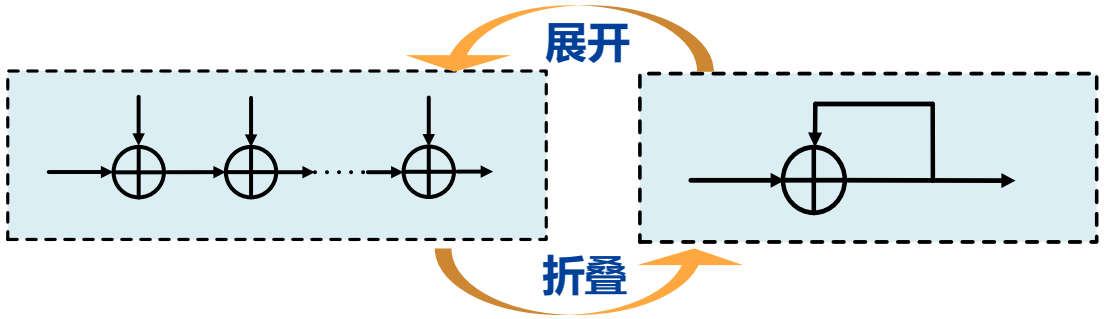
流水线技术



重定时技术


并行处理

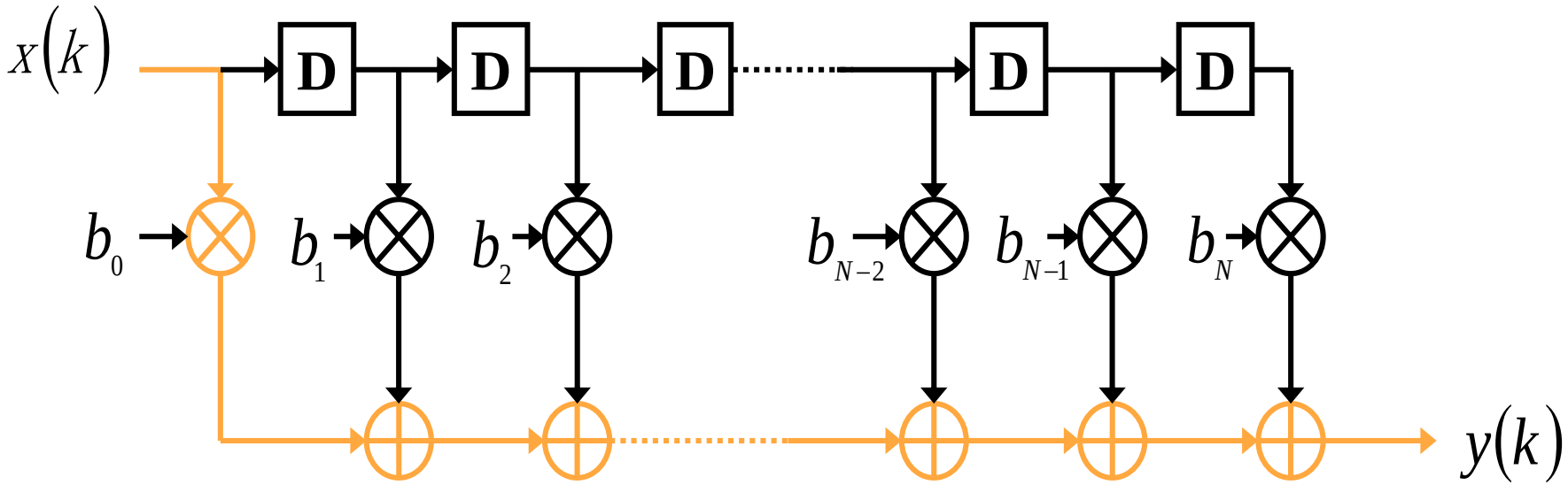
- **为降低面积开销**
  - 折叠



03.FIR滤波器设计——原始架构(1)

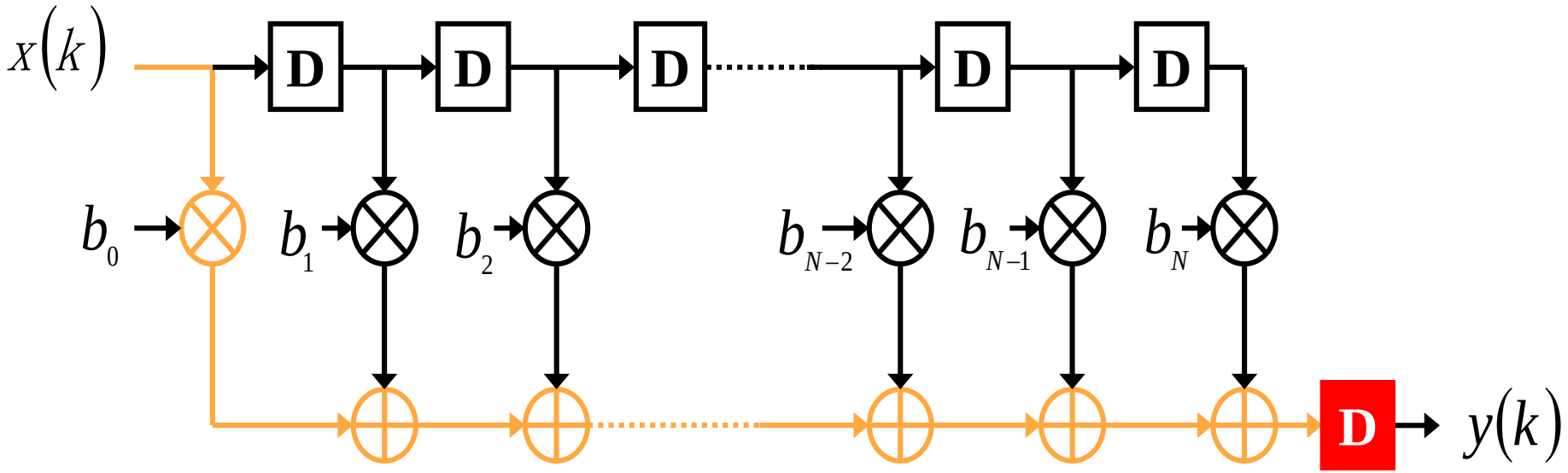
Isomorphic Architecture

- 直接根据算法进行实现



流水线技术

- 改进时序

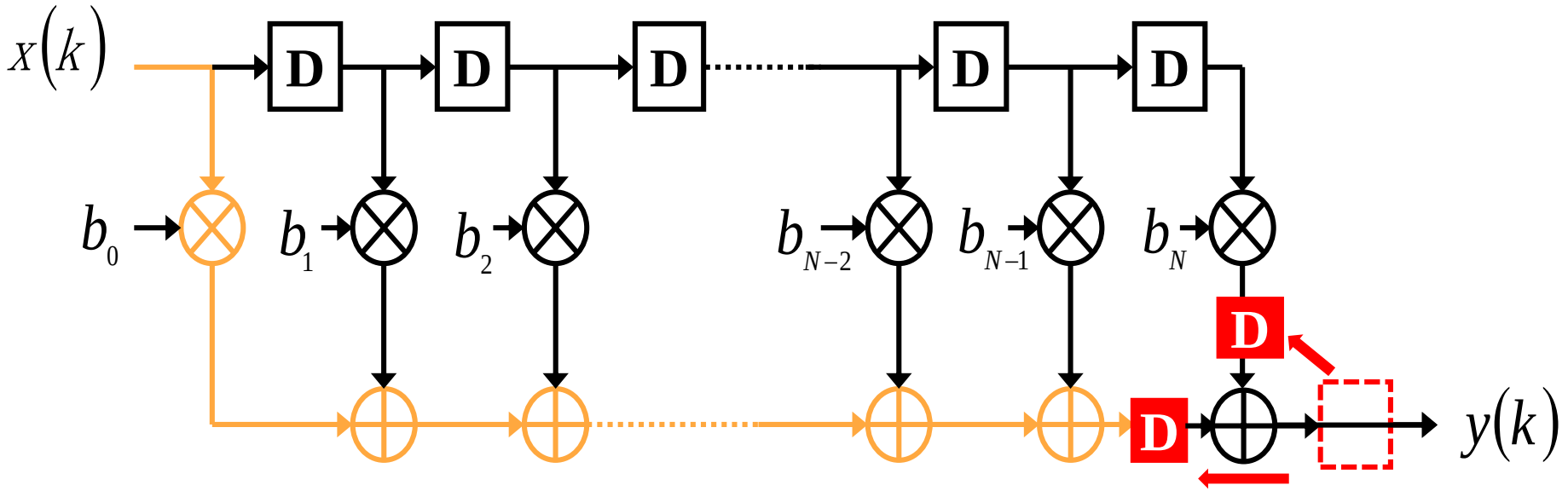


- 在输入和输出处插入寄存器：增大延迟

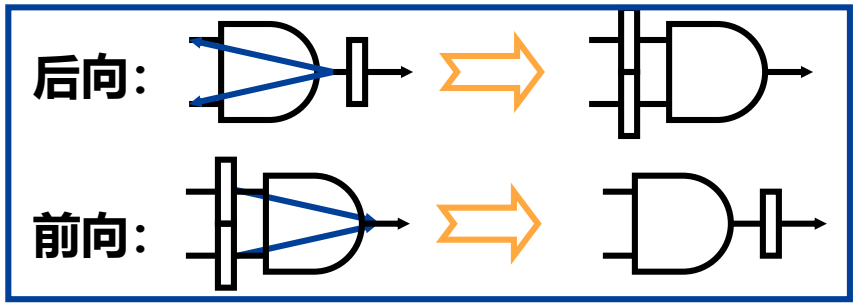
03.FIR滤波器设计——架构（2）

重定时技术

- 改进时序



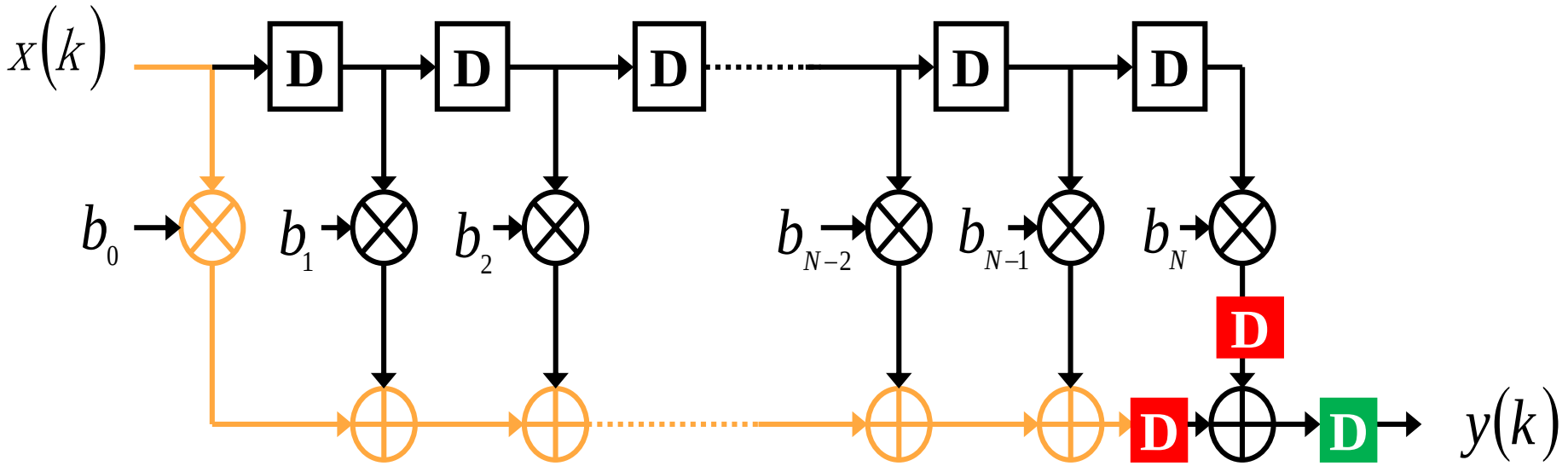
- 在输入和输出处插入寄存器：增大延迟
- 重定时：移动寄存器，不改变功能



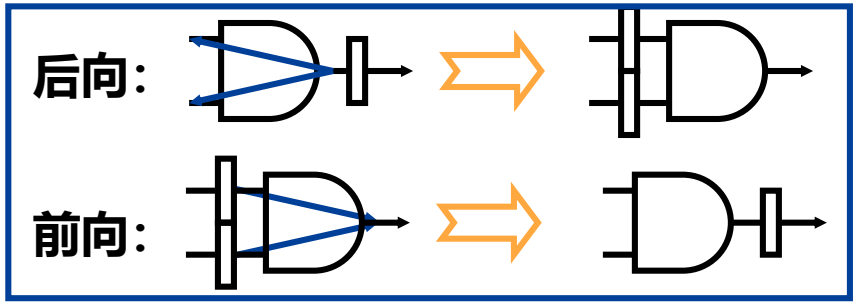
03.FIR滤波器设计——架构（2）

重定时技术

- 改进时序



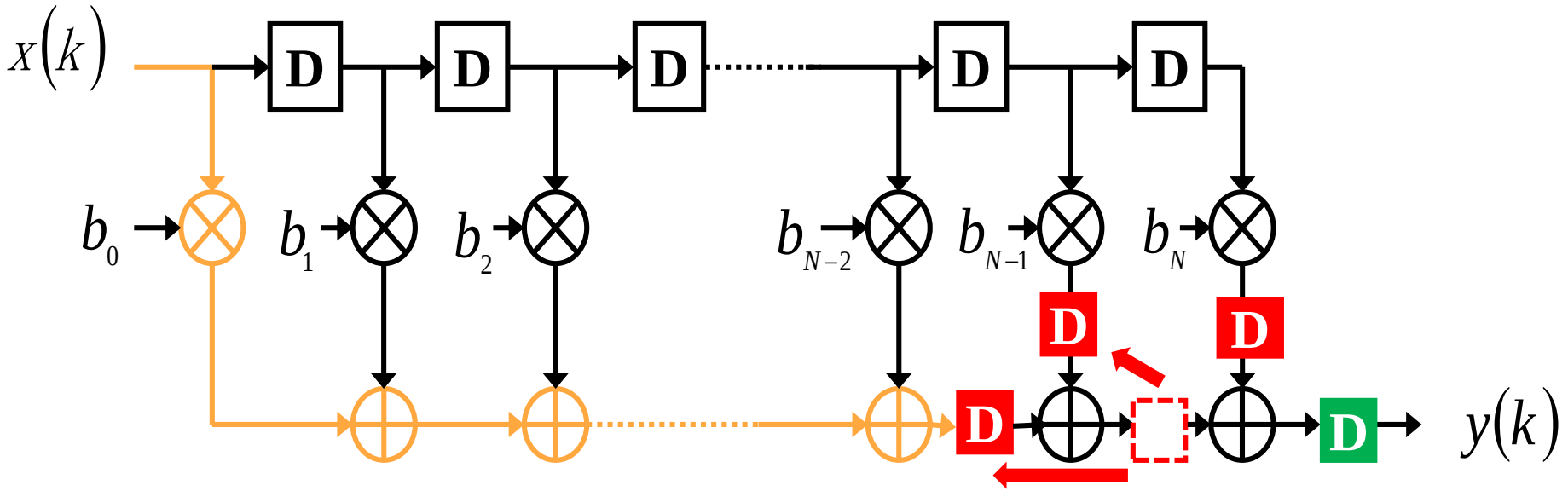
- 在输入和输出处插入寄存器：增大延迟
- 重定时：移动寄存器，不改变功能



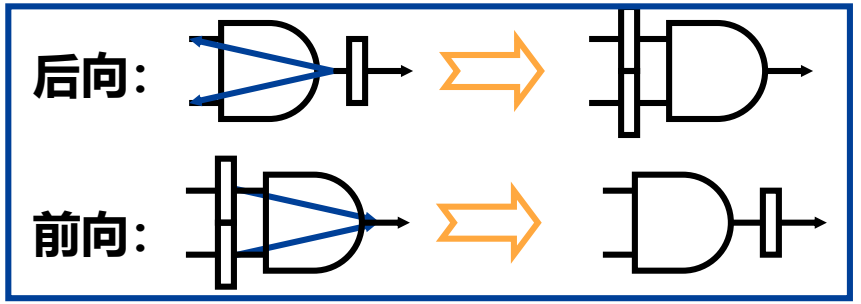
03.FIR滤波器设计——架构（2）

重定时技术

- 改进时序



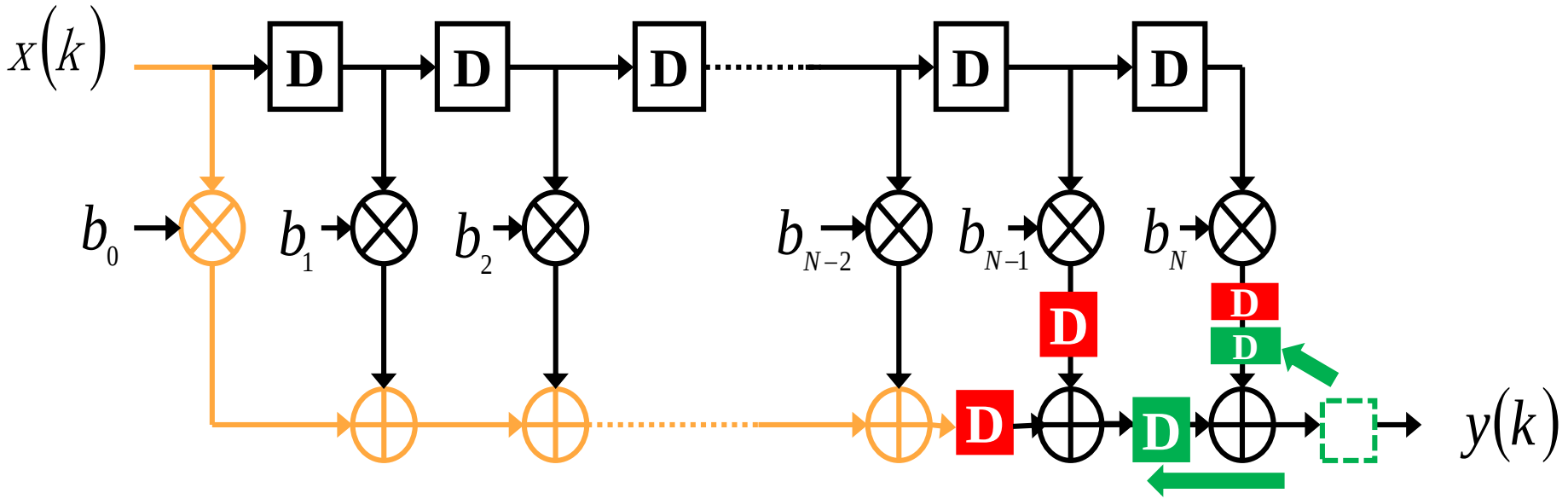
- 在输入和输出处插入寄存器：增大延迟
- 重定时：移动寄存器，不改变功能



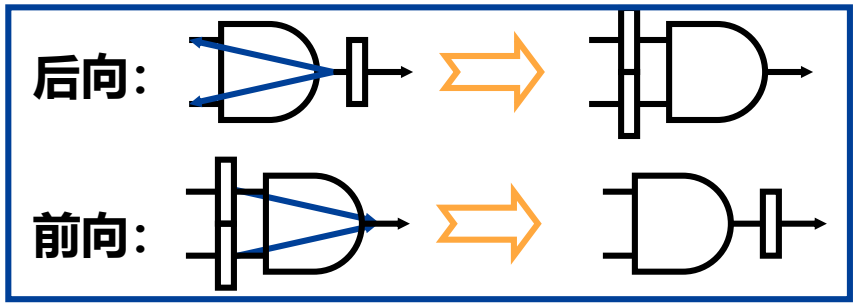
03.FIR滤波器设计——架构（2）

重定时技术

- 改进时序



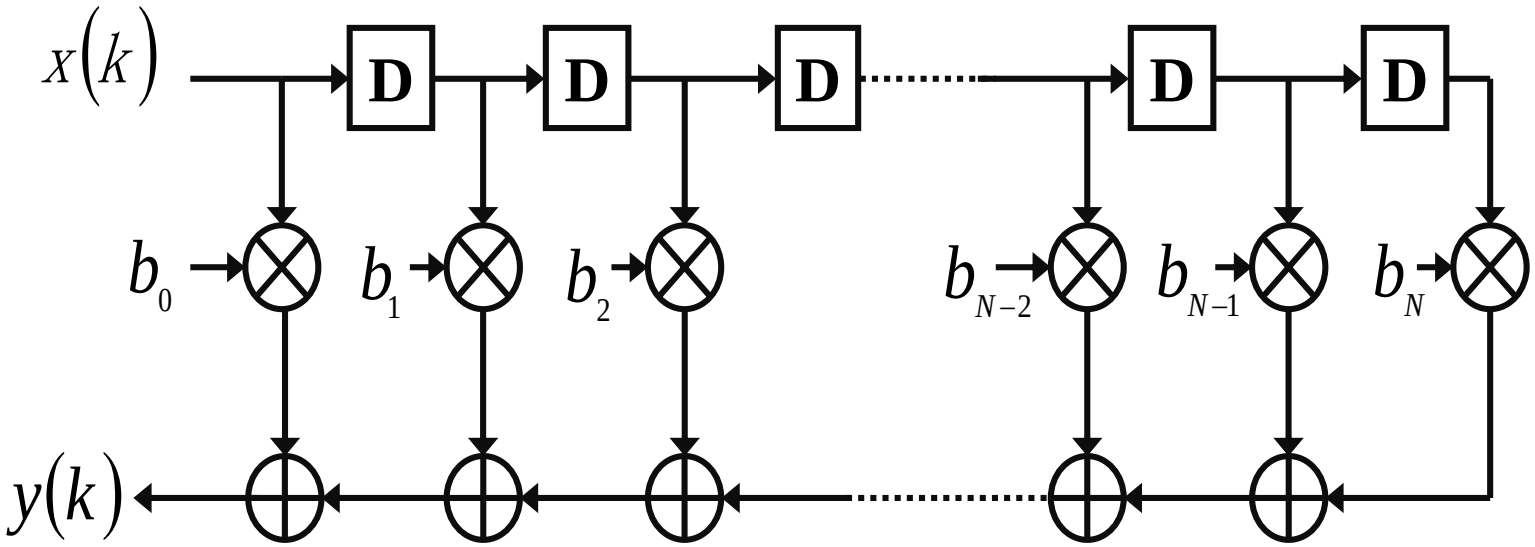
- 在输入和输出处插入寄存器：增大延迟
- 重定时：移动寄存器，不改变功能



03.FIR滤波器设计——架构（3）

重定时技术

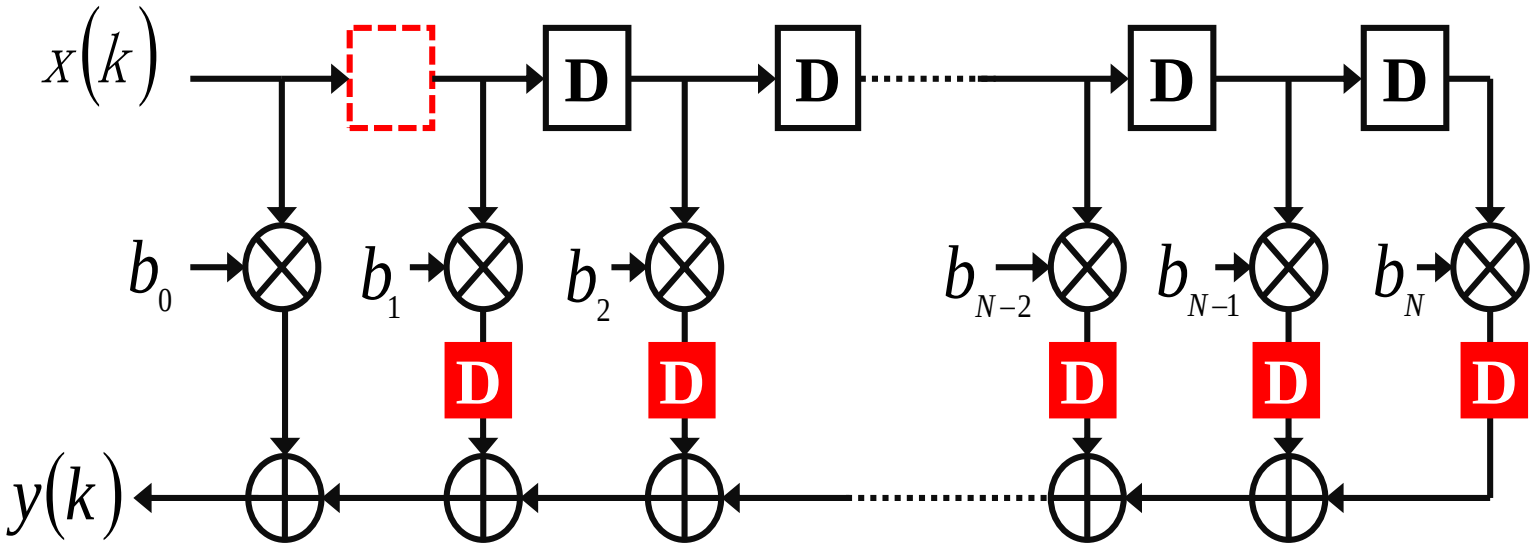
- **优化：**



- **加法器链反向**

重定时技术

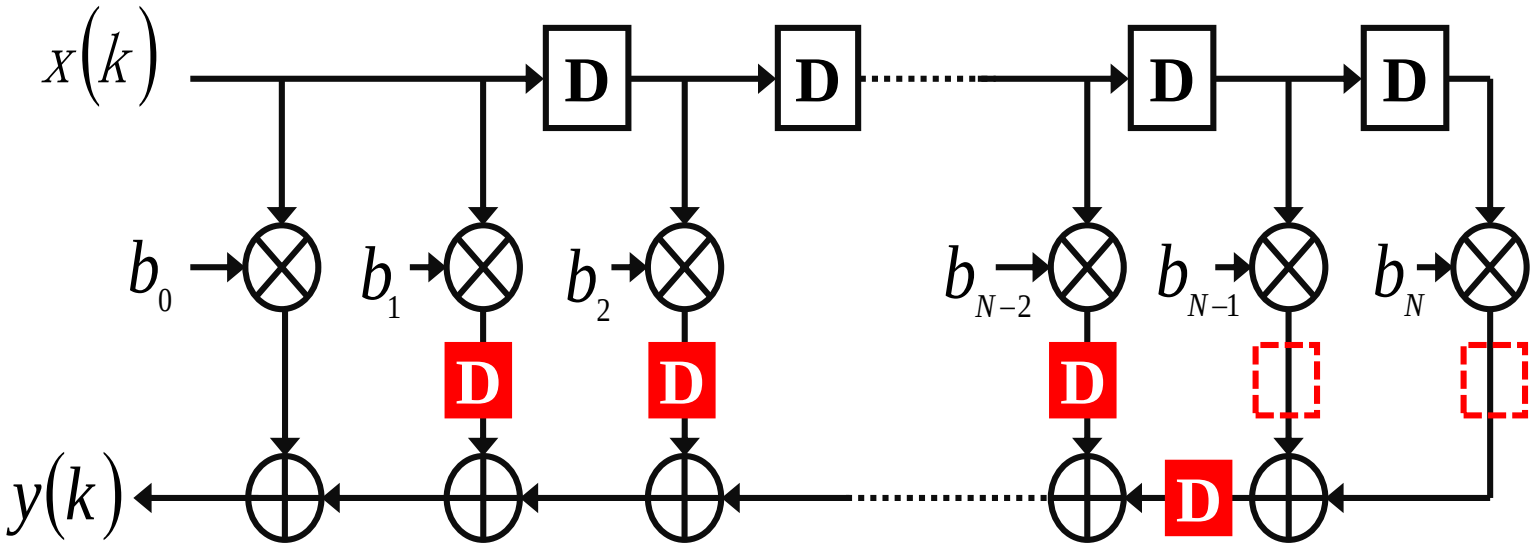
- 优化：



- 加法器链反向
- 进行重定时

重定时技术

- 优化：

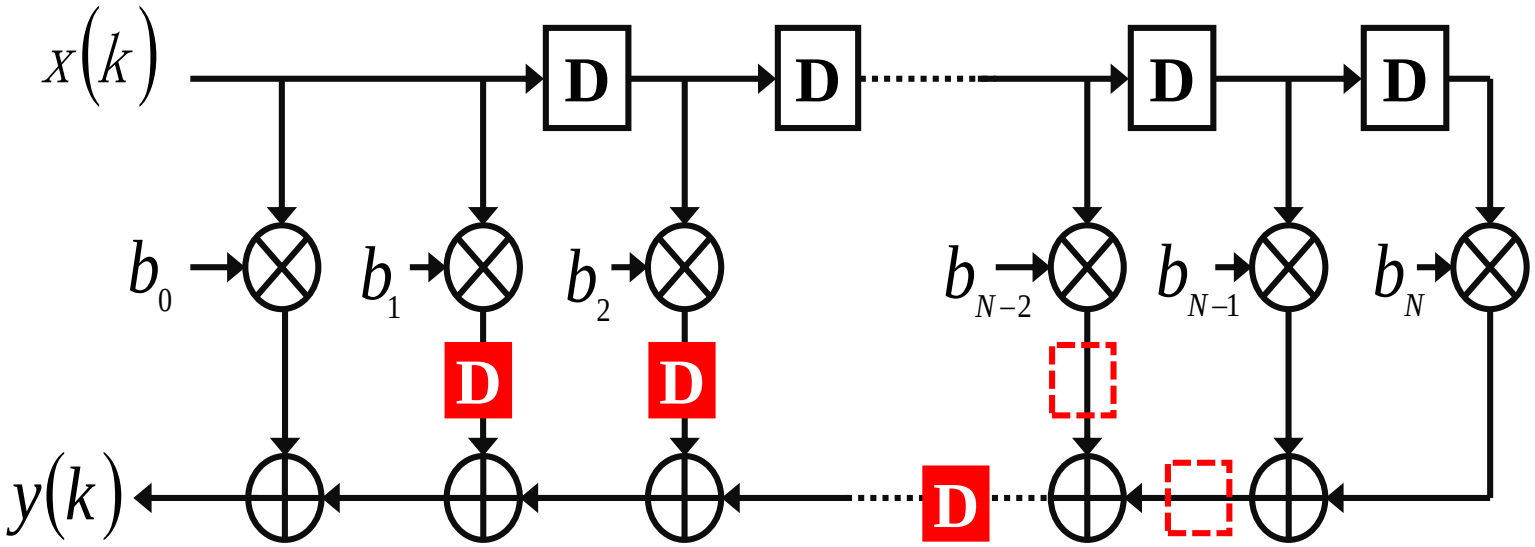


- 加法器链反向
- 进行重定时

03.FIR滤波器设计——架构（3）

重定时技术

- 优化：

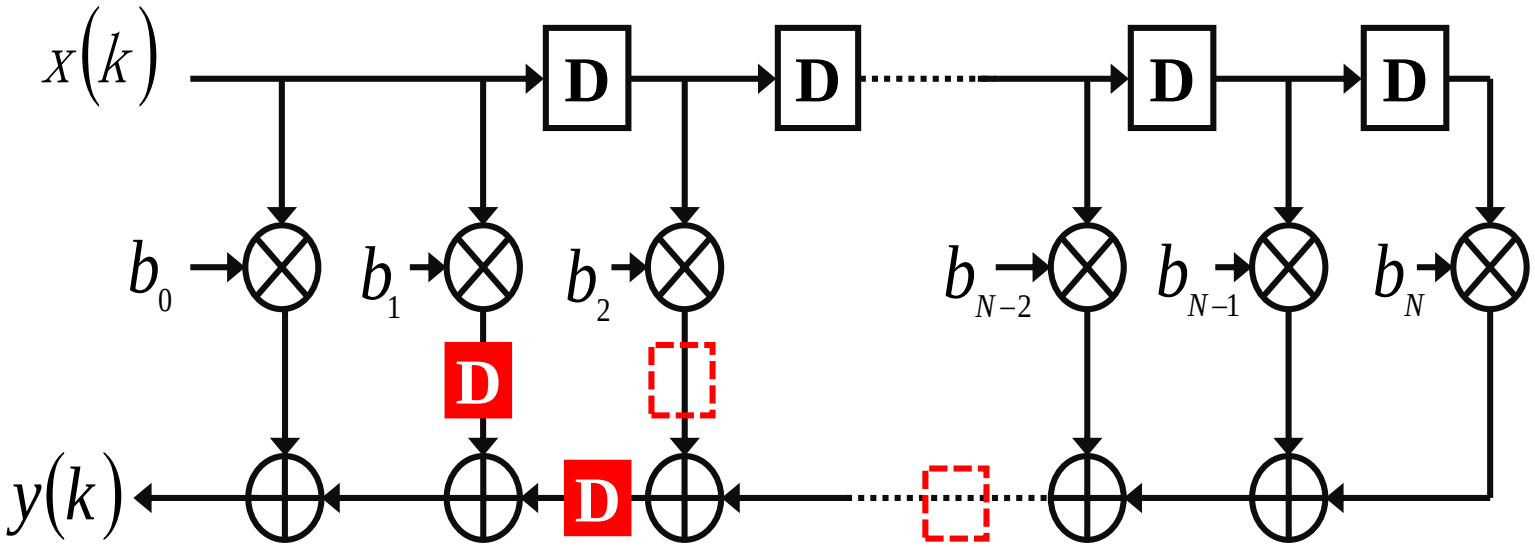


- 加法器链反向
- 进行重定时

03.FIR滤波器设计——架构（3）

重定时技术

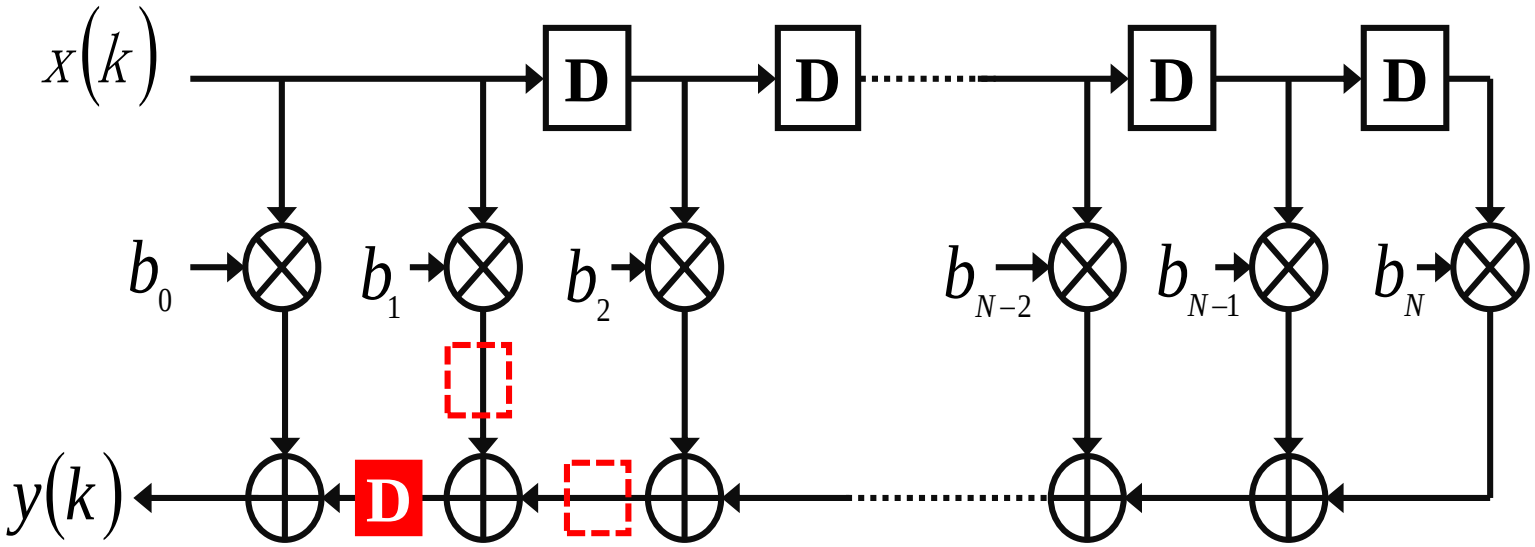
- 优化：



- 加法器链反向
- 进行重定时

重定时技术

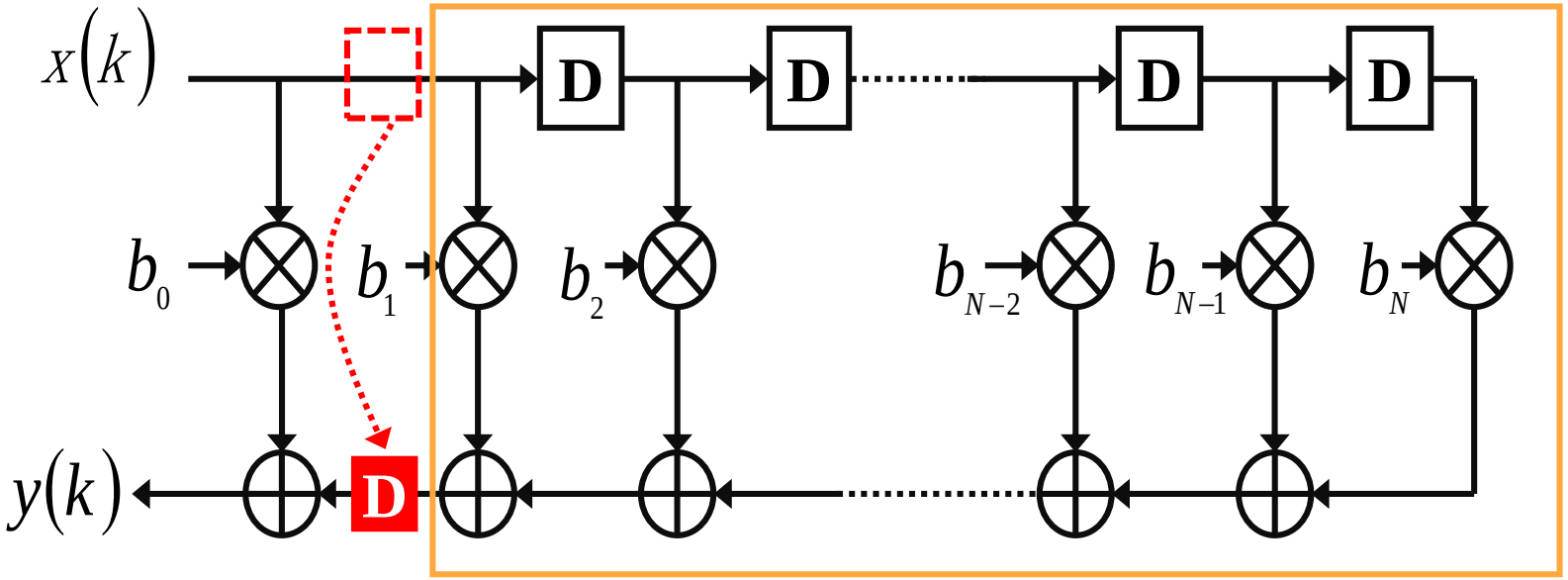
- 优化：



- 加法器链反向
- 进行重定时

重定时技术

- 优化：

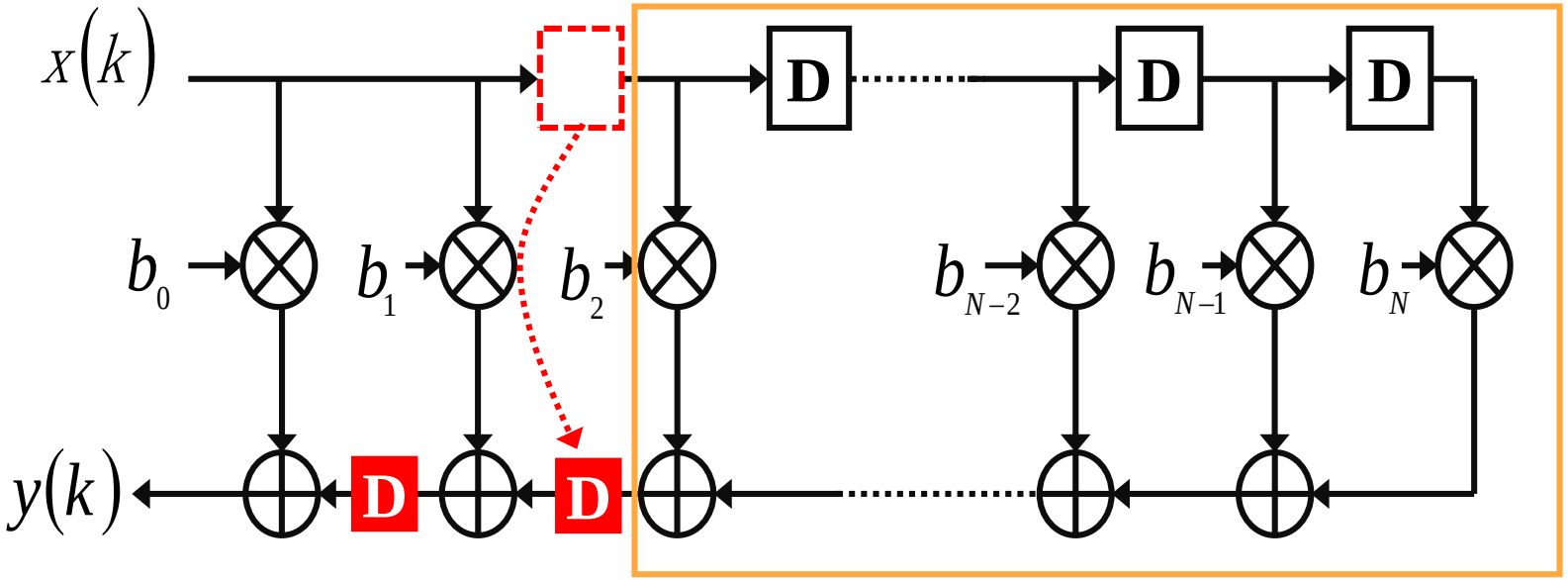


- 加法器链反向
- 进行重定时

03.FIR滤波器设计——架构（3）

重定时技术

- 优化：

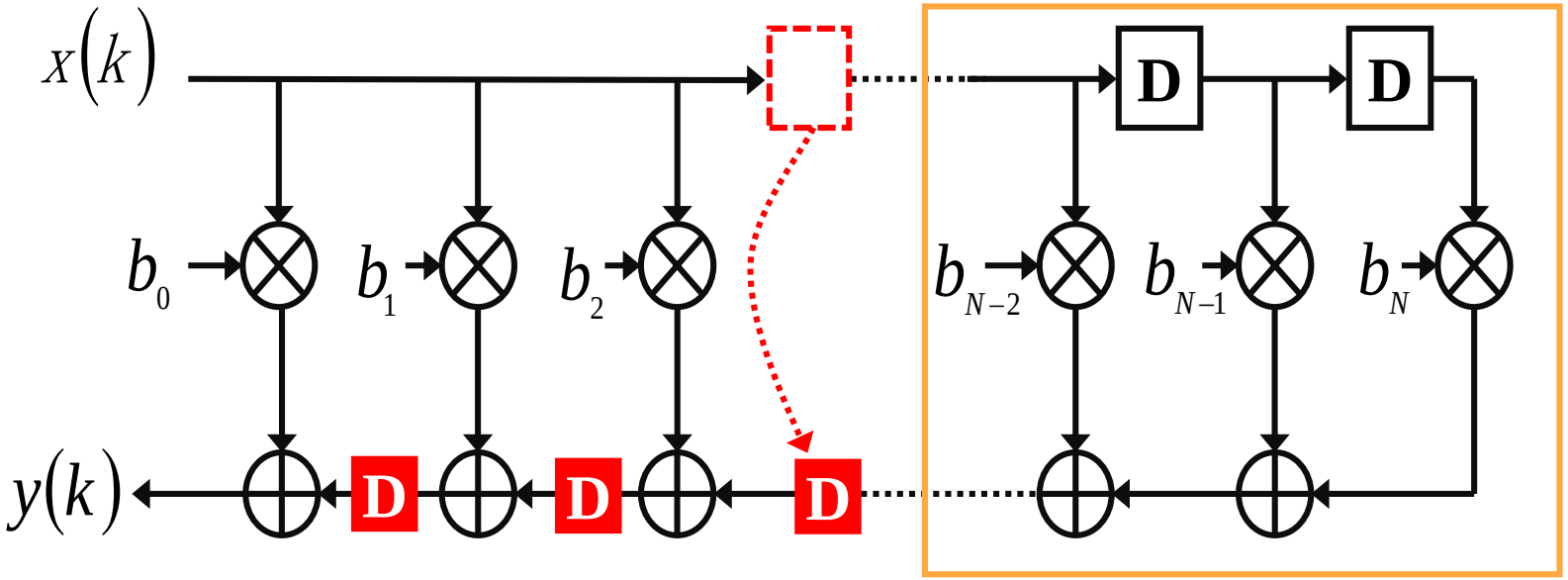


- 加法器链反向
- 进行重定时

03.FIR滤波器设计——架构（3）

重定时技术

- 优化：

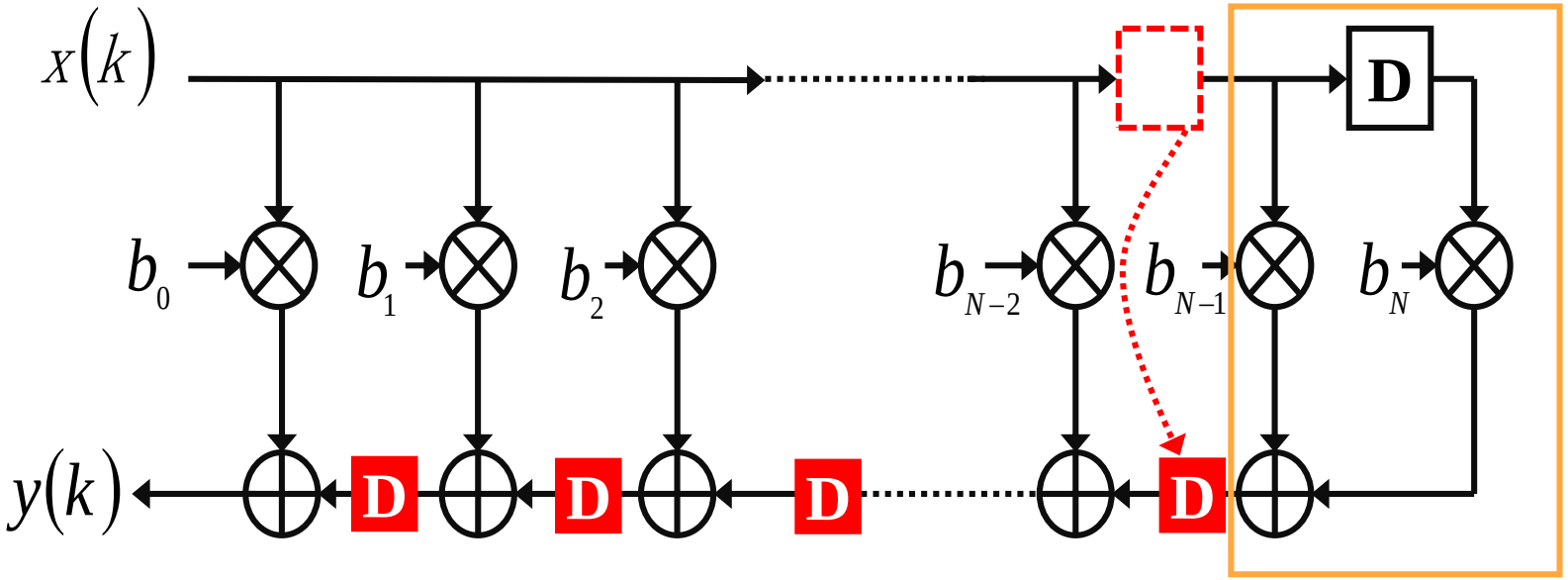


- 加法器链反向
- 进行重定时

03.FIR滤波器设计——架构（3）

重定时技术

- 优化：

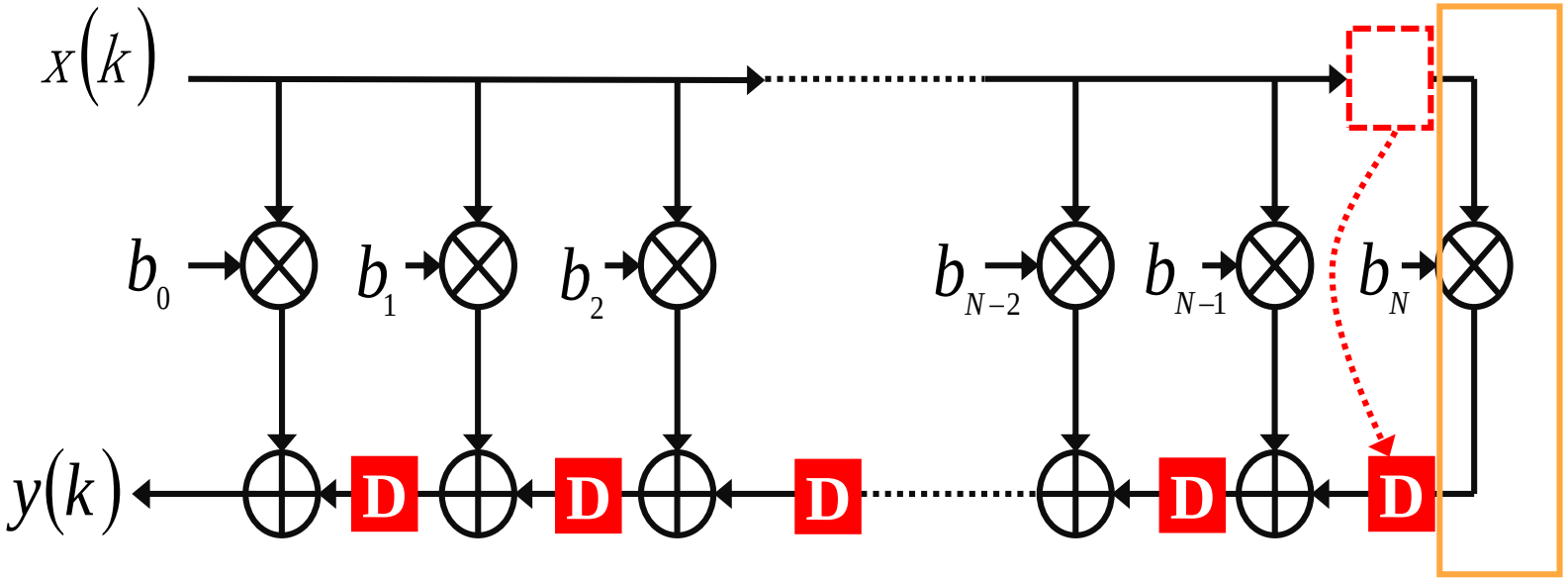


- 加法器链反向
- 进行重定时

03.FIR滤波器设计——架构（3）

重定时技术

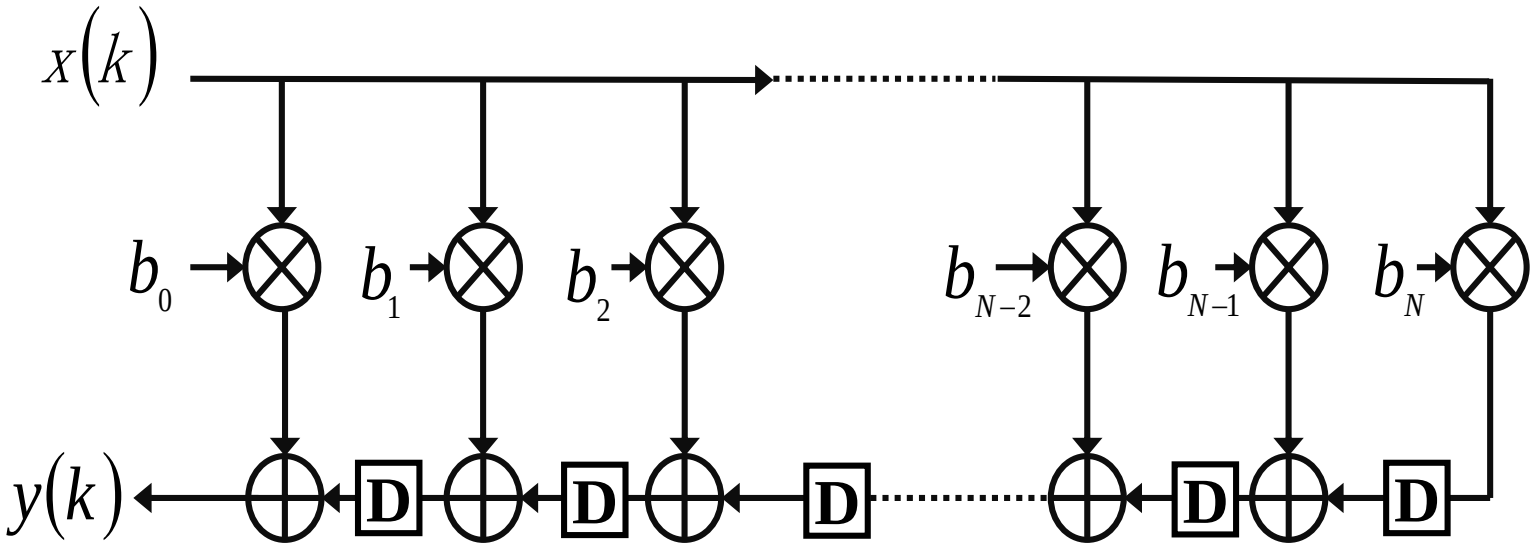
- 优化：



- 加法器链反向
- 进行重定时

重定时技术

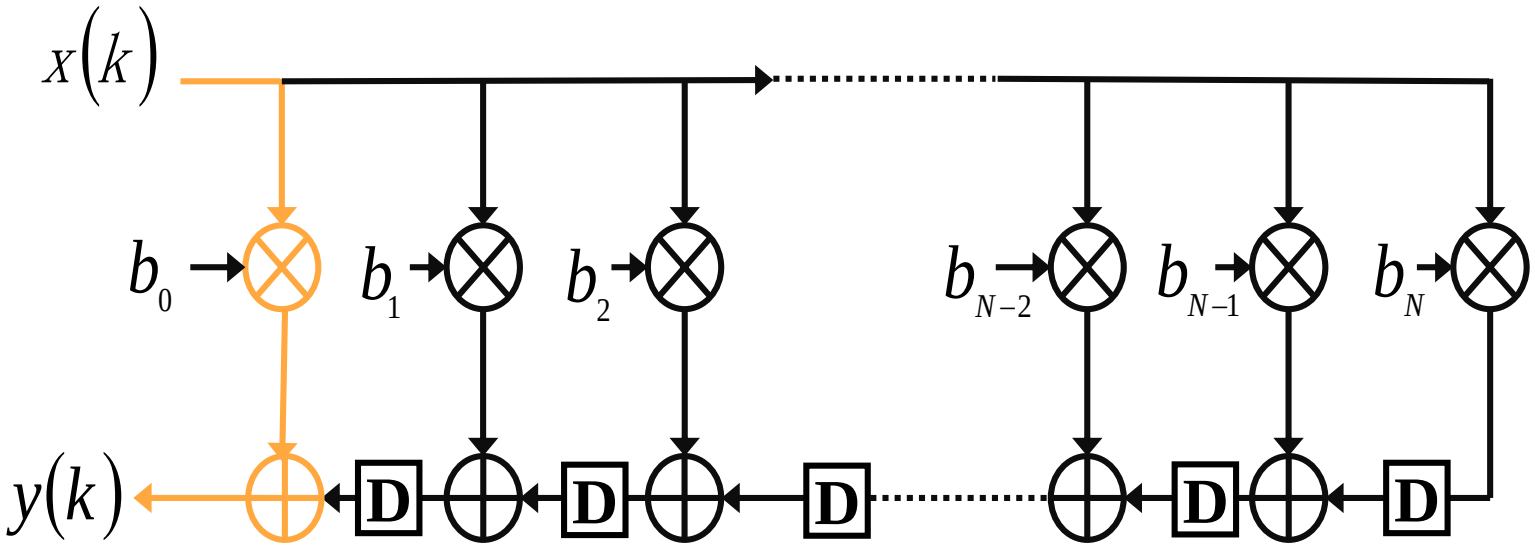
- 优化：



- 加法器链反向
- 进行重定时

重定时技术

- 优化：

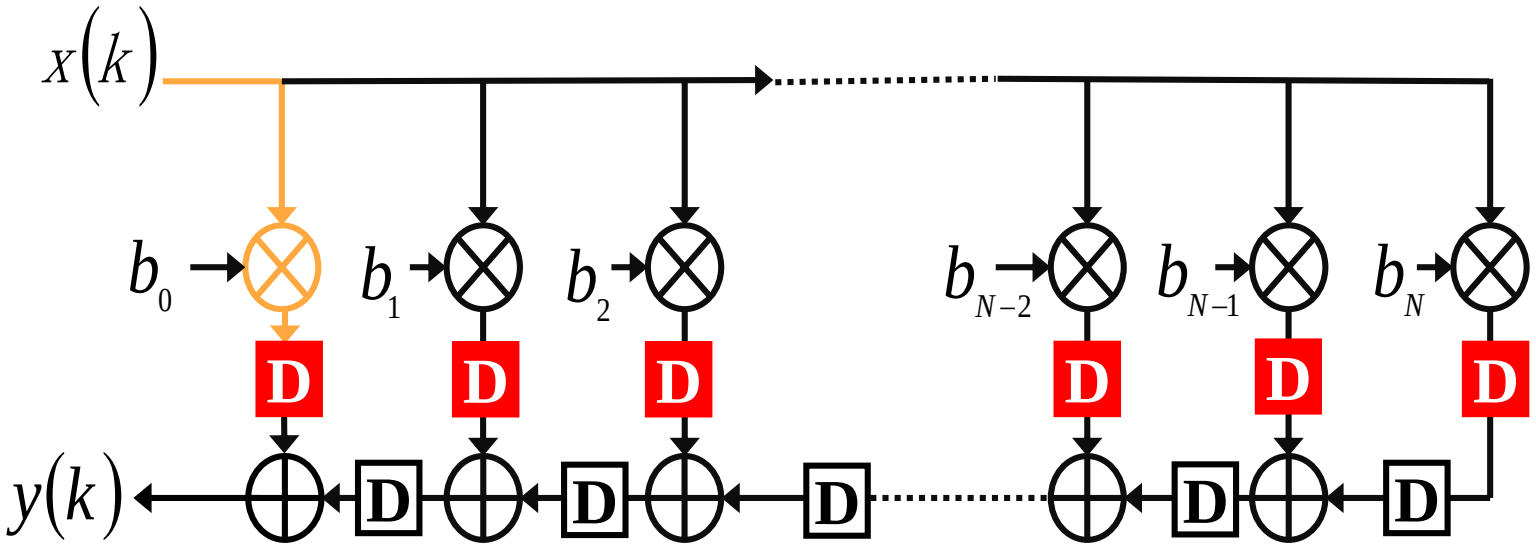


- 加法器链反向
- 进行重定时

03.FIR滤波器设计——架构（4）

重定时后：进一步采用流水线技术

- 对重定时后的电路加入一级流水线

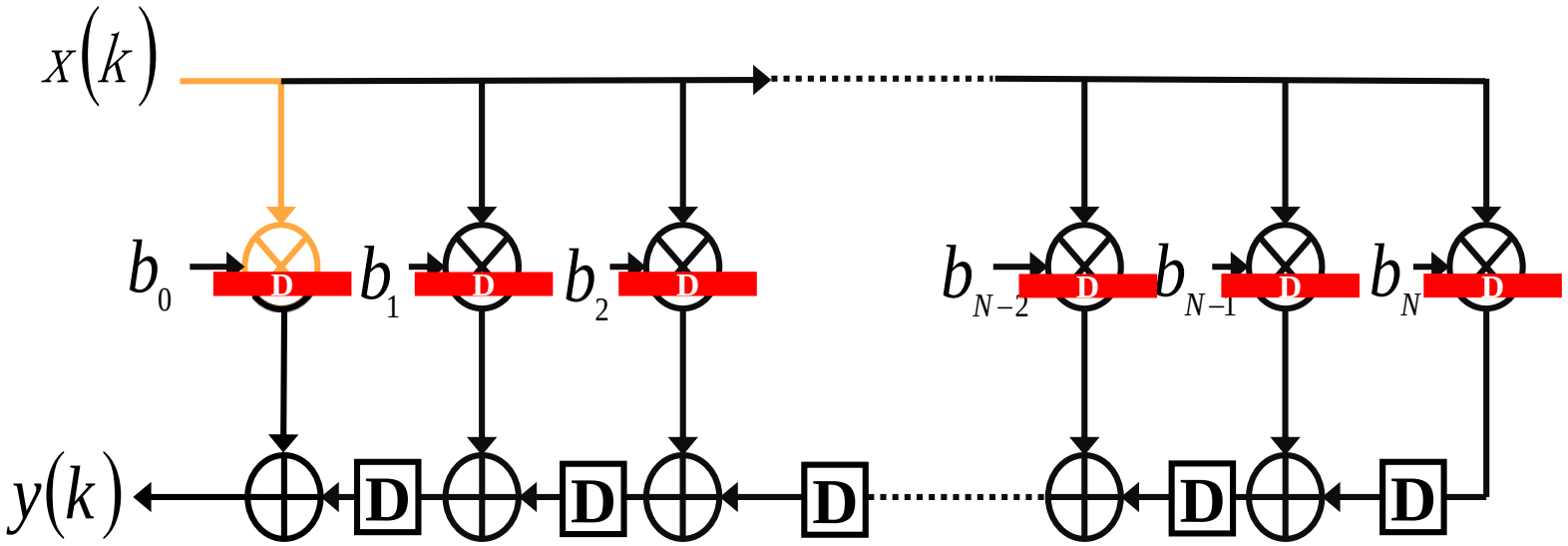


- 最长路径是由乘法器造成的
- 不平衡：从输入到第一级流水线的延时>从第一级到第二级流水线的延时

03.FIR滤波器设计——架构（5）

重定时后：进一步采用流水线技术

- 对重定时后的电路加入一级流水线



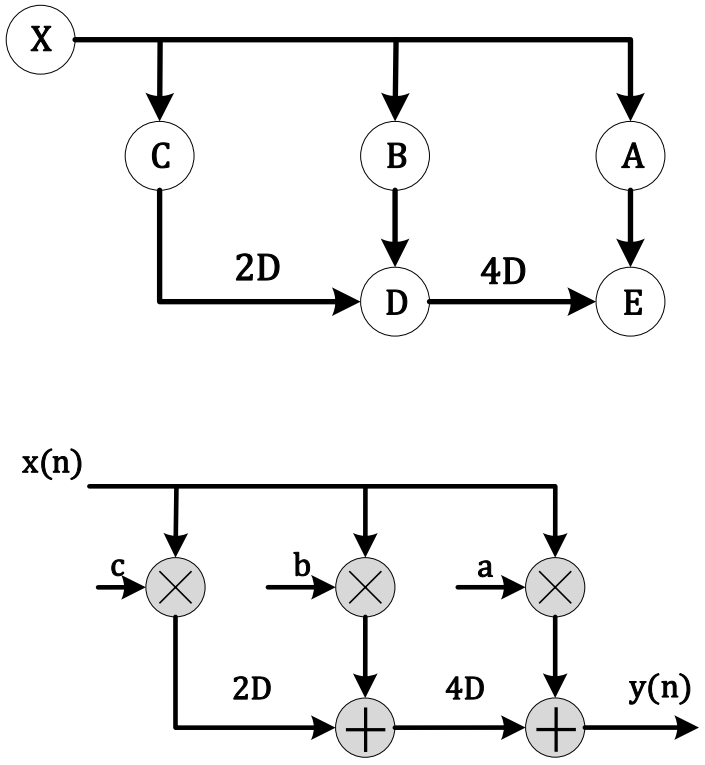
- 将流水线寄存器移入乘法器中
  - 流水级间延迟平衡
  - 改善了时序

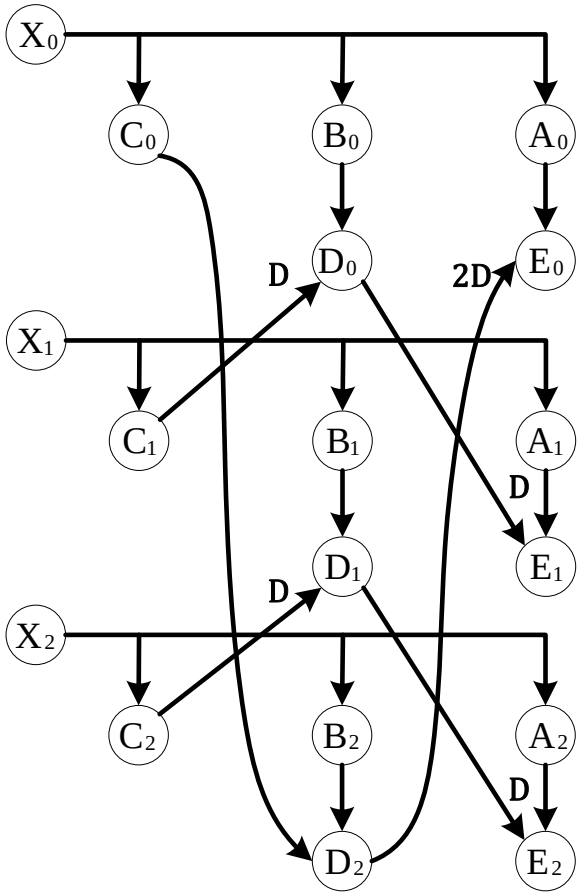
$$T_{clock}=(T_{add}+T_{mult})/2+T_{reg}$$

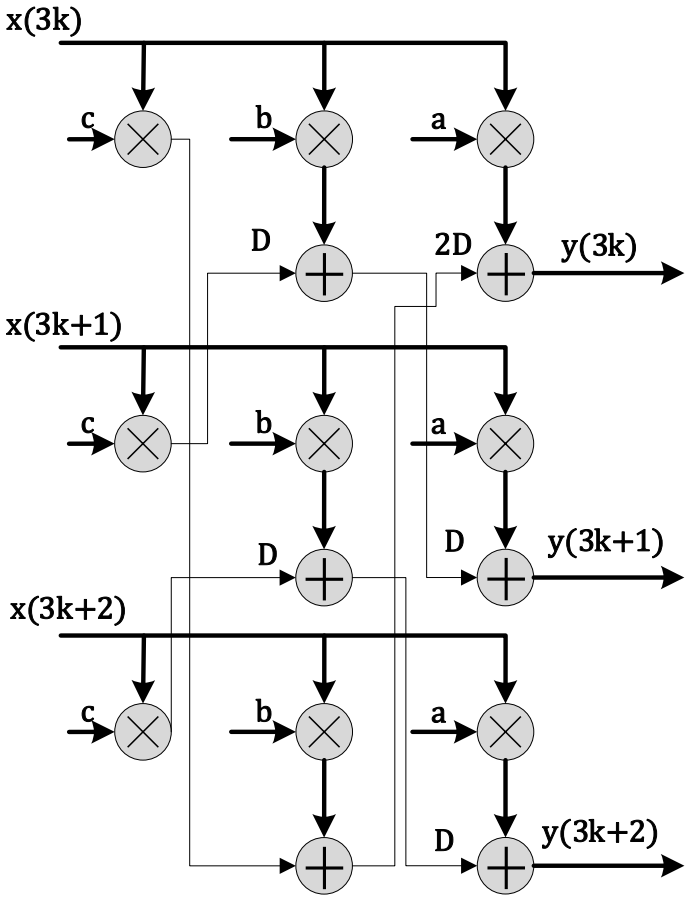
03.FIR滤波器设计——架构（6）

展开

- Word-level parallel processing



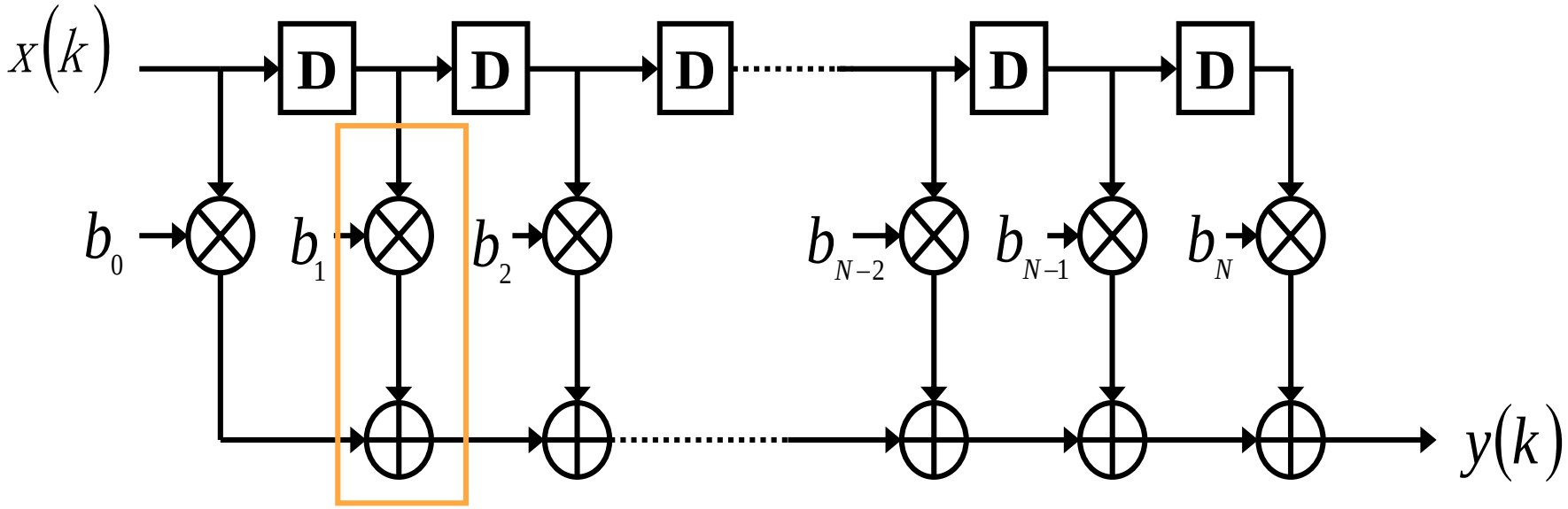




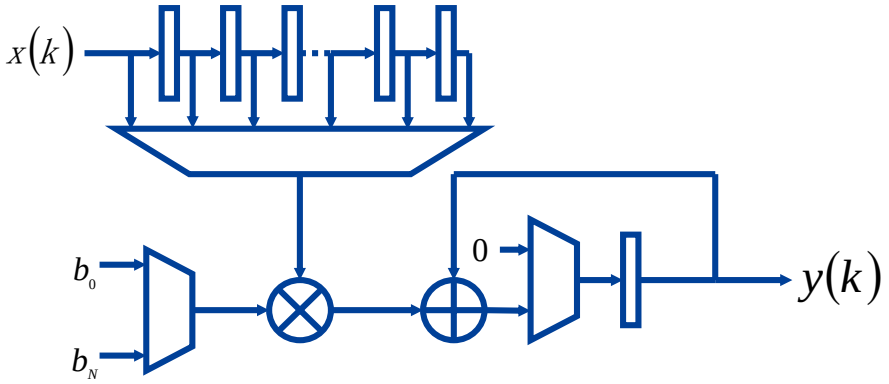
03.FIR滤波器设计——架构（7）

折叠

- 硬件复用



- 确定有规律、可重用的硬件组件
- 添加控制逻辑、多路复用器、存储元件
- 增大 Cycles/Sample



03.FIR滤波器设计——折叠的实现

折叠的实现

- 以31阶FIR滤波器为例

$$y(n)=\sum_{k=0}^{31}h(k)x(n-k)$$ 32个系数$h(k)$已知

- 最少需要的资源：一个乘法器和1个加法器

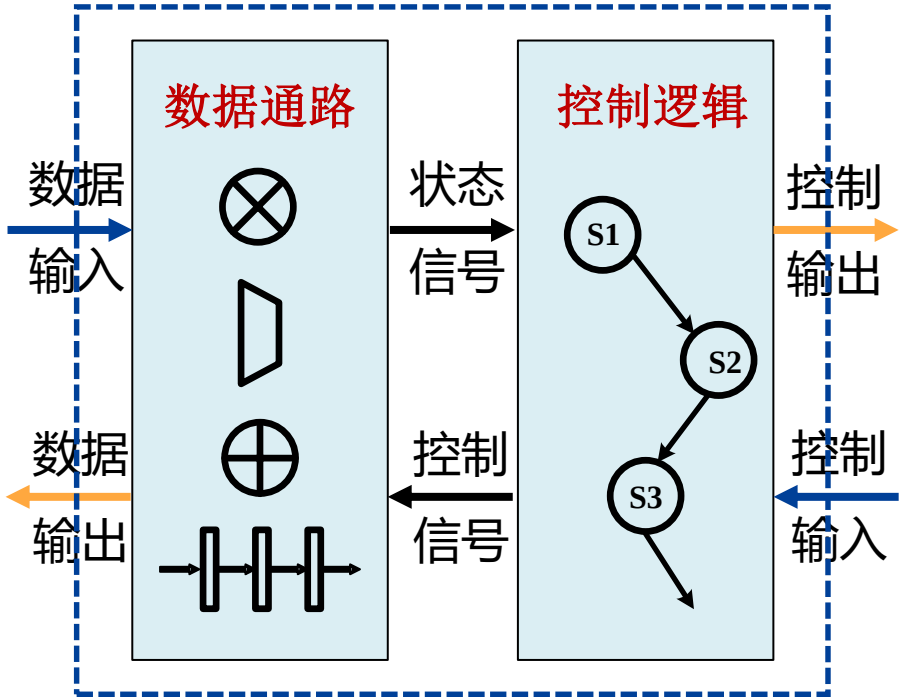
- 电路设计包括两个步骤：

数据通路

- 是算法到结构的映射
- 包括运算单元和存储单元

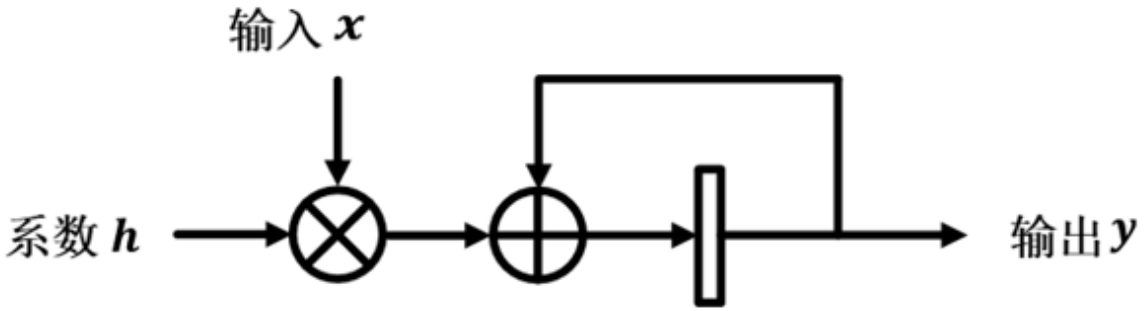
控制逻辑

- 决定何时进行何种运算
- 可通过有限状态机实现



03.FIR滤波器设计——折叠的实现

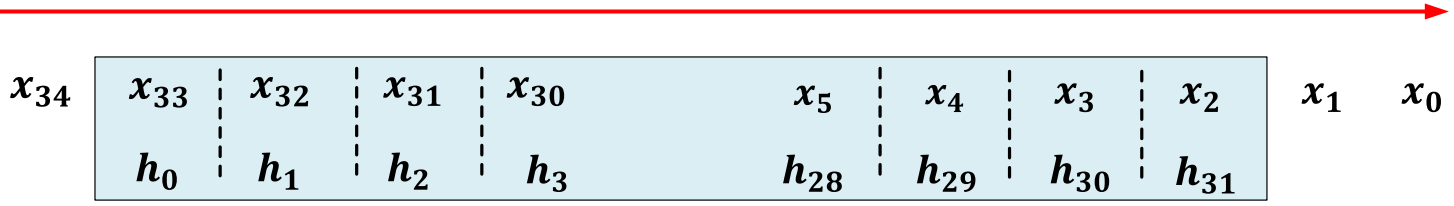
运算单元：一个加法器和一个乘法器



$$y(n)=\sum_{k=0}^{31}h(k)x(n-k)$$

存储单元

能否复用?



$$y_{31}=h_0x_{31}+h_1x_{30}+\cdots+h_{30}x_1+h_{31}x_0$$
$$y_{32}=h_0x_{32}+h_1x_{31}+\cdots+h_{30}x_2+h_{31}x_1$$
$$y_{33}=h_0x_{33}+h_1x_{32}+\cdots+h_{30}x_3+h_{31}x_2$$

无法复用输出结果

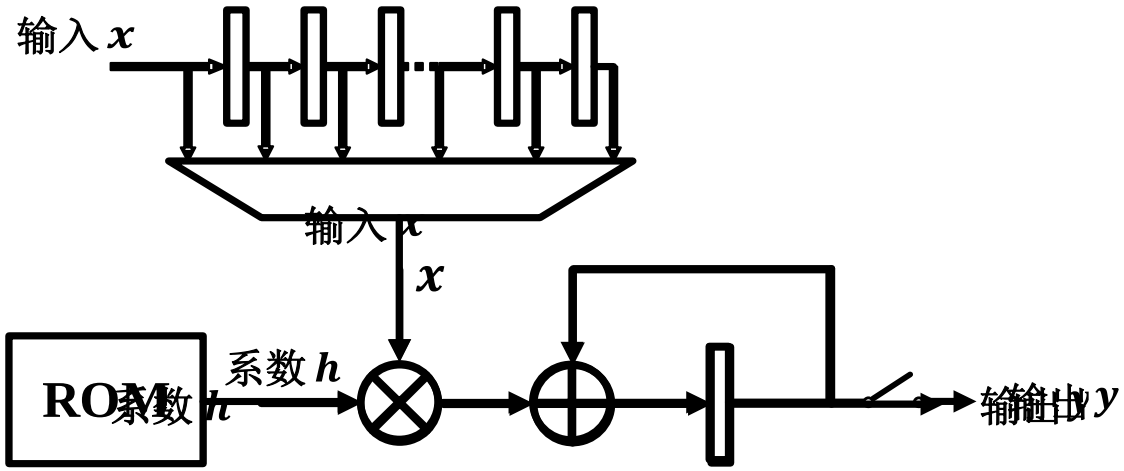
03.FIR滤波器设计——折叠的实现

存储单元

- 输入x：需要存下所有x；可以**用移位寄存器实现**
- 系数h：已知且固定，可用**预先存于ROM中**
- 输出y：只需在正确时刻输出，**不需要存储**

$$y(n)=\sum_{k=0}^{31}h(k)x(n-k)$$

如何设计控制信号?



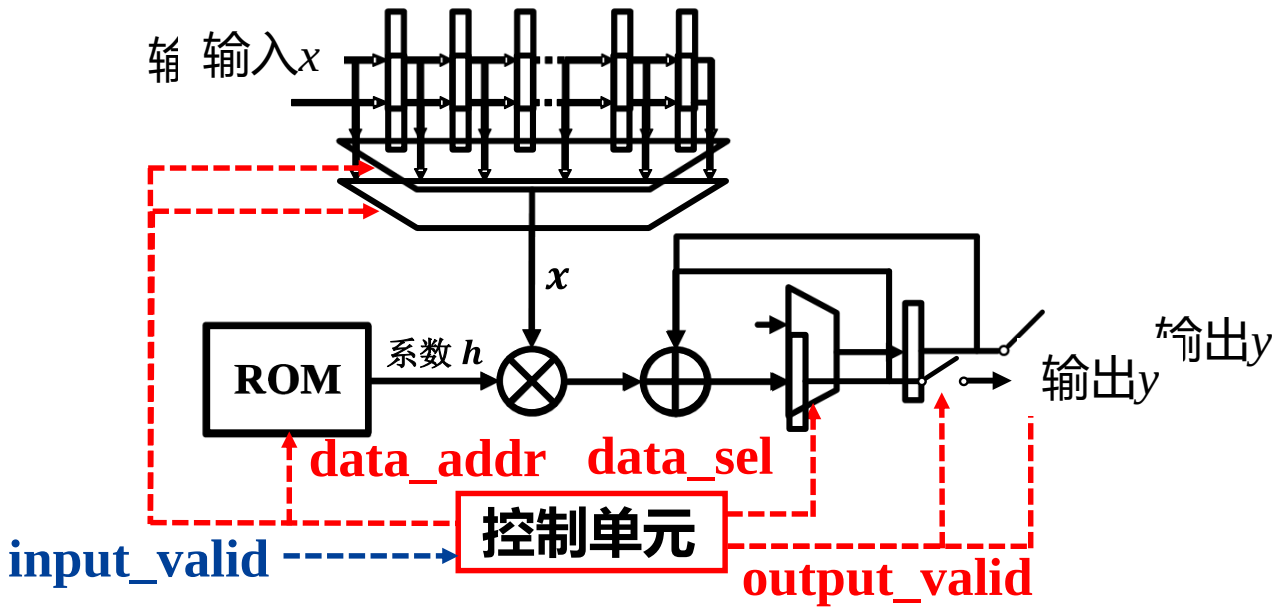
03.FIR滤波器设计——折叠的实现

控制相关信号

- 何时读入输入x
- 输入x和系数h的读地址
- 何时输出y
- 求“乘累加”时的临时值：

  第一次计算不需要累加

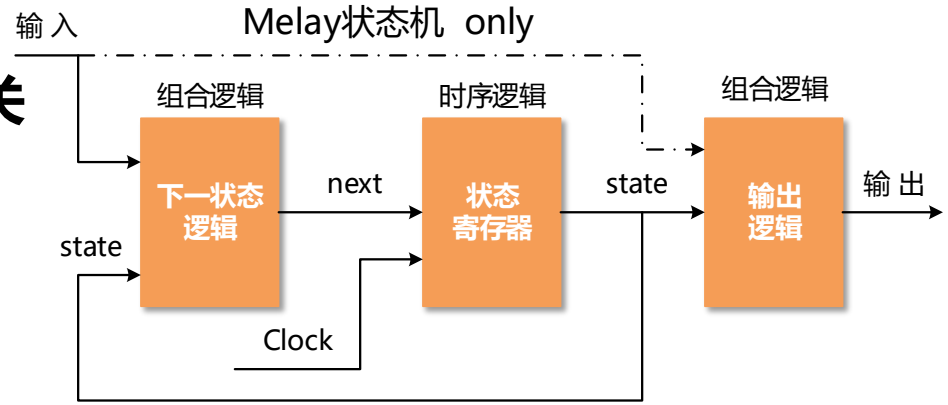
$$y(n)=\sum_{k=0}^{31}h(k)x(n-k)$$



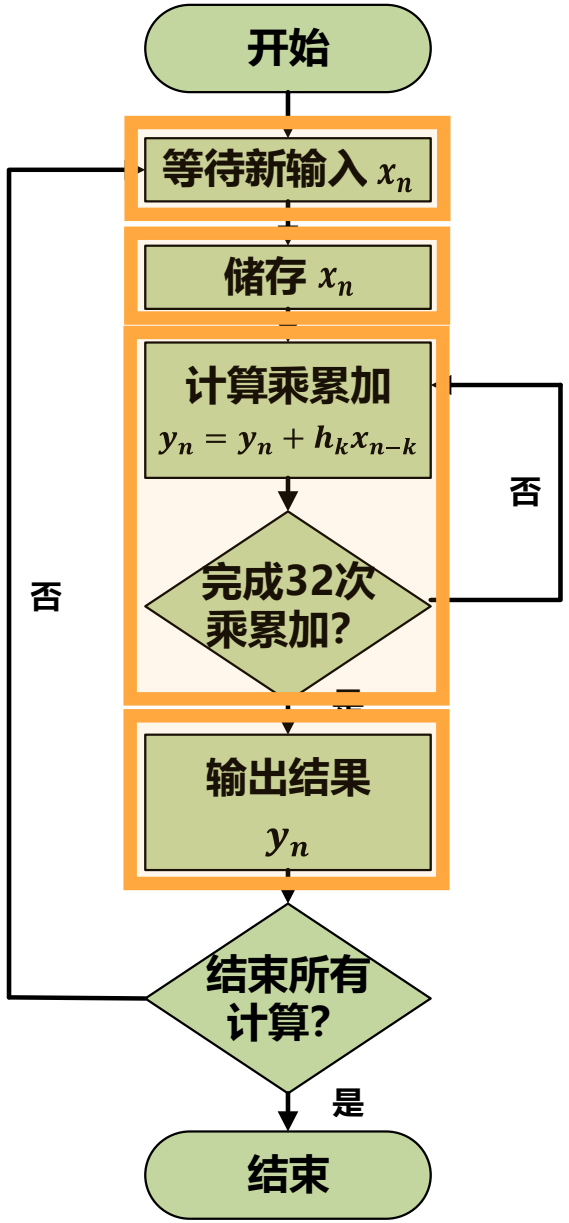
控制逻辑通过有限状态机实现

同步状态机概念

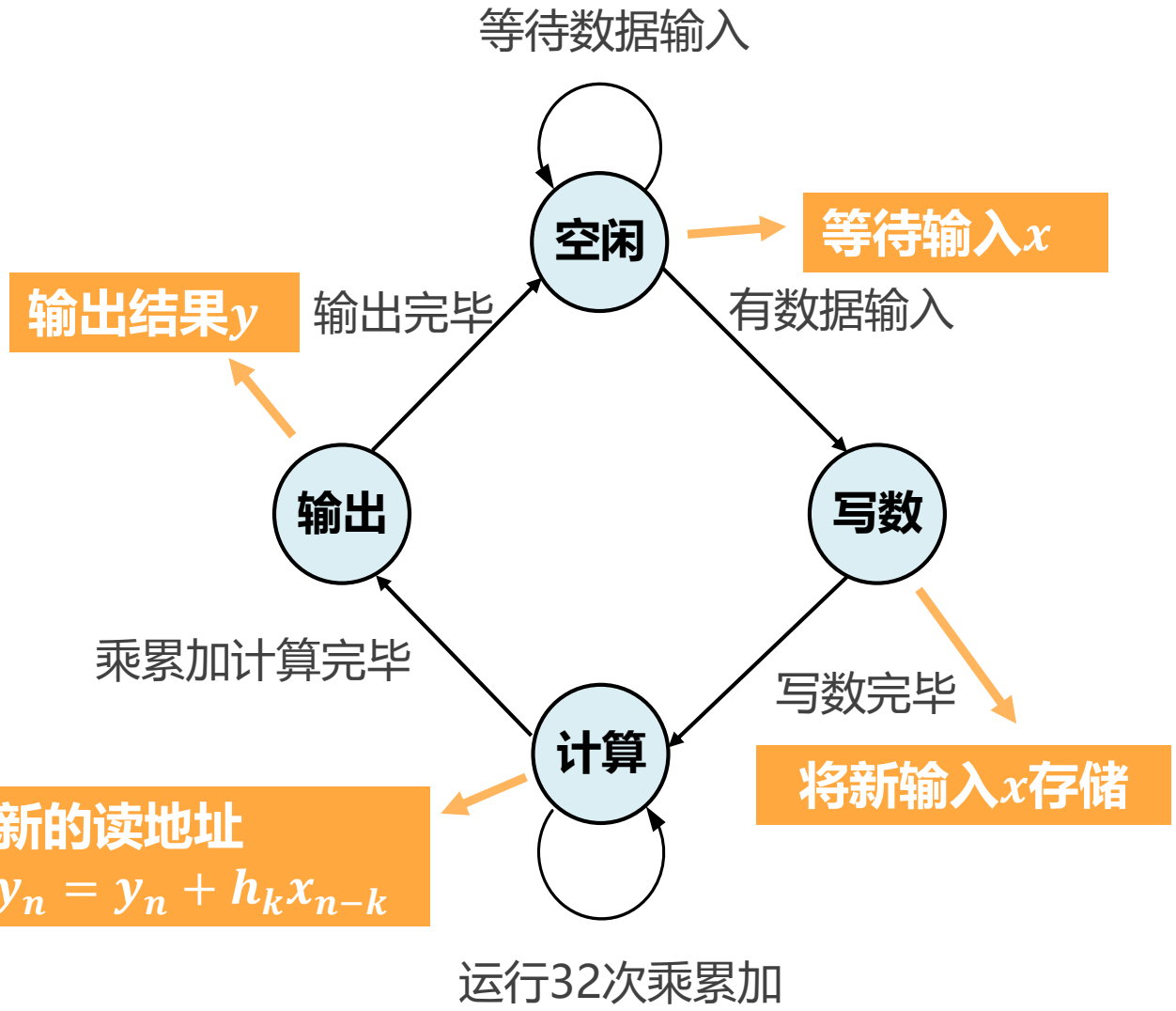
- 同步状态机由下一状态逻辑，状态寄存器和输出逻辑三个部分组成。
  - 驱动方程：决定了状态机的下一个状态，驱动方程是输入信号和当前状态的组合函数
  - 状态寄存器：由一组触发器组成，用来记忆当前状态
  - 状态机的输出：由输出函数得到，它也是当前状态和输入信号的函数
- 同步状态机分类：
  - Moore状态机：输出仅与当前状态有关
  - Mealy状态机：输出与当前状态和输入信号都有关



03.FIR滤波器设计——折叠|状态机设计



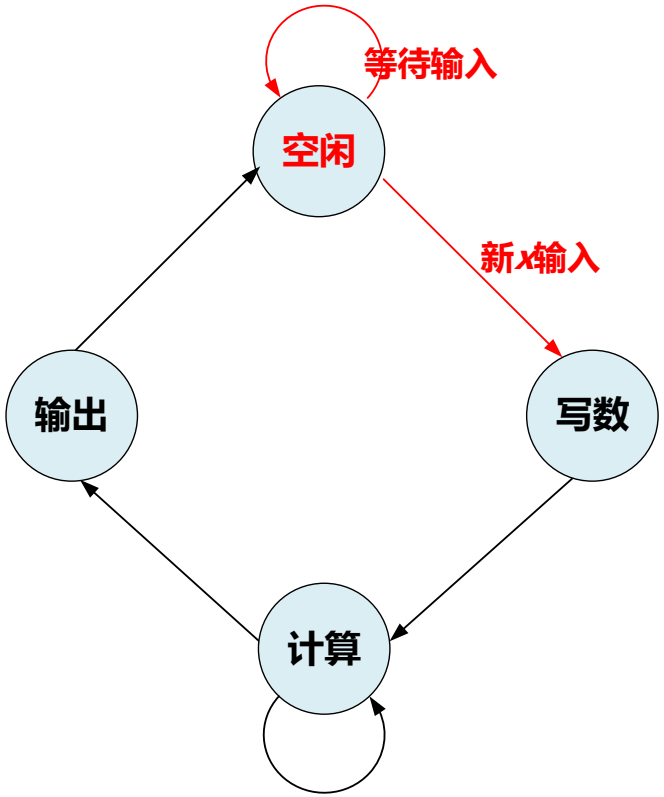
$$y_n = \sum_{k=0}^{31} h_k x_{n-k}$$

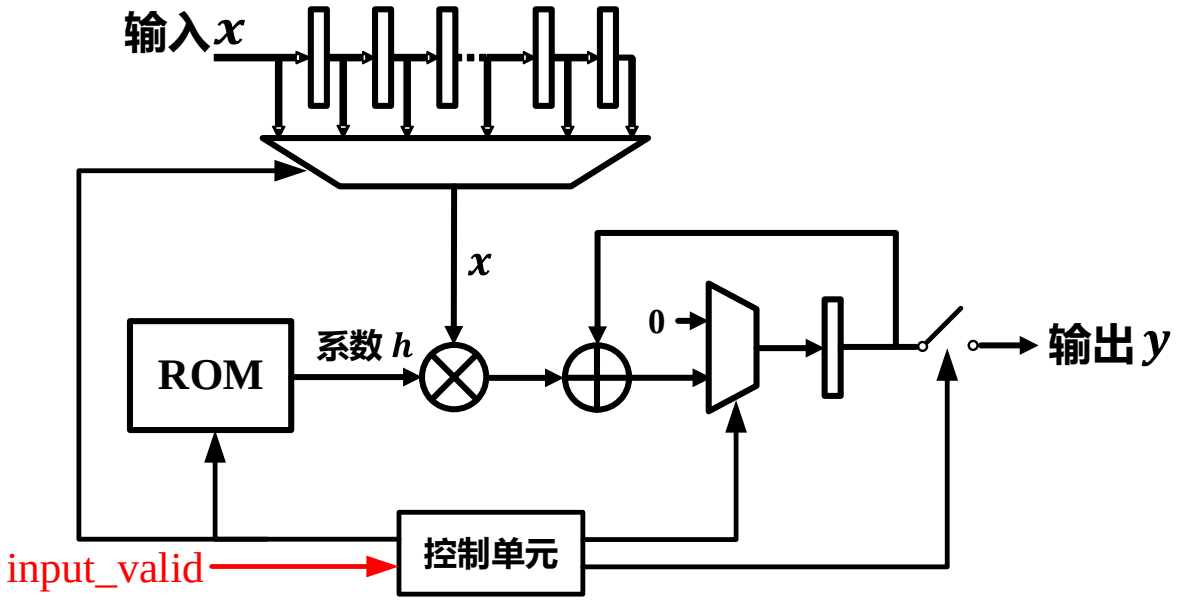


03.FIR滤波器设计——折叠|状态机工作流程

- 空闲
  - 等待新x输入时，状态保持不变
  - 有新x输入时，状态跳转到写数

$$y(n) = \sum_{k=0}^{31} h(k)x(n-k)$$

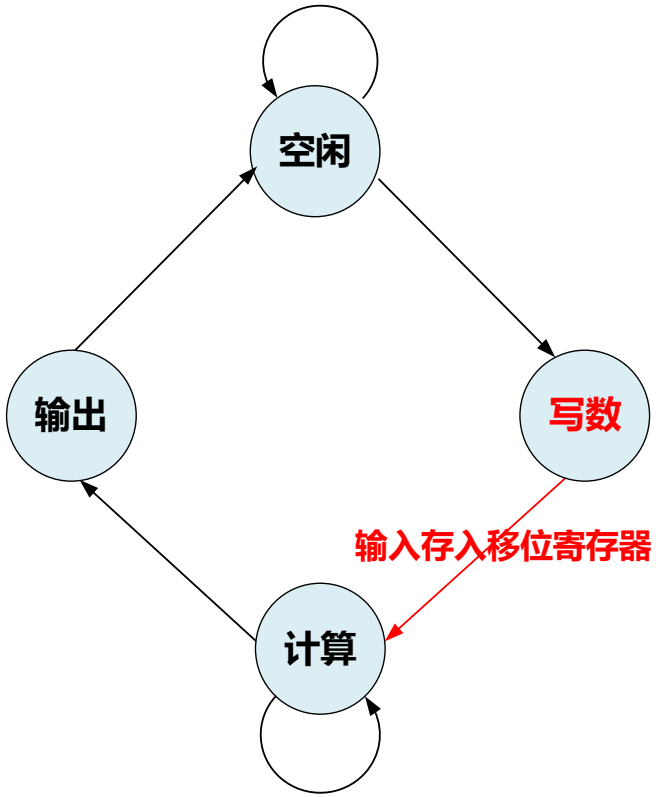


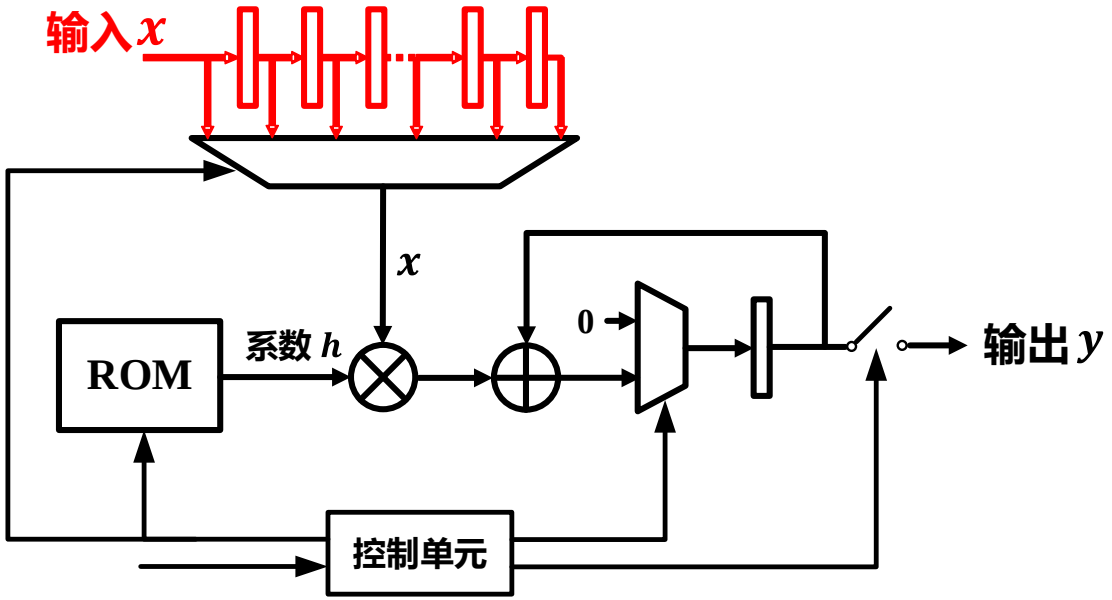


03.FIR滤波器设计——折叠|状态机工作流程

- 空闲 -> 写数
  - 将输入x存入移位寄存器，状态跳转到计算

$$y(n)=\sum_{k=0}^{31}h(k)x(n-k)$$

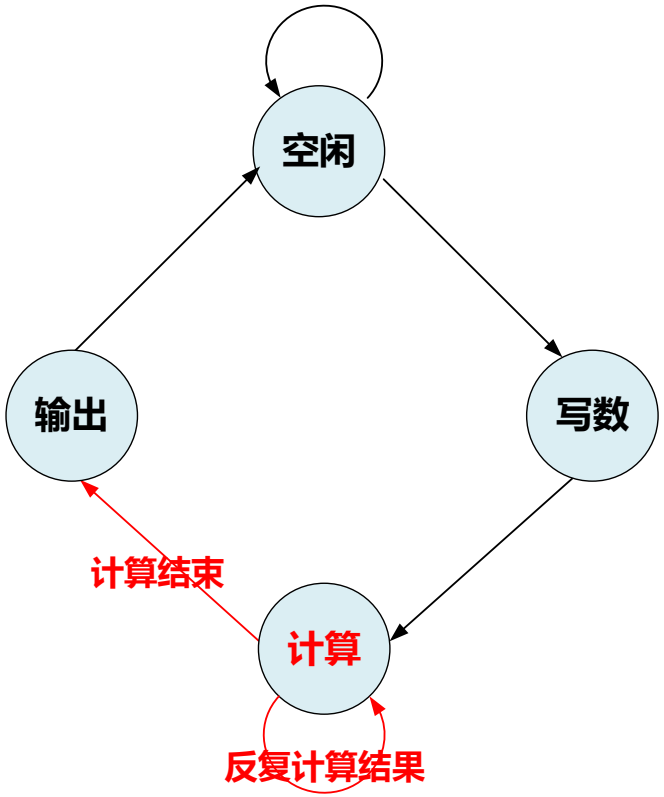


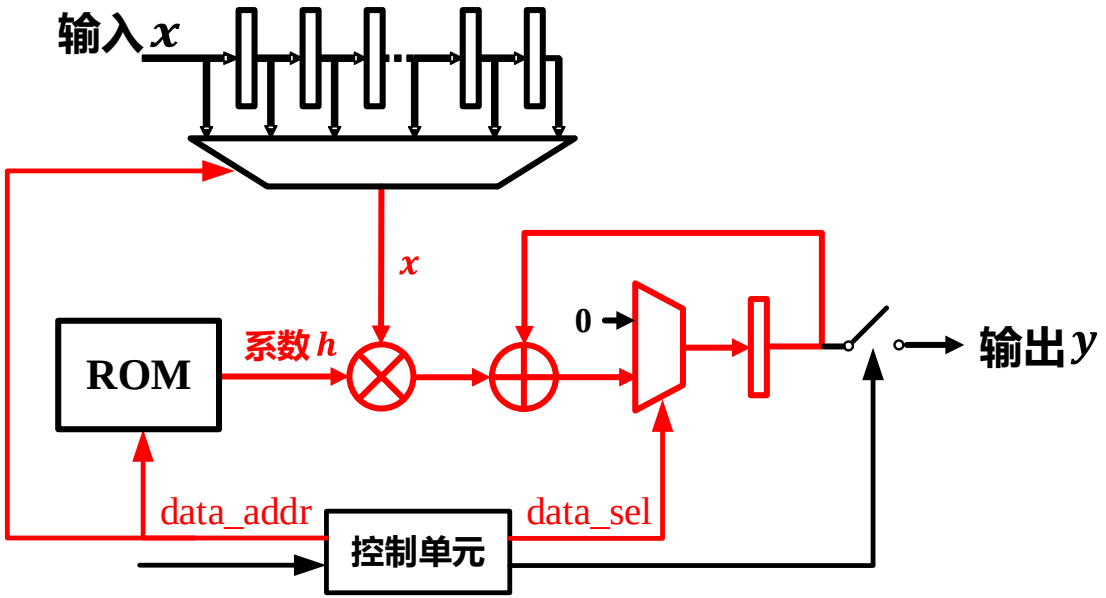


03.FIR滤波器设计——折叠|状态机工作流程

- 空闲 -> 写数 -> 计算
  - 计算未完成时，状态保持不变
  - 计算完成后，状态跳转到输出

$$y(n) = \sum_{k=0}^{31} h(k)x(n-k)$$

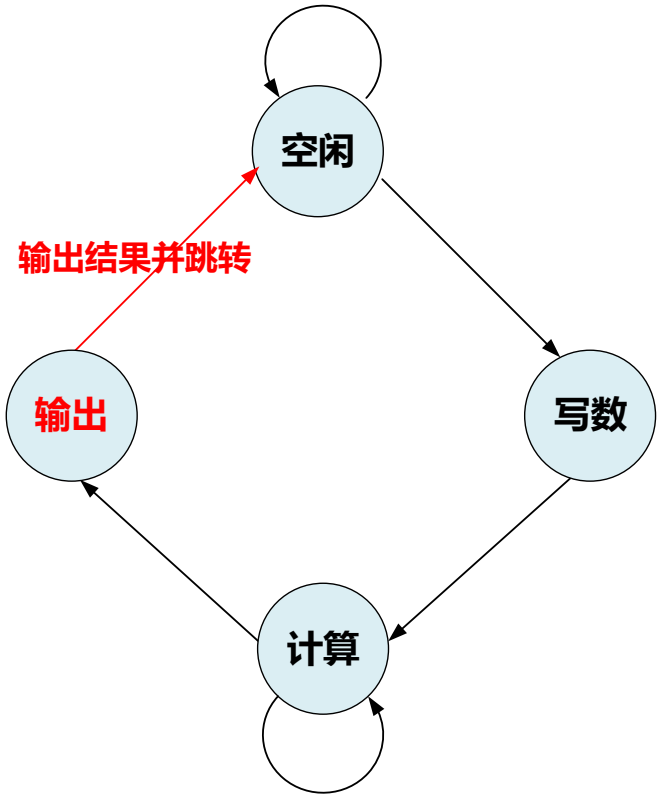


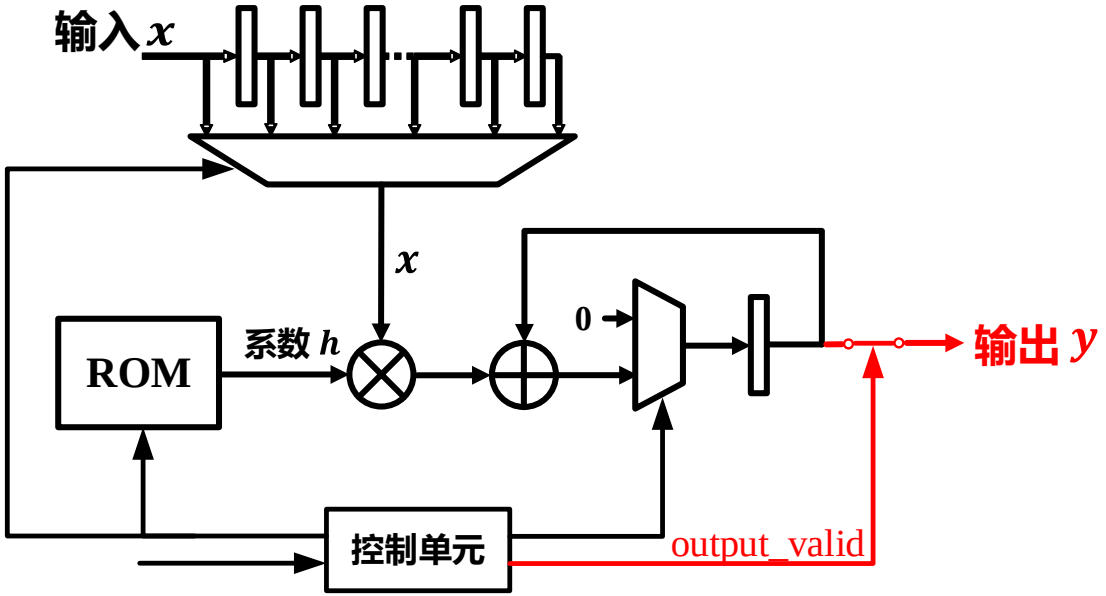


03.FIR滤波器设计——折叠|状态机工作流程

- 空闲 -> 写数 -> 计算 -> 输出
  - 输出结果，状态跳转到空闲

$$y(n) = \sum_{k=0}^{31} h(k)x(n-k)$$





03.FIR滤波器设计——折叠|状态机实现

RTL级状态机性能评判

- 稳定性好
- FSM速度快，满足设计频率要求
- 面积小
- FSM设计清晰易懂、易维护

状态机编码

- 为了使状态机能够在多个状态中切换，必须对每个状态进行有效的编码
- 常用的编码方式有
  - Binary code 二进制编码
  - One hot code 独热码
  - Gray code 格雷码

03.FIR滤波器设计——折叠|状态机实现

状态机设计通常有三段式的和两段式的方法，一段式由于可读性太差，不推荐使用

- **两段式**
  - **第一个always块用于状态的更迭（时序）**
  - **第二个always块用于产生新的状态以及当前状态对应的信号输出（组合）**
- **三段式**
  - **第一个always块用于状态的更迭（时序）**
  - **第二个always块用于产生新的状态（组合）**
  - **第三个always块用于同步产生相应的输出信号（时序）**

Moore状态机的Verilog实现

```
`timescale 1ns/100ps
module state4 (clock, reset, out);
  input reset, clock;
  output [1: 0] out;
  reg [1: 0] out;
  parameter //状态变量枚举
  stateA = 4‘b0000, stateB = 4’b0001…
  reg [3: 0] state, nextstate;

  //定义时序逻辑
  always @( posedge clock)
    if (reset) //同步复位
      state <= stateA;
    else
      state <= nextstate;
```

```
  always @( state) // 定义下一状态的组合逻辑
    case (state)
      stateA: begin
        nextstate = stateB;
        out = 2‘b00; // 输出决定于当前状态
      end
      ……
      stateD: begin
        nextstate = stateA;
        out = 2'b00;
      end
    endcase
endmodule
```

总结

2个并行模块

1) **always** block：下一状态的组合逻辑

2) **always** block：更新状态的时序逻辑

Mealy状态机的Verilog实现

```
module FSM_name (Clock, Resetn, input_signal, output_signal);
    input Clock, Resetn, input_signal;
    output output_signal;
    reg [3:0] state_present, STATE_NEXT;
    parameter [3:0] STATE1 = 4'b0000, STATE2 = 4'b0001…. ;

    // Define the next state combinational circuit and outputs
    always @(input_signal or state_present)
      case (state_present)
        STATE1: if (input_signal) define output and next state;
                else define output and next state;
        STATE2: if (input_signal) define output and next state;
                else define output and next state;
          ……
        default: define output and next state;
    endcase

    // Define the sequential block
    always @(posedge Clock)
      if (Resetn == 0) state_present <= STATE1;
      else state_present <= STATE_NEXT;
endmodule
```

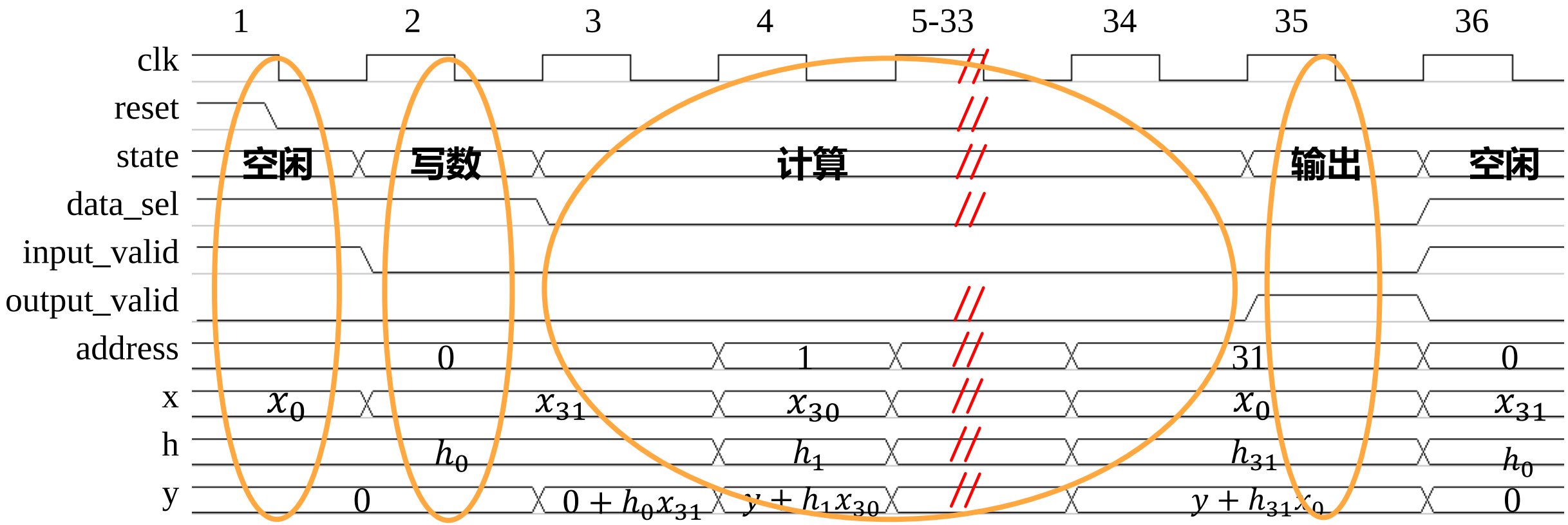
总结

2个并行模块

1) **always** block：下一状态的组合逻辑和输出

2) **always** block：更新状态的时序逻辑

03.FIR滤波器设计——折叠|时序分析



- **状态：输出**
  - **输出计算的y**
  - **耗时1个周期**

计算时间：每35个周期产生1个输出

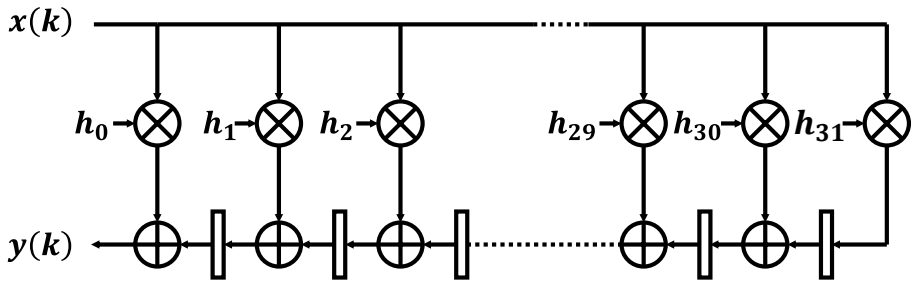
$$y_n = \sum_{k=0}^{31} h_k x_{n-k}$$

03.FIR滤波器设计——折叠与流水线比较

对31阶FIR滤波器，分别采用流水线方法和折叠方法优化，结果比较如下

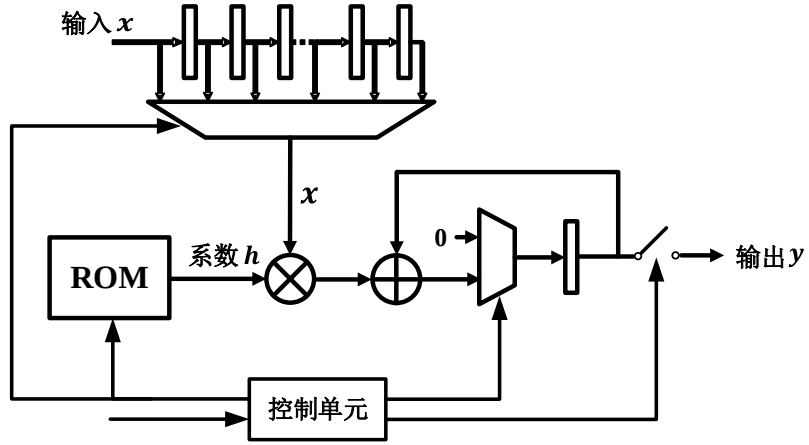
	流水线	折叠
时钟周期	1	N+4
加法器	N	1
乘法器	N+1	1
其他	-	控制和存储单元

推广到N阶



流水线结构

折叠以牺牲速度为代价，减小面积



折叠结构

FIR filter implementation

- Traditional Method
  - MAC (Multiply Accumulate) implementation
- Other Methods
  - DA (Distributed Arithmetic) implementation
  - Add and Shift method

目录

本章总结

有限字长量化

- 量化方法：截尾量化、舍入量化
- 降低硬件成本，但会带来量化误差和溢出

FIR滤波器架构设计

- 可采用流水线、并行处理、重定时等技术进行设计
- 折叠技术设计流程：

数据通路设计：运算和存储单元

↓

控制逻辑设计：算法流程图转化为有限状态机

谢谢！