Lab 02: Code Conversion

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

University of Connecticut
Coded in LaTeX

Theory

Weighting Determination of a Bit

The weight of a bit in a binary number is determined by its position in the sequence. Each column has twice the weight of the column to its right, which is why binary is considered a *base-2* system. This concept extends to other number systems as well.

Example 1 ***Number Systems and Base Conversions***
In base 10 (decimal):

$$9742_{10} = (9 \cdot 10^3) + (7 \cdot 10^2) + (4 \cdot 10^1) + (2 \cdot 10^0)$$

Converting between base 10 and base 2 (binary):

$$10110_2 = (1 \cdot 2^4) + (0 \cdot 2^3) + (1 \cdot 2^2) + (1 \cdot 2^1) + (0 \cdot 2^0) = 22_{10}$$

For base 3 (ternary), the possible digits are 0*,* 1*, and* 2*:*

$$2021_3 = (2 \cdot 3^3) + (0 \cdot 3^2) + (2 \cdot 3^1) + (1 \cdot 3^0) = 61_{10}$$

In general, for a numeral system of base N, a number $d_k d_{k-1} \ldots d_1 d_0$ represents the value:

$$(d_k \cdot N^k) + (d_{k-1} \cdot N^{k-1}) + \ldots + (d_1 \cdot N^1) + (d_0 \cdot N^0)$$

where d_i represents the digit at position i, and k is the position of the leftmost non-zero digit.

Radix and Radix Economy

Another name for the *bases* that were used in the previous example is Radix. The radix of a number system is what determines how many different values that can be used in that system.

Example 2 ***Common Radii***

- ***Binary*** *has a Radix of 2. Values can be* 0 *or* 1*.*
- ***Octal*** *has a Radix of 8. Values can be* $0, 1, 2, 3, 4, 5, 6,$ *or* 7*.*
- ***Hexadecimal*** *has a Radix of 16. Values can be* $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E,$ *or* F*.*

Radix Economy in Number Systems

The concept of *Radix Economy* provides a metric for evaluating the efficiency of number systems in representing information. It is defined by the formula:

$$E = r \cdot \log_r N$$

where E is the economy, r is the radix (base) of the number system, and N is the number of unique values to be represented.

This metric balances two competing factors:

- The number of unique digits required (complexity)
- The number of digit positions needed to represent a given range of values (length)

A lower value of E indicates a more economical system, offering a better trade-off between these factors.

Implications and Applications

Systems with good radix economy can offer several advantages:

- Enhanced human readability
- Reduced system complexity
- More compact and efficient computational implementations
- Potential for reduced physical space requirements in hardware
- Improved circuit performance and reduced power consumption
- Potential for lower overall project costs in hardware design

Optimal Radix vs. Practical Considerations

Mathematically, the optimal radix for maximal economy is $e \approx 2.718$ which is derived by using the second derivative test to determine the maximum value. However, practical considerations often outweigh pure mathematical optimality

Binary (*radix-2*) systems are prevalent in computing despite not having the best radix economy. The simplicity of binary representation, with only two states (0 and 1), greatly simplifies computer architecture and logic design. This simplicity often outweighs the theoretical advantages of more economical bases in real-world applications.

The choice of number system thus involves a balance between theoretical efficiency and practical implementation constraints.

Deliverables

After calculating the minterms for our Z_0, Z_1, Z_2, Z_3 outputs, that is, the equations such that our corresponding output bit is TRUE (1), we get some fairly large equations that were then able to get reduced by boolean logic algebra. Unfortunately, because the lab limited us to only two-input OR, AND, and one-input NOT gates, the simplicity of the circuit was not able to get greatly reduced.

Example 3 *Outputs Z_3, Z_2, Z_1, Z_0:*

- $Z_0 = \overline{D}\overline{C}\overline{B}A + \overline{D}C\overline{B}\overline{A} + \overline{D}CBA + D\overline{C}B\overline{A}$
  $Z_0 = \overline{D}\overline{C}BA + CBA + CBA) + D\overline{C}B\overline{A}$
  $Z_0 = \overline{D}(\overline{C}\overline{B}A + C(\overline{B}\overline{A} \ + BA)) + D\overline{C}B\overline{A}$
  The Z_0 bit could be simplified by grouping the $\overline{D}$ signal to the outside of common multiples and some mild reduction utilizing distributivity and identity boolean algebra rules.

- $Z_1 = \overline{D}\overline{C}B\overline{A} + \overline{D}C\overline{B}A + D\overline{C}\overline{B}\overline{A}$
  $Z_1 = \overline{D}(\overline{C}B\overline{A} + C\overline{B}A) + D\overline{C}\overline{B}\overline{A}$
  The Z_1 bit could also be simplified by grouping the $\overline{D}$ signal to the outside of common multiples.

- $Z_2 = 0$ *(Always FALSE)*

- $Z_3 = 0$ *(Always FALSE)*

Discussion

The practical portion of the lab was more involved with Logicworks as we learned about simplifying Boolean expressions. The inclusion of the Hex Keyboard into the schematic allowed for a nicer way of interacting with the circuits and careful wire routing throughout the workspace was crucial for success in completing this. Ultimately, the simplifications couldn't be applied as much as I initially thought that would be possible when it came to designing the circuit, but nevertheless each network of signals was easily troubleshot and ran through until the target output was reached.

Practically, this was quite engaging as the network of wires needed to be very closely monitored to ensure that a certain system wouldn't change after modification. After some troubleshooting and careful analysis of wire and inverter placement, the circuit was able to run, and the desired truth table was output successfully.

To conclude and summarize, the Lab reinforced concepts learned in class through pen and paper arithmetic and familiarity with the Digital Design Software tools in Logicworks.

Practice Questions

Radix Conversion

Given the decimal number 92_{10} we can change between number systems utilizing the algorithm discussed previously.

Example 4 ***Binary:***

$$92_{10} = (1 \cdot 2^6) + (0 \cdot 2^5) + (1 \cdot 2^4) + (1 \cdot 2^3) + (1 \cdot 2^2) + (0 \cdot 2^1) + (0 \cdot 2^0)$$

$$92_{10} = 1011100_2$$

Example 5 ***Octal:***

$$92_{10} = (1 \cdot 8^2) + (3 \cdot 8^1) + (4 \cdot 8^0)$$

$$92_{10} = 134_8$$

Example 6 ***Hexadecimal:***

$$92_{10} = (5 \cdot 16^1) + (12 \cdot 16^0) \quad \textit{(where } C = 12\textit{)}$$

$$92_{10} = 5C_{16}$$

2's Compliment and Decimal Conversion

Given the signed binary number 11001101_2 swapping between 2's compliment and decimal is extremely easily. Begin by looking at the Most-Significant-Bit (MSB) for the first information about what decimal number this signed binary number contains. An MSB of 1 indicates a negative number, an MSB of 0 represents a positive number.

The two's complement representation facilitates simpler arithmetic operations and algorithms in digital systems. However, this comes at the cost of a reduced range for the maximum positive number that can be represented, compared to unsigned integers of the same bit length.

Example 7 ***Size limits***
Unsigned 8-bit integer:

- *Range:* $0 \rightarrow 2^8 - 1$ *(0 to 255)*

Signed 8-bit integer (two's complement):

- *Positive range (MSB = 0):* $0 \rightarrow 2^7 - 1$ *(0 to 127)*
- *Negative range (MSB = 1):* $-2^7 \rightarrow -1$ *(-128 to -1)*

Note: *The signed range includes one more negative number (-128) than positive number (127) due to the nature of two's complement representation.*

Example 8 ***Two's Complement Conversion:***
Flip the bits and add 1.

$$\begin{array}{rl} 11001101_2 & \textit{(original number)} \\ 00110010_2 & \textit{(flipped bits)} \\ +00000001_2 & \textit{(add 1)} \\ \hline 00110011_2 & \textit{(two's complement result)} \end{array}$$

The two's complement of 11001101_2 *is:*

$$00110011_2$$

Example 9 *Two's Complement to Decimal:*
To convert 11001101_2 *(signed) to decimal, we take the two's complement result and compute its negative value.*

The two's complement result is 00110011_2*, which is positive. Converting this binary number to decimal:*

$$00110011_2 = (0 \cdot 2^7) + (0 \cdot 2^6) + (1 \cdot 2^5) + (1 \cdot 2^4) + (0 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0)$$

$$= (1 \cdot 32) + (1 \cdot 16) + (1 \cdot 2) + (1 \cdot 1) = 32 + 16 + 2 + 1 = 51$$

Since 11001101_2 *is a negative signed number, the decimal value of the original binary number is:*

$$-51_{10}$$

Boolean Logic to Gates

To finish things off, the extraction of 2 input gates from a simple boolean expression is done in the following example accompanied by the corresponding truth table.

In order to correctly populate the table, each expression was treated seperately and ORed together to form the final output column of F.

Example 10 $\overline{A} + C\overline{D} + BA$ *is expressed in logic gates in the following figure*

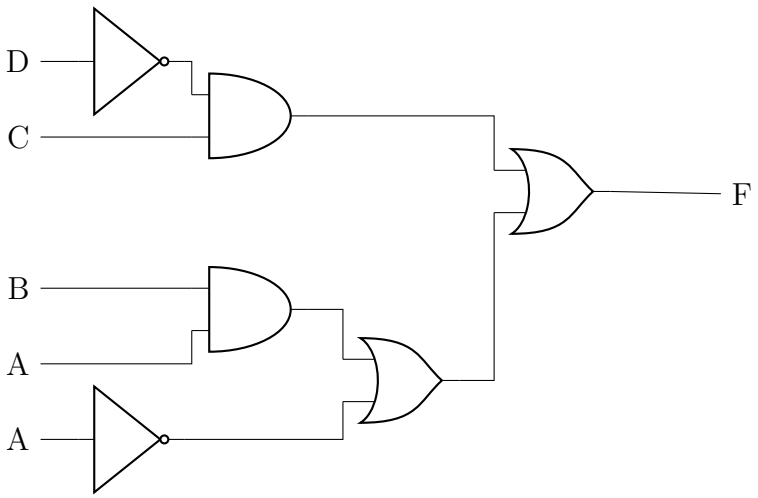

Figure 1: Logic gate diagram for $\overline{A} + C\overline{D} + BA$

D	C	B	A	$\overline{A}$	$C\overline{D}$	BA	F
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	1
0	0	1	1	0	0	1	1
0	1	0	0	1	1	0	1
0	1	0	1	0	1	0	1
0	1	1	0	1	1	0	1
0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	0	0	1	1
1	1	0	0	1	0	0	1
1	1	0	1	0	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	0	0	1	1