DIGITAL SYSTEMS

MOISE GABRIEL

SHEET 5

TT 2019

① XOR gate: $z = a \oplus b$

a	b	z
0	0	0
0	1	1
1	0	1
1	1	1

(a) We will show that $\oplus$ is associative and commutative with the truth table:

Commutativity:

$$a \oplus b = b \oplus a$$

a	b	$a \oplus b$	$b \oplus a$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Associativity: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$

a	b	c	$a \oplus b$	$b \oplus c$	$(a \oplus b) \oplus c$	$a \oplus (b \oplus c)$
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	0	0	0
1	0	0	1	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	0	0
1	1	1	0	0	1	1

According to the truth tables, $\oplus$ is both associative and commutative.

It has an identity element e if and only if

$$e \oplus a = a \oplus e = a, \text{ for all } a \in \{0,1\}$$

I $a=0 \Rightarrow e \oplus 0 = 0 \oplus e = 0 \Rightarrow e = 0$ (because $0 \oplus 1 = 1$)

II $a=1 \Rightarrow e \oplus 1 = 1 \oplus a = 1 \Rightarrow e = 0$ (because $1 \oplus 1 = 0$)

$\therefore \Rightarrow \boxed{e=0}$ is the identity element of XOR

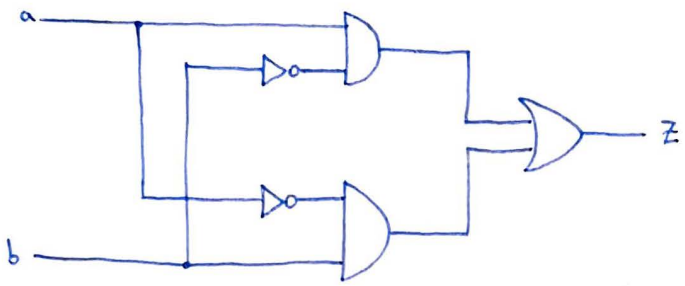
(b) The operation XOR can be interpreted this way: $a \oplus b$ is 1 iff exactly one of the values is 1. This has the formula

$$a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$$

and it has the same truth table

a	b	$(a \wedge \neg b)$		$\vee$	$(\neg a \wedge b)$	
0	0	0	1	0	1	0
0	1	0	0	1	1	1
1	0	1	1	1	0	0
1	1	0	0	0	0	0

Then, we can build a XOR gate from a 2-input OR gate, two 2-input AND gates and two inverters this way:



(c) We will start from our initial formula

$$a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$$

and we will apply the distributivity of disjunction over conjunction ($P \vee (Q \wedge R) \vdash (P \vee Q) \wedge (P \vee R)$)

$$a \oplus b = \big((a \wedge \neg b) \vee \neg a\big) \wedge \big((a \wedge \neg b) \vee b\big)$$

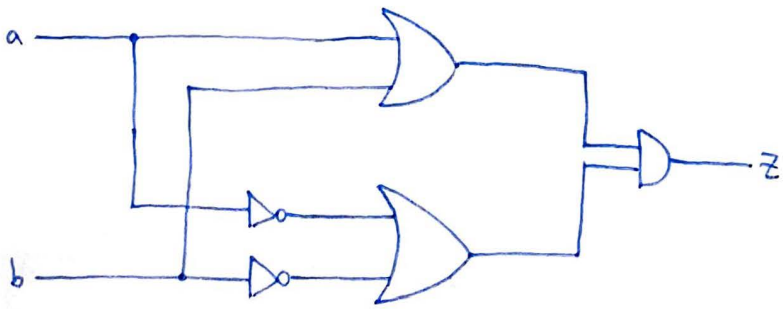
and we apply it again for every bracket:

$$a \oplus b = \Big(\big((a \vee \neg a) \wedge (\neg b \vee \neg a)\big) \wedge \big((a \vee b) \wedge (\neg b \vee b)\big)\Big)$$

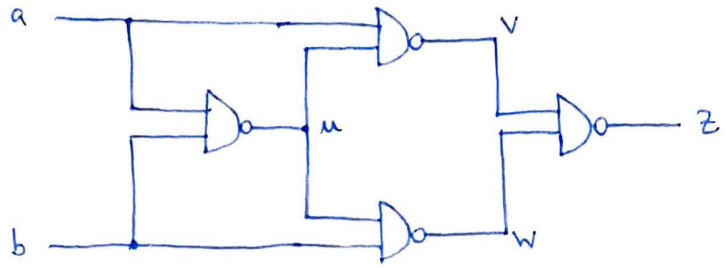
Since $a \vee \neg a = 1$ and $b \vee \neg b = 1$ (law of the Excluded Middle), we remain with:

$$a \oplus b = (\neg b \vee \neg a) \wedge (a \vee b)$$

So, we can build the XOR gate with two 2-input OR gates, one 2-input AND gate and two inverters this way:



(d) We want to show that the following circuit also computes XOR:



To do that, we'll first recall the NAND truth table:

P	Q	NAND(P,Q)
0	0	1
0	1	1
1	0	1
1	1	0

And then we will have (from the circuit) to calculate

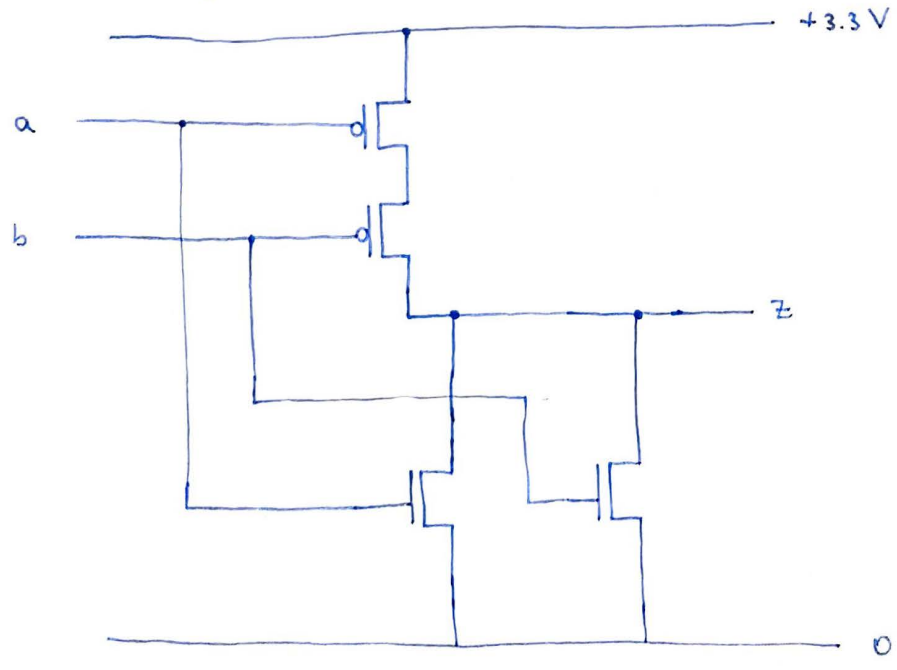
$Z = NAND(V, W)$, where $V = NAND(u, a)$ and $u = NAND(a, b)$

$W = NAND(u, b)$

a	b	u	V	W	Z
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	0

which is the same as for XOR.

(2) (a) The CMOS implementation of a NOR gate is

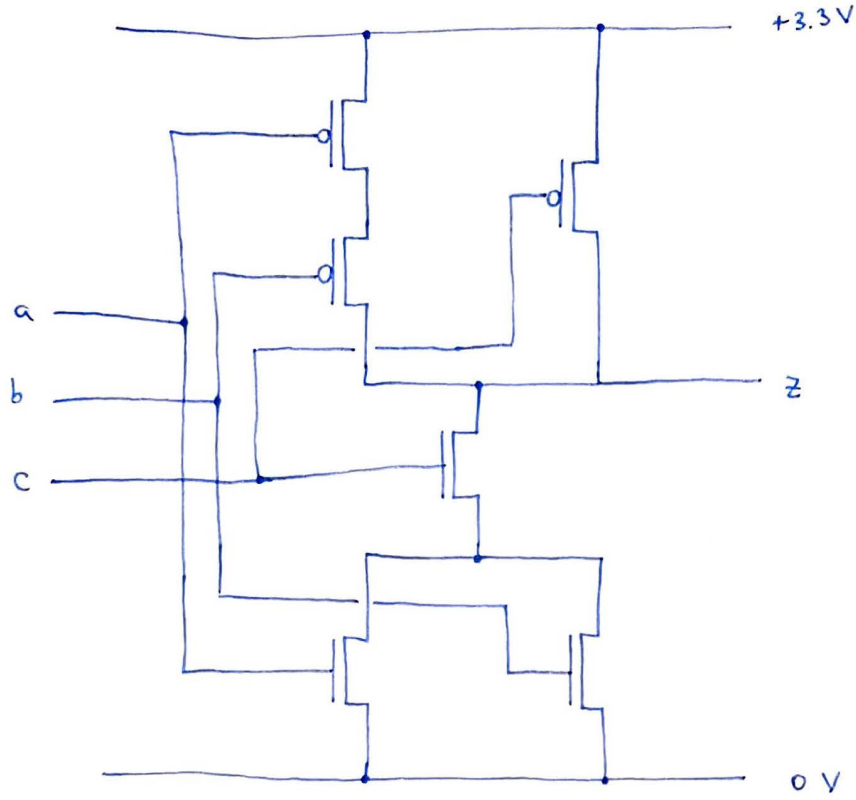


a	b	z
0	0	1
0	1	0
1	0	0
1	1	0

When a and b are 0, both p-type transistors conduct ⇒ z is connected to the Vdd. When either of a or b is 1, the p-type transistors don't conduct and z is connected to the ground.

(b) We want to design a gate for

$$W = \neg((a \vee b) \wedge c) = \neg(a \vee b) \vee \neg c = (\neg a \wedge \neg b) \vee \neg c$$



a	b	c	¬((a∨b)∧c)		
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	0	1	1
1	0	0	1	1	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	0	1	1

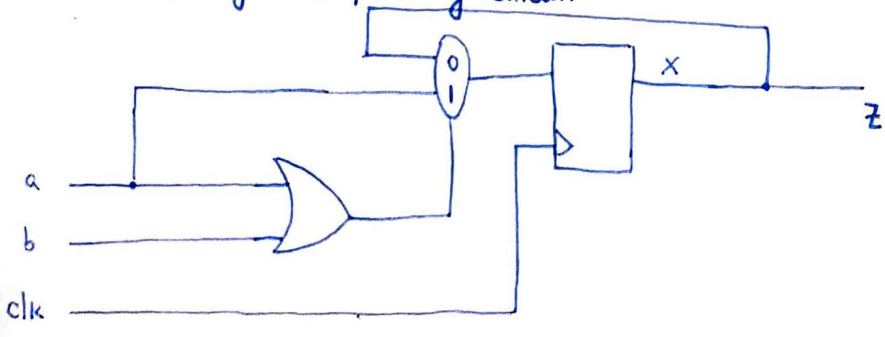
(c) From (a) and (b) we can deduce that a connection of type $\neg a \wedge \neg b$ corresponds to a series connection of two p-type transistors and a parallel one of two n-types and a connection of type $\neg a \vee \neg b$ corresponds to a parallel connection of two p-type and a series one of two n-types (we work with $\neg a$ and $\neg b$ instead of a and b!).

③ (a) if a = 1 at a clock edge, then the output z goes from 0 to 1 and it remains like that until b = 1 at a clock edge.

My interpretation is the following:

a_t	b_t	z_{t+1}	
0	0	z_t	→ if a and b are not on, we don't change the state
0	1	0	→ if z_t was 0, it stays this way, otherwise it goes to 0 from 1 ($= a_t$)
1	0	1	→ if z_t was 1, it stays this way, otherwise it goes to 1 from 0 ($= a_t$)
1	1	1	→ I chose if $a_t = b_t = 1$ z_{t+1} to be equal to 1, so that it agrees with a_t

Then, we get the following circuit:



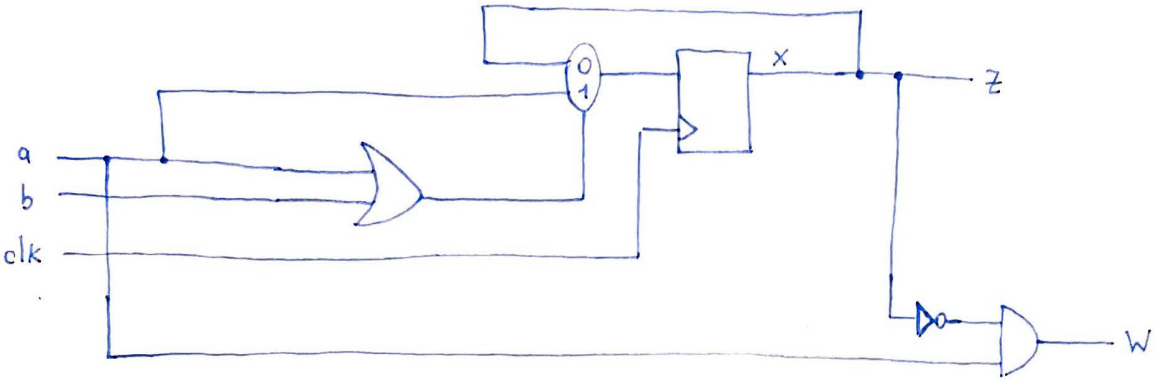
If we have $a=b=0$, then, from the OR gate we will have $z_{t+1} = x_t = z_t$, otherwise $z_{t+1} = a_t$.

(b) We want to produce an additional output w that receives a pulse (one clock cycle) whenever $a=1$, but has to be reset for another pulse (by setting $b=1$ at a clock edge).

To do this, we will use the previous circuit:

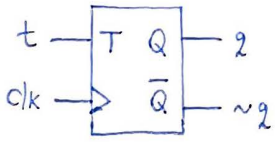
- if z is 1, then we know that a was pressed and we're waiting for b to be pressed, so the system needs a RESET
- if z is 0, then the system is RESET

We therefore have $w = a \wedge \neg z$ (w will only be 1 for a clock cycle, since after a is pressed z becomes 1 at the next clock cycle $\Rightarrow \neg z$ becomes 0 and w will be 0 immediately after the 1)

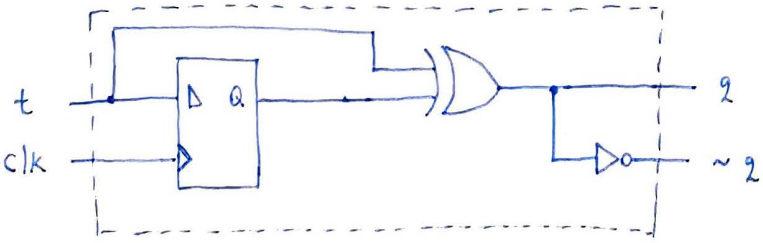


(4) The truth table of a T-type flip-flop is

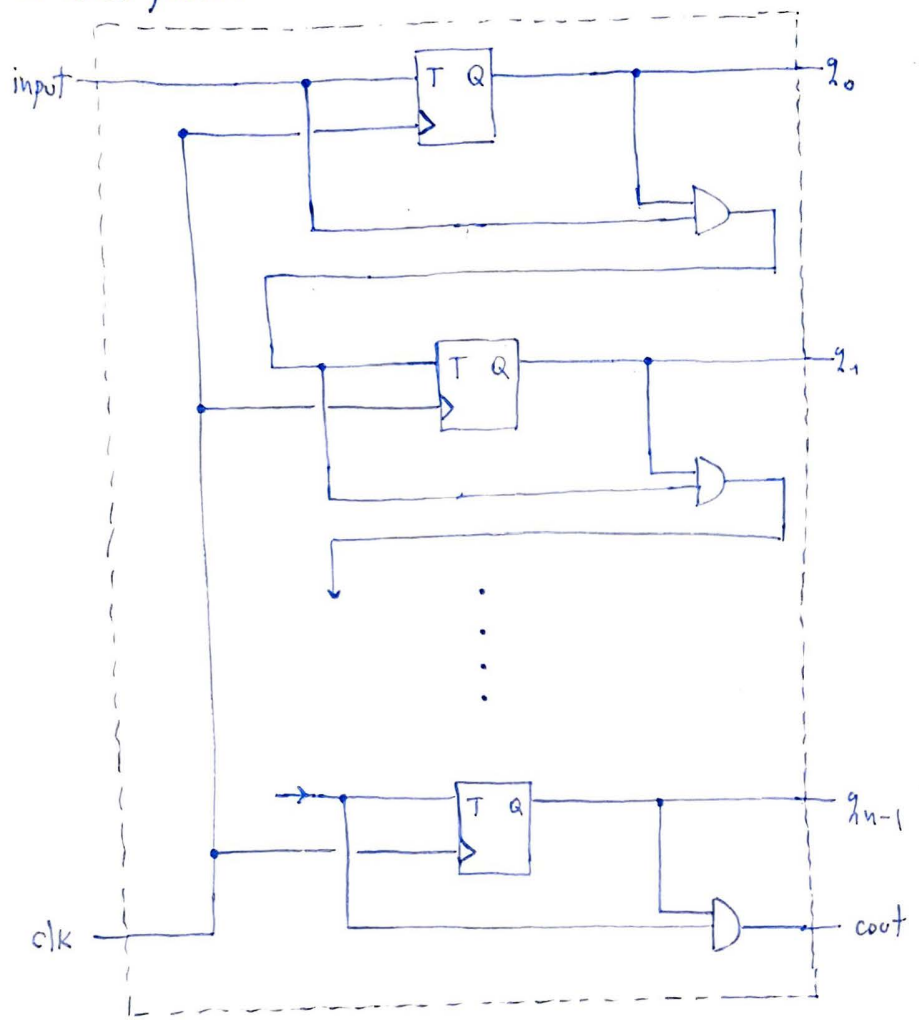
q_t	t	q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	1



(a) The result is basically the same as t XOR the result of a D-type flip-flop, therefore its equivalent circuit is

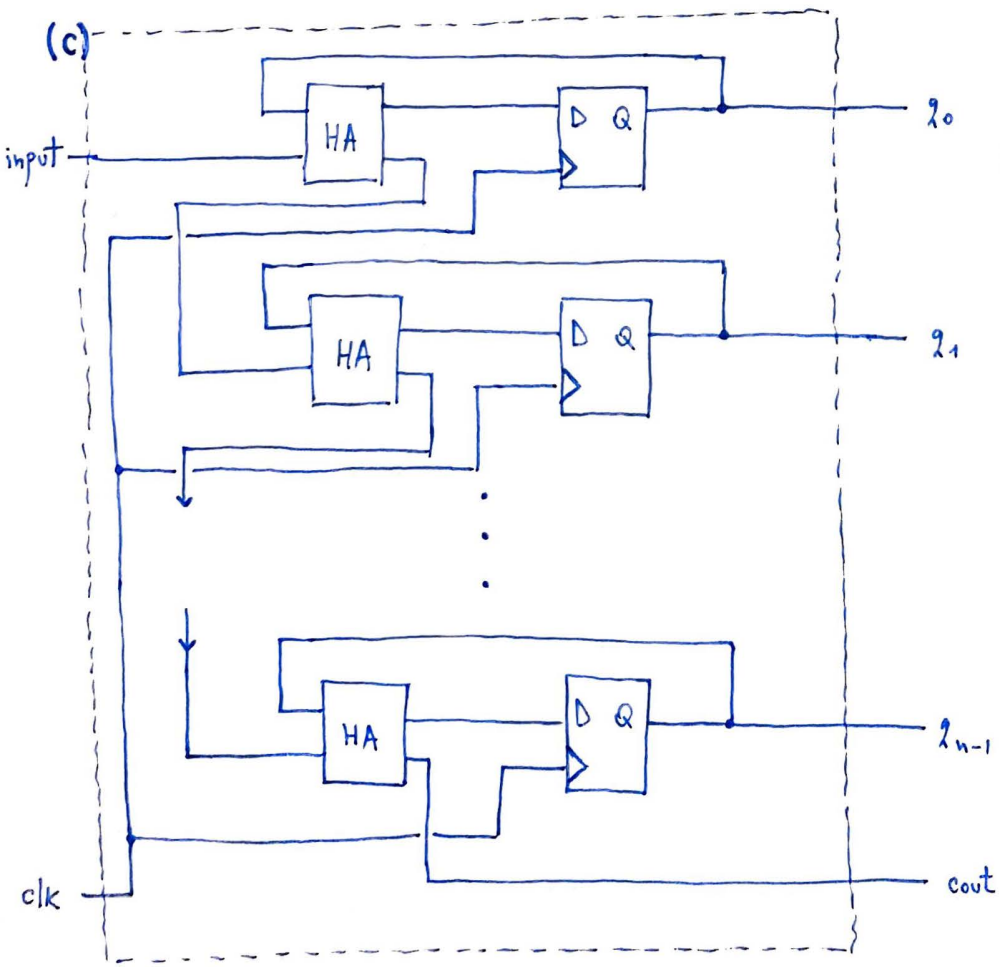


(b) The behaviour is like : we have a carry added at each press and $q_0, q_1, \dots, q_{n-1}$ are the "sums" at each position.



After each step, the output is q_t and the carry is $t \wedge (t \oplus q_t)$

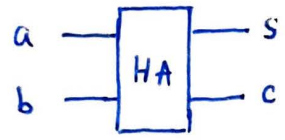
(c)



Here, we have the same thing as above.

(d) Since the T-type flip-flop is created by using an XOR gate to the input and the result of a D-type flip-flop, we can understand more easily the role of the HA part.

Recall that:

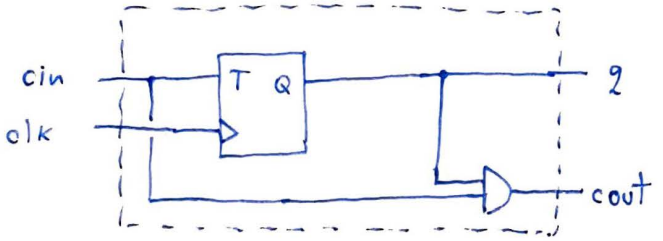


where $s = a \oplus b$

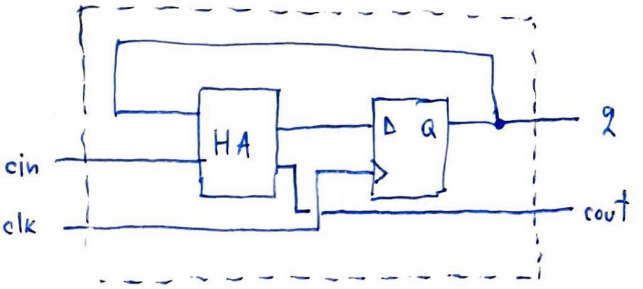
$c = a \wedge b$

To replace the T-type flip-flop we needed to have the XOR operation applied to the input and the previous state (of the D-type flip-flop) and the carry would be just the carry result of the Half-Adder, instead of the conjunction of the result with the input (so the same principle).

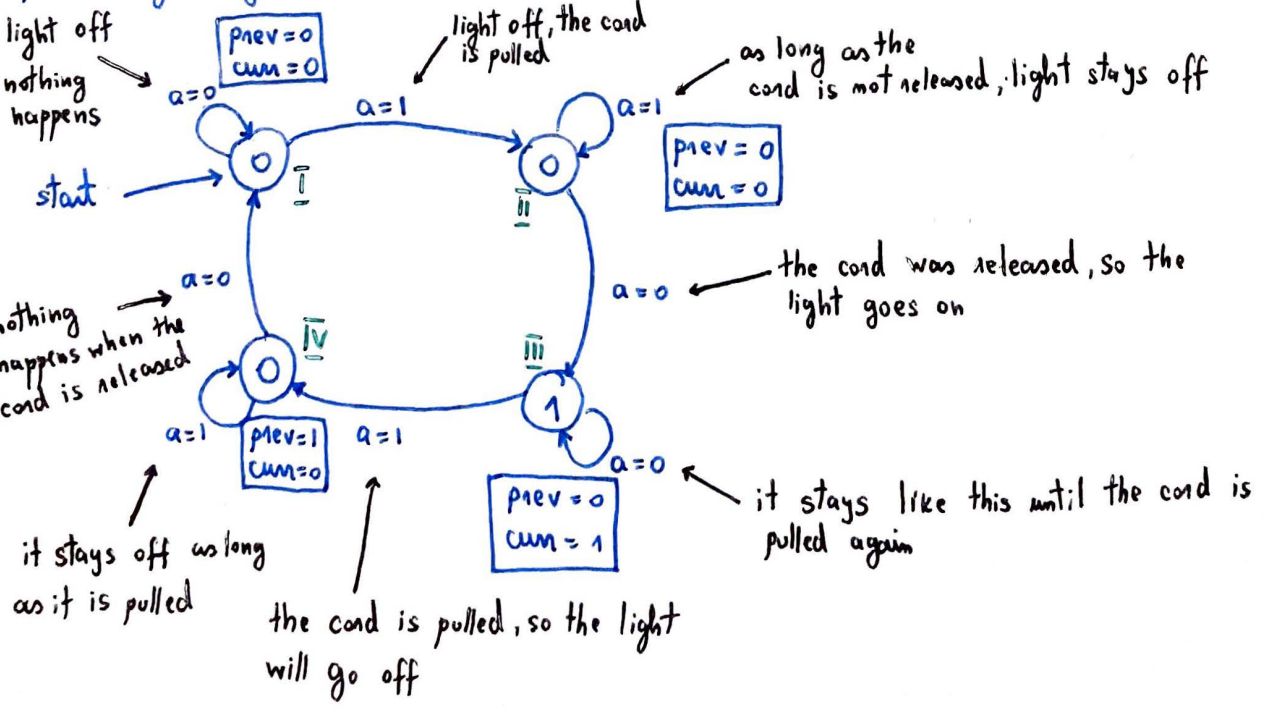
This enforces the idea of equivalent parts of the circuit (like the boxes from (b) and (c)) as we have



equivalent to



⑤ To illustrate the behaviour of the pull-cord light switch, I used the previous two inputs a_{t-1} and a_t and z_{t+1} is going to be curr, where prev and curr are variables that are useful for distinguishing between decisions:



Therefore, we can observe the following behaviour:

a_{t-1}	a_t	prev	curr	
0	0	$prev_0$	$curr_0$	← either in state I (transitioned from IV) or in III (from II) — maybe!
0	1	$curr_0$	$prev_0$	← transition from I to II or from III to IV (see table below)
1	0	$curr_0$	$\neg prev_0$	← transition from II to III or from IV to I (see table below)
1	1	$prev_0$	$curr_0$	← either in state II (transition from I) or in IV (from III) — maybe!

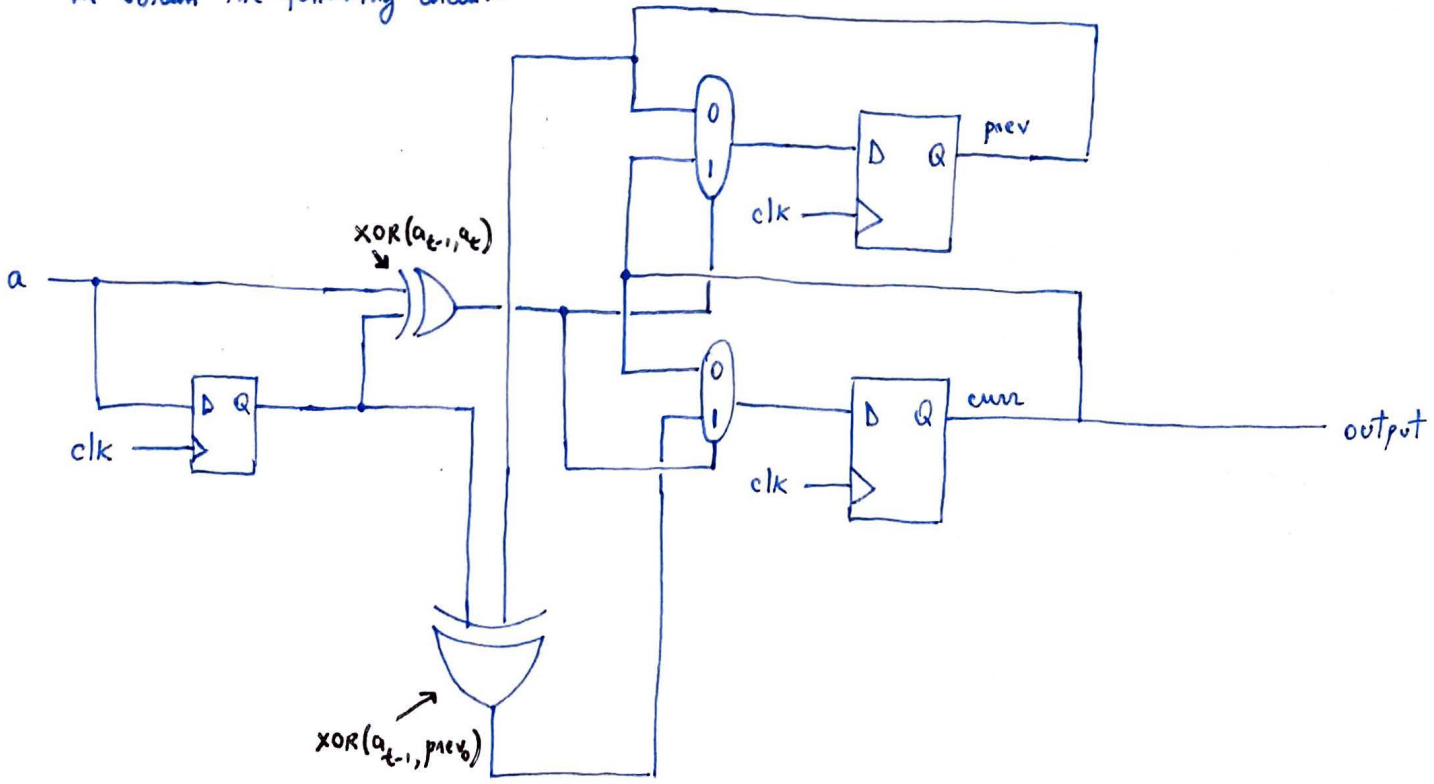
! $prev_0$ and $curr_0$ are the values of prev and curr one clock cycle before

Transitions	$(prev_0, curr_0) \rightarrow (prev, curr)$	
I → II	(0,0) → (0,0)	$(prev, curr) = (curr_0, prev_0)$
II → III	(0,0) → (0,1)	$(prev, curr) = (curr_0, \neg prev_0)$
III → IV	(0,1) → (1,0)	
IV → I	(1,0) → (0,0)	

From our behaviour we can conclude that

a_{t-1}	a_t	(prev, curr)	
0	0	SAME	
0	1	$(curr_0, XOR(0, prev_0))$	if $XOR(a_{t-1}, a_t) = 1 \Rightarrow$ $prev = curr_0$, $curr = XOR(prev_0, a_{t-1})$
1	0	$(curr_0, XOR(1, prev_0))$	
1	1	SAME	otherwise it's the same as before

We obtain the following circuit:



To not overcomplicate the circuit (more than it already it, sorry about that) I decided to use 3 clocks (they are all connected to a global one).

⑥ We know that we can express any Boolean function with the set $\{\wedge, \vee, \neg\}$ or $\{NAND\}$.

(a) We'll show that we can express any Boolean function using just NOR.

Recall that

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

We claim that $NAND(a,b) = NOR(0, NOR(NOR(0,a), NOR(0,b))$

First, $NOR(a,0) = \neg a$ for any a, since

a	NOR(a,0)
0	1
1	0

Then, we have

a	b	NOR(0, NOR(¬a, ¬b))			
0	0	1	0	1	1
0	1	1	0	1	0
1	0	1	0	0	1
1	1	0	1	0	0

which is equivalent to $NAND(a,b)$.

So, since NAND can be expressed by NOR and since NAND can be used to express any Boolean function, then NOR can be used to express every Boolean function.

(b) We will try to express every Boolean function using only XOR, ¬, 0, 1, where 0 and 1 are the Boolean values for false and true.

We need to express 16 such functions, which are

a	b	f_1 ✓	f_2	f_3	f_4 ✓	f_5	f_6 ✓	f_7 ✓	f_8	f_9	f_{10} ✓	f_{11} ✓	f_{12}	f_{13} ✓	f_{14}	f_{15}	f_{16} ✓
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

First, we obviously can express f_1 and f_{16} with the identity functions, then f_7 is actually XOR, f_{10} is ¬ applied to the result of XOR, f_4 is a XOR 0, f_{13} is a XOR 1, f_6 is 0 XOR b and f_{11} is 1 XOR b.

We are now left with 8 functions that we'll prove that cannot be expressed with

what we currently have.

Notice that if we sum all the results of every function we could express so far, we get 0 as a result (modulo 2). We want to obtain functions which all have the sum of the results equal to 1.

Let's suppose we can and therefore we will use two of the functions that we have and we will apply XOR to them.

f_a	f_b	f_a XOR f_b
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4

We know that XOR behaves like addition (modulo 2), so we get that

$$\begin{cases} a_1 + b_1 = c_1 \\ a_2 + b_2 = c_2 \\ a_3 + b_3 = c_3 \\ a_4 + b_4 = c_4 \end{cases}$$

addition is commutative and associative

$$\Rightarrow c_1 + c_2 + c_3 + c_4 = (a_1 + a_2 + a_3 + a_4) + (b_1 + b_2 + b_3 + b_4)$$

Since all the functions we have discovered so far have sum 0, then

$$c_1 + c_2 + c_3 + c_4 = 0$$

So, since we already discovered all the functions with sum 0, the new function we discovered was already in the set.

Analogously, we can apply $\neg$, or use 0 or 1 in any combination with XOR, but the sum will never get to be 1, so the set of functions we discovered is closed under XOR and $\neg$.

Therefore, the set $\{XOR, \neg\}$ cannot express the whole set of Boolean functions.