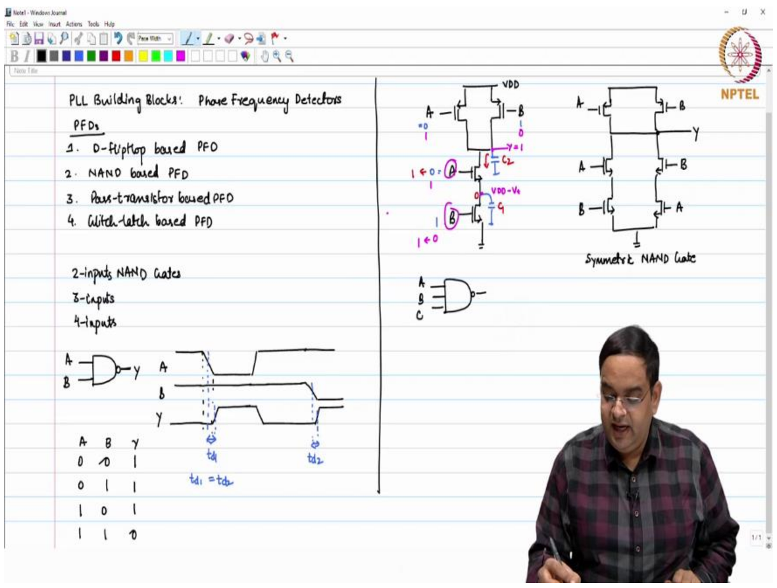
Phase-Locked Loops
Dr. Saurabh Saxena
Department of Electrical Engineering
Indian Institute of Technology Madras

Lecture – 50
Circuit-level Design of PFD: Part III

(Refer Slide Time: 0:15)



PLL Building Blocks: Phase Frequency Detectors

PFDs
1. D-flipflop based PFD
2. NAND based PFD
3. Pass-transistor based PFD
4. Glitch-latch based PFD

2-inputs NAND Gates
3-inputs
4-inputs

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Hello everyone. Welcome to this session. We have been looking at the PLL building blocks and in the last session we looked at PFDs, phase frequency detector is the one which we looked at. We looked at two different phase frequency detectors. One was the D-flip-flop based and the other was the NAND based.

There are other PFDs also or other implementations of the PFDs. And let me just, if I want to summarize, you have seen D-flip-flop based, you know what kinds of problems you have, D-flip-flop based PFD. You have seen NAND based PFD, and there were two different architectures for that. One with all NAND gates and the other with a slight modification on that NAND gate.

The other one which we will not go in this, you can always refer to the literature for that, pass-transistor based PFD, and another one is glitch-latch based PFD. The reason that we have other two PFDs, pass-transistor and glitch-latch based PFD, is specifically to reduce the reset time and to minimize the window during which the average output of the PFD is negative even when the phase error is positive.

But you have other designs also. This list is not ending here, there are many designs. The only reason why I will stop with the NAND based PFD is because D-flip-flop and NAND based PFDs are most commonly used unless that negative region becomes too critical.

So, in the NAND based PFD, you have seen 2-input NAND gates. So, you have seen 2-input NAND gates, 3-inputs and even 4-input NAND gates. Now, when you have different input to the NAND gates, you want that the output of the NAND gate when you implement, the output should make a transition in the same way whether input A changes or input B changes.

For example, if you have a NAND gate like this, A and B and the output is Y. Let us say A is 1 and B is also 1 or logic high. And A makes a transition like this, and then after some time, it becomes, it is back to high. If A makes a transition like this and B remains high, your Y output A NAND B, your Y will be equal to 1.

Because what you have is, so, our operation is, if you look at it, A, B and Y, when A is 0 and B is 0, this is 1. When A is 0, B is 1, then also it is 1. When this is 1, this is 0, then also this is 1. And when A and B both are 1, the output is 0. So, the output was 0 and after some time, when you make a transition at A, what happens is that A and B both, A becomes 0, B is equal to 1. So, after some time, your Y will become 1. And then let us say when A is back again, you will have Y as 0 again. So, this will happen.

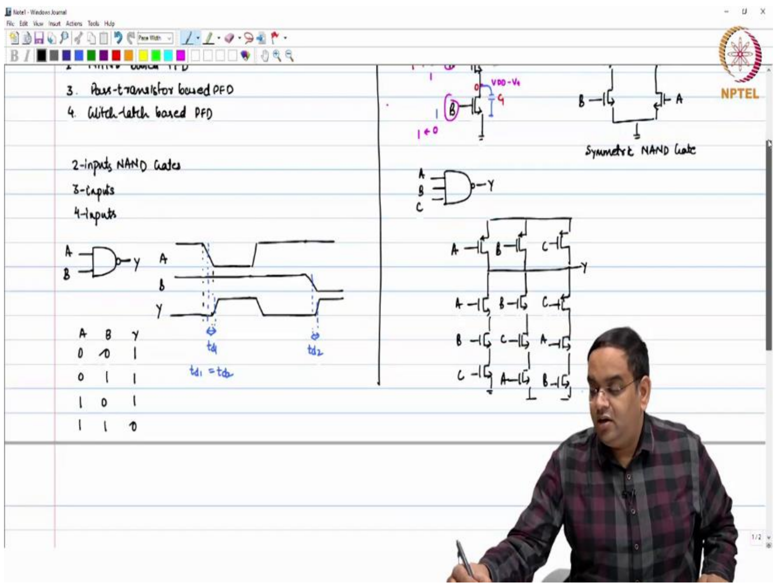
Now, let us say that in place of A, your B changes. So, A remains the same, B changes, B becomes low. In that case, your Y will again become high, this is going to happen. Now, in this case, if you look at it, the delay between the transition time between A and Y and the transition between your B and Y, these two transition delays should be same ideally. $t_{d1} = t_{d2}$, final logic is surely going to be the same, but the PLL operation would also require that this should also be same and the reason is because this is NAND based PFD, your A and B signals will be like your reference clock and your feedback clock. And if you have a phase error, you do not want the output to be changing with respect to the sign of the phase error. It should change by the same amount whether it is positive or it is negative.

But what happens when you implement this NAND gate using transistors? So, when we implement using transistors, the logic which you see is like this and then you have something like this. So, this is your typical NAND gate, you have A and B and this can be A and B. Now, here the parasitic capacitance at each node actually differs. So, consider the case when A and B both were equal to 1. At that time, this was discharged or if you think the other way when A is 0 and B is 1, either A is 0 and B is 1.

When A goes to 1, at that time, because B was already 1, so, this would have been discharged to 0, and then when A goes 0 to 1, at that time, you are going to discharge only this node, the capacitor which you are having here, C_1 and C_2 here. In place, if you have A as 1 initially and B as 0 and if A was 1, the output node, this Y node is actually, if you look at it, B was 0. So, this node, because B is 0, and this is 1. So, output node was initially 1, and this particular node, because A is 1, this is charged to you can say $VDD - V_t$ and when A, B goes from 0 to 1, you discharge this particular node from $VDD - V_t$ to 0. So, kind of charging and discharging which is happening in the NAND gate, it depends on whether A is triggering the change in the output or B is triggering the change in the output. This will make the input transition to output transition delay dependent on the signals A and B which is not desirable.

So, what we would like to do is we want to make it symmetric with respect to the inputs. So, you can make it symmetric by using the symmetric gate where for each transition, same thing happens. So, I can do this, in place of using one branch, I will use two branches. So, they both are connected like this. So, then if the transition is triggered at Y from high to low, then whether A triggers the transition or B triggers the transition, it is going to be same from A and B to output. So, you can say this is like a symmetric NAND gate. It is possible to do in case when you have 2-input NAND gate. For 3-input NAND gate, it becomes little difficult because at some point of time, you cannot have all the possible combinations, with two, it is easy. So, here I have A, B, C and Y.

(Refer Slide Time: 10:07)



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

So, you will have a typical NAND gate which you are using like this which is A, B and C here, A, B and C, and then in place of using only one branch, we can use, see there are three positions, so total number of combinations are large, but using only single branch will have more errors or more variation in the transition delay. So, in place of using that, I will try to use another one where I can have A, B, C.

It will normally be B, C, A and then, so, each transistor occurs at the same position in the branch once, C, A, B. So, you can do this, and this is your Y output. Similarly, you can do for 4-input NAND gates. It is true that if we have these three locations and we would like to have all possible combinations, then you may use another three such branches to have all possible combinations but that is going to increase the parasitics and this becomes more cumbersome when you start designing for 4 inputs. So, somewhere you have to make peace with the transitions and then you can use it.

But ideally, if you ask me the question, then ideally if I do not want any variation with respect to the change at the input to the change in the output, if I do not want any change in the transition delays, then I should use all possible combinations like this. So, this completes our discussion on the NAND based PFDs. Thank you.