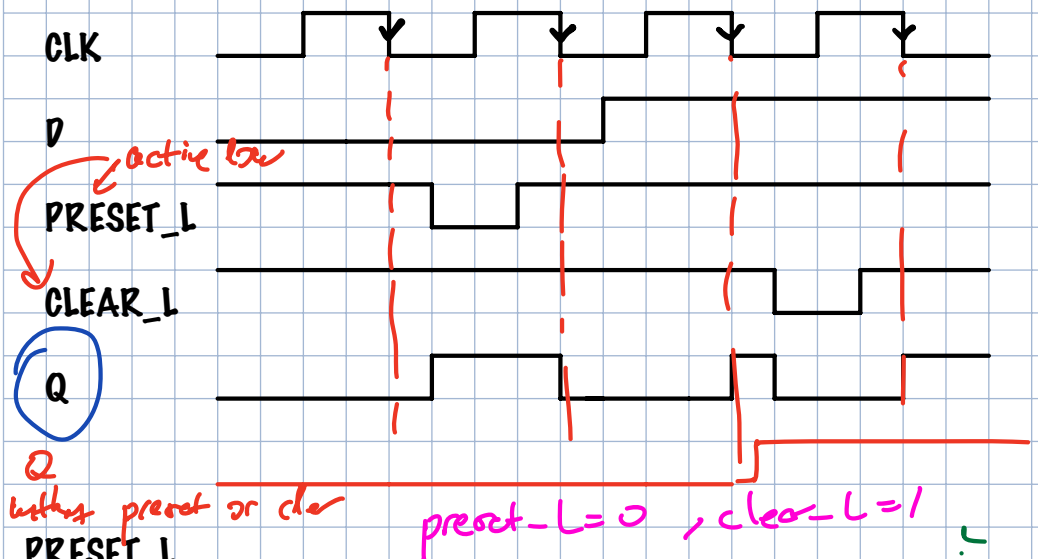
Chapter 7 - Sequential Circuits Part 2

D-FF with asynchronous clear and preset

reset — set

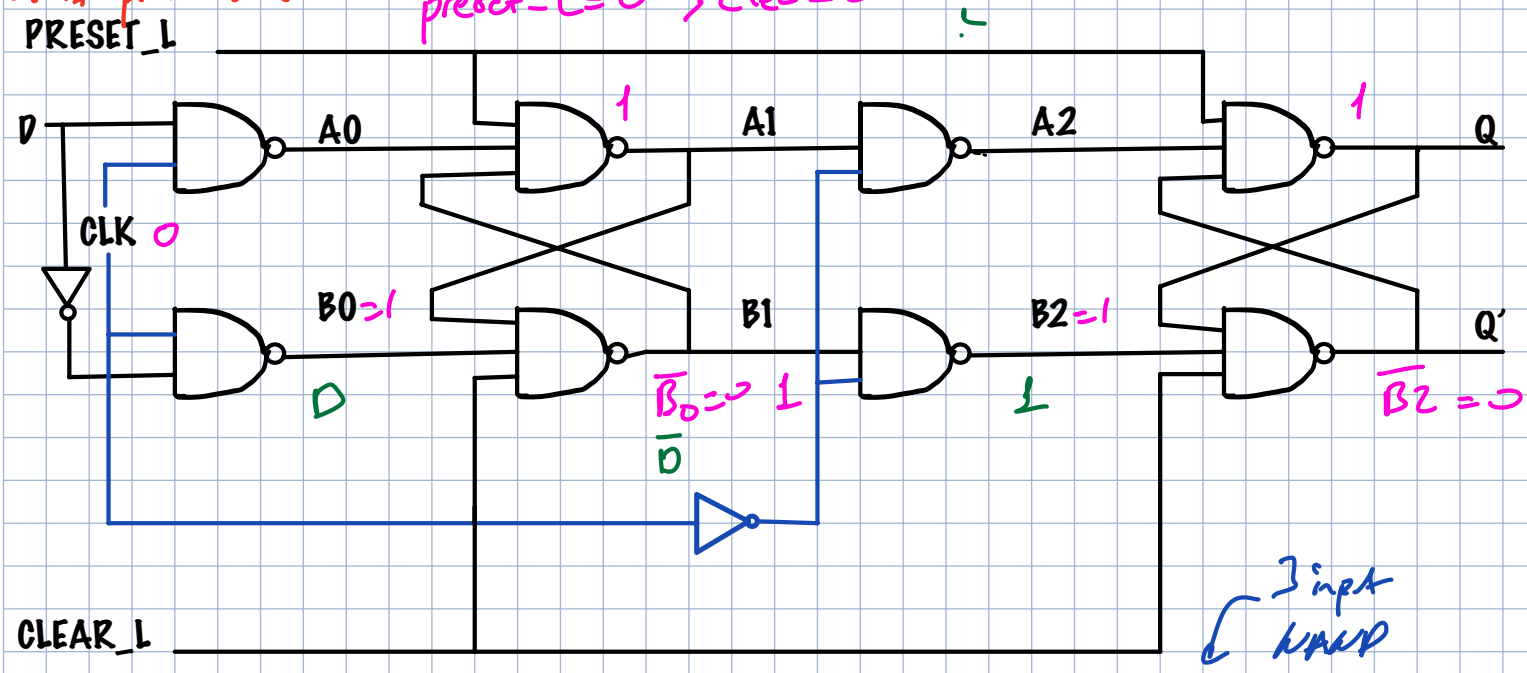
↳ not following the CLK



active low

Q with preset or clear

- PRESET_L = 0 -> Q = 1
- CLEAR_L = 0 -> Q = 0
- both cannot be zero at the same time



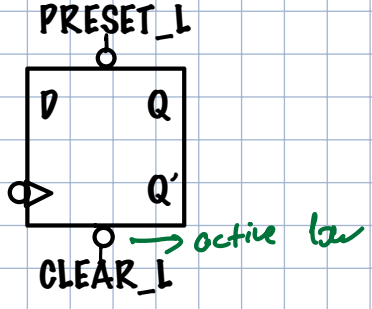
- CLK=0: A0=1, B0=1
  - PRESET_L=0, CLEAR_L=1: A1=1, B1=0
    - A2=0, B2=1: Q=1, Q'=0
- CLK=1: A2=1, B2=1
  - PRESET_L=0, CLEAR_L=1: Q=1, Q'=0

3 input NAND

preset_l or clear_l	a	b	atpt
active	0	0	1
active	0	1	1
active	1	0	1
inactive	1	1	1
inactive	0	0	1
inactive	0	1	1
inactive	1	0	1
inactive	1	1	0

output = $\overline{a \cdot b}$

symbol:



→ active low

VHDL code for D-FF with asynchronous clear and preset

since these operations are asynchronous.

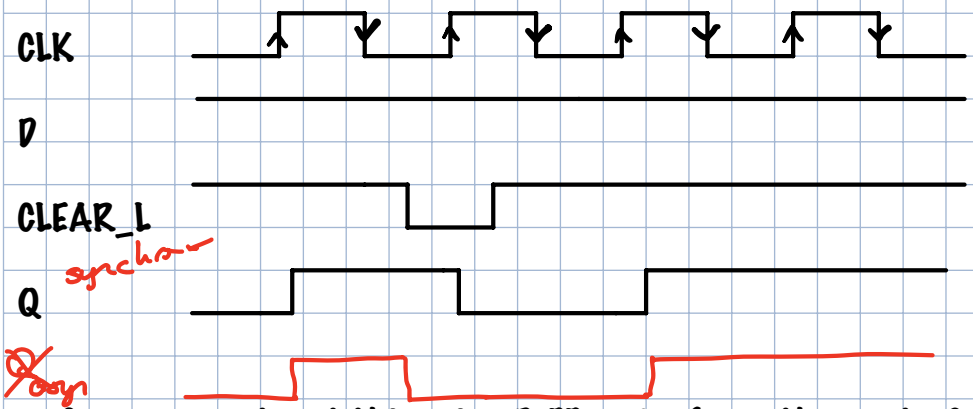
```
process(CLK,CLEAR_L,PRESET_L)
begin
    if PRESET_L='0' then
        Q<='1';
    elsif CLEAR_L='0' then
        Q<='0';
    elsif falling_edge(CLK) then
        Q<=D;
    end if:
end process;
```

implied memory

VHDL code for negative edge triggered D-FF

```
process(CLK) begin
    if falling_edge(CLK) then
        Q <= D;
        Qn<= not D;
    end if;
end process;
```

other-ck



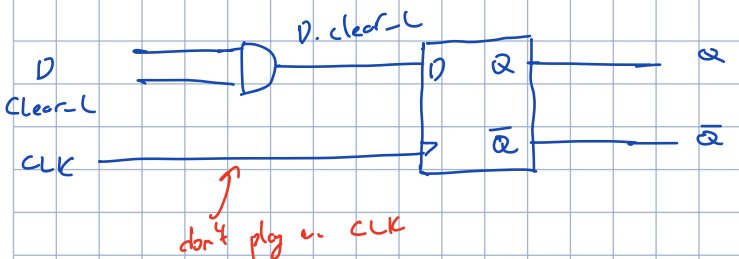
synchronous

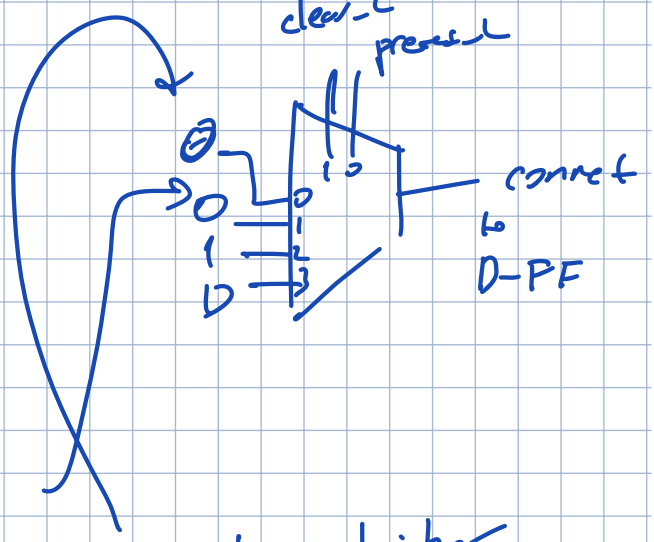
Can we construct this using D-FF and a few other gates?

only at active edge of CLK

→ D-FF with synchronous clear

(positive edge triggered)

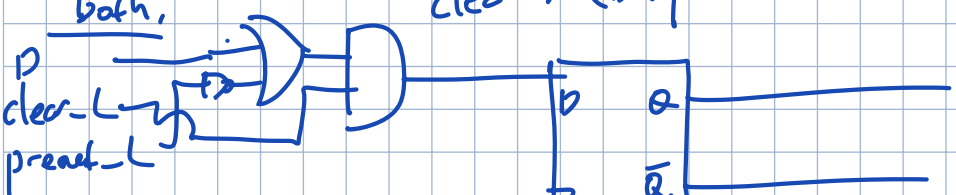




synchronous active low preset

both?

clear · (D+preset_L)

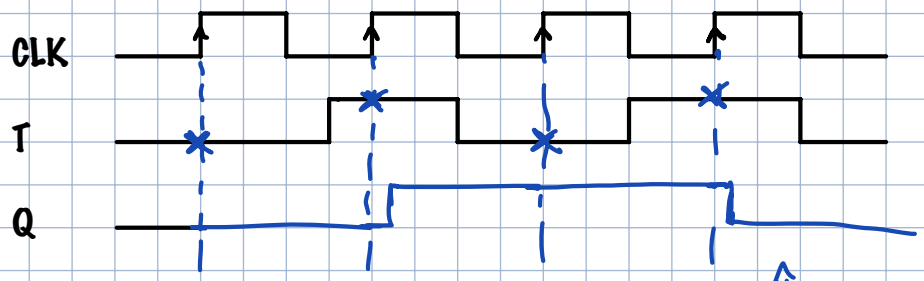


⇒ clear has higher priority than preset

CLK

Characteristic table:

T	Q(t+1)
0	Q(t)
1	Q'(t)

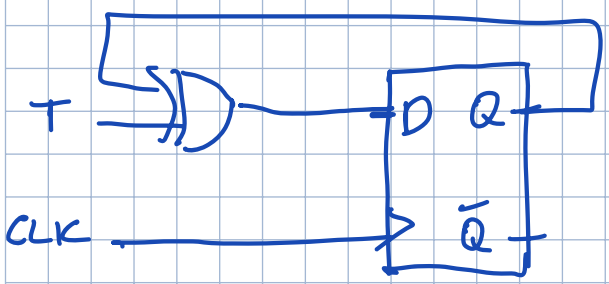


Midterm 11.11.2023 15:00–17:00

compare with D-FF.

Can we make T-FF from D-FF?

T "Toggle" FF (positive edge triggered)



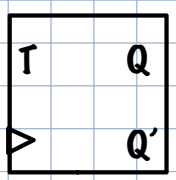
$D = f(T, Q)$ $Q(t+1) = D = f(T, Q)$

T	Q(t)	D
0	0	0
0	1	1
1	0	1
1	1	0

$D = T \oplus Q$

$= \bar{T}Q + T\bar{Q}$

symbol:



VHDL code for T-FF ← inside architecture

```
process(CLK) begin
    if rising_edge(CLK) then
        Q<=D;
    endif;
end process;
D<=Q when T='0' else not Q;
```

D-FF

XOR

Note: Q & D has to be signal because they appear both at the input & output of this code

```
architecture ...
    signal Q,D: std_logic;
    begin
        Qout <= Q;
        ...
```

← declared as output in entity part of the code

Characteristic table:

J K	Q(t+1)	
0 0	Q(t)	no change
0 1	0	reset
1 0	1	set
1 1	Q'(t)	toggle

Can we make JK-FF from D-FF?

We know that Q(t+1)=D

What should be D as a function of inputs and output?

Q/JK

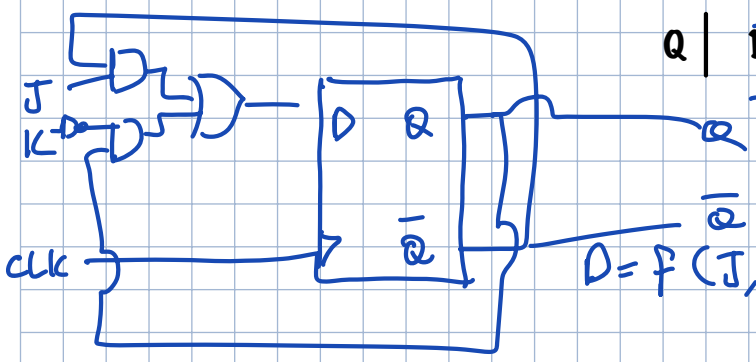
Q \ JK	00	01	11	10
0			1	1
1	1			1

J, K

$J\bar{Q}$

$\bar{K}Q$

JK FF (Jack Kilby)



$D = f(J, K, Q)$, $Q(t+1) = D$

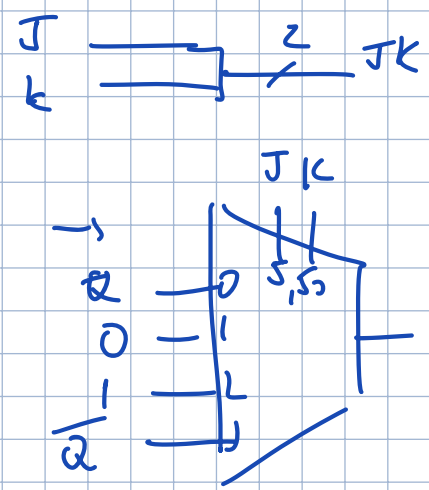
J	K	Q	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

VHDL code for JK-FF (inside architecture)

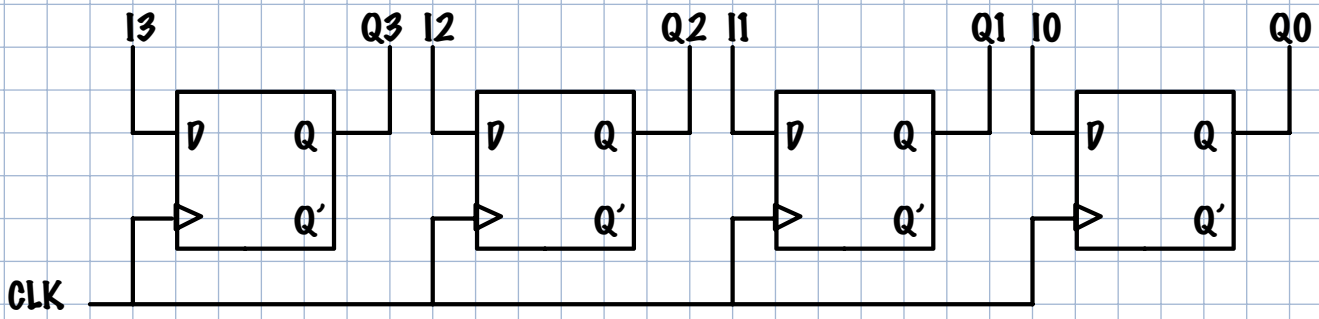
```
process(CLK) begin
    if rising_edge(CLK) then
        Q<=D;
    endif;
end process;
D<=Q when J='0' and K='0' else
   '1' when J='1' and K='0' else
   '0' when J='0' and K='1' else
    not Q;
```

Alternatively, we can use the concatenation operator

```
JK<=J&K;
with JK select
    D<=Q when "00",
       '1' when "10",
       '0' when "01",
       not Q when others;
```



4-bit register



input: I = (I3, I2, I1, I0)
output: Q = (Q3,Q2,Q1,Q0)

How many states? $2^4=16$ (flip-flop's outputs)
Number of input combinations? $2^4=16$
Number of output combinations? $2^4=16$

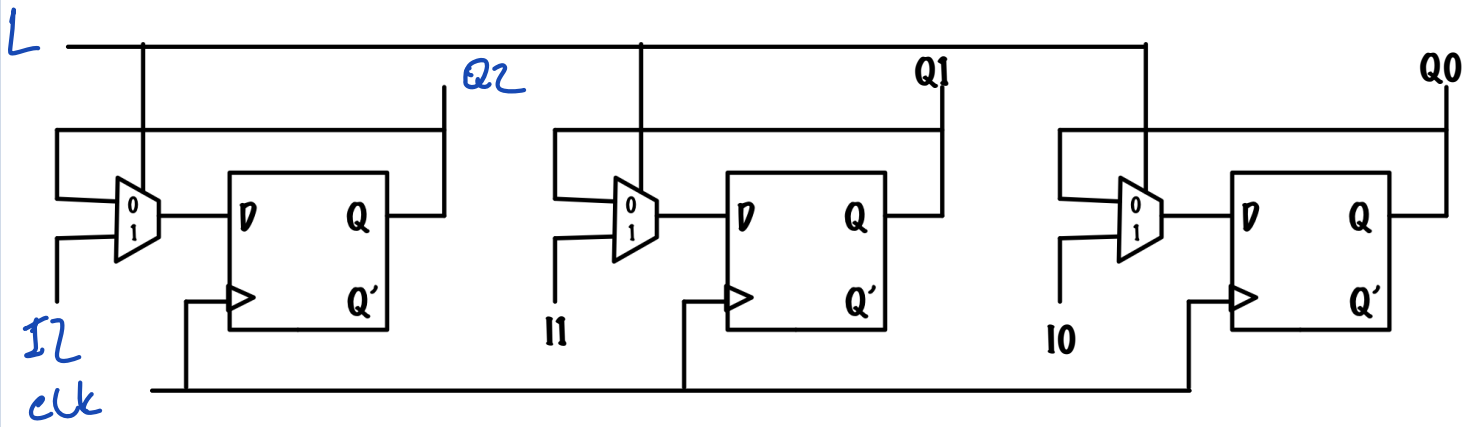
For an n-bit register:
How many states? 2^n
Number of input combinations? 2^n
Number of output combinations? 2^n

Can this register memorize the past?
Output changes (follows input) at every rising edge of CLK)

VHDL code for 4-bit register

```
entity reg4 is
    port(I: in std_logic_vector(3 downto 0);
         CLK: in std_logic;
          Q: out std_logic_vector(3 downto 0));
end reg4;

architecture reg4arch of reg4 is
begin
    process(CLK) begin
        if rising_edge(CLK) then
            Q<=I;
        end if;
    end process;
end reg4arch;
```



L=0 hold current values
L=1 load new input values

VHDL code for 3-bit register with load

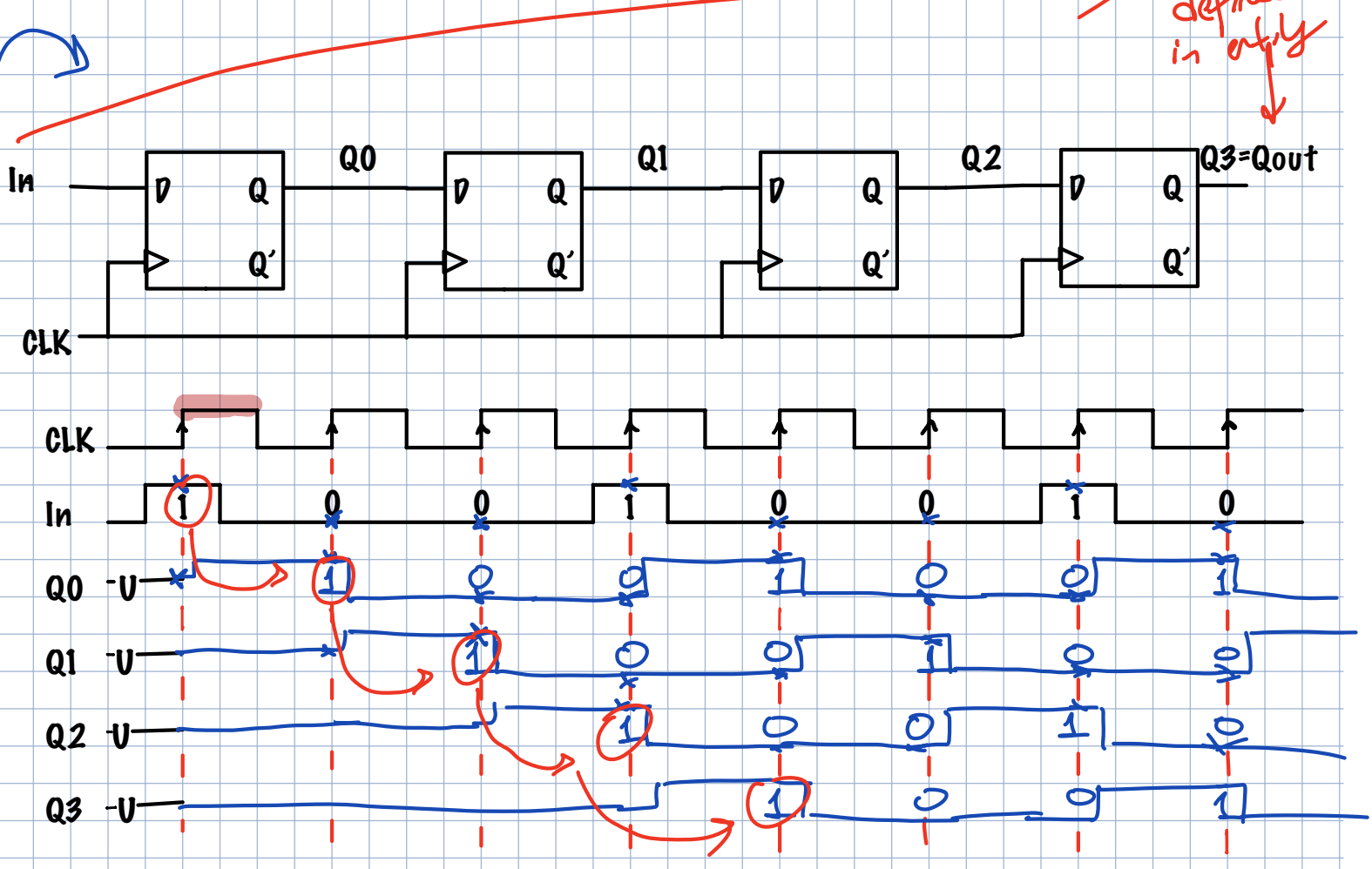
```
process(CLK) begin
    if rising_edge(CLK) then
        if L='1' then
            Q<=I;
        end if;
    end if;
end process;
```

implied memory

Registers

4-bit register

3-bit register with load



4-bit shift register

VHDL code for 4-bit shift register

```
architecture shift_arch of shift4 is
    signal Q: std_logic_vector(3 downto 0);
begin
    process(CLK) begin
    if rising_edge(CLK) then
        Q(3)<=Q(2);
        Q(2)<=Q(1);
        Q(1)<=Q(0);
        Q(0)<=In;
    end if;
    end process;
    Qout<=Q(3);
end shift_arch;
```

$Q_3 Q_2 Q_1 Q_0$

$Q_- = 1100$, $I = 1$

t_+ t_-

ordering here does not matter

Q(3) <= In
Q(2) <= Q(1)
Q(1) <= Q(0)
Q(0) <= Q(2)

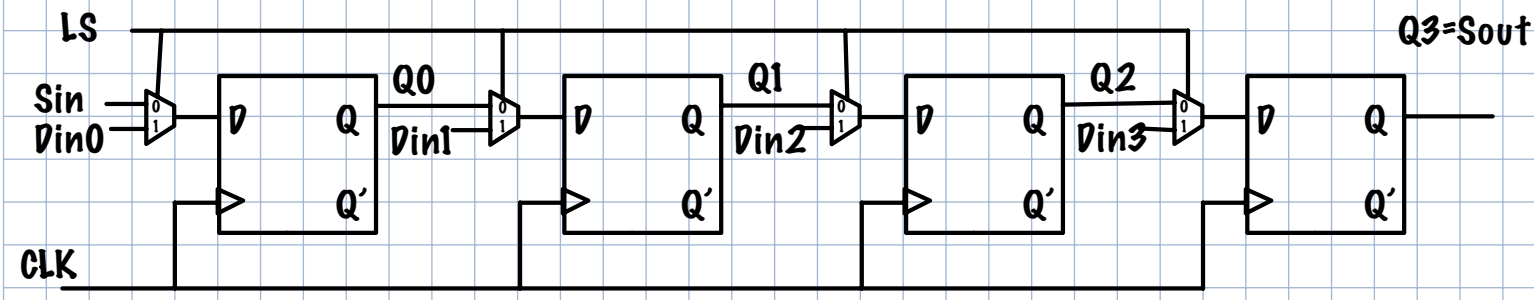
Alternative VHDL code for 4-bit shift register

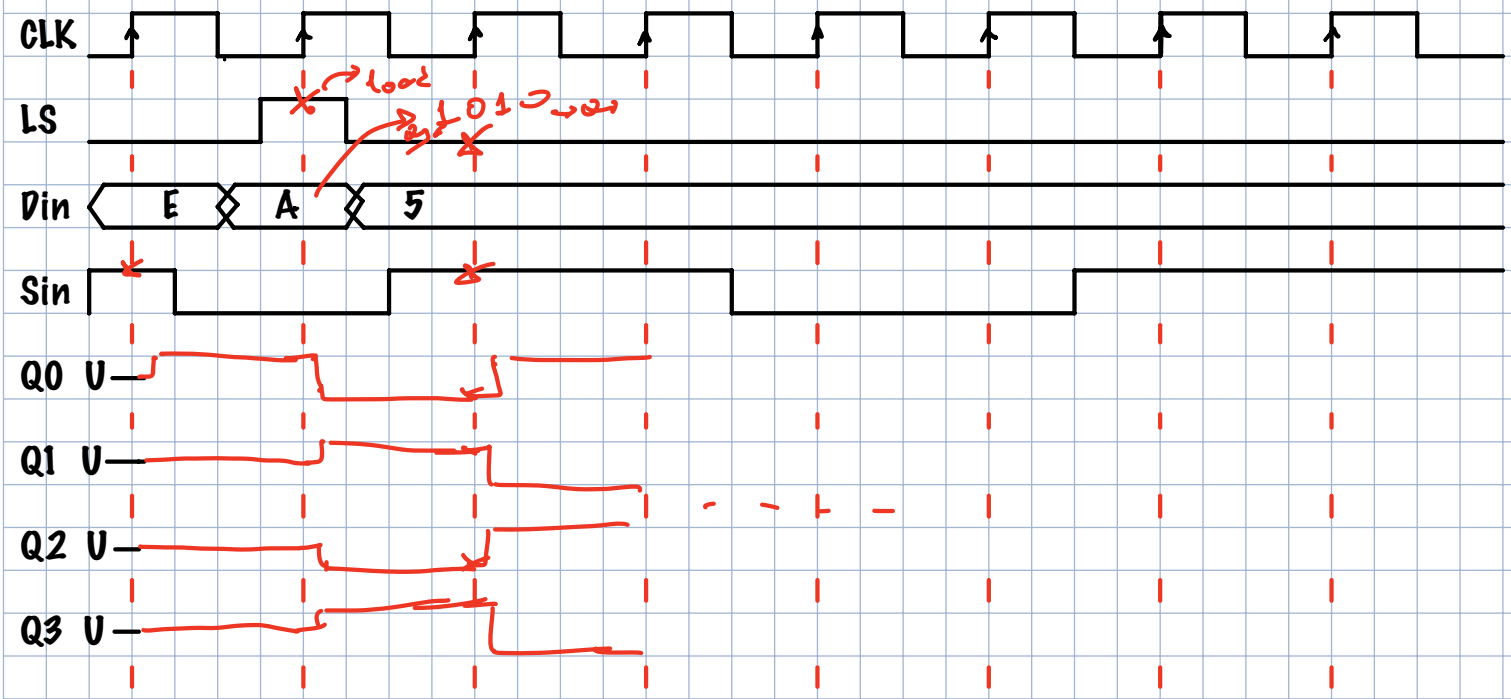
```
architecture shift_arch of shift4 is
    signal Q: std_logic_vector(3 downto 0);
begin
    process(CLK) begin
    if rising_edge(CLK) then
        Q<=Q(2 downto 0)&In;
    end if;
    end process;
    Qout<=Q(3);
end shift_arch;
```

What does the following VHDL code represent?

```
entity mystery is
    port(Din: in std_logic_vector(3 downto 0);
         CLK: in std_logic;
         LS,Sin: in std_logic;
         Sout: out std_logic;
         Qout: out std_logic_vector(3 downto 0);
end mystery;

architecture mystery_arch of mystery is
    signal Q: std_logic_vector(3 downto 0);
begin
    process(CLK) begin
        if rising_edge(CLK) then
            if LS='1' then
                Q<=Din;
            else
                Q<=Q(2 downto 0)&Sin;
            end if;
        end if;
    end process;
    Qout<=Q;
    Sout<=Q(3)
end mystery_arch;
```


LS
Q3=Sout
Sin
Din0
D
Q
Q'
Q0
Din1
Q1
Din2
Q2
Din3
CLK


CLK
LS
load
shift 0 1 ...
Din
E
A
5
Sin
Q0 U
Q1 U
Q2 U
Q3 U