When trying to minimize a sum-of-products expression using the reduction identity, our goal is to find two product terms that can be written as one smaller product term, eliminating the "don't-care" variable.

This is easy to do when two the product terms come from adjacent rows in the truth table.

For example, look at the bottom two rows in this truth table.

Since the Y output is 1 in both cases, both rows will be represented in the sum-of-products expression for this function.

It's easy to spot the don't care variable: when C and B are both 1, the value of A isn't needed to determine the value of Y.

Thus, the last two rows of the truth table can be represented by the single product term (B AND C).

Finding these opportunities would be easier if we reorganized the truth table so that the appropriate product terms were on adjacent rows.

That's what we've done in the Karnaugh map, K-map for short, shown on the right.

The K-map organizes the truth table as a two-dimensional table with its rows and columns labeled with the possible values for the inputs.

In this K-map, the first row contains entries for when C is 0 and the second row contains entries for when C is 1.

Similarly, the first column contains entries for when A is 0 and B is 0.

And so on.

The entries in the K-map are exactly the same as the entries in the truth table, they're just formatted differently.

Note that the columns have been listed in a special sequence that's different from the usual binary counting sequence.

In this sequence, called a Gray Code, adjacent labels differ in exactly one of their bits.

In other words, for any two adjacent columns, either the value of the A label changed, or the value of the B label changed.

In this sense, the leftmost and rightmost columns are also adjacent.

We write the table as a two-dimensional matrix, but you should think of it as cylinder with its left and right edges

touching.

If it helps you visualize which entries are adjacent, the edges of the cube shows which 3-bit input values differ by only one bit.

As shown by the red arrows, if two entries are adjacent in the cube, they are also adjacent in the table.

It's easy to extend the K-map notation to truth tables for functions with 4 inputs, as shown here.

We've used a Gray code sequencing for the rows as well as the columns.

As before, the leftmost and rightmost columns are adjacent, as are the top and bottom rows.

Again, as we move to an adjacent column or an adjacent row, only one of the four input labels will have changed.

To build a K-map for functions of 6 variables we'd need a 4x4x4 matrix of values.

That's hard to draw on the 2D page and it would be a challenge to tell which cells in the 3D matrix were adjacent.

For more than 6 variables we'd need additional dimensions.

Something we can handle with computers, but hard for those of us who live in only a three-dimensional space!

As a practical matter, K-maps work well for up to 4 variables, and we'll stick with that.

But keep in mind that you can generalize the K-map technique to higher dimensions.

So why talk about K-maps?

Because patterns of adjacent K-map entries that contain 1's will reveal opportunities for using simpler product terms in our sum-of-products expression.

Let's introduce the notion of an implicant, a fancy name for a rectangular region of the K-map where the entries are all 1's.

Remember when an entry is a 1, we'll want the sum-of-products expression to evaluate to TRUE for that particular combination of input values.

We require the width and length of the implicant to be a power of 2, i.e., the region should have 1, 2, or 4 rows, and 1, 2, or 4 columns.

It's okay for implicants to overlap.

We say that an implicant is a prime implicant if it is not completely contained in any other implicant.

Each product term in our final minimized sum-of-products expression will be related to some prime implicant in the K-map.

Let's see how these rules work in practice using these two example K-maps.

As we identify prime implicants, we'll circle them in red.

Starting with the K-map on the left, the first implicant contains the singleton 1-cell that's not adjacent to any other cell containing 1's.

The second prime implicant is the pair of adjacent 1's in the upper right hand corner of the K-map.

This implicant is has one row and two columns, meeting our constraints on an implicant's dimensions.

Finding the prime implicants in the right-hand K-map is a bit trickier.

Recalling that the left and right columns are adjacent, we can spot a 2x2 prime implicant.

Note that this prime implicant contains many smaller 1x2, 2x1 and 1x1 implicants, but none of those would be prime implicants since they are completely contained in the 2x2 implicant.

It's tempting draw a 1x1 implicant around the remaining 1, but actually we want to find the largest implicant that contains this particular cell.

In this case, that's the 1x2 prime implicant shown here.

Why do we want to find the largest possible prime implicants?

We'll answer that question in a minute… Each implicant can be uniquely identified by a product term, a Boolean expression that evaluates to TRUE for every cell contained within the implicant and FALSE for all other cells.

Just as we did for the truth table rows at the beginning of this chapter, we can use the row and column labels to help us build the correct product term.

The first implicant we circled corresponds to the product term (not A) AND (not B) AND C, an expression that evaluates to TRUE when A is 0, B is 0, and C is 1.

How about the 1x2 implicant in the upper-right hand corner?

We don't want to include the input variables that change as we move around in the implicant.

In this case the two input values that remain constant are C (which has the value 0) and A (which has the value 1), so the corresponding product term is A AND (not C).

Here are the two product terms for the two prime implicants in the right-hand K-map.

Notice that the larger the prime implicant, the smaller the product term!

That makes sense: as we move around inside a large implicant, the number of inputs that remain constant across the entire implicant is smaller.

Now we see why we want to find the largest possible prime implicants: they give us the smallest product terms!

Let's try another example.

Remember that we're looking for the largest possible prime implicants.

A good way to proceed is to find some un-circled 1, and then identify the largest implicant we can find that incorporates that cell.

There's a 2x4 implicant that covers the middle two rows of the table.

Looking at the 1's in the top row, we can identify two 2x2 implicants that include those cells.

There's a 4x1 implicant that covers the right column, leaving the lonely 1 in the lower left-hand corner of the table.

Looking for adjacent 1's and remembering the table is cyclic, we can find a 2x2 implicant that incorporates this last un-circled 1.

Notice that we're always looking for the largest possible implicant, subject to constraint that each dimension has to be either 1, 2 or 4.

It's these largest implicants that will turn out to be prime implicants.

Now that we've identified the prime implicants, we're ready to build the minimal sum-of-products expression.

Here are two example K-maps where we've shown only the prime implicants needed to cover all the 1's in the map.

This means, for example, that in the 4-variable map, we didn't include the 4x1 implicant covering the right column.

That implicant was a prime implicant since it wasn't completely contained by any other implicant, but it wasn't needed to provide a cover for all the ones in the table.

Looking at the top table, we'll assemble the minimal sum-of-products expression by including the product terms for each of the shown implicants.

The top implicant has the product term A AND (not C), and the bottom implicant has the product term (B AND C).

And we're done!

Why is the resulting equation minimal?

If there was some further reduction that could be applied, to produce a yet smaller product term, that would mean there was a larger prime implicant that could have been circled in the K-map.

Looking the bottom table, we can assemble the sum-of-products expression term-by-term.

There were 4 prime implicants, so there are 4 product terms in the expression.

And we're done.

Finding prime implicants in a K-map is faster and less error-prone that fooling around with Boolean algebra identities.

Note that the minimal sum-of-products expression isn't necessarily unique.

If we had used a different mix of the prime implicants when building our cover, we would have come up with different sum-of-products expression.

Of course, the two expressions are equivalent in the sense that they produce the same value of Y for any particular combination of input values - they were built from the same truth table after all.

And the two expressions will have the same number of operations.

So when you need to come with up a minimal sum-of-products expression for functions of up to 4 variables, K-maps are the way to go!

We can also use K-maps to help us remove glitches from output signals.

Earlier in the chapter we saw this circuit and observed that when A was 1 and B was 1, then a 1-to-0 transition on C might produce a glitch on the Y output as the bottom product term turned off and the top product term turned on.

That particular situation is shown by the yellow arrow on the K-map, where we're transitioning from the cell on the bottom row of the 1-1 column to the cell on the top row.

It's easy to see that we're leaving one implicant and moving to another.

It's the gap between the two implicants that leads to the potential glitch on Y.

It turns out there's a prime implicant that covers the cells involved in this transition - shown here with a dotted red outline.

We didn't include it when building the original sum-of-products implementation since the other two product terms provided the necessary functionality.

But if we do include that implicant as a third product term in the sum-of products, no glitch can occur on the Y output.

To make an implementation lenient, simply include all the prime implicants in the sum-of-products expression.

That will bridge the gaps between product terms that lead to potential output glitches.