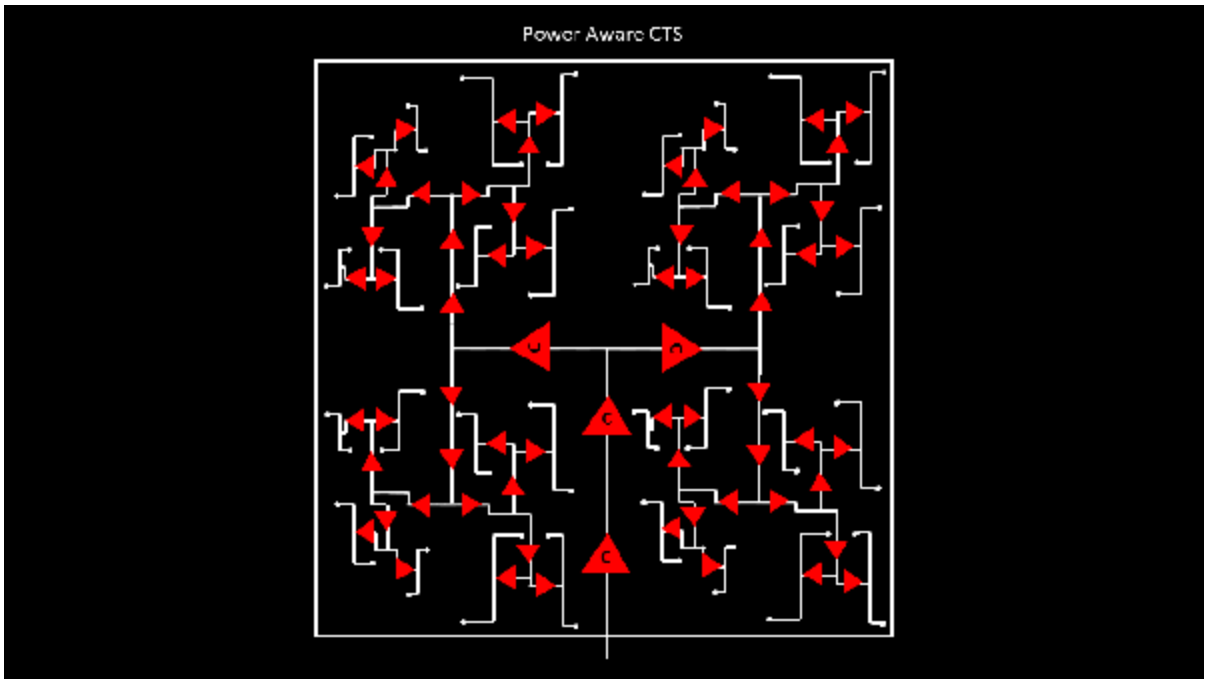
Power aware clock tree synthesis

Kunal Ghosh

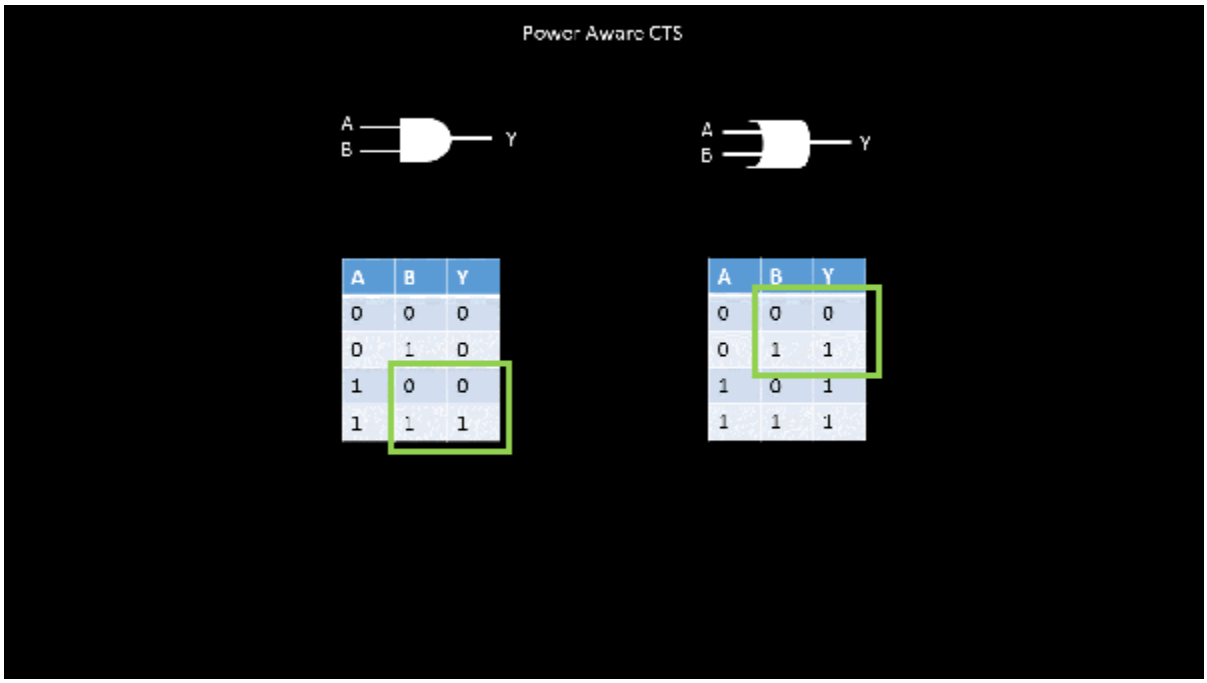
How about saving some power for the below **clock tree network**?

It's tricky and if we go with the right approach, we might end up saving almost 'half' of the power that the below clock network might take as of now. (Apologies to experts on this topic as I might go to very basic level, but I can guarantee that it would be interesting).

By the way, **this technique can be used only under certain conditions where some parts of the chip remain silent**, while the other part switches. Stay with me to know 'how?'

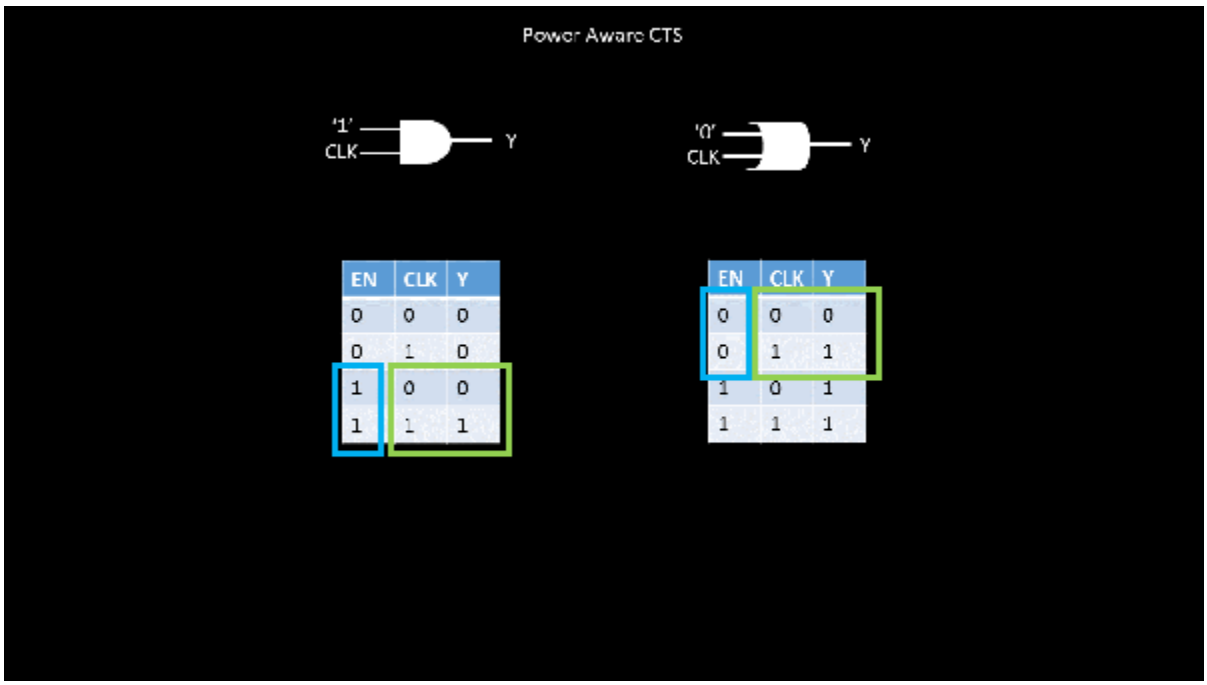


And, if I say, the solution lies in the functioning of below gates (already apologized to experts of this field for going to basics :)), you will be surprised to watch how we trick the **clock network using handful below gates to reduce power significantly** (at-least half)



There are many ways to view things. As an eg. you would view the above **gates as AND/OR gates with 2 inputs A and B**, but I would look the functioning of these gates as an opportunity to reduce power. How?

Look at the below image now, and you will realize that, if I hold one of the inputs at **constant logic level,** these gates will then behave like a **simple buffer** i.e. in both below cases, Y = CLK (like in a buffer Y = input)

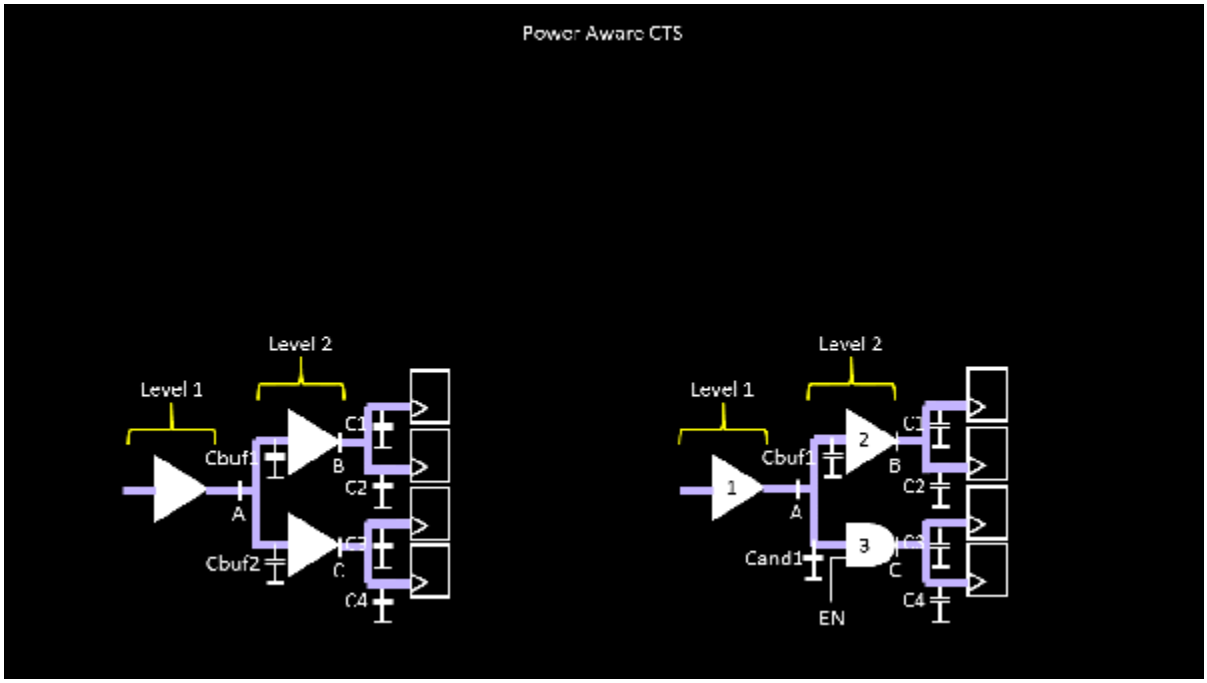


So, I will call one input of the gates as ‘EN’ i.e. enable signal, and the other one as ‘CLK’. **In case of an ‘AND’ gate, whenever ‘EN’ is high, and in case of OR gate whenever the ‘EN’ is low, the output ‘Y’ is ‘CLK’**

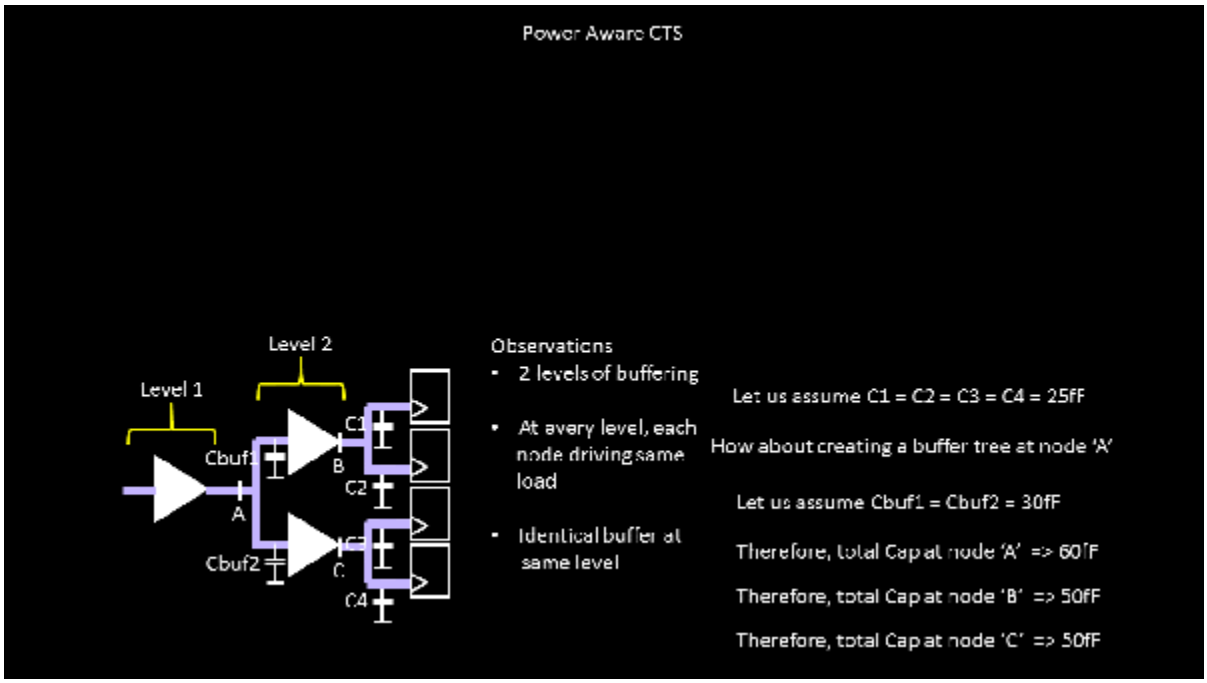
With these basics in place, next step is identified functionality for those parts of your chip which remain silent or do not function while another part of your chip, which functions.

If we can isolate these parts, we can combine that with above concepts and come up with a clock network which will potentially save huge power, in turn, which will help you save some battery on your mobile phones.

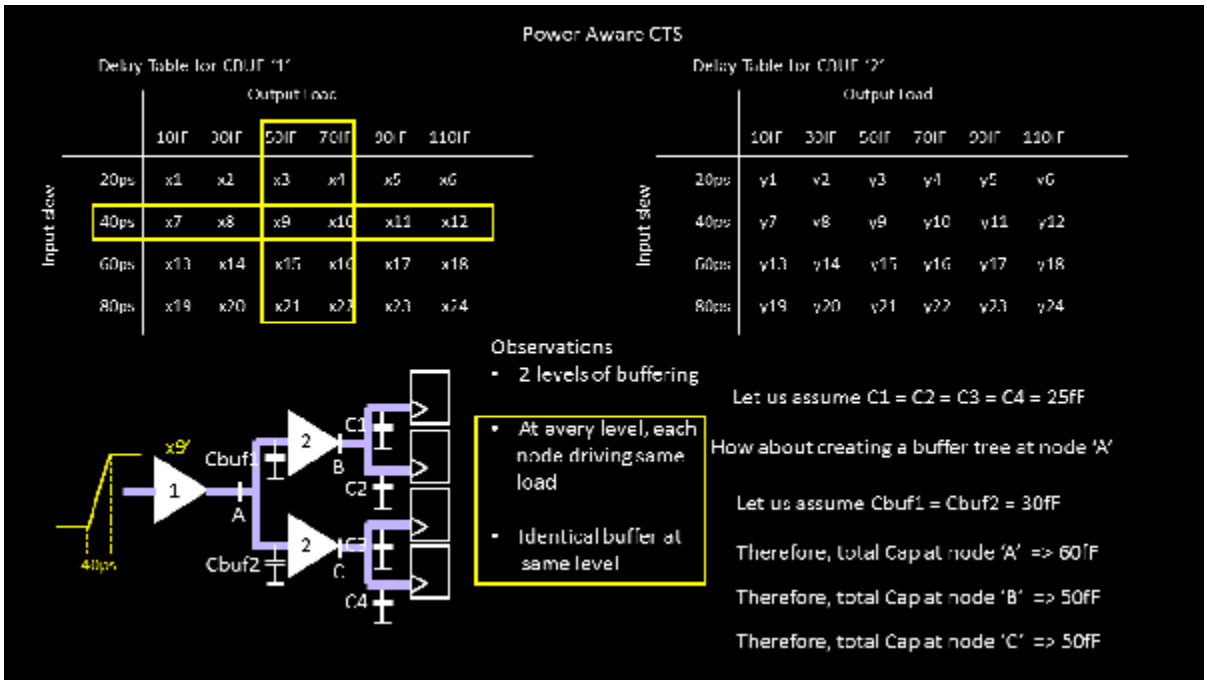
To nail the bigger picture, I will use the below small one to explain, what I mean by above statement



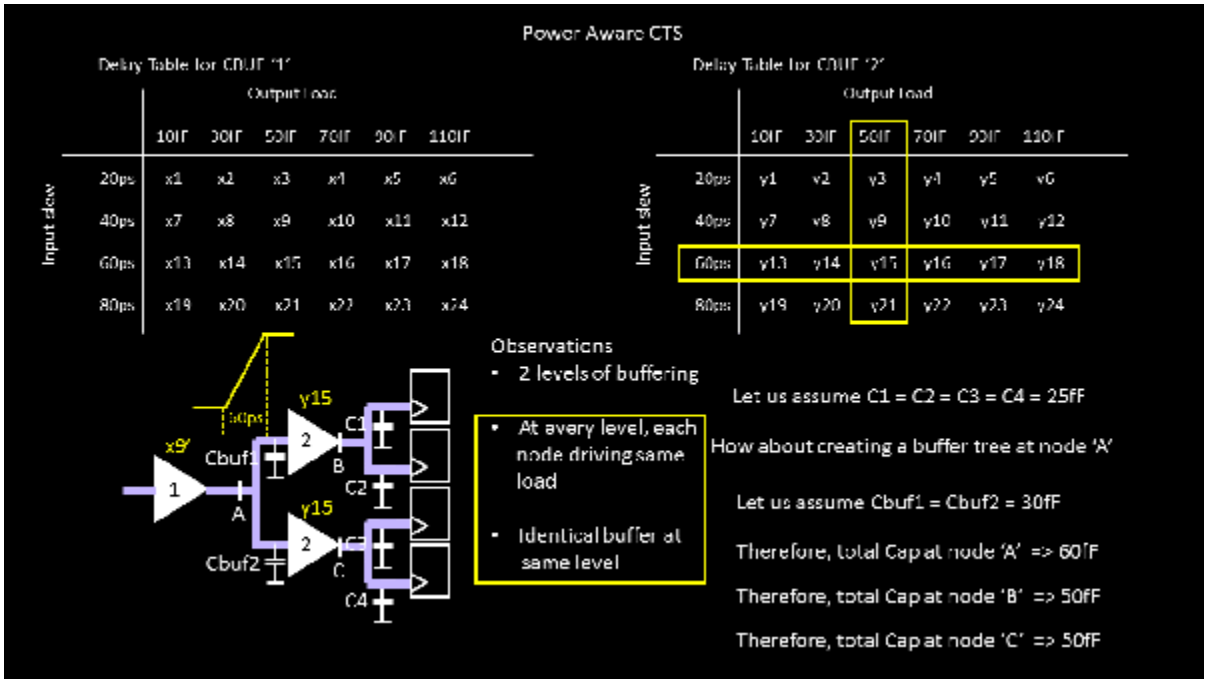
While trying **to build a clock tree which is power aware**, let’s go back a step ahead and look, what are the top observations for a clock network, and below image covers the same.



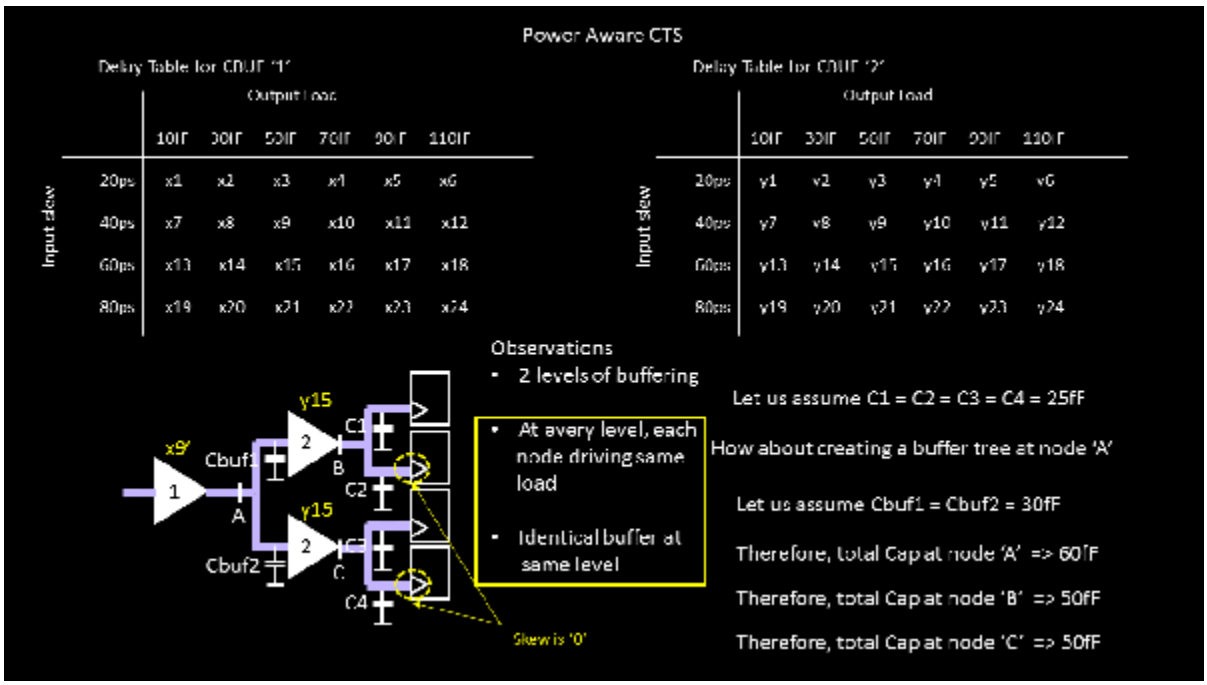
Assuming a slew of '40ps' for the first buffer and capacitance of 60fF on node 'A', the delay of the first buffer can be easily evaluated using below **NLDM table** and it comes out to be x9′



And similarly, for the **second level of buffering**, the delay of the buffers '2' and '3' comes out to be y15 assuming a slew of '60ps' and capacitance of '50fF' at node 'B' and 'C' (as I have **built a close-to-perfect clock tree**:))



This in turn results to **'zero' skew at clock endpoints**

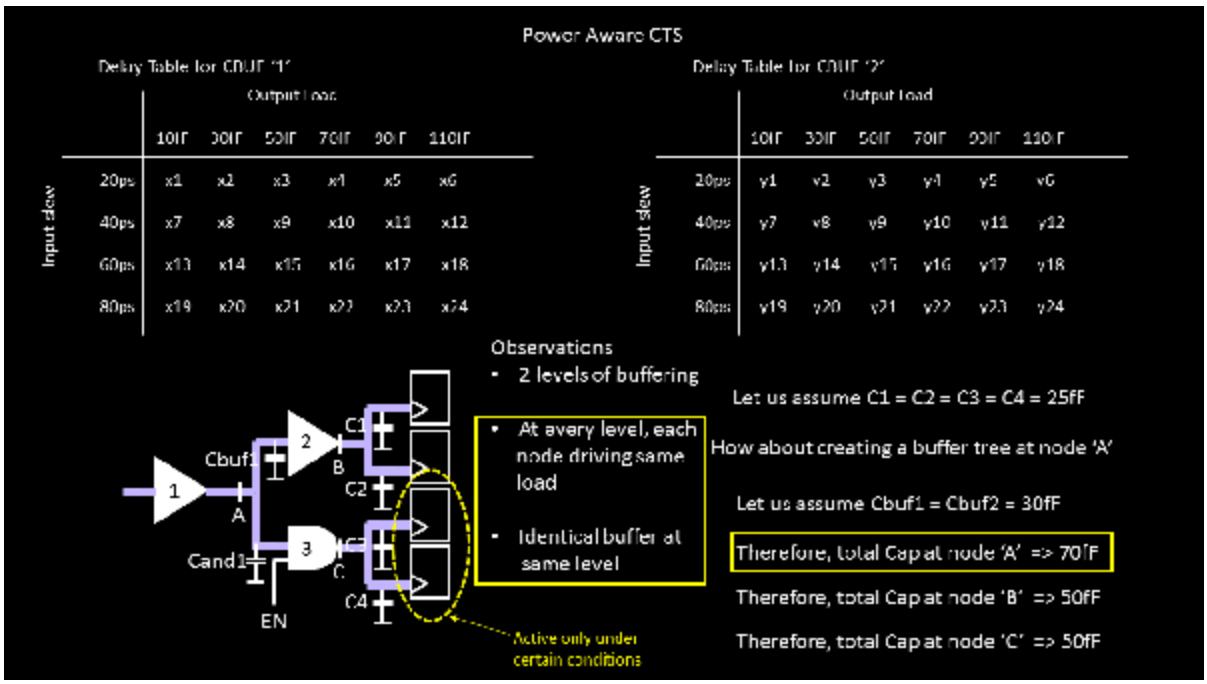


A detailed explanation on the above is present in my [Clock Tree Synthesis Course on Udemy]

Now, that's important thing we jumped into. While we are trying **to modify the above clock tree to be power aware**, we need to make sure the above observations are retained. That's where the challenge lies.

Let's describe the problem statement here:

In below case where 2 flops are active under certain values of 'EN' and the other 2 flops are active all the time, we can use an AND gate to get that functionality

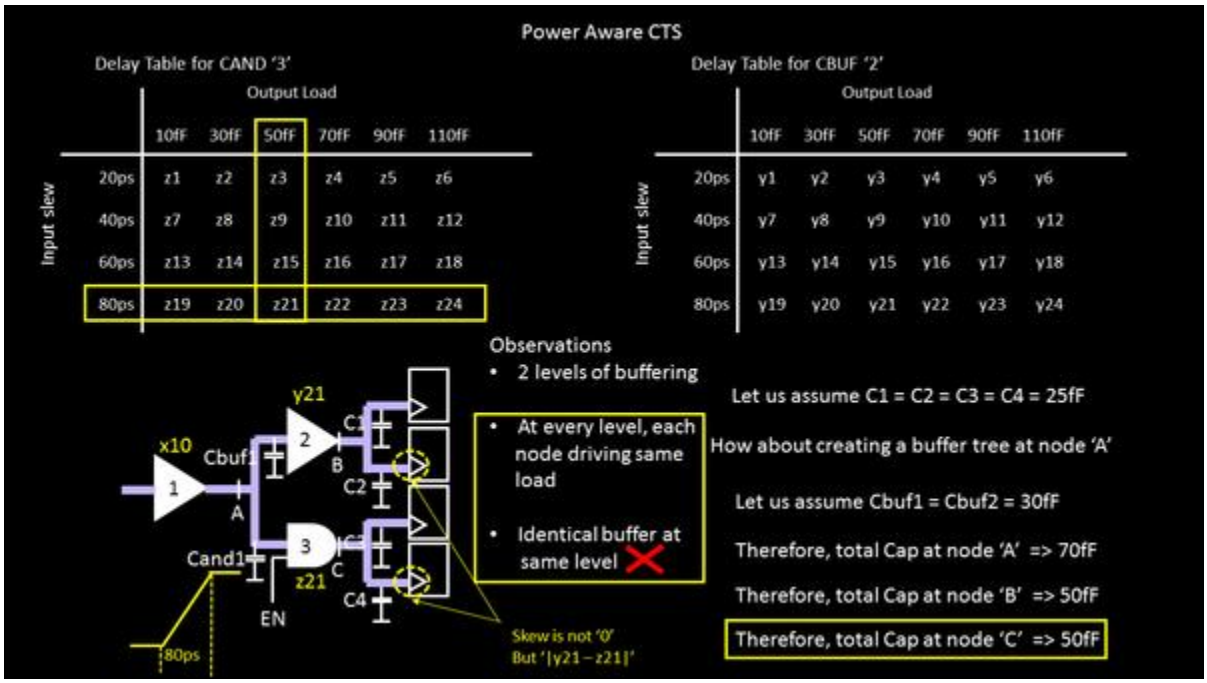


But look what happens, just as we plug the AND gate in the circuit:

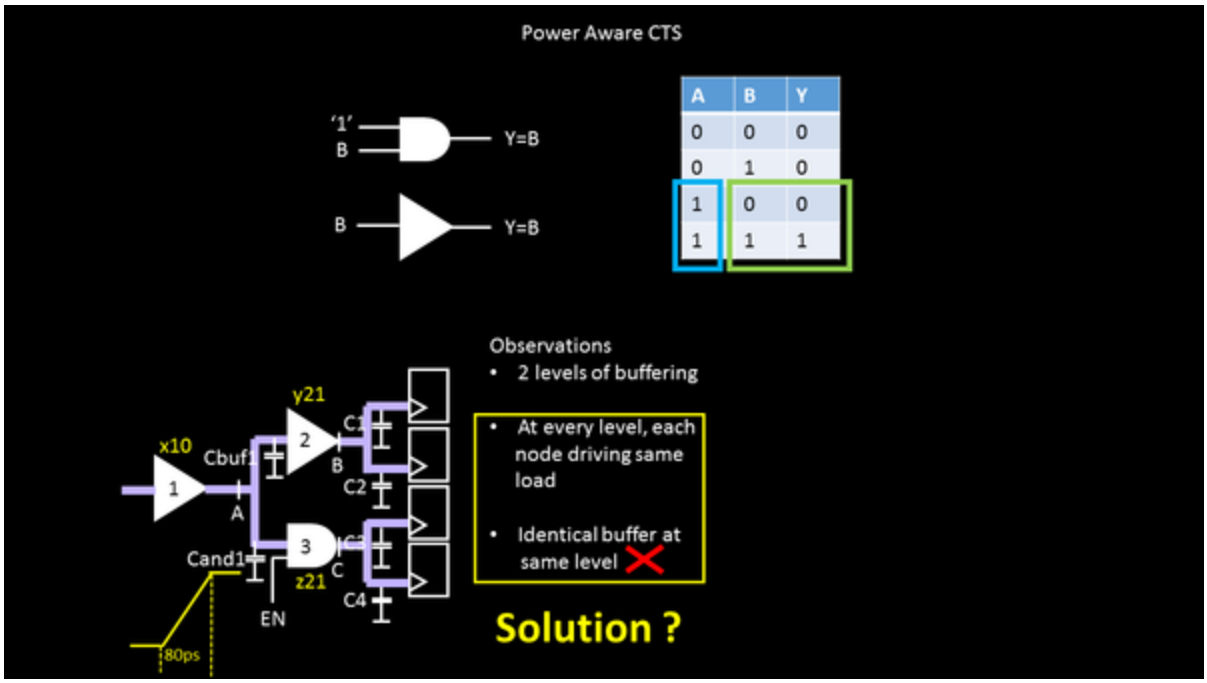
1) **the output capacitance for buf '1' varies — No problem**

2) **the output capacitance at node 'B' and node 'C' remains same — Excellent**

3) **the delay of level 2 gates (buffer for top path and AND gate for bottom path) varies — Big problem, as this will disturb the skew, which we had designed to be 'zero'**



And here's the solution to the problem posted in my previous article. 'AND gate itself'. If you observe carefully, you tie one of the inputs of an **AND gate to 'logic 1' and it will behave like a buffer**. Advantage …. I will show you soon. Disadvantage… Area overhead

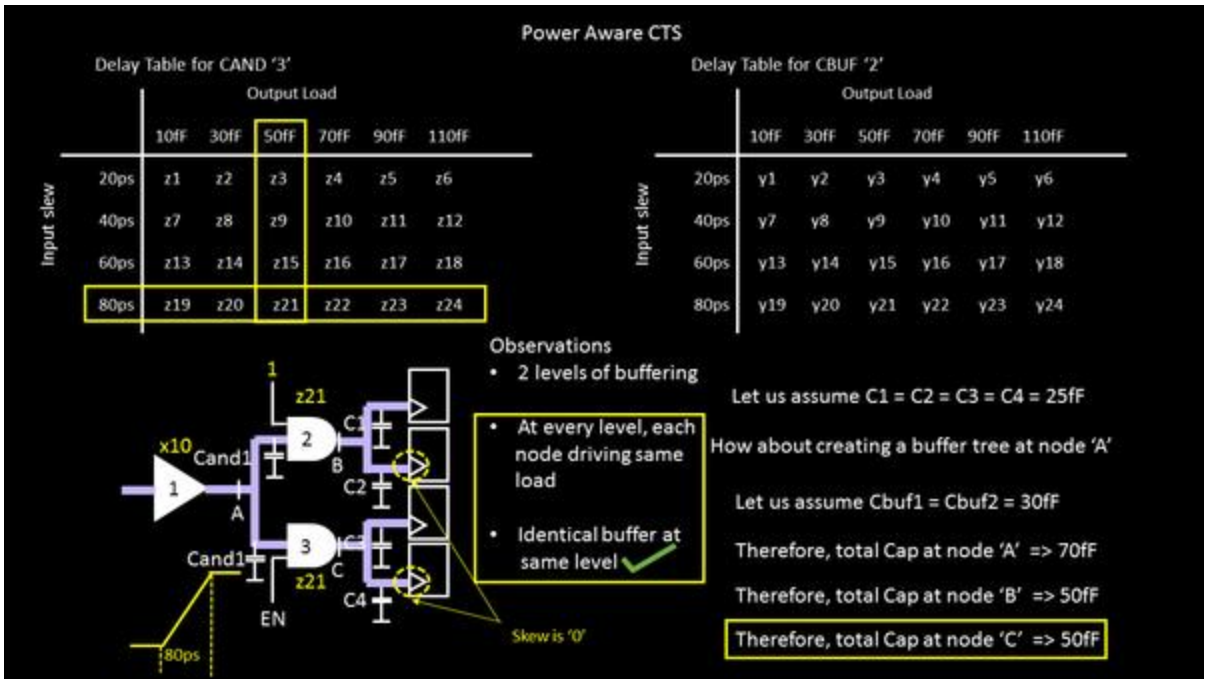


There are 2 advantages of using **AND gate as buffer**

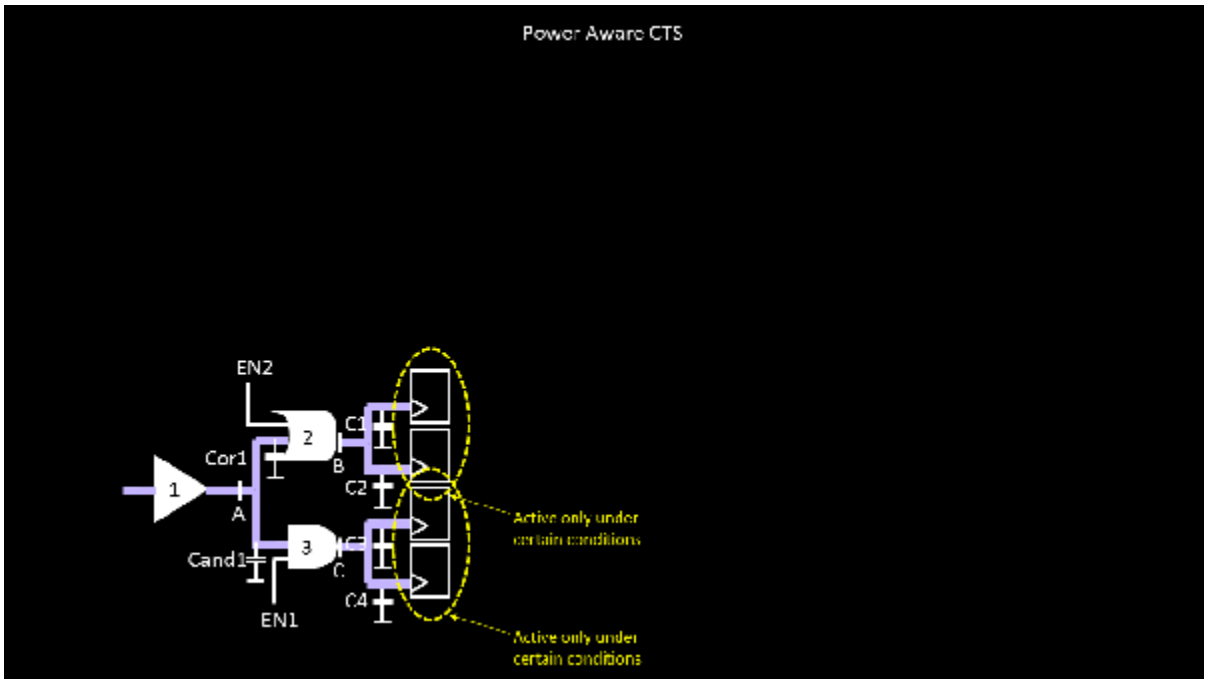
1) you tend to use identical buffer at level 2 i.e. AND gate as buffer

2) you save power, by turning on/off the EN pin of AND gate 3 and disabling clock to whole bunch of flops connected to its output

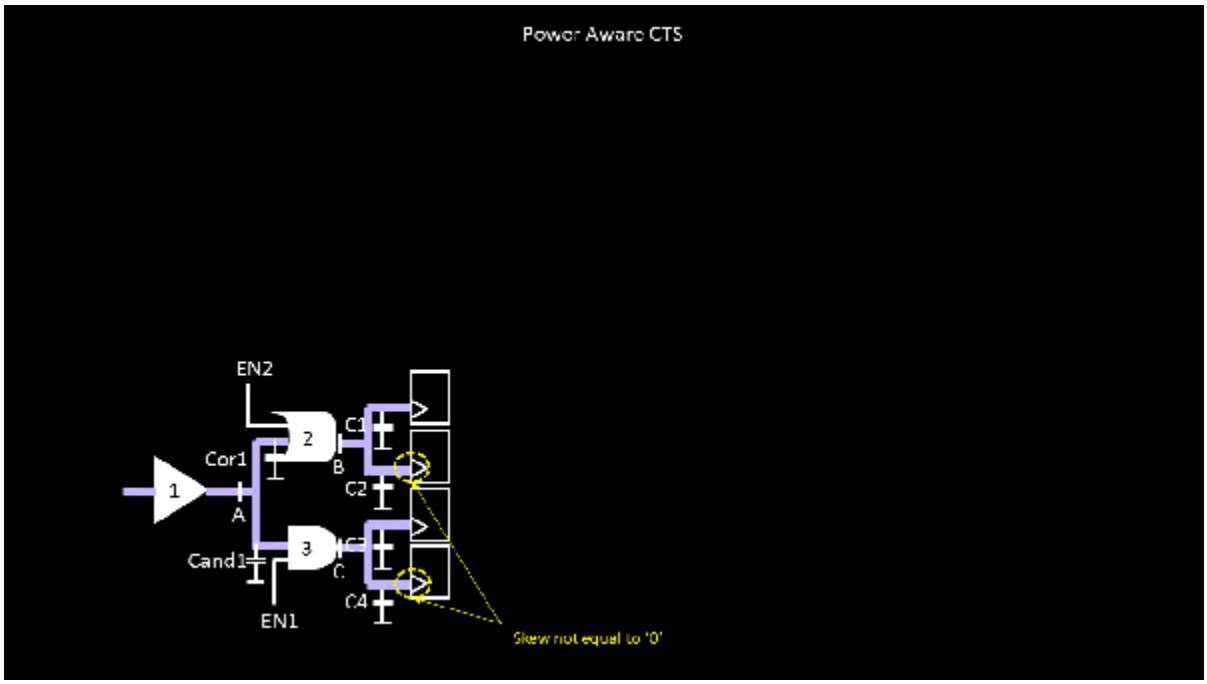
3) you maintain zero skew while doing above 2.



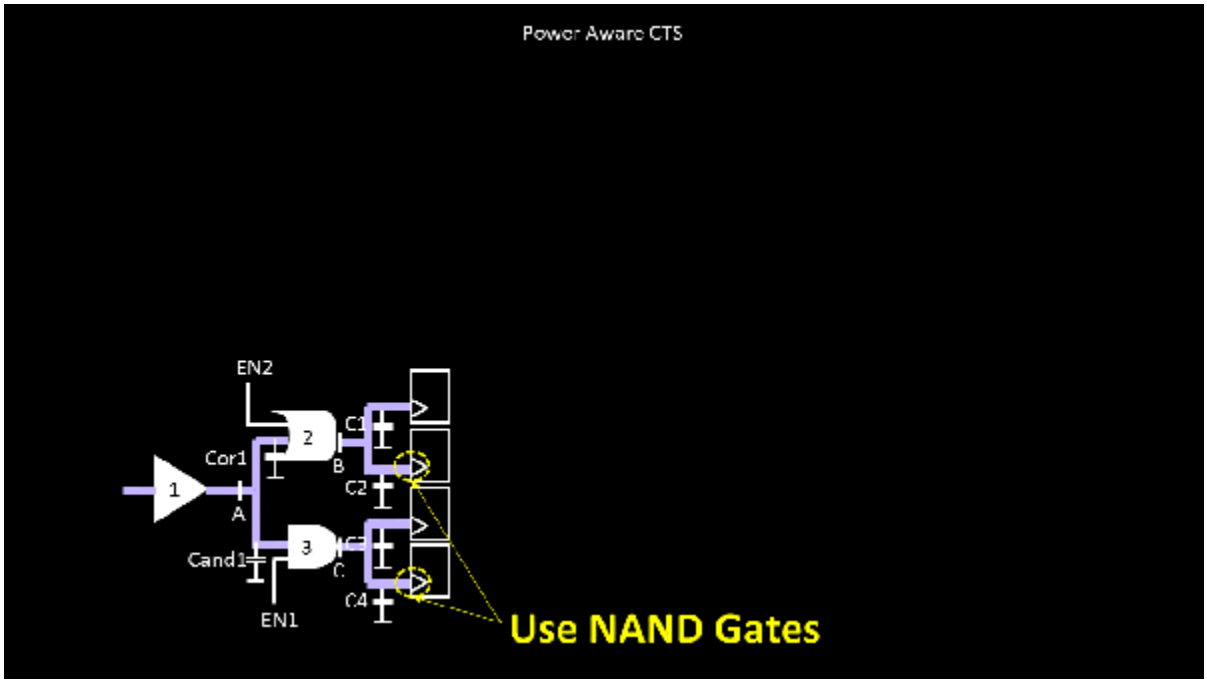
Let's take another valid scenario below, where **both set of flops are active under certain conditions and conditions are driven by the values on the EN1 and EN2 pins**



This is where again, we hit the scenario of non-zero skew due to different delay values of AND and OR gate for same set of input slew/output load conditions.

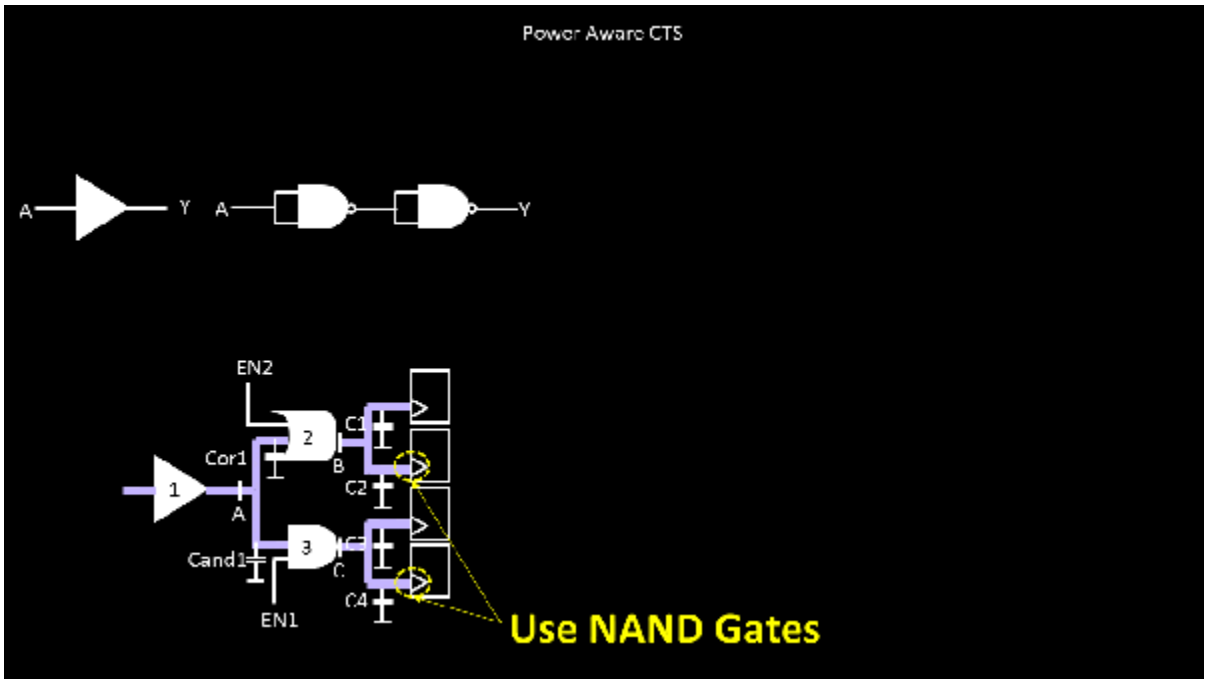


We were looking for a solution for below scenario, and you will be amazed to see, **how an 'Universal Gate' solves the below problem** (of-course, at the **cost of increased area, but you save lot of power,** which is a prime necessity in your smartphones)

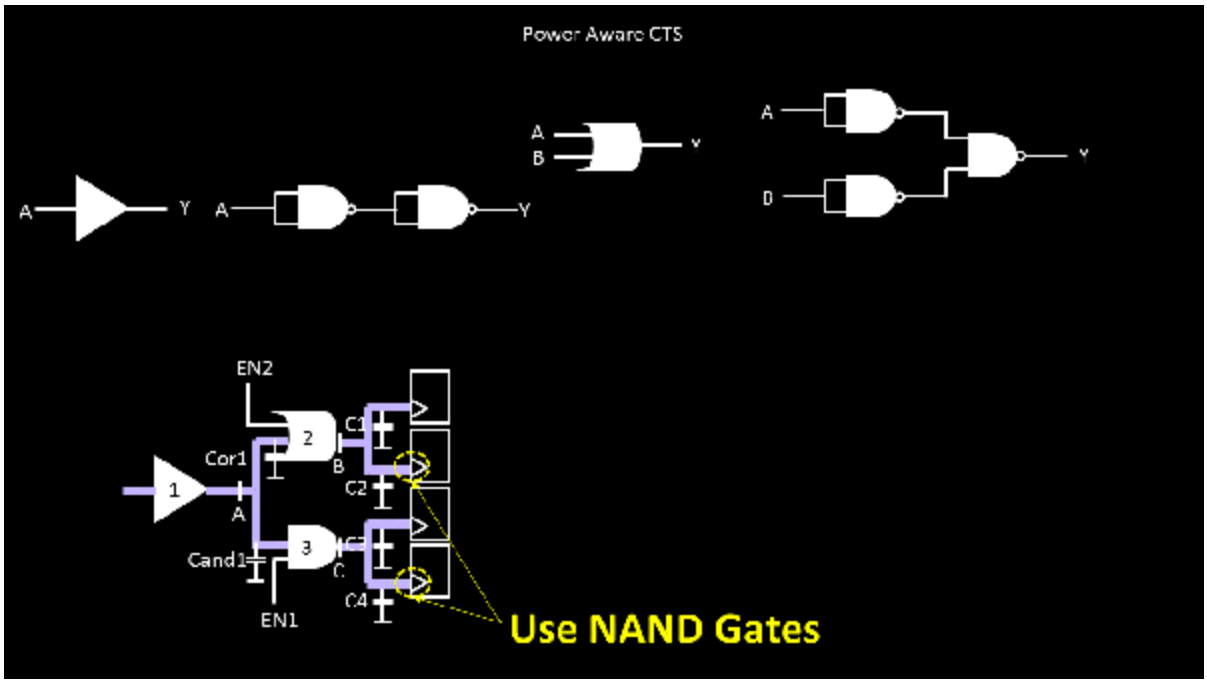


The question is 'How'? Let's go back to school and recollect '**We can build any gate using NAND/NOR gate,** and hence they are called "universal gates" '

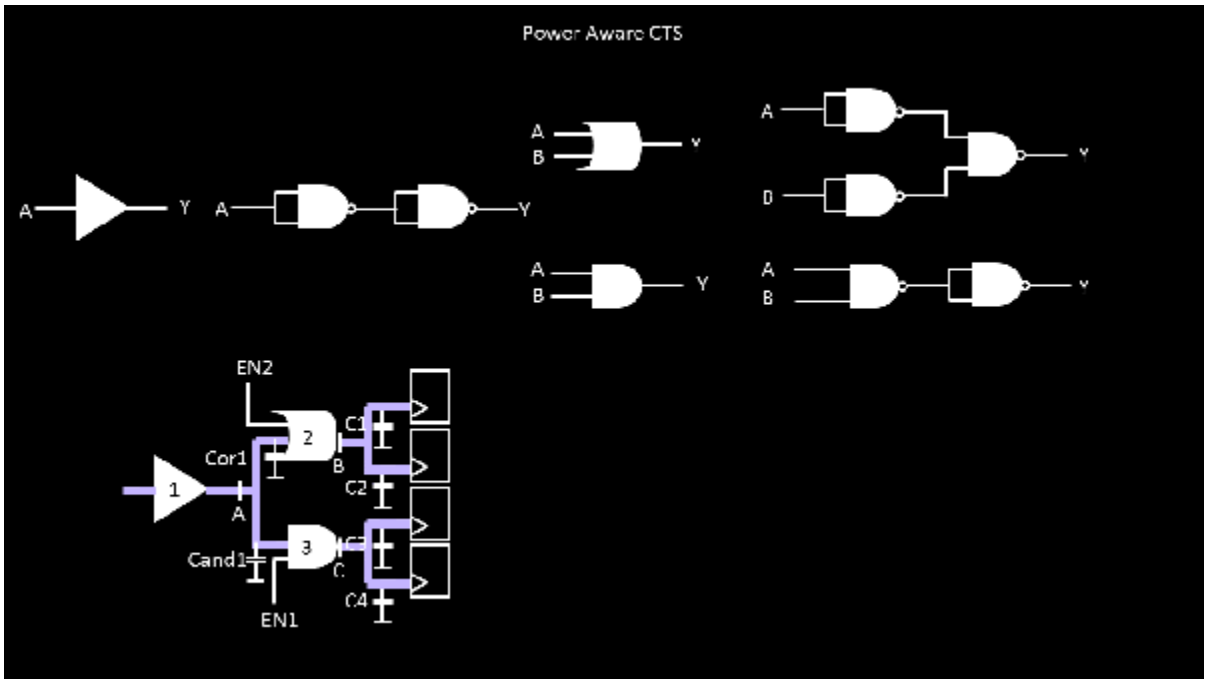
So, a buffer can be reconstructed something like below….



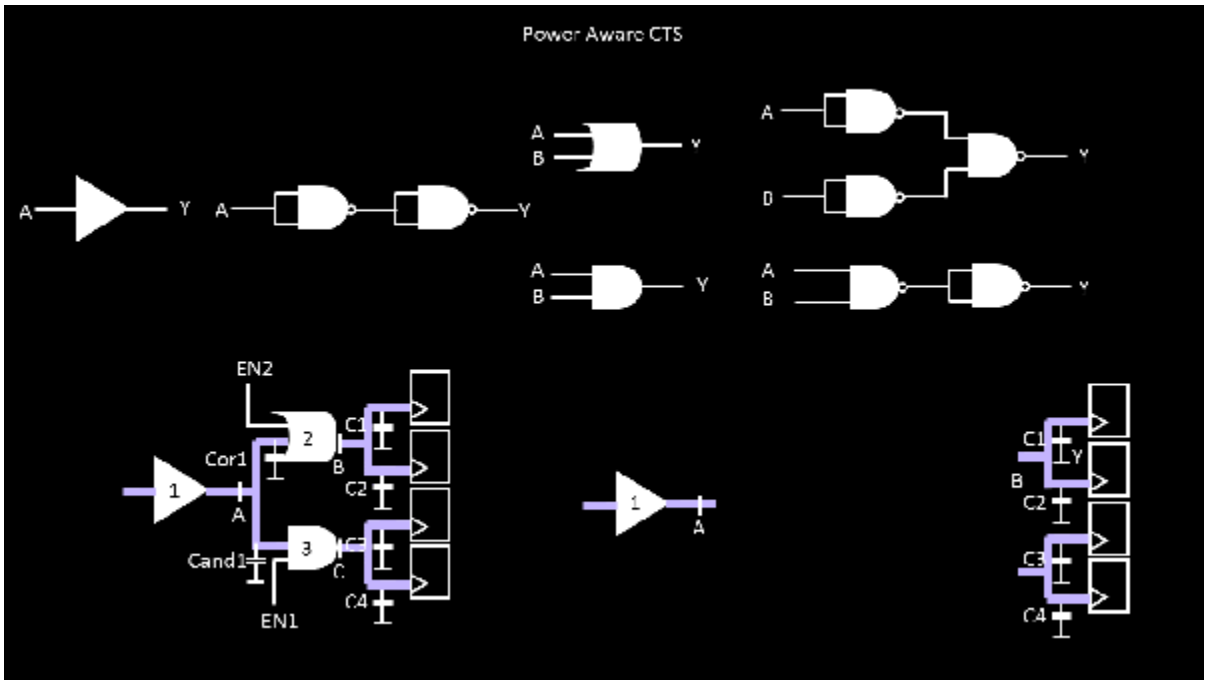
…. **OR gate** becomes something like below….



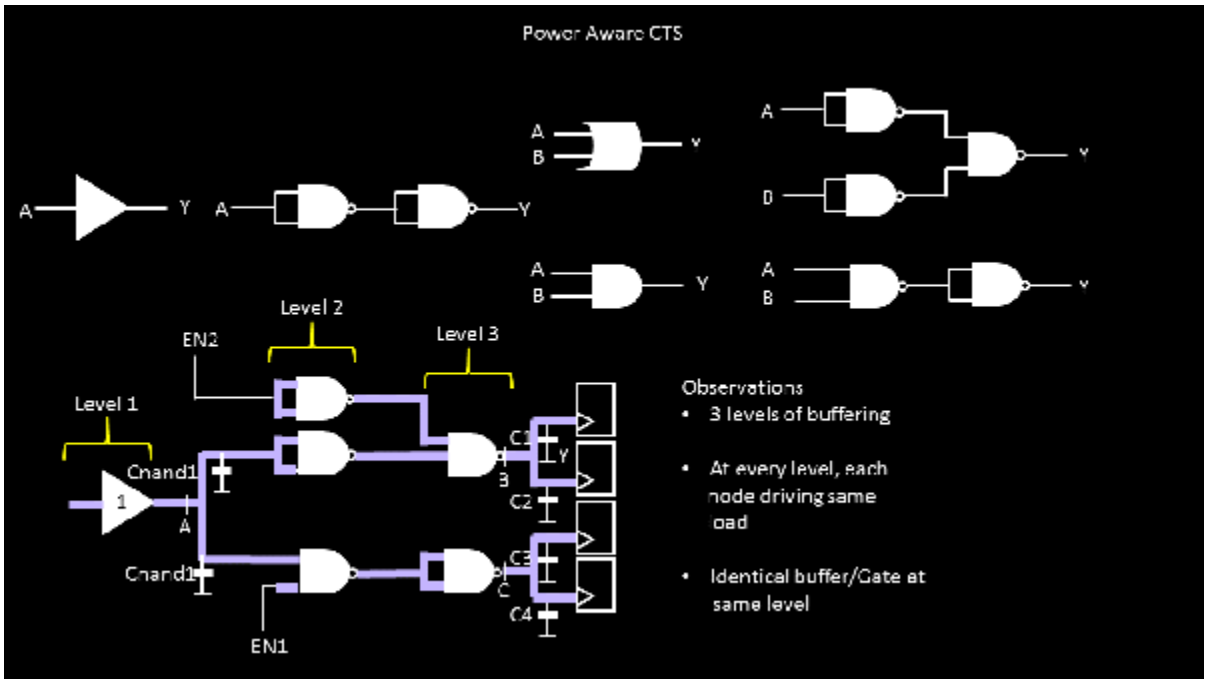
….and finally, **AND gate** becomes like below



I have solved 50% of the problem. The rest 50% lies in plugging the right kind of gate (**using NAND gate**) at below right place



And there you go, you plug the right logic and achieve things needed for a '**zero skew' clock tree network, while saving immense amount of power**



You might think, we are adding a lot of area into the design. But think of a clock tree network that was huge enough like below. **Using this simple and effective technique, we save almost half of switching and short circuit power**