5.4 (Carry-look-ahead generators)

Using the libraries given by the tables noted below, redesign the CLA generator that was shown in Figure 5.2(c):

(b) Table 3.14 and 3.15,

For each of the designs, compare the delays in the generation of the following carries: $c_4$, $c_8$, $c_{12}$ and $c_{16}$.

(b) The CLA generator supplied in Figure 5.2(c) fits the requirements for this problem. The delay through each step of the 16-bit addition would be $8.8 \, \text{ns}$. For each step of the addition the carry delays would be

\[
\begin{align*}
    c_4 &= 8.8 \, \text{ns} \\
    c_4 &= 17.6 \, \text{ns} \\
    c_4 &= 26.4 \, \text{ns} \\
    c_4 &= 35.2 \, \text{ns}
\end{align*}
\]

(c) The CLA generator can be redesigned by replacing groups of gates with logic supplied in Table 3.16.

The delay through each step of the 16-bit addition would be $6.4 \, \text{ns}$. For each step of the addition the carry delays would be

\[
\begin{align*}
    c_4 &= 6.4 \, \text{ns} \\
    c_4 &= 12.8 \, \text{ns} \\
    c_4 &= 19.2 \, \text{ns} \\
    c_4 &= 25.6 \, \text{ns}
\end{align*}
\]
Design a 64-bit CLA adder, using:

(a) one level of CLA,

Compare the delays of these adders.

(a) To design a 64-bit adder with a single level of CLA, we will need 16 4-bit CLA adders. If we let the Figure 5.2(b) be represented by

![Diagram of a 4-bit adder](image)

Then the single level CLA adder can be implemented by chaining these components as shown below:

![Diagram of a 64-bit adder](image)
Design a logic unit that will perform the following combinations of operations:

(b) XOR and XNOR,
5.10 (Decoders)
Design a 4-to-16 decoder, using:
(a) 1-to-2 decoders,
5.11 (Encoders)
Design a (b) 4-to-2, unary to binary code encoder, which is a true complement of a binary to unary code decoder.

(b) 4-to-2 Encoder
Design the serial and parallel versions of a comparator that can compare the following types of number representation:
(a) sign-magnitude,

(a) Let the two numbers to be compared be $M_1$ and $M_2$. The outputs are: $M_1 > M_2$, $M_1 = M_2$ or $M_1 < M_2$. The algorithm to perform sign-magnitude comparison could be defined as:

First, compare $MSB(M_1)$ and $MSB(M_2)$.
If $(MSB(M_1) > MSB(M_2))$ then
  
  output $(M_1 < M_2)$

else
  If $(MSB(M_1) < MSB(M_2))$ then
    output $(M_1 > M_2)$

else
  If $(MSB(M_1) = 0)$ then
    compare the rest of the bits of both inputs
    using either a serial or a parallel comparator
    Output the results of this comparison
  else
    compare the rest of the bits of both inputs
    using either a serial or a parallel comparator
    If the output is $(M_1 = M_2)$ then
      output $(M_1 = M_2)$
    If the output is $(M_1 > M_2)$ then
      output $(M_1 > M_2)$
    If the output is $(M_1 < M_2)$ then
      output $(M_1 < M_2)$
end
5.14 (Comparators)
Design the comparators, that would evaluate the following single relations:

(b) $X = Y$,

Assuming that $X = a_1a_0$ and $Y = b_1b_0$, the following implementations can be expanded or combined to compare values with a greater number of digits.