Application

4.3 Show how De Morgan's Laws may be stated graphically.

Show graphically how your answer may be used to convert a three-layer {NOT AND OR} gate network to one comprising NAND alone, and a {NOT OR AND} network to one comprising NOR alone.

4.4 By adding columns to a truth-table, show that a three-input AND gate does indeed implement the same function as two two-input devices. Do the same for OR. Now try it with XOR.

As with AND and OR, you must first surmise what a three-input XOR gate might do.

4.5 How might a XOR gate be fabricated from complementary pairs of n-c/n-o switches?

Clue: Experiment with one input controlling both switches, while the other feeds through them. It's legitimate to employ NOT (or other) gates in the composition of another type.

Registers

Latch

While a bistable device to 'latch' a single bit can be made directly from complementary pairs, as shown in Figure 4.29, it will not be easy to use. For example, if we keep output and input separate, they will always represent the inverse of each other. Partly for this reason, and because we wish to build everything from standard components, we choose to make a one-bit latch from gates.

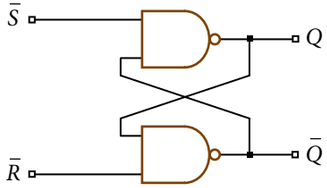

Figure 4.35 A *RS* latch from NAND gates.

Figure 4.35 shows how two NAND gates may be connected, with mutual feedback, to create a bistable. Like the earlier one, this bistable suffers a race condition, when both inputs are 0. Otherwise, it's well behaved, and can be 'set' via $\overline{S} = 0$ and 'reset' via $\overline{R} = 0$, with the alternate input 1.

Outputs Q and $\overline{Q}$ will complement each other, except when $\overline{S}\,\overline{R} = 00$.

The device has an interesting property in that it will retain its previous state when $SR = 11$.

A problem with the latch of Figure 4.35 is that we cannot simply connect a data signal. We have somehow to arrive at $\overline{S}$ and $\overline{R}$. The solution is to add another NAND gate, to invert our data signal D.

The additional gate affords a bonus: we can now "turn off" the device, using the spare input. Because it "turns on" the D input when asserted, convention bestows the name 'enable', or '*EN*', for short.

The complete D latch is shown in Figure 4.36.

Another problem remains: our latch is 'transparent', in that the input is 'visible' from the output. What we mean by this is that any change on the input will almost immediately appear as output. If we wish to chain devices together, to allow data to pass through a system, we need something more.

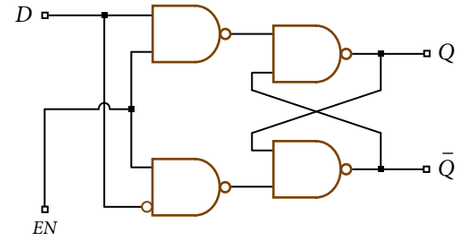

Figure 4.36 The (transparent) *D* latch.

Flip-flop

For one memory 'cell' to communicate with another requires each to possess *two* latches: one to register a datum for output, the other to record fresh input. If synchronous transfer is to take place on the tick of a common clock then we shall also need a 'tock' on which to move input to output. A "two-phase" clock is one which can provide both tick and tock. This need be nothing more than a wire on which the logic value changes from 0 to 1 and back, repeatedly. It can be generated by an unstable system with feedback, such as the one depicted in Figure 4.37.

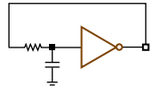

Figure 4.37 A digital clock.

The first 'inverter' charges (or discharges) a capacitance via a resistance. Once charged (discharged), the second will act and the output invert. The rate at which the clock ticks depends on how great both the resistance and capacitance are. System speed is dictated by the slowest 'clocked' component.

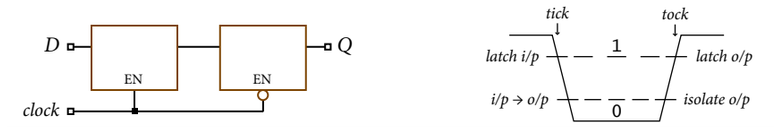

Figure 4.38 The (opaque) *D* flip-flop and its timing.

Figure 4.38 depicts our two-latch device, which is called a 'flip-flop' because of its two-phase action, along with a diagram showing its operation as the clock signal changes. Note how the "no-man's land" between the designated voltage range for each logic value is exploited to isolate the first latch from the second. The second one can only accept input *after* the first has been disabled.

Unlike a latch, a flip-flop is not transparent; input can change, but output will not follow until the next "clock cycle" is complete. The clocked "double buffer" ensures this.

A data register is just a collection of w flip-flops, where w is the 'word' width required. For example, a 64-bit register is simply an aggregate of sixty-four (1-bit) flip-flops, with no interaction between.

We can add an extra input gate to each flip-flop to allow both *clock* and *enable* register control.