CMPT 250 : Week 4 (Sept 26 to Oct 1)

1. DESIGN OF ARITHMETIC HARDWARE

2. ALU DESIGN

The adder/subtractor module described in CMPT 150 (Malvino, page 85) is a simple example of an arithmetic/logic unit or *ALU*. The design of such modules follows standard combinational logic principles described previously. Some "tricks" that tend to simplify the implementation are described here.

To begin, an ALU is formally specified by a black-box and a function select table. The black-box identifies two data input buses, a data output bus, and a function select bus. It may also include a carry-in and carry-out or propagate and generate outputs to permit chaining.

To illustrate the design procedure, consider an ALU that is to provide the following functions:

```
f2 f1 f0| FUNCTION
--------|--------------------
0  0  0 | out <- 0
0  0  1 | out <- A and B
0  1  0 | out <- B'
0  1  1 | out <- 0' = (11...1)
1  0  0 | out <- A plus B
1  0  1 | out <- A minus B
1  1  0 | out <- A plus 1
1  1  1 | out <- A minus 1
```

To begin, try to express each function the "same way" since this will reduce the number of distinct subcircuits that will be required. In this example it is possible to define every "artihmetic" function using the full adder equation:

```
f2 f1 f0| FUNCTION
--------|---------------------
0  0  0 | out <- 0
0  0  1 | out <- A and B
0  1  0 | out <- B'
0  1  1 | out <- 0'
1  0  0 | out <- A plus B plus 0
1  0  1 | out <- A plus B' plus 1
1  1  0 | out <- A plus 0 plus 1
1  1  1 | out <- A plus (1)' plus 1
                 = A plus 11..10 plus 1
                 = A plus 11...1 plus 0
                 = A plus 0' plus 0
```

From this table, a processor component can be designed, as given in figure 2.1.

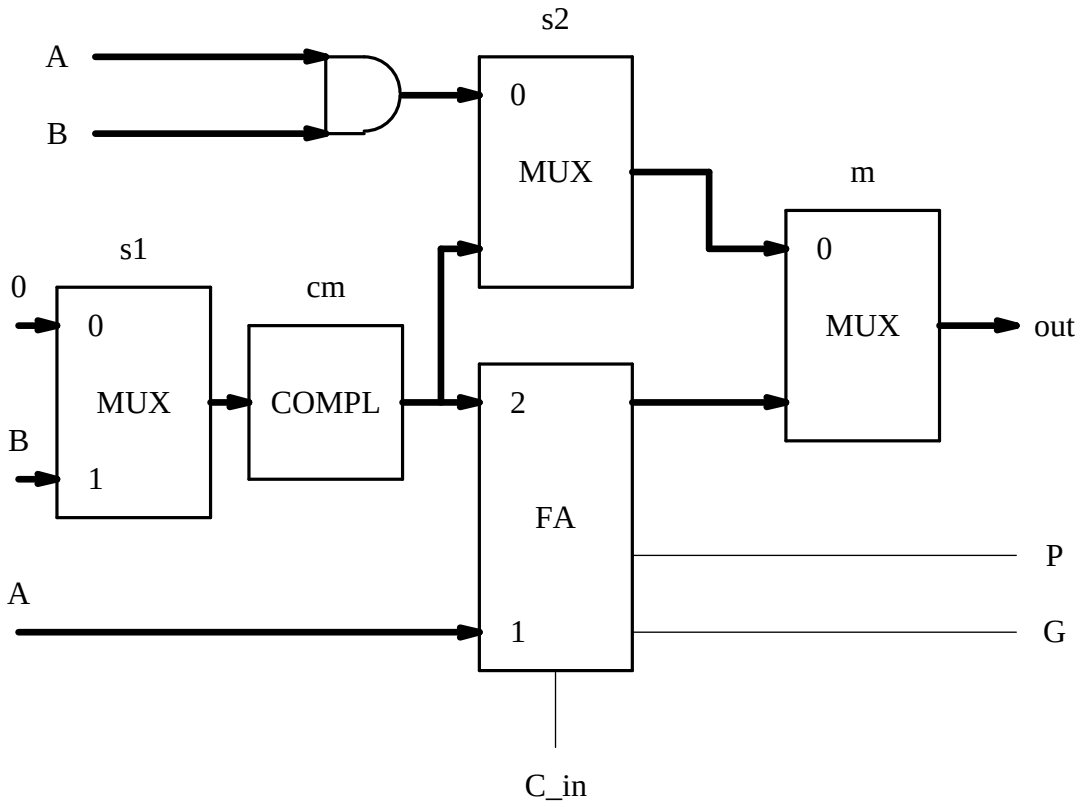

Figure 2-1: 8-function ALU processor component

The controller for this processor consists only of a control point enabler, since, there is no sequencing (in effect, a 1-state ASM). The control point selector functions are defined in the following function table:

f2	f1	f0	s1	cm	s2	m	C_in
0	0	0	0	0	1	0	x
0	0	1	x	x	0	0	x
0	1	0	1	1	1	0	x
0	1	1	0	1	1	0	x
1	0	0	1	0	x	1	0
1	0	1	1	1	x	1	1
1	1	0	0	0	x	1	1
1	1	1	0	1	x	1	0

In addition, the carry-in to the least significant bit will need to be defined as indicated in the function select table for the third operand above.

3. UNSIGNED MULTIPLICATION

3.1. COMBINATIONAL MULTIPLIERS

The design of a commbinational multiplier is based on how we multiply manually. To multiply a 4 bit number by a 2 bit number generates two partial products `a3,a2,a1,a0` and `b3,b2,b1,b0` that must be added together to determine the final product:

```
            x3 x2 x1 x0
                  y1 y0
            -----------
            a3 a2 a1 a0
         b3 b2 b1 b0
         --------------
      p5 p4 p3 p2 p1 p0
```

where ai = xi*y0 and bi = xi*y1

If P0 = a3,a2,a1,a0 and P1 = b3,b2,b1,b0

then

X times Y = (2 times P1) plus P0

Therefore begin by designing a 4x1 multiplier that is capable of adding a term after multiplying its operands:

By inspection of the analysis of manual multiplication above, the 4x1 multiplier can be used to compute a partial product and any accumulated sum from previous partial products to it. Thus two such modules can be "chained" together to obtain the desired 4x2 multiplier:

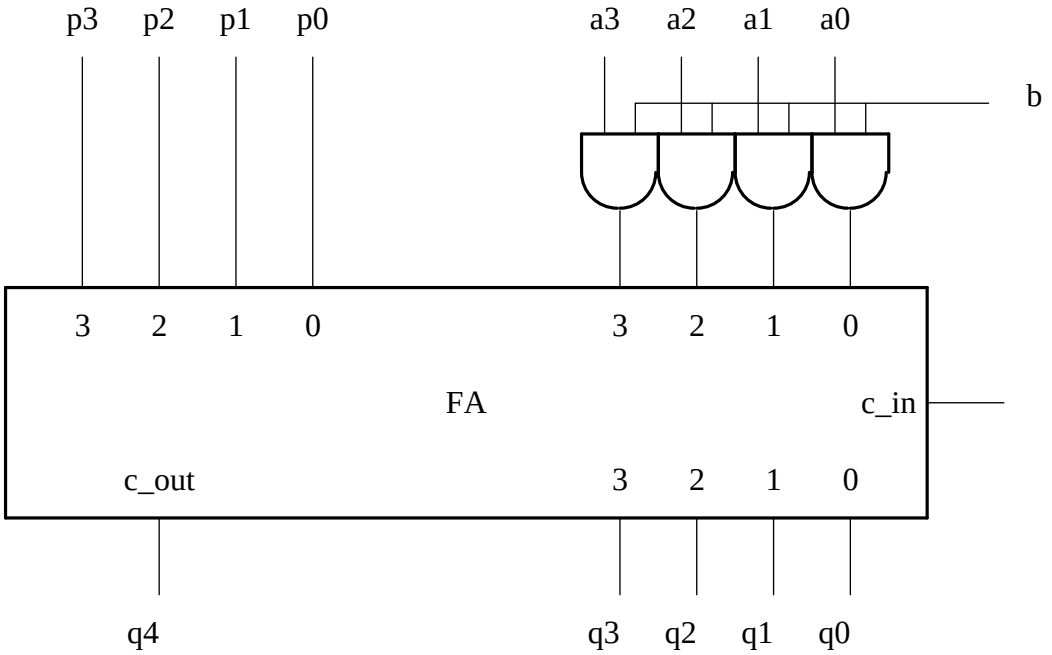

Figure 3-1: A 4x1 combinational multiplier

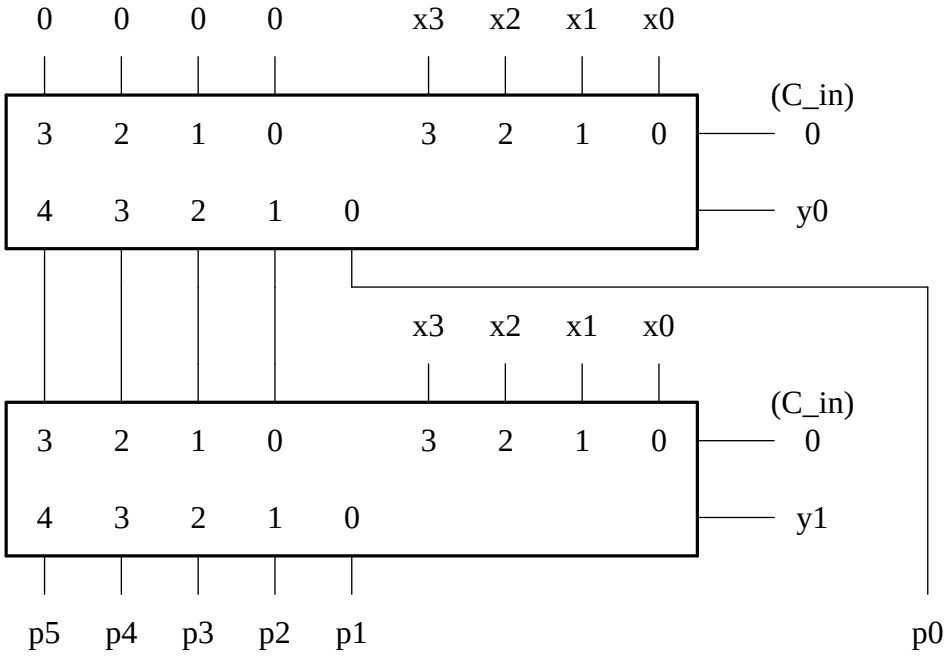

Figure 3-2: A 4x2 combinational multiplier

By including a carry-in (from the full adder circuit on which its structural description is based), the module can be used to build multipliers of any size, by chaining them in two-dimensional arrays. For example to obtain an 8 x 2 multiplier using 4x1 packages, the following array is obtained:

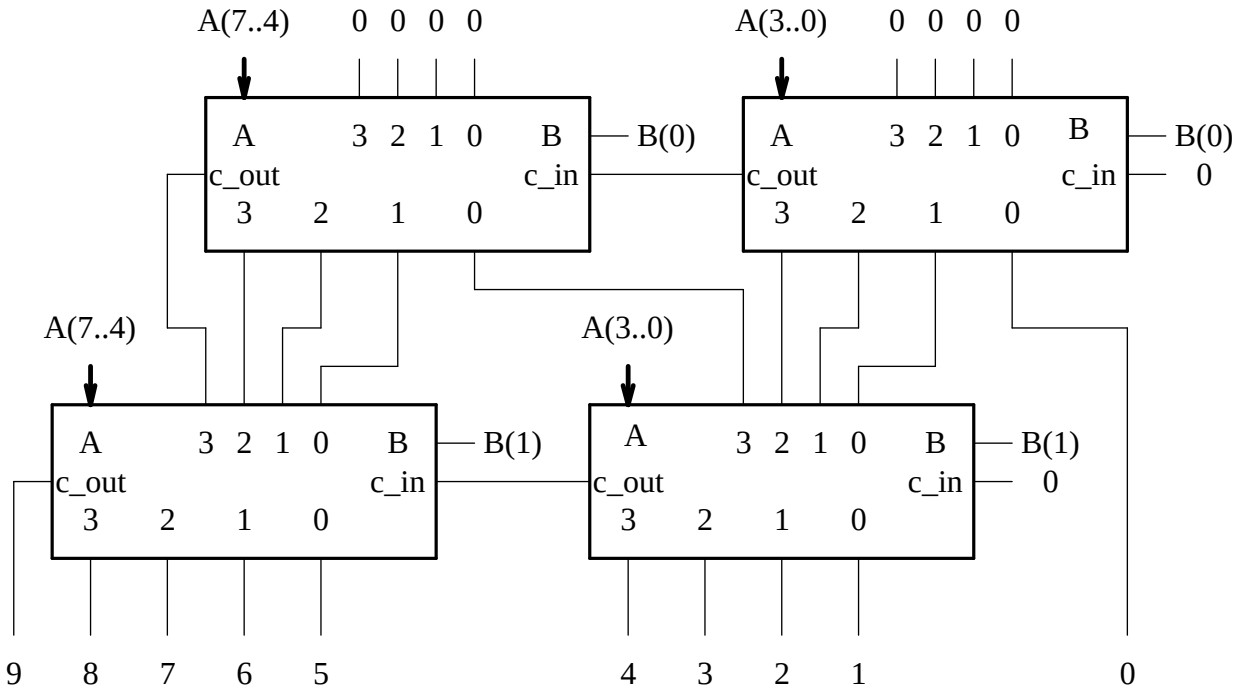

Figure 3-3: An 8x2 combinational multiplier

4. UNSIGNED ADDITION

From the previous two examples, then it is clear that efficient ADDER circuit design contributes to the performance of circuits that perform other arithmetic functions. This section looks at ways to design "fast" ADDERS.

The basic building blocks for addition circuits are defined by the following Boolean functions, where *x*, *y*, and c_{in} denote binary inputs, *s* the sum output and c_{out} the carry output:

1. **1-bit HALF-ADDER**: $s = x \oplus y, \quad c_{out} = xy$. This device computes the sum of two 1-bit quantities. Note that it also provides the Boolean functions **And** and **xor**.

2. **1-bit FULL-ADDER**: $s = x \oplus y \oplus c_{in}, \quad c = xy + c_{in}[x \oplus y]$. This device computes the sum of three 1-bit quantities.

To add two n-bit numbers the following tabular notation is often used:

$$
\begin{array}{r}
c_{n-1}\, c_{n-2} \ldots c_2\, c_1\, 0 \\
x_{n-1}\, x_{n-2} \ldots x_2\, x_1\, x_0 \\
y_{n-1}\, y_{n-2} \ldots y_2\, y_1\, y_0 \\
\hline
c_n s_{n-1}\, s_{n-2} \ldots s_2\, s_1\, s_0
\end{array}
$$

Each column can be implemented with a 1-bit full adder. In each case the carry-in for a given stage is defined by the carry-out from the previous stage to the right.

This method of implementing large adders is inefficient, since the carry-out is required to propagate through the entire system. This means the propagation delay is proportional to the number of bits in each operand. In fact if the propagation delay of any gate is 1 ns, then the propagation delay for an n-bit adder is 2n ns. To speed up addition it is necessary to compute the carry-out sooner. Consider the carry-out at stage $n-1$. Let $g_n = x_n y_n$ and $p_n = x_n \oplus y_n$. Then:

$$c_n = g_{n-1} + c_{n-1} \cdot p_{n-1}$$

This recurrence relation can be solved by repeated substitution:

$$c_1 = g_0 + c_0 \cdot p_0$$

$$\begin{aligned} c_2 &= g_1 + c_1 \cdot p_1 \\ &= g_1 + g_0 p_1 + c_0 p_0 p_1 \end{aligned}$$

$$\begin{aligned} c_3 &= g_2 + c_2 \cdot p_2 \\ &= g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2 \end{aligned}$$

$$\begin{aligned} c_4 &= g_3 + c_3 \cdot p_3 \\ &= g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + c_0 p_0 p_1 p_2 p_3 \end{aligned}$$

These equations are all sums-of-products and so c_i can be computed with two levels of gates as functions of $g_0 \ldots g_3, p_0 \ldots p_3, c_0$. A package that implements logical circuits for these equations is called a *carry-propagate generator* or CPG.

The sum of each stage is defined by $s_i = p_i \oplus c_i$, and so is computed with an additional level of exclusive-or gates. A 4-bit adder can be implemented with a 4-level circuit, sometimes called a *carry-look-ahead adder* or CLA.

To obtain larger adders, the 4-bit CLA can be chained as could the 1-bit full adder. For 16-bit addition, this yields a circuit where the carry bit is propagated through 16 levels as opposed to the 32 levels of a chain of 16 1-bit adders. So an adder based on a 4-bit CLA is twice as fast. The logic diagram for a 16 bit adder can be found 2 pages ahead.

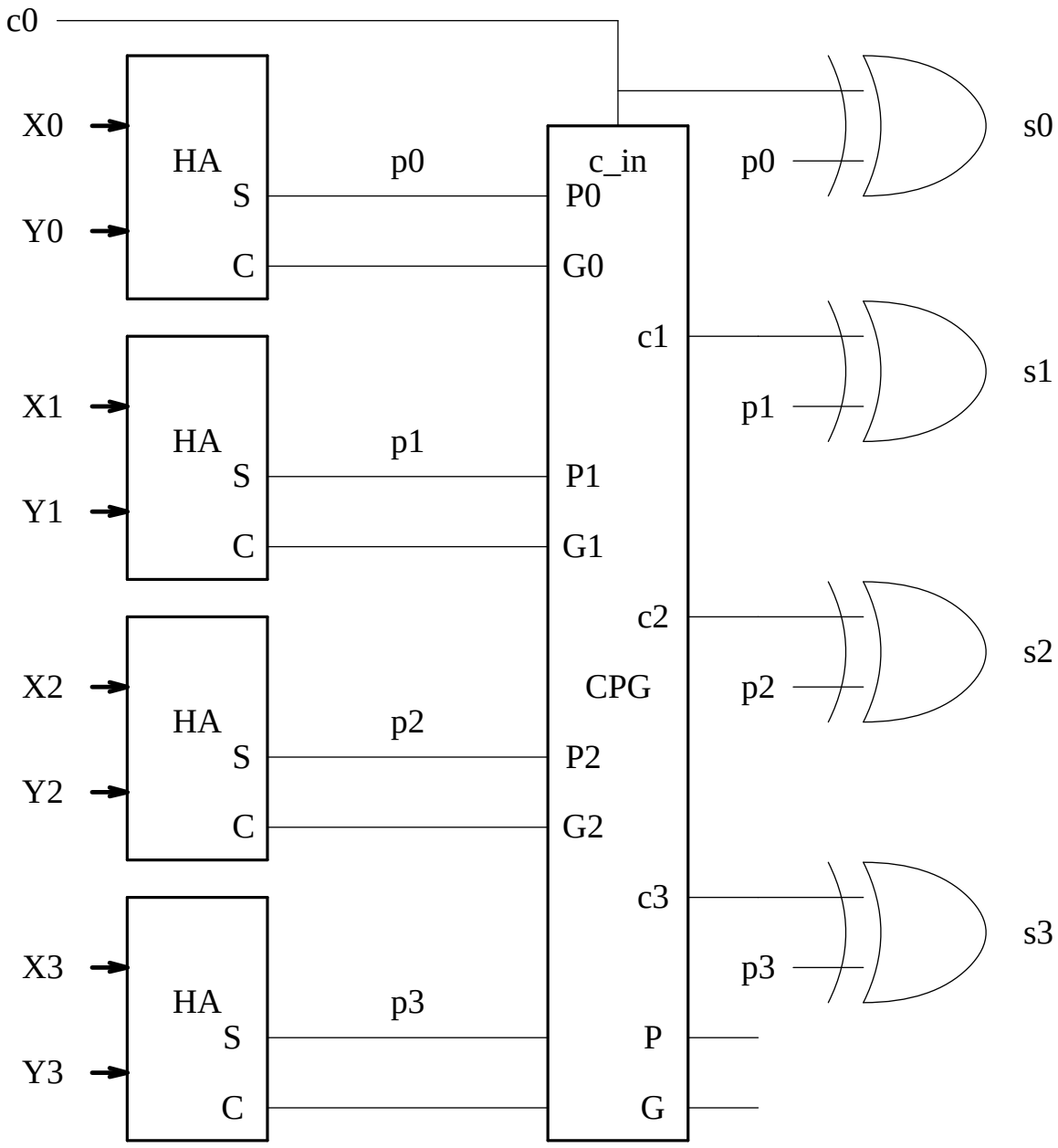

Figure 4-1: 4-bit addition using a carry-propagate generator

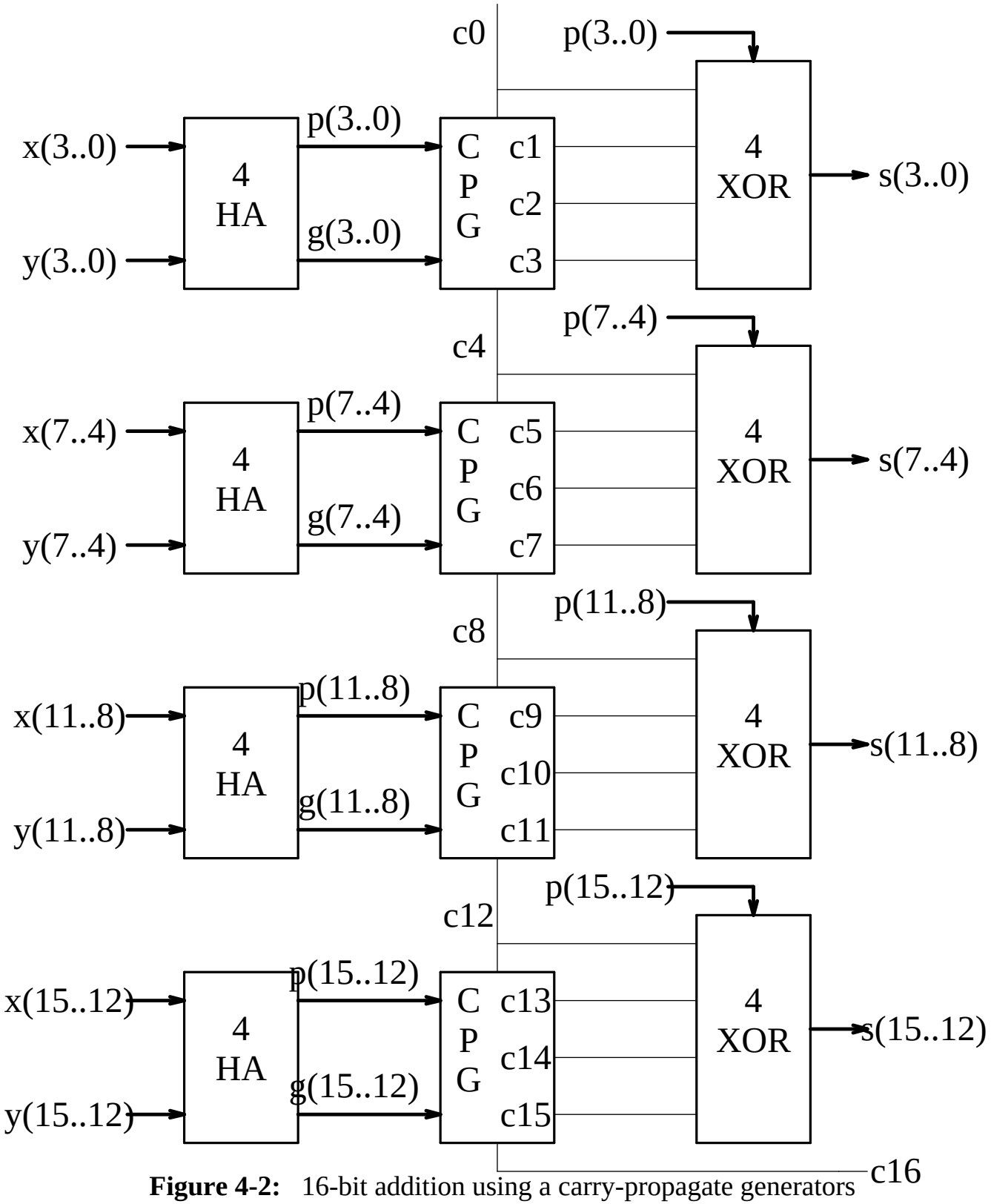

Figure 4-2: 16-bit addition using a carry-propagate generators