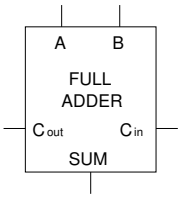
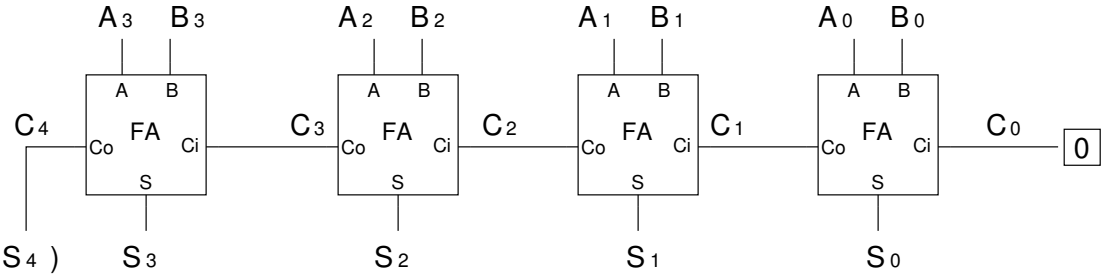
3.1 Binary Addition

Recall the specification of one-bit addition, in the truth-table below. The box to the right represents an implementation of this function. It has three inputs, A, B and C_{in}; and two outputs, S and C_{out}.

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Connecting four FAs in series, results in a 4-bit *ripple-carry adder*.



The equations below define the i^{th} instances of outputs S and C.

$$\begin{aligned} S_i &= \mathit{parity}(A,\ B,\ C_i) = A \oplus B \oplus C_i \\ C_{i+1} &= \mathit{majority}(A_i,\ B_i,\ C_{in}) = A_i\, B_i + A_i\, C_i + B_i\, Ci \end{aligned}$$

Expanding these equations for output C_4, the most-significant bit of the sum, results in an expression in terms of the inputs.

$$\begin{aligned} C4 &= A_3\, B_3 + A_3\, C_3 + B_3\, C_3 \\ &= A_3\, B_3 + (A_3 + B_3)\, C_3 \\ &= A_3\, B_3 + (A_3 + B_3)\, [A_2, B_2 + (A_2 + B_2)\, C_2] \\ &= A_3\, B_3 + (A_3 + B_3)\, [A_2, B_2 + (A_2 + B_2)\, [A_1\, B_1 + (A_1 + B_1)\, C_1]] \\ &= A_3\, B_3 + (A_3 + B_3)\, [A_2, B_2 + (A_2 + B_2)\, [A_1\, B_1 + (A_1 + B_1)\, [A_0\, B_0 + (A_0 + B_0)\, C_0]]] \end{aligned}$$

For an n-bit adder, the depth of the term C_n is proportional[1] to n. In other words, the time needed for ripple-carry addition grows linearly with n.

[1]This proportionality is even true for an SOP form because *and/or* devices are limited to around 5 inputs. Hence a "wide" *and* gate must be implemented by a tree of 5-input *and*s.

3.1.1 Speeding Up Addition

Carry propagation can be reduced to $\mathbf{O}(\log n)$, by devising a tree to perform *look-ahead*. The key insight is to think of one-bit addition differently. Instead of carry-in/carry-out logic, look at a one-bit adder in terms of whether it generates or propagates a carry bit.

A	B	G	P
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\begin{aligned} G_i &= A_i \cdot B_i \\ P_i &= A_i \oplus B_i \end{aligned}$$

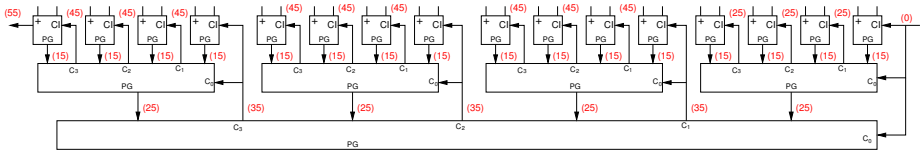
We can still recover *sum* and *carry-out* as

$$\begin{aligned} S_i &= C_i \oplus G_i \\ C_{i+1} &= G_i + P_i \cdot C_i \end{aligned}$$

In a 4-bit adder this design is not an improvement over the original *full adder*. The advantage becomes evident when we develop *propagate* and *generate* values for the *group*.

$$\begin{aligned} P^* &= P_3 \cdot P_2 \cdot P_1 \cdot P_0 \\ G^* &= G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1 \\ C^* &= G^* + P^* \cdot C_0 \end{aligned}$$

This generate/propagate construction is also valid for a group of four groups, a group of four groups of four groups, and so on, hierarchically. The result is a carry/propagate tree needing only C_0 to compute the carries.



Each level of the tree adds the same delay to computing the four carry values for its group. The diagram above shows representative delay times (in nonoseconds, based on the MSI ICs, the 74LS181, 4-bit arithmetic-logic device and the 74LS182 look-ahead carry generator). A 64-bit addition (in the worst case) has a 55 ns delay to generate the C_{64}. Since each 74LS181 takes about 10 ns to develop it's value for C_4, the ripple-carry form of this 64-bit adder would impose a delay of around 80 ns in the worst case. (According to the 74LS181 data sheet, the average time is 68 ns.)

3.1.2 Ripple Adder Logic

This section shows the intermediate results for an ad hoc expansion of C_4, followed by a normal SOP and 2-level minimization.

Expansion of Term C_4

```
> (define c4
 (let* ((c1 '((a0 & b0) + ((a0 + b0) & c0)))
        (c2 '((a1 & b1) + ((a1 + b1) & ,c1)))
        (c3 '((a2 & b2) + ((a2 + b2) & ,c2)))
        (c4 '((a3 & b3) + ((a3 + b3) & ,c3))))
       c4))
> c4
((a3 & b3)
  +
  ((a3 + b3)
    &
    ((a2 & b2)
      +
      ((a2 + b2)
        &
        ((a1 & b1)
          +
          ((a1 + b1) & ((a0 & b0) + ((a0 + b0) & c0))))))))
```

Normal Form

Figure 3.1 shows the normal-SOP expansion of C_4. The normal SOP has 235 clauses.

SOP Minimization

A minimal 2-level term, shown below, contains 31 clauses.

```
> (qm-proc (dnf-proc c4))
((a3 a2 a1 a0 b0) (a3 a2 a1 a0 c0) (a3 a2 b1 a0 b0) (a3 a2 b1 a0 c0) (a3 b2 a1 a0 b0)
 (a3 b2 a1 a0 c0) (a3 b2 b1 a0 b0) (a3 b2 b1 a0 c0) (b3 a2 a1 a0 b0) (b3 a2 a1 a0 c0)
 (b3 a2 b1 a0 b0) (b3 a2 b1 a0 c0) (b3 b2 a1 a0 b0) (b3 b2 a1 a0 c0) (b3 b2 b1 a0 b0)
 (b3 b2 b1 a0 c0) (a3 a2 a1 b0 c0) (a3 a2 b1 b0 c0) (a3 b2 a1 b0 c0) (a3 b2 b1 b0 c0)
 (b3 a2 a1 b0 c0) (b3 a2 b1 b0 c0) (b3 b2 a1 b0 c0) (b3 b2 b1 b0 c0) (a3 a2 a1 b1)
 (a3 b2 a1 b1) (b3 a2 a1 b1) (b3 b2 a1 b1) (a3 a2 b2) (b3 a2 b2) (a3 b3))
```

Multilevel Minimization

The steps taken below were done manually, so there is no claim of minimality (or correctness). The goal is to reduce to a reasonably small, multi-level boolean


```
> (dnf~proc c4)
((~a3 b3 ~a2 b2 ~a1 b1 ~a0 b0 c0)
 (~a3 b3 ~a2 b2 ~a1 b1 a0 ~b0 c0)
 (~a3 b3 ~a2 b2 ~a1 b1 a0 b0 ~c0)
 (~a3 b3 ~a2 b2 ~a1 b1 a0 b0 c0)
 (~a3 b3 ~a2 b2 a1 ~b1 ~a0 b0 c0)
 (~a3 b3 ~a2 b2 a1 ~b1 a0 ~b0 c0)
 (~a3 b3 ~a2 b2 a1 ~b1 a0 b0 ~c0)
 (~a3 b3 ~a2 b2 a1 ~b1 a0 b0 c0)
 (~a3 b3 ~a2 b2 a1 b1 ~a0 ~b0 ~c0)
 (~a3 b3 ~a2 b2 a1 b1 ~a0 ~b0 c0)
 (~a3 b3 ~a2 b2 a1 b1 ~a0 b0 ~c0)
 (~a3 b3 ~a2 b2 a1 b1 ~a0 b0 c0)
 (~a3 b3 ~a2 b2 a1 b1 a0 ~b0 ~c0)
 (~a3 b3 ~a2 b2 a1 b1 a0 ~b0 c0)
 (~a3 b3 ~a2 b2 a1 b1 a0 b0 ~c0)
 (~a3 b3 ~a2 b2 a1 b1 a0 b0 c0)
 (~a3 b3 a2 ~b2 ~a1 b1 ~a0 b0 c0)
 (~a3 b3 a2 ~b2 ~a1 b1 a0 ~b0 c0)
 (~a3 b3 a2 ~b2 ~a1 b1 a0 b0 ~c0)
 (~a3 b3 a2 ~b2 ~a1 b1 a0 b0 c0)
 (~a3 b3 a2 ~b2 a1 ~b1 ~a0 b0 c0)
 (~a3 b3 a2 ~b2 a1 ~b1 a0 ~b0 c0)
 (~a3 b3 a2 ~b2 a1 ~b1 a0 b0 ~c0)
 (~a3 b3 a2 ~b2 a1 ~b1 a0 b0 c0)
 (~a3 b3 a2 ~b2 a1 b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 ~a0 ~b0 c0)
 (~a3 b3 a2 ~b2 a1 b1 ~a0 b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 ~b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 ~b0 c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 b0 c0)
 (~a3 b3 a2 b2 a1 ~b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 b2 a1 ~b1 ~a0 ~b0 c0)
 (~a3 b3 a2 b2 a1 ~b1 ~a0 b0 ~c0)
 (~a3 b3 a2 b2 a1 ~b1 ~a0 b0 c0)
 (~a3 b3 a2 b2 a1 ~b1 a0 ~b0 ~c0)
 (~a3 b3 a2 b2 a1 ~b1 a0 ~b0 c0)
 (~a3 b3 a2 b2 a1 ~b1 a0 b0 ~c0)
 (~a3 b3 a2 b2 a1 ~b1 a0 b0 c0)
 (~a3 b3 a2 b2 a1 b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 b2 a1 b1 ~a0 ~b0 c0)
 (~a3 b3 a2 b2 a1 b1 ~a0 b0 ~c0)
 (~a3 b3 a2 b2 a1 b1 ~a0 b0 c0)
 (~a3 b3 a2 b2 a1 b1 a0 ~b0 ~c0)
 (~a3 b3 a2 b2 a1 b1 a0 ~b0 c0)
 (~a3 b3 a2 b2 a1 b1 a0 b0 ~c0)
 (~a3 b3 a2 b2 a1 b1 a0 b0 c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 ~a0 b0 c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 ~a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 ~b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 ~b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 b0 c0)
 (~a3 b3 a2 ~b2 a1 ~b1 ~a0 ~b0 c0)
 (~a3 b3 a2 ~b2 a1 ~b1 a0 ~b0 ~c0)
 (~a3 b3 a2 ~b2 a1 ~b1 a0 ~b0 c0)
 (~a3 b3 a2 ~b2 a1 ~b1 a0 b0 ~c0)
 (~a3 b3 a2 ~b2 a1 ~b1 a0 b0 c0)
 (~a3 b3 a2 ~b2 a1 b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 ~a0 ~b0 c0)
 (~a3 b3 a2 ~b2 a1 b1 ~a0 b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 ~a0 b0 c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 ~b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 ~b0 c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 b0 ~c0)
 (~a3 b3 a2 ~b2 a1 b1 a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 ~a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 ~b1 a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 ~a0 b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 ~b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 ~b0 c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 b0 ~c0)
 (~a3 b3 a2 b2 ~a1 b1 a0 b0 c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 ~a0 b0 c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 ~a1 b1 a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 ~a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 ~b1 a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 ~b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 ~a0 b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 ~b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 ~b0 c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 b0 ~c0)
 (a3 ~b3 ~a2 b2 a1 b1 a0 b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 ~a0 ~b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 ~a0 ~b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 ~a0 b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 ~a0 b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 a0 ~b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 a0 ~b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 a0 b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 ~b1 a0 b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 ~a0 ~b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 ~a0 ~b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 ~a0 b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 ~a0 b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 a0 ~b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 a0 ~b0 c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 a0 b0 ~c0)
 (a3 b3 ~a2 ~b2 ~a1 b1 a0 b0 c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 ~a0 ~b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 ~a0 ~b0 c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 ~a0 b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 ~a0 b0 c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 a0 ~b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 a0 ~b0 c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 a0 b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 ~b1 a0 b0 c0)
 (a3 b3 ~a2 b2 ~a1 b1 ~a0 ~b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 b1 ~a0 ~b0 c0)
 (a3 b3 ~a2 b2 ~a1 b1 ~a0 b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 b1 ~a0 b0 c0)
 (a3 b3 ~a2 b2 ~a1 b1 a0 ~b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 b1 a0 ~b0 c0)
 (a3 b3 ~a2 b2 ~a1 b1 a0 b0 ~c0)
 (a3 b3 ~a2 b2 ~a1 b1 a0 b0 c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 ~a0 ~b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 ~a0 ~b0 c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 ~a0 b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 ~a0 b0 c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 a0 ~b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 a0 ~b0 c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 a0 b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 ~b1 a0 b0 c0)
 (a3 b3 a2 ~b2 ~a1 b1 ~a0 ~b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 b1 ~a0 ~b0 c0)
 (a3 b3 a2 ~b2 ~a1 b1 ~a0 b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 b1 ~a0 b0 c0)
 (a3 b3 a2 ~b2 ~a1 b1 a0 ~b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 b1 a0 ~b0 c0)
 (a3 b3 a2 ~b2 ~a1 b1 a0 b0 ~c0)
 (a3 b3 a2 ~b2 ~a1 b1 a0 b0 c0)
 (a3 b3 a2 b2 ~a1 ~b1 ~a0 ~b0 ~c0)
 (a3 b3 a2 b2 ~a1 ~b1 ~a0 ~b0 c0)
 (a3 b3 a2 b2 ~a1 ~b1 ~a0 b0 ~c0)
 (a3 b3 a2 b2 ~a1 ~b1 ~a0 b0 c0)
 (a3 b3 a2 b2 ~a1 ~b1 a0 ~b0 ~c0)
 (a3 b3 a2 b2 ~a1 ~b1 a0 ~b0 c0)
 (a3 b3 a2 b2 ~a1 ~b1 a0 b0 ~c0)
 (a3 b3 a2 b2 ~a1 ~b1 a0 b0 c0)
 (a3 b3 a2 b2 ~a1 b1 ~a0 ~b0 ~c0)
 (a3 b3 a2 b2 ~a1 b1 ~a0 ~b0 c0)
 (a3 b3 a2 b2 ~a1 b1 ~a0 b0 ~c0)
 (a3 b3 a2 b2 ~a1 b1 ~a0 b0 c0)
 (a3 b3 a2 b2 ~a1 b1 a0 ~b0 ~c0)
 (a3 b3 a2 b2 ~a1 b1 a0 ~b0 c0)
 (a3 b3 a2 b2 ~a1 b1 a0 b0 ~c0)
 (a3 b3 a2 b2 ~a1 b1 a0 b0 c0))
```

Figure 3.1: Normal SOP form for C_4

system. Starting from the minimized SOP term for C_4,

(a) Identify common subterms

```
x = a1 a0 c0
y = b1 a0 c0
z = a1 b0 c0
u = a1 a0 b0
v = b1 a0 b0
w = a1 b0 c0
s = b1 b0 c0
t = a1 b1

((a3 a2 u)+(a3 b2 u)+(b3 a2 u)+(b3 b2 u)
+(a3 a2 x)+(a3 b2 x)+(b3 a2 x)+(b3 b2 x)
+(a3 a2 v)+(a3 b2 v)+(b3 a2 v)+(b3 b2 v)
+(a3 a2 y)+(a3 b2 y)+(b3 a2 y)+(b3 b2 y)
+(a3 a2 w)+(a3 b2 w)+(b3 a2 w)+(b3 b2 w)
+(a3 a2 s)+(a3 b2 s)+(b3 a2 s)+(b3 b2 s)
+(a3 a2 t)+(a3 b2 t)+(b3 a2 t)+(b3 b2 t)
+(a3 a2 b2)
+(b3 a2 b2)
+(a3 b3))
```

(b) Distribute `u`, ..., `t`.

```
((((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2)) u)
+(((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2)) x)
+(((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2)) v)
+(((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2)) y)
+(((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2)) w)
+(((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2)) s)
+(((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2)) t)
+(a3 a2 b2)
+(b3 a2 b2)
+(a3 b3))
```

(c) Identify common subterms

```
p = ((a3 a2)+(a3 b2)+(b3 a2)+(b3 b2))

((p u) + (p x) + (p v) + (p y) + (p w) + (p s) + (p t)
 +(a3 a2 b2) +(b3 a2 b2) +(a3 b3))
```

(d) Distribute

```
(((u + x + v + y + w + s + t) p)
+(a3 a2 b2)
+(b3 a2 b2)
+(a3 b3))
```

(e) Close, but too many operands in `(u + x + v + y + w + s + t)`. It would need to be implemented as something like `((u + x + v + y) + (w + s + t))`,

adding a gate-delay. Perhaps we should try again ...

3.1.3 Verification of Propagate-Generate Logic

Fig. **??** shows an input file for the *Boole* verification tool. Figure **??** translates of the ripple-carry (RC) and generate-propagate (GP) adder specifications to the input language of a boolean verification tool called *Boole*. The first 9 lines declare variables and their ordering. The last 4 lines are commands:

```
sop GPS4 = RCS4;    compare the S4 bits
sop GPC5 = RCC5;    compare the C5 bits
satisfy G0 = P0;    generate conditions, if any, where G0 = P0
profile GPS4;       show the size of the BDD for GP bit S4.
```

When *Boole* is invoked with this input file the output is:

```
sop GPS4 = RCS4;   ⇒   1
sop GPC5 = RCC5;   ⇒   1
satisfy G0 = P0;   ⇒   !A0 & !B0
profile GPS4;      ⇒    A4:   1 #
                        A3:   1 #
                        A2:   2 ##
                        A1:   4 ####
                        A0:   8 ########
                        B4:  16 ################
                        B3:  16 ################
                        B2:   8 ########
                        B1:   4 ####
                        B0:   2 ##
                        C0:   1 #
                        leaf: 1 #
                        Total: 64
```

The S_4 and C_5 terms are equivalent; $G_0 = P_0$ if $A_0 = B_0 = 0$ (and possibly other conditions); The BDD representation for GP bit S_4 contains 64 nodes under the variable ordering $\langle A_4, A_3, A_2, A_1, A_0, B_4, B_3, B_2, B_1, B_0, C_0 \rangle$.

A different variable ordering can dramatically affect the BDD size. Under ordering $\langle A_0, B_0, A_1, B_1, A_2, B_2, A_3, B_3, A_4, B_4, C_0 \langle$, *Boole* outputs

```
sop GPS4 = RCS4;   ⇒  1
sop GPC5 = RCC5;   ⇒  1
satisfy G0 = P0;   ⇒  !A0 & !B0
profile GPS4;      ⇒   A0:   1 #
                       B0:   2 ##
                       A1:   3 ###
                       B1:   3 ###
                       A2:   3 ###
                       B2:   3 ###
                       A3:   3 ###
                       B3:   3 ###
                       A4:   2 ##
                       B4:   2 ##
                       C0:   1 #
                       leaf: 1 #
                       Total: 27
```



```
vars A4 A3 A2 A1 A0;
vars B4 B3 B2 B1 B0;
vars C0;
vars RCS0 RCS1 RCS2 RCS3 RCS4;
vars RCC1 RCC2 RCC3 RCC4 RCC5;
vars P0 P1 P2 P3 P4;
vars G0 G1 G2 G3 G4;
vars GPS0 GPS1 GPS2 GPS3 GPS4;
vars GPC1 GPC2 GPC3 GPC4 GPC5;

RCS0 := A0 ^ B0 ^ C0;
RCC1 := (A0 * B0) + (C0 * (A0 + B0));

RCS1 := A1 ^ B1 ^ RCC1;
RCC2 := (A1 * B1) + (RCC1 * (A1 + B1));

RCS2 := A2 ^ B2 ^ RCC2;
RCC3 := (A2 * B2) + (RCC2 * (A2 + B2));

RCS3 := A3 ^ B3 ^ RCC3;
RCC4 := (A3 * B3) + (RCC3 * (A3 + B3));

RCS4 := A4 ^ B4 ^ RCC4;
RCC5 := (A4 * B4) + (RCC4 * (A4 + B4));

G0 := A0 * B0;
P0 := A0 ^ B0;
GPS0 := P0 ^ C0;
GPC1 := G0 + (P0 * C0);

G1 := A1 * B1;
P1 := A1 ^ B1;
GPS1 := P1 ^ GPC1;
GPC2 := G1 + (P1 * GPC1);

G2 := A2 * B2;
P2 := A2 ^ B2;
GPS2 := P2 ^ GPC2;
GPC3 := G2 + (P2 * GPC2);

G3 := A3 * B3;
P3 := A3 ^ B3;
GPS3 := P3 ^ GPC3;
GPC4 := G3 + (P3 * GPC3);

G4 := A4 * B4;
P4 := A4 ^ B4;
GPS4 := P4 ^ GPC4;
GPC5 := G4 + (P4 * GPC4);

sop GPS4 = RCS4;
sop GPC5 = RCC5;
satisfy G0 = P0;
profile GPS4;
```

Figure 3.2: Verification of propagate-generate addition with respect to ripple-carry addition. Bits S_4 and C_5 are compared.

