# Chapter 5. Fast Addition (Part 2)

## 5.4. Conditional Sum Adder

1. The main issue to speed up an addition is how to overcome its long carry propagation chain.
2. The idea of conditional sum is
   (a) Instead of solving a true sum with a true carry-in signal, we solve two conditional sums with two assumed carry-in signal values. Carry-out signal is also solved in the similar way as the sum.
   (b) The true output (sum and carry-out) is then chosen from two sets of conditional outputs once the true carry-in is available.

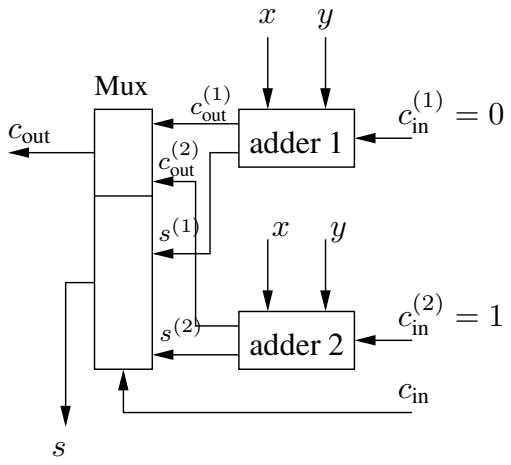



Figure 5.1: A diagram for conditional sum adder.

3. Two Sub-steps for Conditional Sum Addition:
   - First sub-step is to obtain conditional results (including sum bit and carry-out).
   - The next sub-step is to use the carry-out signal to select the results.
   - Then the above two steps can be repeated for higher weight.

| $i$ | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| $A$ | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | |
| $B$ | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | |
| $c_{i+1}^{(0)}$ | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | Step 0 |
| $s_i^{(0)}$ | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | |
| $c_{i+1}^{(1)}$ | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | |
| $s_i^{(1)}$ | 1 | 0 | 0 | 1 | 0 | 1 | 1 | | |
| $c_{i+1}^{(0)}$ | 1 | | 0 | | 1 | | 0 | | Step 1 |
| $s_i^{(0)}$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | |
| $c_{i+1}^{(1)}$ | 1 | | 0 | | 1 | | | | |
| $s_i^{(1)}$ | 1 | 0 | 1 | 1 | 0 | 1 | | | |
| $c_{i+1}^{(0)}$ | 1 | | | | 1 | | | | Step 2 |
| $s_i^{(0)}$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | |
| $c_{i+1}^{(1)}$ | 1 | | | | | | | | |
| $s_i^{(1)}$ | 0 | 1 | 1 | 1 | | | | | |
| $c_{i+1}$ | 1 | | | | | | | | Step 3 |
| $s_i$ | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | |

Table 5.1: 8-bit conditional sum adder

- The number of repeated steps should be proportional to the logarithm of the operand size.

4. An example of 8-bit conditional sum adder is illustrated as follows:

   - Perform conditional sum addition $A + B$, where $A = 1010\ 1100$ and $B = 1100\ 0101$.
   - The time delay is proportional to the logarithm of the operand size. For $n$-bit conditional sum addition, the number of steps is $\log_2 n + 1$.

5. A conditional sum adder is not practically efficient if the conditional sums are calculated from bit by bit, but this idea can be used to build more efficient carry select adders which we will discuss in the next section.

# 5.5. Carry Select Adder

1. Consider a $k$-bit conditional sum unit with carry-in $c_j$ and carry-out $c_{i+1}$, $i = j + k - 1$, which can be portion of a larger adder of $n$-bit, where $k < n$.

2. The unit includes two $k$-bit ripple-carry adders, and one multiplexer unit that can select one out of two groups of multi-bit signals.

3. Let the carry-in signal $c_j$ to one $k$-bit RCA is 0 ($c_j^{(1)} = 0$) and that to the other RCA is 1 ($c_j^{(2)} = 1$).

4. Before a true carry-in signal $c_j$ is available, the two conditional sums are generated directly from the input operands with the given values of $c_j^{(1)} = 0$ and $c_j^{(2)} = 1$, respectively.

5. When the true carry signal $c_j$ is available, what we need to do is to use $c_j$ to select one out of two conditional sum outputs as the output. That is, to select one group out of the two groups of outputs, $c_{i+1}^{(1)}, s_{i...,j}^{(1)}$, and $c_{i+1}^{(2)}, s_{i,...,j}^{(2)}$.

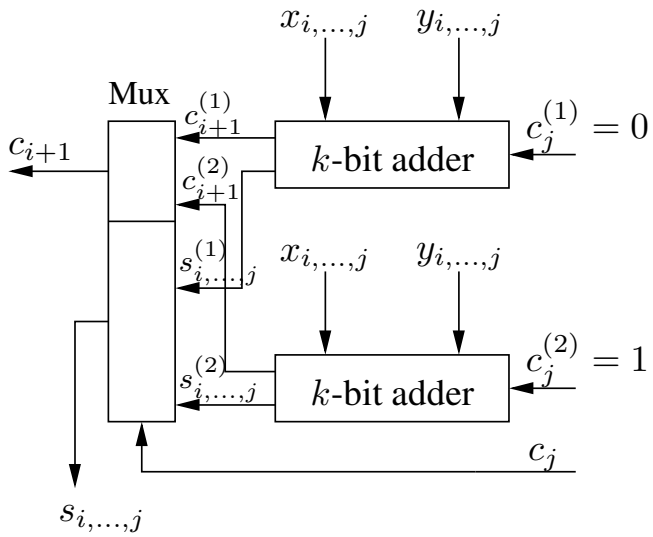



Figure 5.2: A segment (Group $\ell$) in a Carry-Select Adder

6. If the time delay contributed by this $k$-bit conditional sum unit is measured from the availability of the true carry-in $c_j$ to the true sum bits $s_m$ and carry-out $c_{j+1}$ are selected, then it should be equal to the time delay caused by the multiplexer which is $2\Delta_G$.

7. Assume that a large $n$-bit adder is divided into $L$ groups, each group of length $k_1, k_2, \ldots, k_L$, where $\sum_{i=1}^{L} k_i = n$.

8. Consider group $\ell$ of length $k_\ell$:

   - It takes $(\ell - 1) \times 2\Delta_G$ for its carry-in $c_j$ signal to be available. On the other hand, the latency of $k_\ell$-bit RCA is $2k_\ell\Delta_G$.
   - Assume the inputs to the multiplexor (MUX) in group $\ell$ are available at exactly the same time as $c_j$. Then we have $(\ell - 1) \times 2\Delta_G = 2k_\ell\Delta_G$, or $k_\ell = \ell - 1$ for $k_\ell \geq 1, \ell = 1, 2, \ldots, L$.
   - Then optimum setting of group size can be given by: $k_1 = 1, k_2 = 1, k_3 = 2, k_4 = 3, \ldots, k_L = L - 1$. Note that first group $k_1 = 1$ does not contain the multiplexer part since $c_0 = 0$ is usually assumed for a $n$-bit adder.

9. For $n$-bit adder, $1 + L(L-1)/2 \geq n$, or $L(L-1) \geq 2(n-1)$. Clearly, the time delay of a $n$-bit carry-select adder is $T \sim O(\sqrt{n})$.

## 5.6. Manchester Adder

1. Consider a one-bit addition with inputs $x_i, y_i, c_i$ and outputs $s_i, c_{i+1}$. The logic equations involved are

$$\begin{cases} s_i &= x_i \oplus y_i \oplus c_i \\ c_{i+1} &= x_i y_i + c_i(x_i \oplus y_i) \end{cases}$$

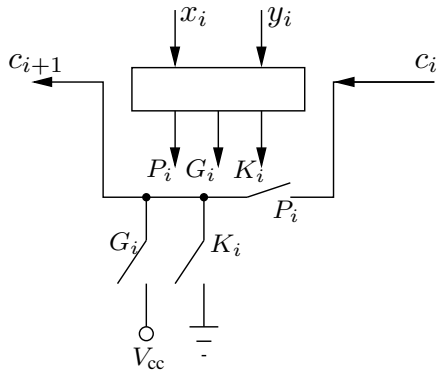
2. Define three signals $P_i, G_i, K_i$ for the Manchester adder as follows

$$\begin{cases} P_i &= x_i \oplus y_i & \text{called carry propagate:} & \text{If } P_i = 1 \text{ then } c_{i+1} = c_i. \\ G_i &= x_i y_i & \text{called carry generate:} & \text{If } G_i = 1 \text{ then } c_{i+1} = 1. \\ K_i &= \overline{x}_i \overline{y}_i & \text{called carry kill:} & \text{If } K_i = 1 \text{ then } c_{i+1} = 0. \end{cases}$$

3. The outputs for a Manchester adder can be given as

$$\begin{aligned} s_i &= P_i \oplus c_i \\ c_{i+1} &= \begin{cases} c_i & \text{if } P_i = 1, \\ 1 & \text{if } G_i = 1, \\ 0 & \text{if } K_i = 1. \end{cases} \end{aligned}$$

4. A Manchester adder can be implemented with four logic gates, three signal-controlled switches, and some buffer.




5. The time delay contributed by a one-bit Manchester adder in a large adder is very small, and it is usually equivalent to that of a buffer.

6. One bit Manchester adder is not often used but its idea can be utilized to build a carry skip adder as we will discuss in the next section.

## 5.7. Carry Skip Adder

1. Consider a section of $k$-bit in a larger adder of $n$-bit, $k < n$, with carry-in $c_j$ and carry-out $c_{i+1}$, $i = j + k - 1$.

2. Define two signals $P_{i:j}, G_{i:j}$ as follows

$$\begin{cases} P_{i:j} = P_i P_{i-1} \\ G_{i:j} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \cdots + P_i P_{i-1} \cdots P_{j+1} G_j \end{cases}$$

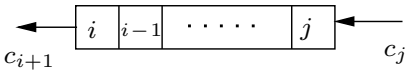
   where $P_{i:j}$ is called *group-carry propagate* and $G_{i:j}$ is called *group-carry generate*. We also have $P_m = x_m \oplus y_m$ and $G_m = x_m y_m$ for all $m = j, j+1, \ldots, i$.

3. Note that for a section of $k = i - j + 1$ bits,

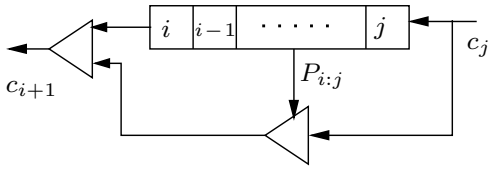
$$\begin{cases} \text{If} \quad P_{i:j} = 1 \quad \text{then} \quad c_{i+1} = c_j. \\ \text{If} \quad G_{i:j} = 1 \quad \text{then} \quad c_{i+1} = 1. \end{cases}$$

4. Building a carry skip adder:

- Let a section of $k$-bit be built with Ripple-Carry Adder, along with a generation of the signals $P_{i:j}$ and $G_{i:j}$.




$(a)$




$(b)$

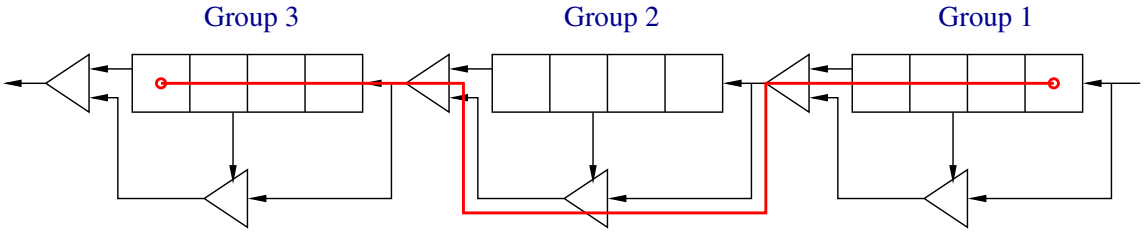
- So its carry-out signal $c_{i+1}$ can be obtained as

$$c_{i+1} = \begin{cases} c_j & \text{if } P_{i:j} = 1, \\ 1 & \text{if } G_{i:j} = 1, \\ 0 & \text{if neither } P_{i:j} \text{ nor } G_{i:j} \text{ equal to } 1. \end{cases} \tag{5.1}$$

$$= G_{i:j} + P_{i:j}c_j.$$

- A diagram for this section $k$-bit to indicate how the carry-out signal is generated can be show in the figure.

5. Example 1 (Carry skip adder): Now consider an example of 12-bit carry-skip adder consisting of three groups each of size 4-bit.




- Define three variables by letting

$$\begin{cases} t_r & \text{denote carry ripple time through one bit} \\ t_{s(k)} & \text{denote the time to skip on group} \\ t_b & \text{denote the time delay of a buffer (OR gate)} \end{cases}$$

- The longest carry propagation path is shown in the figure as in red, which corresponds to the worst case for time delay incurred in this 12-bit adder.

$$T_{\text{carry}} = 3t_r + t_b + t_s + t_b + 3t_r \approx 15\Delta_G,$$

where we assume that $t_r = 2\Delta_G$ and $t_b = t_s = \Delta_G$.

6. For $n$-bit adder of $n/k$ groups, we have

$$\begin{aligned} T_{\text{carry}} &= (k-1)t_r + t_b + \big(\frac{n}{k} - 2\big)(t_s + t_b) + (k-1)t_r \\ &= (4k + \frac{2n}{k} - 7)\Delta_G. \end{aligned}$$

7. To solve the optimal size of $k$:

$$\frac{\partial T_{\text{carry}}}{\partial k} = 0 \;\Rightarrow\; k_{\text{opt}} = \sqrt{\frac{n}{2}}.$$

It follows

$$T_{\text{opt}} = (4\sqrt{2n} - 7)\Delta_G \sim O(\sqrt{n})\Delta_G.$$

8. Example 2 (Carry skip adder): Estimate the time delay for a 32-bit carry skip adder.

- Since $n = 32$, we solve $k = \sqrt{(n/2)} = 4$ and

$$T_{\text{opt}} = (4\sqrt{64} - 7)\Delta_G = 25\Delta_G.$$

- Compared with other types of adder:
  - Ripple carry adder: $64\Delta_G$.
  - Carry lookahead adder: $13\Delta_G$.
  - Carry select adder: $?\Delta_G$

# 5.8. Carry-Save Adder (CSA)

1. Recall a full adder with inputs $x_i, y_i, c_i$ and outputs $s_i, c_{i+1}$ and then we have the logic equations as follows

$$\begin{cases} s_i &= x_i \oplus y_i \oplus c_i \\ c_{i+1} &= x_i y_i + c_i(x_i \oplus y_i) \end{cases}$$

2. View $x_i, y_i, c_i$ as three input operand bits $x, y, z$, and view $s_i, c_{i+1}$ as two output bits $s, c$:

$$\begin{cases} s &= x \oplus y \oplus z \\ c &= xy + yz + zx \end{cases}$$

Its truth table can be shown as

| $x$ | $y$ | $z$ | $c\ s$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 0 |
| 0 | 0 | 1 | 0 1 |
| 0 | 1 | 0 | 0 1 |
| 0 | 1 | 1 | 1 0 |
| 1 | 0 | 0 | 0 1 |
| 1 | 0 | 1 | 1 0 |
| 1 | 1 | 0 | 1 0 |
| 1 | 1 | 1 | 1 1 |

It can be seen that $(cs)$ can be viewed as a 2-bit binary number that is equal to the number of 1s among the three inputs $x, y, z$.

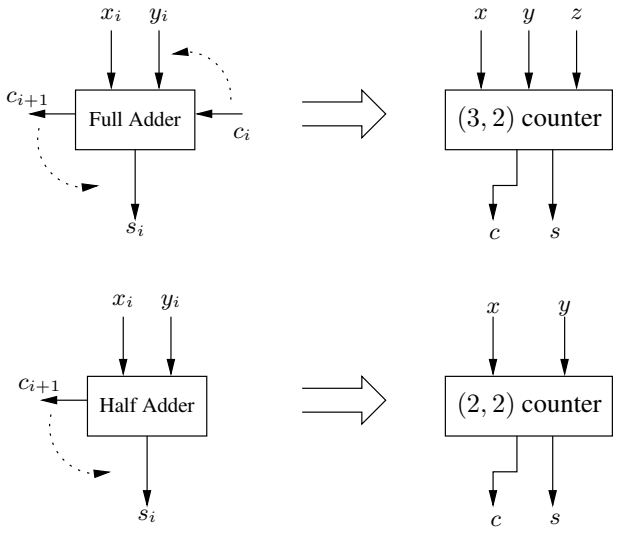



Figure 5.3: $(3, 2)$ counter and $(2, 2)$ counter.

3. This unit is used as cell or building block for construction of carry save adder, and in this case it is called $(\mathbf{3}, \mathbf{2})$ **counter**.

4. Example: A architecture diagram for a carry save adder with four 4-bit operands.

   - Since a 4-bit integer $\in [0, 2^4 - 1] = [0, 15]$, a sum of four such numbers should have no more than 6 bits. (note that $4 \times [0, 15] \subset [0, 2^6 - 1]$).

- Organize the architecture diagram in at most $6$ columns. The inputs to $(3,2)$ counters within a column will have the same weight. From right to left, the six columns are named as Column $i, i = 0, 1, 2, 3, 4, 5$. Then the weight of cell inputs at Column $i$ will have a weight of $2^i$.
- The architecture requires two rows or units of carry save addition (CSA) and one row or unit of carry propagation addition (CPA).

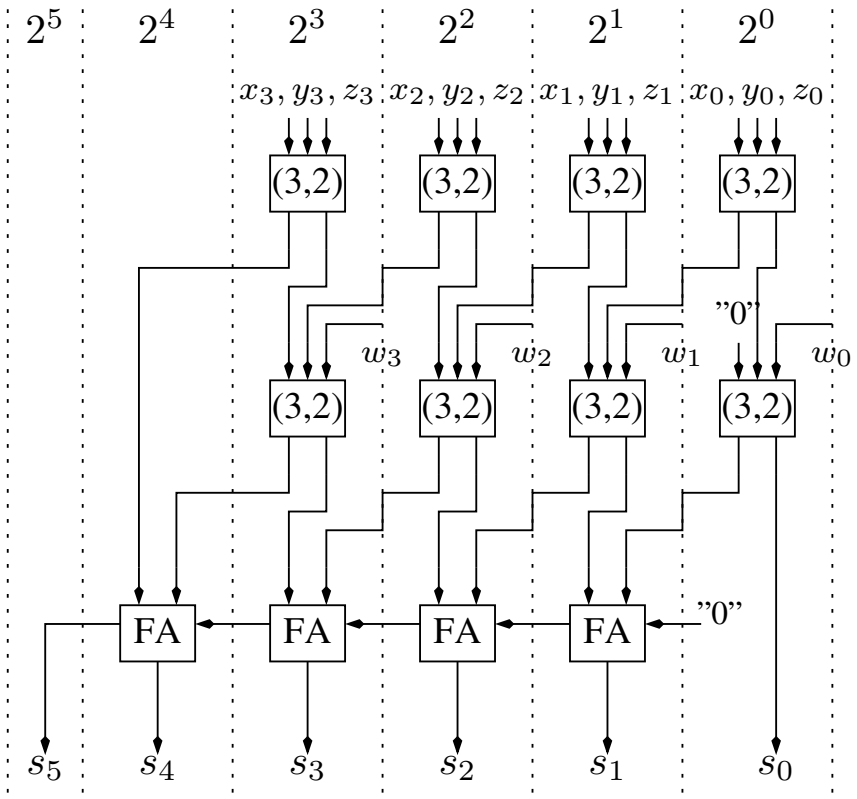



Figure 5.4: CSA for four 4-bit operands using (3,2) as building block.

- The architecture complexities are
  - Circuit complexity: $C = 12\ (3,2)'s = 12$FAs.
  - Critical path delay: $T = 6\Delta_{(3,2)} = 6\Delta_{\text{FA}} = 12\Delta_G$.
- Note that two FAs can be replaced with half adders (HA). Then the complexity is reduced

into

$$\begin{aligned} C &= 10\text{FAs} + 2\text{HAs} \\ T &= 11\Delta_G. \end{aligned}$$

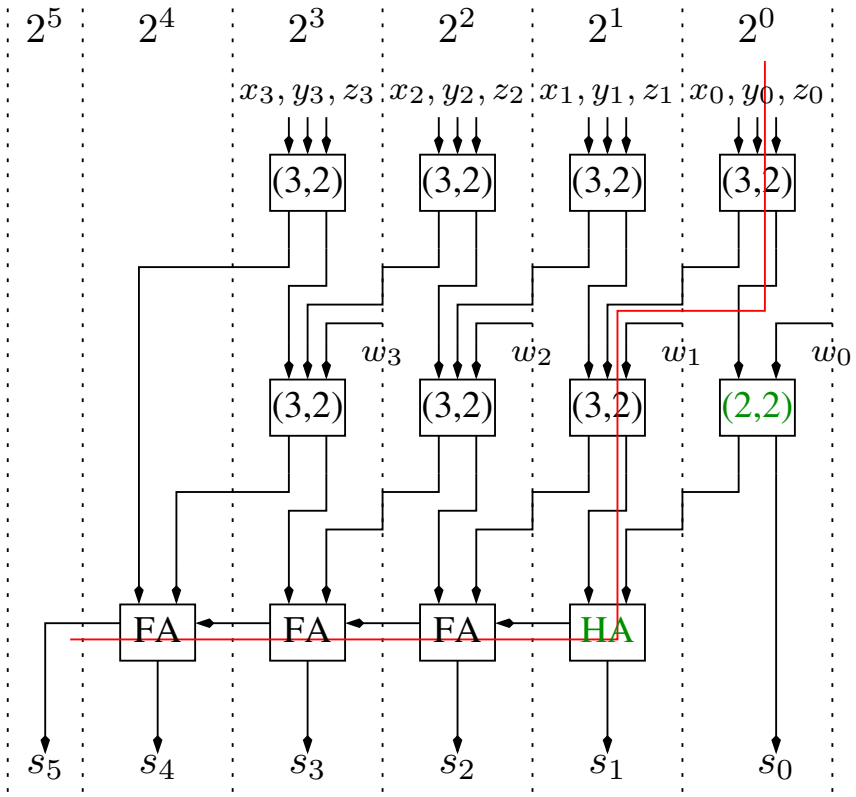



Figure 5.5: CSA for four 4-bit operands using (3,2) and (2,2) counters as building block.

5. A $k$-operand carry save adder needs $k-2$ CSA units and one CPA unit. Its time delay can be roughly estimated as $T = (k-2)T_{\text{CSA}} + T_{\text{CPA}}$, where $T_{\text{CSA}} = \Delta_{(3,2)}$ and $T_{\text{CPA}}$ depends on $k$ and the size of the operands. The sum can be as large as $n + \lceil \log_2 k \rceil$ bits.

6. Wallace tree is one well-known method to further reduce the number of CSA units for $k$-operand carry save adder.

   - Suppose that $\ell$ CSA units are needed for $k$-operand addition. One CSA unit reduces $3n$ bits to $2n$ bits, or reduces $k$ bits to $\frac{2}{3}k$ bits. Assume that $\ell$ CSA units are needed to reduce

$k$ bits to 2 bits, then we have

$$k \times \left(\frac{2}{3}\right)^{\ell} \leq 2 \;\; \Rightarrow \;\; \ell = \frac{\log\left(k/2\right)}{\log\left(3/2\right)}.$$

For $3 \leq k \leq 63$, the values of $\ell$ can be calculated based on the above inequality and they are put into the following table

| $k$ | $\ell$ |
|---|---|
| $3$ | $1$ |
| $4$ | $2$ |
| $5, 6$ | $3$ |
| $7, 8, 9$ | $4$ |
| $10, 11, 12, 13$ | $5$ |
| $14, 15, \cdots, 19$ | $6$ |
| $20, 21, \cdots, 28$ | $7$ |
| $29, 30, \cdots, 42$ | $8$ |
| $43, 44, \cdots, 63$ | $9$ |

- CSA units for $k$-operand carry save addition organized based on the above table is called Wallace Tree.

7. Example: Slice diagram for a $6$-operand CSA Wallace Tree.
   - Only one column of CSA is shown.
   - CPA is not required.
   - Make sure that inputs to the column correspond to the outputs.
8. Other counters
   - $(7,\ 3)$ counter:
   - $(15,\ 4)$ counter
   - Those counters can be built using $(3,\ 2)$ counters (How?), or directly from logic gates.
9. Compressors
   - A compressor $(k;\ m)$ is a variation of counter with extra carry inputs and carry outputs.
   - Note that compressor's output carries do not depend on the incoming carries.
   - A popular example is $(7;\ 2)$ compressor.

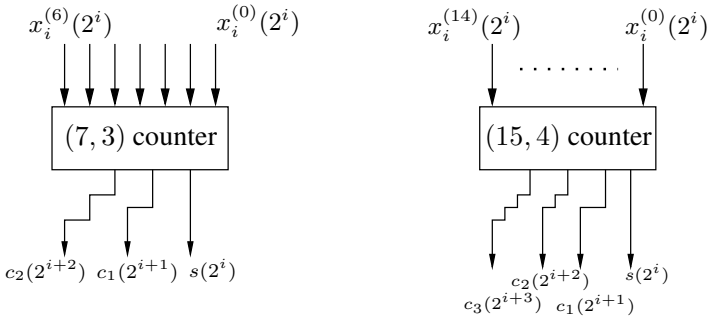



Figure 5.6: $(7,3)$ counter and $(15,4)$ counter.

10. Multi-column counters

    - A $\ell$-column counter is denoted by $(k_{\ell-1}, k_{\ell-2}, \ldots, k_0, m)$, where $k_i$ is the number of input bits at $i^{\text{th}}$ column with weight $2^i$.
    - One condition for such a multi-column counter is

      $$2^m - 1 \geq \sum_{i=0}^{\ell-1} k_i 2^i.$$

      If all $\ell$ columns have the same height $(k_0 = k_1 = \cdots = k_{\ell-1} = k)$, then

      $$2^m - 1 \geq k \sum_{i=0}^{\ell-1} 2^i = k(2^\ell - 1).$$

      .
    - A common example is $(5,5,4)$ counter.

# 5.9. Carry-Free Adder Using Signed-Digit Number Representation

1. In Chapter 2 we have discussed radix-$r$ carry-free addition algorithm using signed-digit (SD) number representation. The algorithm is reviewed below:

2. The circuits block diagram to implement Algorithm 5.1 can be shown below. Step 1 requires a module called carry generator while Step 2 is realized with module called sum generator.

**Algorithm 5.1** Radix-$r$ carry-free addition cell using signed-digit numbers

**Input:** $x_i, x_{i-1}, x_{i-2}, y_i, y_{i-1}, y_{i-2} \in \{\overline{a}, \ldots, \overline{1}, 0, 1, \ldots, a\}$
**Output:** $s_i \in \{\overline{a}, \ldots, \overline{1}, 0, 1, \ldots, a\}$
1: Obtain $u_i$ and $c_{i-1}$ using the algorithm given in Chapter 2.
2: $s_i = u_i + c_{i-1}$.

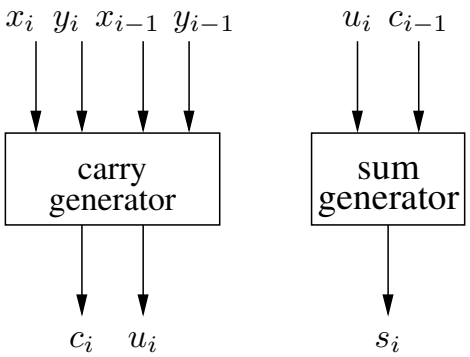



Figure 5.7: Cells to implement Step 1 and Step 2 in carry-free addition

3. When Binary Signed-Digit (BSD) numbers are used, the carry-free addition algorithm is summarized in Table 2 in Chapter 2. A 4-bit carry-free BSD adder is given in the diagram. Note each signal is represented with BSD and it has three possible values in $\{-1, 0, 1\}$.

4. Detailed gate level design for the two modules are left for homework. Note that how to encode the binary signed digit into binary signals can have impact on system efficiency.

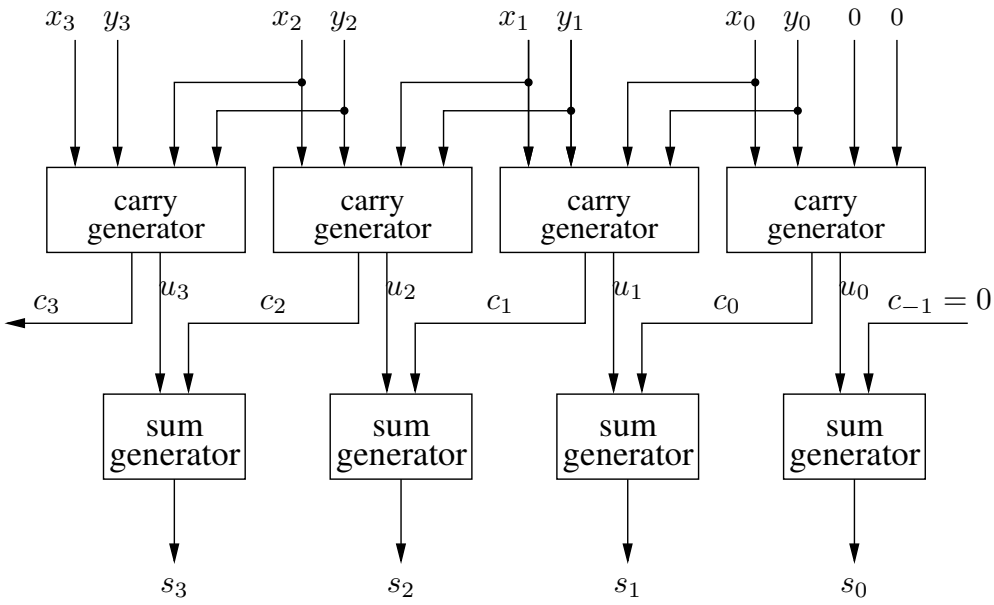



Figure 5.8: 4-bit BSD carry-free adder