Lab 04: I/O and POSTNET Hardware Implementation

Arturo Salinas-Aguayo
CSE 2301: Principles and Practice of Digital Logic Design
Dr. Mohammad Khan, Section 003L-1248
Electrical and Computer Engineering Department

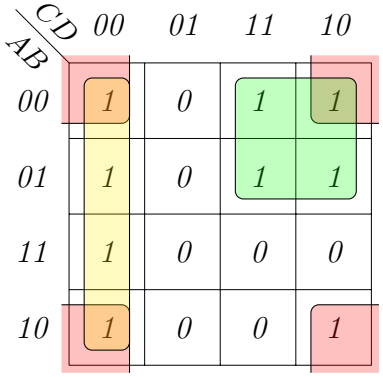
College of Engineering, University of Connecticut
Coded in LaTeX

Theory

Karnaugh Maps in Depth

Looking back at last week's answer,

Example 1 ***Karnaugh Map for*** $\sum m(0, 2, 3, 4, 6, 7, 8, 10, 12)$



This produces:

$$F(A, B, C, D) = \overline{A}C + \overline{B}\,\overline{D} + \overline{C}\,\overline{D}$$

$$F(A, B, C, D) = \overline{A}C + \overline{D}(\overline{B} + \overline{C}) \quad \textit{(Distribution)}$$

Karnaugh maps, are a tabular, graphical way to simplify Boolean expressions by grouping terms into a grid based on their truth tables. Each square in a K-map represents a **minterm**, which corresponds to a *unique* combination of variable values. These minterms are placed such that adjacent squares differ only by one variable, which allows for the user to easily identify opportunities for simplification. These are ordered in a Grey code.

There are actually multiple ways to organize a K-map, but the most general process involves circling adjacent squares that contain a 1, or **TRUE** value. By grouping these **TRUE** values in the largest possible rectangles, we can cancel out variables that appear in both true and complimented form.

A **"Don't Care"** state is marked with an X. These are used when a certain input combination does not impact the desired output. "Don't care" states, marked with an X are used when certain input combinations do not impact the output or are not possible in the design. These states can be treated as either 1 or 0, depending on whether they help in forming larger groups, further reducing the complexity of the Boolean expression.

Why are "Don't Care" states useful?

They provide flexibility in simplifying the logic. Since their output *doesn't matter*, they can be grouped with the 1's to create larger rectangles, thus reducing the number of terms in the final Boolean equation.

POSTNET (2 out of 5, 74210 weighting) and XS3:

Example 2 ***POSTNET and XS3 Encoding:***

- ***POSTNET****: Postal Numeric Encoding Technique, used by the U.S. Postal Service to encode ZIP codes into barcodes. It uses a 2-out-of-5 code where each digit is represented by five bars, and exactly two of the bars are tall. The weighting used to decode these bars is 74210, meaning the bars have respective weights of* $7, 4, 2, 1, 0$.
- ***XS3*** *(Excess-3): A binary-coded decimal (BCD) system where each digit is represented by its binary equivalent plus 3 (hence the term "excess-3"). For example, the decimal number 5 is represented in* $XS3$ *as* 0100_2 *(since 5 + 3 = 8 in binary).*

Number of Non-Valid Input Codes

To determine how many non-valid input codes are in your circuit, you need to consider the total number of possible input combinations based on the number of inputs. Let's assume there are n inputs. Non-valid input codes are those combinations that the circuit is not designed to handle or those that result in undefined behavior, typically represented as "don't care" states in the Karnaugh map (K-map).

Given that each bit can only be either 0 or 1, there are a total of 2^n possible input combinations. For instance, if there are 5 inputs, there would be $2^5 = 32$ possible combinations. If we have specific output codes for the decimal numbers 0 through 9, assuming these represent valid input combinations, we can calculate the number of non-valid input codes by subtracting the number of valid input codes from the total number of possible input combinations. Therefore:

$$\text{Number of Non-Valid Input Codes} = 2^n - 10$$

For $n = 5$, this would be:

$$\text{Number of Non-Valid Input Codes} = 32 - 10 = 22$$

POSTNET example

Example 3 ***POSTNET*** *for Groton, CT*

The ZIP code for Groton, CT is 06340-6049. This translates to:

Example 4 ***Conversion of Binary to Ternary and Hexadecimal***
Given the binary number 11010010_2*, we first convert it to decimal.*
Binary to Decimal

$$\begin{aligned} &1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= 128 + 64 + 0 + 16 + 0 + 0 + 2 + 0 \\ &= 210_{10} \end{aligned}$$

Decimal to Ternary

To convert decimal 210_{10} *to ternary, we repeatedly divide the number by 3 and record the remainders:*

Dividend	*Divisor*	*Quotient*	*Remainder*
210	*3*	*70*	*0*
70	*3*	*23*	*1*
23	*3*	*7*	*2*
7	*3*	*2*	*1*
2	*3*	*0*	*2*

Reading the remainders from bottom to top gives 21210_3*.*

Decimal to Hexadecimal

Similarly, to convert decimal 210_{10} *to hexadecimal, we divide by 16:*

Dividend	*Divisor*	*Quotient*	*Remainder*
210	*16*	*13*	*2*
13	*16*	*0*	*13*

The remainder 13 *corresponds to hexadecimal digit* D*, and* 2 *is itself. Therefore, the number* 210_{10} *in hexadecimal is* $D2_{16}$*.*

Summary:

- *The binary number* 11010010_2 *converts to* 210_{10} *in decimal.*
- *The decimal number* 210_{10} *converts to* 21210_3 *in ternary.*
- *The decimal number* 210_{10} *converts to* $D2_{16}$ *in hexadecimal.*

Example 5 *Conversion of the POSTNET (9) symbol to decimal without checksum*

Example 6 ***Simplifying a Logic Expression Using Boolean Logic Theorems***

Given Formula:

$$\overline{D}(C\overline{B} + \overline{A}) + \overline{\overline{D} + A}$$

$$\overline{D}(C\overline{B} + \overline{A}) + \overline{\overline{D}}\overline{A} \quad \textit{(Demorgan's Theorem)}$$

$$\overline{D}(C\overline{B} + \overline{A}) + D\overline{A} \quad \textit{(Involution)}$$

$$\overline{D}C\overline{B} + \overline{D}\overline{A} + D\overline{A} \quad \textit{(Distribution)}$$

$$\overline{D}C\overline{B} + \overline{A}(\overline{D} + D) \quad \textit{(Distributive Law)}$$

Final Answer:

$$\overline{D}C\overline{B} + \overline{A} \quad \textit{(Complement Law)}$$