Problem 7.12 The sequence for the Johnson counter gives a state transition table which looks like this:

	Q3	Q2	Q1	Q0	Q3+	Q2+	Q1+	Q0+		T3	T2	T1	T0
0	0	0	0	0	1	0	0	0	8	1	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	1
2	0	0	1	0	x	x	x	x	x	x	x	x	x
3	0	0	1	1	0	0	0	1	1	0	0	1	0
4	0	1	0	0	x	x	x	x	x	x	x	x	x
5	0	1	0	1	x	x	x	x	x	x	x	x	x
6	0	1	1	0	x	x	x	x	x	x	x	x	x
7	0	1	1	1	0	0	1	1	3	0	1	0	0
8	1	0	0	0	1	1	0	0	12	0	1	0	0
9	1	0	0	1	x	x	x	x	x	x	x	x	x
10	1	0	1	0	x	x	x	x	x	x	x	x	x
11	1	0	1	1	x	x	x	x	x	x	x	x	x
12	1	1	0	0	1	1	1	0	14	0	0	1	0
13	1	1	0	1	x	x	x	x	x	x	x	x	x
14	1	1	1	0	1	1	1	1	15	0	0	0	1
15	1	1	1	1	0	1	1	1	7	1	0	0	0

D3 (Q3, Q0, Q1, Q2)

1	x	1	1
0	x	x	x
0	0	0	x
x	x	1	x

D2 (Q3, Q0, Q1, Q2)

0	x	1	1
0	x	x	x
0	0	1	x
x	x	1	x

D1 (Q3, Q0, Q1, Q2)

0	x	1	0
0	x	x	x
0	1	1	x
x	x	1	x

D0 (Q3, Q0, Q1, Q2)

0	x	0	0
0	x	x	x
1	1	1	x
x	x	1	x

T3 (Q3, Q0, Q1, Q2)

1	x	0	0
0	x	x	x
0	0	1	x
x	x	0	x

T2 (Q3, Q0, Q1, Q2)

0	x	0	1
0	x	x	x
0	1	0	x
x	x	0	x

T1 (Q3, Q0, Q1, Q2)

0	x	1	0
0	x	x	x
1	0	0	x
x	x	0	x

T0 (Q3, Q0, Q1, Q2)

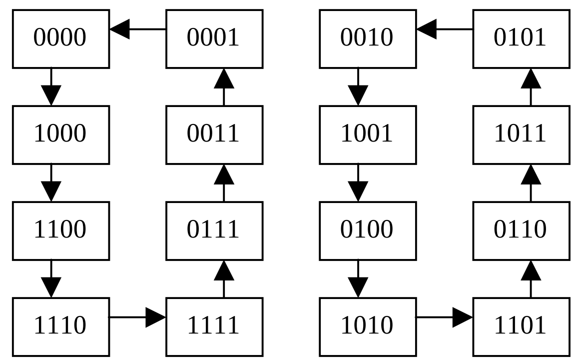
0	x	0	0
1	x	x	x
0	0	0	x
x	x	1	x

D3=\Q0
D2=Q3
D1=Q2
D0=Q1

T3=Q3Q0+\Q3\Q0=Q3 XOR \Q0
T2=Q3\Q2+\Q3Q2=Q2 XOR Q3
T1=Q2\Q1+\Q2Q1=Q1 XOR Q2
T0=Q1\Q0+\Q1Q0=Q0 XOR Q1

7.13 and 7.12 b) After we do the K-maps and implementation for 7.12, we plug in for what the “don’t cares” turned into to get the actual values for Di and Ti. For Di, this is enough information to immediately to draw the full state transition graph. For the T implementation, we still have to figure out what the next states will be given Qi and Ti. Once we have the implemented values of the “don’t cares”, we can draw the state transition graphs. Amazingly, it turns out that they are both the same! On second look, this turns out not to be so surprising. We’re implementing a Johnson counter, or Mobius counter, which is basically just a shift register with the output inverted and fed back into the input. It’s not surprising then that the minimum logic for the D implementation is just that: D3=\Q0, otherwise Di=Q(i+1). For the T implementation, we find that they follow a similar pattern: T3=\Q0 XOR Q3, otherwise Ti = Q(i+1) XOR Qi, which means that we’ve just turned the T flip flops into D flipflops by XORing an input with their current state. Comparing the implementations, clearly the D is the simplest, followed by the JK (which, like the T implementation, is just converting JKs into Ds), and the T implementation takes 4 gates compared to zero for D and JK. JK takes twice as many wires as D. Bottom line: if you’re building a shift register, use D flip flops.

	Q3	Q2	Q1	Q0	Q3+	Q2+	Q1+	Q0+
0	0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	1	0	1	0	0	1
3	0	0	1	1	0	0	0	1
4	0	1	0	0	1	0	1	0
5	0	1	0	1	0	0	1	0
6	0	1	1	0	1	0	1	1
7	0	1	1	1	0	0	1	1
8	1	0	0	0	1	1	0	0
9	1	0	0	1	0	1	0	0
10	1	0	1	0	1	1	0	1
11	1	0	1	1	0	1	0	1
12	1	1	0	0	1	1	1	0
13	1	1	0	1	0	1	1	0
14	1	1	1	0	1	1	1	1
15	1	1	1	1	0	1	1	1



	Q3	Q2	Q1	Q0	T3	T2	T1	T0	Q3+	Q2+	Q1+	Q0+
0	0	0	0	0	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	0
2	0	0	1	0	1	0	1	1	1	0	0	1
3	0	0	1	1	0	0	1	0	0	0	0	1
4	0	1	0	0	1	1	1	0	1	0	1	0
5	0	1	0	1	0	1	1	1	0	0	1	0
6	0	1	1	0	1	1	0	1	1	0	1	1
7	0	1	1	1	0	1	0	0	0	0	1	1
8	1	0	0	0	0	1	0	0	1	1	0	0
9	1	0	0	1	1	0	1	1	0	0	1	0
10	1	0	1	0	0	1	1	1	1	1	0	1
11	1	0	1	1	1	1	1	0	0	1	0	1
12	1	1	0	0	0	0	1	0	1	1	1	0
13	1	1	0	1	1	0	1	1	0	1	1	0
14	1	1	1	0	0	0	0	1	1	1	1	1
15	1	1	1	1	1	0	0	0	0	1	1	1