— more formal treatment/introduction to seq. circuits:

LECTURE #5

(~1.5 lectures)

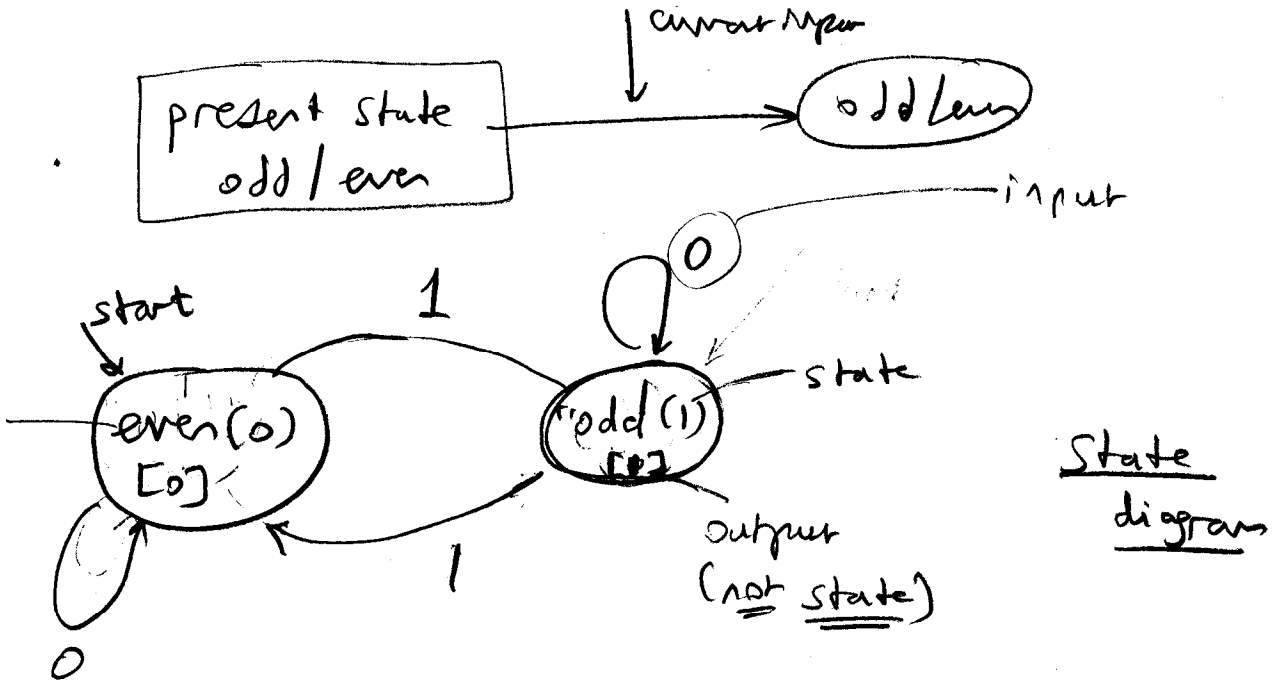
Ex1 Build a system that tells you whether I have seen an odd or even # 1's in input stream.

* Key idea: Recursion:

Assume that I have seen an odd # 1's so far. Then, if I see a 1, then.

~~If~~ next state := even

If I see 0, next state = odd.



State diagram

Before:

Input Variable: X (1 bit) = what I see currently.

State: Q (1 bit) = { 1 — I've seen an odd # bits so far; 0 — — an even # bits so far.

output: Y (1 bit) = { 1 — — — — — — — — ; 0

In this example, $Y = Q$.

~~Two types of sequential machines: Mealy, and Moore.~~

(next-state) transition equations

	"Q"	"Q+"
X	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

list all possibilities

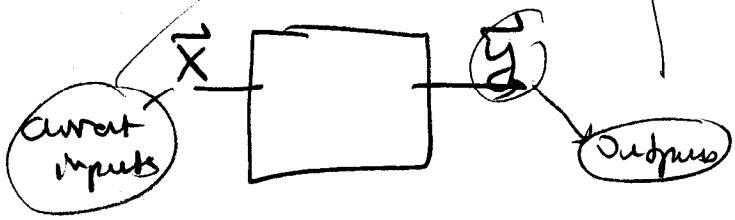
State table (transition)

We see that $Q^+ = X \oplus Q$ — "next state equation"

Q^+ ↑ next state

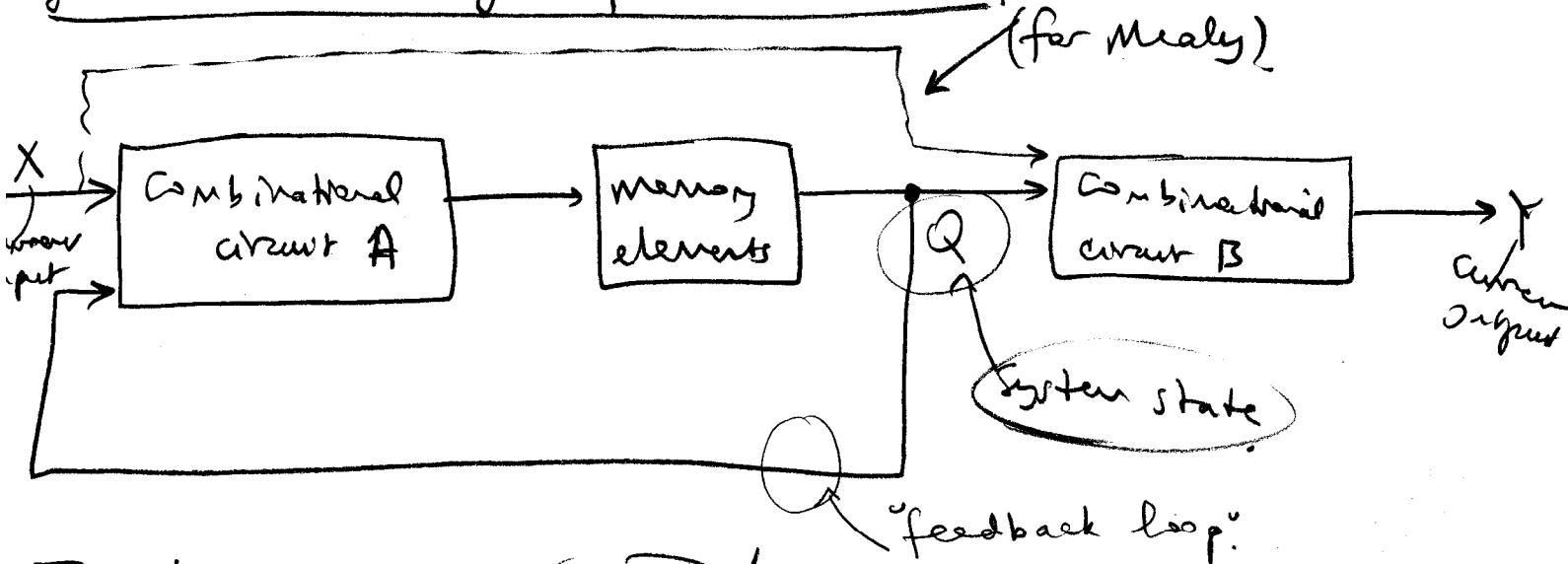
$Y = Q$ — "current output equation"

In general, in a logic circuit, if



the outputs are completely determined by current inputs then, circuit is <u>combinational</u> otherwise, the circuit is <u>sequential</u>.

<u>general structure of sequential circuits</u>:



Two types of ~~FSM~~ (machine)/seq. circuits

(1) <u>MOORE</u> : Output Y is completely determined by <u>system state</u> Q.

(2) <u>MEALY</u> : ~~output Y is otherwise (i.e. output~~ output Y is ~~not completely~~ determined by state Q <u>and</u> current input X.

Ex1: Back to (odd/even) example.

(Ask class);

MOORE machine because Y is determined by system Q.

(Y ⊆ Q but special case of moore machine. not necessary)

Ex 2 ~~I could have~~

Let's examine the same problem: Say whether we've seen an odd/even # 1's so far.

- Above, ~~we~~ (Ex1), we ① looked at current input, ~~&~~ (transitioned to a new state ~~&~~ THEN

② reported the output to the outside world.

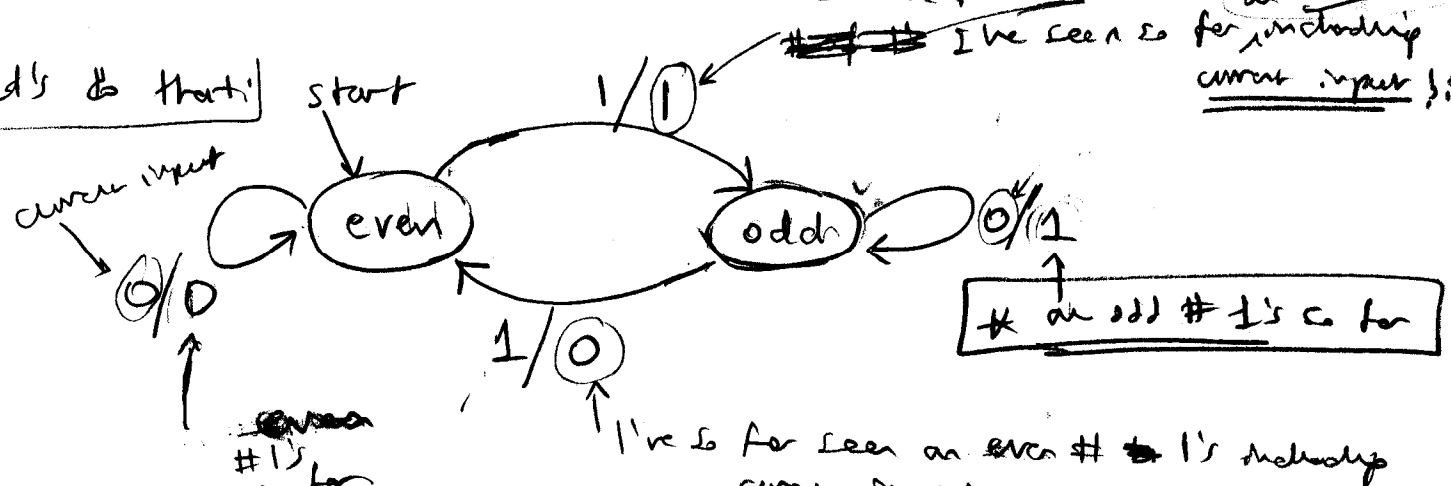
But we can ~~also~~ also (instead):

① look at current input, ~~&~~

(Report ~~&~~ the output to the outside world

THEN ② Transition to ~~a~~ the new state.

Let's do that! start



MEALY machine because output Y depends on X and Q.

[Ask class]. Write (1) next-state equation

$$Q^+ = X \oplus Q$$

State table

X	Q	Q^+	Y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

(2) current output equation

$Y = 1$ iff $(Q=0$ and $X=1)$

OR $(Q=1$ and $X=0)$

$$Y = X \cdot Q' + X'Q = X \oplus Q$$

Let's contrast Ex1 (Moore) and Ex2 (Mealy) which we used to solve same design problem.

* Fundamentally, what is the difference between these two designs?

— Mathematically, [redacted], next-state equations appear the same; (But) the output is different [redacted]

MOORE:

$Y = Q_{Moore}$

MEALY

$Y = X \oplus Q_{Mealy}$

clearly these Q's represent different things

MOORE DESIGN: Q_{Moore} : we're, already transitioned to a new state & then report output

MEALY DESIGN: Q_{Mealy} : - we report output & then transition to a new state.

Clearly, timing is crucial. in this difference. ↑

- Now, we go further down into: timing.
- There are 2 types of design paradigms to design either

Machine: Synchronous or asynchronous

1) uses a clock:

a global signal with regular transitions & a period:

T (clock period)

[1)

T (clock period)

↑ (Also possible)]

2) multiple clocks also possible: sych. r. design uses at least 2 clocks; AND

2 Every memory element is clocked

[otherwise]

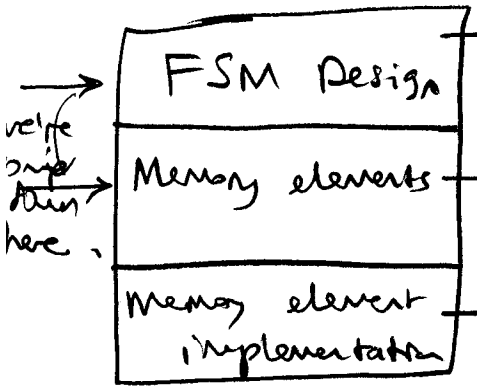
(e.g. there is at least one memory element not clocked.)

Sequential machines:

	MOORE	MEALY
SYNC		
ASYNC		

→ all possible.

In order to understand timing of sequential circuits, we will go down to a lower level of abstraction and describe the memory elements at an intermediate level of abstraction.



---

* In this course we focus exclusively on synchronous sequential machines
  - vast majority of designs in industry.

Synchronous:

- as design gets complex, synch. easier to design and debug

[- asynchronous design: introduces asynchronicity very subtle bugs into design. - must be very careful.]

Now, let us go back to our even/odd FSM example and show how to implement it as:

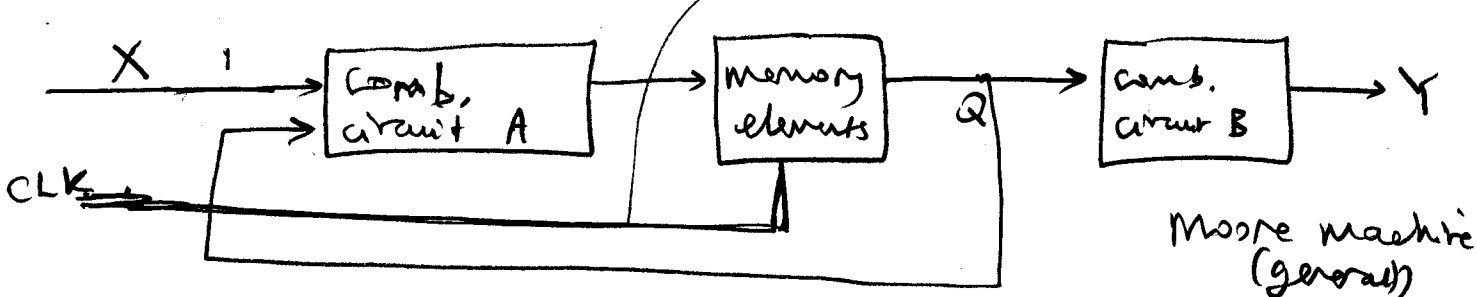
(a) synchronous Moore machine

(b) " Mealy machine

(a): Synchronous Moore machine:

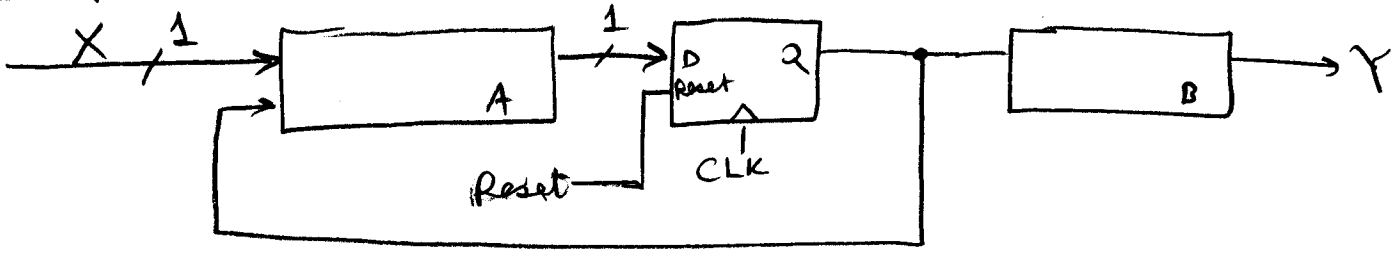
and state assignment: "even" = 0 (state bit)
"odd" = 1

We need 1 bit to hold the system state: Let's use a D-FF to hold this state.



Moore machine (general)

This example:



- Must design the comb. circuits A and B

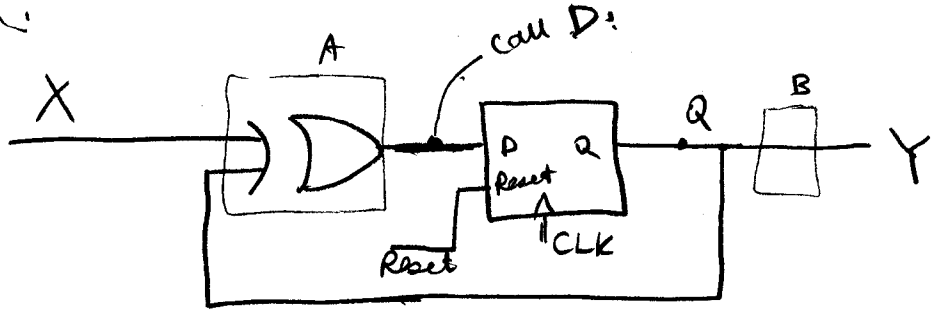
A: should implement the "next-state logic". — (look at state tables)

B: " " " "output logic". — (look at output equations)

A: $Q^+ = X \oplus Q$

B: $Y = Q$.

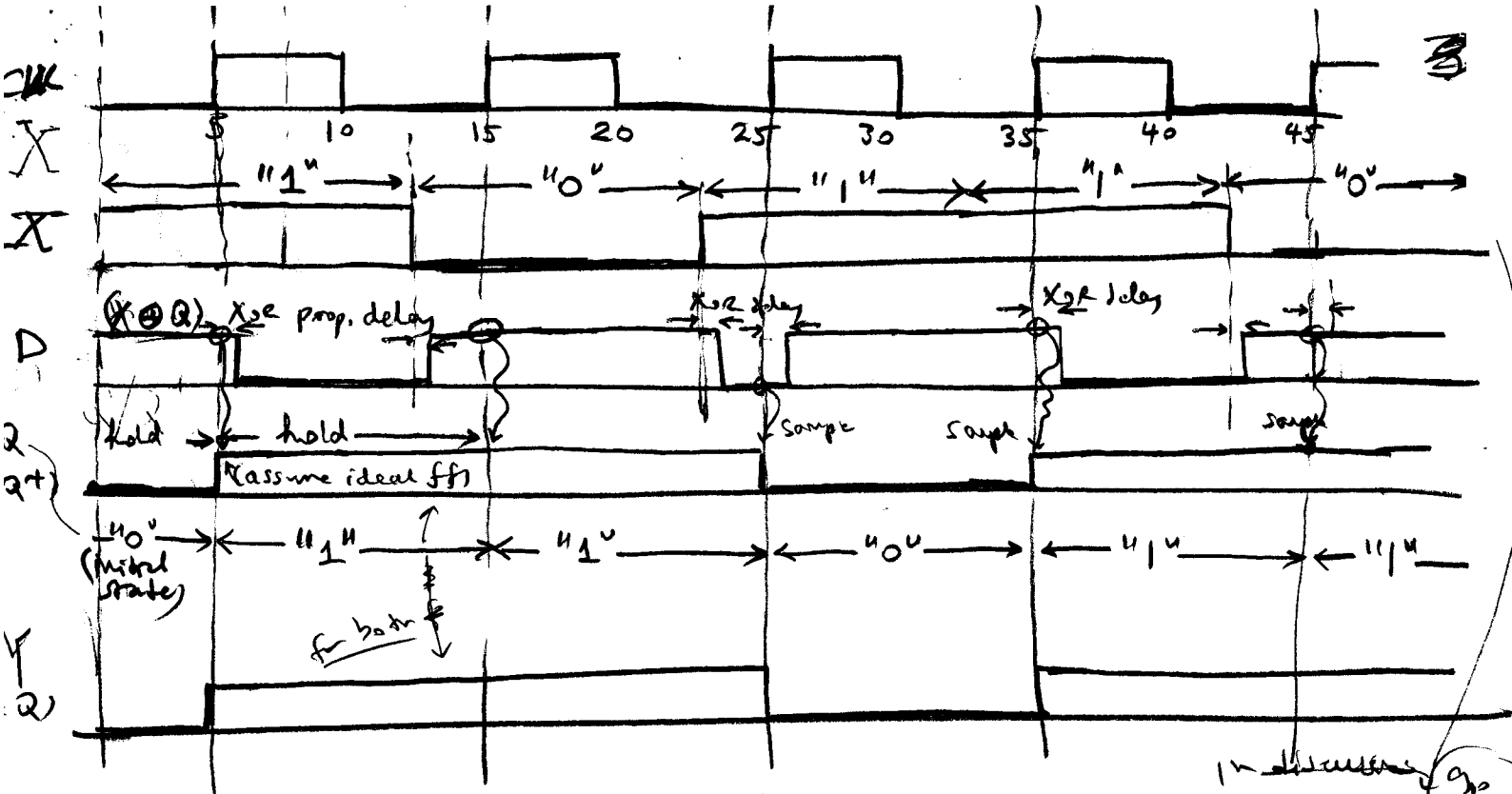
Then:



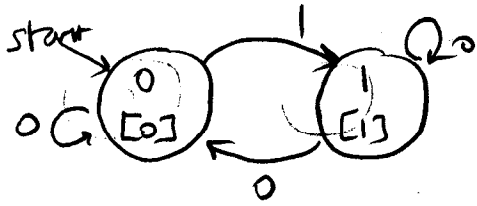
(Implementation of the ~~[odd/even]~~ parity checker example as a Synchr Moore machine using PET-FFs.)

- Very important to understand the timing behavior of this ~~[illegible]~~ implementation. Does it really implement what we want at the upper layer?

Let's examine the timing diagram (next page)



- Assume Q has ~~[redacted]~~ been initialized to zero (~~[redacted]~~ in discussion, we'll ~~[redacted]~~ using Reset.



- The input sequence is: 1, 0, 1, 1, 0.

- First, let's determine what Q and Y should be:

discrete time	X (input)	Q (state)	Y (output)
0	1	0	0
1	0	1	1
2	1	1	1
3	1	0	0
4	0	1	1
5	—	1	1

initial state ("every reset machine")

copy

initial state

output we're interested in.

"by the end of discrete time = 1, I've seen one 1."

*need this 6th time slot (even though 5 input bits!) *

- Now, lets fill out timing diagram: (based on ff behavior)
- Now, let us check whether the timing diagram ~~go to the~~ is correct.

Key aspects to notice:

① Note that the output becomes ready after the active clock edge.

② Note how important the XOR propagation delay is: That delay prevents the new value of Q (i.e. Q^+) to race through the circuit and update again. — That delay is crucial for the design to work.

③ Note how important it is for the input $\hat{X}$ to settle to its correct (next) value before the active clock edge arrives. X has to transition to its correct value before the active clock edge.

In general, in a ff-based design, for a Moore machine ~~Note~~ the following are the sequence of events:

1) Inputs have to get ready, and make sure effects have propagated to data input of ff.

2) Active clock edge arrives – (~~next state is obtained~~ state is sampled)

3) Two things happen concurrently:

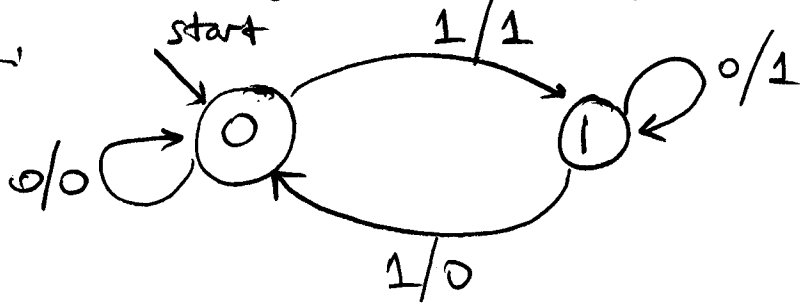
(a) output is being computed (through comb. circuit B)

(b) the next state is being computed through circuit [INPUT changes to its next value here]

– Clearly, timing is very important. — before we go into a detailed analysis of timing, let's examine the synchronous Mealy machine implementation of the same problem.

(b) Synchronous Mealy machine implementation

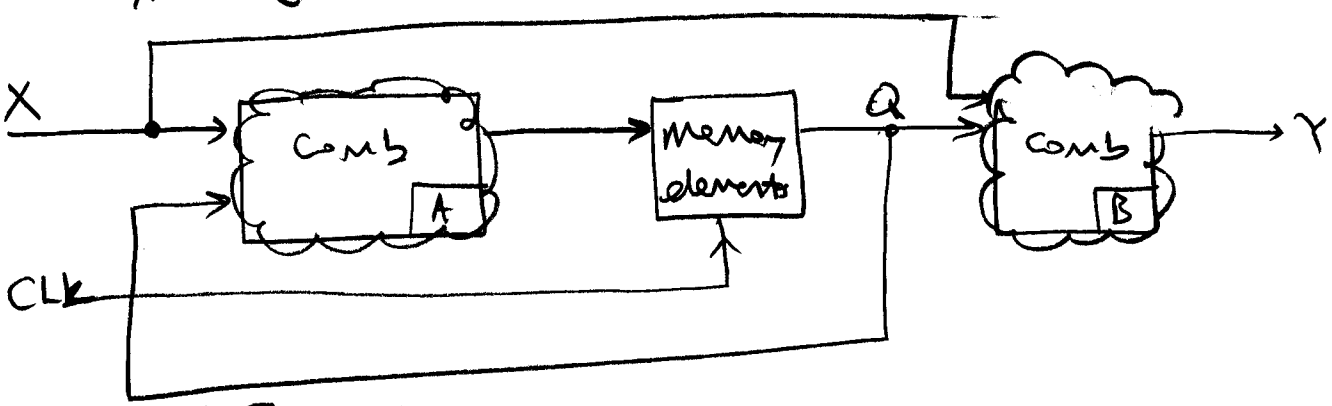
[Recall:



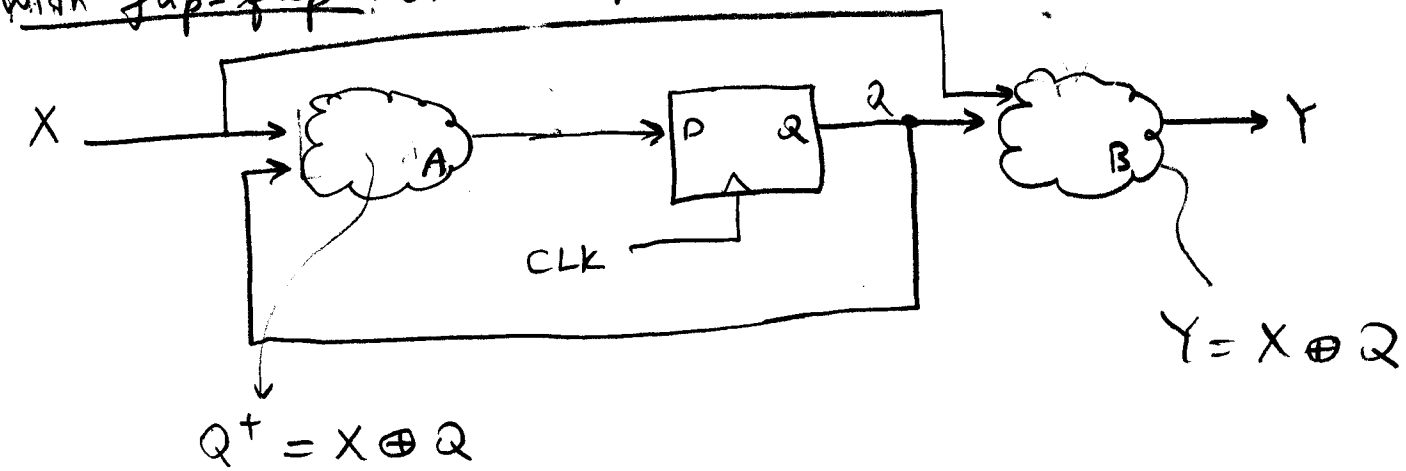
and: $Q^+ = X \oplus Q$

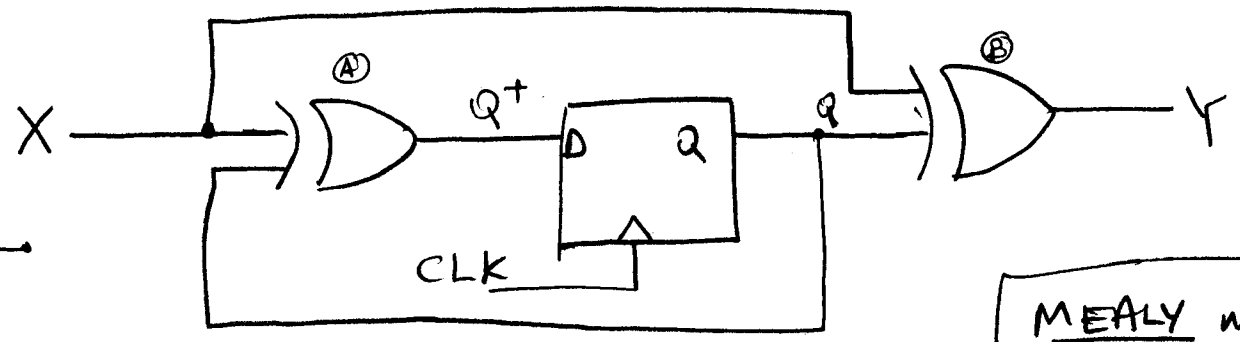
and $Y = X \oplus Q$.]

general synchronous Mealy machine:



with flip-flop: (two examples)





MEALY machine
(PET-FF design for parity checker)

Let us a perform a similar analysis

- First, the behavior we expect:

(Input sequence: Assume same 1, 0, 1, 1, 0.)

discrete time	X	Q	Y
0	1	0	1
1	0	1	1
2	1	1	0
3	1	0	1
4	0	1	1

initial state

note: initial state not reported (unlike Moore)

$(=X \oplus Q)$

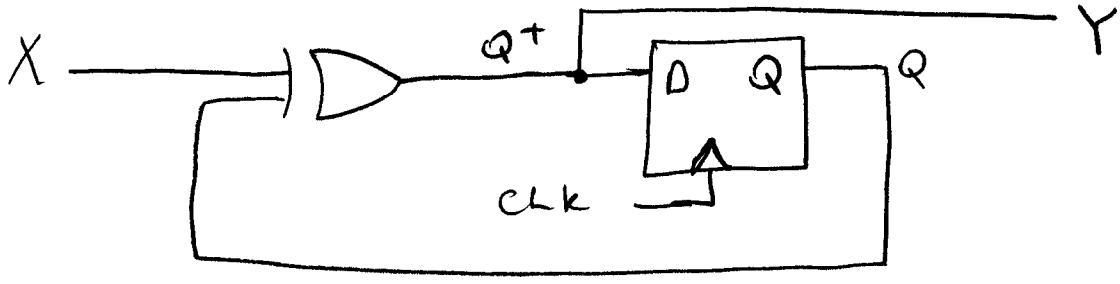
means: "Including the input at discrete time 0, I've seen one 1 so far".

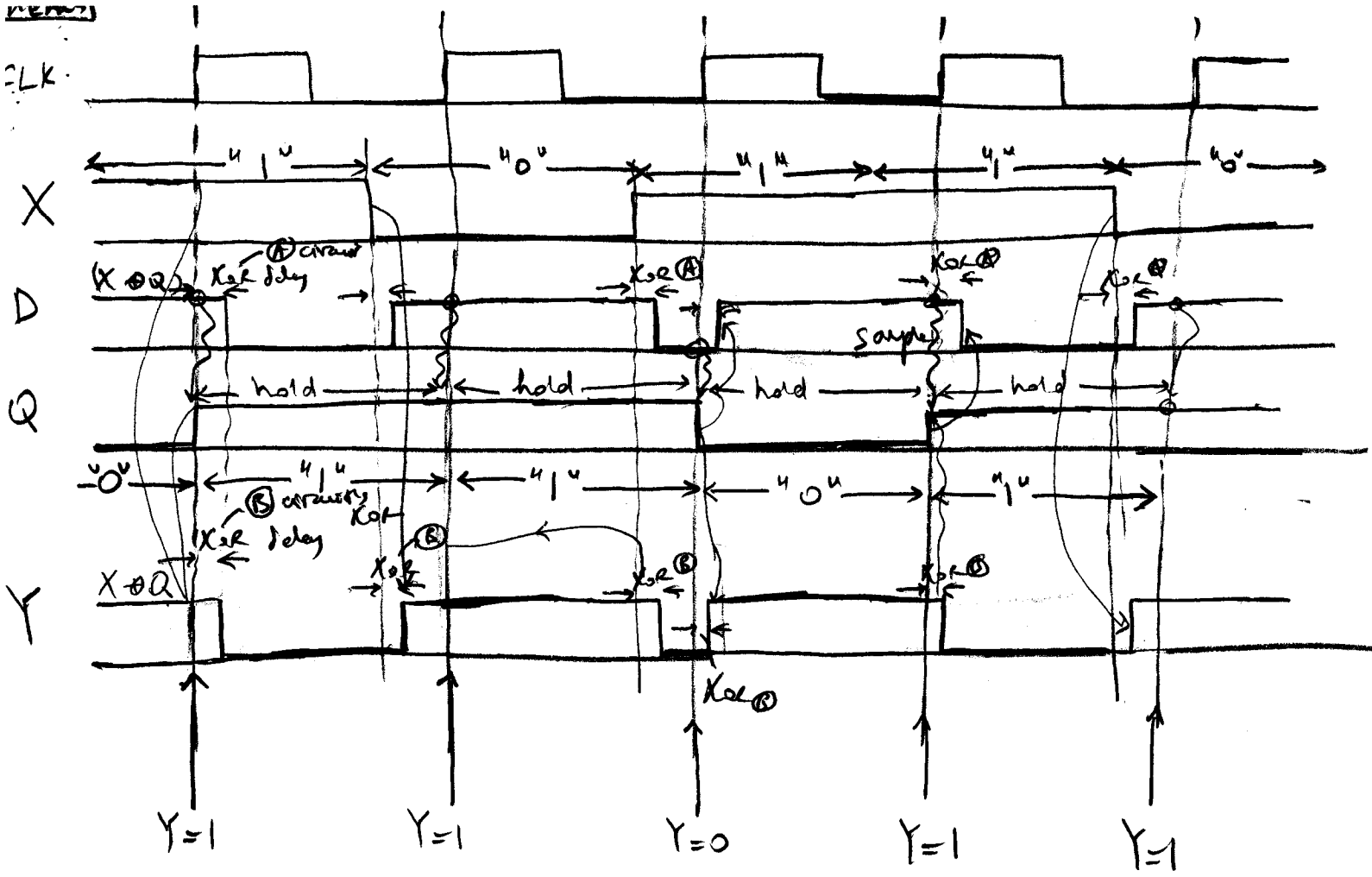
relevant output.

finishes in 5 discrete time units (same as input length)

$(Y = Q^+ = X \oplus Q)$
↑ using

→ OR: optimize combinational circuits A & B together.





Remarks:

1) Clearly, the analysis needs far more care than the Moore machine.

2) Must interpret Y correctly. The values of Y that we consider as outputs of the machine are the values of Y at the active clock edge.

[— A good idea is to place a ff at the output with the same ~~clocking~~ active clock edge so that the output ff can hold the sampled output. This effectively turns the machine into a Moore-type machine but with a 2-bit state vector; so the original 1-bit Moore machine would have been a better

choice than to do that. — So, if the output of this circuit will be fed to another synchronous circuit, it will be better to use a Moore design.]

(3) Note that the timing diagram of the D signal is the same as the Moore machine since they use the same next-state logic eqns and the X input signal is the same (as a timing waveform) in both cases.

4) Note that the Q waveform is same as the Moore machine. (~~Today~~ Both designs sample Q at the clock edge.)

[∴ X, CLK same ; D, Q same in both cases]

5) Output Y is the only difference.

— Basically, the Y_{Moore} is the sampled and held version of the Y_{Mealy}.

— Y_{Moore} a more reliable signal to use in ~~Better to choose~~ subsequent sub circuits.

In general,

	MOORE	MEALY
Advantages	1) The output signal is synchronous: safe to use as input to subsequent synchronous circuits. 2) easier to design, understand, debug.	1) since a more general machine, can solve same problem w/ fewer states.
Disadvantages	1) More state bits may be necessary than Mealy machine.	1) output signal is asynchronous if input X is asynchronous

Design guidelines:

1- Usually good to implement the problem by whichever one is more natural for the problem at end & come up w/ a design (Moore or Mealy).

2 - Then if you have a Mealy machine & would actually want a Moore machine:

Thm: Every Mealy machine can be "converted" into a Moore machine.

Method:

Method of conversion:

- First, write explicit equations for output in terms of initial state & input. Then redefine "state" appropriately to create a Moore machine.

Ex | In our example:

Mealy machine: $Y = Q \oplus X$

$$Q^+ = Q \oplus X$$

Write in discrete-time:

$$Y_0 = \boxed{Q_0 \oplus X_0} = Q_0^{Moore}$$

$$Y_1 = Q_1 \oplus X_1 = \boxed{Q_0 \oplus X_0 \oplus X_1} = Q_1^{Moore}$$

$$Y_2 = Q_2 \oplus X_2 = \boxed{Q_0 \oplus X_0 \oplus X_1 \oplus X_2} = Q_2^{Moore}$$

$$Q_1 = Q_0 \oplus X_0$$

$$Q_2 = Q_1 \oplus X_1$$

Write next-state eq'ns for newly defined state:

$$Q_1^{Moore} = Q_0^{Moore} \oplus X_1$$

$$Q_2^{Moore} = Q_1^{Moore} \oplus X_2$$

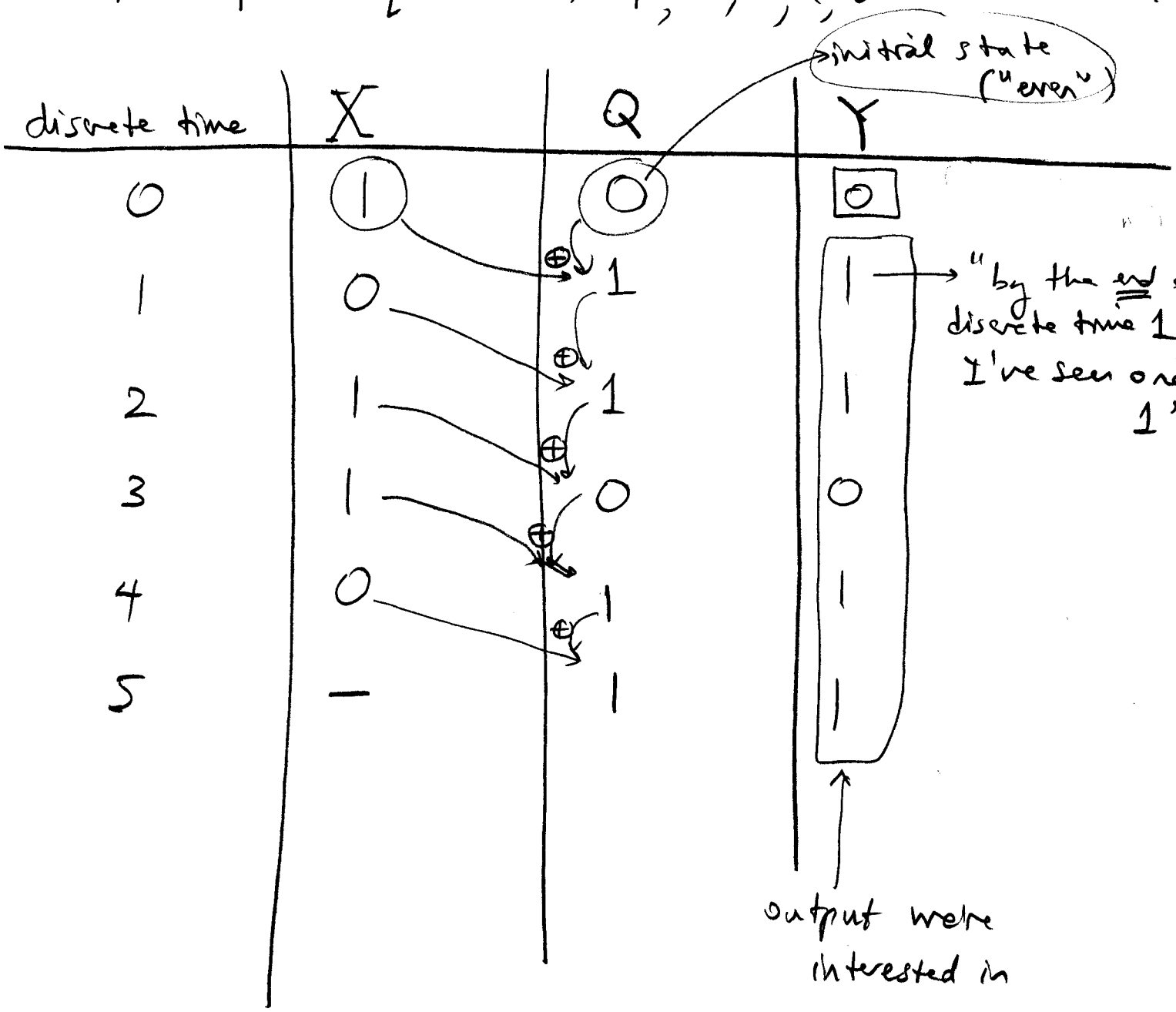
$$\Rightarrow Q^+_{Moore} = Q_{Moore} \oplus X. \quad \text{(which is the Moore implementation)}$$

- Above example was simple but you can use this method in general (see hw).

MOORE DESIG
(PARITY-CHECKE

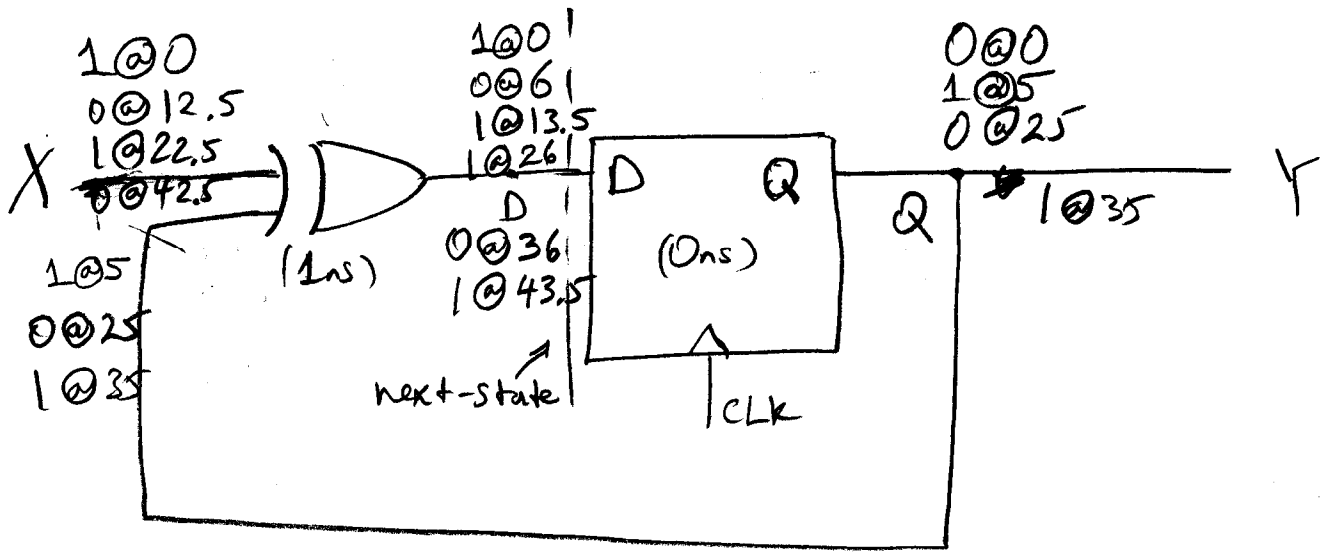
· Let us examine in discrete time domain:

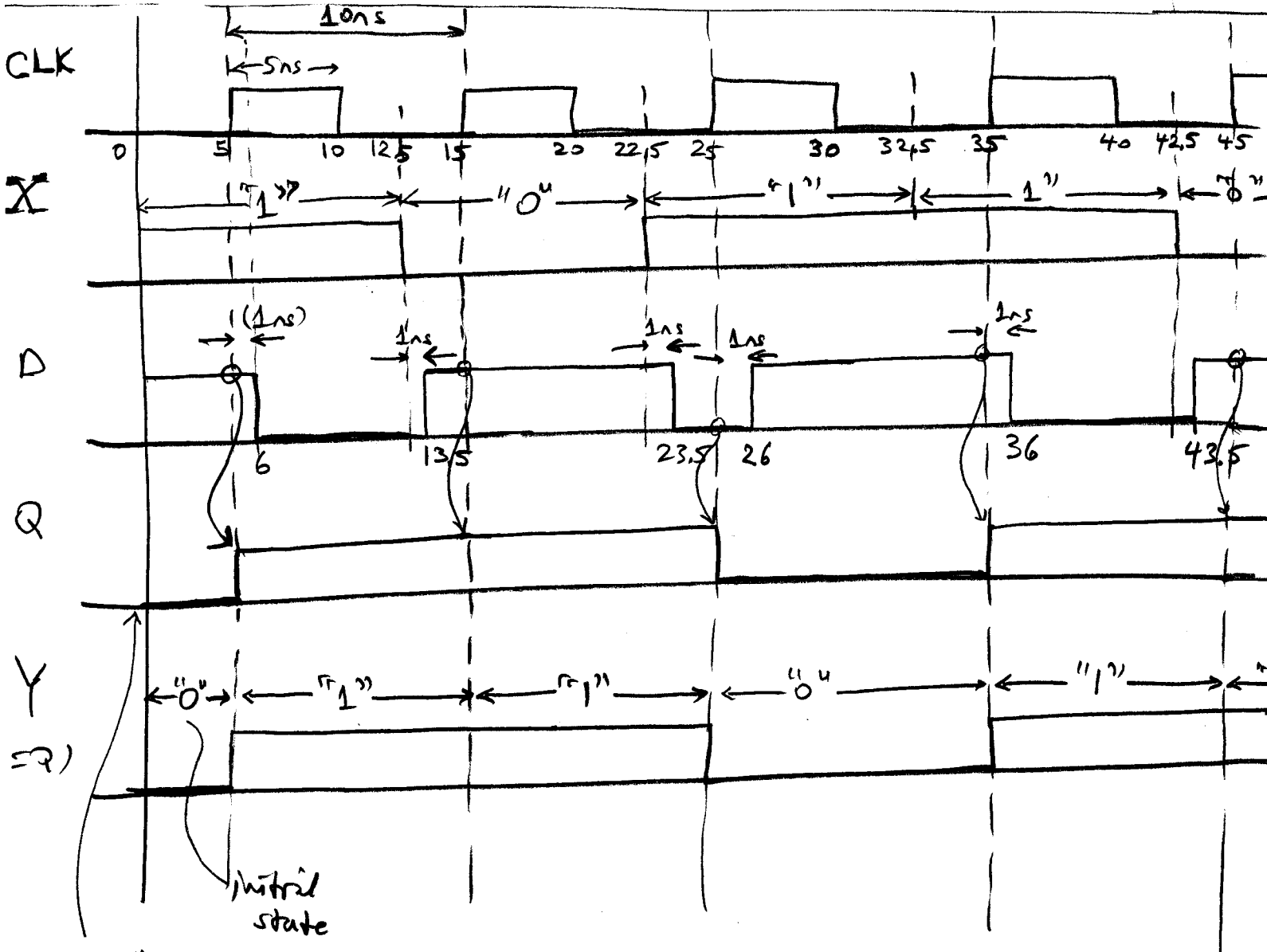
The input sequence is: 1, 0, 1, 1, 0



P.5

(Moore) parity checker





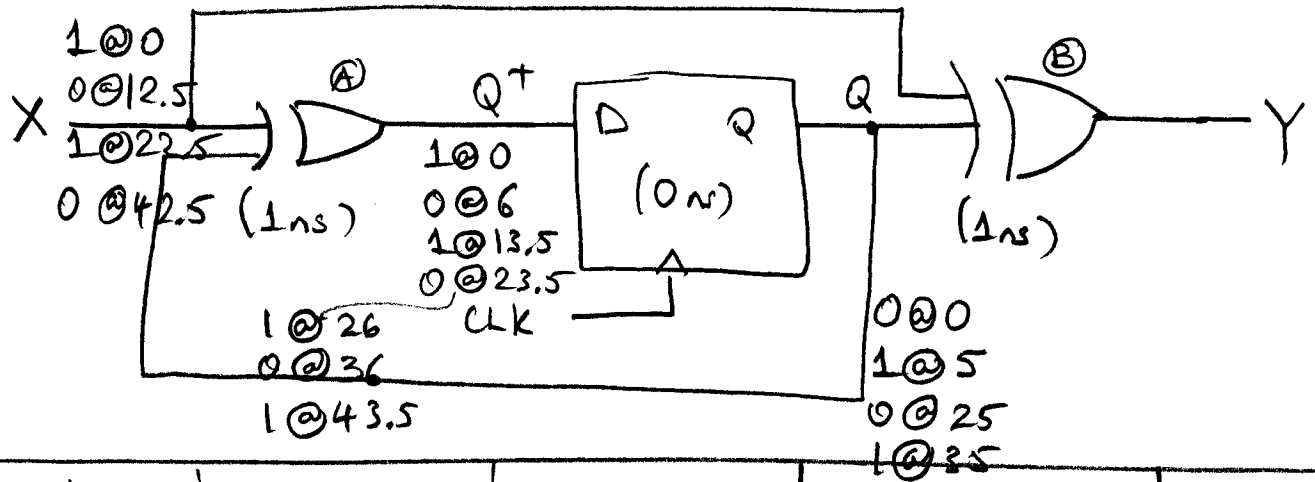
MEALY DESIG
(MACHINE :
PARITY-CHEC

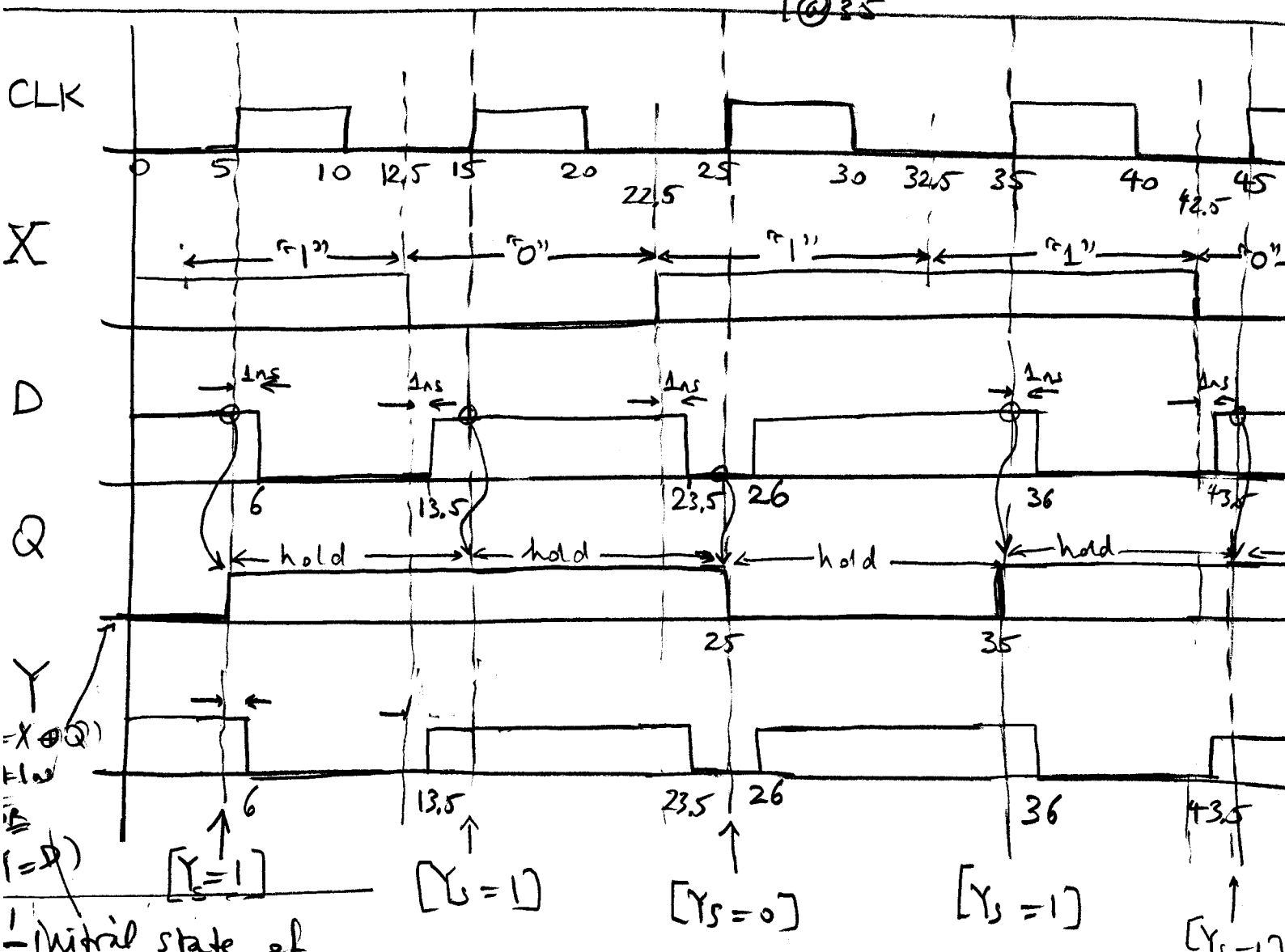
• First, write down how the machine should operate (expected behavior). We simplify by using discrete time.

discrete time	X	Q (initial state)	Y_s (= X ⊕ Q)
0	1	0	1 → "Knowing input at t=0, I've seen one 1 so far."
1	0	1	1
2	1	1	0
3	1	0	1
4	0	1	1

Y_s : stands for <u>sampled</u> Y.

(see next page for the continuous Y.)





- Initial state of Q is 0 (since initially, the machine has seen zero 1's ⇒ an even # of 1's.)

= Assume that the machine has been in this state for a long time