Digital System Design

< Block 1 : VHDL Basic >.

1. Introduction.

Modelling → Entry → Test → Synthesis → Implementation → Verification.

明确 input/output　用VHDL表达　验证正确性　变化为实际电路　actual device　performance.

2. Facts, Entity, Architectures, Statements.

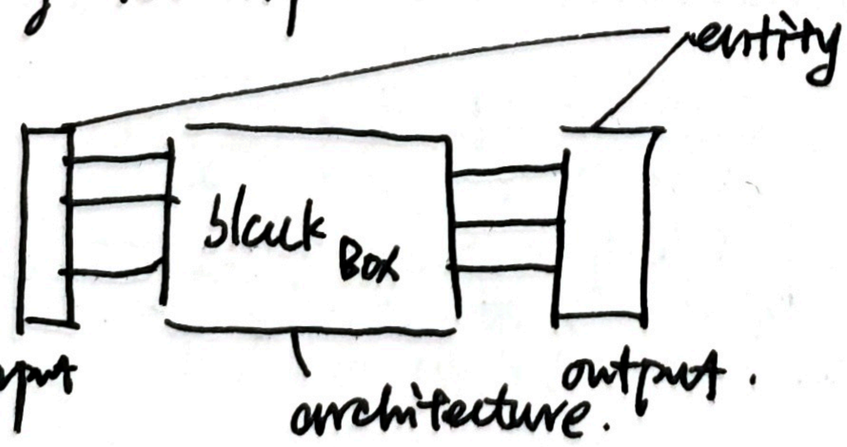
VHDL　Concurrent language, execute in parrallel, describle hardware.

Code Basic

```
library IEEE;
use IEEE.std_logic_1164.all;
entity e_name is
  port(  A, B, C : in  std_logic;
         F       : out std_logic);
end e_name;
architecture arch of e_name (is)
begin 「可插入中间信号 signal D: std_logic;
    F <= A and B and C;
    F <= '1' when (A = '0' and B = '0') else
         '0' when (C = '0') else
         '0' ; -- 第一个满足的条件，那个赋值，下面就不会执行了.
end arch;
```

3. Circuit Modelling Techniques.

Black Box



Code vector

```
a_data : in  std_logic_vector (4 downto 0);
b_data : out std_logic_vector (1 downto 0);
```

设计风格
1) dataflow modelling，一般描述并发行为
2) behavioral modelling，一般描述顺序行为.
3) structural modelling

code
Structural modelling

```
component big_xnor is
  port ( A, B : in std_logic;
         F    : out std_logic);
end component; --位于 architecture 的中间信号声明区域.
-- Method1 of map port, 位置映射 (Implicit mapping)
label: big_xnor port map (x, y, z);
-- Method2 of map port, 名称映射 (Direct mapping).
label: big_xnor port map (A => x, B => y, F => z);
-- 上述代码, x,y,z 是 entity 中的接口.
```

4. Sequential Statements and Testbenches.

The heart of the behavioral style is the process statement

code
Sequential Statement

```
[label]: process (A,B,C...) -- 敏感列表.
option
   variable temp : Integer range 0 to 10; --可声明 variable.
   Begin
     if (Condition) then <statements>;
     elsif (Condition) then <statements>;
     else <statements>;
     end if;
   Case (INPUT) is
       when "00" => x <= a; y <= b;
       when "01" => <statements>;
       when OTHERS => <statements>;
   end case;
end process;
```

Note 1) process 块整体是一个并发语句,内部顺序执行,但是注意 Signal 在 Process 内只有 process 结束时才会变化,那并不是实时结果的而是 take the last one, 想做到实时变化,得用 variable

2) case 与 when 类似,只有一个会被触发,但是 case 可以在一个<statements>中对多个 signal 赋值.

Test Benches A simulation tools can verify the correctness of model in terms of functionality and timing. The test is on Unit Under Test (UUT).

code
Test Bench

```
entity bif_xnor_tb is                                  } 无entity体.(port)
end big_xnor_tb; -- 无 input/output
architecture behaviour of bif_xnor_tb is
    component bif_xnor is
        port ( ... );                                  } 声明
    end component;
  I Signal 声明 ex. Signal A, B, F = std_logic]
begin
  label = bif_xnor port map (A=>A, B=>B, F=>F);        } 实例化
  process                                              } 设置测试
  begin                                                  语句.
    A<='0'; B<='1'; wait for 10 ns;
      :
    wait;
  end process;
end behaviour;
```

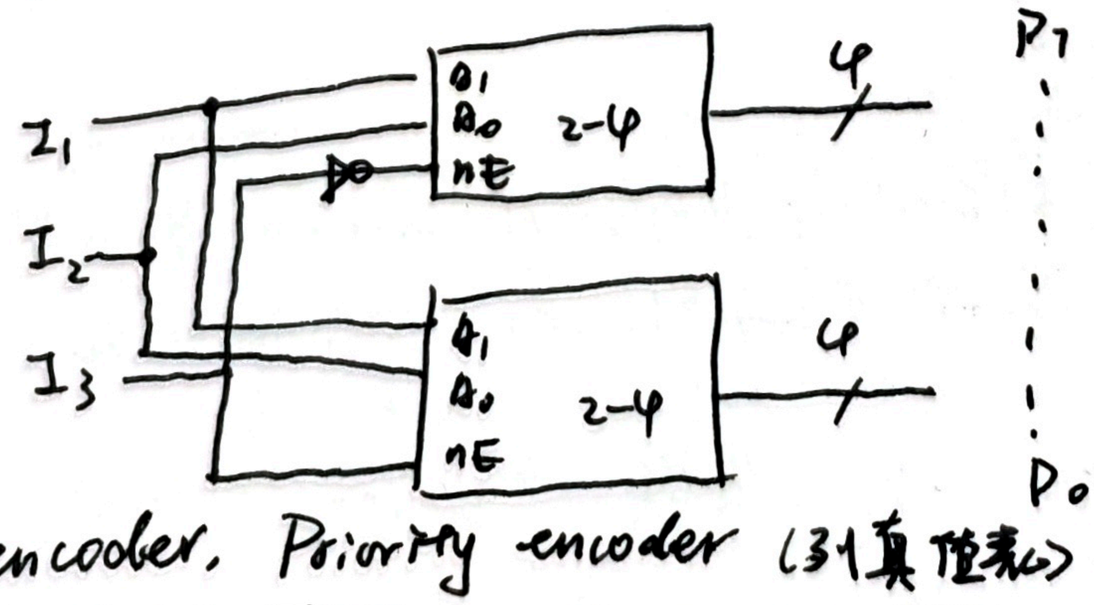
code
clock

```
signal CLK = std_logic;
constant period = TIME := 10 us;

process
  begin
    CLK <='1'; wait for (period/2);
    CLK <='0'; wait for (period/2);
end process;
```

5. Hierarchical Design & Combintional Blocks.

Decoder Expansion

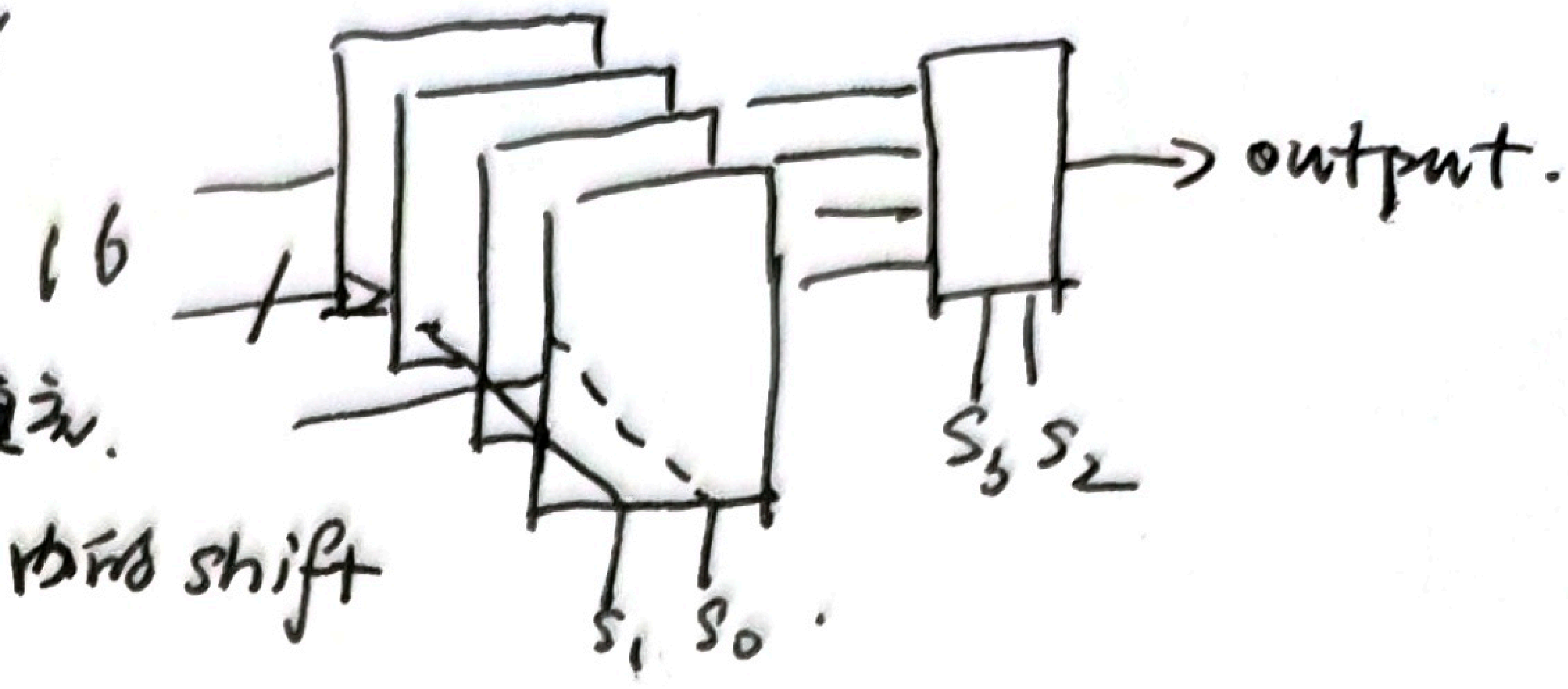


Others decoder, encoder, Priority encoder (31真值表).

code
MUX

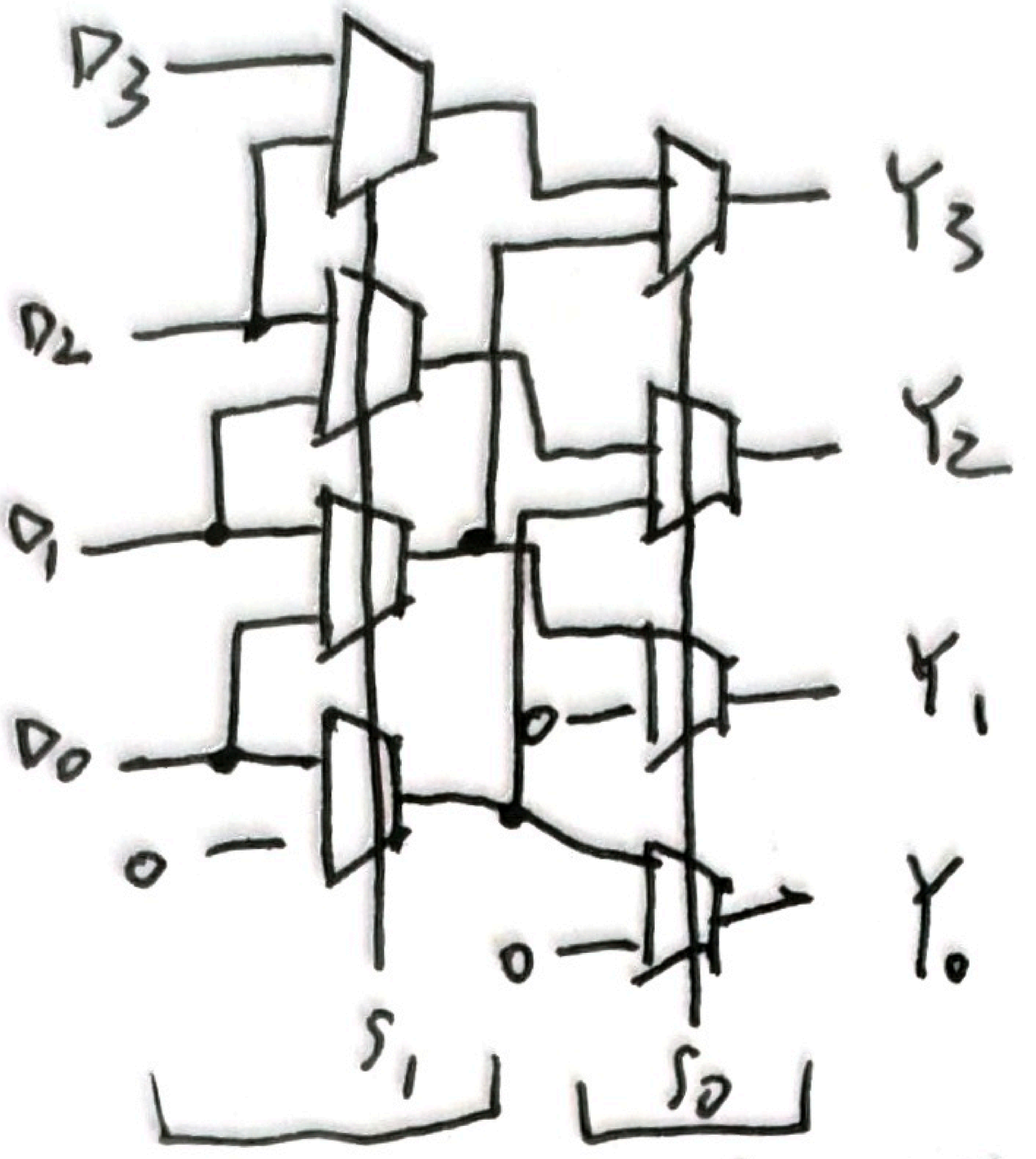
```
port ( D3, D2, D1, D0 : in std_logic_vector (7 downto 0);
       ... );
...
  with S select                                        } select-when并发语句.
      Y <= D3 when "11";
           D2 when "10";
           D0 when others;
```

Note 可以用5个 4 to 1 MUX 组成1个16-1 MUX



shifting 定义 left/right shift，以及 incoming bits 填充.

例如使用8个 2 to 1 MUX 组成一个 4bit 内的 shift



输出D0或D1　输出X0或X1.

左移并补0.

基本的 decoder 和 MUX 可以用来组成更大的电路.

注意

D1 <= D0(2 down to 0) & '0';

X1 <= X0(1 down to 0) & "00";

& 表示并置符.

<Block 2: ALU, Sequential Blocks, Buffers>.

1. 补码.

对于正数，有符号数(补码)是本身，负数是绝对值按位取反再加1.

补码的首位是负数加权，对于两个signed number，相同首位相加，结果首位不同于两个加数，则发生 overflow.

2. Adders, Subtractors, ALUs.

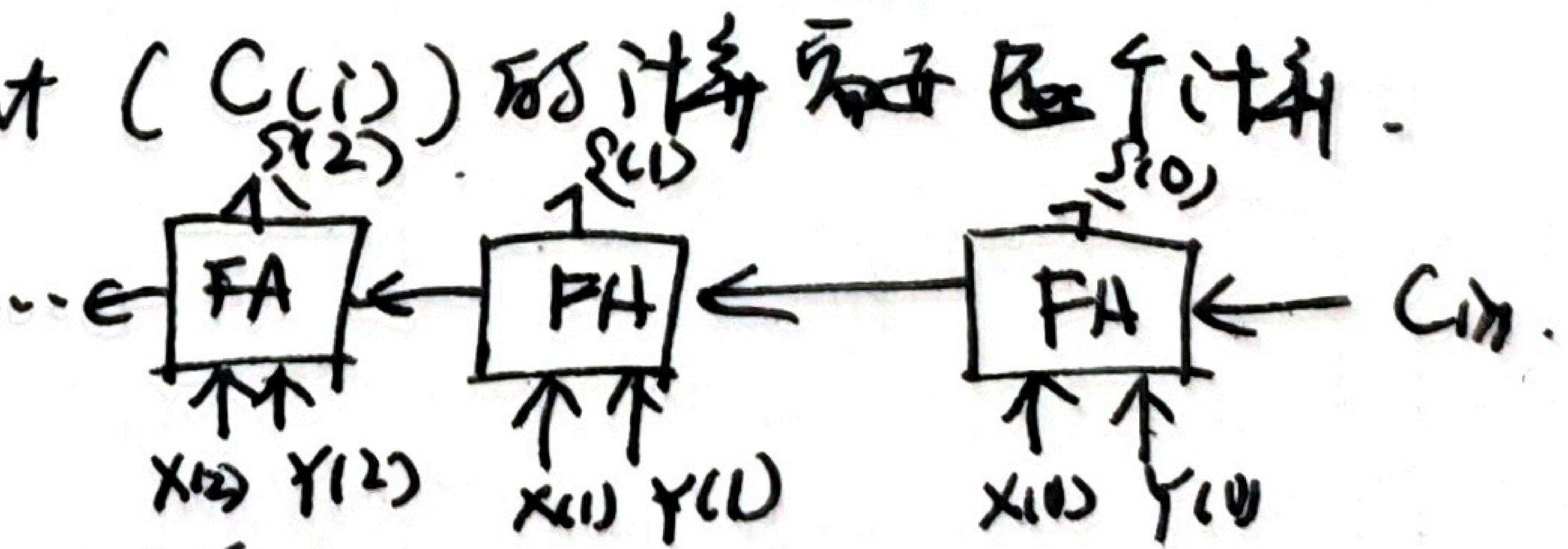
FA $S = X \oplus Y \oplus C_{in}$, S <= X xor Y xor C.

$C_{out} = X \cdot Y + X \cdot C_{in} + Y \cdot C_{in}$, C <= (X and Y) or (X and Cin) or (Y and Cin).

Note 描述 Ripple Adder 的两种方法:

① 描述中间变量 Signal C: std_logic_vector(n downto 0). 然后对 X, Y, C 可直接 X xor Y xor C，但是每个 Cout (C(i)) 的计算需要逐个计算.

② 先描述每个 FA，再级连



③ 由 Adder 描述 Subtractors 十分简单，A - B = A + (-B) = A + (B的补码).

对Y按位取反，把 Cin 设为1即可.

Code generate

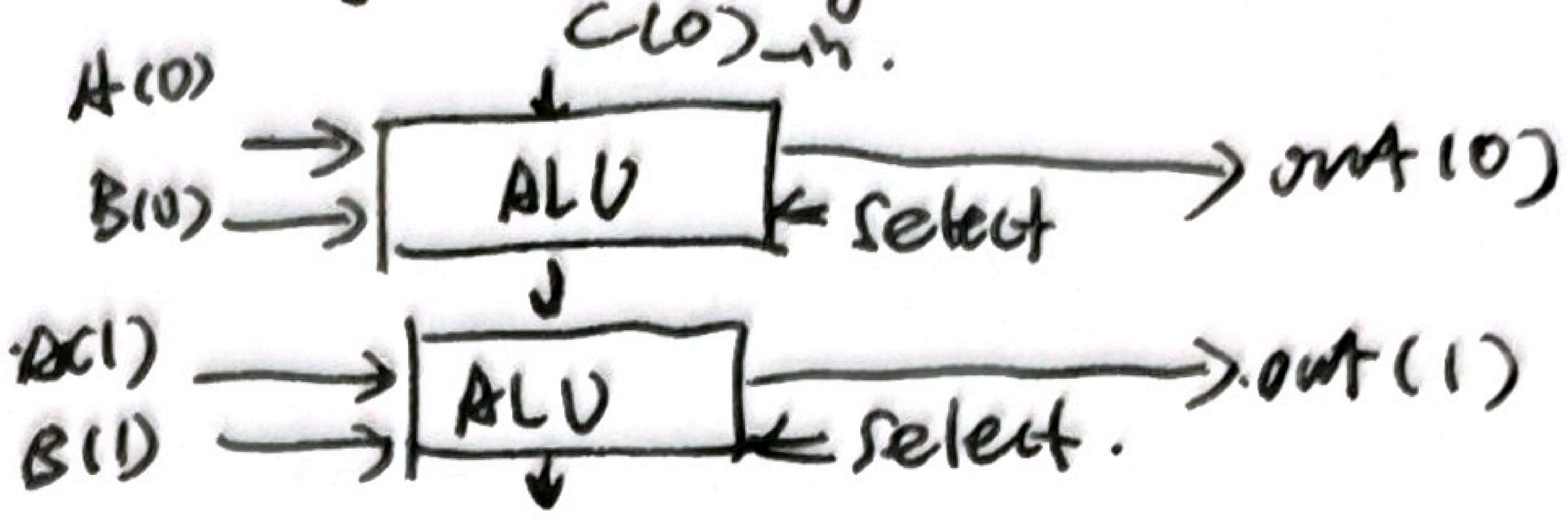
```
generate_label:
  for i in 1 to 3 generate
      label: FA port map (X => X(i), Y => Y(i), Cin => C(i),
                          S => S(i), Cout => C(i+1));
  end generate generate_label;
```

注意需要处理边界条件 (Input/Output).

ALU

1) 有时相当于FA并行也不熟 另直接使用 IEEE.std_logic_unsigned.all 库 可以用 Sum <= A+B，SUM <= A−B 来实现操作.（+，− 要求位数相同）

2) 同样，实现多 bit ALU，可以直接并行，也可以先设计 Bit-Slicing the ALU，再实例化拼接.

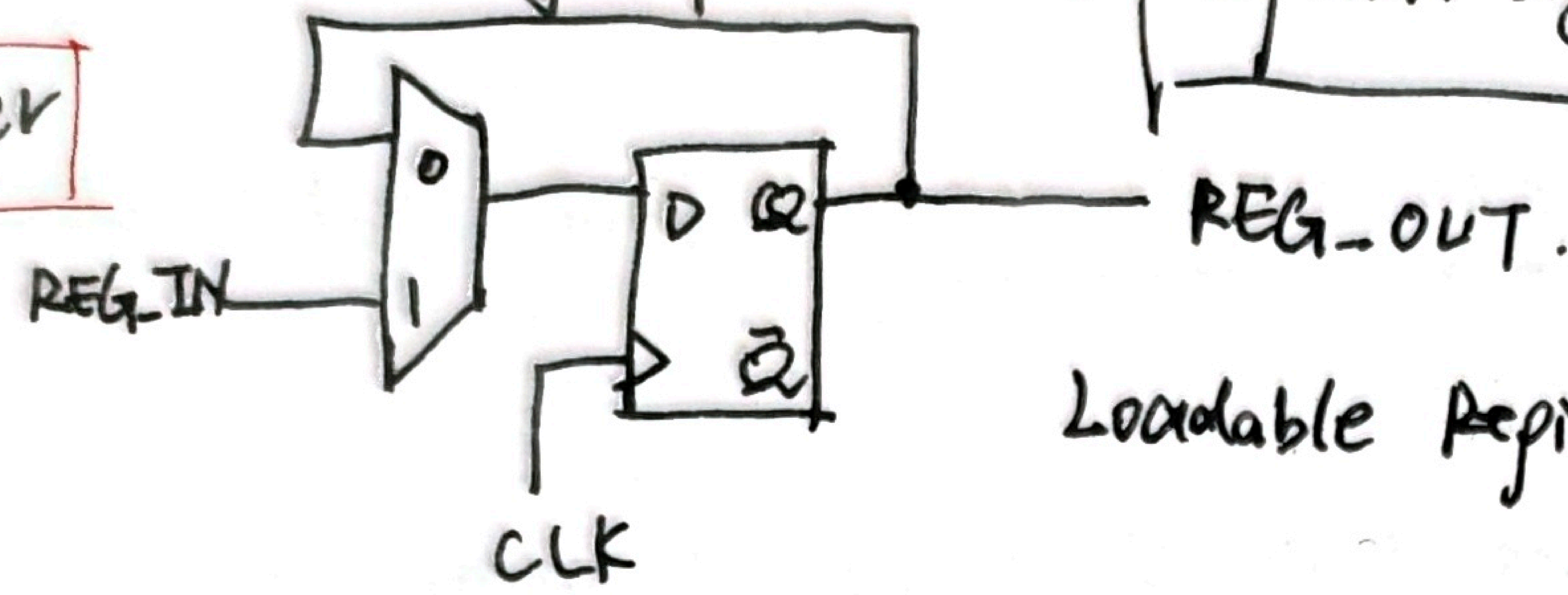


3. Flip-Flops, Shift Registers and Counters

a. **Dff** if (CLK'event and CLK = '1') then Q <= D; end if.

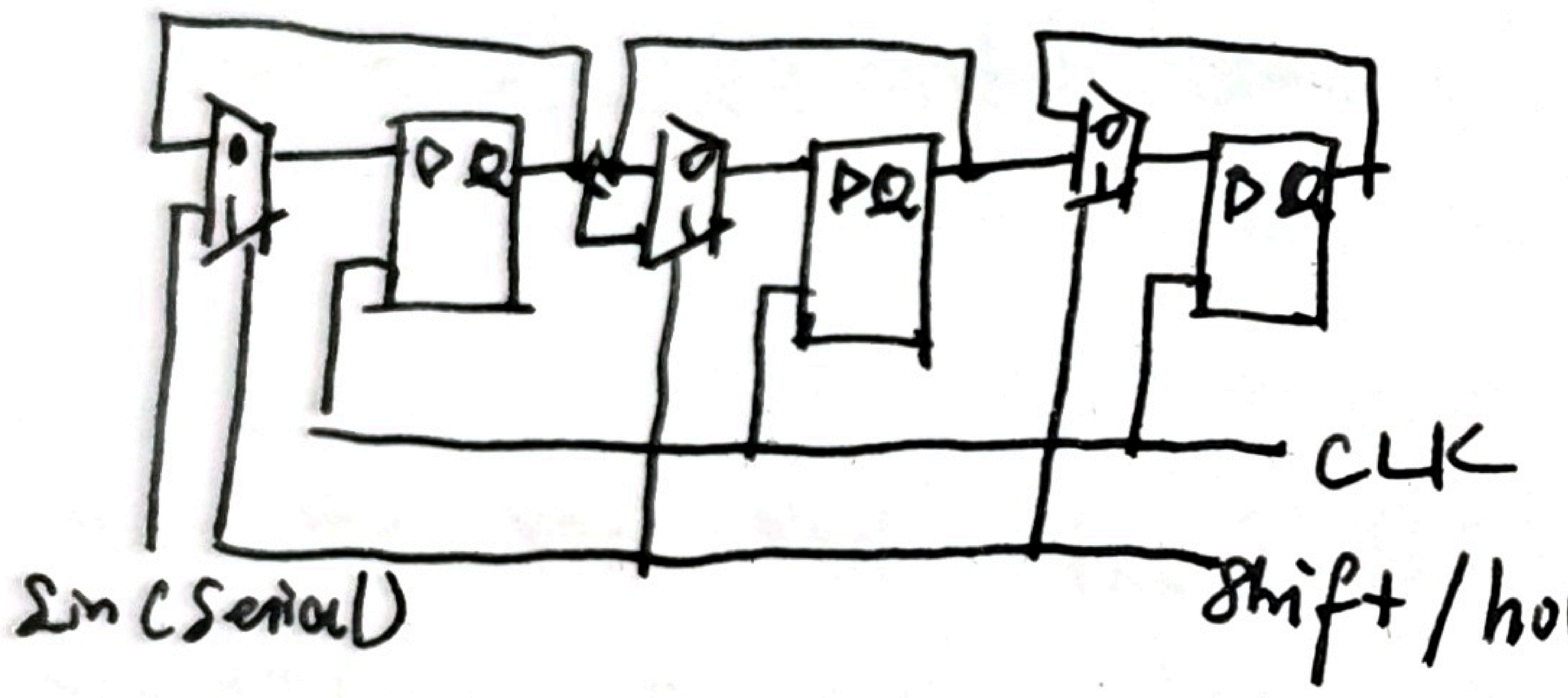
Note 当 process 中未考虑到真值表的所有可能组合时，会生成不必要的 latch.
ensure that you provide a default signal assignment or a catch-all condition

b. **Register**



Loadable Register => D Flip-flop with load enable.

Other1 shift Register with Output Hold.
Serial input.



注意使用

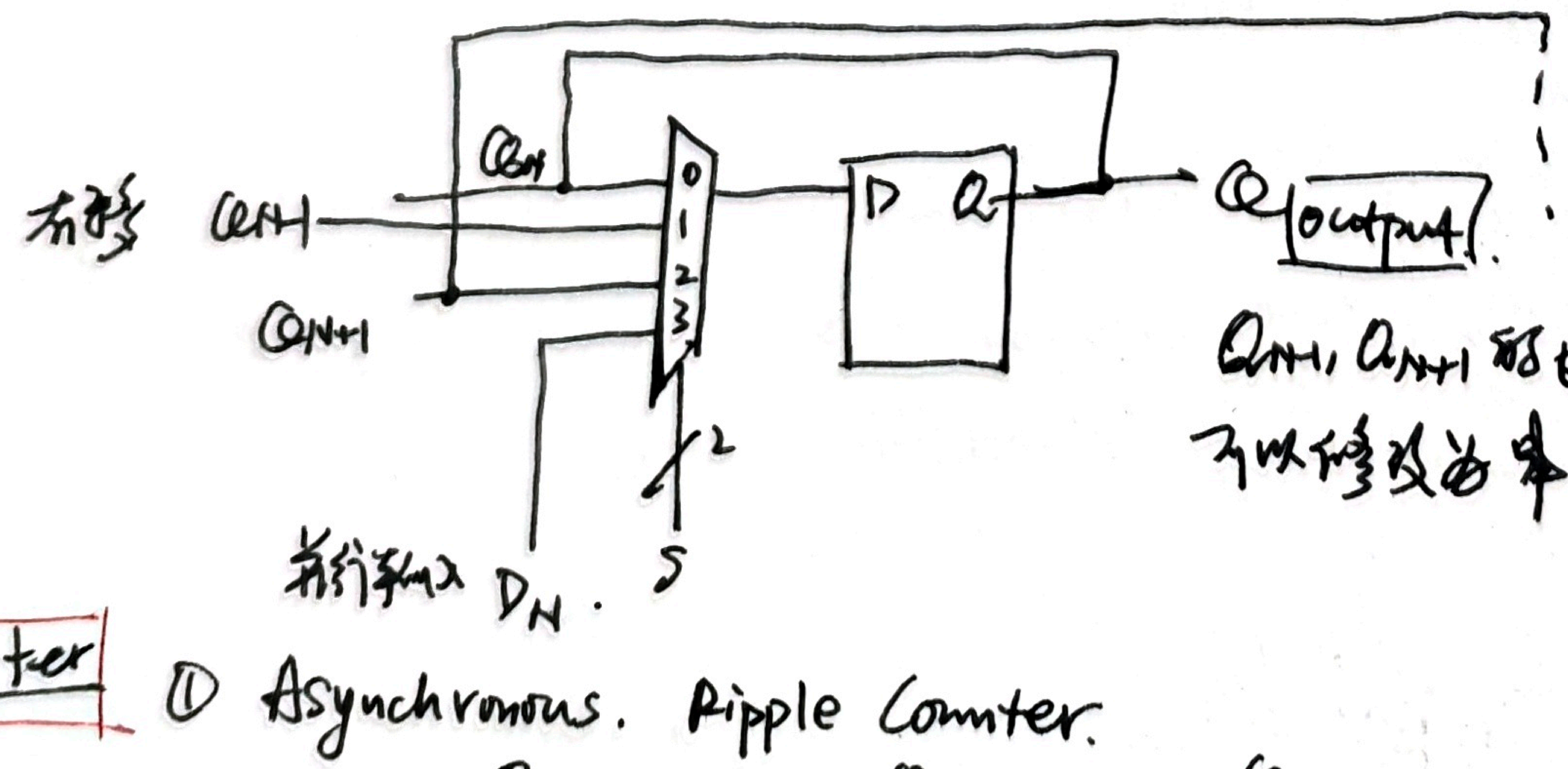
S <= Qint(2 downto 0) & S_IN

串行移位，并在上升沿将移位后的 S => Qout.

所有D的下一个状态不再是Q，而不是Q移位.

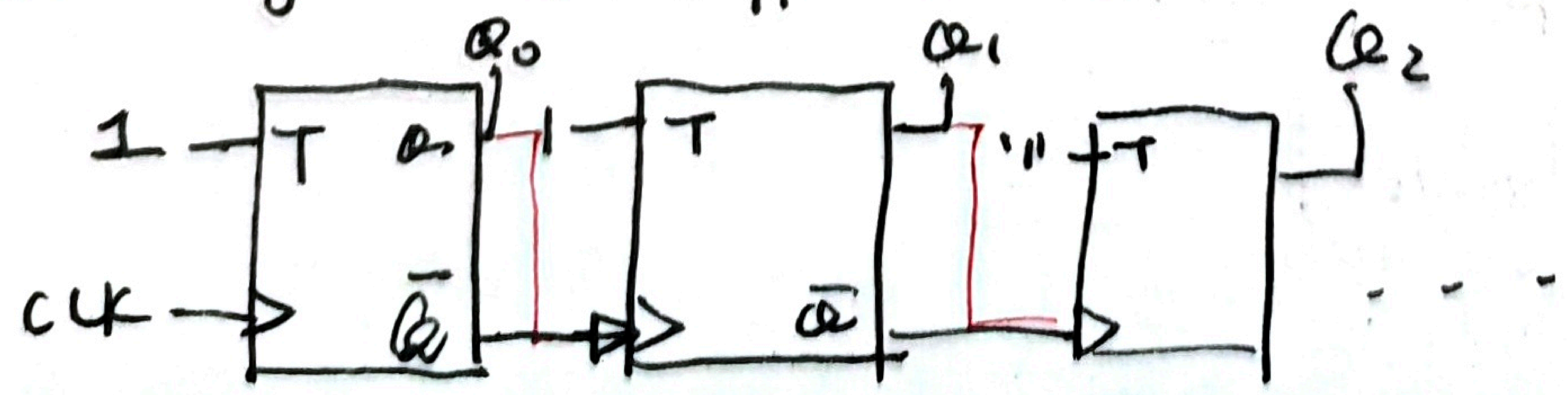
Other2
parallel input. 将 MUX的 Input 从单个 serial 变为所有的 parallel 即可.

Universal shift Register. 可以做出一个统一的 Register，有 hold，right / left shift 和 load 功能.



Q_{N-1}，Q_{N+1} 的也可以被重新定义，可以修改为串行输入.

c. **Counter** ① Asynchronous. Ripple Counter.

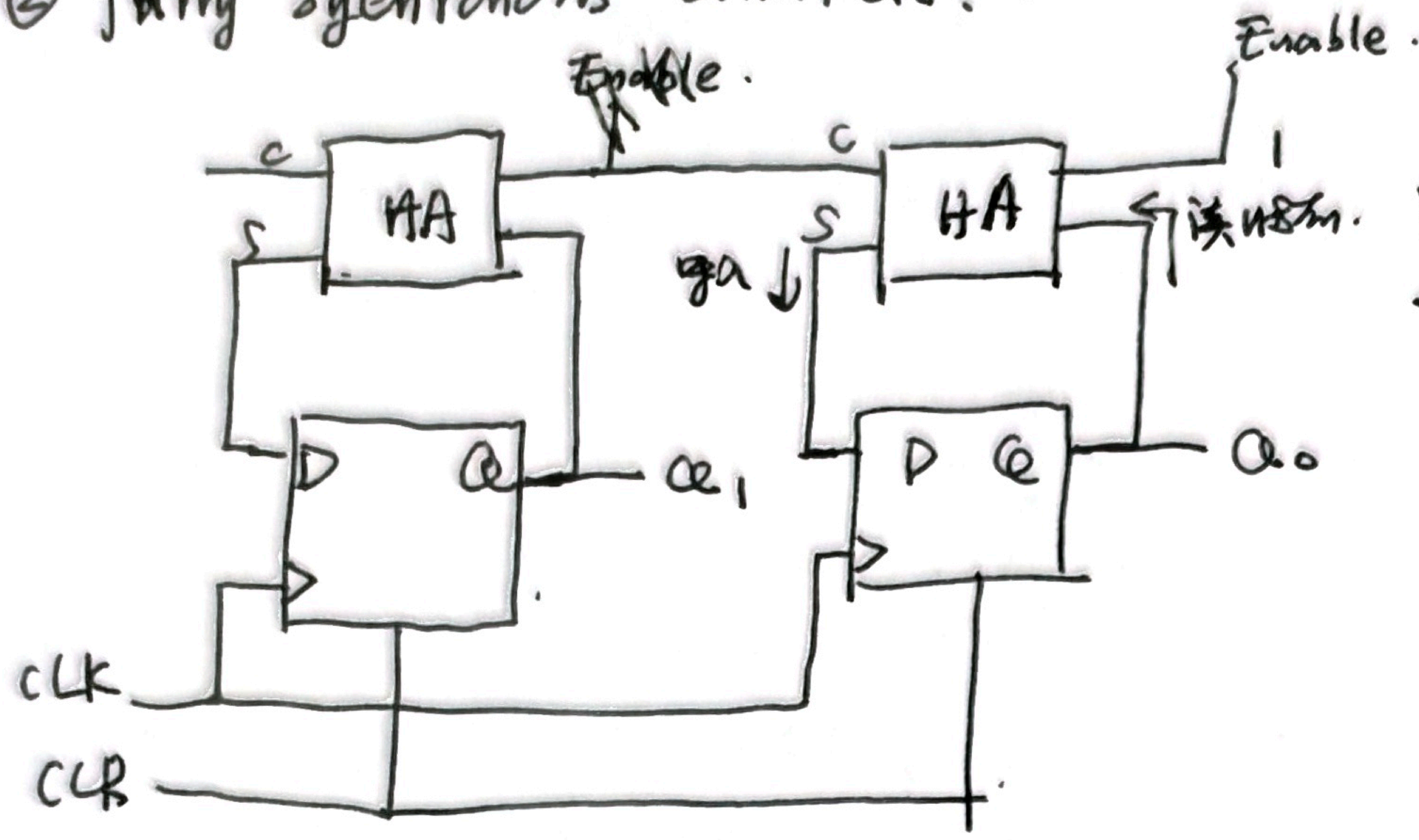


下面的T 触发器以前面T触发器 Q̄的上升沿（Q的下降沿）作为触发.

Q̄ 当下一个 clock，up 正计时. Q 当下一个 clock，down，倒计时.

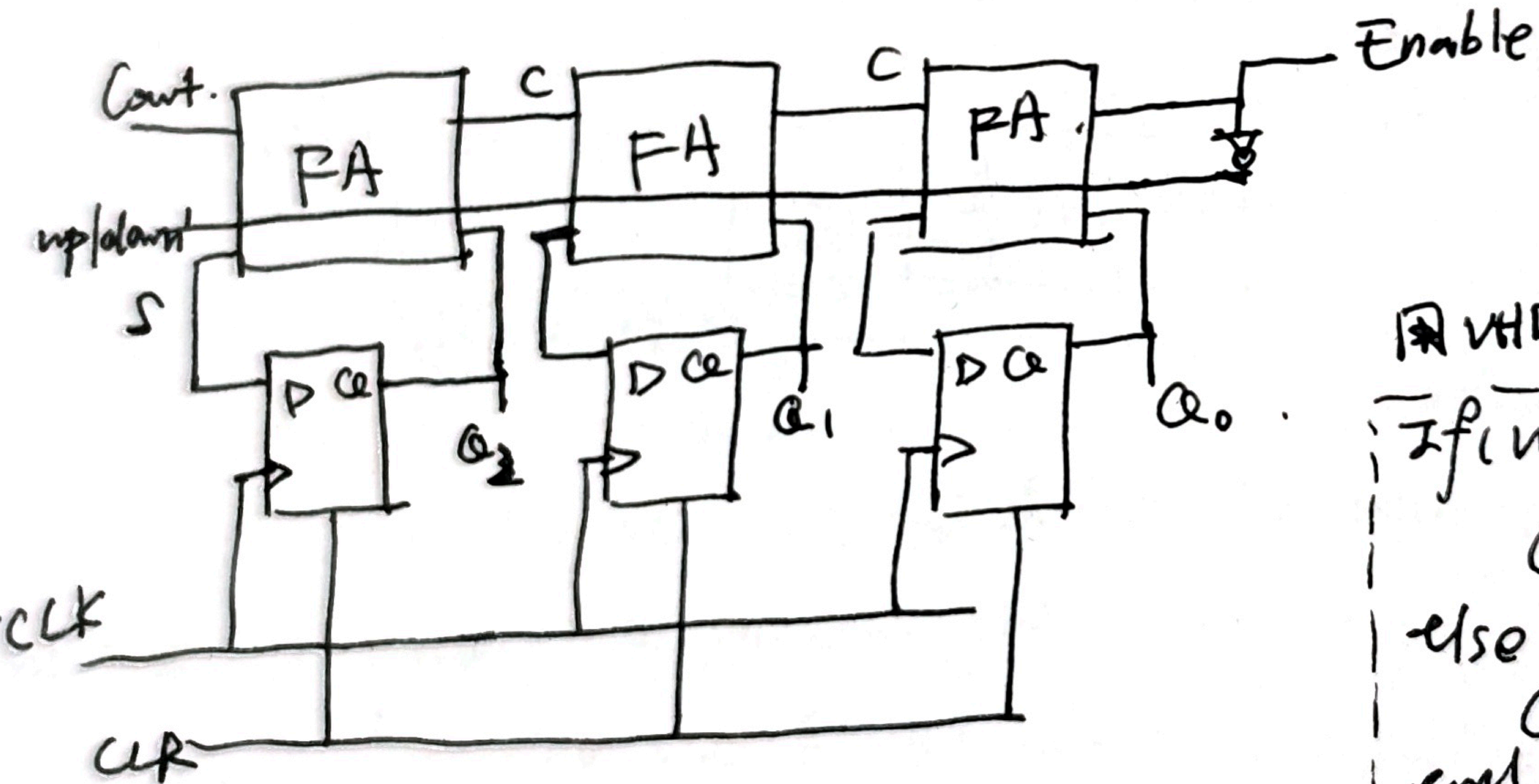
但是存在 propagation delay, change are not immediate, we should wait the output become stable to read the correct results.

② fully sychronous counters.（更精确，但需要更多硬件资源）.



当Enable为1时，CLK每上升沿，最右侧S变化，S变化之后下一个HA的input也变化，如此形成counter.

Note 计数器其实是在做加法，且每次加1的加法，它是一种组合与时序合成的作法. 可以将时序当做一个存储器. ~~用于~~用于为组合提供输入和输出的场所. 那么，减法倒计时器 Down Counter是否也可以这样做.



Enable { Enable = 1, up
Enable = 0, down.

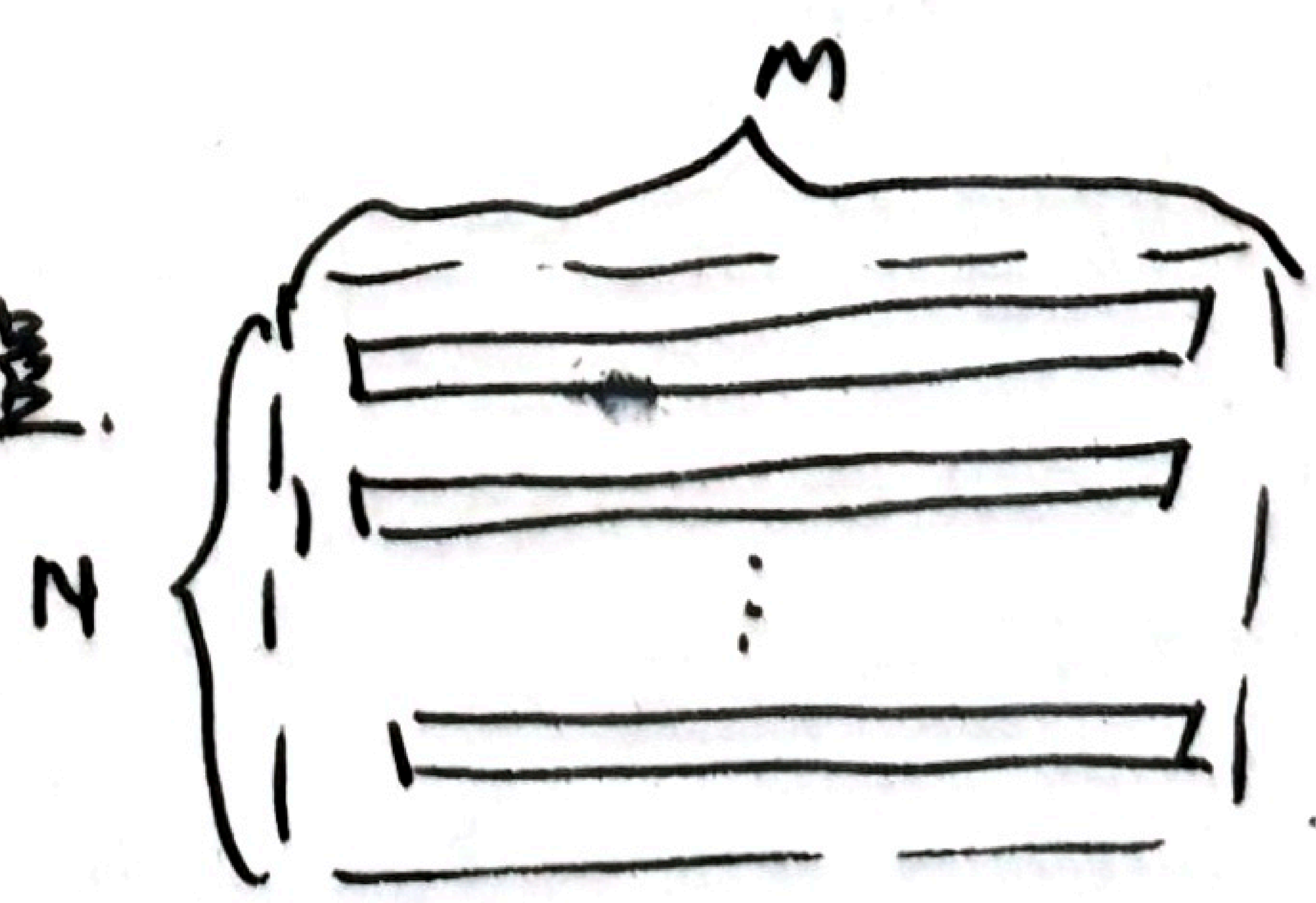
用VHDL代码实现，只需要

```
If(up) then
    Qint <= Qint + 1;
else
    Qint <= Qint - 1;
end if;
```

补充 1) Latches are often incorrectly generated when there is an existing condition to a circuit that does not have a corresponding output specification.

4. FIFO and LIFO Buffers.

1) M×N-bit. 多少位，M个字，每个字有N个位置.



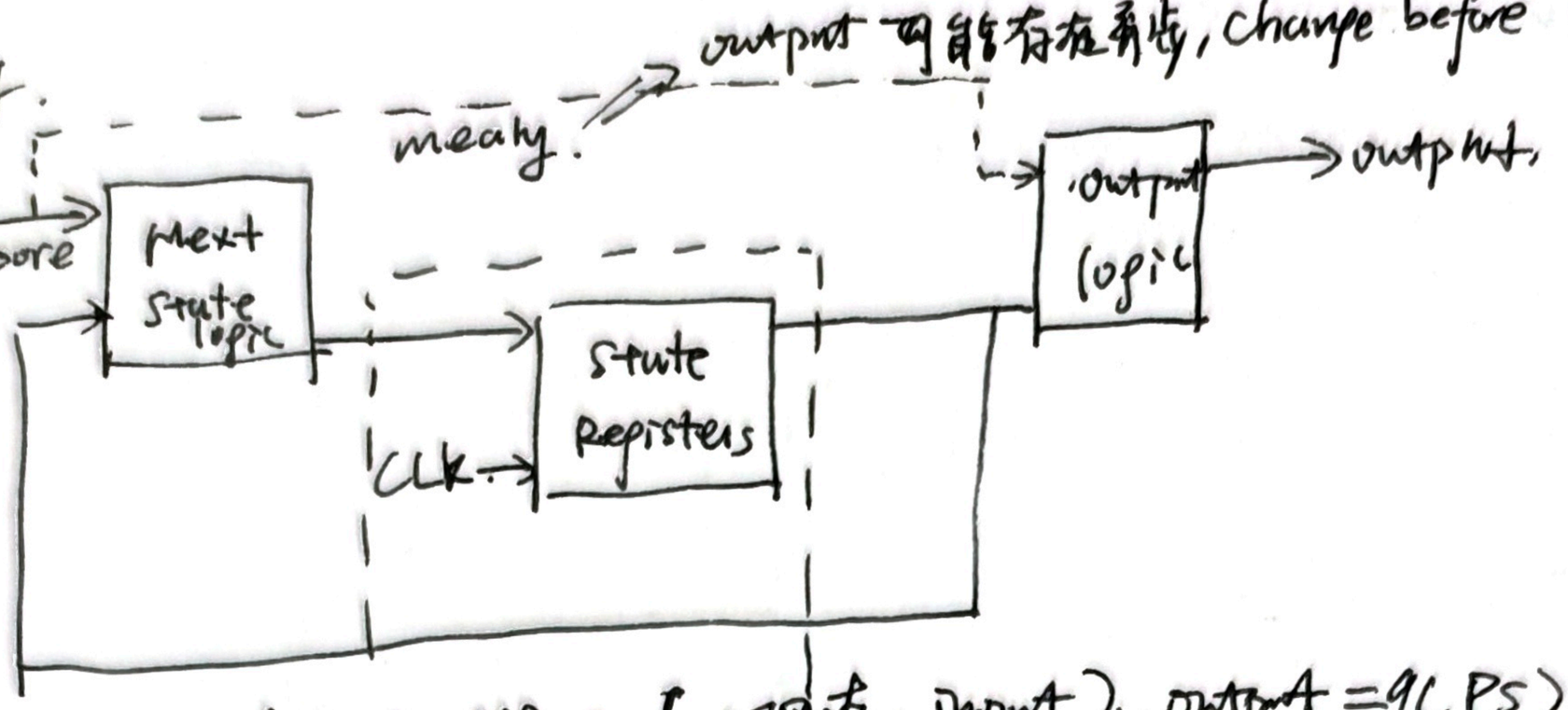
N个(M×1) shift Register 组成

2) FIFO 先进先出，LIFO 后进先出.

3) 时刻记住full时不可写入，否则 overflow，数据 lost，full和empty标志位是必要的！

<Block> = FSM and Datapath Control >.

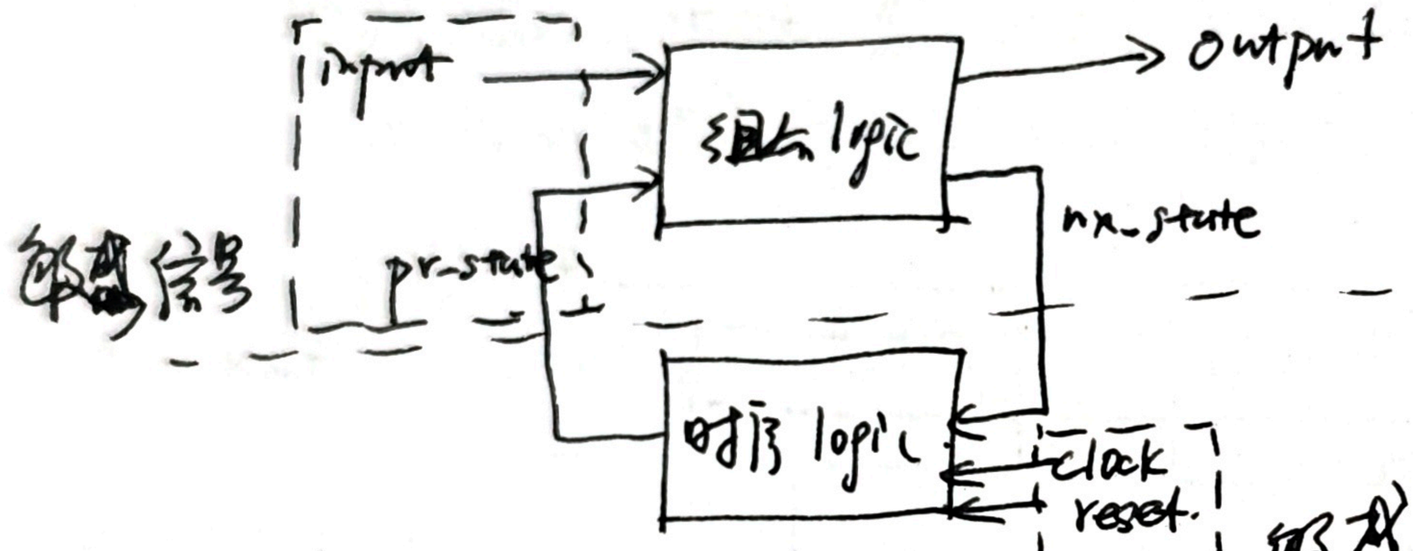
1. FSM Theory.



对于 Moore machines = NS = f (现态, input), output = g(PS)
PS - present_state.

对于 Mealy machines = NS = f (PS, input), output = f(PS, inputs).

2) 一个对时序部分有帮助的框图：（使用 state diagrams 和 state transition tables 都可对FSM刻画）.



使用两个process，一个表示组合logic另一个表示时序logic.

2. FSM VHDL设计.

State Assignment
1) 选择 initial state code 应为 reset, "0000"/"111111".
2) Minimise the no. of state variables that change on each transition.
3) 电路的复杂度（对 nx-logic）取决于 State assignment.

对于 unused states 的处理：
① Minimal cost : ignore, make circuit unreliable.
② Minimal risk : unused states always transit to a safe state
make the circuit be complex.

时延 设 logic 组合逻辑 需要 T_D 时间才能处理数据，则时钟周期最小应满足.
$T_{min} > T_D$, 即 $f_{max} < \frac{1}{T_D}$，才保证在下一个时钟沿来之前数据被处理好，即 circuit stable.

代码风格 使用两个process分别处理组合和时序 logic.

```
type state is (state1, state2.....);
signal PS, NS : state;
process
    case PS is
        when state0 =>
    end case ....
```

} 命名状态变量

} 对时序变量用 case-when 处理

State Variables Output))一般状态引用由 type state is (...)声明，具体的编码方式留给 synthesis tool 3，但有时可能要求输出编码.

Code 编码

① with PS select

Y <= '0' when ST0;
'1' when ST1;

② type state is (ST0, ST1)
attribute enum_encoding : ~~string~~ string;
attribute enum_encoding of state :
type is "0 1";
signal PS, NS : state;

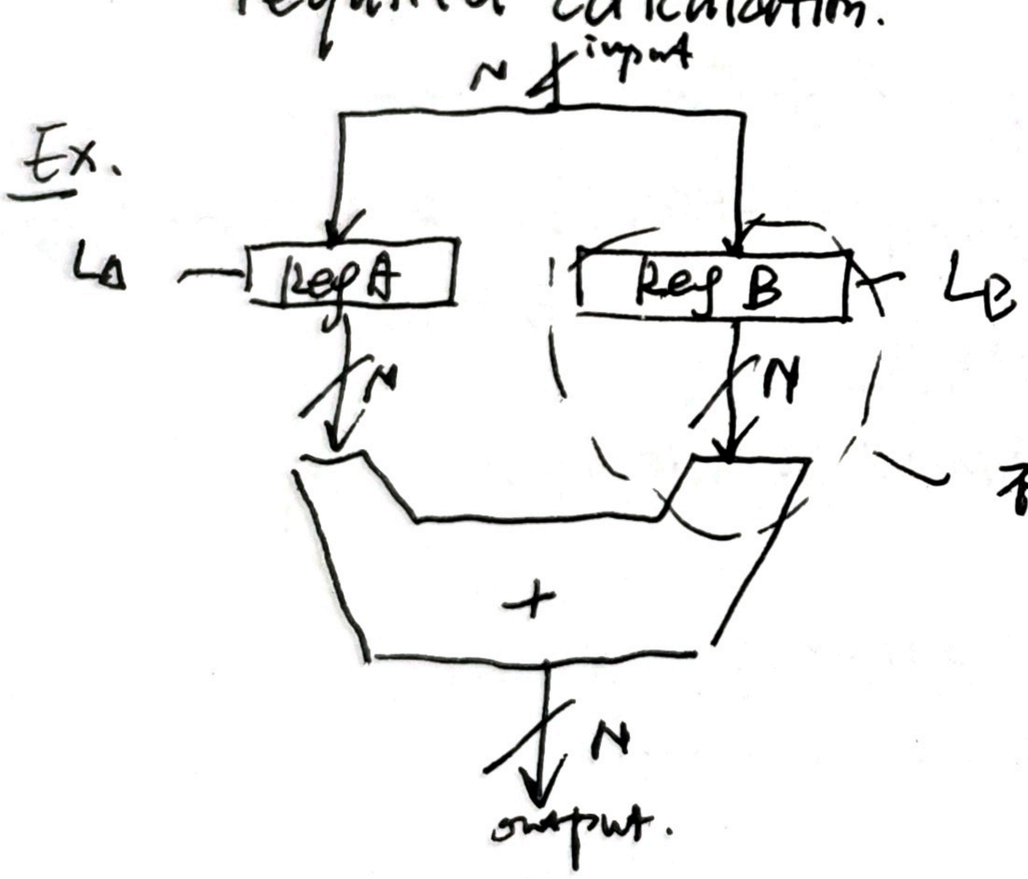
3. Datapath Control.

Methology **Datapath**: provides status to control unit and makes decisions on next state.

FSM 控制 → **Control unit**: determines the sequence of operations.

question number of input ports are limited ⇒ load the operands sequentially
Need to control data flow and store data in order to perform required calculation.

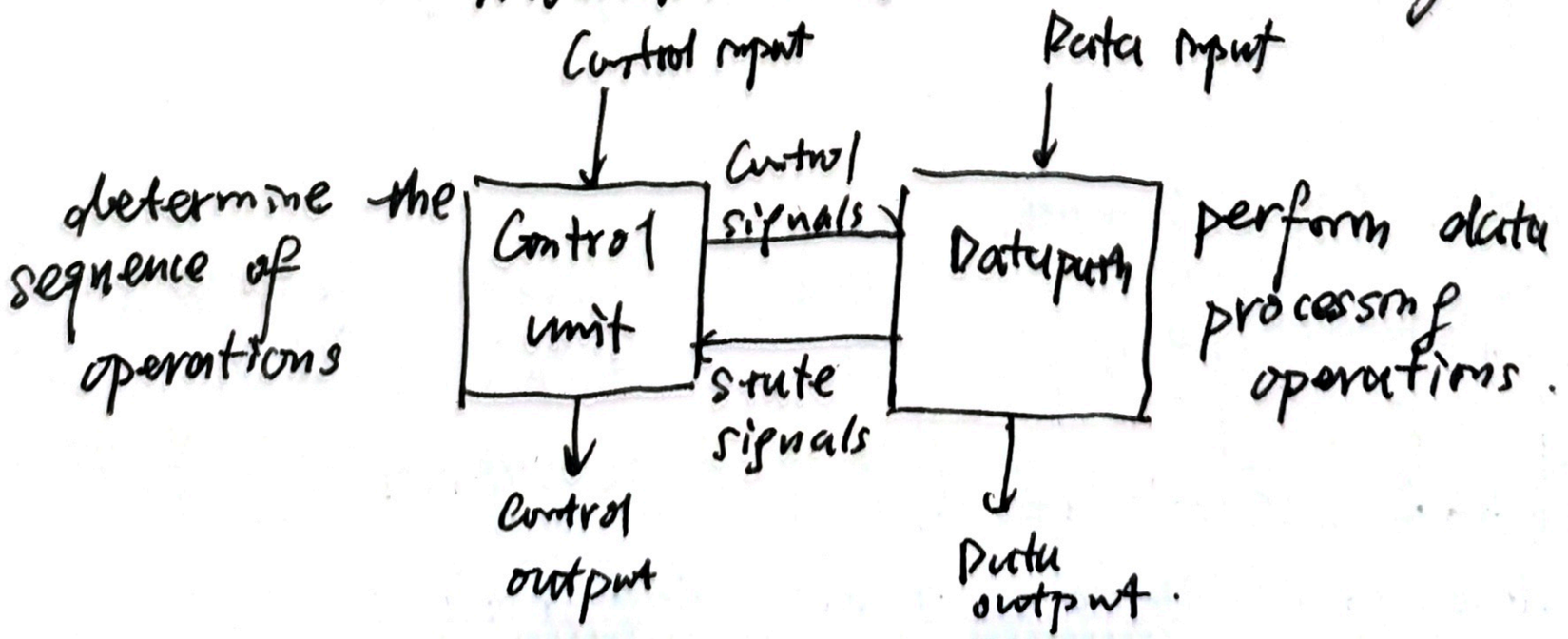
Ex.



Break whole task into smaller subtasks which can be undertaken sequentially within FSM.

不让 output directly linked to input，因为是组合 logic，容易出现问题.

Large Digital System ※ sequential circuit made up of interconnected flip-flops and gates: Partitioned into modular subsystems —— modular and hierarchical design.



Register Transfer

Register transfer operations = movement of data stored in register and processing performed.

Register transfer Notation (RTN) = notation which describe the datapath.

使用 R + num的方式表示 Register.

① 简单transfer： $R2 \leftarrow R1$，(R1存的值transfer给R2).

② 条件transfer： K_1： $R2 \leftarrow R1$，$R1 \leftarrow R2$，当 K_1 和时钟沿同时满足时 R2和R1交换值.

在表示为(算法) if(t) then
$R_2 \leftarrow R1$; $R1 \leftarrow R2$;
end if;

③ 带操作的transfer：
Micro-Operations.

Arithmetic = $R0 \leftarrow R1 + R2$.

~~R2to~~ $R0 \leftarrow R1 - R2$ 或 $R0 \leftarrow R1 + (R2)'_{+1}$.

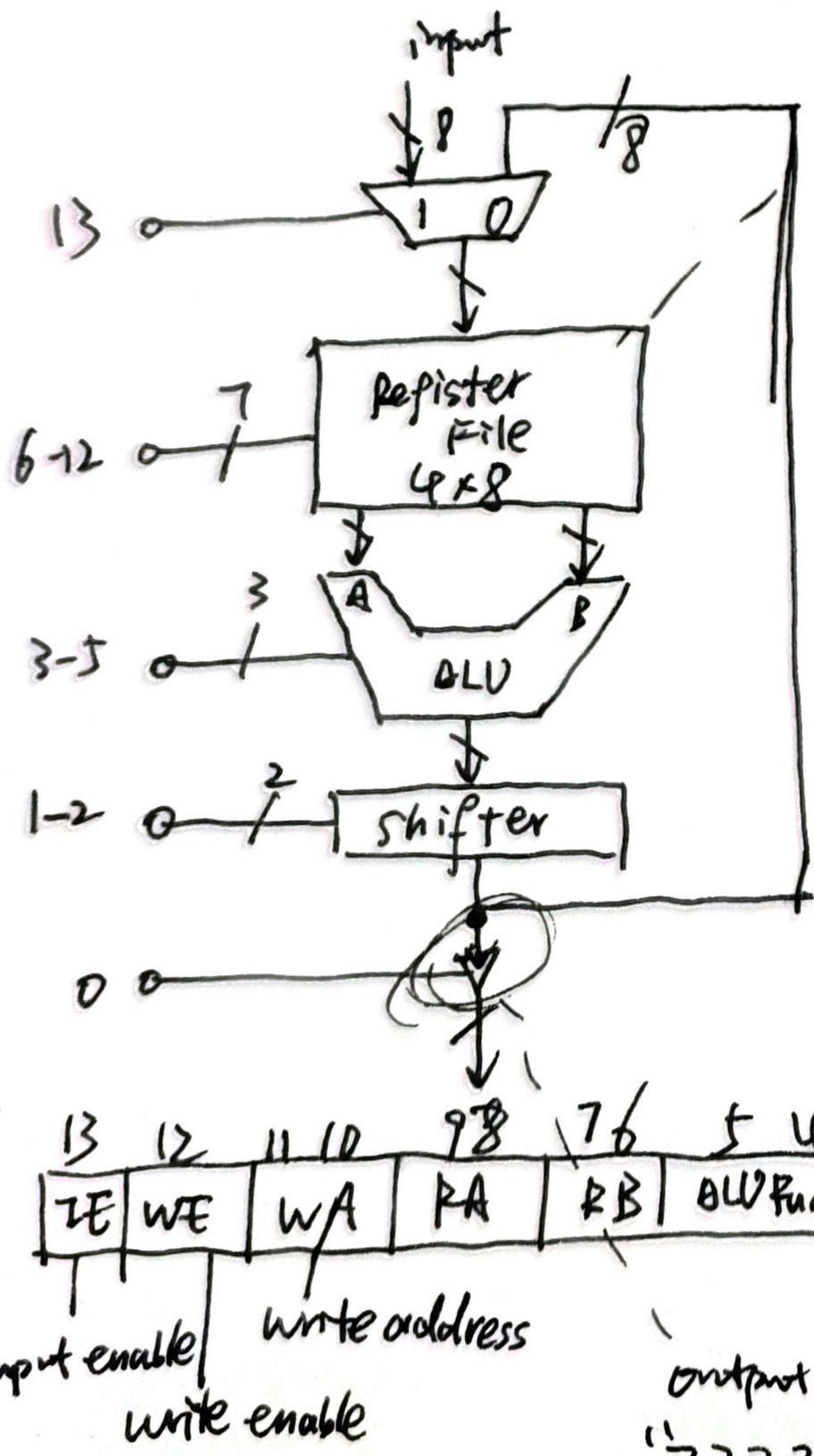
Shifter = $R0 \leftarrow sl\ R0$, $R0 \leftarrow sr\ R0$

等算法中用 & 符号，一般表示补 0.

此外还有 logic运算. and, not, or, xor

(4个word(位宽)寄存器8bit，R0, R1, R2, R3).
单本输入，两条行为选择输出的 Register file.

Powerful datapath



datapath 很大时：there can be large number of control signals.

Control word：a collection of control signals sent to the datapath blocks.

Control word 去掉指具体应用去设计的东西.

算法 检查8bit数中有 N 个"1".

R0 ← mpA
R2 ← 0000 0000
R1 ← 0000 0001
while (R0 != 0) do }.
R3 ← R0 AND R1
R2 ← R2 + R3
R0 ← sr R0.
}
outputA ← R2, done ← '1'.

output 一直都是 "ZZZZZZZZ"(高阻态)，在赋值之前 (OE=1).

Note 对于 dataPath，置0或置1或许是不很直观的事情.

R2 ← (R2 xOR R2)：置0，R2 ← (R2 xOR R2) + 1，置1.

R2 ← (R2 and R2) 或 R2 ← (R2 or R2) 不变.

R2 ← sr (R2 and R2) 或 R2 ⇐ sr (R2 or R2) 对自己移位.

自身相 xOR，自身相 and，or 可以完成置0，置1，和自身移位的操作.

总结：Just design a ~~certain~~ certain datapath with suitable micro-operations and give a FSM that properly controls the steps and dataflow.

<Block 4：Memory and Arithmetic>.

1. Digital Memory Devices.

Define memory Bits that are stored in a structured way.

Size 与十进制不同，这里的计数法是2的幂指数计数法. $K \sim 2^{10}$, $M \sim 2^{20}$, $G \sim 2^{30}$, $T \sim 2^{40}$.

一般有 100 GB 指 100×2^{30} bytes，大写的B指字节，1 Byte = 8 bit.

align data words in parallel/serial.

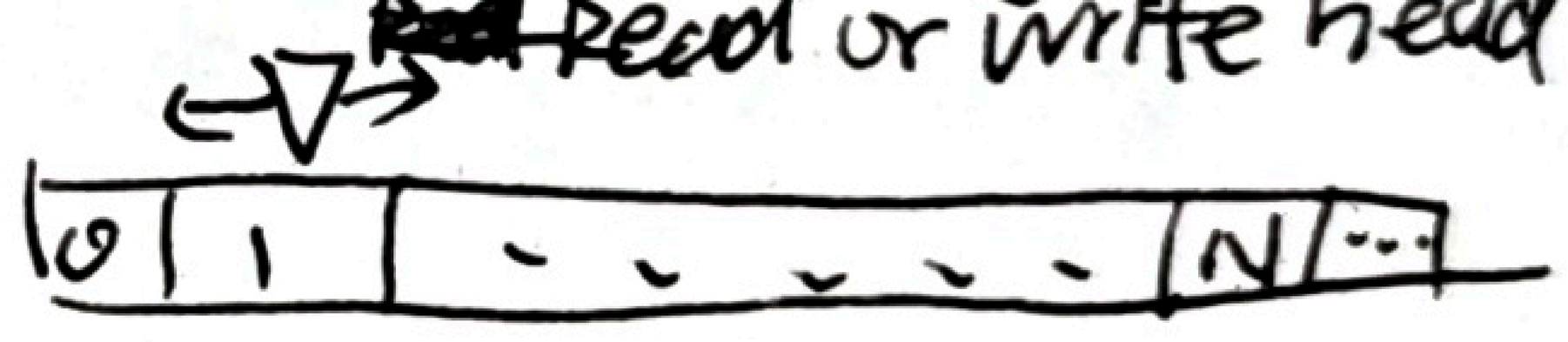
RWM (Read/Write memory)

① ROM：Read only memory

② RAM：Random-access memory

与 Random-access 对应的是 Serial Access. Data bits are stored serilly

~~Read~~ Read or write head → 不可随机的access.



只可顺序的 travel 到下一个 location.

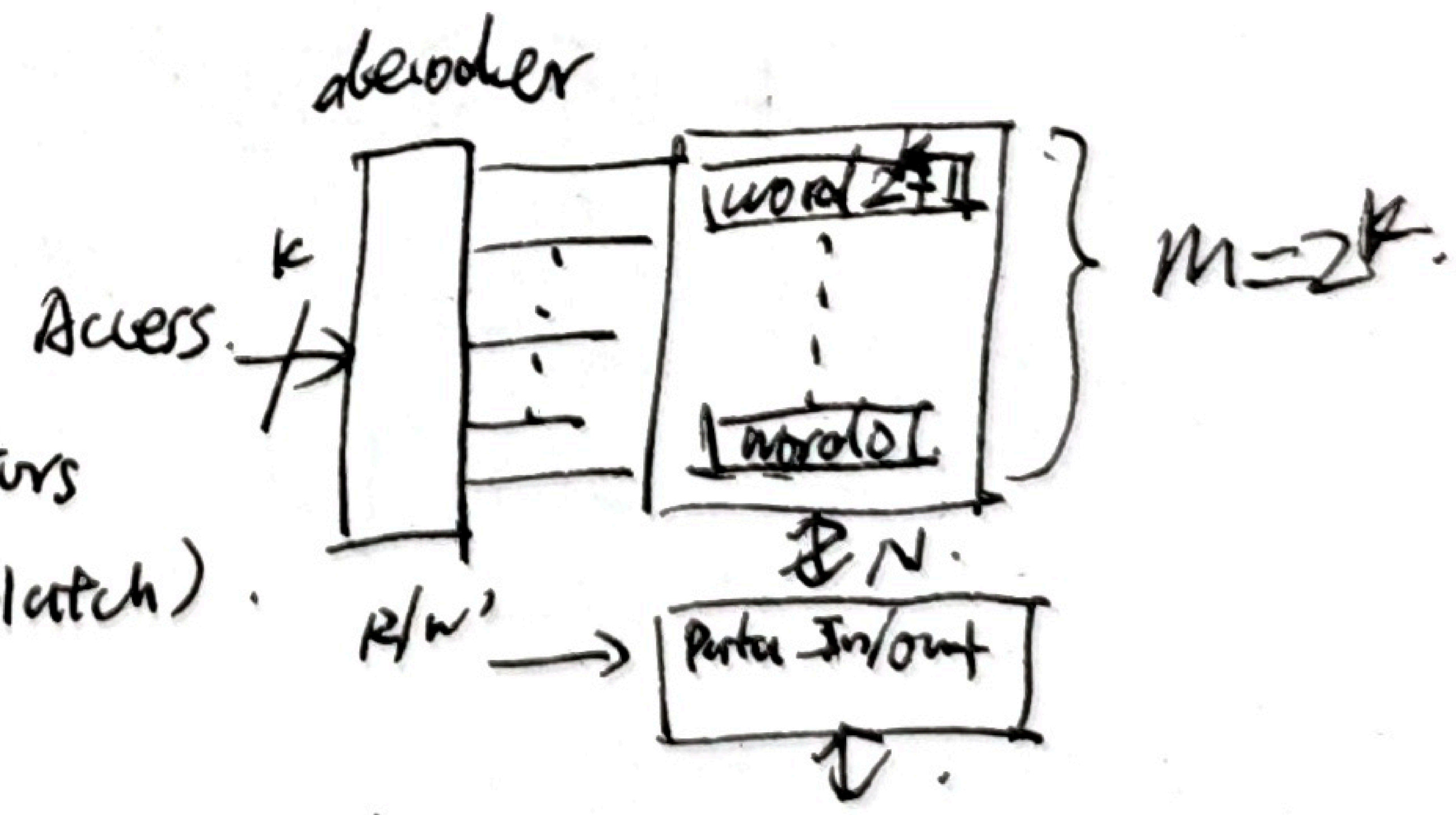
与 Serial Access 最大的区别：read or write a bit is independent of the bit's location.

但是大多数都是 volatile (lose memory when power is removed).

a) Static RAM (SRAM).

断电或覆盖 → 数据丢失.

基本存储单元是 D-latch (由4到6个 transistors 组成或不完全是 D-latch).



区别于 FIFO, LIFO Buffer 不同，
这里每个 Register 是一个字.

b) ~~Dynam~~ Dynamic RAM (DRAM).

该数据或时间长了会丢失.

Refreshed periodically by reading it and writing it back.

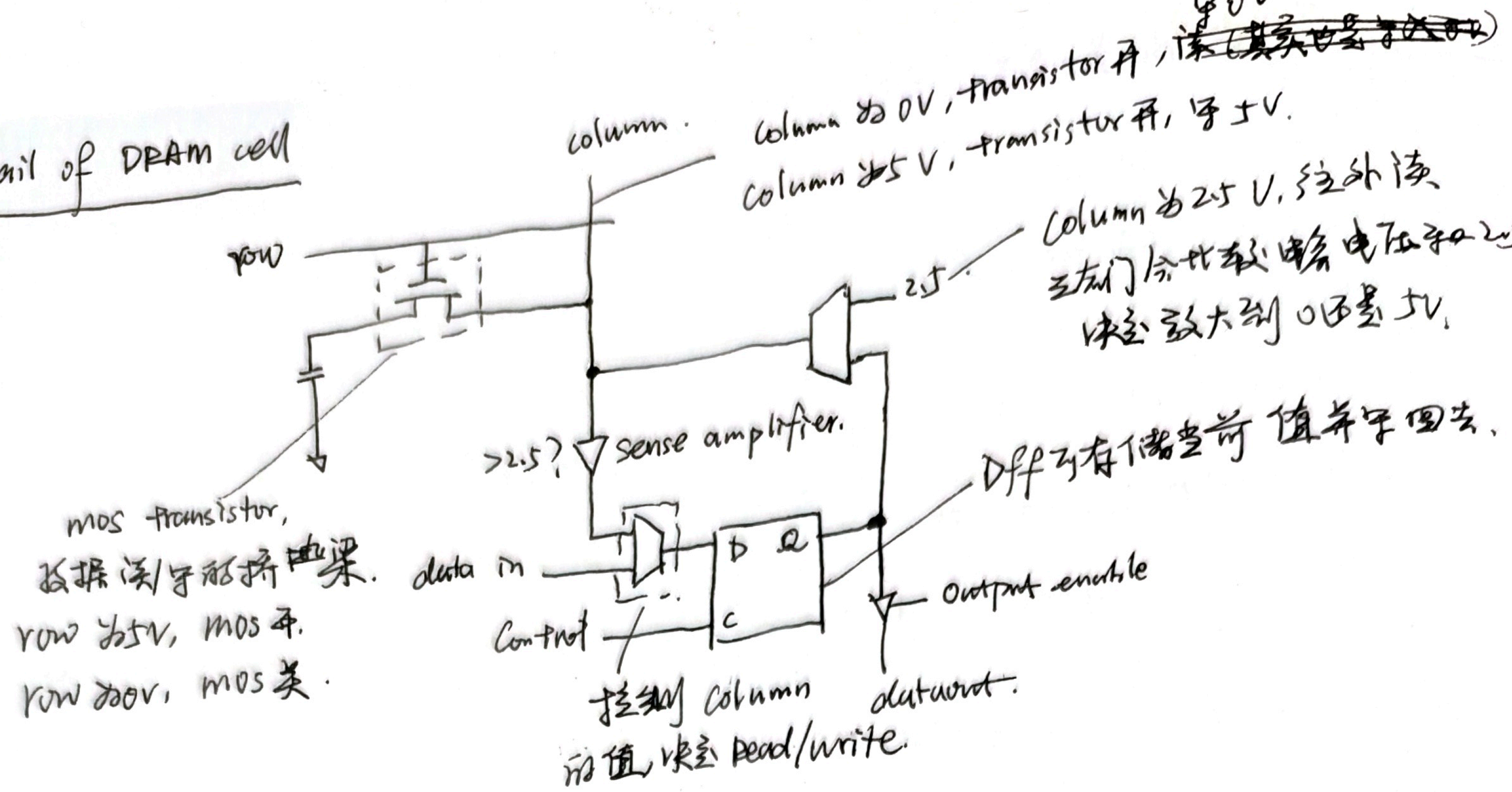
process：由 capacitor 和 transistor 存储一个 bit → mos transistor

当 transistor 关闭时，charge on capacitor 会保持 (remain)

但是 imperfect isolation across transistor cause charge to leak

DRAM 的 density 比 SRAM 更大，cost per bit lower.

detail of DRAM cell



2. Binary Arithmetic.

Unsigned Multiplication. 移位乘即可. (A×B=S, N bit × N bit → 2N bit).

可以有 ~~long~~ longhand multiplication (竖乘式方法)

和 Alternative scheme 两种方法 (Running sum)

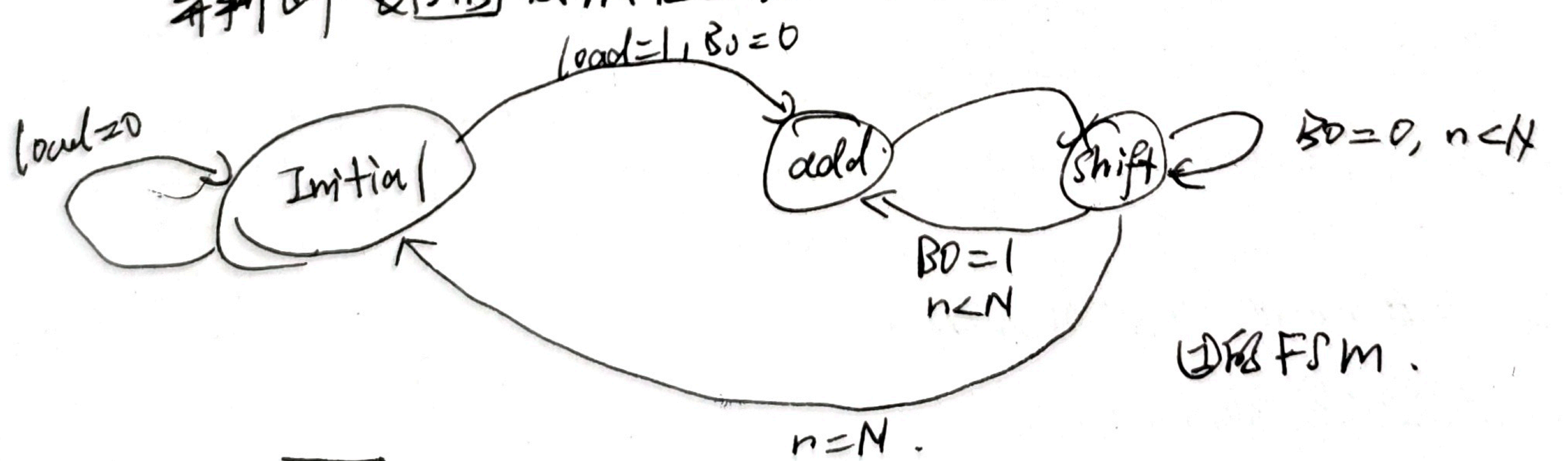
→ 其实这两种方法都是 long hand multiplication

对于Alternative, 也有两种算法.

① 固定S不动 (Running sum), 每次移位A和B (A<<1, B>>1), 判断B0是否为1, 决定加还是跳过, 可以由组合logic或 control unit 控制的datapath完成.

用 and 代替判断B0是否为0.

② 固定A不动 (最高N比特), 将 [S|B] 合并为2N bit, 每次 [S|B] 右移 并判断 [S|B] 最低位是否为0. 节省了N bit 内存.



①的FSM.

②的补充: allocate [2N] bits to the running sum and multiplier.
Combine the Running Sum and the multiple.

注意此时可能有进位, 需要多一个Register存储carry out的信息.

Signed multiplication

- Q1：因为移位，首位没对齐，符号错误 ↦ 扩展到相应符号位再加.
- Q2：running sum 溢出 ⟼ 每次移位，扩展一位.
- Q3：负数时，最高位未被正常对待 ↦
  - ① Correction at End
  - ② Correction at Last step.

~~对于~~ 对于 signed 数，在MSB 填充任意 bit 与MSB相同的 bit，数值不变，所以用该操作对 Signed 数进行扩展

Correction at end：

Correction is N bit and is shifted N bits to left. $2^N(2^N - A)$.

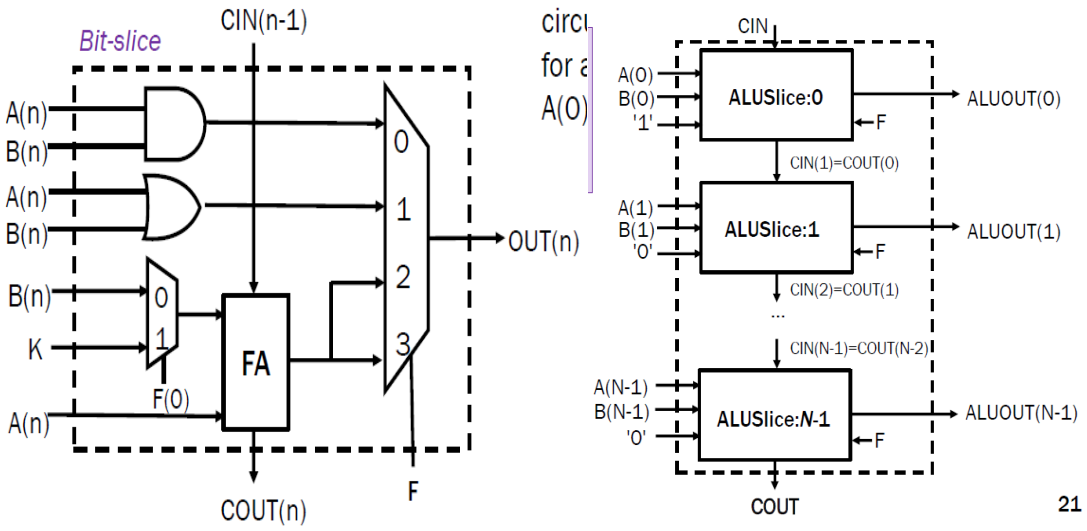
Correction ~~at~~ in last step：

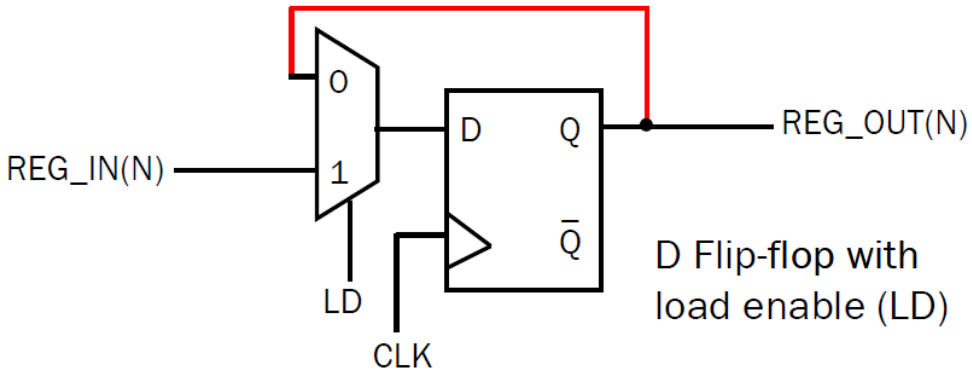
Correction is (N+1) bits and is shifted (N-1) bits to left.

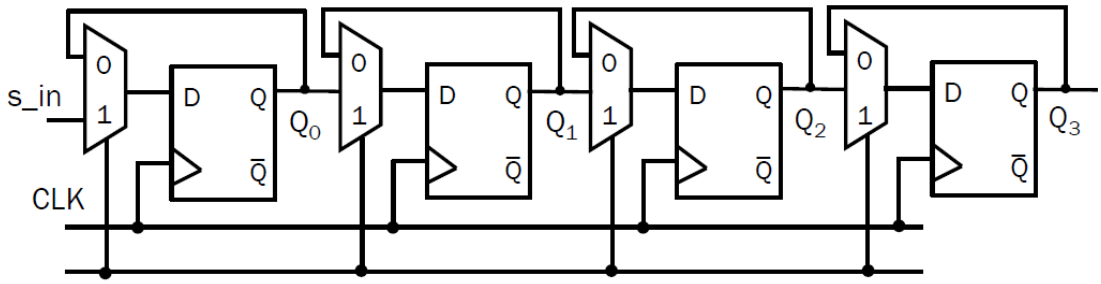
$2^{N-1}(\underline{2^{N+1} - A})$

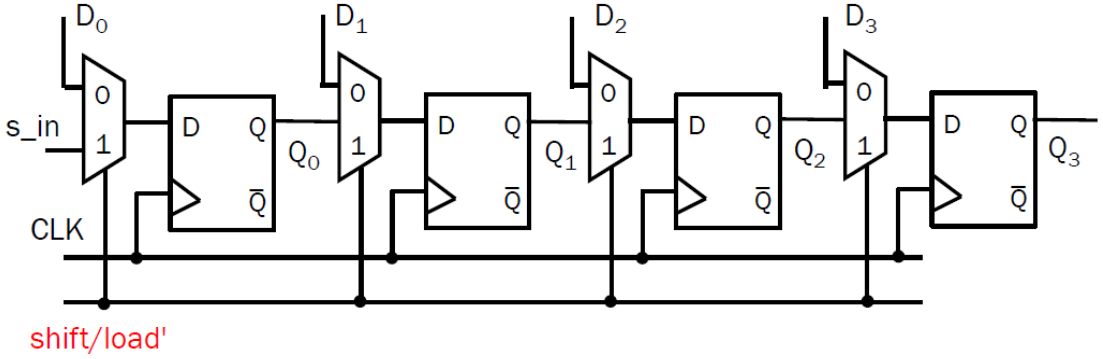
增添一位再取反！

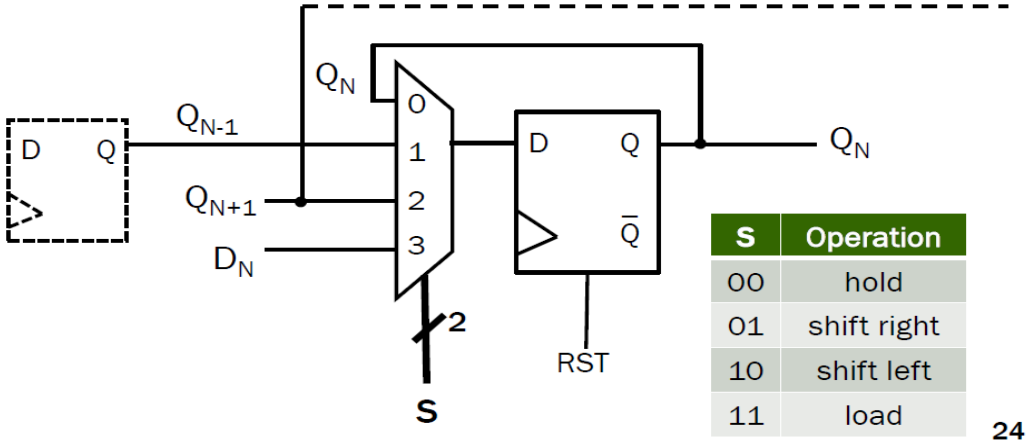
相比前一个移位多移一位的移位！


Bit-slice
CIN(n-1)
A(n)
B(n)
A(n)
B(n)
B(n)
K
A(n)
F(0)
FA
OUT(n)
COUT(n)
F
circ
for a
A(0)
CIN
A(0)
B(0)
'1'
ALUSlice:0
F
ALUOUT(0)
CIN(1)=COUT(0)
A(1)
B(1)
'0'
ALUSlice:1
F
ALUOUT(1)
CIN(2)=COUT(1)
...
CIN(N-1)=COUT(N-2)
A(N-1)
B(N-1)
'0'
ALUSlice:N-1
F
ALUOUT(N-1)
COUT
21

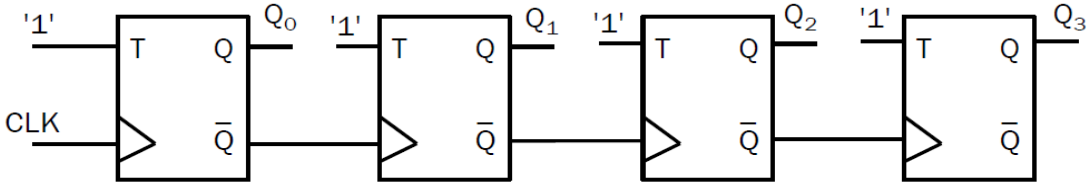

REG_IN(N)
REG_OUT(N)
LD
CLK
D Flip-flop with
load enable (LD)


s_in
CLK
shift/hold'


s_in
CLK
shift/load'



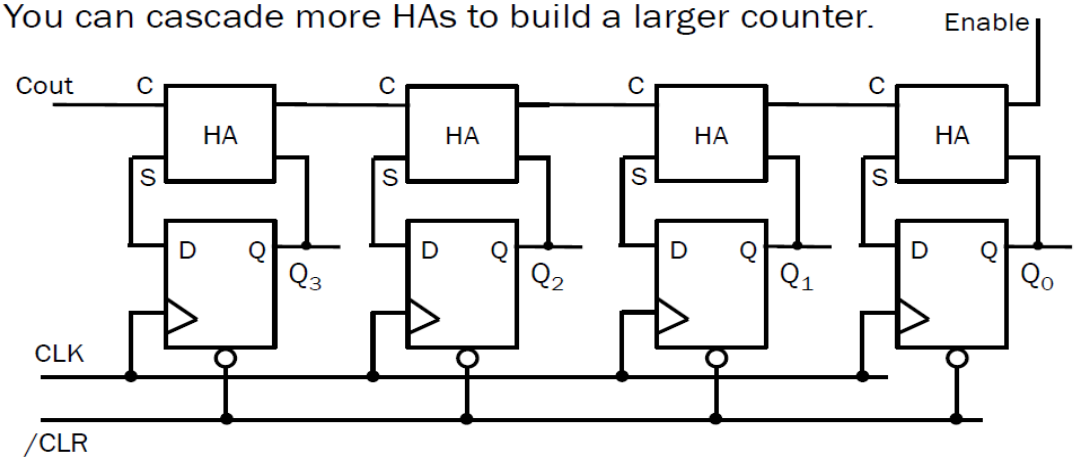
S	Operation
00	hold
01	shift right
10	shift left
11	load



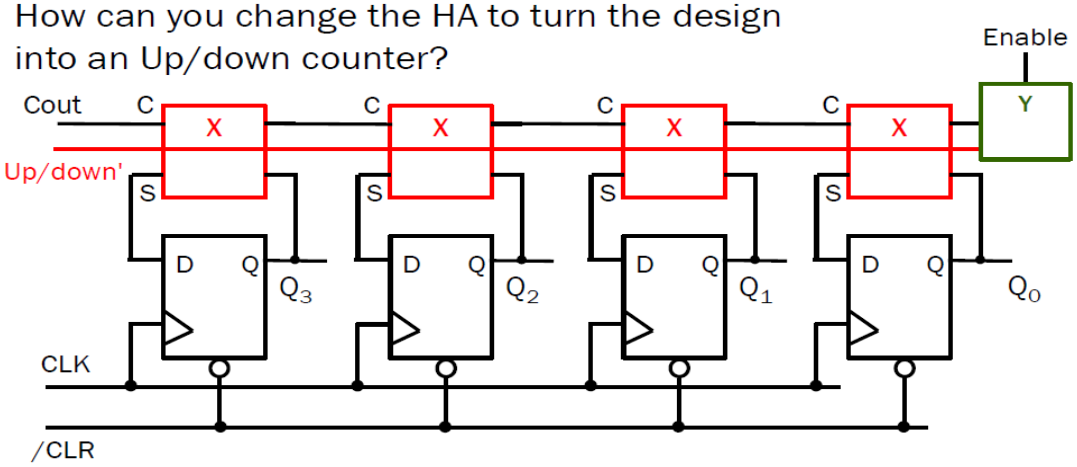
4-bit binary ripple counter

4-bit synchronous counter

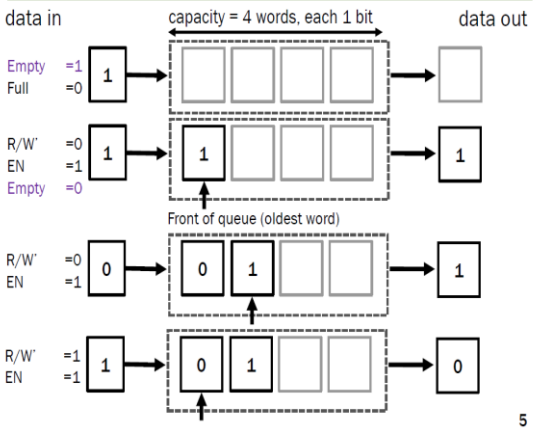
You can cascade more HAs to build a larger counter.



How can you change the HA to turn the design into an Up/down counter?

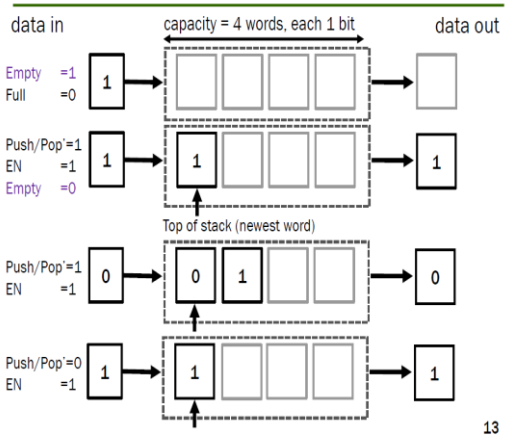


4 x 1 FIFO: Example



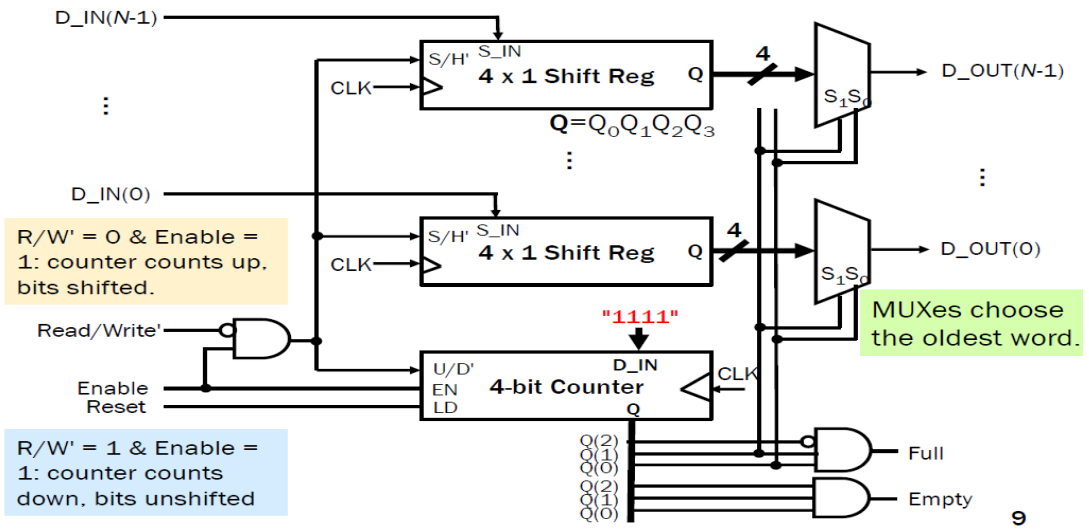
5

4 x 1 LIFO: Example



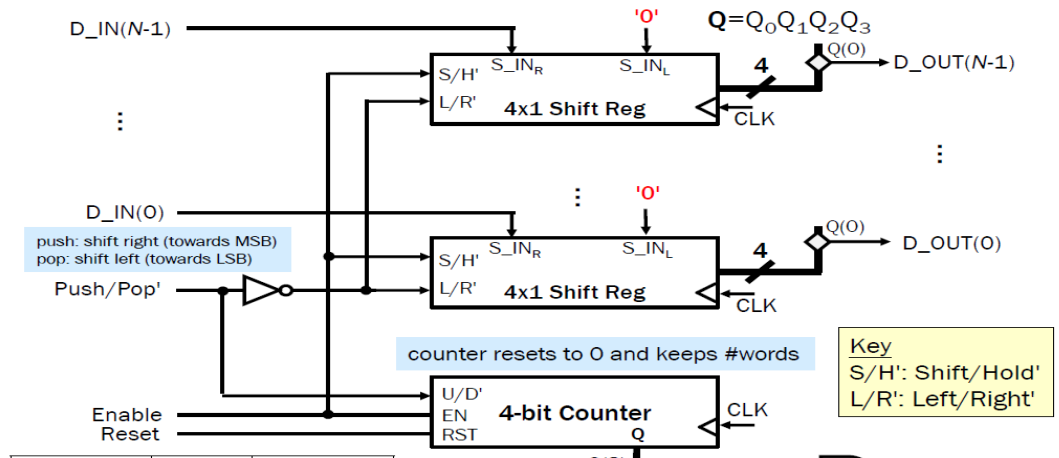
13

Building FIFO: Completed



9

LIFO: Shift Register Implementation



Push/Pop'	Enable	Operation
X	0	No change
0	1	Pop
1	1	Push

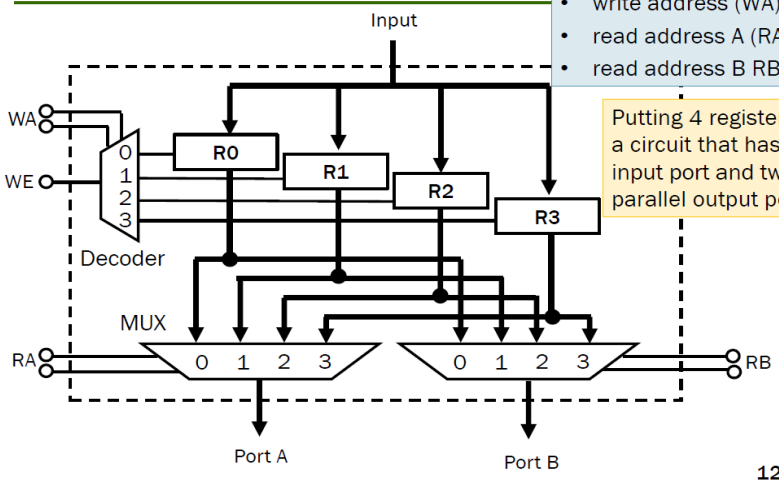
Again, let $M = 4$

16

Register File

Control signals are:
- write enable (WE)
- write address (WA)
- read address A (RA)
- read address B RB)

Putting 4 registers in a circuit that has 1 input port and two parallel output ports.



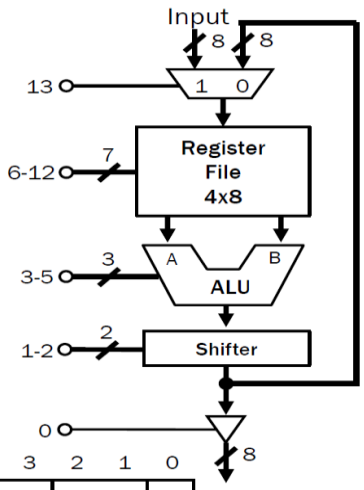
12

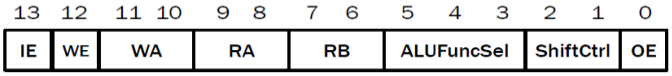
ALU

Control	Function	Control	Function
000	NOT A	100	Add
001	AND	101	A + 1
010	OR	110	Subtract
011	XOR	111	A - 1

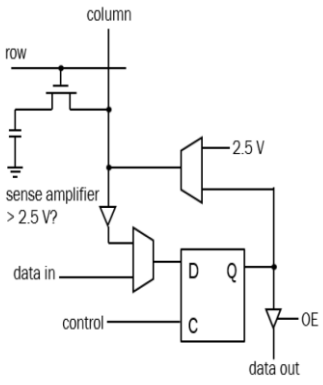
Shifter

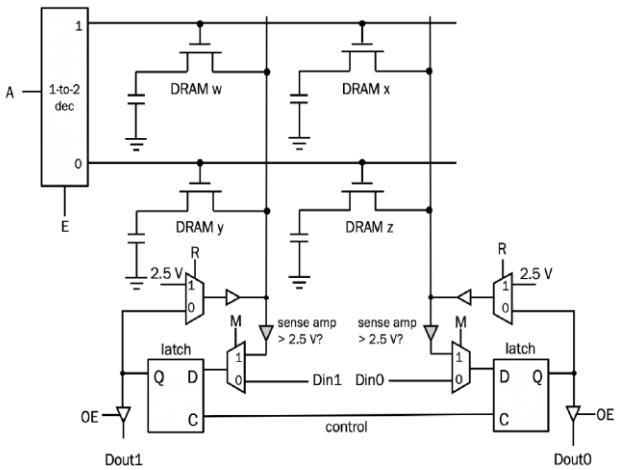
Control	Function
00	Pass
01	Shift left
10	Shift right
11	Pass

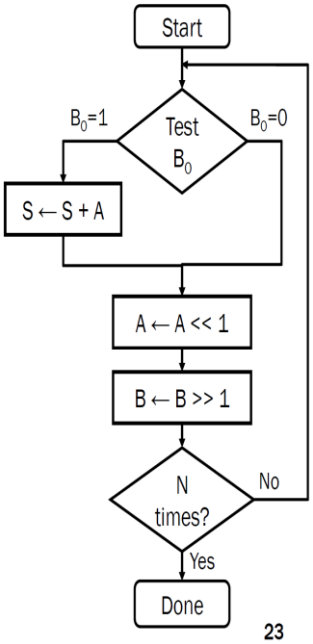




4

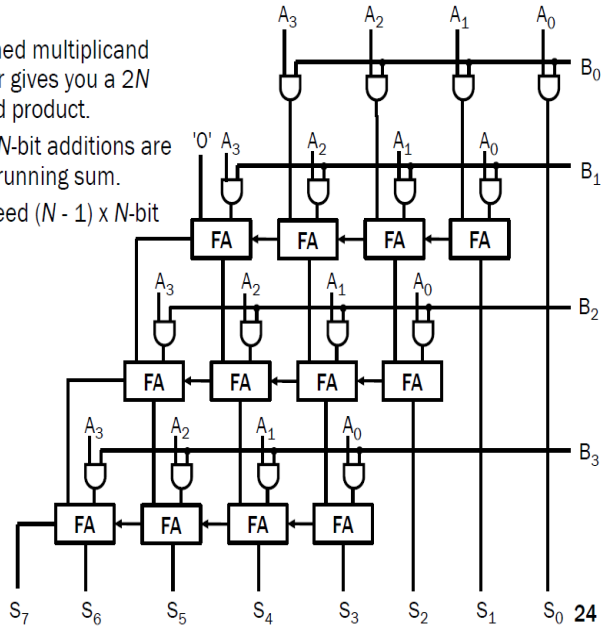






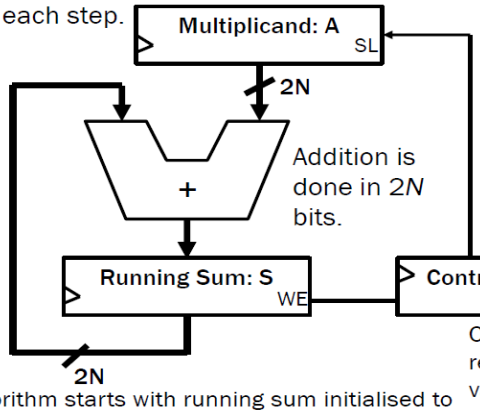
23

- *N* bits unsigned multiplicand and multipler gives you a 2*N* bits unsigned product.
- Only (*N* – 1) *N*-bit additions are required for running sum.
- So we just need (*N* - 1) x *N*-bit adders.



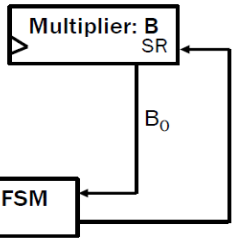
24

The *N*-bit multiplicand starts in the right half of the register and is shifted left 1 bit on each step.



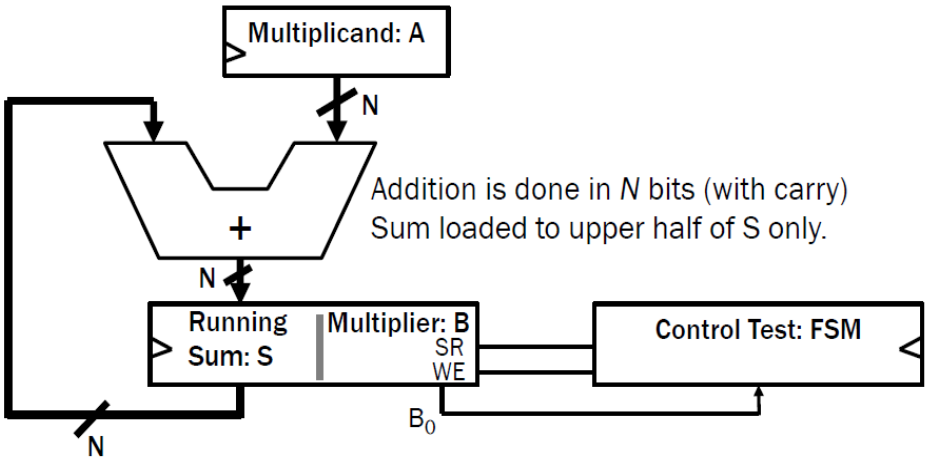
Algorithm starts with running sum initialised to 0, multiplicand loaded to A and multiplier loaded to B (connections not shown here).

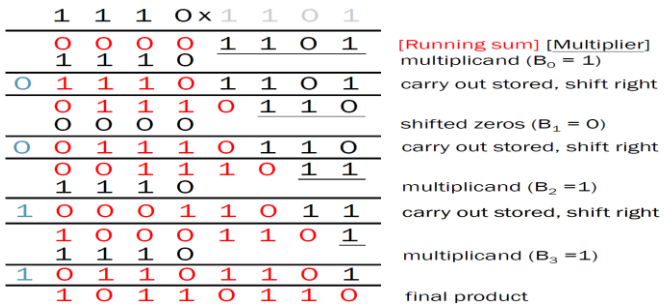
The multiplier is shifted in the opposite direction at each step.

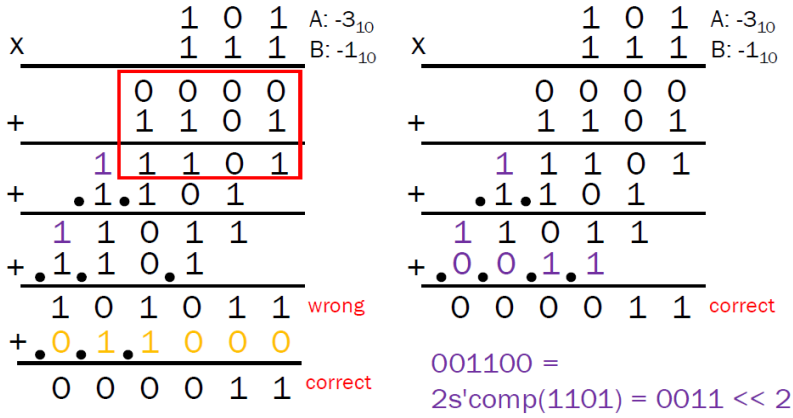


Control unit decides when to shift registers A and B and when to write new values into the register S.

25







a) You are asked to design a digital system, whose specification is given as:

The system counts the number of button presses that a person can do in a second.

If the system shall be implemented by an FPGA development board, like what you did in our lab and design project, explain how you would model the inputs and outputs of the system using digital devices and create initial design of your system (i.e. entry stage in the design process).

[5 marks]

Solutions

Modelling 1: input – the push button and the clock signal

(1 mark)

Modelling 2: output – some kinds of displays, e.g. LEDs or 7-segment displays to show the number of presses

(1 mark)

Entry 1: provided the design models in HDL like VHDL or Verilog

(2 marks)

Entry 2: the design is likely to be modelled as a FSM as well

(1 mark)

i) What is the key advantage of the bit-slicing technique in circuit construction?

(2 marks)

i) This technique allows easy expansion of the ALU to an arbitrary bit width.

(2 marks)

Solution

```
comb_proc: process(PS)
begin
```

(1 mark)

```
    Sound_Bell <= '0';      -- pre-assign output
    Cooker_On <= '0';       -- pre-assign output
```

(1 mark)

VHDL code

```
architecture Arch_Microwave_Oven of Entity_Microwave_Oven is
    type state_type is (Initial, Cooking, Stop_Cooking); -- e.g. 3 states
```

(2 mark)

```
    attribute ENUM_ENCODING: STRING;
    attribute ENUM_ENCODING of state_type:
```

(2 mark)

```
        type is "100 010 100";
    signal PS, NS: state_type;
```

(2 mark)

Exercises on Datapath and Control

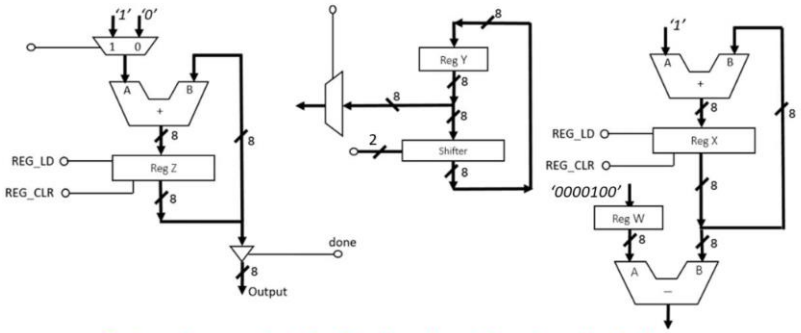
Q8

The following algorithm is used to count the number of consecutive 1's in an 8-bit binary number. Answer the following questions

```
i = 0, Number_consecutive_1s = 0, Number_of_Bits = 8
while ( i < Number_of_Bits ) do {
    if ( i > 0 and Binary_Number(i)  = '1' and Binary_Number(i-1)  = '1') {
            Number_consecutive_1s  = Number_consecutive_1s + 1
    }
    i = i + 1
}
output(Number_consecutive_1s);
```

a) Express the algorithm in RTL/RTN. You should generate a done signal when the algorithm finishes. In addition, you should use no more than four registers

b) The figure shows the datapath associated with the algorithm described above. Explain the correspondence between the variables used in the algorithm and the registers (labelled W, X, Y, and Z) in the figure



Datapath associated with the algorithm described above