3e8.14 Cntr.1 The only difference between a ’163 and a ’161 is that the CLR_L input of ’161 is asynchronous. Thus, the counter will go from state 1010 to state 0000 immediately, before the next clock tick, and go from state 0000 to state 0001 at the clock tick. Observing the state just before each clock tick, it is therefore a modulo-10 counter, with the counting sequence 0, 1, ..., 9, 0, 1, ....

Note that this type of operation is not recommended, because the width of the CLR_L pulse is not well controlled. That is, the NAND gate will negate the CLR_L pulse as soon as either one of its inputs goes to 0. If, say, the counter’s QB output clears quickly, and its QD output clears slowly, it is possible for CLR_L to be asserted long enough to clear QB but not QD, resulting in an unexpected next state of 1000, or possibly metastability of the QD output.

3e8.81 Cntr.3 This problem can be a bit confusing since the states in Table 7–11 have the same names as the ’163 data inputs. Therefore, we shall use the names SA, SB, and so on for the states.

The idea is to normally allow the counter to count to the next state, but to force it to go to SA or SB when the wrong input is received. The CLR_L input is used to go to SA (0000), and the LD_L input is used to go to SB (0001; the counter’s A–D data inputs are tied LOW and HIGH accordingly).

Inspecting the state table on page 582 of the text, we see that state A should be loaded when X=1 and the machine is in state SA, SD, or SH. Thus,

$$\begin{aligned} \text{CLR\_L} &= [X\cdot(QC'\cdot QB'\cdot QA' + QC'\cdot QB\cdot QA + QC\cdot QB\cdot QA)]' \\ &= [X\cdot(QC'\cdot QB'\cdot QA + QB\cdot QA)]' \end{aligned}$$

All of the other next-states when X=1 are the natural successors obtained by counting.

Similarly, state SB should be loaded when X=0 and the machine is in state SB, SC, SE, SF, or SH. In addition, notice that in state SG, the next state SE is required when X=0; the load input must be used in this case too, but a different value must be loaded. Thus, LD_L will be asserted in six of the eight states when X=0.

Optionally, LD_L could be easily asserted in the remaining two states as well, since the required next states (SB and SE) are ones that we must generate in other six cases anyway. Thus, we can connect X to the LD_L input, so we always load when X=0. Then, we load either SB (0010) or SE (0100) depending on the current state. Therefore, we can write the following equations for data inputs A–D:

$$\begin{aligned} A &= 0 \\ B &= SA + SB + SC + SE + SF + SH \\ &= QB' + QC'\cdot QA' + QC\cdot QA \\ C &= B' \\ D &= 0 \end{aligned}$$

Alternatively, we could realize C as an AND-OR circuit and complement to get B:

$$\begin{aligned} C &= QC'\cdot QB\cdot QA + QC\cdot QB\cdot QA' \\ B &= C' \end{aligned}$$

The logic diagram follows directly from these equations. The output logic can be realized using the equations on page 570 of the text.