# Problem 2

Design a dual-function counter. The counter has one control input, *x*, and a two-bit state. When *x* = 0, the counter should function as a standard-up counter, that is, it should cycle through the states 00, 01, 10, 11. When *x* = 1, the counter should function as a Gray Code counter, that is, it should cycle through the states 00, 01, 11, 10.

**Step 1: Arrange the present states.**

Once again, we begin by organizing the present state information. Instead of creating a state diagram, we will organize the information in the state table according to the description of the present and next state information as given in the specification.

The primary difference between this problem and the previous one is the inclusion of the control input *x*. In general, a given present state will give two different next states, depending on whether the value of *x* is 0 or 1. Since that is the case, the value of *x* must be considered as a part of the **total present state**.This requires an adjustment to the state table that was used previously.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Present State | | | Next State | | D inputs | |
| *x* | A | B | A+ | B+ | DA | DB |
| 0 | 0 | 0 |  |  |  |  |
| 0 | 0 | 1 |  |  |  |  |
| 0 | 1 | 0 |  |  |  |  |
| 0 | 1 | 1 |  |  |  |  |
| 1 | 0 | 0 |  |  |  |  |
| 1 | 0 | 1 |  |  |  |  |
| 1 | 1 | 0 |  |  |  |  |
| 1 | 1 | 1 |  |  |  |  |

By organizing the table as we have, with *x* as the most significant input, the table can be divided into two sections. The first section contains all possible present states when the input *x* = 0. The second section contains all possible present states when the input *x* = 1. Since the specification calls for a different sequence of states, i.e., a different set of next states for each set of present states, when the value of *x* changes, we can easily provide the information for each of the two cases of the counter’s operation.

**Step 2: Arrange the next states.**

As indicated previously, the division of the table makes it easy for us to consider the state machine according to each of its functions. With that being the case, we can fill in each half of the table according to the sequence that we know to be followed for a particular control input value. We obtain the table as shown below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Present State | | | Next State | | D inputs | |
| *x* | A | B | A+ | B+ | DA | DB |
| 0 | 0 | 0 | 0 | 1 |  |  |
| 0 | 0 | 1 | 1 | 0 |  |  |
| 0 | 1 | 0 | 1 | 1 |  |  |
| 0 | 1 | 1 | 0 | 0 |  |  |
| 1 | 0 | 0 | 0 | 1 |  |  |
| 1 | 0 | 1 | 1 | 1 |  |  |
| 1 | 1 | 0 | 0 | 0 |  |  |
| 1 | 1 | 1 | 1 | 0 |  |  |

In the upper half of the state table, the next states correspond to those that would form the standard counting sequence. In the lower half of the table, the next states correspond to those that would form a Gray Code counting sequence.

**Step 3: Obtain the excitation information.**

We can shade this table just as we did the last one, so that we can keep track of how to compare the next states to the present states, and how to fill in the corresponding excitation information.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Present State | | | Next State | | D inputs | |
| *x* | A | B | A+ | B+ | DA | DB |
| 0 | 0 | 0 | 0 | 1 |  |  |
| 0 | 0 | 1 | 1 | 0 |  |  |
| 0 | 1 | 0 | 1 | 1 |  |  |
| 0 | 1 | 1 | 0 | 0 |  |  |
| 1 | 0 | 0 | 0 | 1 |  |  |
| 1 | 0 | 1 | 1 | 1 |  |  |
| 1 | 1 | 0 | 0 | 0 |  |  |
| 1 | 1 | 1 | 1 | 0 |  |  |

Remember, the darker columns show the state transition and excitation information for flip-flop A, and the lighter columns show the same information for flip-flop B.

The table is filled using the same excitation information as was used in the previous example. The correct excitation information can be inserted into the table for the proper set of transitions. (A → A+, B → B+) Here is the state table filled in with that information:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Present State | | | Next State | | D inputs | |
| *x* | A | B | A+ | B+ | DA | DB |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

**Step 4: Obtain the logic equations.**

With the excitation information complete, each input must now be mapped, so that a set of logic equation can be determined. With the exception of needing 3-variable Karnaugh maps instead of the 2-variable maps of the previous example, the process of obtaining the logic equations is the same as in the first example.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DA |  | AB | | | |  | DB |  | AB | | | |
|  |  | 00 | 01 | 11 | 10 |  |  |  | 00 | 01 | 11 | 10 |
| *x* | 0 | 0 | 1 | 0 | 1 |  | *x* | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |  | 1 | 1 | 1 | 0 | 0 |
|  |  | DA = A’B + *x*B + *x*’AB’ | | | |  |  |  | DB = *x*’B’ + *x*A’ | | | |

**END OF PROBLEM 2**