

2.15 Starting A Sequence

Figure 2.16 shows that it is possible to use feedback to terminate a process. However, the circuit is still incomplete because it does not contain a mechanism that allows the sequence to start. Fortunately, adding a starting condition is trivial. To understand why, recall that a counter contains a separate input line that resets the count to zero. All that is needed to make our circuit start running is another input (e.g., from a button that a user pushes) connected to the counter reset.

When the user pushes the button, the counter resets to zero, which causes the counter's output to become 000. When it receives an input of all zeros, the decoder turns on the first output, and turns off the last output. When the last output turns off, the nand gate allows the clock pulses through, and the counter begins to run.

Although it does indeed start the sequence, allowing a user to reset the counter can cause problems. For example, consider what happens if a user becomes impatient during the startup sequence and presses the button a second time. Once the counter resets, the sequence starts again from the beginning. In some cases, performing an operation twice simply wastes time. In other cases, however, repeating an operation causes problems (e.g., some disk drives require that only one command be issued at a time). Thus, a production system uses complex combinatorial logic to prevent a sequence from being interrupted or restarted before it completes.

2.16 Iteration In Software Vs. Replication In Hardware

One of the fundamental differences between software and hardware arises from the way software and hardware handle operations that must be applied to a set of items. In software, the fundamental paradigm for handling multiple items consists of *iteration* — a programmer writes code that repeatedly finds the next item in a set and applies the operation to the item. That is, because the underlying system can only apply the operation to one item at a time, a programmer must explicitly specify the number of items in the set and the order in which they are to be processed. Iteration is so essential to programming that most programming languages provide a compact syntax that allows the programmer to express the iteration clearly (e.g., a *for loop*).

Although hardware can be built to perform iteration, doing so is difficult, and the resulting hardware is clumsy. Instead, the fundamental hardware paradigm for handling multiple items consists of *replication* — a hardware engineer creates multiple copies of the underlying gates, and allows each copy to act on one item. For example, suppose we need to compute a Boolean operation on a set of thirty-two Boolean values. The ideal hardware solution consists of replicating the necessary gate thirty-two times, and allowing each instance to operate on one of the thirty-two items. For example, to compute the Boolean *not* of thirty-two values, a hardware designer might use thirty-two inverters.