

Chapter 3

Clocked circuits

In the previous chapter we looked at how we could compute boolean functions. We are now ready to bring time back into the picture. In this chapter this is done by introducing an external measure of time: the clock.

3.1 Precharge logic

In Figure 3.1 we have depicted a clocksignal and its inverse. For now we assume that we have both signals available to us, and that the inverse of the clock is “really” the inverse, e.g., the two signals are each other’s exact opposite.

We first reexamine the circuits for our boolean functions, also called *combinational logic*. In the previous chapter it seemed as if we had to construct each function twice: once for the pullup and once for the pulldown circuit. If our signal needs to be valid all the time, there is not much we can do about

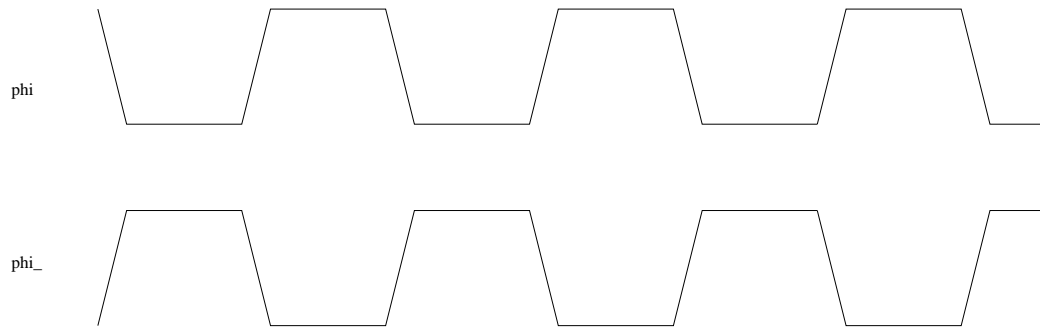


Figure 3.1: Clock and inverse clock.

this wastefulness (well ... there is another solution, but that one is wasteful in a different way ...). However, suppose you would need the signal only to become valid during a certain phase of the clock and stay valid during the remainder of the phase (the two halves of the cycle are called phases). If this is the case we can apply the technique depicted in Figure 3.2.

The idea here is that we use one phase, here when the clock is low, during which we don't care about the signal, to charge up the output through a p-transistor. While the output is being charged up to the high voltage we don't want the pulldown network to interfere, hence we use an n-transistor to cut the path to ground. When the clocksignal becomes high, the p-transistor is cut, the n-transistor is tied, and if there is a path to ground the output is pulled low. If there is no path to ground, as would be the case when the outcome of the boolean function was *true*, the output remains high, just like we would want.

This seems like a nice trick, we have saved almost half the area of our circuits! There are a number of things we need to be careful about though.

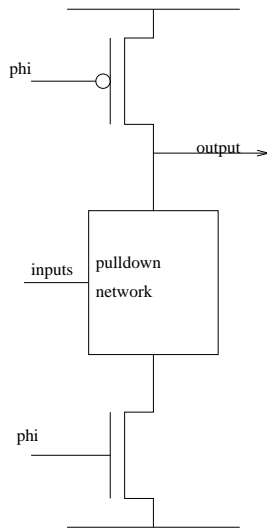


Figure 3.2: Basic idea of precharged circuits.

- First, the output has to look like a not too leaky capacitor, that is, the charge should remain on the output for at least a clockperiod if both the p-transistor and the pulldown network are cut.
- Second, the inputs to the pulldown circuit need to be either low or stable at the beginning of the compute phase (here when the clock is high). If some signals are high at the beginning of the compute phase that shouldn't be, there may be a path to ground that can discharge the output node before the inputs stabilize. This is a problem because the precharge transistor is off during the compute phase, hence the output node will not regain its high value, even if eventually all paths to ground are cut. If signals are low at the beginning of the compute phase that should have been high, and will stabilize at that value later during the compute phase, there is no problem. The output node may

discharge a little later, but if we have chosen our clockperiods long enough this is not a problem.

- Third, we have to prevent *static charge sharing*. Static charge sharing is the term we use to describe the situation where the charge on the output node may be redistributed to internal nodes of the circuit during the compute phase, and may thus become low(er), even if there is no path to ground. The problem is illustrated in Figure 3.3. Suppose both **a** and **b** are low during the precharge phase. Then, during the compute phase, **a** becomes high, effectively connecting the output and the intermediate node between the two upper n-transistors. (Remember, we said it was ok for the inputs to go from low to high during the compute phase.) If the internal node was low, for instance because **a** and **b** had both been high during a previous cycle, the charge on the output will now be redistributed over the output and the internal node. If the output capacitance is C_0 and the capacitance of the inner node C_1 the output voltage may drop to $V_h \cdot C_0 / (C_0 + C_1)$. The problem is especially severe if the internal node is shared between circuits, because this will give it greater capacitance.

Of course we can also build precharge logic with an n-transistor as the precharge transistor. The circuit that computes on the high clock phase is depicted along the original precharge circuit in Figure 3.6. The circuit that precharges low is not very often used, because it uses many, inherently weaker, p-transistors and only one n-transistor.

For the low-precharged circuit to work the inputs must be high or stable

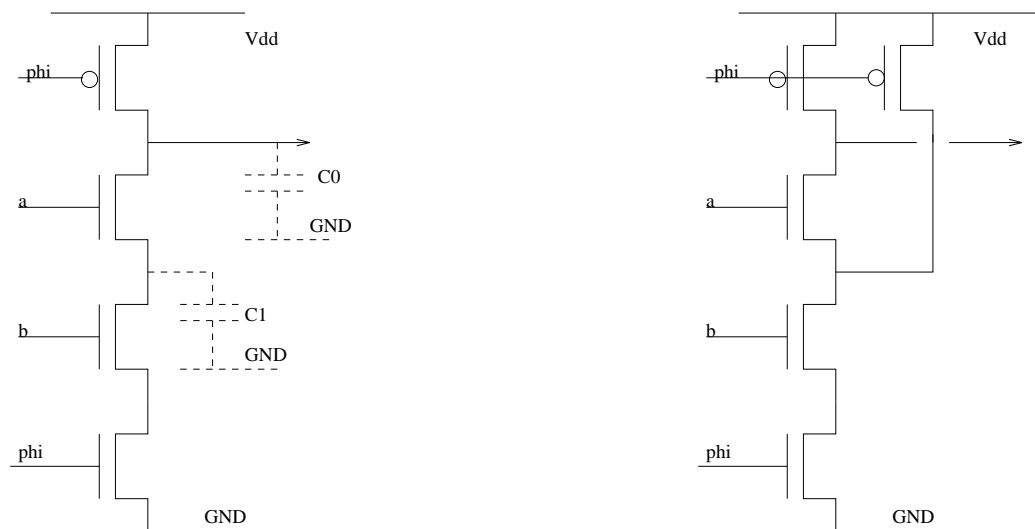


Figure 3.3: Static charge sharing in precharged NAND gate.

at the beginning of the compute phase (denoted $h \rightarrow x$ in Figure 3.6), and the outputs go from low to their stable value during the compute phase ($1 \rightarrow f(x)$). We see that the condition on the outputs of the circuits that precharge high, matches the condition on the inputs of the circuit that precharges low. We can therefore compose several alternating stages of these precharge circuits, this is called *domino logic*. As long as we meet the conditions on the output and input for the first and the last stage, keep our clock period long enough, and precharge the internal nodes this is a perfectly safe way to use precharge logic.

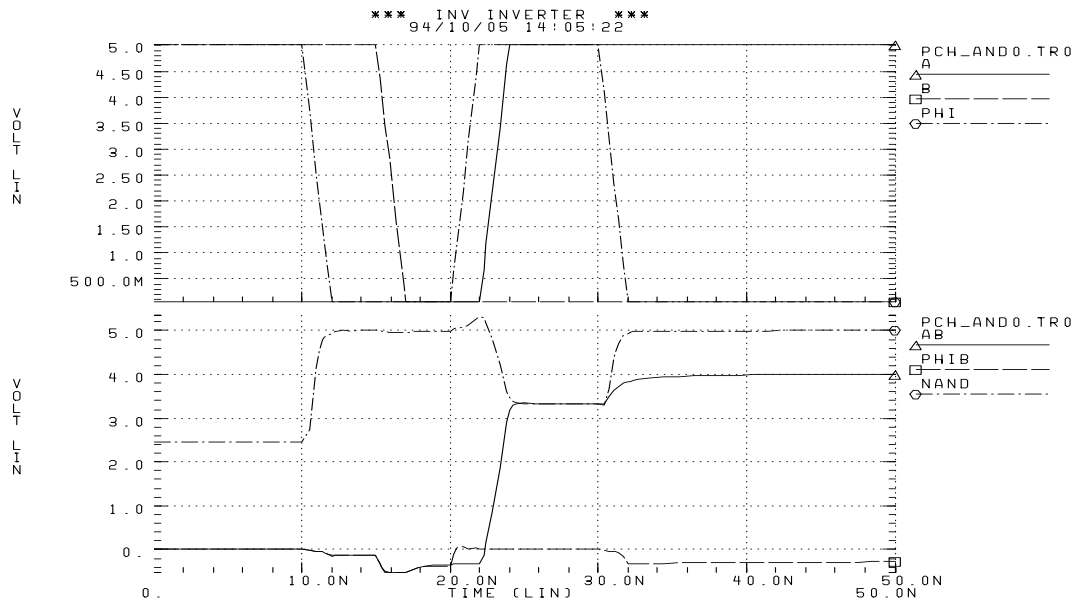


Figure 3.4: Static charge sharing in precharged NAND gate.

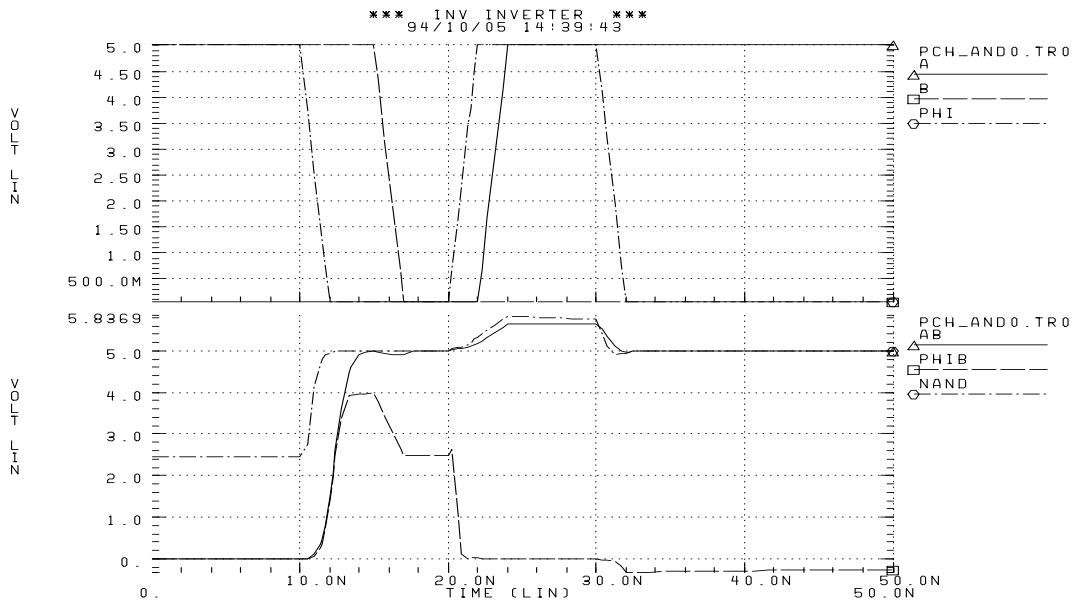


Figure 3.5: Static charge sharing solved.

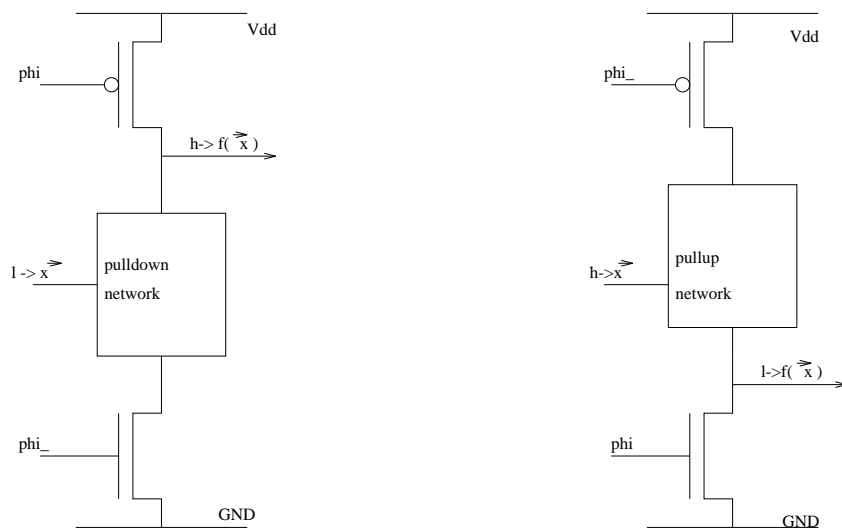


Figure 3.6: Composable precharge structures.

3.2 State-holding elements

Sometimes we need to hold values for longer than just one clockperiod. The basic circuit that allows us to do that is shown in Figure 3.7. It is just the same circuit as the inverter, except in that we have disconnected the two gates of the transistors. We now have four possible values for the (now two) inputs of this circuit. We avoid tying both the n and the p-transistor simultaneously, which leaves us with three combinations; tying just the n-transistor (reset), just the p-transistor (set), or neither (hold).

If the output goes to other gates only, the charge will sit there for a few milliseconds to a second (depending on the temperature amongst other things). Even though this corresponds to tens of thousands of cycles with today's clock speeds, this may still not be long enough. If we want the

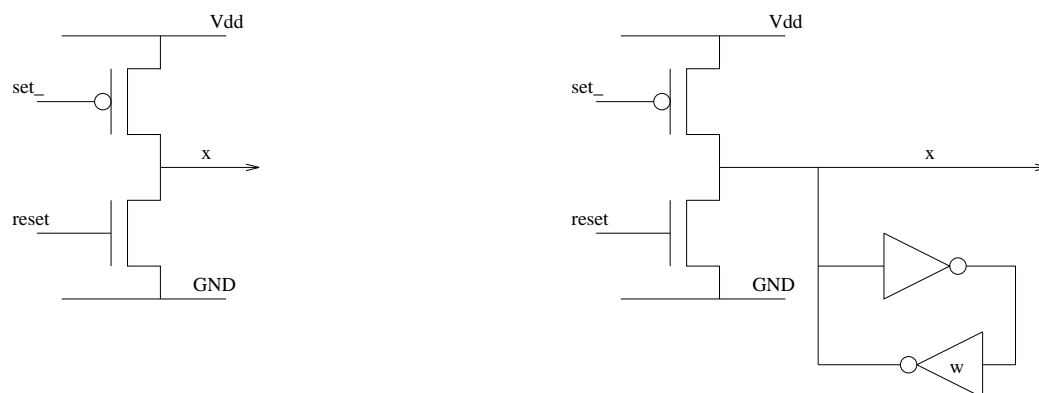


Figure 3.7: Basic state holding circuits in CMOS.

output to hold its value indefinitely, we add a staticizer, also shown in Figure 3.7. A staticizer is a pair of coupled inverters, with the inverter that may compete with the original state holding element so small that the set and reset transistors will always win. Since the output of an inverter is always driven, and because the pair of inverters is stable once the new value has gone around the loop once, this circuit can maintain its state as long as the high-voltage supply (Vdd) is maintained.

Also with this circuit we need to be a bit careful because charge sharing, albeit of a slightly different variety, can occur. The problem is illustrated in Figures 3.8 and 3.9. If the output of the state-holding element goes to the gate of a very large transistor (a bus driver, say) a significant capacitance exists, not only between the output and the substrate (GND for the n-transistor), but also between the output and the bus (the drain of the bus-driving n-transistor). Hence, if the bus is high when the state node is set, and then goes low when the output is not driven, the voltage on the output drops because

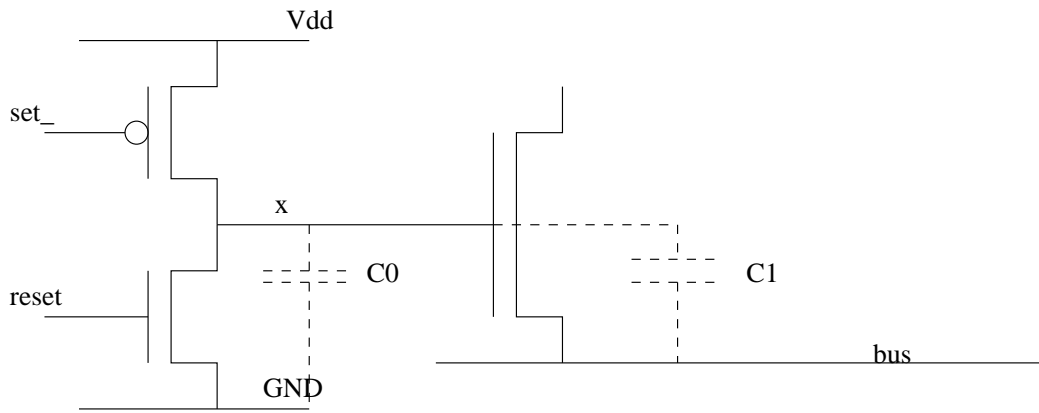


Figure 3.8: Dynamic charge sharing.

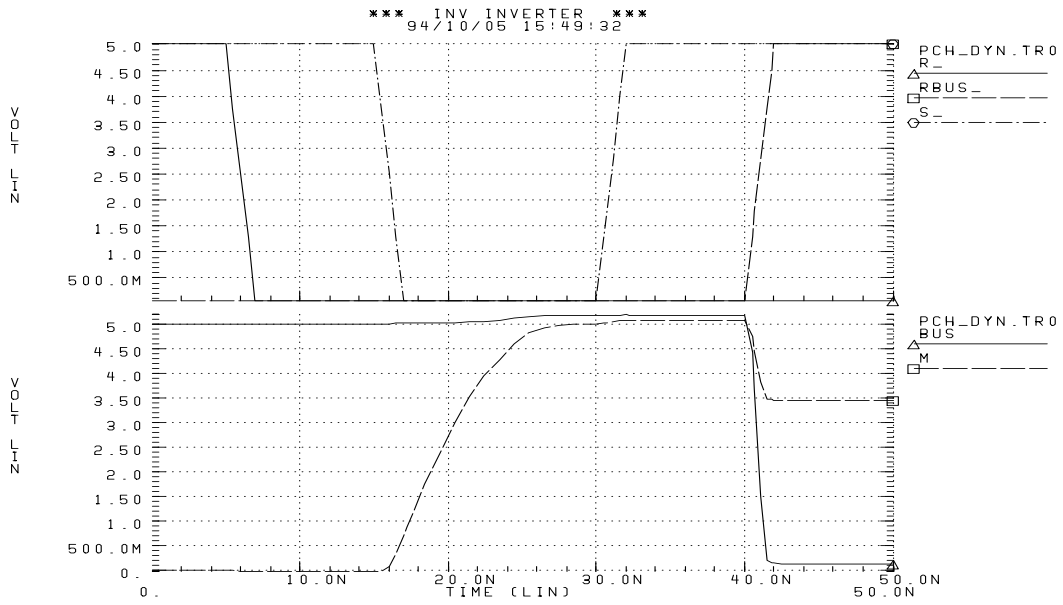


Figure 3.9: Dynamic charge sharing.

the capacitance to the bus suddenly counts. Here also, if the capacitance to ground is C_0 , and the capacitance to the bus C_1 the new voltage may be as low as $V_h \cdot C_0 / (C_0 + C_1)$. Unlike the situation with the internal nodes, there is a limit to how low the voltage can sink, because C_0 and C_1 are related. This kind of charge sharing is called *dynamic charge sharing*. Adding a staticizer tends to take care of the problem.

3.3 Nonoverlapping clock signals

In this section we introduce a slightly different kind of state-holding element. It is the little “switch” box depicted in Figure 3.10. It is a shorthand for a p and an n-transistor in parallel controlled with a signal and its negation, so that either both transistors are tied, or both are cut. Hence we can think of it as a voltage-controlled switch. We need both the n and the p-transistor to conduct both highs and lows well. We can also draw the symbol for a switch with an inverting bubble on its control port, which represents a switch that is cut when the depicted signal is high.

Switches work equally well in both directions (remember, the transistors are symmetric). We can give a direction to the switches by combining them with an inverter. Two of such combinations in series, controlled by a `shift` signal and its inverse, look like a shift-register cell. However, this is a very risky design, because in actuality there is a time when both `shift` and `shift_` are high-ish and both switches are somewhat open. This is like opening the sluice doors on one side before they are really closed on the other: not a very good idea!

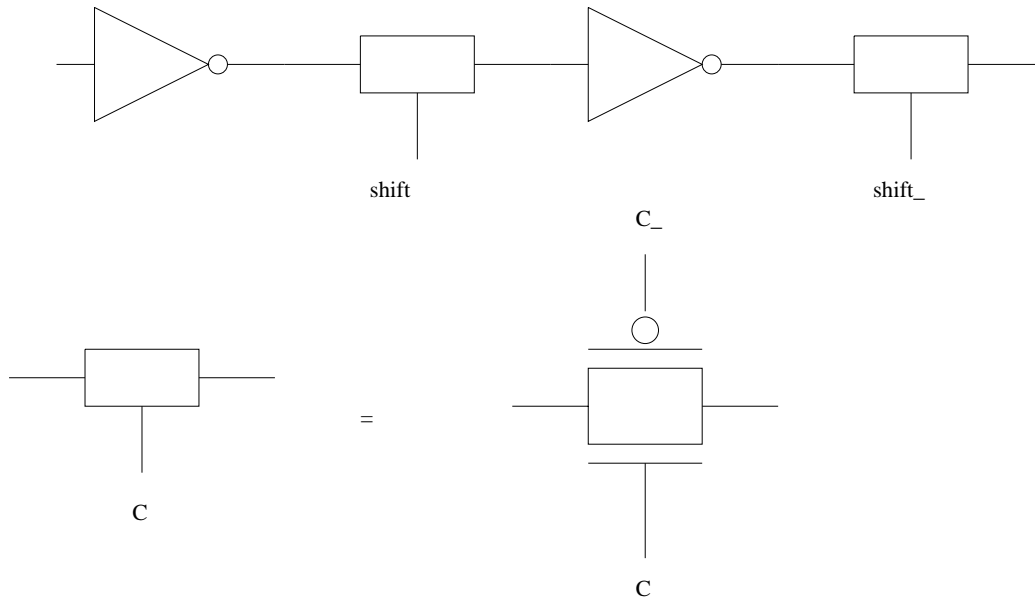


Figure 3.10: Unsafe shift register.

We solve this problem by introducing a different clock, one that has two phases that are guaranteed never to be high (or high-ish) at the same time. This allows us to construct a shift register cell where one of the two switches (sluice doors) is always cut. This shift register cell is depicted in Figure 3.12. We have also indicate a possible way of staticizing the design. If the second switch is controlled by `phi1` rather than `shift & phi1`, the second staticizer is not needed for clock rates higher than a kilohertz or so.

In Figure 3.13 we have indicated how we may transform the combination of an inverter and a switch to create a circuit that is functionally identical, but somewhat easier to lay out. One might be tempted to create an even more convenient (layout-wise) circuit by reversing the order of the transistors in the pullup and pulldown chains, but this is not a good idea, as it introduces

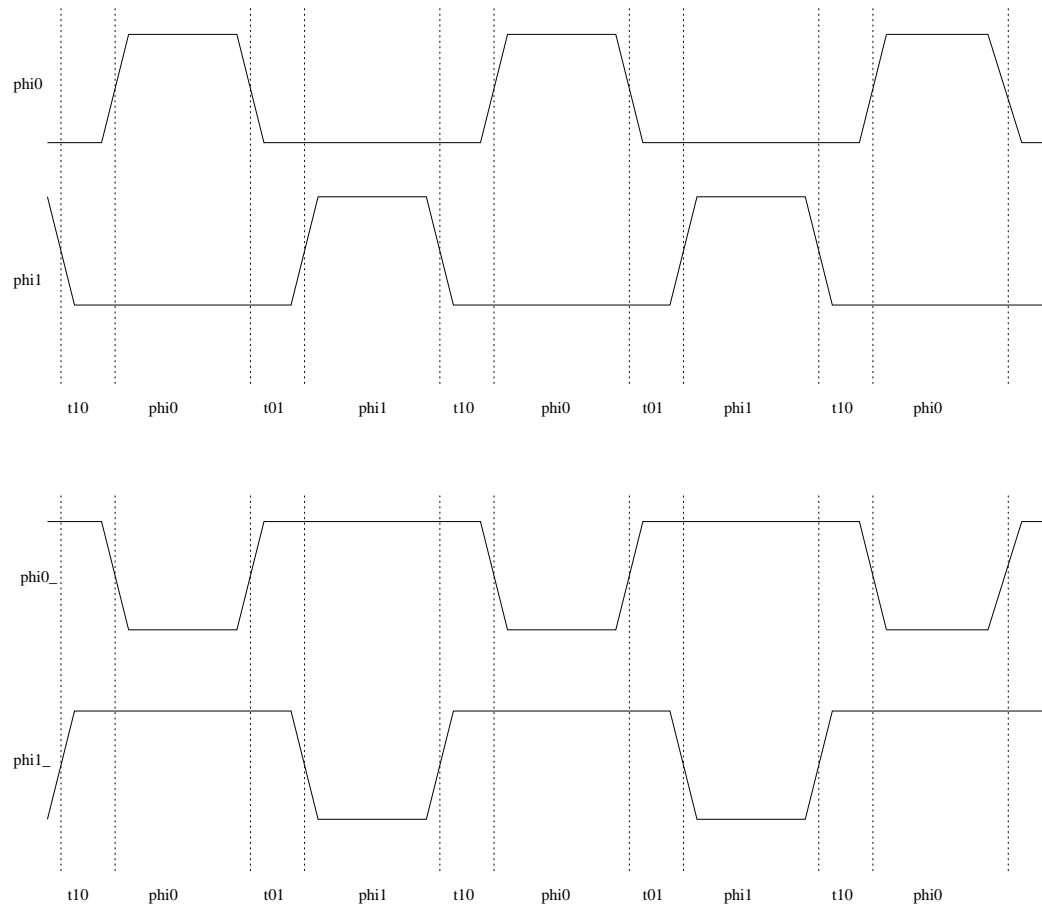


Figure 3.11: 2-phase nonoverlapping clocks.

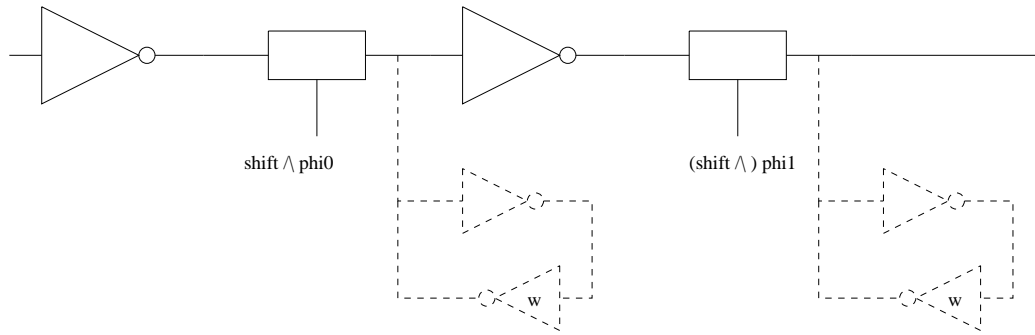


Figure 3.12: Safe shift register.

charge sharing problems.

We may also use the nonoverlapping clock to do precharge or domino logic. In Figure 3.14 it is indicated how circuits that precharge on phi0 and compute on phi1 may be designed.

We end this chapter with two register designs which combine some of the state holding techniques. The reader is encouraged to convince themselves of their correct operation.

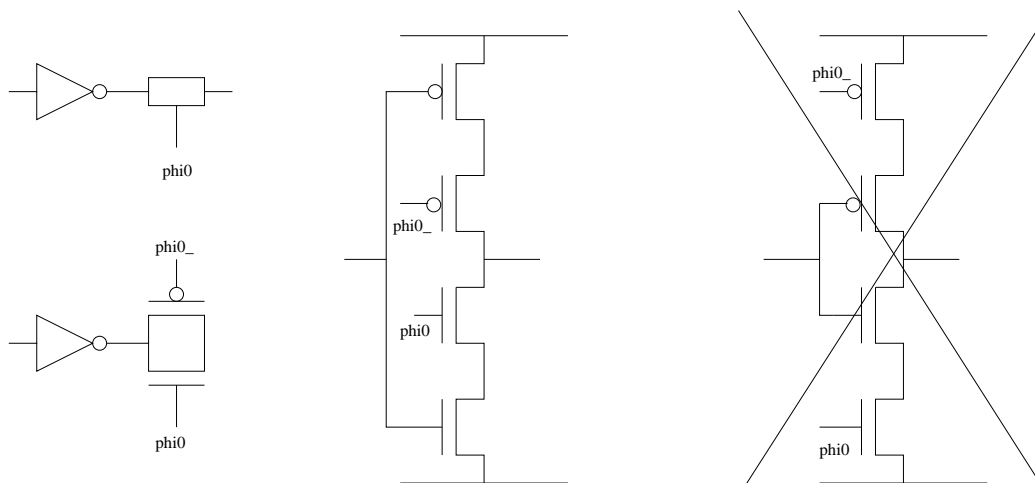


Figure 3.13: Transformations of (half a) shift register, charge sharing.

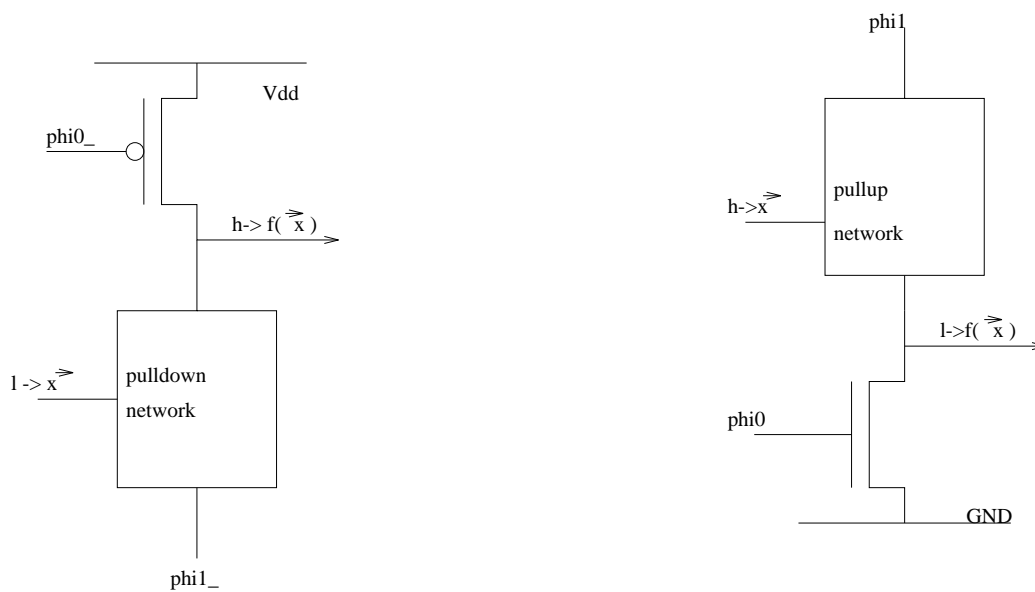


Figure 3.14: Precharge logic with nonoverlapping clock.

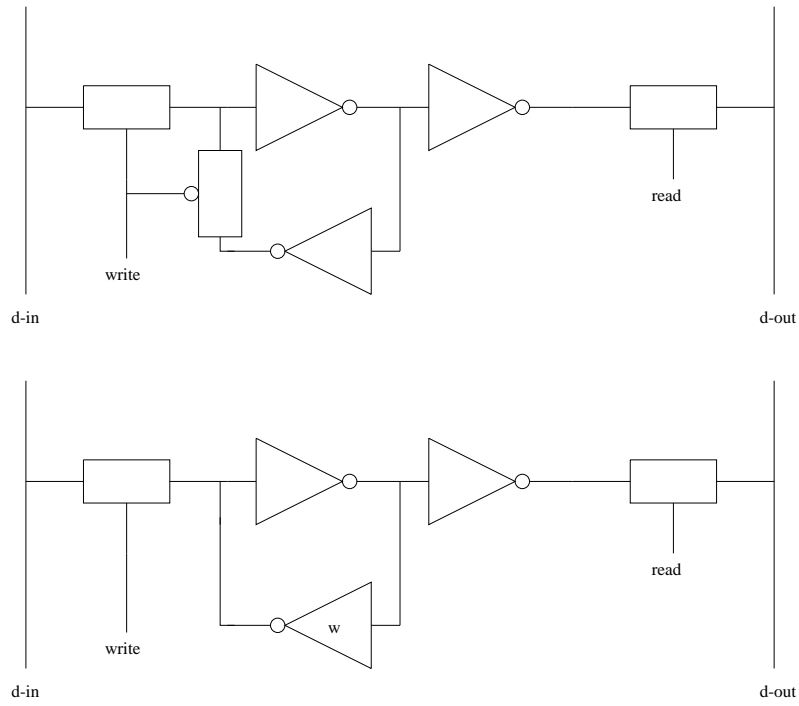


Figure 3.15: Two register designs.