Development of a fast 4 bit ALU slice:

The ALU will be implemented in two stages of PLA. The first stage will calculate the carry generate (g#) and propagate (p#) and exclusive-or (x#) for each bit as well as a carry generate (G) and propagate (P) for the overall stage. In addition, it will perform the complement of the B operand as required for subtraction (by adding extra lines to each output equation). To simplify this first stage, a pair of new control signals, ADD and SUB, will be generated by the control logic. When ADD is false, the P and G signals will not be generated. When subtracting, both ADD and SUB will be true.

The second stage will accept the signals from the first stage and perform the addition or logical operation. These units can be joined to form wider ALUs with the addition of a high level carry look-ahead unit that will input the overall Ps and Gs from each 4 bit first stage slice and provide carry inputs to each second stage slice as well as the overall carry output.

Let the sets of four operand bits be labeled A3..A0 and B3..B0 and the four output bits be labeled s0..s3 and the carry in be c. The overall propagate signal shall be P and the overall generate signal shall be G.

The notion of propagate at the individual bit level can be expressed by the OR of the two operand bits and the generate by the AND, i.e.:

\[ p2 = A2 + B2, \quad g2 = A2 \cdot B2. \]
Note that:
\[ P^\# = A^# \cdot B^# \]

The combined Propagate can be derived from:
\[
P = p_3 \cdot p_2 \cdot p_1 \cdot p_0
\]
\[
= (a_0 + b_0) \cdot (a_1 + b_1) \cdot (a_2 + b_2) \cdot (a_3 + b_3)
\]
\[
= a_0 \cdot a_1 \cdot a_2 \cdot a_3 + a_0 \cdot a_1 \cdot a_2 \cdot b_3 + a_0 \cdot a_1 \cdot b_2 \cdot a_3 + a_0 \cdot a_1 \cdot b_2 \cdot b_3 + a_0 \cdot b_1 \cdot a_2 \cdot a_3 + a_0 \cdot b_1 \cdot a_2 \cdot b_3 + a_0 \cdot b_1 \cdot b_2 \cdot a_3 + a_0 \cdot b_1 \cdot b_2 \cdot b_3 + b_0 \cdot a_1 \cdot a_2 \cdot a_3 + b_0 \cdot a_1 \cdot a_2 \cdot b_3 + b_0 \cdot a_1 \cdot b_2 \cdot a_3 + b_0 \cdot a_1 \cdot b_2 \cdot b_3 + b_0 \cdot b_1 \cdot a_2 \cdot a_3 + b_0 \cdot b_1 \cdot a_2 \cdot b_3 + b_0 \cdot b_1 \cdot b_2 \cdot a_3 + b_0 \cdot b_1 \cdot b_2 \cdot b_3.
\]

This results in a 31 line equation by the time we account for the inversion of B for subtraction. But if we instead calculate \( \overline{P} \):
\[
\overline{P} = p_3 \cdot p_2 \cdot p_1 \cdot p_0
\]
\[
= \overline{a_3} \cdot b_3 + \overline{a_2} \cdot b_2 + \overline{a_1} \cdot b_1 + \overline{a_0} \cdot b_0
\]

and then use the active low output on the PLA to invert it we can greatly simplify the equations.

The overall Generate is derived from:
\[
G = g_3 + (p_3 \cdot g_2) + (p_3 \cdot p_2 \cdot g_1) + (p_3 \cdot p_2 \cdot p_1 \cdot g_0).
\]

Unfortunately, because the B bits are not already complemented for the first stage, the equations must be almost doubled. Considerable simplification would be made possible by two more levels of logic, one to implement the B complement and another to implement the individual p terms before the P and G terms, at the cost of slowing the execution cycle.

The second stage in the ALU 4 bit slice is similar to the PLA used for the single bit slice, with the exception that the individual carry terms (except c0) are not available, so the following equations must be used in place of the respective carry terms:

\[
c_1 = g_0 + (p_0 \cdot c_0)
\]
\[
c_2 = g_1 + (p_1 \cdot g_0) + (p_1 \cdot p_0 \cdot c_0)
\]
\[
c_3 = g_2 + (p_2 \cdot g_1) + (p_2 \cdot p_1 \cdot g_0) + (p_2 \cdot p_1 \cdot p_0 \cdot c_0)
\]

These equations will add many lines to the PLA, however, it will be offset by not having to use separate terms for subtractions and by not actually having to generate the carry outputs.

New terms \( \overline{p_2 p_1 p_0} \), \( \overline{p_2 p_1 g_0} \), \( \overline{p_2 g_1} \), \( \overline{p_1 g_0} \) and \( \overline{p_1 p_0} \) have been added to the first level PLA to aid in the calculations of \( \overline{c^#} \). For example:
\[
\overline{c_3} = (g_2 \cdot \overline{g_1 p_2} \cdot \overline{p_2 p_1 g_0} \cdot \overline{p_2 p_1 p_0}) + (g_2 \cdot \overline{g_1 p_2} \cdot \overline{p_2 p_1 g_0} \cdot \overline{c_0})
\]

Now, logic functions AND, OR, XOR and NOR must now be derived from g, p, x and \( \overline{p} \) respectively.