Chapter 7

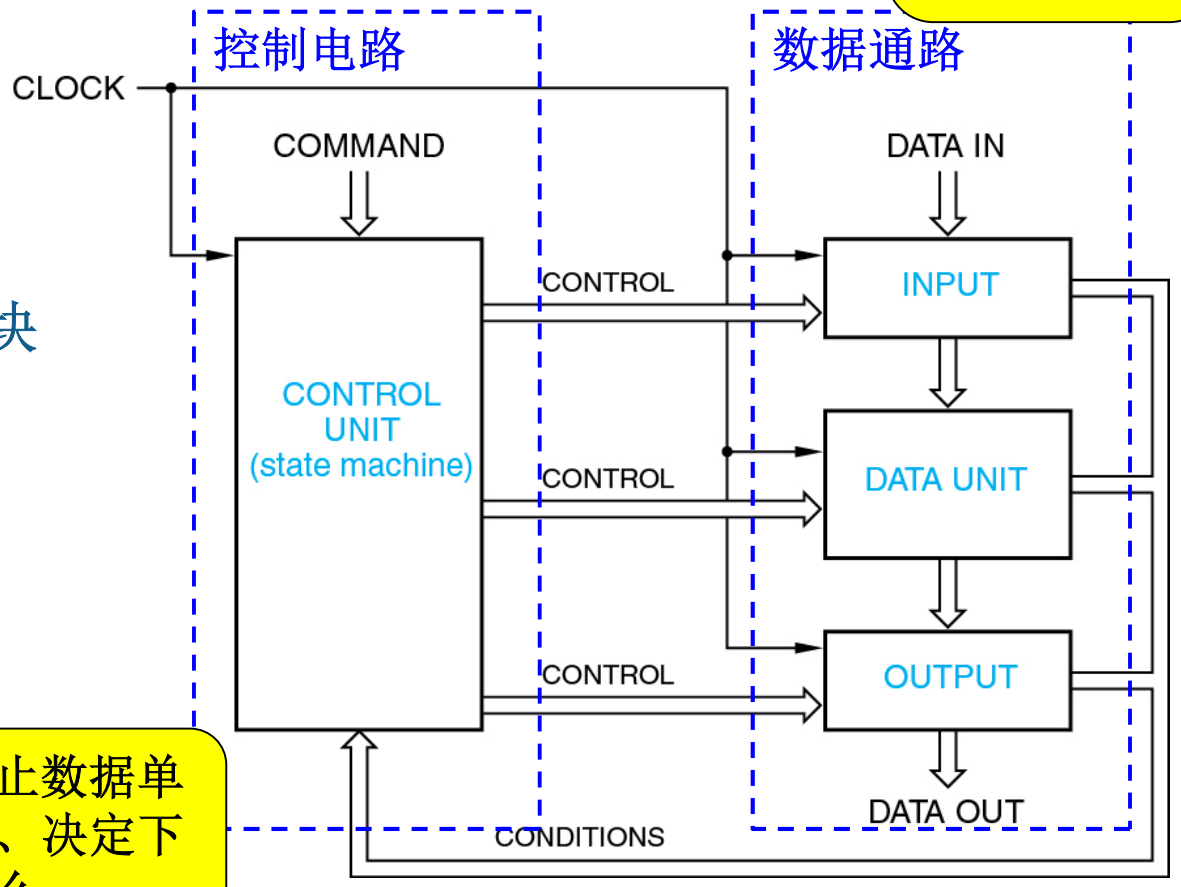
数字系统设计（同步）

目录

数字系统的概念

存储、传输、组合、处理“数据”

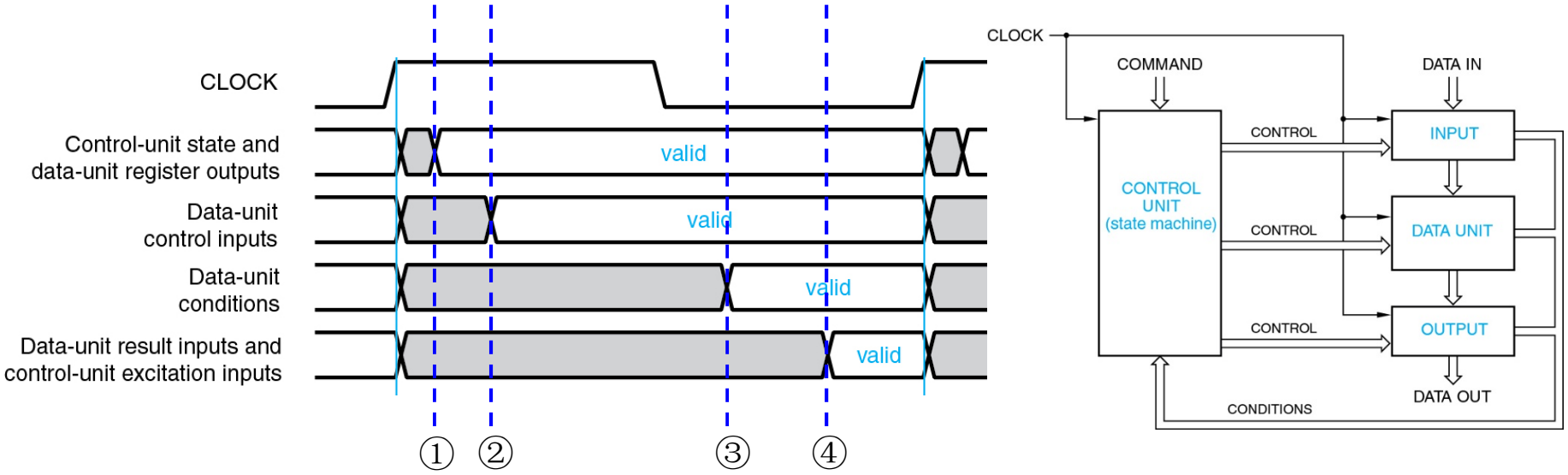
- 数字系统按照功能分为数据处理电路和控制电路。
- 数据通路
  - 组合型功能
  - 寄存器
  - 特定时序功能模块
  - 存储器操作
- 控制部件
  - 状态机



启动、停止数据单元的动作、决定下一步做什么

同步系统的时序

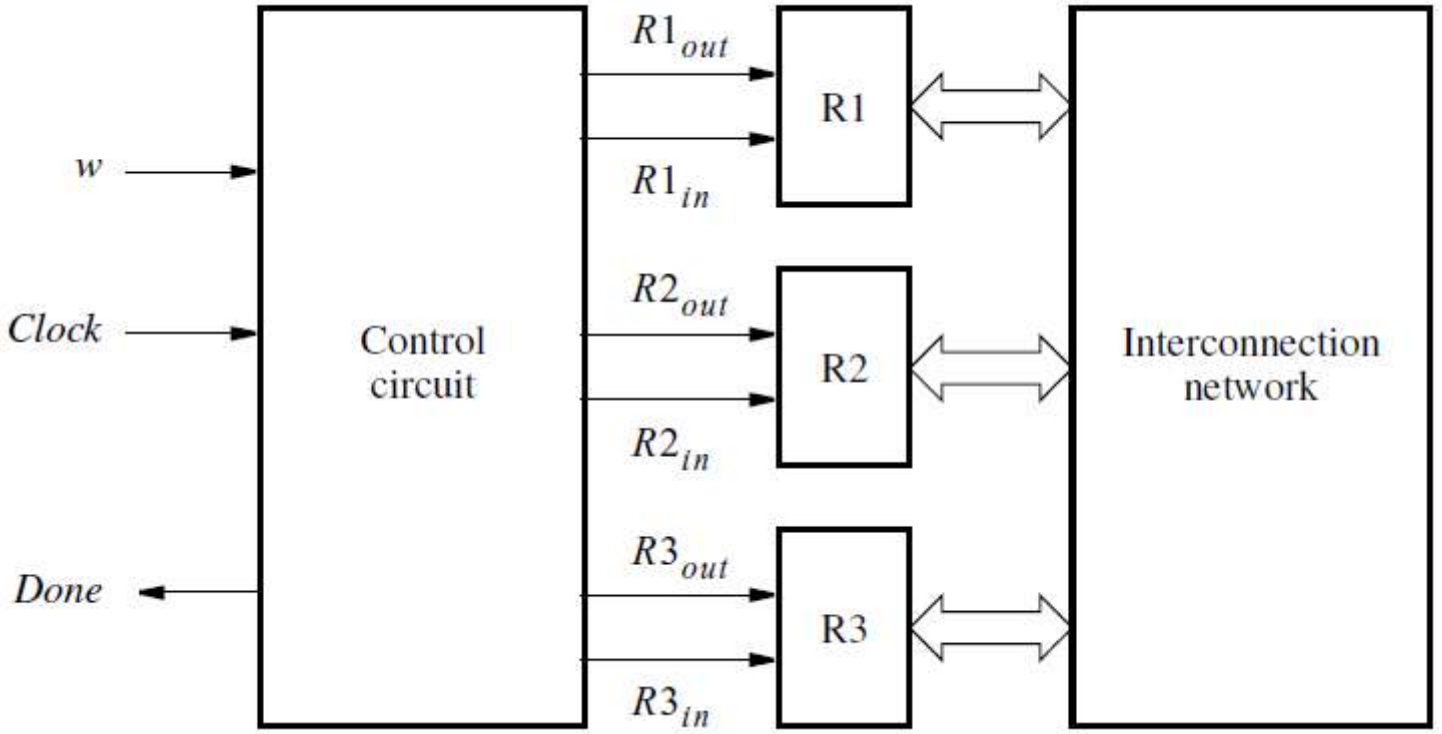
- 同步系统的关键特点：所有电路单元使用同一个时钟信号



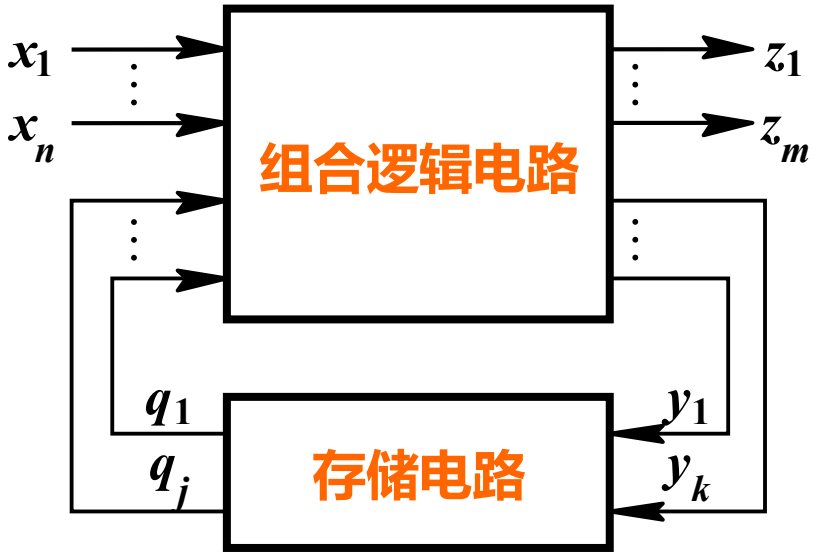
- 时序特点：
  - 时钟沿之后，控制单元的状态和数据单元的寄存器输出有效
  - 经过一个组合逻辑延迟，FSM的Moore输出有效，给数据单元用于控制
  - 时钟周期后期，数据单元输出有效，送给控制单元
  - 时钟周期末，状态机的次态逻辑结果有效；数据单元运算结果被载入到数据单元寄存器

数字系统例：数据交换网络控制器

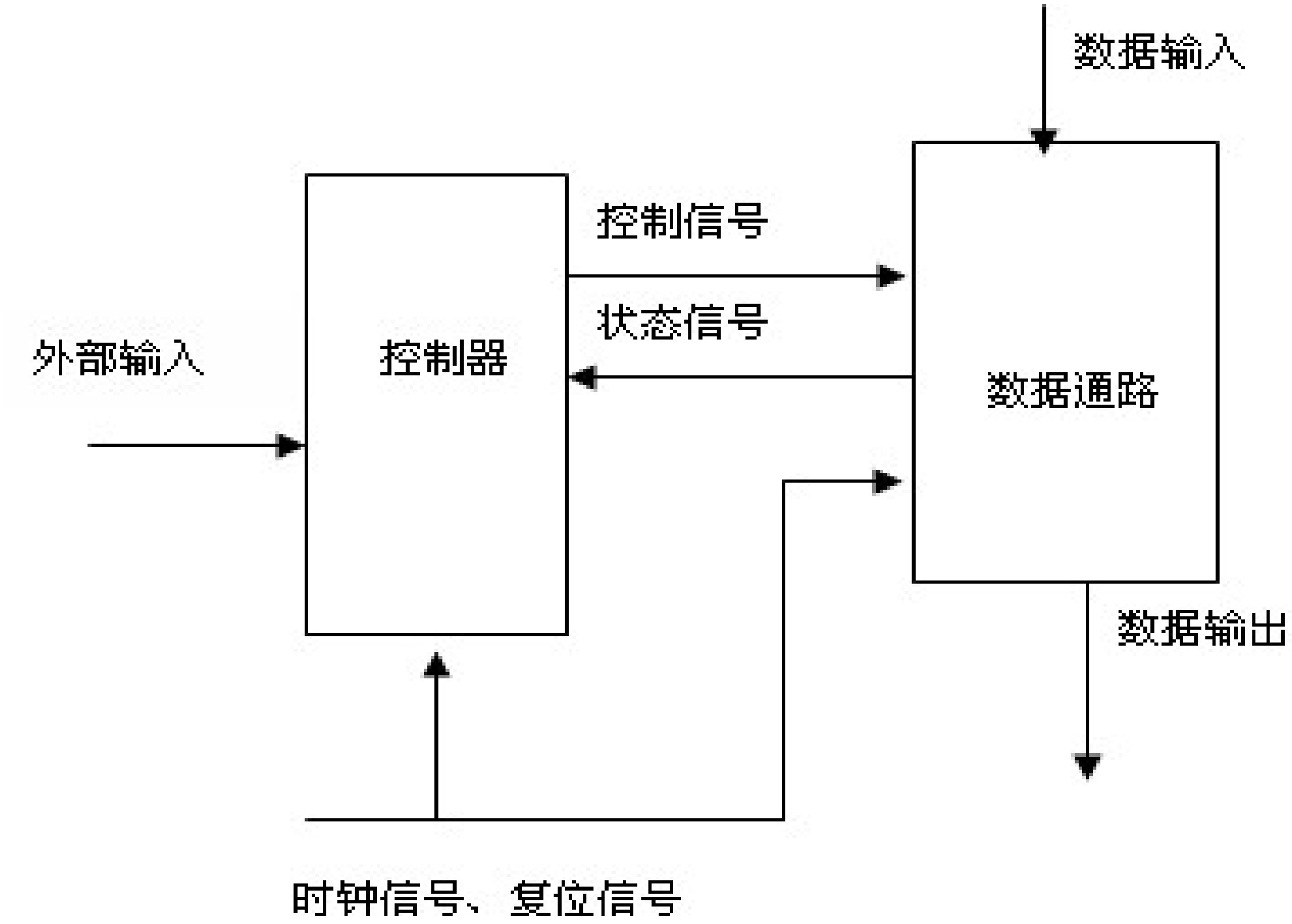
- 当*w=1*时, 交换R1和R2的内容；交换完成后，done=1。



模型一：组合逻辑与存储电路模型

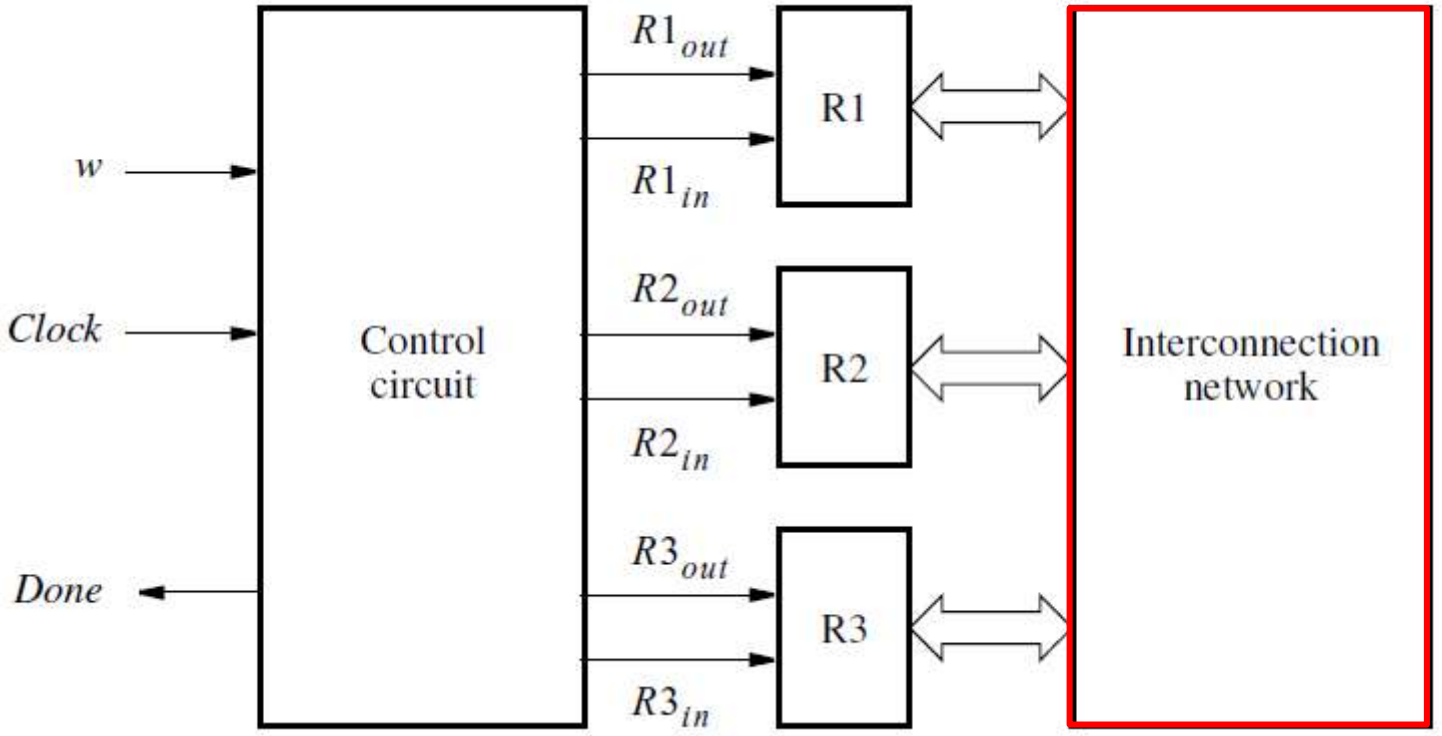


模型二：数据通路与控制电路

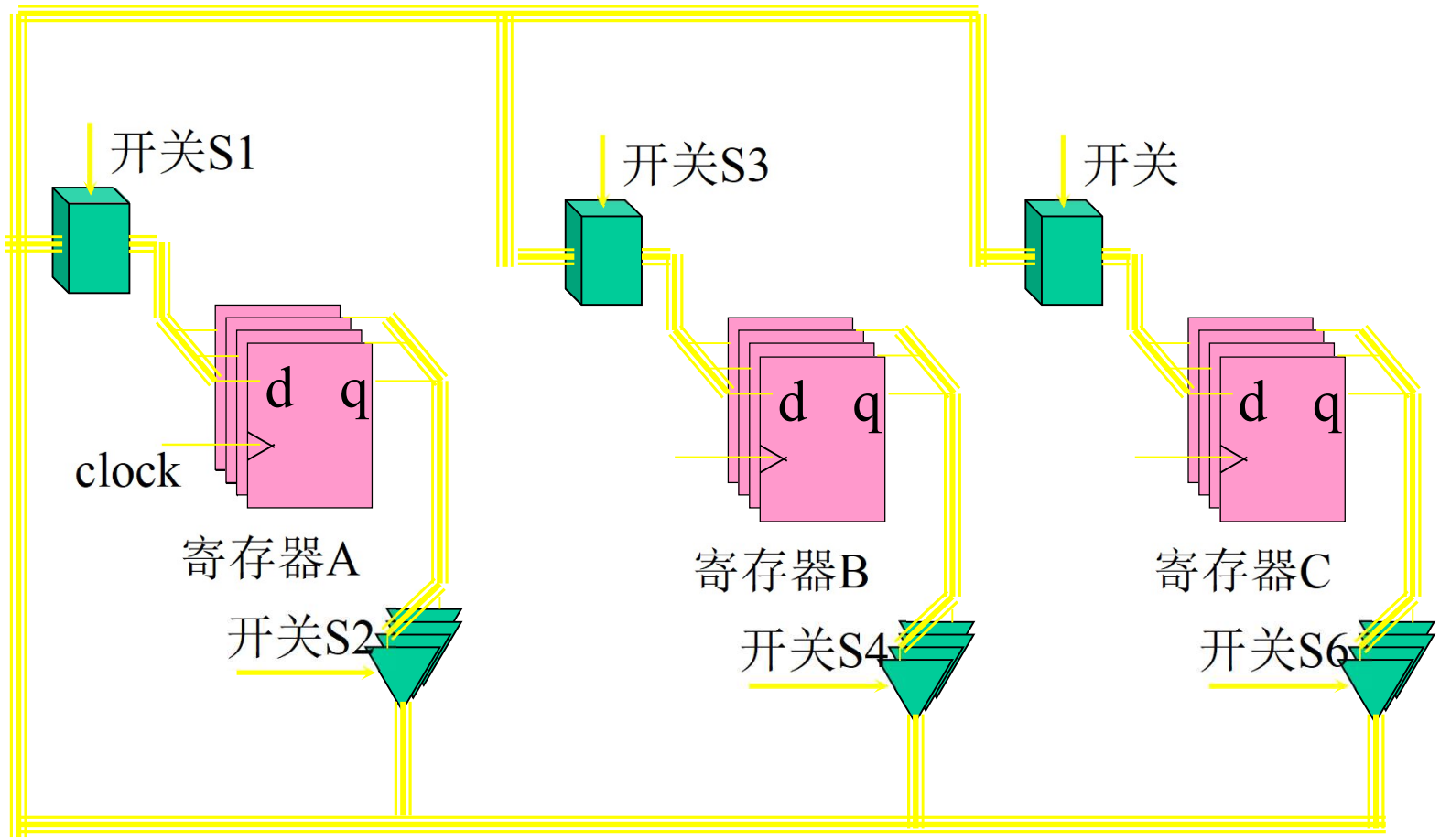


例2：数据交换网络控制器

- 当*w=1*时, 交换R1和R2的内容；交换完成后，done=1。



状态机应用：数据流动控制



问题：如何准确实现数据在各个寄存器、逻辑电路中的流动控制？比如：如何实现寄存器*C*的值准确存入到寄存器A中？

目录

总线

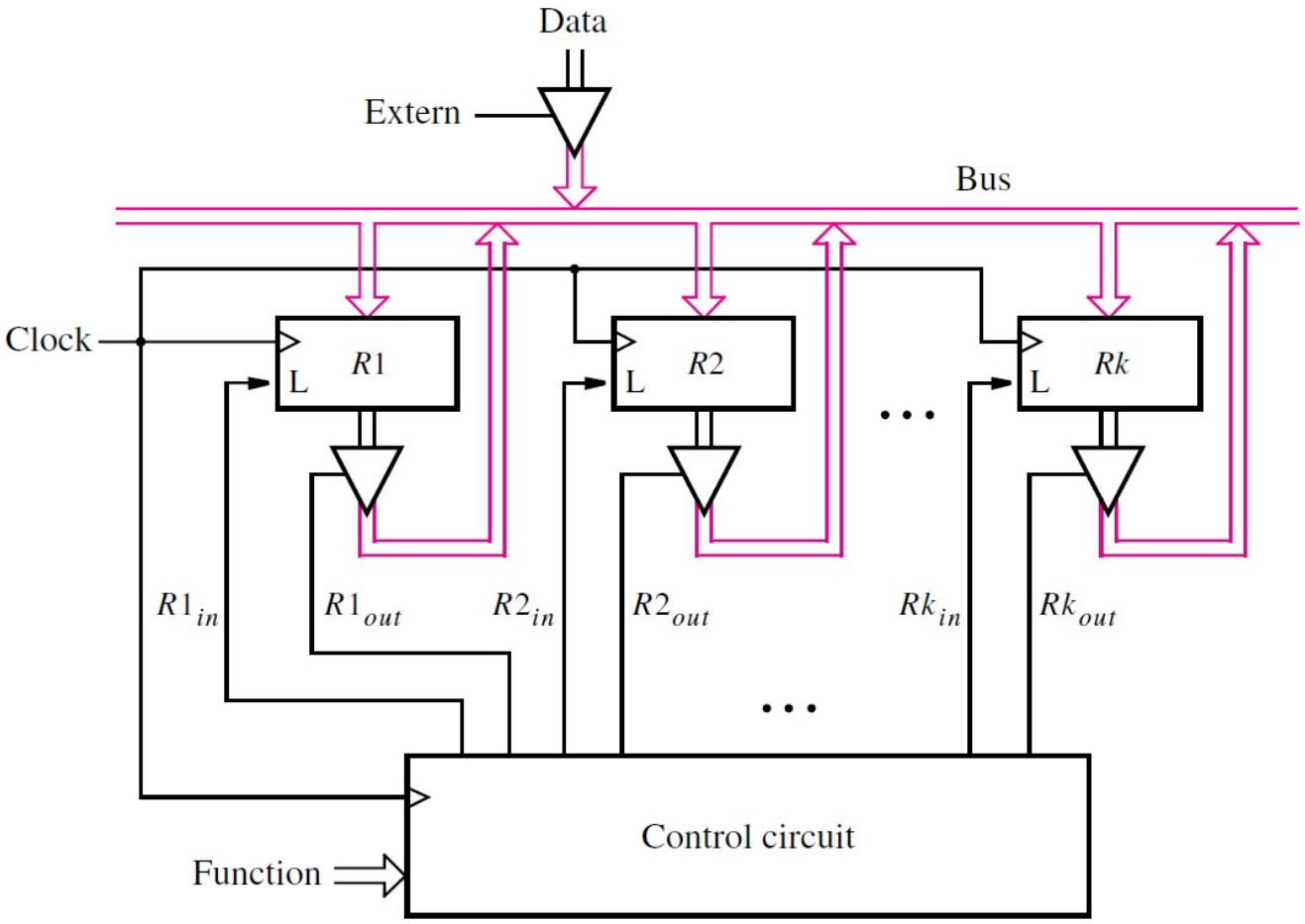


Figure 7.2. A digital system with *k* registers.

三态缓冲器

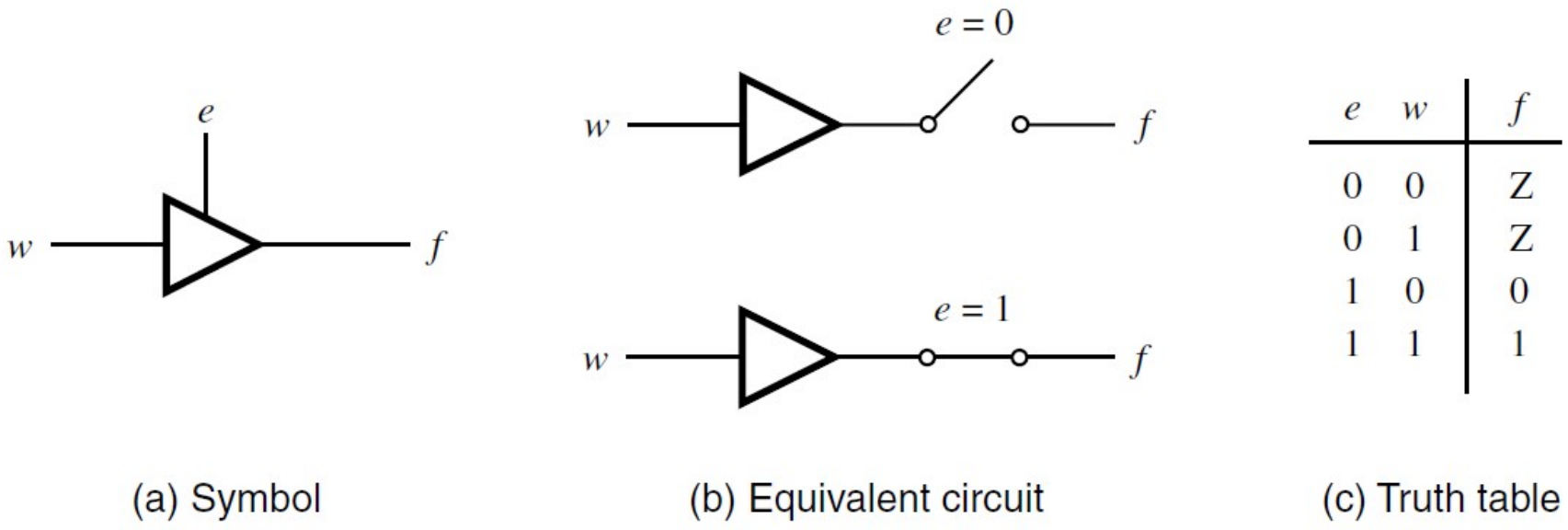


Figure 7.1. Tri-state driver.

基于三态缓冲器的总线

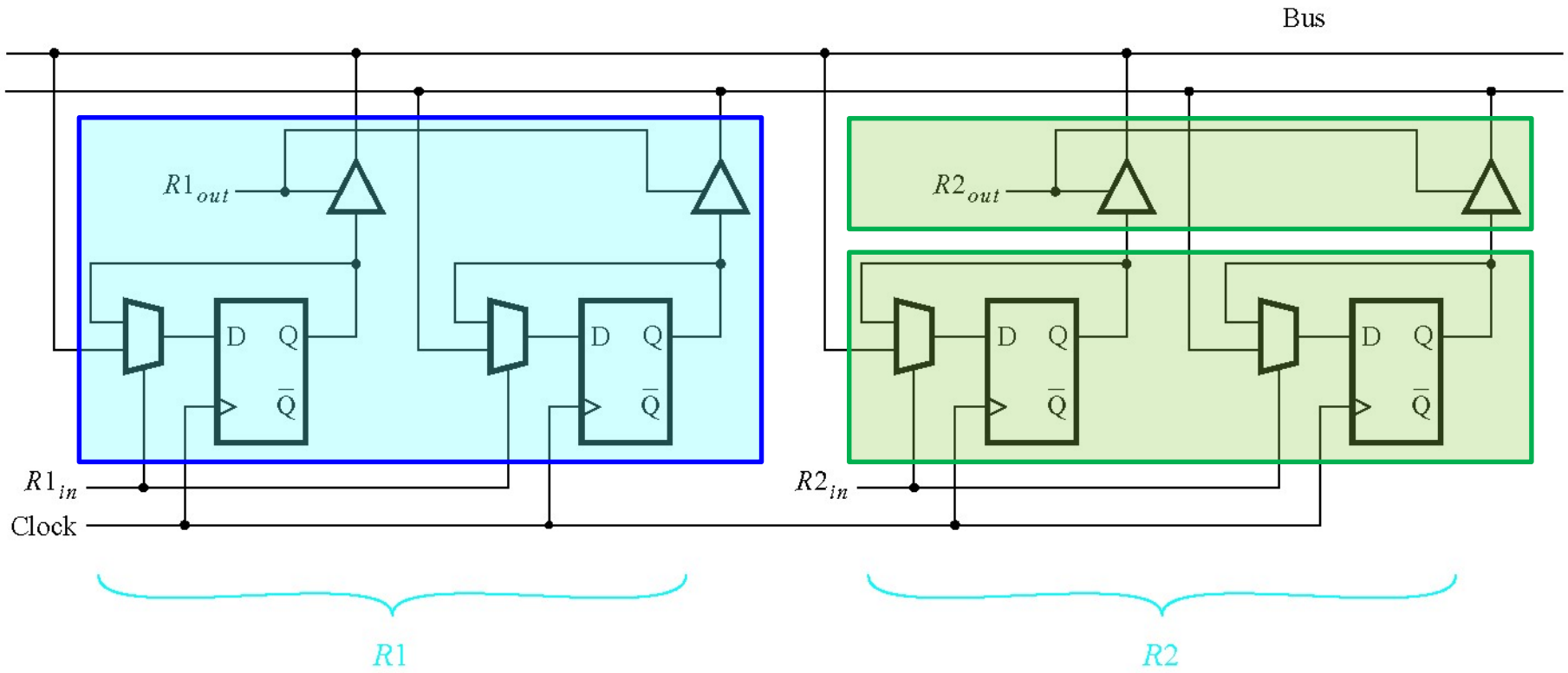


Figure 7.3. Details for connecting registers to a bus.

基于多路选择器的总线

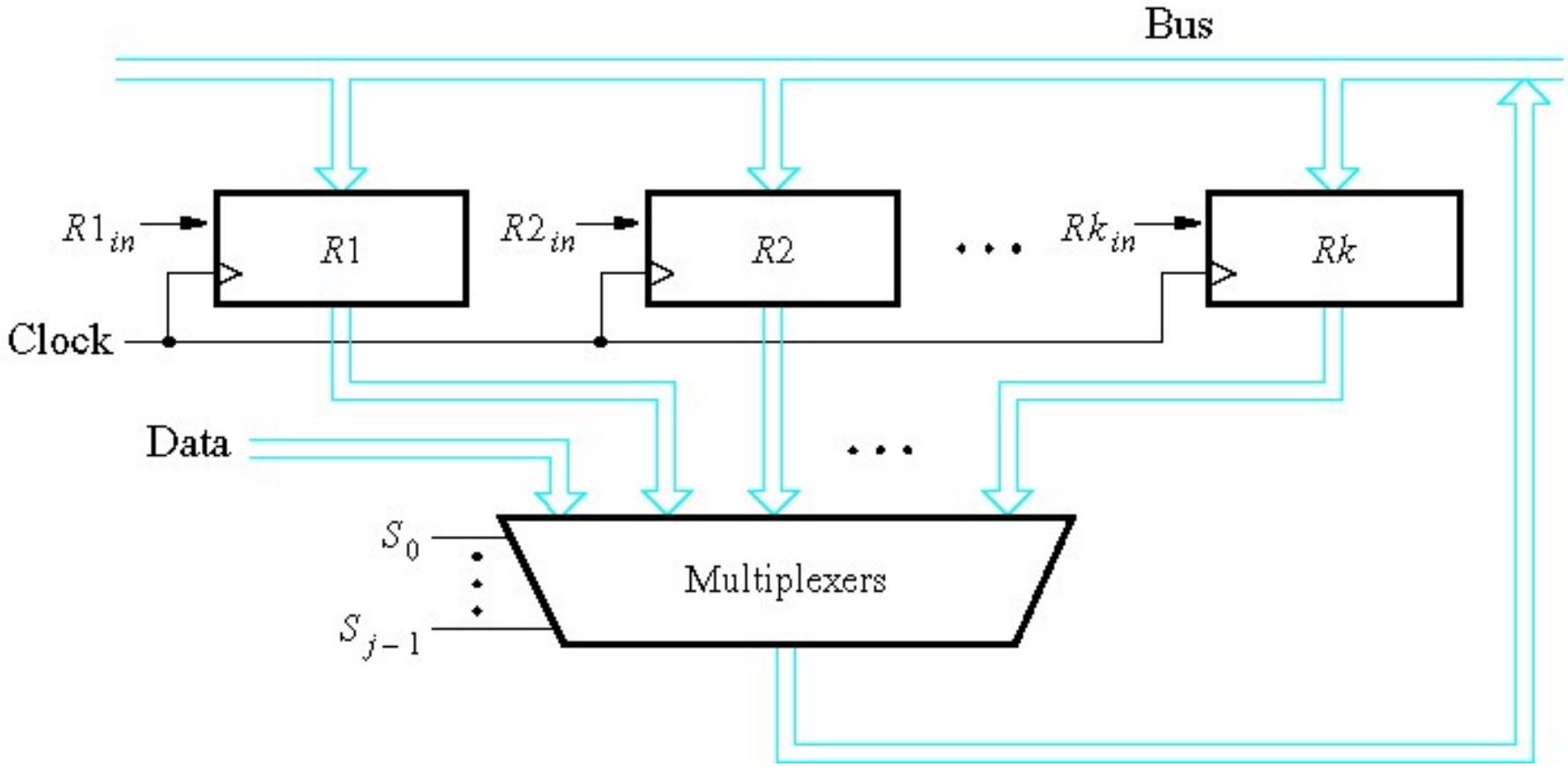
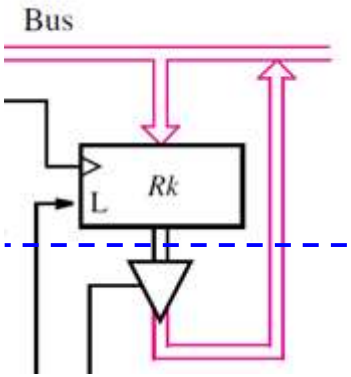


Figure 7.4. Using multiplexers to implement a bus.

三态总线的Verilog描述

```
module regn (R, L, Clock, Q);

    input [7:0] R;
    input L, Clock;
    output reg [7:0] Q;

    always @(posedge Clock)
      if(L)   Q <= R;

endmodule
```

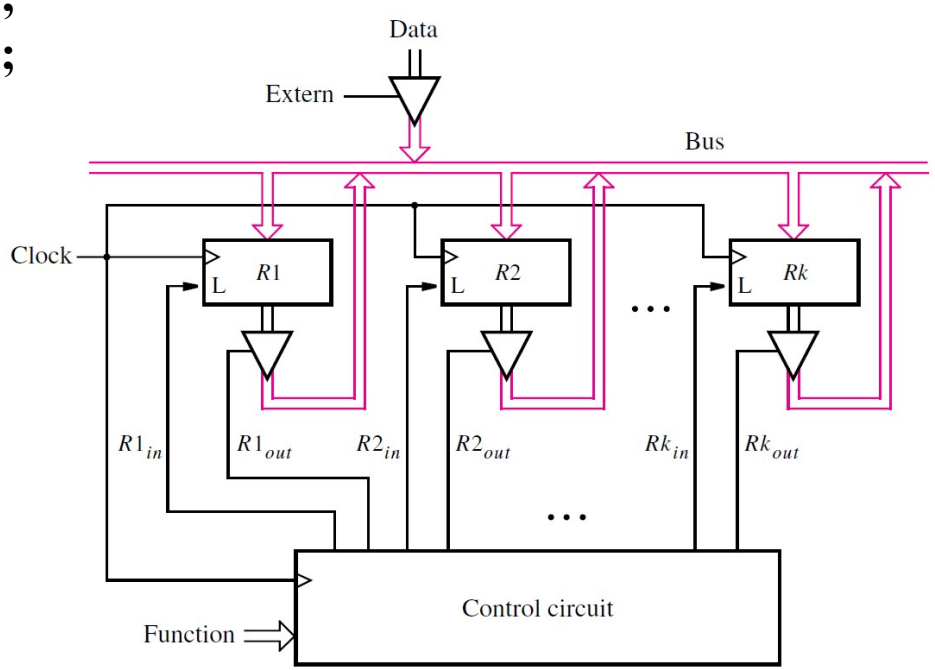
```
module trin (Y, E, F);

    input [7:0] Y;
    input E;
    output wire [7:0] F;

    assign F = E ? Y : 8'bz;

endmodule
```



数据通路

- 模块内部总线BusWires
- 基于三态总线描述

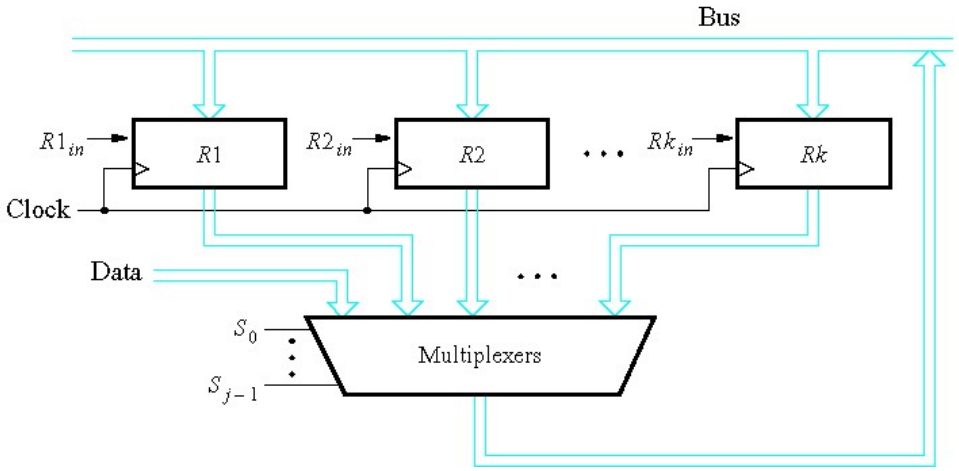
```
// Instantiate registers
regn reg 1 (BusWires, R1in, Clock, R1);
regn reg 2 (BusWires, R2in, Clock, R2);
regn reg 3 (BusWires, R3in, Clock, R3);

// Instantiate tri-state drivers
trin tri_ext (Data, Extern, BusWires);
trin tri_1 (R1, R1out, BusWires);
trin tri_2 (R2, R2out, BusWires);
trin tri_3 (R3, R3out, BusWires);
```



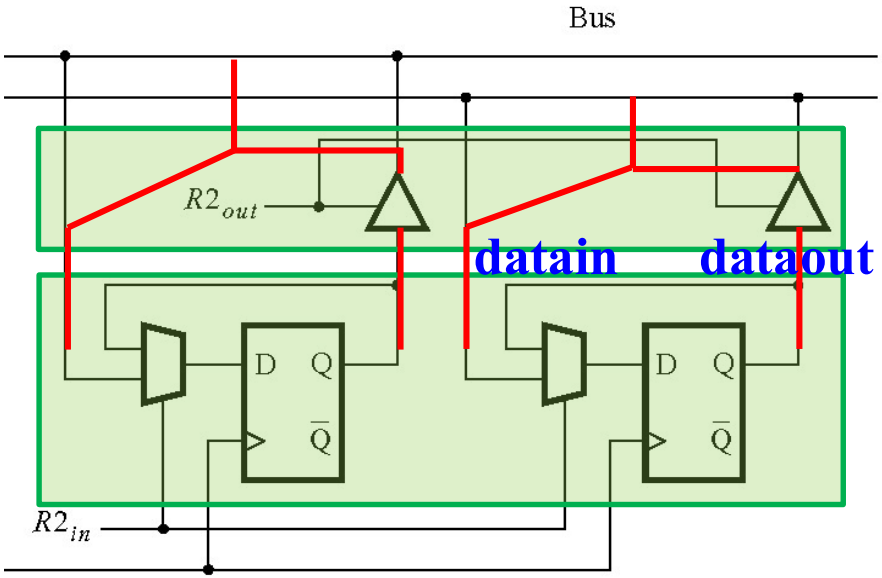
数据通路

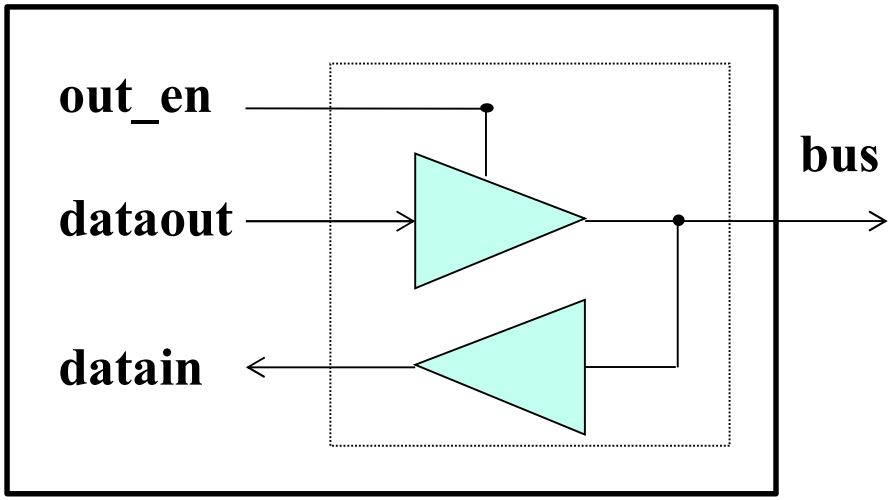
- 模块内部总线BusWires
- 基于多路选择器描述

```
// Instantiate registers
regn reg 1 (BusWires,  R1in, Clock, R1);
regn reg 2 (BusWires,  R2in, Clock, R2);
regn reg 3 (BusWires,  R3in, Clock, R3);

// Define the multiplexers
always @(y, Data, R1, R2, R3)
        if (y == A) BusWires = Data;
        else if (y == B) BusWires = R2;
        else if (y == C) BusWires = R1;
        else BusWires = R3;
```



三态总线的Verilog描述(2)

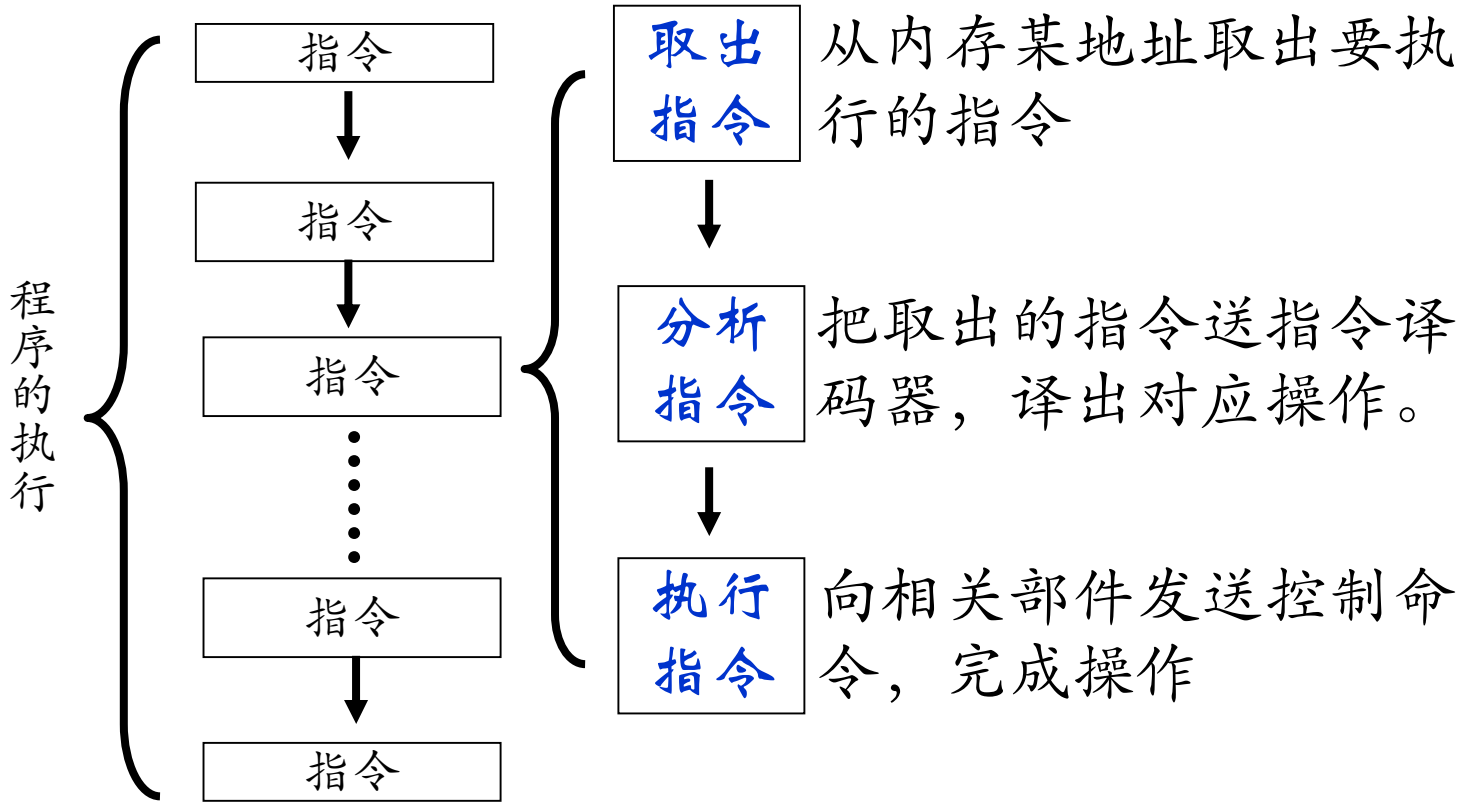





```
wire  out_en, in_en;
wire [7:0]  outbuf, datain;
inout [7:0] bus;
assign  bus =  (out_en) ? dataout : 8'hzz;
assign  datain = bus;
```

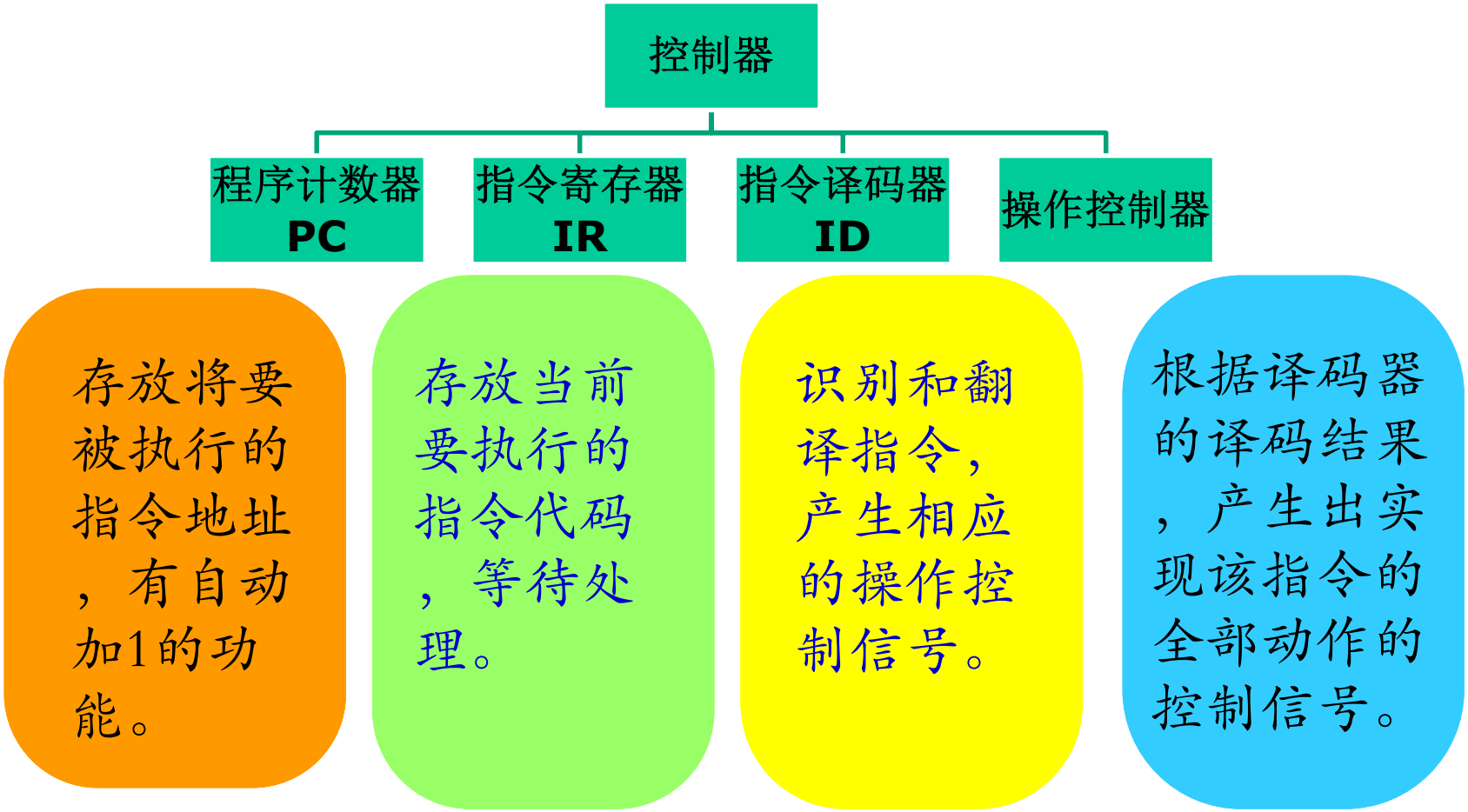
目录

指令的执行过程


程序的执行
指令
指令
指令
指令
指令
取出指令
从内存某地址取出要执行的指令
分析指令
把取出的指令送指令译码器，译出对应操作。
执行指令
向相关部件发送控制命令，完成操作

指令的执行过程

控制器：指挥中心，指挥各部件协调的工作。

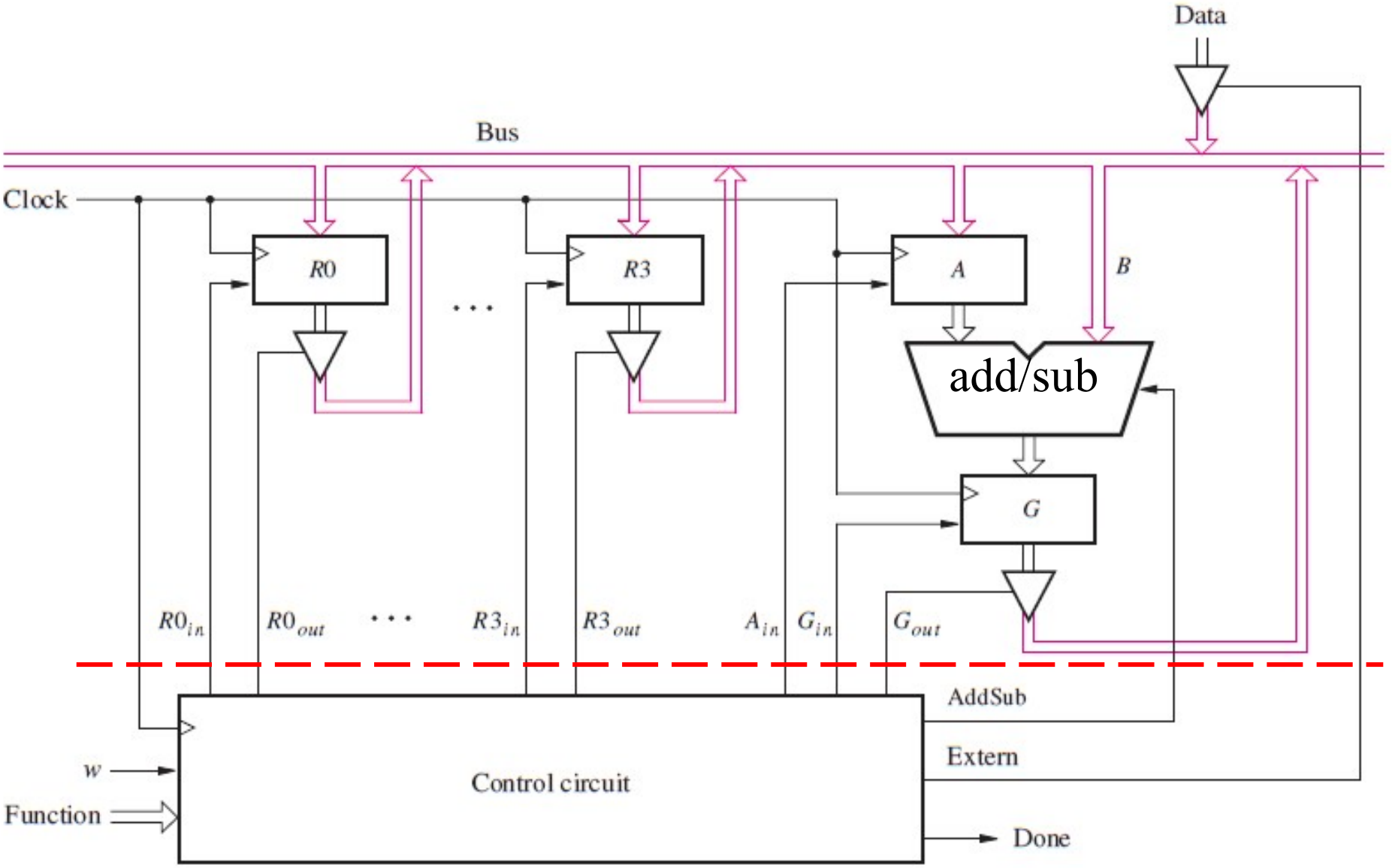


简单处理器指令集

	Operation	Function Performed
单周期	Load *Rx, Data*	*Rx* ← *Data*
单周期	Move *Rx, Ry*	*Rx* ← *[Ry]*
多周期	Add *Rx, Ry*	*Rx* ← *[Rx]*+*[Ry]*
多周期	Sub *Rx, Ry*	*Rx* ← *[Rx]*-*[Ry]*

Table 7.1. Operations performed in the processor.

处理器模块框图


Data
Bus
Clock
R0
R3
A
B
add/sub
G
$R0_{in}$
$R0_{out}$
$R3_{in}$
$R3_{out}$
A_{in}
G_{in}
G_{out}
AddSub
Extern
w
Function
Control circuit
Done

控制逻辑

生成与时钟精确配合的开关时序是计算逻辑的核心。

	T1	T2	
(Load):I_0	*Extern*, R_{in}=*X*, *Done*		
(Move):I_1	R_{in}=*X*, R_{out}=*Y*, *Done*		
(Add): I_2	R_{out}=*X*, A_{in}	R_{out}=*Y*, G_{in}, *AddSub=0*	G_{out}, R_{in}=*X*, *Done*
(Sub): I_3	R_{out}=*X*, A_{in}	R_{out}=*Y*, G_{in}, *AddSub=1*	G_{out}, R_{in}=*X*, *Done*

$$Extern = I_0 T_1$$

$$A_{in} = (I_2 + I_3)\ T_1$$

$$G_{in} = (I_2 + I_3)\ T_2$$

$$G_{out} = (I_2 + I_3)\ T_3$$

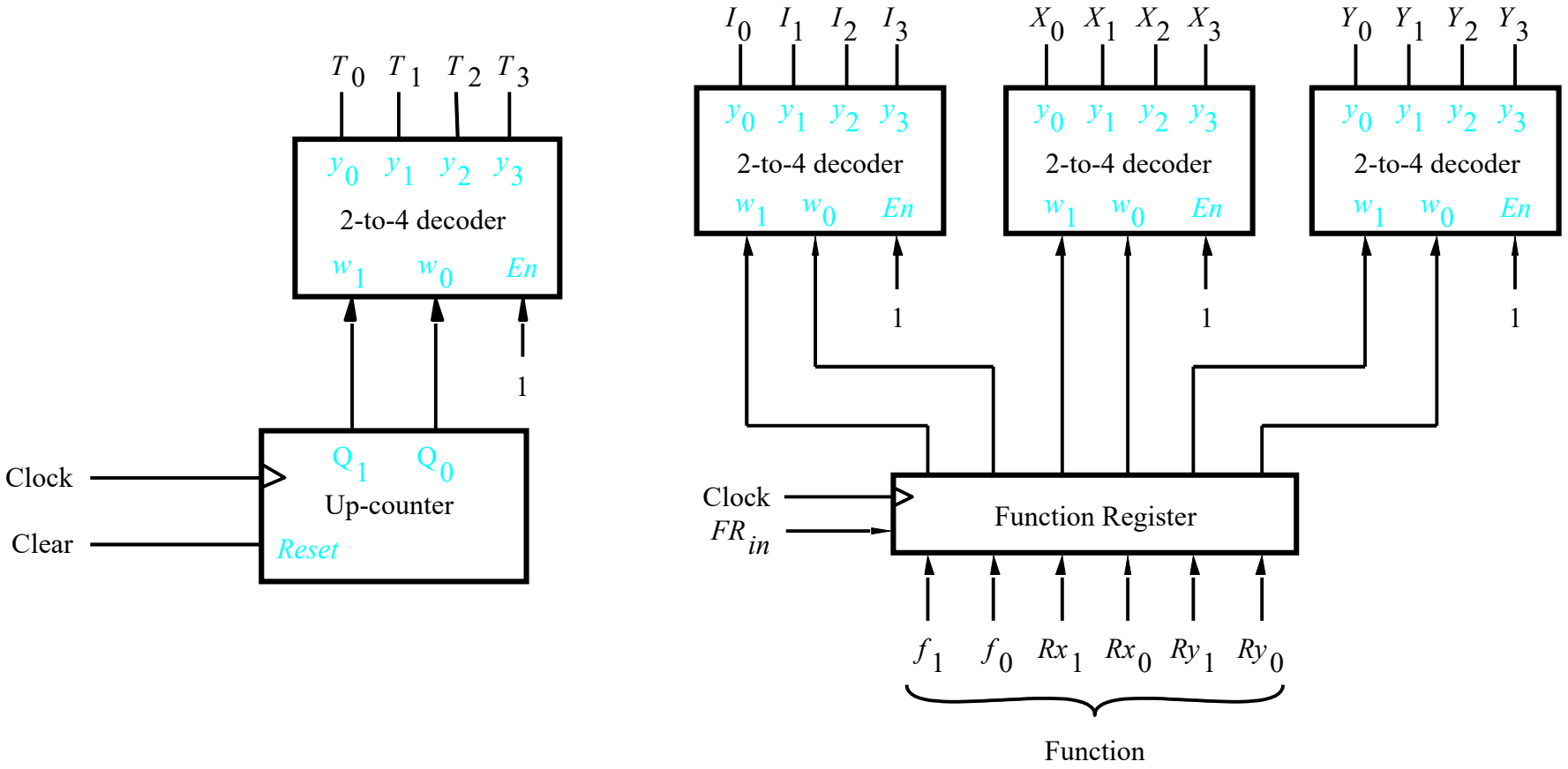
$$AddSub = I_3$$

$$Done = (I_0 + I_1)\ T_1 + (I_2 + I_3)\ T_3$$

$$R0_{in} = (I_0 + I_1)T_1 X_0 + (I_2 + I_3)T_3 X_0$$

$$R0_{out} = I_1 T_1 Y_0 + (I_2 + I_3)(T_1 X_0 + T_2 Y_0)$$

操作控制（周期控制）



$$Clear = \overline{w}T_0 + Done$$

$$FR_{in} = wT_0$$

操作控制（计数器）

```
module upcount (Clear, Clock, Q);
    input Clear, Clock;
    output reg [1:0] Q;

    always @(posedge Clock)
        if (Clear)
            Q <= 0;
        else
            Q <= Q + 1;

endmodule
```

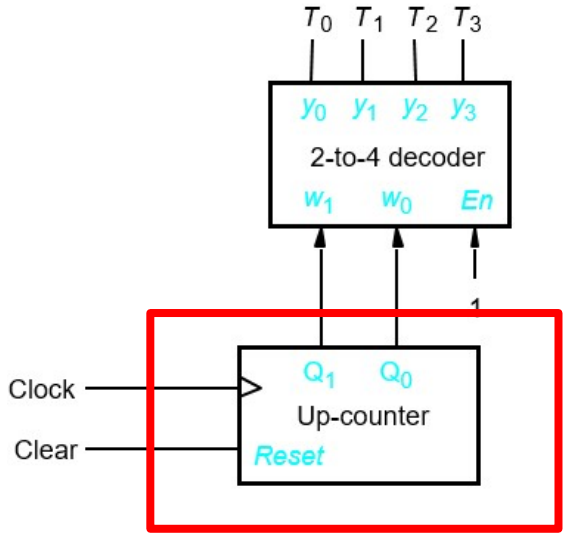
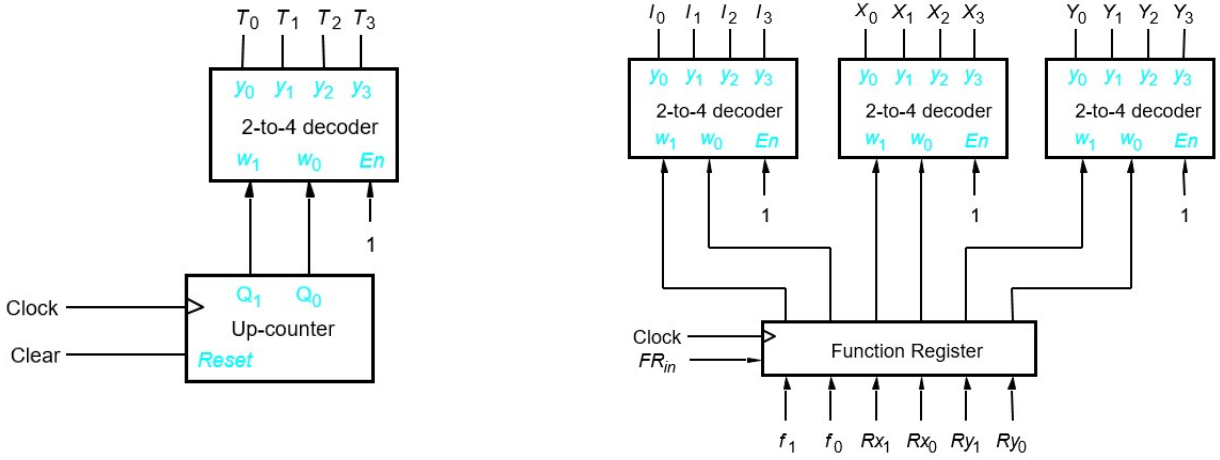


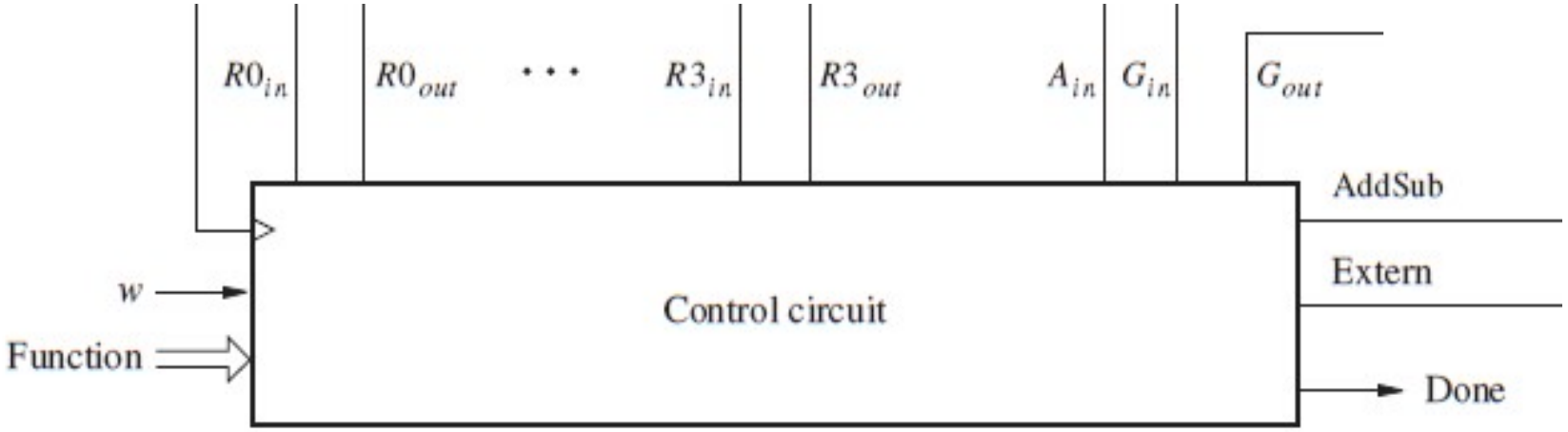
Figure 7.12. A two-bit up-counter with synchronous reset.

操作控制（译码器）

```
wire [0:3] T, I, Xreg, Y;
upcount counter (Clear, Clock, Count);
regn functionreg (Func, FRin, Clock, FuncReg);
dec2to4 decT (Count, 1'b1, T);
dec2to4 decI (FuncReg[1:2], 1'b1, I);
dec2to4 decX (FuncReg[3:4], 1'b1, Xreg);
dec2to4 decY (FuncReg[5:6], 1'b1, Y);
```



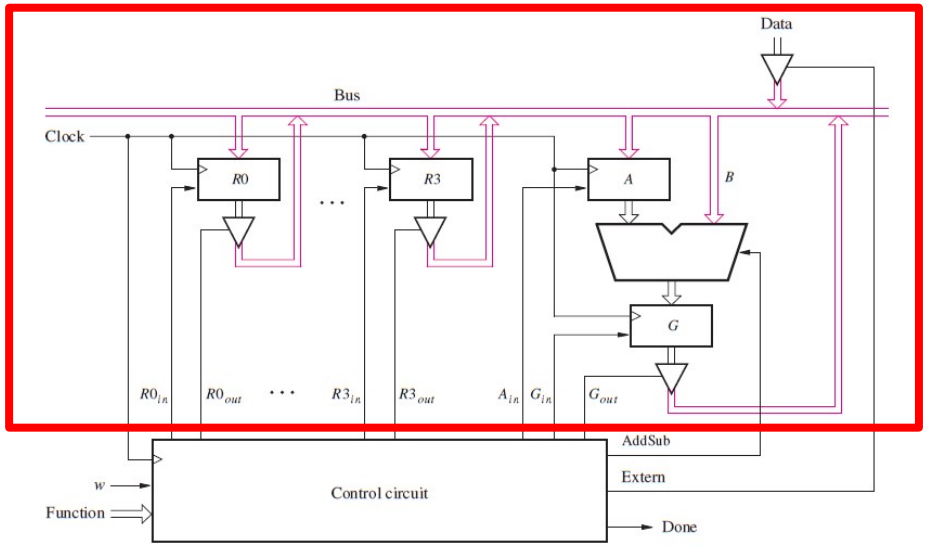
操作控制器




```
assign Clear = Reset | Done | (~w & T[0]);
assign Func = {F, Rx, Ry};
assign FRin = w & T[0];
assign Extern = I[0] & T[1];
assign Done = ((I[0] | I[1]) & T[1]) | ((I[2] | I[3]) & T[3]);
assign Ain = (I[2] | I[3]) & T[1];
assign Gin = (I[2] | I[3]) & T[2];
assign Gout = (I[2] | I[3]) & T[3];
assign AddSub = I[3];
assign R0in = .....
.....
```

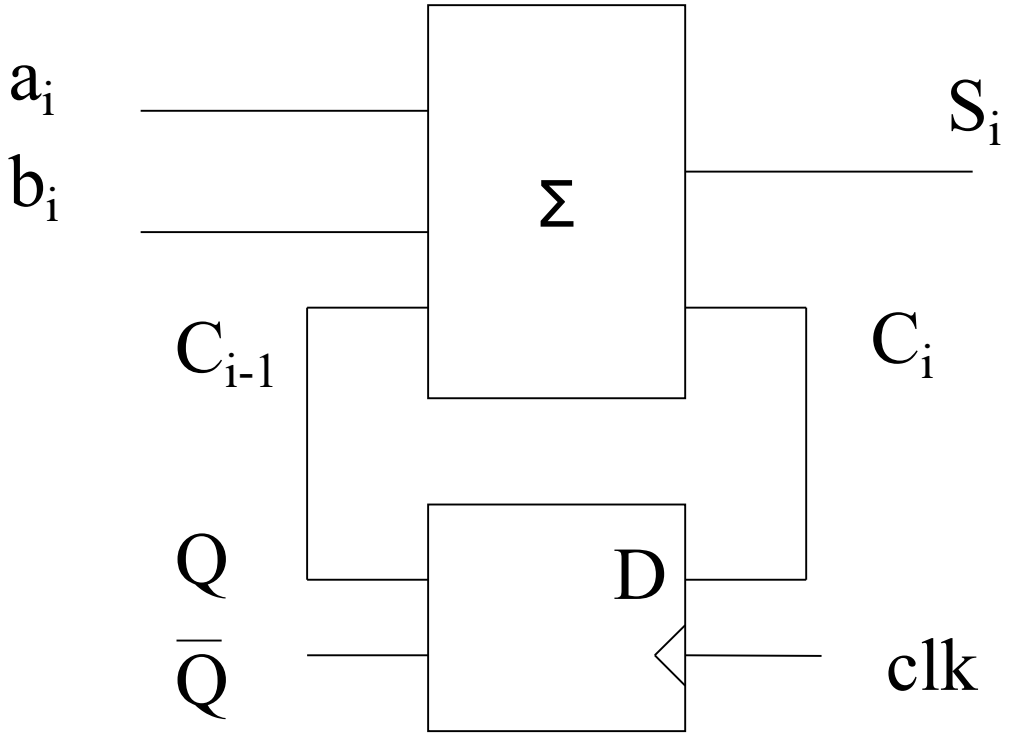
数据通路

```
// bus
trin tri ext (Data, Extern, BusWires);
regn reg 0 (BusWires, Rin[0], Clock, R0);
regn reg 1 (BusWires, Rin[1], Clock, R1);
regn reg 2 (BusWires, Rin[2], Clock, R2);
regn reg 3 (BusWires, Rin[3], Clock, R3);

trin tri 0 (R0, Rout[0], BusWires);
trin tri 1 (R1, Rout[1], BusWires);
trin tri 2 (R2, Rout[2], BusWires);
trin tri 3 (R3, Rout[3], BusWires);
regn reg A (BusWires, Ain, Clock, A);

// alu unit
always @(AddSub, A, BusWires)
        if (!AddSub)
                Sum = A + BusWires;
        else
                Sum = A - BusWires;

regn reg G (Sum, Gin, Clock, G);
trin tri G (G, Gout, BusWires);
```

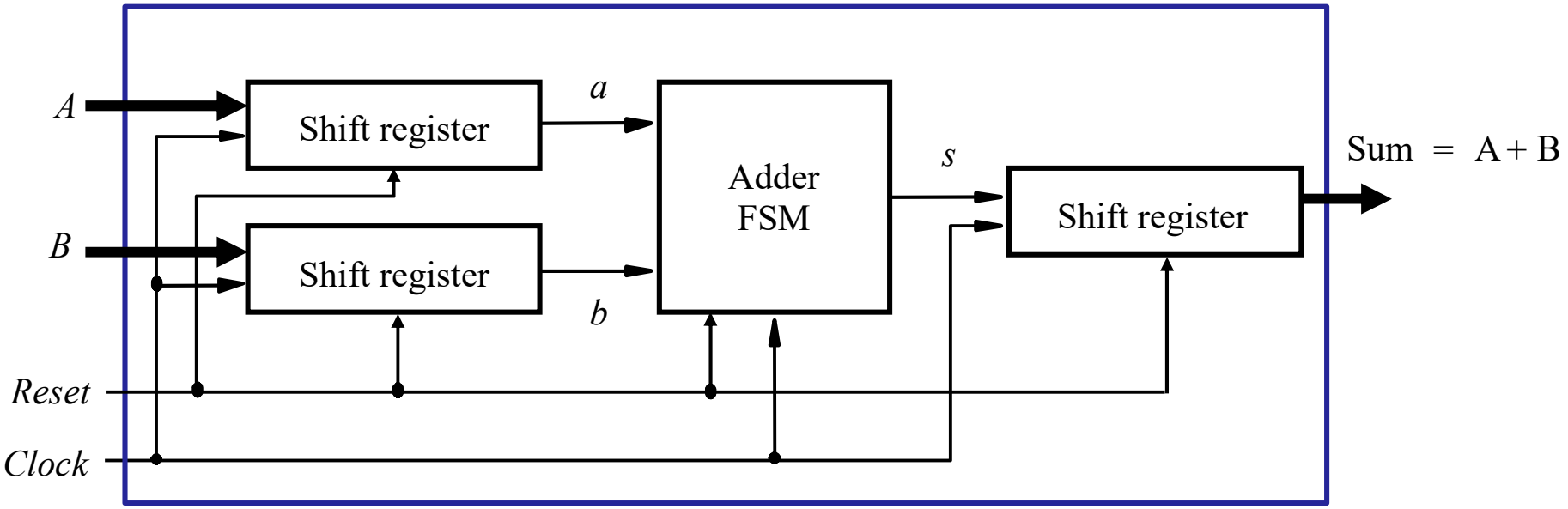


目录

串行加法器

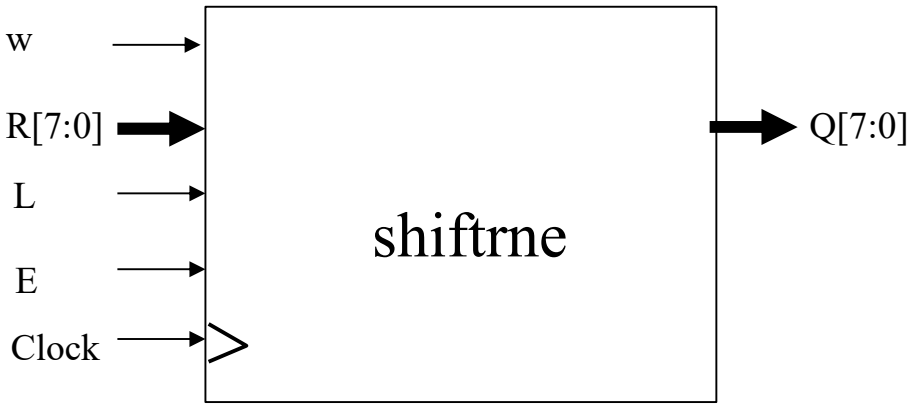


串行进位加法器



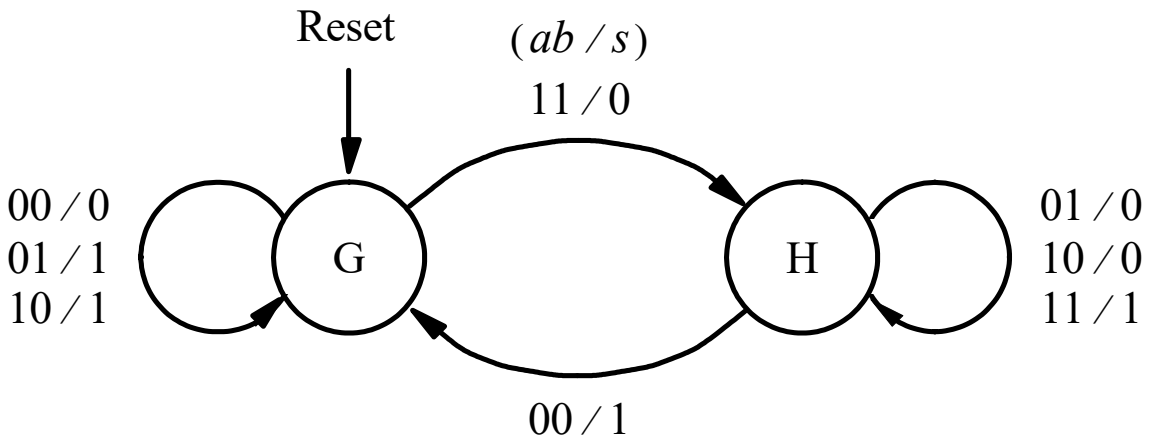
- 输入信号
  - A：8位
  - B：8位
  - Clock：1位
  - Reset: 1 位
- 输出信号：
  - Sum：8位

8位移位寄存器



状态机（Mealy）

G: carry-in = 0
H: carry-in = 1



	Present state	Next state				Output			
		ab =00	01	10	11	00	01	10	11
	y	*Y*				*s*			
G	0	0	0	0	1	0	1	1	0
H	1	0	1	1	1	1	0	0	1

状态机（Mealy）

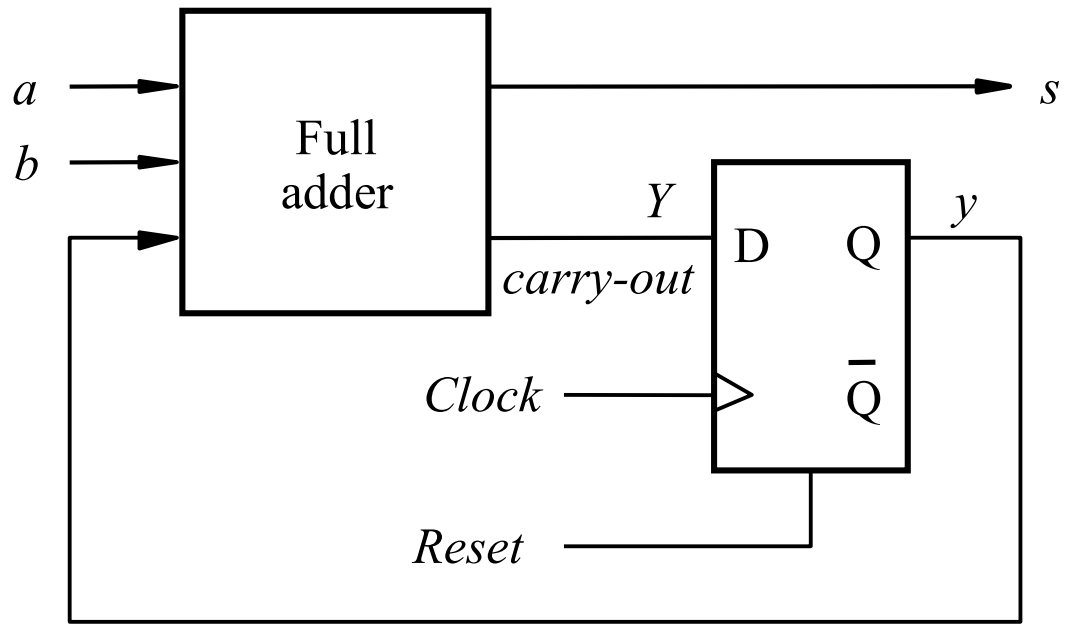


Figure 6.43. Circuit for the adder FSM in Figure 6.39.

HDL描述-移位寄存器

```
module shiftrne (R, L, E, w, Clock, Q);

    input [7:0] R;
    input L, E, w, Clock;
    output reg [7:0] Q;

    always @(posedge Clock)
        if (L)
            Q <= R;
        else if (E)
        begin
            Q[7] <= w;
            Q[6:0] <= Q[7:1];
        end

endmodule
```

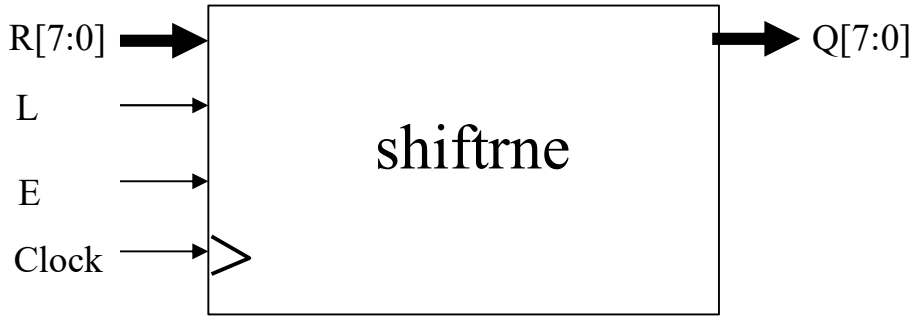
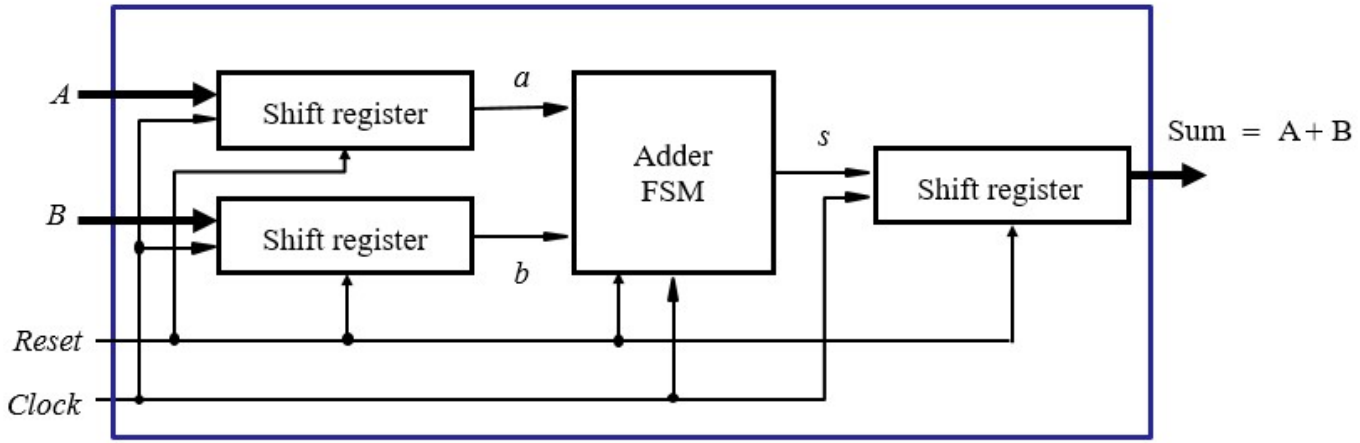


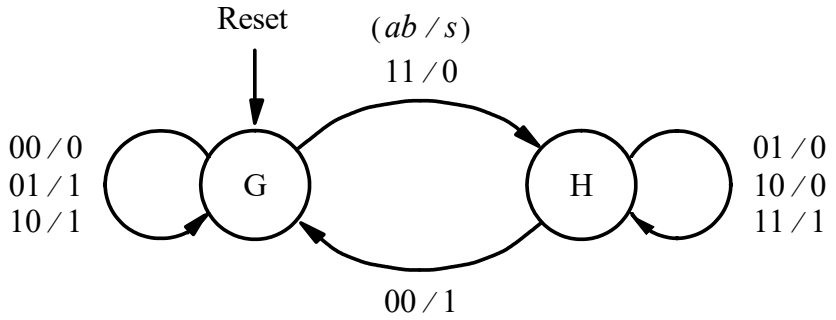
Figure 6.48. Code for a left-to-right shift register with an enable input.

HDL描述

```
module serial_adder (A, B, Reset, Clock, Sum);
    input [7:0] A, B;
    input Reset, Clock;
    output wire [7:0] Sum;
    reg [3:0] Count;
    reg s, y, Y;
    wire [7:0] QA, QB;
    wire Run;
    parameter G = 1'b0, H = 1'b1;

    shiftrne shift_A (A, Reset, 1'b1, 1'b0, Clock, QA);
    shiftrne shift_B  (B, Reset, 1'b1, 1'b0, Clock, QB);
    shiftrne shift_Sum  (8'b0, Reset, Run, s, Clock, Sum);  // shift register
```



```
// Adder FSM
// Output and next state combinational circuit
always @(QA, QB, y)
    case (y)
        G: begin
            s = QA[0] ^ QB[0];
            if (QA[0] & QB[0])   Y = H;
            else  Y = G;
            end
        H: begin
            s = QA[0] ~^ QB[0];
            if (~QA[0] & ~QB[0])        Y = G;
            else    Y = H;
            end
        default: Y = G;
    endcase

// Sequential block
always @(posedge Clock)
    if (Reset) y <= G;
    else       y <= Y;

// Control the shifting process
always @(posedge Clock)
    if (Reset)       Count = 8;
    else if (Run)    Count = Count - 1;
assign Run = |Count;

endmodule
```



G: carry-in = 0
H: carry-in = 1

Present state	Next state				Output			
	ab =00	01	10	11	00	01	10	11
y	*Y*				*s*			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

综合结果

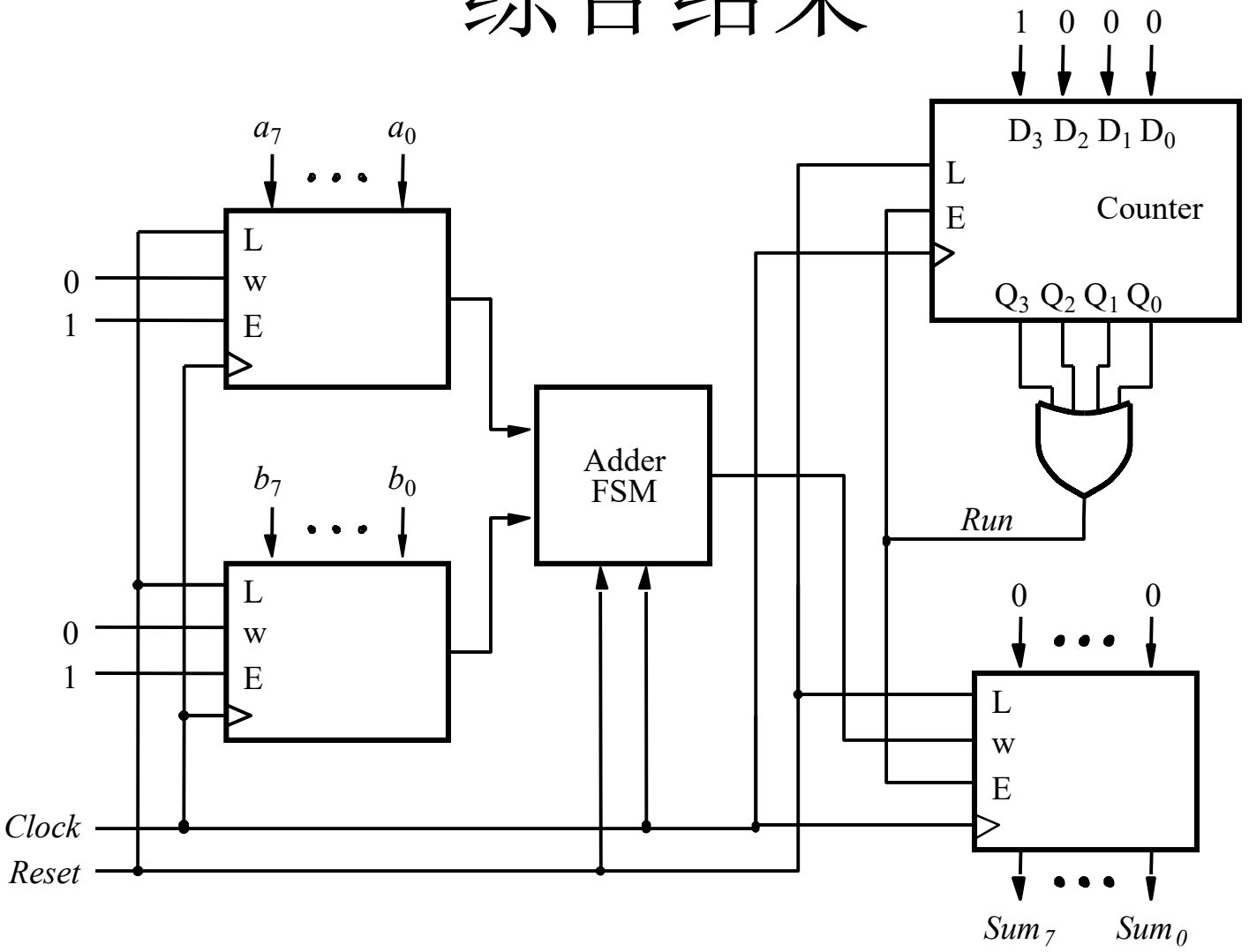
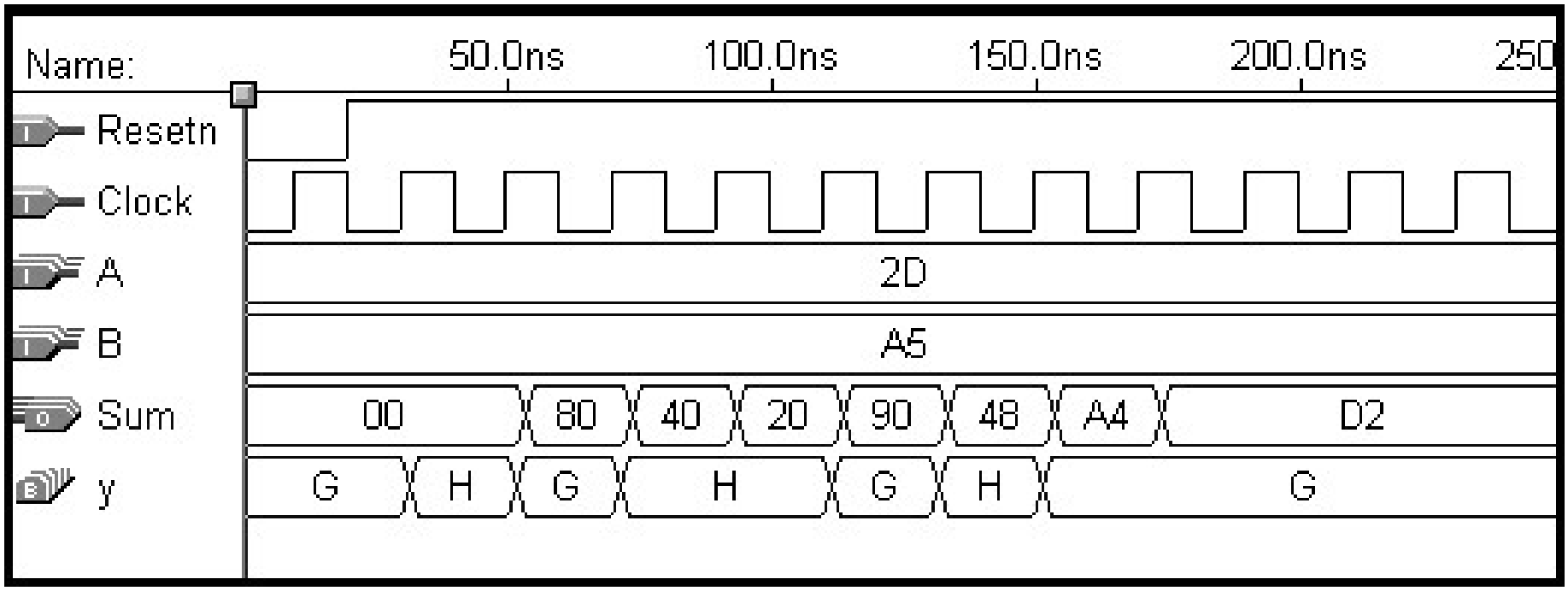


Figure 6.50. Synthesized serial adder.

仿真结果


Name:
50.0ns
100.0ns
150.0ns
200.0ns
250
Resetn
Clock
A
2D
B
A5
Sum
00
80
40
20
90
48
A4
D2
y
G
H
G
H
G
H
G

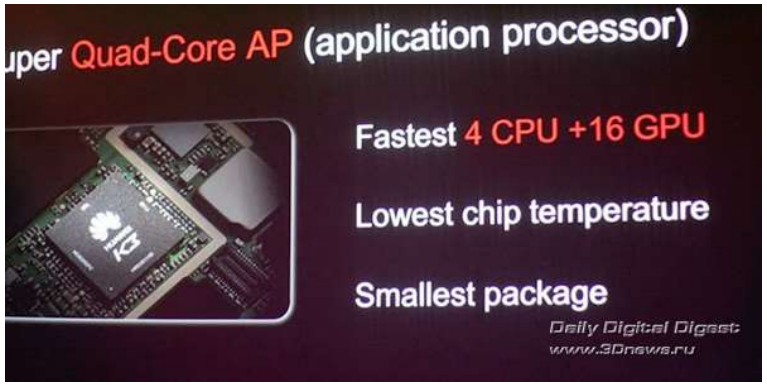
问题引入

华为P6 2013年

华为mate40Pro 2020年

问题引入

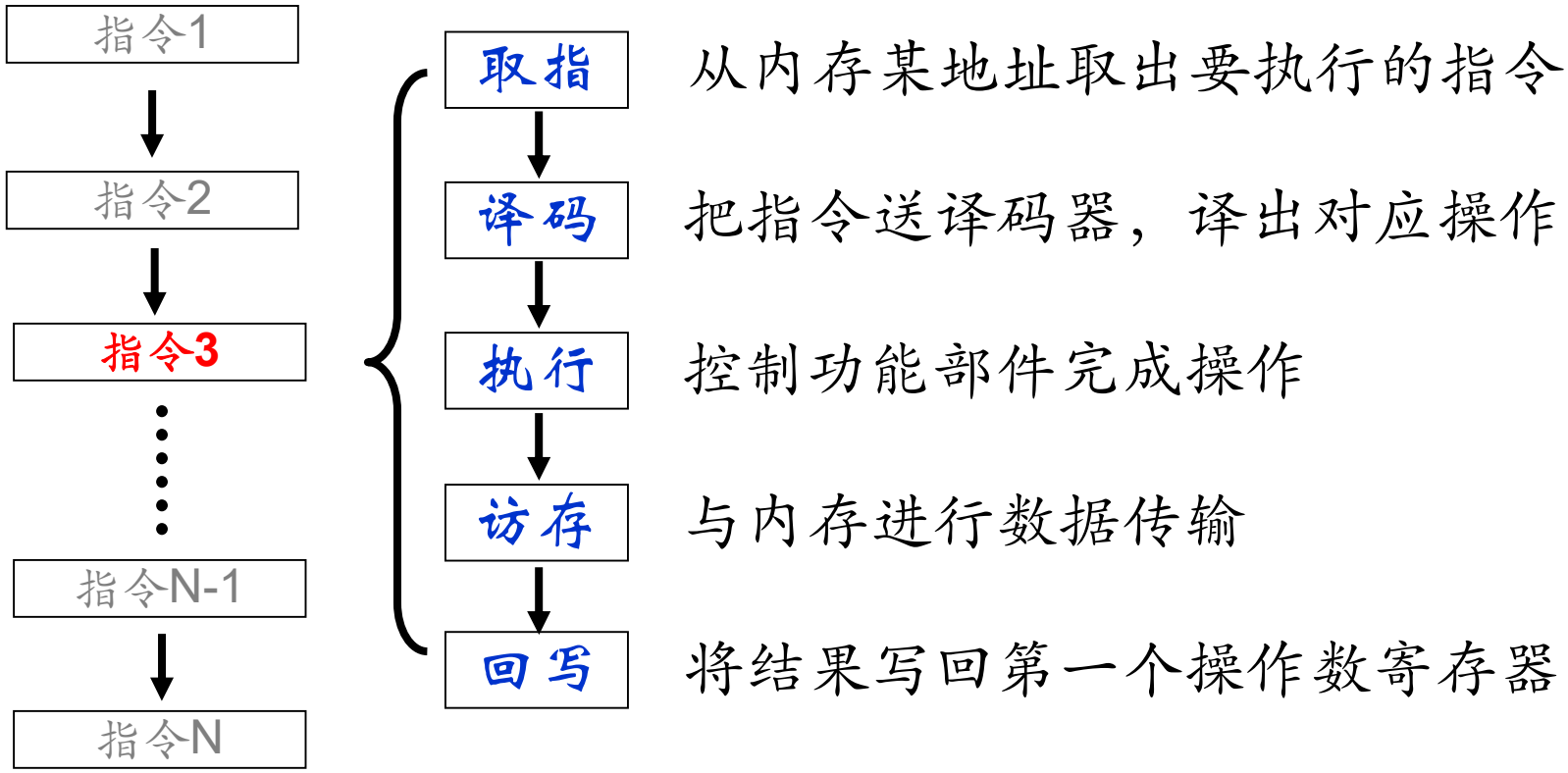
- 数字系统性能优化的主要方法
  - 提升频率 1.5Ghz—>3.13Ghz：
  - 提高吞吐量
    - 多核：提升芯片吞吐量，四核—>八核
    - 流水线：提升单核的吞吐量



麒麟K3V2: 四核 1.5GHz

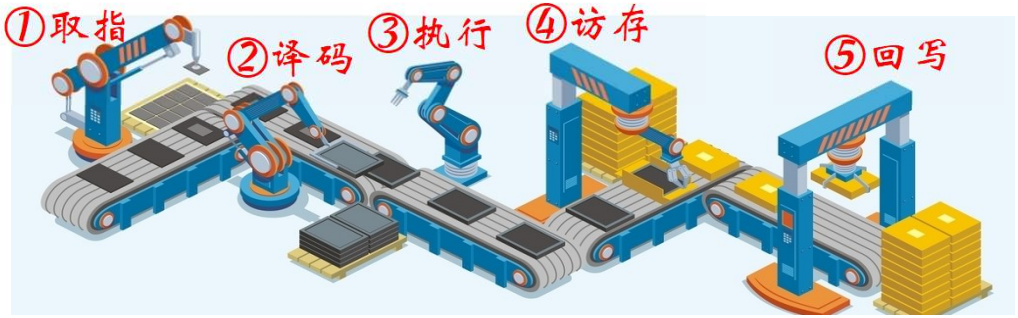
麒麟9000: 八核 3.13GHz

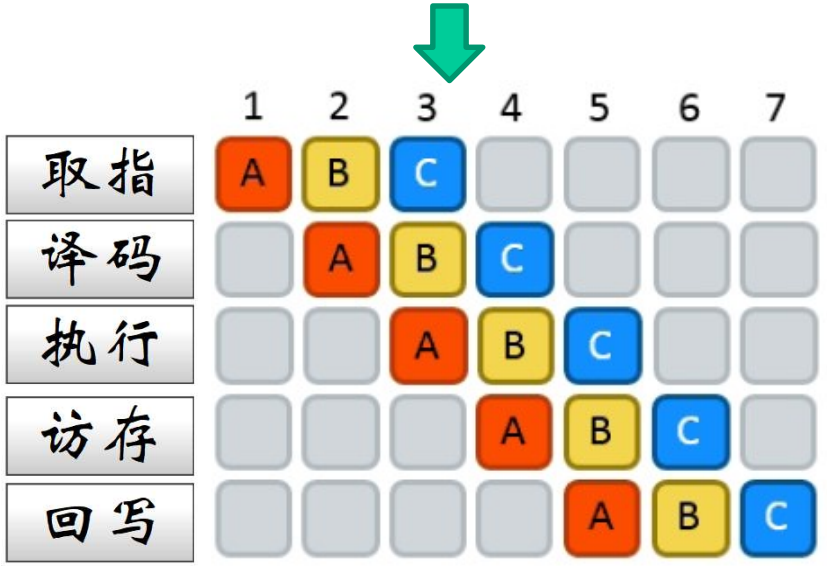
流水线技术的基本原理


指令1
指令2
指令3
指令N-1
指令N
取指
译码
执行
访存
回写
从内存某地址取出要执行的指令
把指令送译码器，译出对应操作
控制功能部件完成操作
与内存进行数据传输
将结果写回第一个操作数寄存器

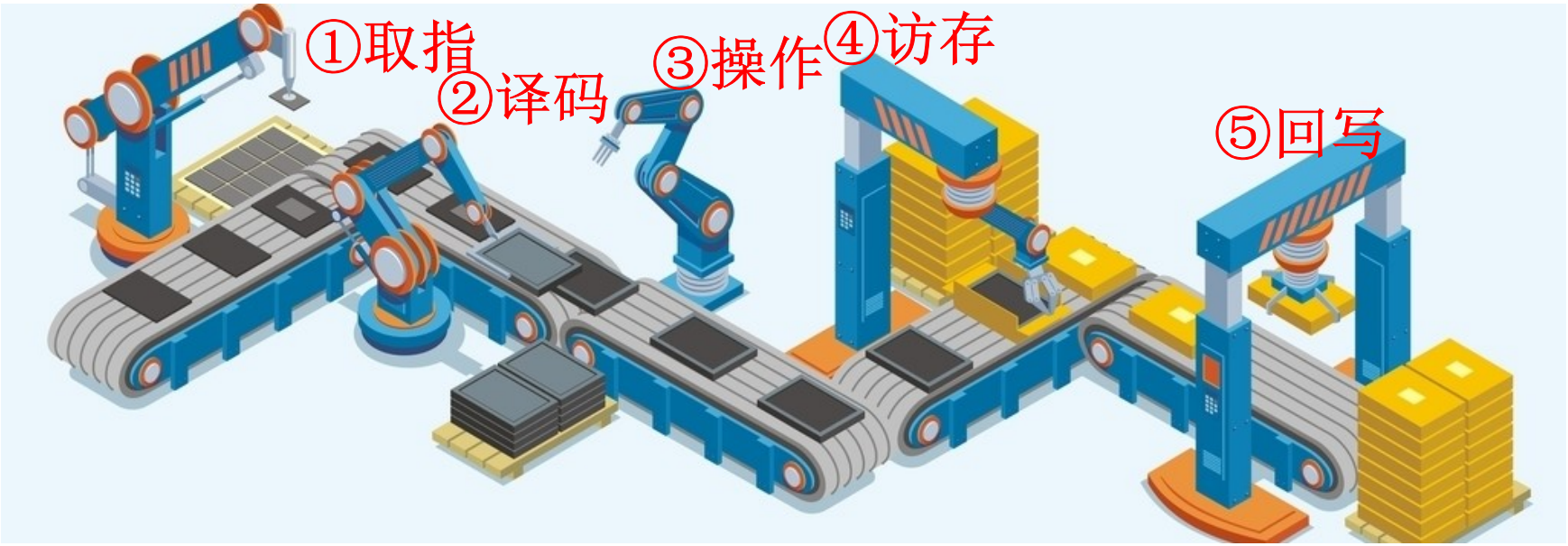
流水线技术的基本原理


周期
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
取指 A B C
译码 A B C
执行 A B C
访存 A B C
回写 A B C


①取指
②译码
③执行
④访存
⑤回写


1 2 3 4 5 6 7
取指 A B C
译码 A B C
执行 A B C
访存 A B C
回写 A B C

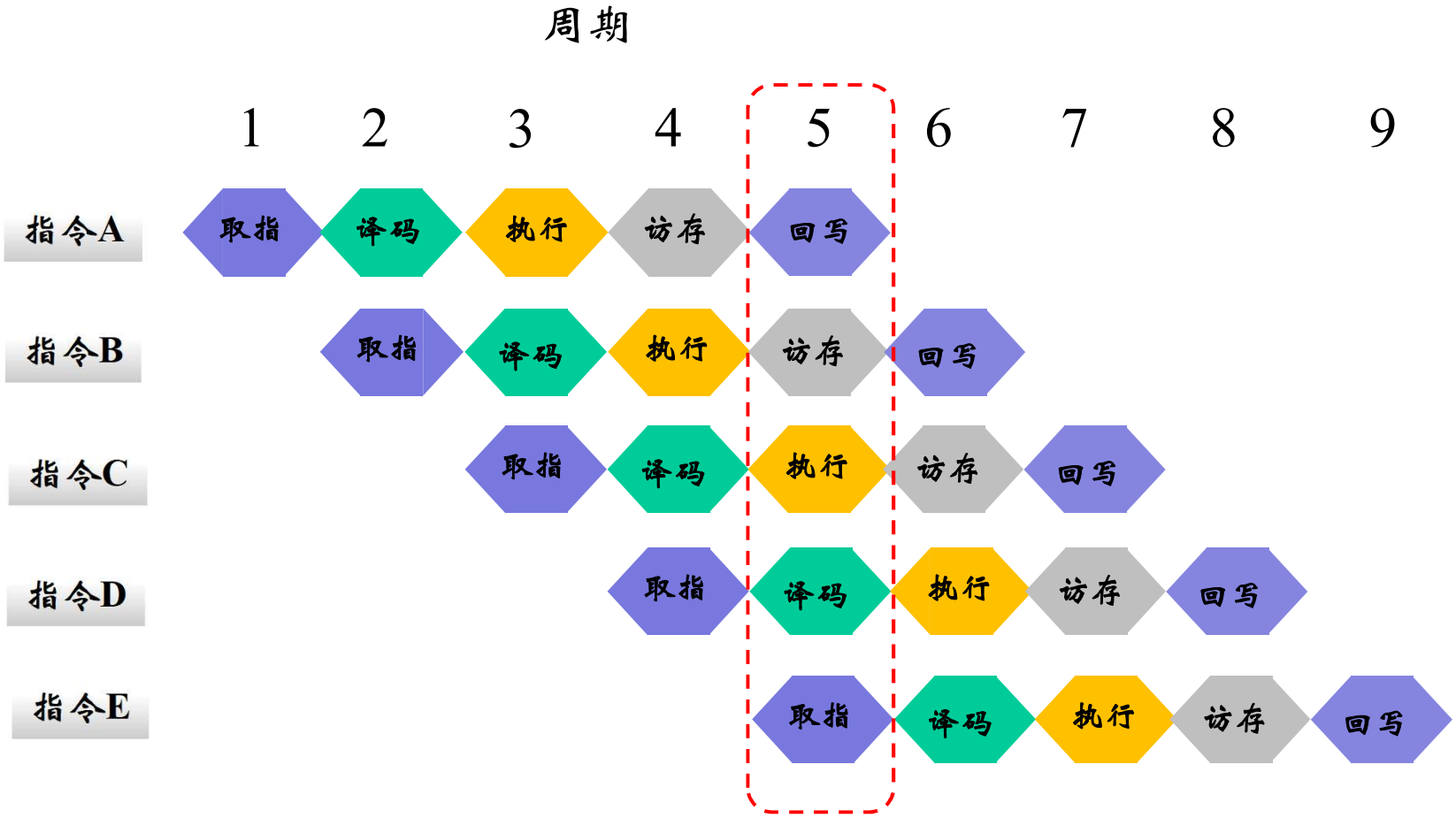
流水线结构设计的关键点



流水线设计的关键：

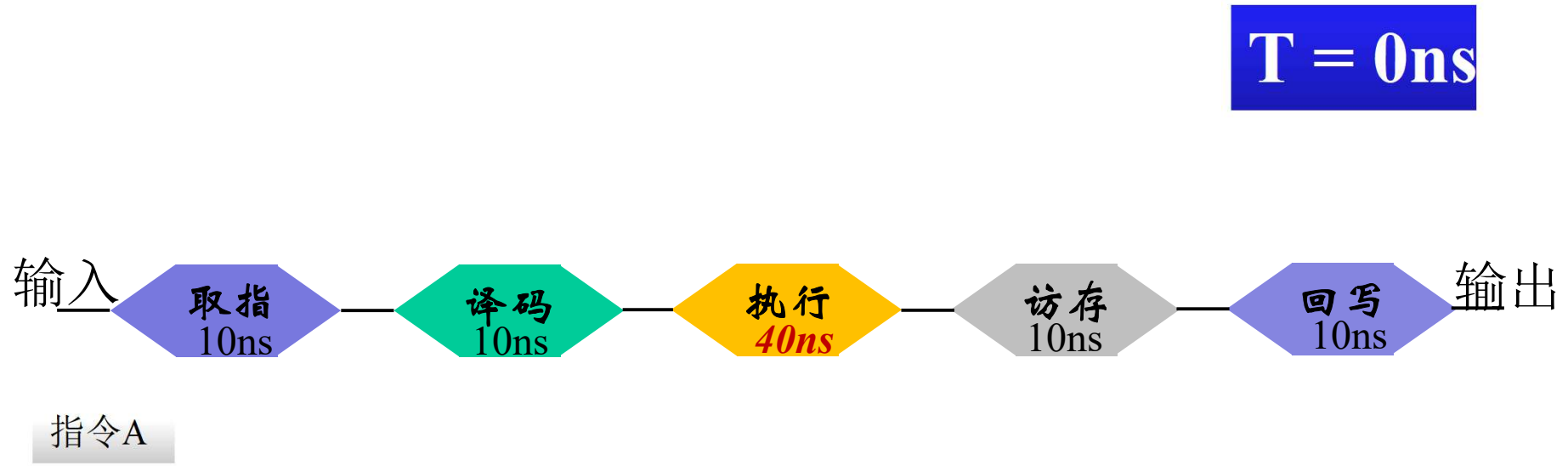
①顺序拆分：按先后顺序将一个完整操作过程拆分成若干步骤

流水线结构设计的关键点



指令执行速度提升了约5倍？

流水线结构设计的关键点



流水线结构设计的关键点

T = 10ns

指令A

A-1

指令B

流水线结构设计的关键点

T = 20ns

指令A　　A-2

指令B　　B-1

指令C

流水线结构设计的关键点

T = 30ns

指令A

指令B

指令C

指令D

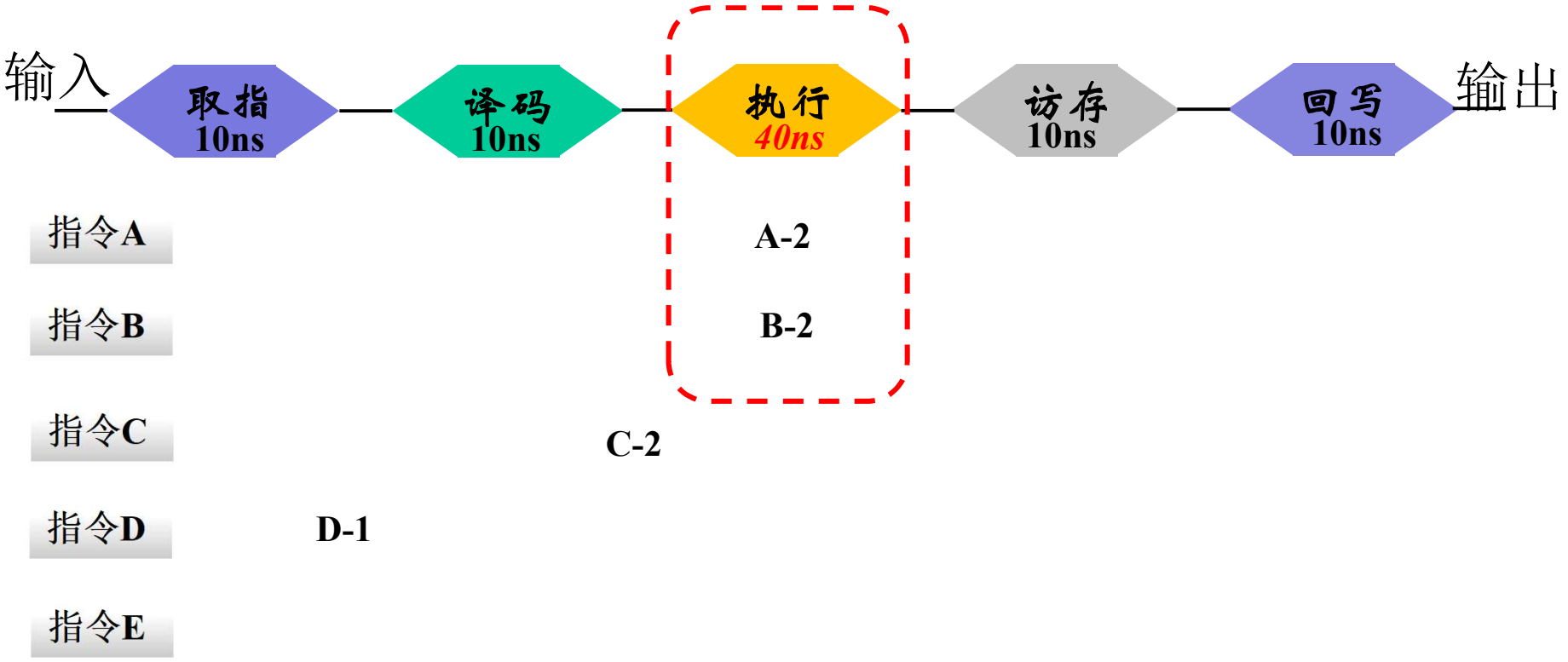
A-2

B-2

C-1

流水线结构设计的关键点

T = 40ns



流水线结构设计的关键点

T = 40ns

指令A

指令B

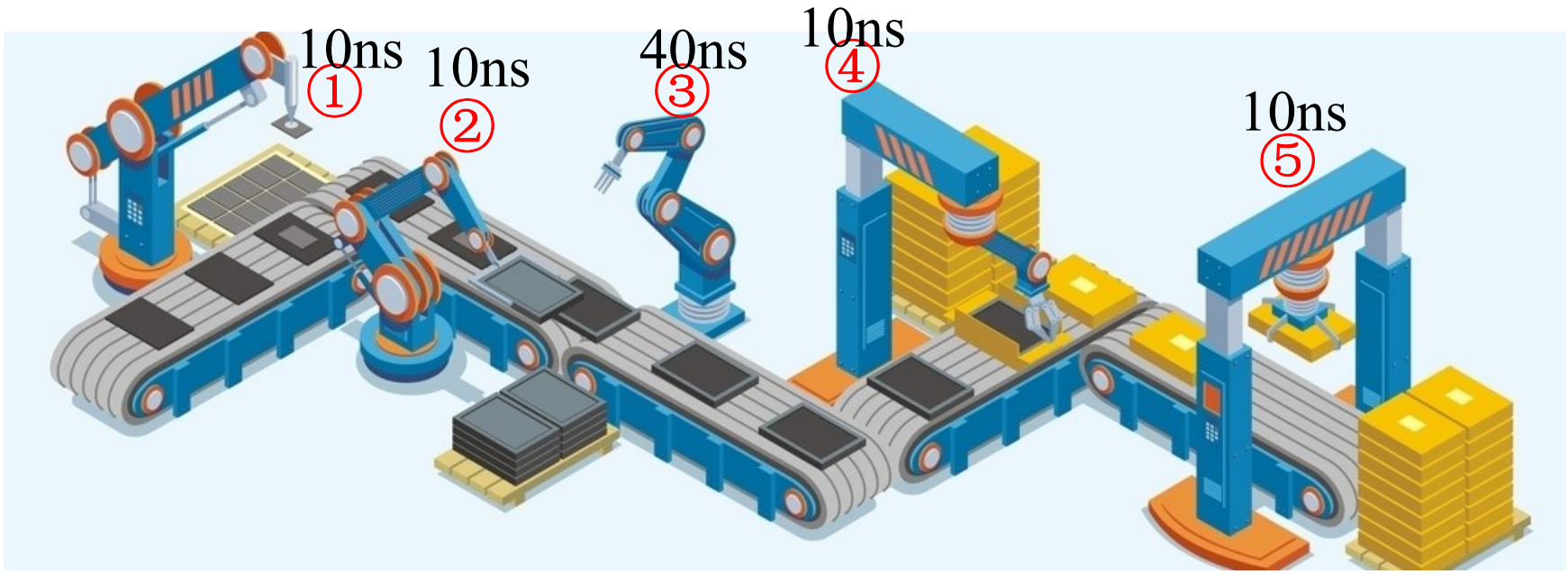
A-2

流水线结构： **40ns**/指令

非流水线结构： **80ns**/指令

指令执行速度提升了约2倍

流水线结构设计的关键点



流水线设计的关键：

①顺序拆分：按先后顺序将一个完整操作过程拆分成若干步骤

②木桶原则：最高频率取决于耗时最长的步骤

流水线结构设计的关键点

T = 40ns

指令A

指令B

A-2

流水线结构设计的关键点

T = 50ns

指令A　　A-2

指令B　　B-1

流水线结构设计的关键点

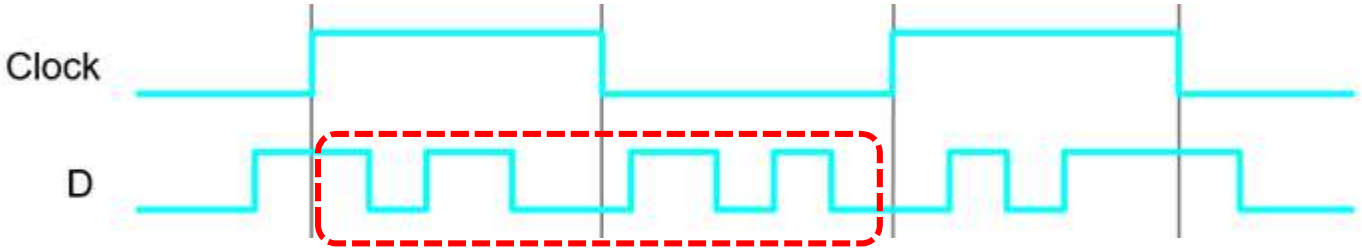
50ns<T<60ns

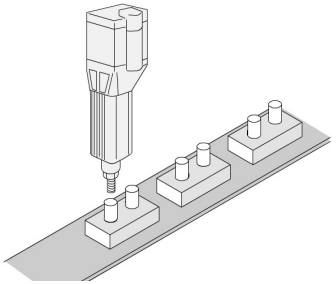
指令A

A-2

指令B

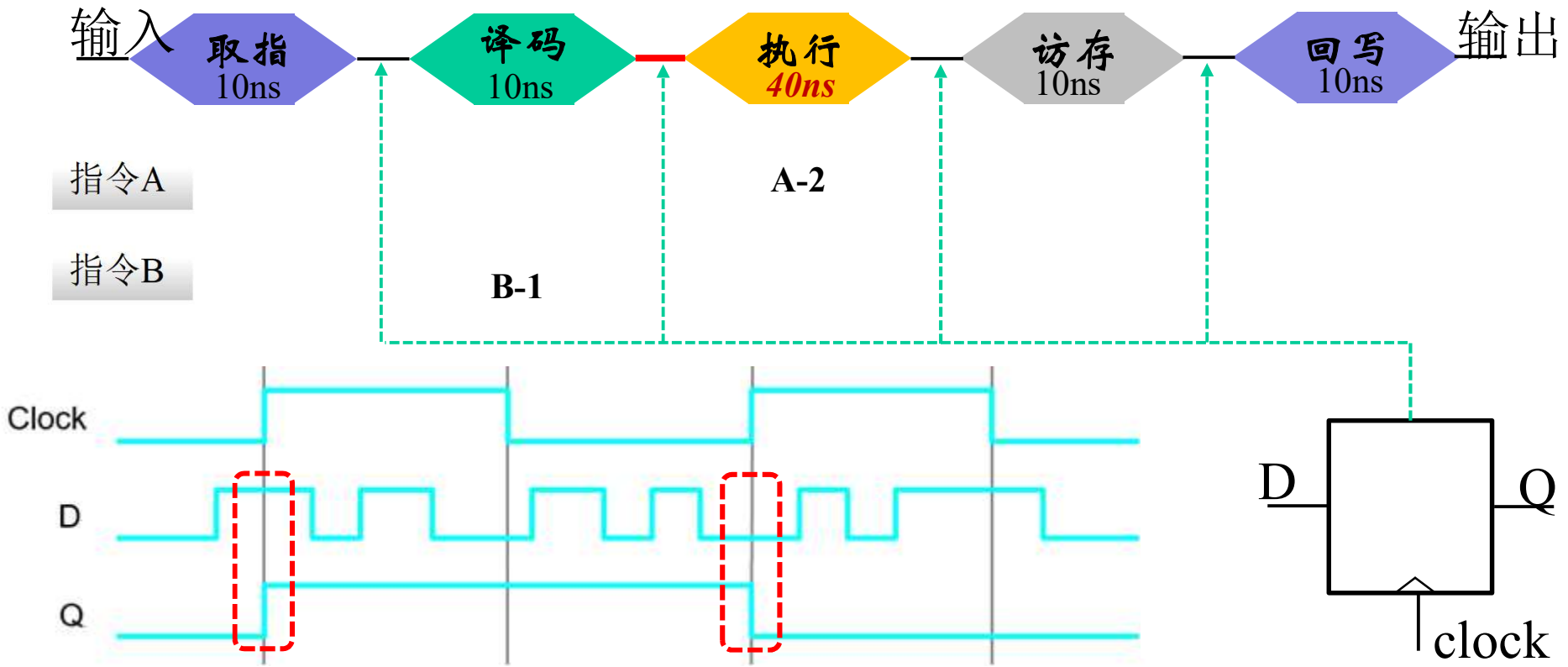
B-1





流水线结构设计的关键点

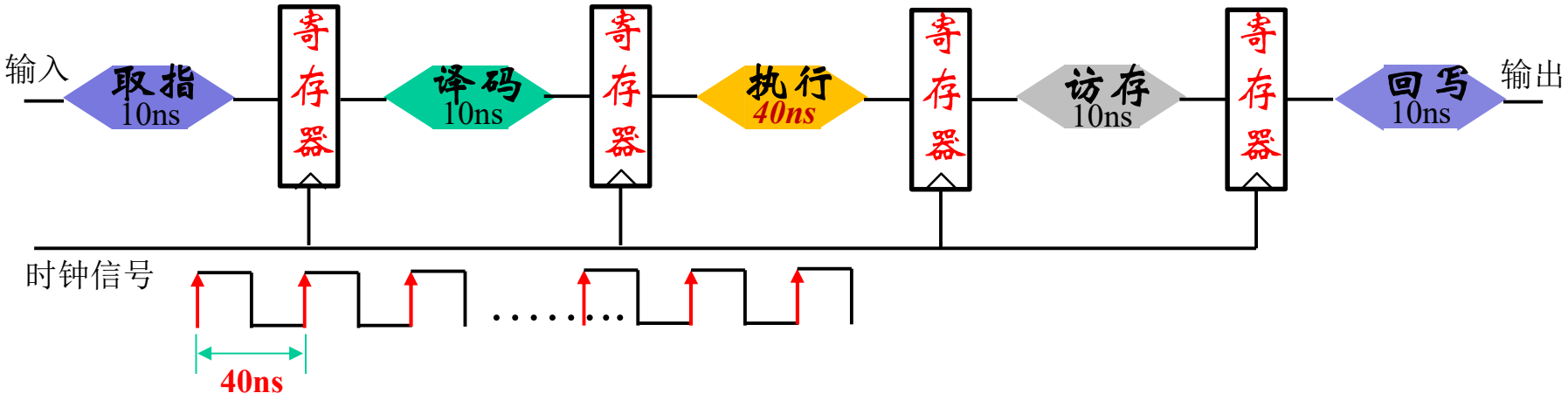
50ns<T<60ns



同步传递：插入寄存器

流水线结构设计的关键点

流水线的实现方法：



流水线结构设计的关键点

流水线设计的关键：

①顺序拆分：按先后顺序将一个完整操作过程拆分成若干步骤

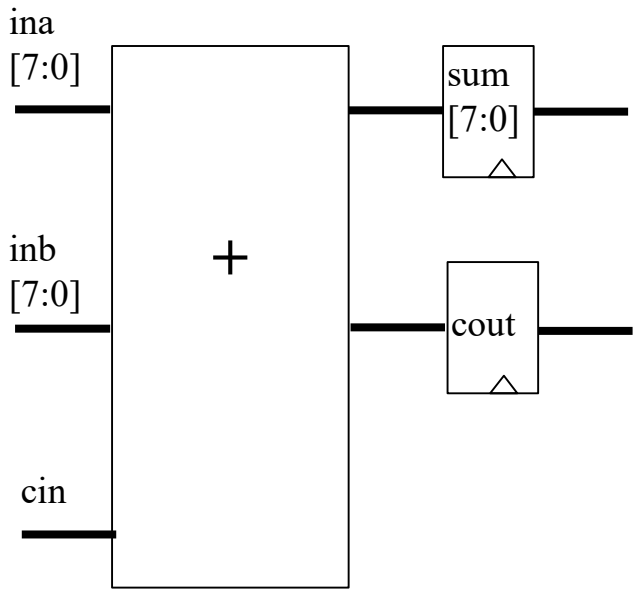
②木桶原则：最高频率取决于耗时最长的步骤

③同步传递：插入寄存器

流水线结构计实例

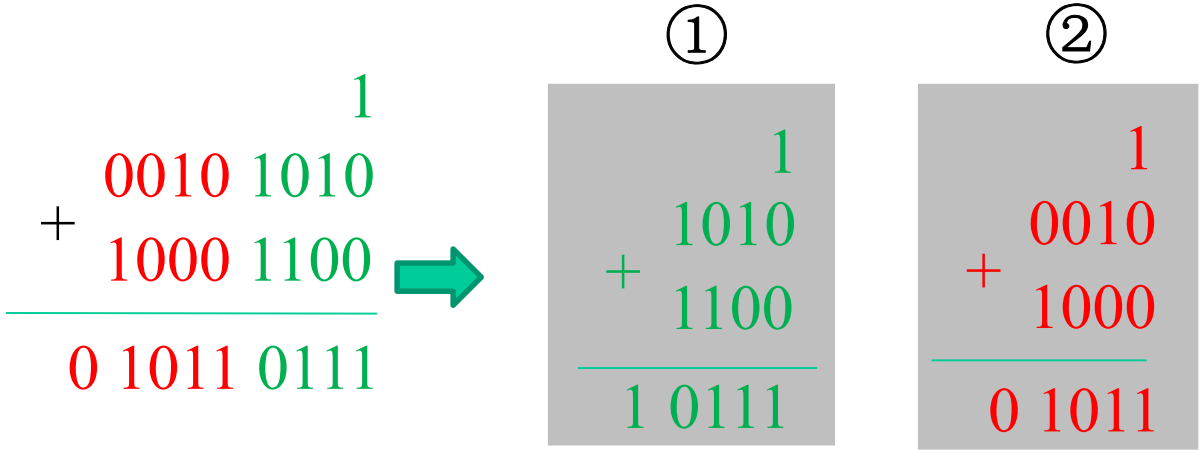
8位全加器:非流水线 —> 2级流水线

```
          1
  0010 1010
+ 1000 1100
-----------
0 1011 0111
```



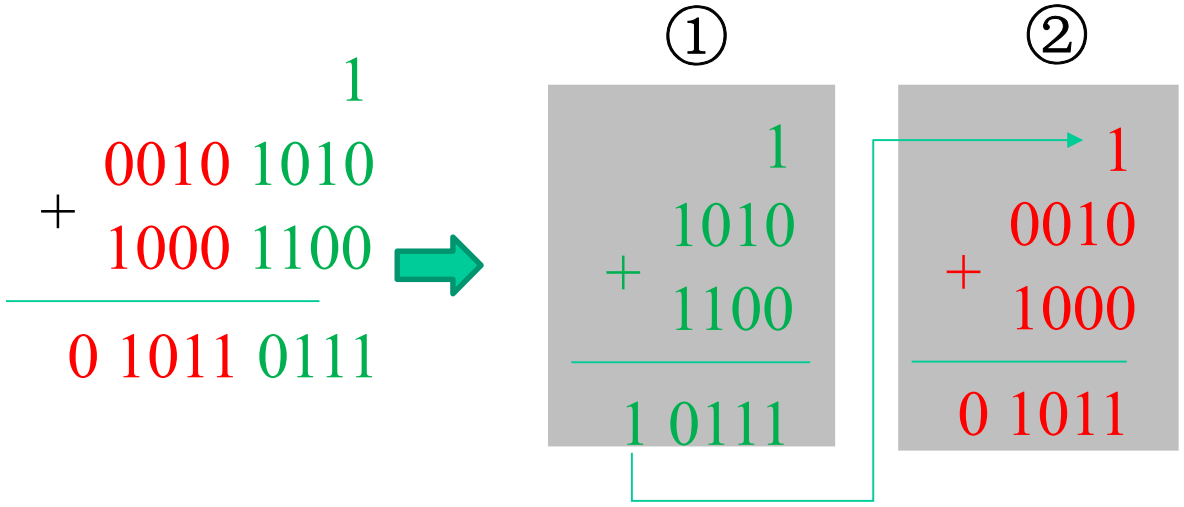
流水线结构计实例

- 顺序拆分
- 木桶原则



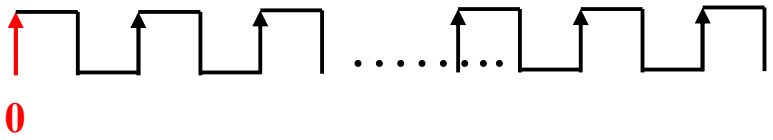
流水线结构计实例

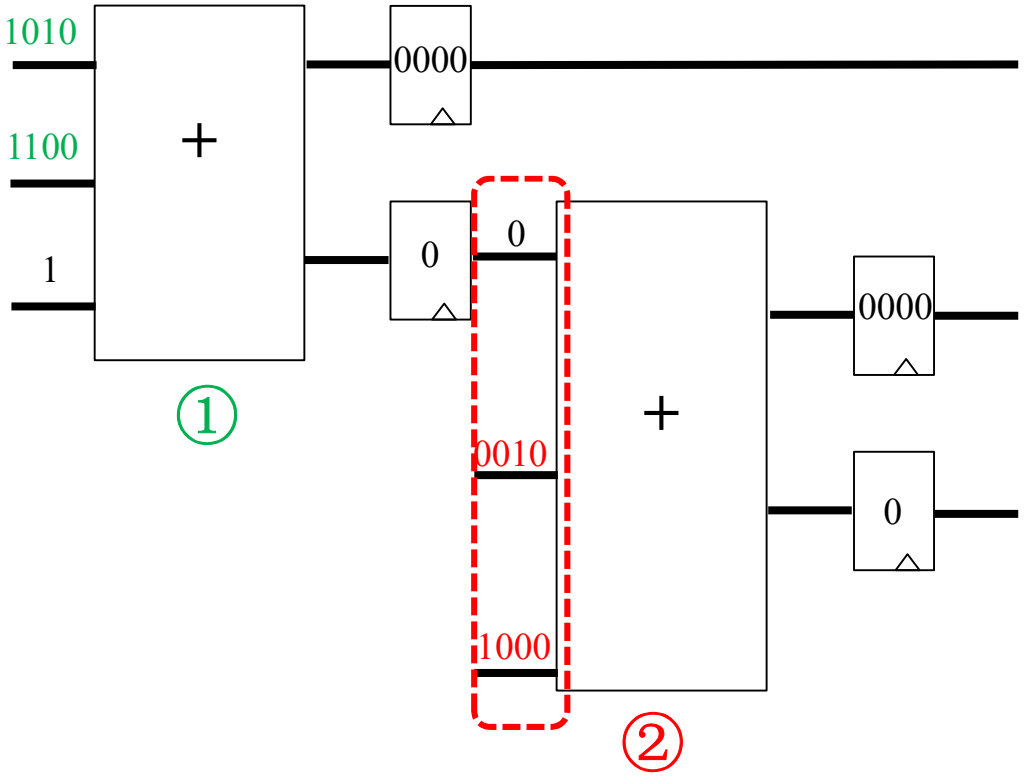
- 顺序拆分
- 木桶原则
- 同步传递



流水线结构计实例

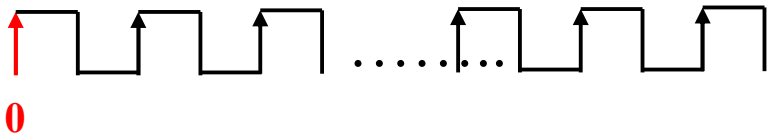
时刻	cin	ina[7:0]	inb[7:0]
0	1	0010 1010	1000 1100
t	0	1000 0001	0001 1000
2t	0	1001 0010	0010 1001

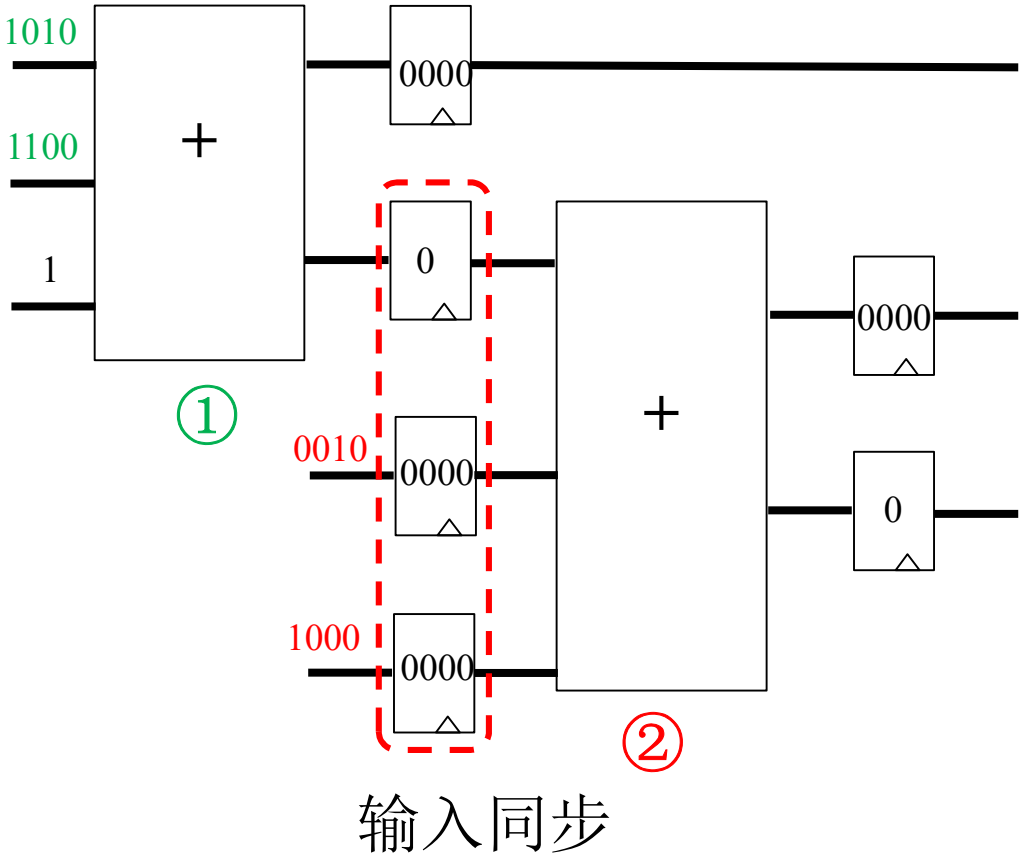




流水线结构计实例

时刻	cin	ina[7:0]	inb[7:0]
0	1	0010 1010	1000 1100
t	0	1000 0001	0001 1000
2t	0	1001 0010	0010 1001

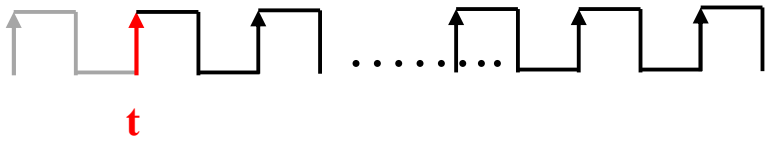


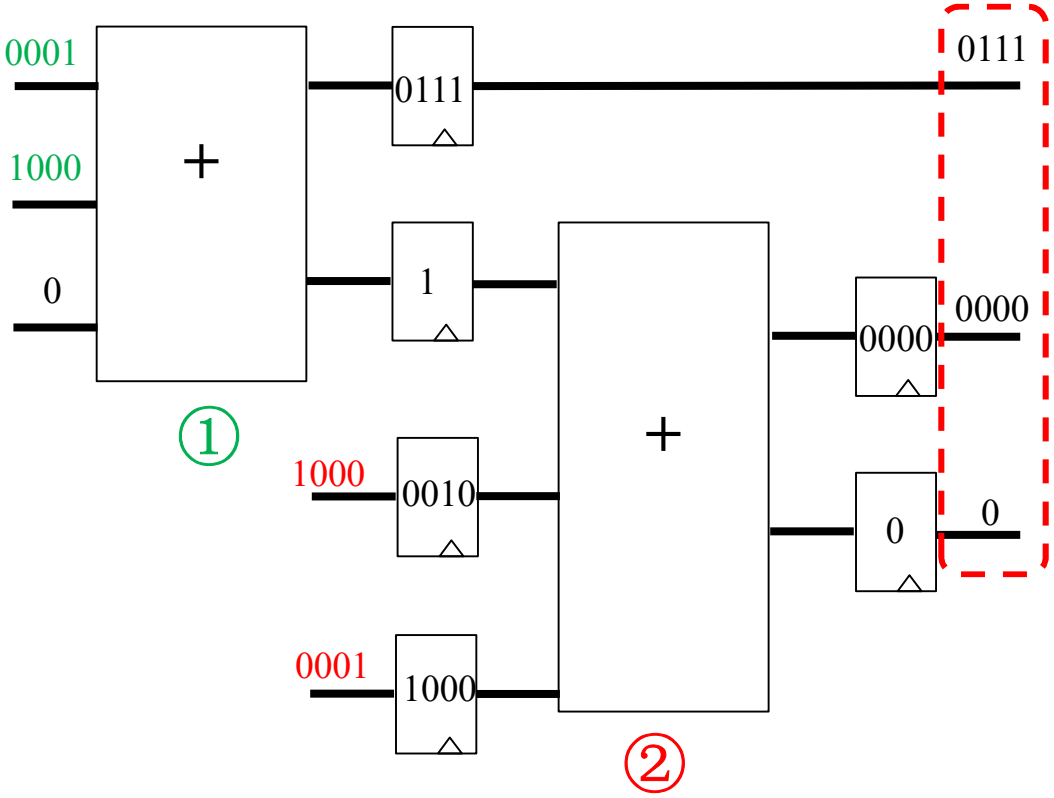


输入同步

流水线结构计实例

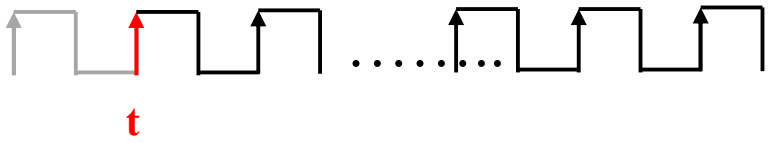
时刻	cin	ina[7:0]	inb[7:0]
0	1	0010 1010	1000 1100
t	0	1000 0001	0001 1000
2t	0	1001 0010	0010 1001

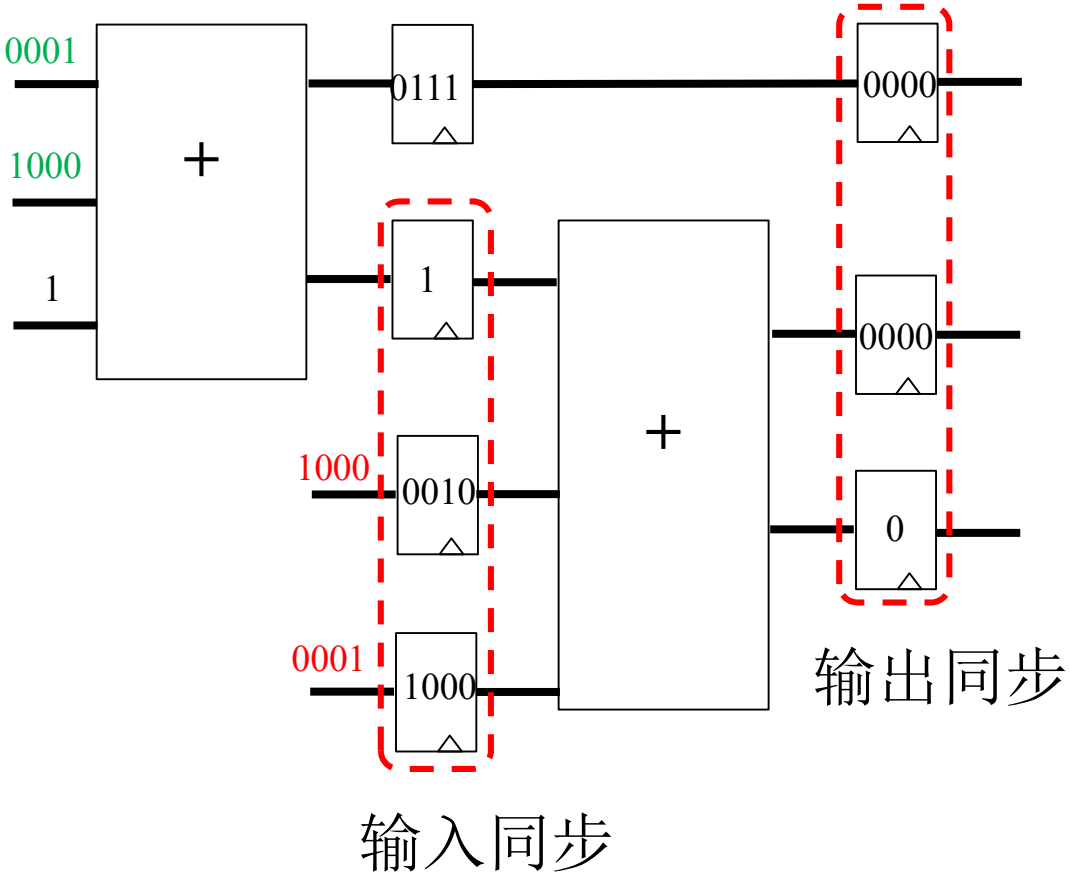




流水线结构计实例

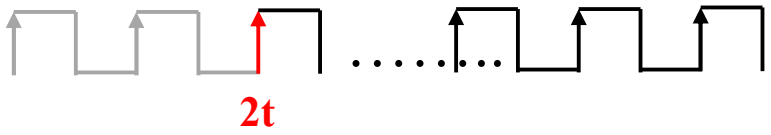
时刻	cin	ina[7:0]	inb[7:0]
0	1	0010 1010	1000 1100
t	0	1000 0001	0001 1000
2t	0	1001 0010	0010 1001

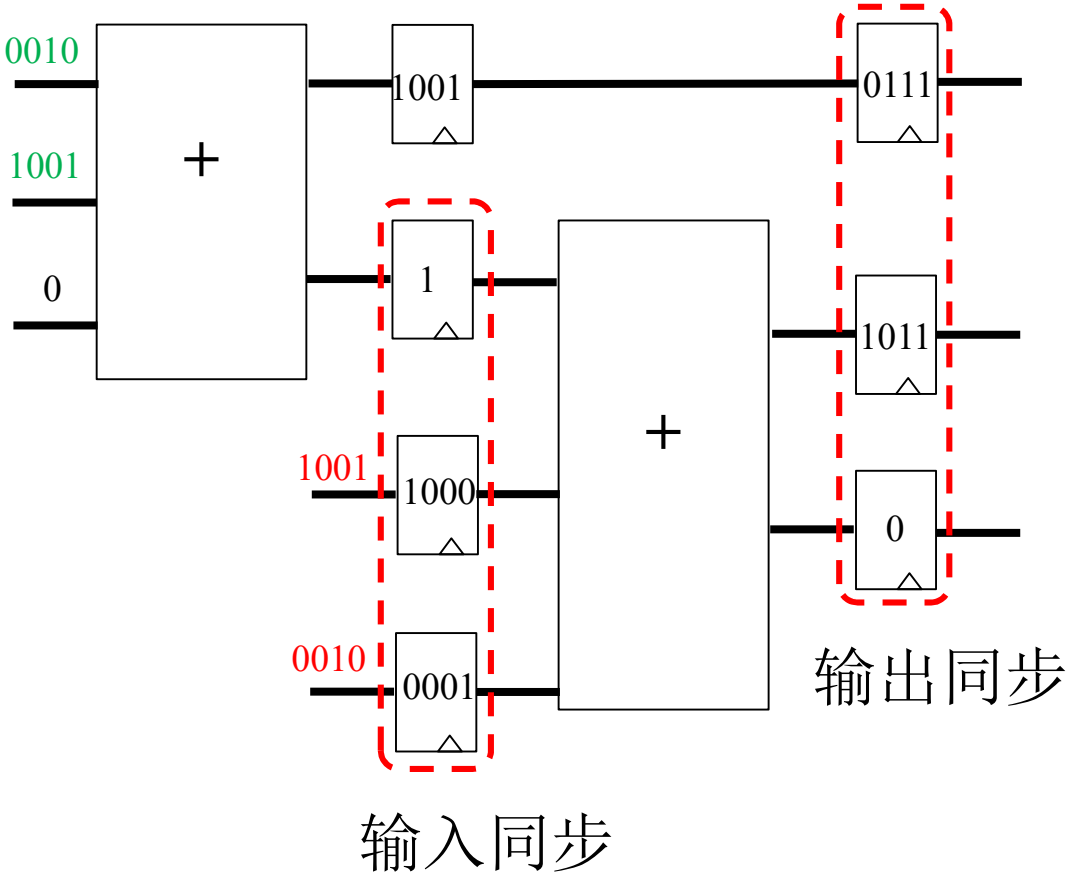




流水线结构计实例

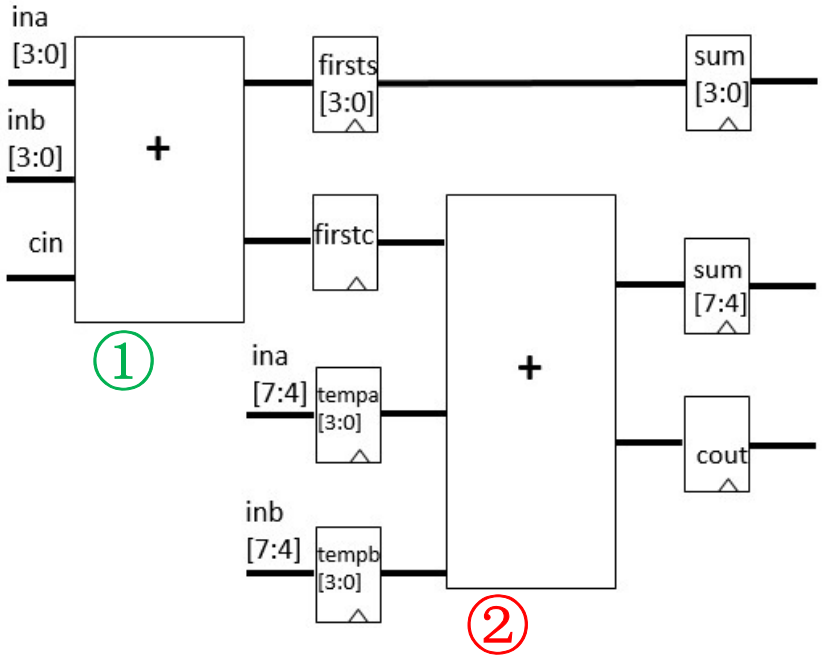
时刻	cin	ina[7:0]	inb[7:0]
0	1	0010 1010	1000 1100
t	0	1000 0001	0001 1000
2t	0	1001 0010	0010 1001





流水线结构计实例

```
module adder_pipe2(cout,sum,ina,inb,cin,clk);
input[7:0] ina,inb; input cin,clk; output reg[7:0] sum;
output reg cout; reg[3:0] tempa,tempb,firsts; reg firstc;

always @(posedge clk)
begin
    {firstc,firsts}=ina[3:0]+inb[3:0]+cin; //①
    tempa=ina[7:4];  //输入同步
    tempb=inb[7:4]; //输入同步
end

always @(posedge clk)
begin
    {cout,sum[7:4]}=tempa+tempb+firstc; // ②
    sum[3:0]=firsts; //输出同步
end
endmodule
```



总结

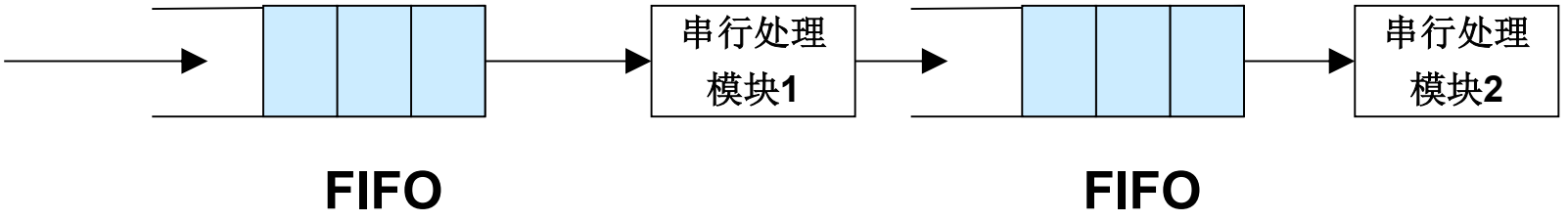
- 提升吞吐量是数字系统性能优化的主要方法
  - 多核
  - 流水线
- 流水线设计的关键
  - **①顺序拆分：**将一个组合逻辑过程拆分成若干步骤
  - **②木桶原则：**最高频率取决于耗时最长的步骤
  - **③同步传递：**在相邻操作步骤之间插入寄存器
- 流水线技术实现的注意事项
  - 各级输入输出数据同步

思考题：流水级越多越好吗？

串行设计

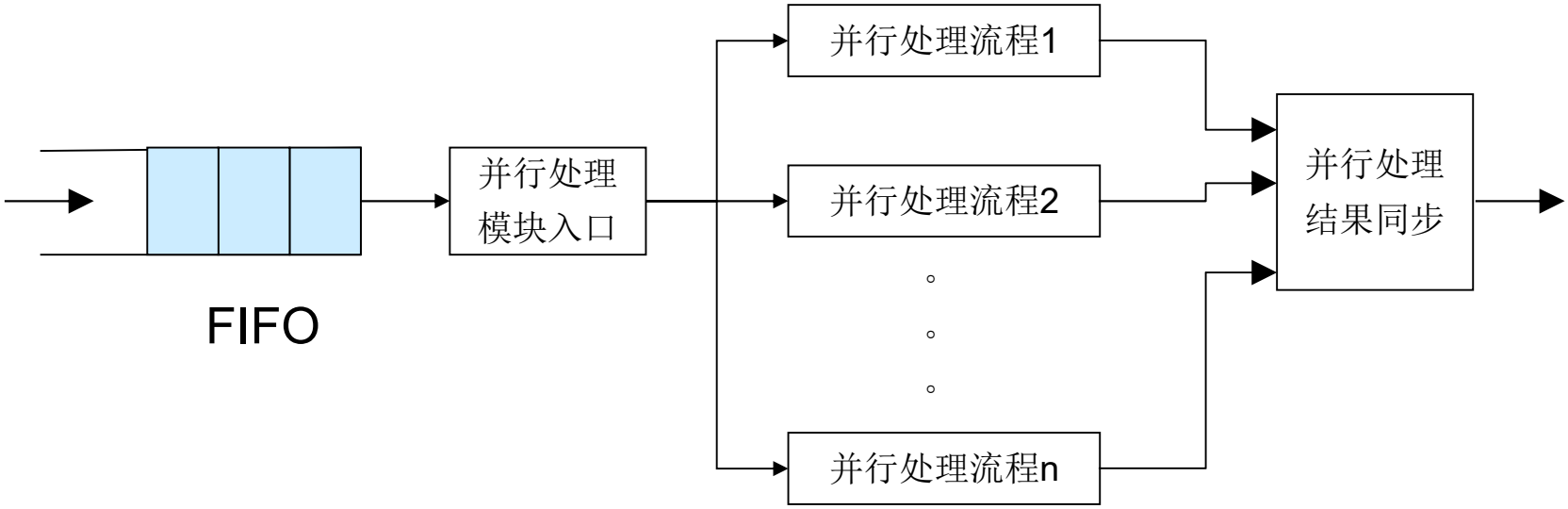
串行设计是最常见的一种设计；

- 当一个功能模块对输入的处理是分步骤进行的，并且后一个步骤只依赖前一个步骤的结果时，功能模块的设计就需要采用串行设计的思想。
- 一般采用FIFO（First In First Out）进行缓冲处理



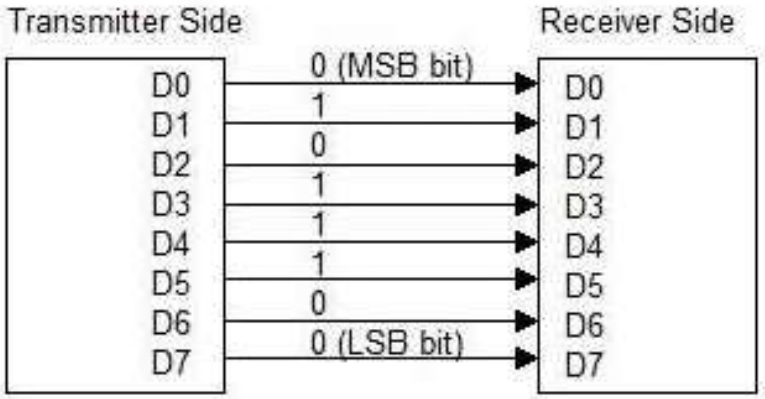
并行设计

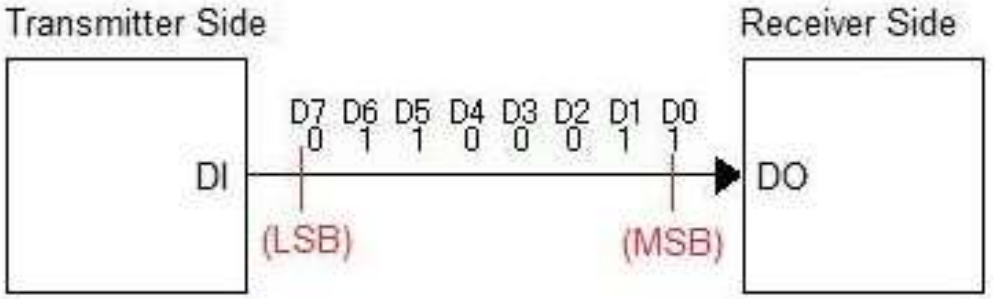
并行设计采用几个处理流程同时处理到达的负载，提高处理的效率，并行处理要求这些处理之间是独立的。



目录

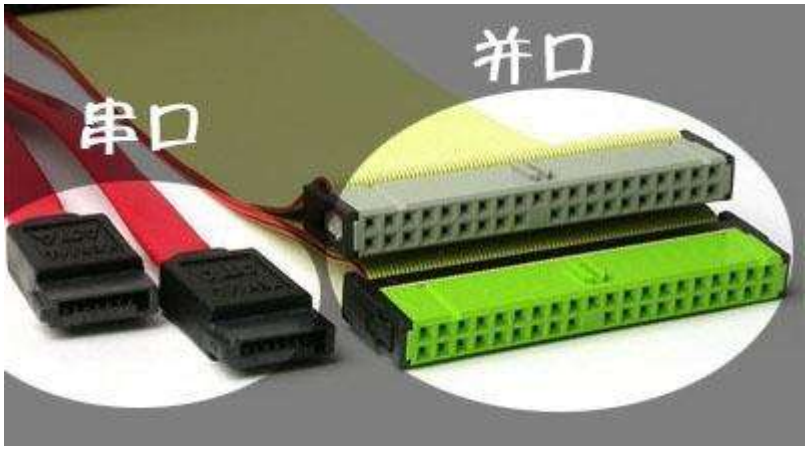
并行通信 VS. 串行通信


Transmitter Side
Receiver Side
D0
D1
D2
D3
D4
D5
D6
D7
0 (MSB bit)
1
0
1
1
1
0
0 (LSB bit)
D0
D1
D2
D3
D4
D5
D6
D7


Transmitter Side
Receiver Side
DI
DO
D7 D6 D5 D4 D3 D2 D1 D0
0 1 1 0 0 0 1 1
(LSB)
(MSB)

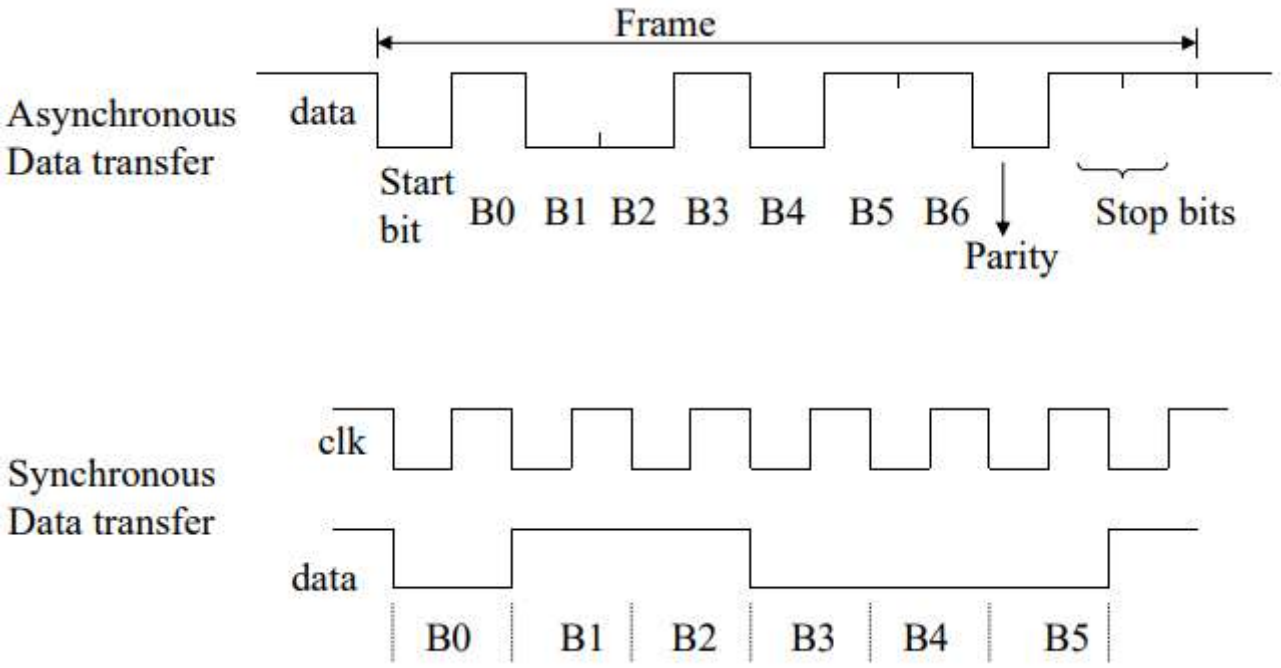
并行通信 VS. 串行通信

- 串行通信优势
  - 端口少
  - 长距离通信
  - 容易实现
- 串行通信劣势
  - 传输速率低
  - 需要额外开销（20%）



异步通信 VS. 同步

- 异步：无需时钟信号、需要额外数据位
- 同步：无需额外数据位、需要时钟信号



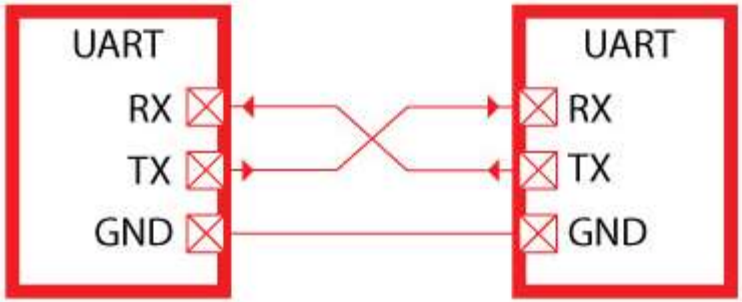
常用串行通信协议

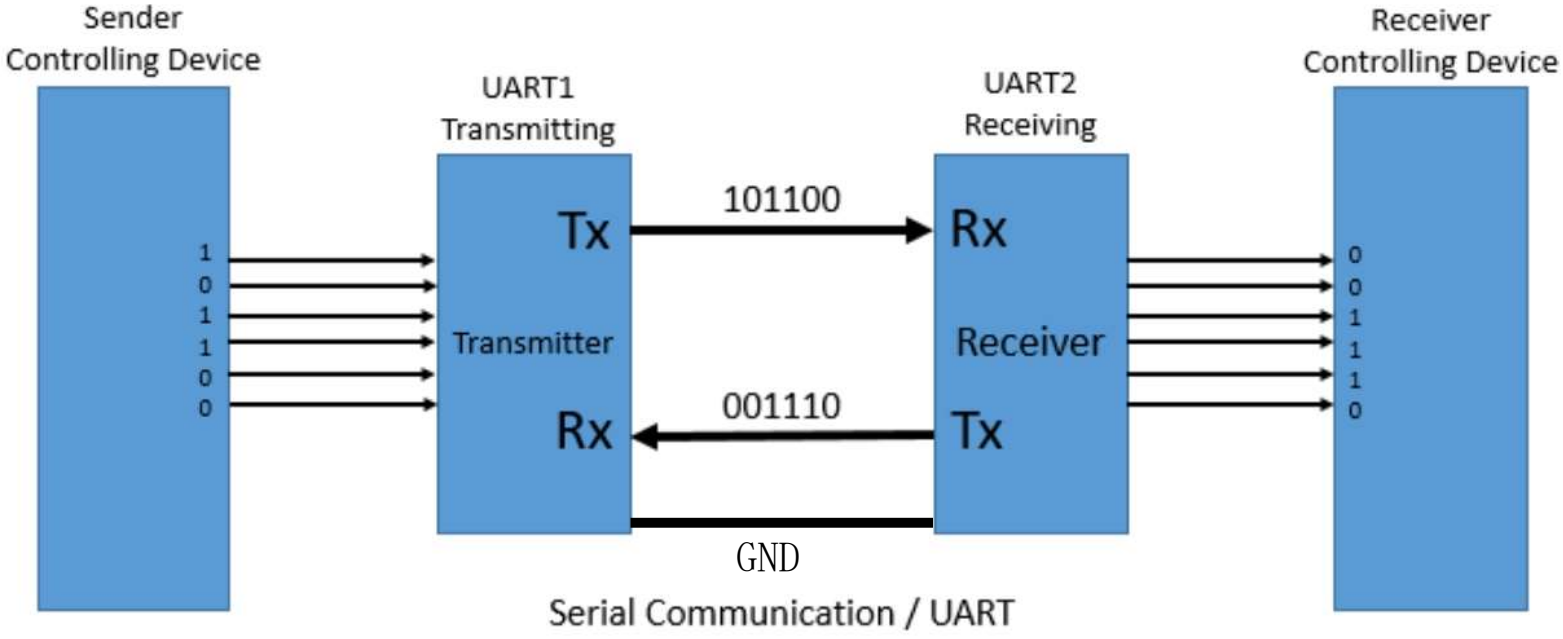
- USB(Universal Serial Bus)
- UART(Universal Asynchronous Receiver Transmitter）
- SPI(Serial Peripheral Interface)
- I2C(Inter Integrated Circuit)
- CAN(Controller Area Network, CAN)

常用串行通信协议对比

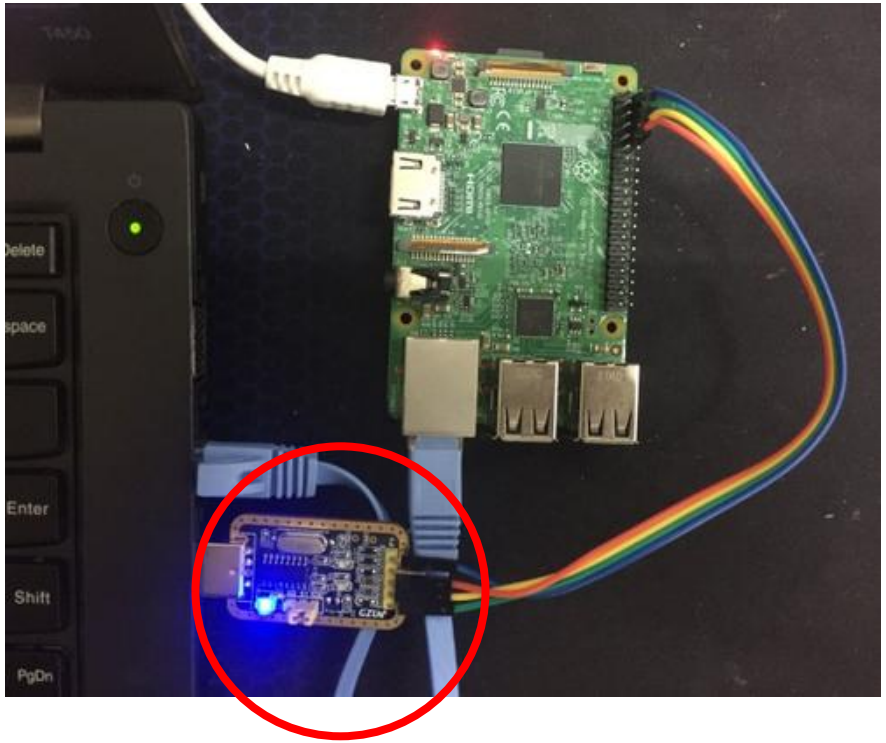
接口	复杂度	传输速度	传输距离	通信方式
I^2C	低 有地址	标准模式：100kbps 快速模式：400kbps 高速模式：3.4Mbps	板级总线	半双工 一对多 主从 同步
UART串口	很低	RS232：20Kbps RS485：10Mbps	RS232：20M RS485：1KM	全双工 一对一 无主从 异步
SPI	中 无地址	18MB/S	板级总线	全双工 一对多 主从 同步
CAN	高	1MB/S	10KM	半双工 多对多 无主从 异步
USB	高	低速模式：1.5Mbps 全速模式：12Mbps 高速模式：25~480Mbps	5M	USB2.0半双工 USB3.0全双工 一对一 主从 异步

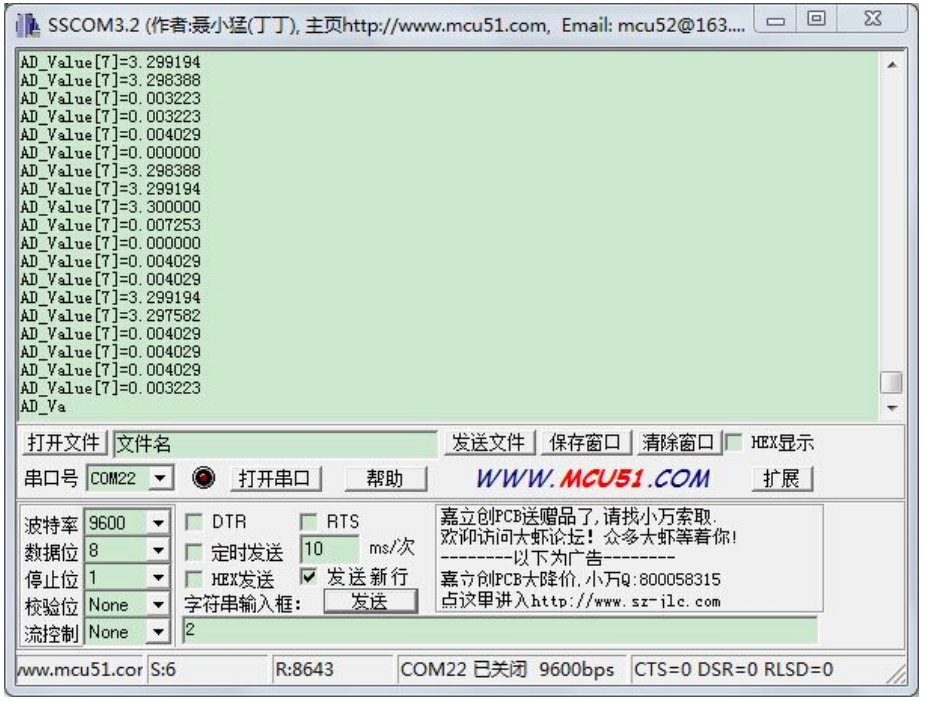
UART


UART
RX
TX
GND
UART
RX
TX
GND

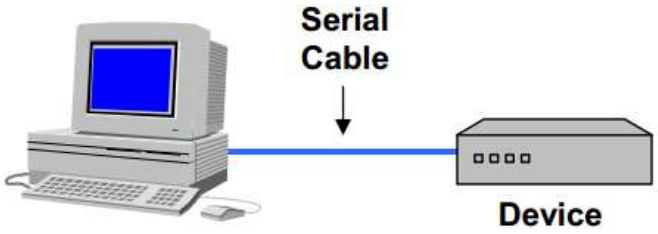

Sender
Controlling Device
1
0
1
1
0
0
UART1
Transmitting
Tx
Transmitter
Rx
101100
001110
GND
UART2
Receiving
Rx
Receiver
Tx
Receiver
Controlling Device
0
0
1
1
1
0
Serial Communication / UART

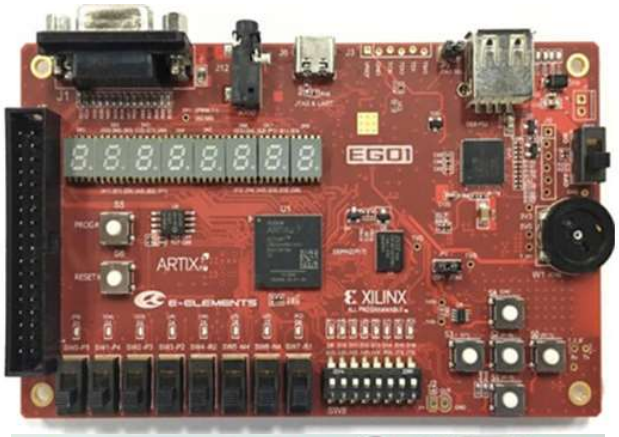
基于UART采集数据

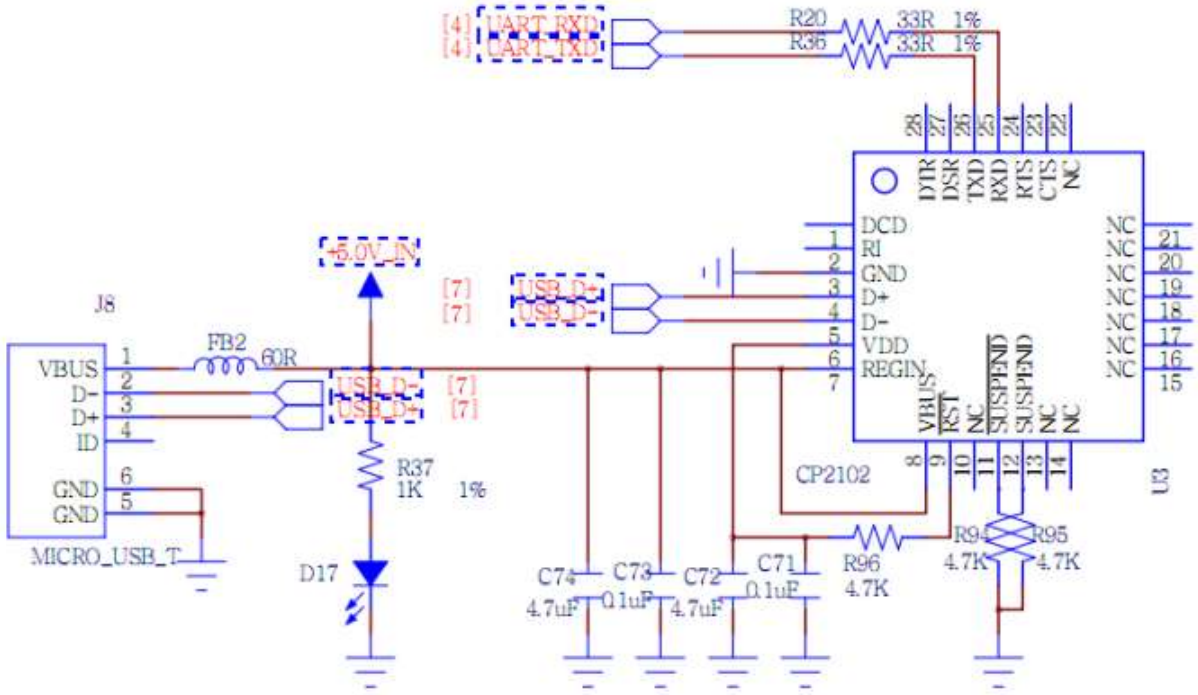



SSCOM3.2 (作者:聂小猛(丁丁), 主页http://www.mcu51.com, Email: mcu52@163....
AD_Value[7]=3.299194
AD_Value[7]=3.298388
AD_Value[7]=0.003223
AD_Value[7]=0.003223
AD_Value[7]=0.004029
AD_Value[7]=0.000000
AD_Value[7]=3.298388
AD_Value[7]=3.299194
AD_Value[7]=3.300000
AD_Value[7]=0.007253
AD_Value[7]=0.000000
AD_Value[7]=0.004029
AD_Value[7]=0.004029
AD_Value[7]=3.299194
AD_Value[7]=3.297582
AD_Value[7]=0.004029
AD_Value[7]=0.004029
AD_Value[7]=0.004029
AD_Value[7]=0.003223
AD_Va
打开文件
文件名
发送文件
保存窗口
清除窗口
HEX显示
串口号 COM22
打开串口
帮助
WWW.MCU51.COM
扩展
波特率 9600
数据位 8
停止位 1
校验位 None
流控制 None
DTR
RTS
定时发送
10
ms/次
HEX发送
发送新行
字符串输入框:
发送
嘉立创PCB送赠品了,请找小万索取.
欢迎访问大虾论坛! 众多大虾等着你!
--------以下为广告--------
嘉立创PCB大降价,小万Q:800058315
点这里进入http://www.sz-ilc.com
2
ww.mcu51.cor
S:6
R:8643
COM22 已关闭 9600bps
CTS=0 DSR=0 RLSD=0

UART



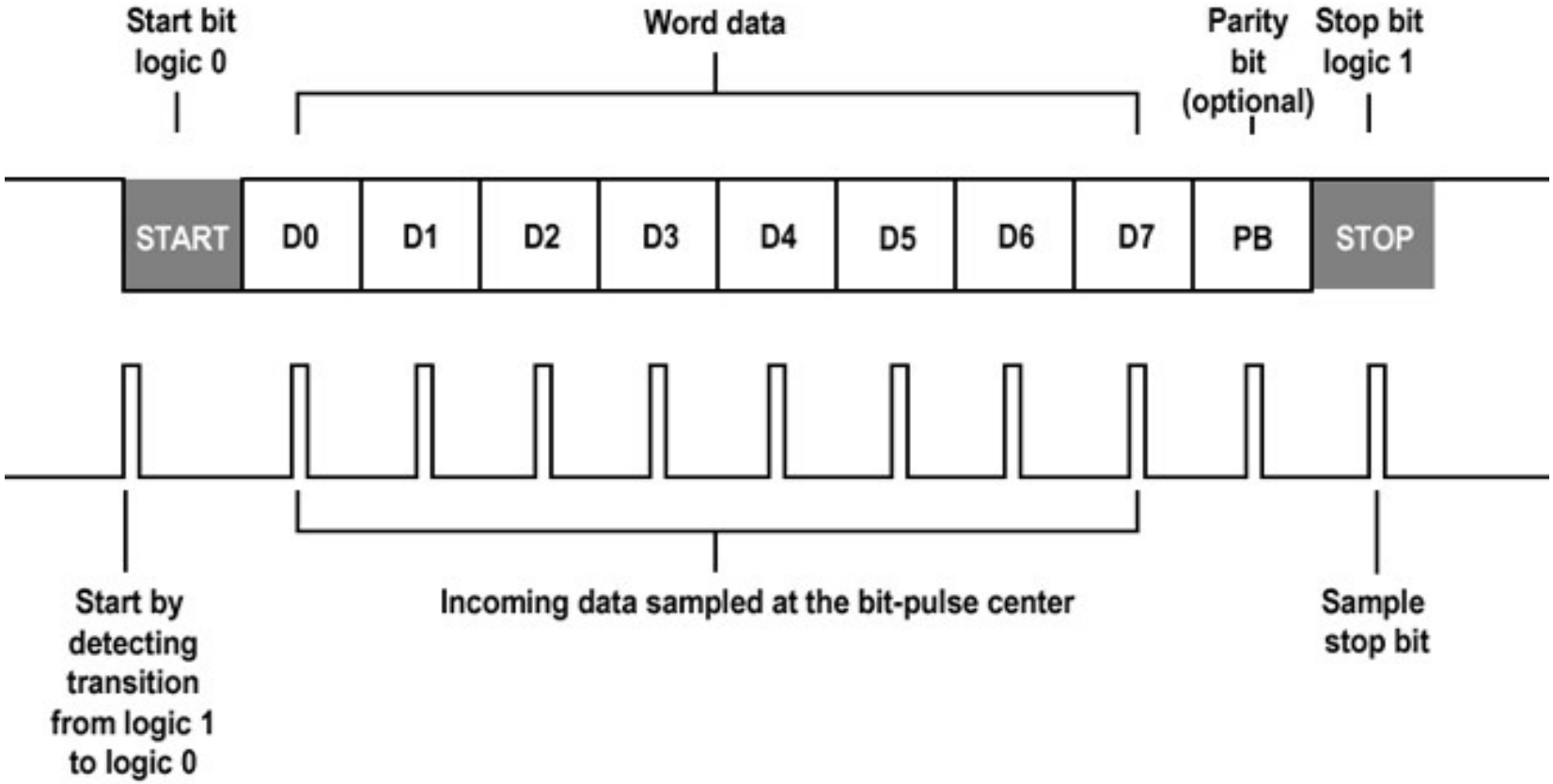




管脚约束如下:

CP2102 标号	原理图标号	FPGA IO PIN
25	UART_RX	T4 （FPGA 串口发送端）
26	UART_TX	N5 （FPGA 串口接收端）

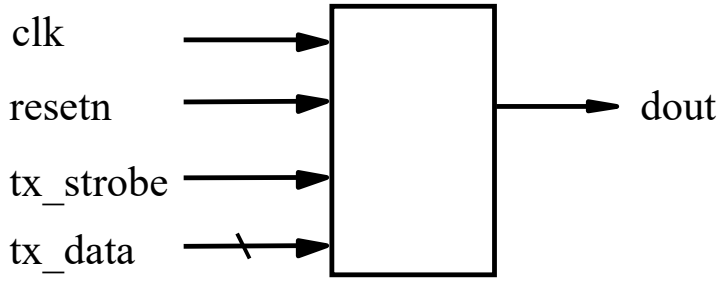
UART

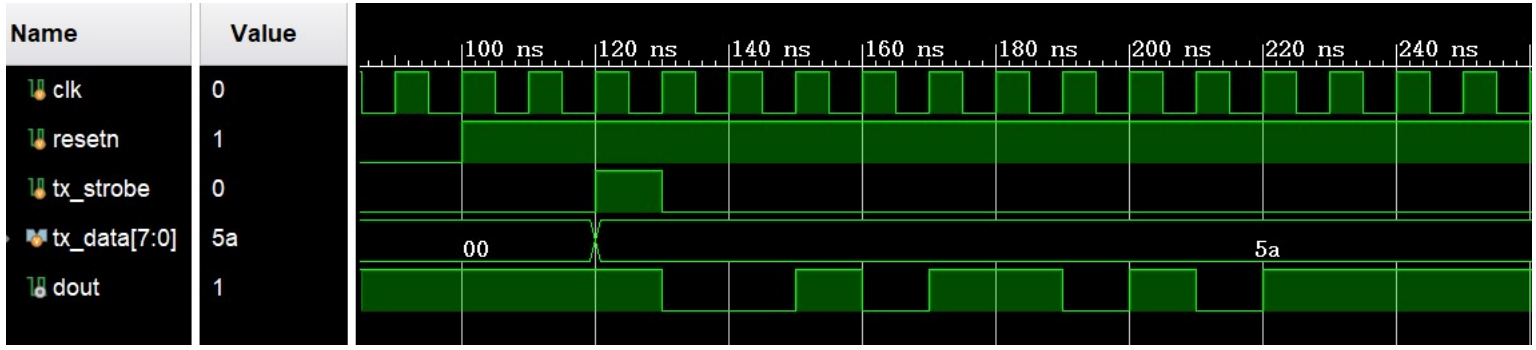

Start bit
logic 0
Word data
Parity
bit
(optional)
Stop bit
logic 1
START
D0
D1
D2
D3
D4
D5
D6
D7
PB
STOP
Start by
detecting
transition
from logic 1
to logic 0
Incoming data sampled at the bit-pulse center
Sample
stop bit

串口收发器设计

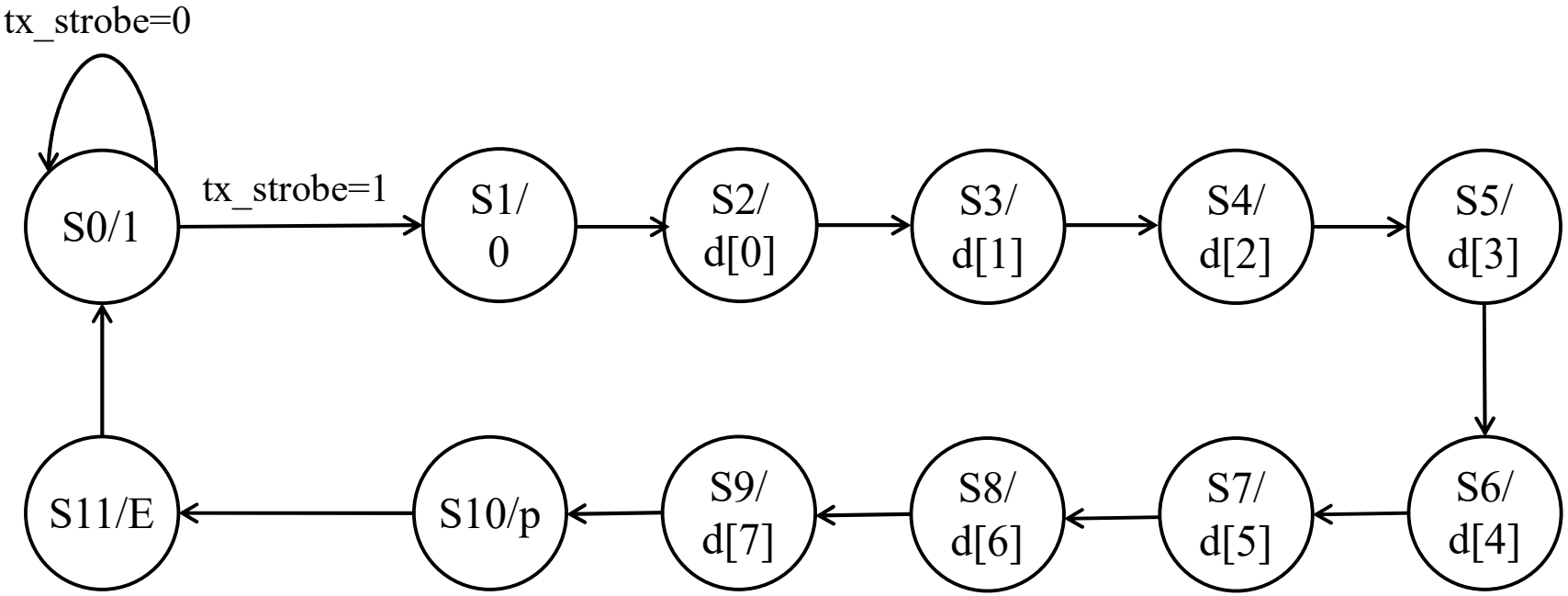
发送端：

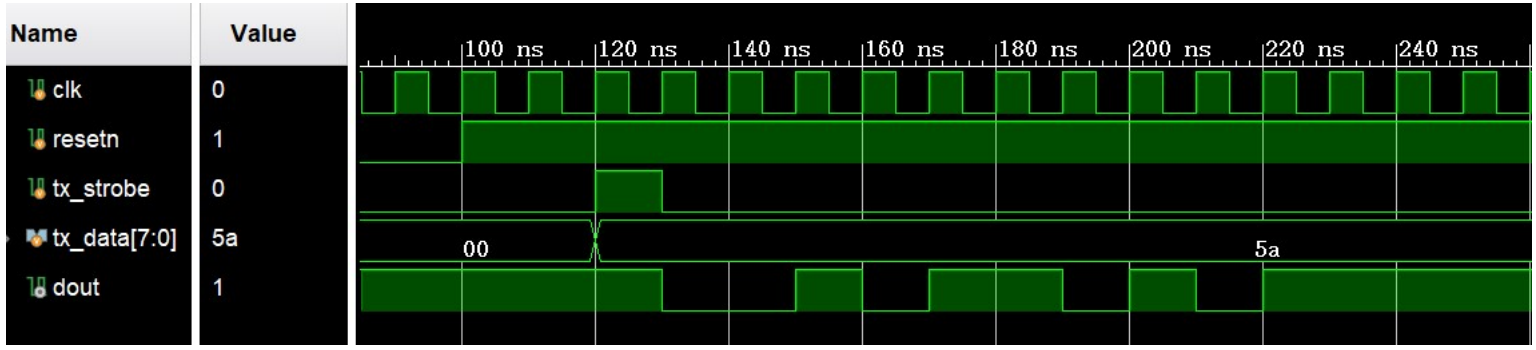
- 串口格式
  - 1位起始位，8位数据位，1位校验位（奇校验）
- 待发送数据tx_data为8位
- tx_data有效信号：tx_strobe
- 将数据按从低位到高位的顺序依次从dout端口发出





串口收发器设计


tx_strobe=0
S0/1
tx_strobe=1
S1/
0
S2/
d[0]
S3/
d[1]
S4/
d[2]
S5/
d[3]
S6/
d[4]
S7/
d[5]
S8/
d[6]
S9/
d[7]
S10/p
S11/E


Name
Value
clk
0
resetn
1
tx_strobe
0
tx_data[7:0]
5a
dout
1
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns
00
5a

串口收发器设计

- 并行数据锁存

```
always @(posedge clk or negedge resetn) begin
    if(!resetn)
        data_in <= 0;
    else if(strobe)
        data_in <= tx_data;
end
```

串口收发器设计

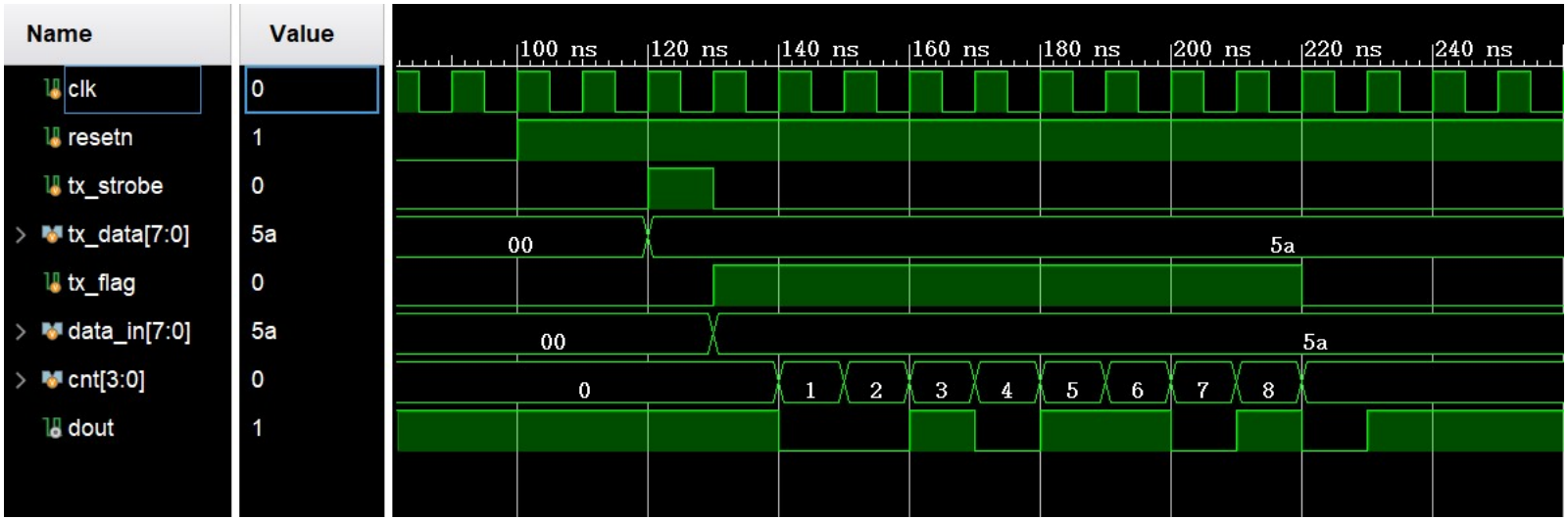
- 状态机

```
always @(posedge clk or negedge resetn) begin
    if(!resetn)
        state_c <= 4'b0000;
    else
        state_c <= state_n;
end
```

```
always @(*) begin
    case (state_c)
        4'b0000:    dout=1'b1;
        4'b0001:    dout=1'b0;
        4'b0010:    dout=data_in[0];
        4'b0011:    dout=data_in[1];
        4'b0100:    dout=data_in[2];
        4'b0101:    dout=data_in[3];
        4'b0110:    dout=data_in[4];
        4'b0111:    dout=data_in[5];
        4'b1000:    dout=data_in[6];
        4'b1001:    dout=data_in[7];
        4'b1010:    dout=~(^data_in);
        4'b1011:    dout=1'b1;
        default:    dout=1'b1;
    endcase
end
```

```
always @(*) begin
    case (state_c)
        4'b0000:    if(tx_strobe == 1'b1)
                        state_n = 4'b0001;
                    else state_n = 4'b0000;
        4'b0001:    state_n = 4'b0010;
        4'b0010:    state_n = 4'b0011;
        4'b0011:    state_n = 4'b0100;
        4'b0100:    state_n = 4'b0101;
        4'b0101:    state_n = 4'b0110;
        4'b0110:    state_n = 4'b0111;
        4'b0111:    state_n = 4'b1000;
        4'b1000:    state_n = 4'b1001;
        4'b1001:    state_n = 4'b1010;
        4'b1010:    state_n = 4'b1011;
        default:    state_n = 4'b0000;
    endcase
end
```

串口收发器设计 非状态机方式



串口收发器设计 非状态机方式

```
always @(posedge clk or negedge resetn) begin
    if(!resetn)
        data_in <= 0;
    else if(tx_strobe)
        data_in <= tx_data;
end

always @(posedge clk or negedge resetn) begin
    if (!resetn)
        tx_flag <= 1'b0;
    else if (tx_strobe)
        tx_flag <= 1'b1;
    else if (cnt == 8)
        tx_flag <= 1'b0;
end
```

串口收发器设计 非状态机方式

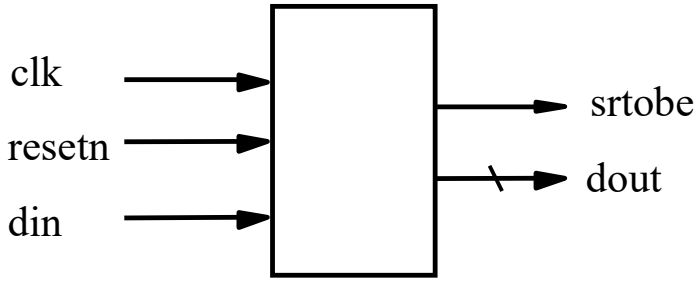
```
always @(posedge clk or negedge resetn) begin
    if(!resetn)
        cnt <= 4'b0000;
    else if (cnt == 8)
        cnt <= 'd0;
    else if (tx_flag)
        cnt <= cnt + 1'b1;
end
```

```
always @(posedge clk or negedge resetn) begin
    if(!resetn) dout <= 1'b1;
    else if (tx_flag)
        case (cnt)
            0:  dout<=1'b0;
            1:  dout<=data_in[0];
            2:  dout<=data_in[1];
            3:  dout<=data_in[2];
            4:  dout<=data_in[3];
            5:  dout<=data_in[4];
            6:  dout<=data_in[5];
            7:  dout<=data_in[6];
            8:  dout<=data_in[7];
            default:    dout<=1'b1;
        endcase
    else
        dout <= 1'b1;
end
```

串口收发器设计

接收端：

- 串行数据格式
  - 1位起始位，4位数据位，1位校验位
- 解码电路检测到校验位正确后，输出数据及一个时钟周期的数据有效信号
- 如校验错误，则不输出数据及数据有效



串口收发器设计

- 状态机

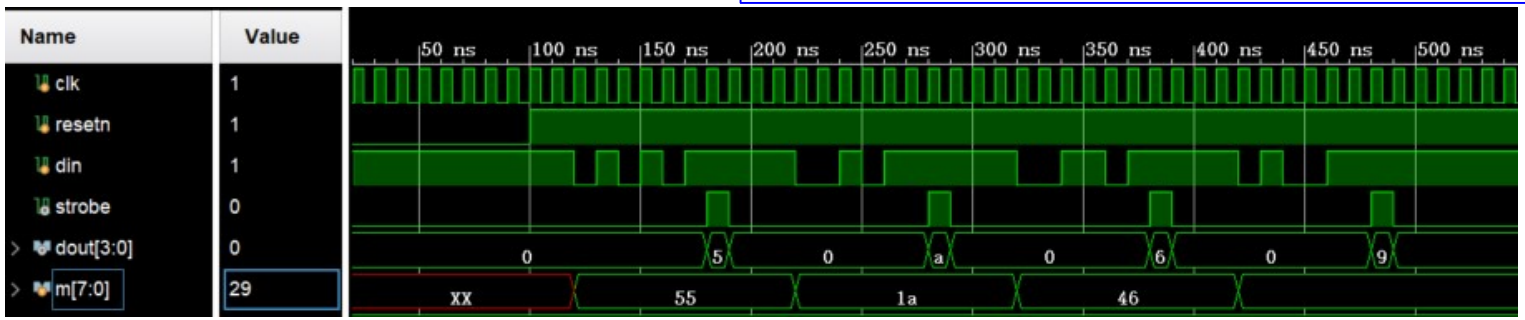
```
always @(posedge clk or negedge resetn) begin
    if(!resetn)      state <= 3'b000;
        else
            case (state)
            3'b000: if (!din) state <= 3'b001;
            3'b001: state <= 3'b010;
            3'b010: state <= 3'b011;
            3'b011: state <= 3'b100;
            3'b100: state <= 3'b101;
            default: state <= 3'b000;
            endcase
end
```

串口收发器设计

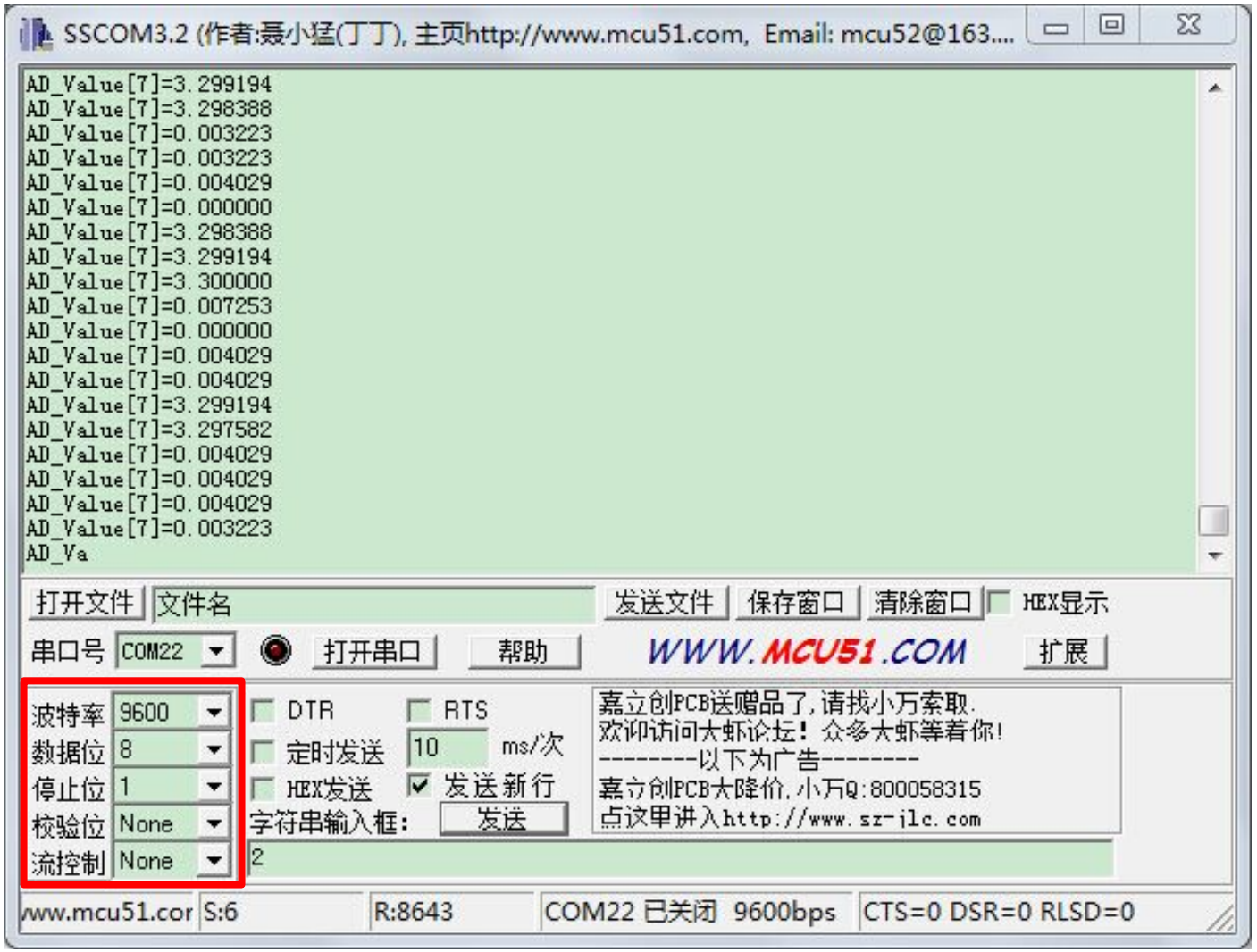
- 数据通路

```
always @(posedge clk or negedge resetn) begin
        if(!resetn) dbuf<=4'b0000;
        else
            case(state)
                1: dbuf[0]<=din;
                2: dbuf[1]<=din;
                3: dbuf[2]<=din;
                4: dbuf[3]<=din;
            endcase
end
```

```
always @(posedge clk or negedge resetn) begin
    if(!resetn) begin
        strobe <= 1'b0;
        dout <= 'd0;
    end
    else
        if (state==3'b101 && ^dbuf == ~din)begin
            strobe <= 1'b1;
            dout <= dbuf;
        end
        else begin
            strobe <= 1'b0;
            dout <= 4'b0;
        end
end
```



UART


SSCOM3.2 (作者:聂小猛(丁丁), 主页http://www.mcu51.com, Email: mcu52@163....
AD_Value[7]=3.299194
AD_Value[7]=3.298388
AD_Value[7]=0.003223
AD_Value[7]=0.003223
AD_Value[7]=0.004029
AD_Value[7]=0.000000
AD_Value[7]=3.298388
AD_Value[7]=3.299194
AD_Value[7]=3.300000
AD_Value[7]=0.007253
AD_Value[7]=0.000000
AD_Value[7]=0.004029
AD_Value[7]=0.004029
AD_Value[7]=3.299194
AD_Value[7]=3.297582
AD_Value[7]=0.004029
AD_Value[7]=0.004029
AD_Value[7]=0.004029
AD_Value[7]=0.003223
AD_Va
打开文件
文件名
发送文件
保存窗口
清除窗口
HEX显示
串口号
COM22
打开串口
帮助
WWW.MCU51.COM
扩展
波特率
9600
数据位
8
停止位
1
校验位
None
流控制
None
DTR
RTS
定时发送
10
ms/次
HEX发送
发送新行
字符串输入框:
发送
嘉立创PCB送赠品了,请找小万索取.
欢迎访问大虾论坛! 众多大虾等着你!
--------以下为广告--------
嘉立创PCB大降价,小万Q:800058315
点这里进入http://www.sz-jlc.com
2
www.mcu51.cor
S:6
R:8643
COM22 已关闭 9600bps
CTS=0 DSR=0 RLSD=0