Below is a step‐by‐step look at what the slide is showing and how one would analyze it. The core design problem is:

**Given a 3‐bit input** , **produce a 3‐bit output** **which equals “input − 2.”**

Because we are working in only 3 bits, the outputs for small inputs become negative in 2’s‐complement form. For example, “0 − 2” (decimal) is if interpreted as in 3‐bit 2’s‐complement.

**1. Construct the Truth Table**

List out all eight input combinations from to in decimal, then compute in 3‐bit 2’s‐complement:

|  |  |  |  |
| --- | --- | --- | --- |
| **Decimal** |  | **“”** | (3‐bit 2’s‐complement) |
| 0 | 000 |  | 110 |
| 1 | 001 |  | 111 |
| 2 | 010 | 0 | 000 |
| 3 | 011 | 1 | 001 |
| 4 | 100 | 2 | 010 |
| 5 | 101 | 3 | 011 |
| 6 | 110 | 4 | 100 |
| 7 | 111 | 5 | 101 |

In other words:

* → output
* → output
* → output
* … and so on.

**2. Fill the K‐Map(s)**

A **Karnaugh map** (K‐map) for a 3‐bit input is a 2×4 or 4×2 grid in **Gray code** order.  
One common layout (columns = , row = ) is:

X2X1  
 00 01 11 10  
X0  
0 ? ? ? ?  
1 ? ? ? ?

* **Columns** go 00 → 01 → 11 → 10 so that each step flips only one bit.
* **Rows** go then .

Because the output itself is **3 bits** (), you actually fill **three** separate K‐maps (one per output bit).

**(a) K‐Map for**

From the truth table:

That corresponds to

Placing 1’s in those K‐map cells and 0 in the rest, you find that is 1 in two adjacent columns ( and ) for **both** and . The largest group covers those entire two columns, yielding a simplified expression:

which is the “equivalence” function (i.e. exactly when and match).

**(b) K‐Map for**

From the table:

In terms of , that is Notice that those all have . Hence:

**(c) K‐Map for**

From the table:

Those are exactly the cases where . Hence:

**3. Final Simplified Logic**

Putting it all together:

If you wire up a 3‐bit combinational circuit whose outputs follow those three equations, it implements “” under 3‐bit 2’s‐complement arithmetic.

**Takeaways**

1. **K‐Map Setup**
   * Label rows/columns in Gray code so adjacent cells differ by only one input bit.
2. **Filling Outputs**
   * For each possible input, compute “” in 3‐bit 2’s‐complement and mark the corresponding 1/0 in the map for each output bit.
3. **Grouping and Simplifying**
   * Combine adjacent 1‐cells to get minimal SOP (Sum of Products). Here, we see the neat patterns:
     + “equivalence” of and .
     + is simply .
     + is simply .

Hence, the slide is illustrating how to go from the truth table for “input minus 2” to a Karnaugh map, and then derive the simplest Boolean expressions for each output bit.