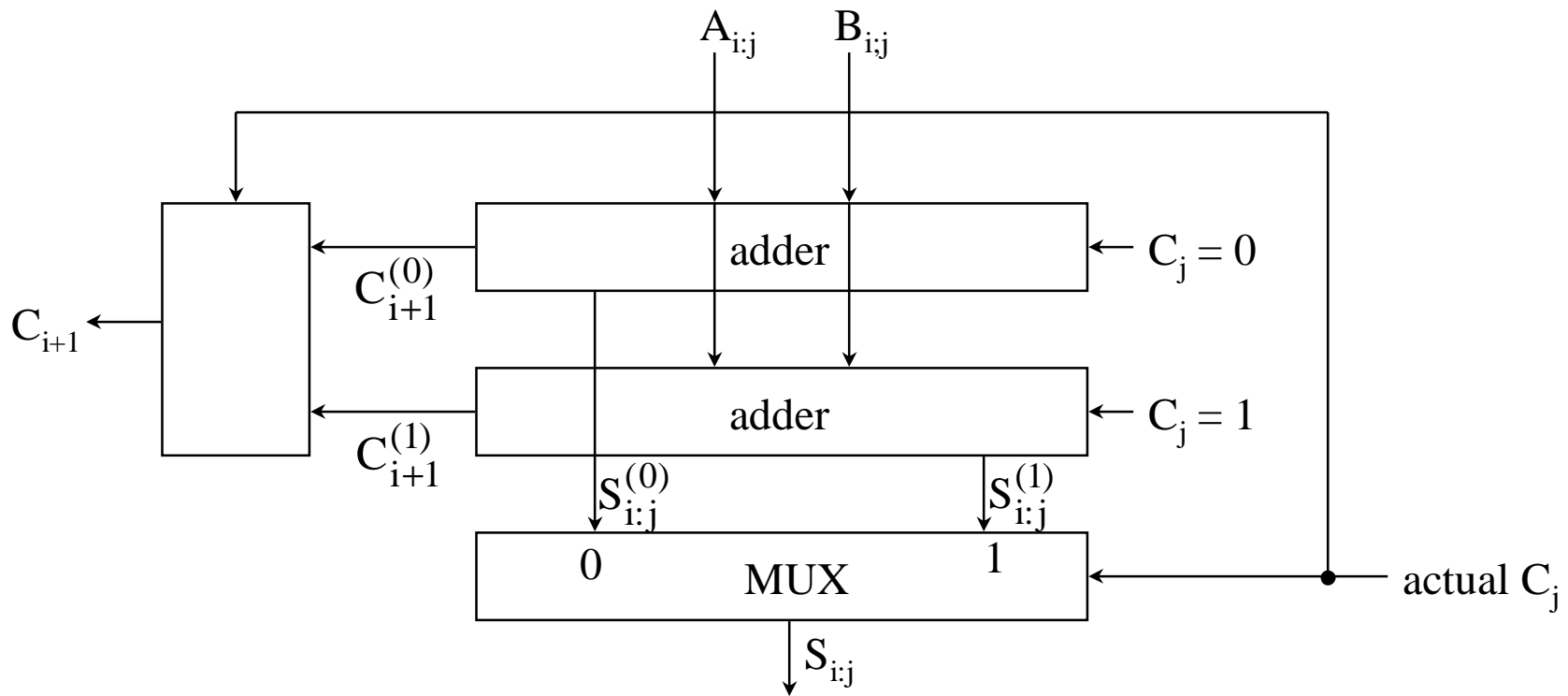


## Architecture of Carry-Select Adders

- Calculate sums and carry out for a group of bits twice (in parallel), assuming that the carry in to the group is a 0 and also assuming that the carry in is a 1. When the carry in arrives, use it to select the correct sum and carry out bits.
- In a carry-select architecture, group-level carries ripple between groups. In some designs, the number of bits within a group is increased for the higher-order groups to balance the delays.



## **Carry Select Adder with Sharing**

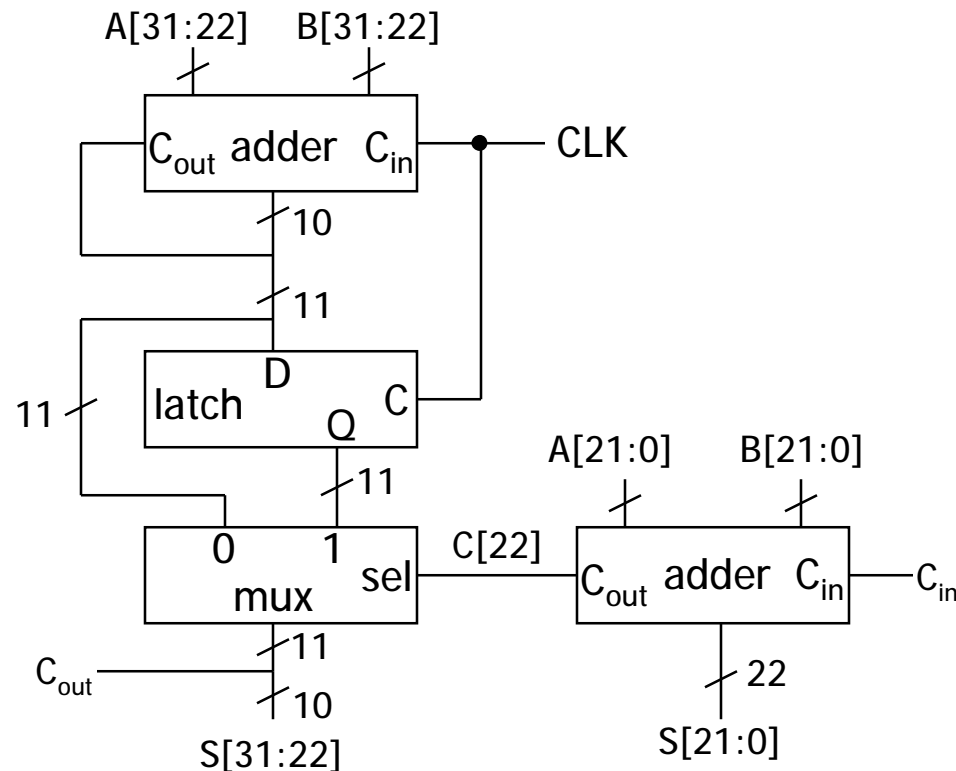
- The basic idea is to use a single, shared adder instead of the two distinct adders which are normally used in each (higher-level) section of a carry select adder.
- Two additions are still performed in each higher-level stage:
  - One is done in the first half of the clock cycle.
  - The second is done in the second half-cycle.
  - Latches are used to store the addition results from the first half-cycle.
- A two-stage CSAS design, partitioned as 22+10 bits, is shown on the next page.
  - At the rising edge of the clock, the latch becomes transparent and the second-stage adder begins computing using a carry in value of 1.
  - These second-stage values are stored in the latch as the clock goes low, at which point the second-stage adders begin a new computation with a carry in value of 0.
  - By the end of the clock cycle, the carry out from the first-stage adder is used to select the appropriate second-stage results.
  - The final sum and carry-out values could then be stored in latches or flip-flops.

Ref: B. Amelifard, F. Fallah and M. Pedram, "Closing the Gap between Carry Select Adder and Ripple Carry Adder: A New Class of Low-power High-Performance Adders," *Proceedings, Sixth International Symposium on Quality Electronic Design*, pp. 148-152, 2005.

## Adder Designs

### Example: Two-Stage 32-bit CSAS

- Note that the first-stage uses the full clock cycle while the second stage must complete an addition in each half-cycle. Thus, the number of bits added in the first stage can be roughly double the number of bits added in the second stage.

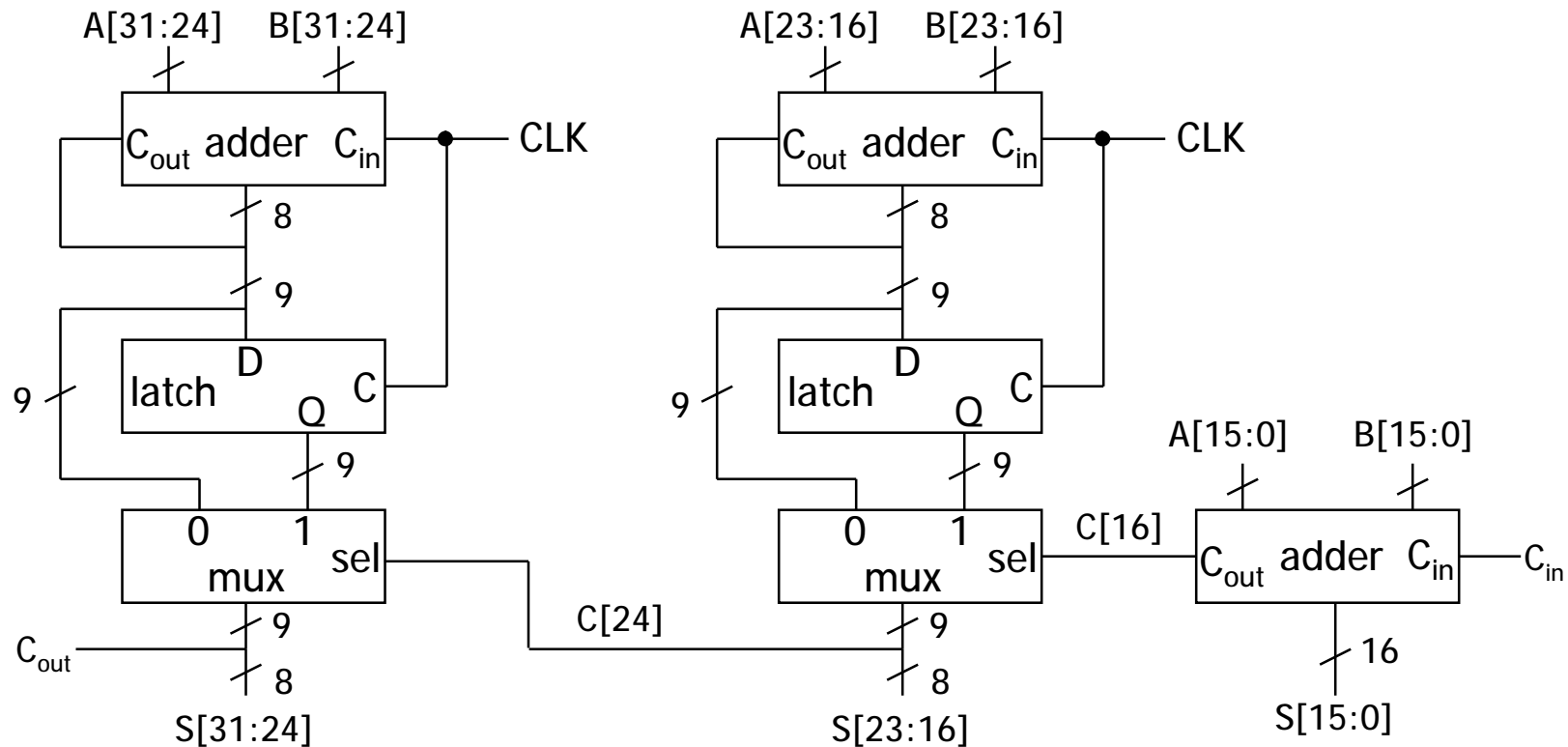


Ref: B. Amelifard, F. Fallah and M. Pedram, "Closing the Gap between Carry Select Adder and Ripple Carry Adder: A New Class of Low-power High-Performance Adders," *Proceedings, Sixth International Symposium on Quality Electronic Design*, pp. 148-152, 2005.

## Adder Designs

### Example: Three-Stage 32-bit CSAS

- The same idea can be extended to more than two stages. For example, a three-stage CSAS, partitioned as 16 + 8 + 8 bits, is shown below:



- Ref: B. Amelifard, F. Fallah and M. Pedram, "Closing the Gap between Carry Select Adder and Ripple Carry Adder: A New Class of Low-power High-Performance Adders," *Proceedings, Sixth International Symposium on Quality Electronic Design*, pp. 148-152, 2005.

## **Carry Look-Ahead (CLA) Adders**

- Given two multi-bit operands A and B, we define the functions  $P_i$  (propagate),  $G_i$  (generate) and  $Psum_i$  (partial sum) at each bit position as follows: (Note that some references define propagate using the exclusive OR function.)

$$P_i = A_i + B_i, \quad G_i = A_i B_i, \quad Psum_i = A_i \oplus B_i$$

- Assume that the carry into bit position  $i$  is  $C_i$ . Then, the sum at bit position  $i$ ,  $S_i$ , and carry into the next higher bit position,  $C_{i+1}$ , are given by:

$$S_i = Psum_i \oplus C_i, \quad C_{i+1} = G_i + P_i C_i$$

- A CLA adder uses the fact that these functions can be computed at all bit positions in parallel, and this information can be used to speed up the carry (and hence the sum) computations.

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$\begin{aligned} C_3 &= G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0) \\ &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0 \end{aligned}$$

$$\begin{aligned} C_4 &= G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0) \\ &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0 \end{aligned}$$

## **Carry Look-Ahead (CLA) Adders - Continued**

- The previous equations determine the first four carries in parallel. While the previous set of substitutions could be continued to higher-order carries, they become impractical due to the large number of terms involved and large fan-in gates that would be required.
- One alternative is to construct a hierarchical structure consisting of *groups*, *blocks* and *sections*. For example, we can define the group generate and propagate functions as:

$$G_{3:0} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0, \quad P_{3:0} = P_3P_2P_1P_0$$

- Using these functions, the equation for  $C_4$  can be written as follows:

$$C_4 = G_{3:0} + P_{3:0}C_0$$

- Similarly, we can define group generate and propagate functions for other, non-overlapping 4-bit groups:

$$G_{i+3:i} = G_{i+3} + P_{i+3}G_{i+2} + P_{i+3}P_{i+2}G_{i+1} + P_{i+3}P_{i+2}P_{i+1}G_i$$

$$P_{i+3:i} = P_{i+3}P_{i+2}P_{i+1}P_i$$

## **Carry Look-Ahead (CLA) Adders - Continued**

- Given the definitions of these group generate and propagate functions, we can create the following group carries:

$$\begin{aligned}C_8 &= G_{7:4} + P_{7:4}C_4 = G_{7:4} + P_{7:4}(G_{3:0} + P_{3:0}C_0) \\ &= G_{7:4} + P_{7:4}G_{3:0} + P_{7:4}P_{3:0}C_0\end{aligned}$$

$$\begin{aligned}C_{12} &= G_{11:8} + P_{11:8}C_8 = G_{11:8} + P_{11:8}(G_{7:4} + P_{7:4}G_{3:0} + P_{7:4}P_{3:0}C_0) \\ &= G_{11:8} + P_{11:8}G_{7:4} + P_{11:8}P_{7:4}G_{3:0} + P_{11:8}P_{7:4}P_{3:0}C_0\end{aligned}$$

$$\begin{aligned}C_{16} &= G_{15:12} + P_{15:12}C_{12} \\ &= G_{15:12} + P_{15:12}(G_{11:8} + P_{11:8}G_{7:4} + P_{11:8}P_{7:4}G_{3:0} + P_{11:8}P_{7:4}P_{3:0}C_0) \\ &= G_{15:12} + P_{15:12}G_{11:8} + P_{15:12}P_{11:8}G_{7:4} \\ &\quad + P_{15:12}P_{11:8}P_{7:4}G_{3:0} + P_{15:12}P_{11:8}P_{7:4}P_{3:0}C_0\end{aligned}$$

- This can be extended to another level of hierarchy (i.e., *blocks*) by forming block generate and propagate functions, which allows us to get block carries. For example:

$$\begin{aligned}G_{15:0} &= G_{15:12} + P_{15:12}G_{11:8} + P_{15:12}P_{11:8}G_{7:4} + P_{15:12}P_{11:8}P_{7:4}G_{3:0} \\ P_{15:0} &= P_{15:12}P_{11:8}P_{7:4}P_{3:0} \Rightarrow C_{16} = G_{15:0} + P_{15:0}C_0\end{aligned}$$

## Parallel Prefix Adders

- Parallel prefix adders make use of recursive relations between group propagate and generate functions. Consider 3 bit positions  $i$ ,  $m$  and  $j$ , where  $i > m > j$ :

$$P_{i;j} = (P_i P_{i-1} \mathbf{L} P_m) (P_{m-1} P_{m-2} \mathbf{L} P_j) = P_{i:m} P_{m-1;j}$$

$$G_{i:m} = G_i + P_i G_{i-1:m}$$

$$= G_i + P_i (G_{i-1} + P_{i-1} G_{i-2:m}) = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2:m}$$

$$= G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \mathbf{L} + P_i P_{i-1} \mathbf{L} P_{m+1} G_m$$

$$G_{i;j} = (G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \mathbf{L} + P_i P_{i-1} \mathbf{L} P_{m+1} G_m) + (P_i P_{i-1} \mathbf{L} P_m) G_{m-1;j}$$

$$= G_{i:m} + P_{i:m} G_{m-1;j}$$

- Define an operator "o" which implements the recursive relations for  $P_{i;j}$  and  $G_{i;j}$ , i.e.:

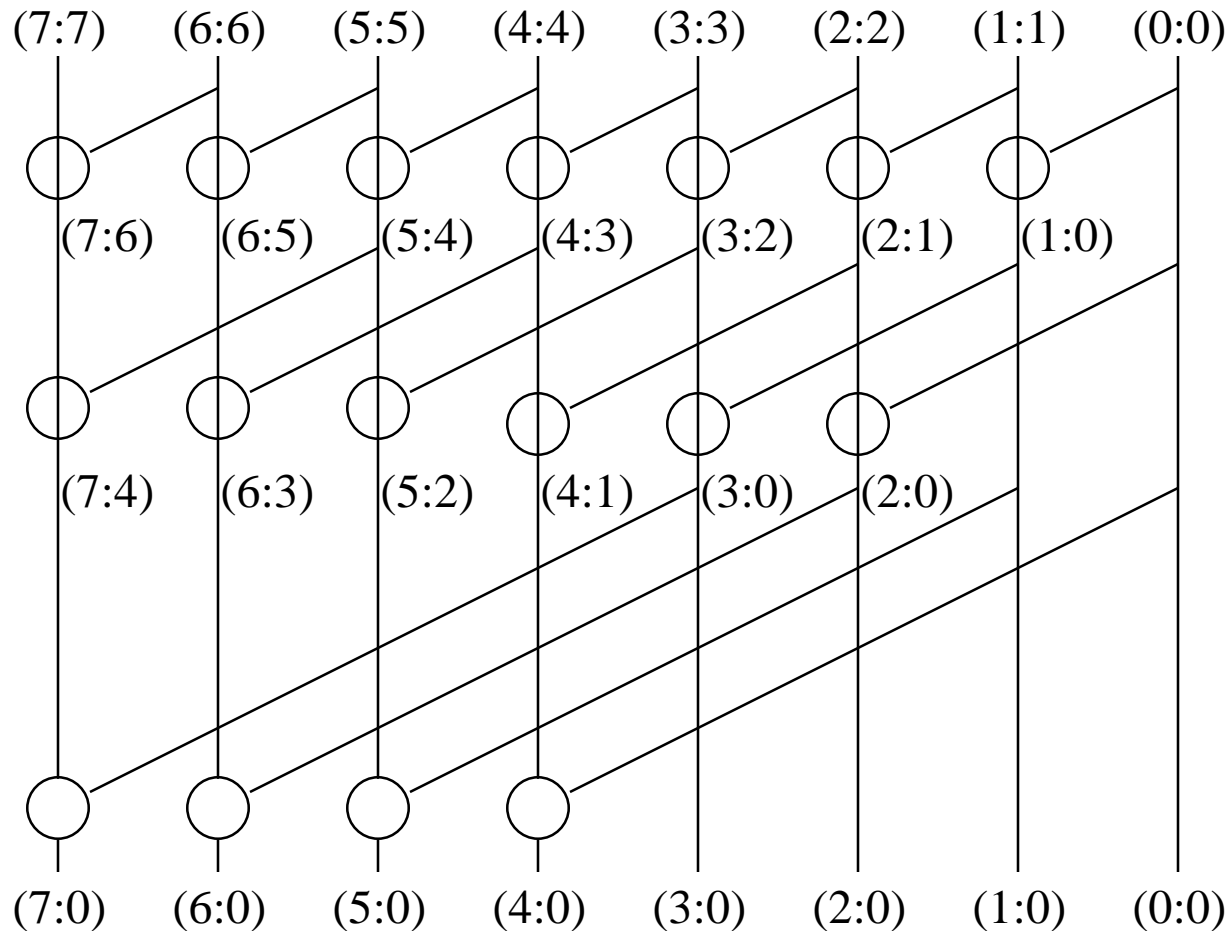
$$(P_{i:m}, G_{i:m}) \mathbf{o} (P_{m-1;j}, G_{m-1;j}) = (P_{i:m} P_{m-1;j}, G_{i:m} + P_{i:m} G_{m-1;j}) \\ = (P_{i;j}, G_{i;j})$$

- It can be shown that this operator is *associative* (i.e., a series of o operators can be evaluated in any order). This provides a great deal of flexibility in devising various adder architectures, as illustrated on the following pages.

Simon Knowles, "A Family of Adders," *15th IEEE Symposium on Computer Arithmetic*, pp. 277-284, 2001.

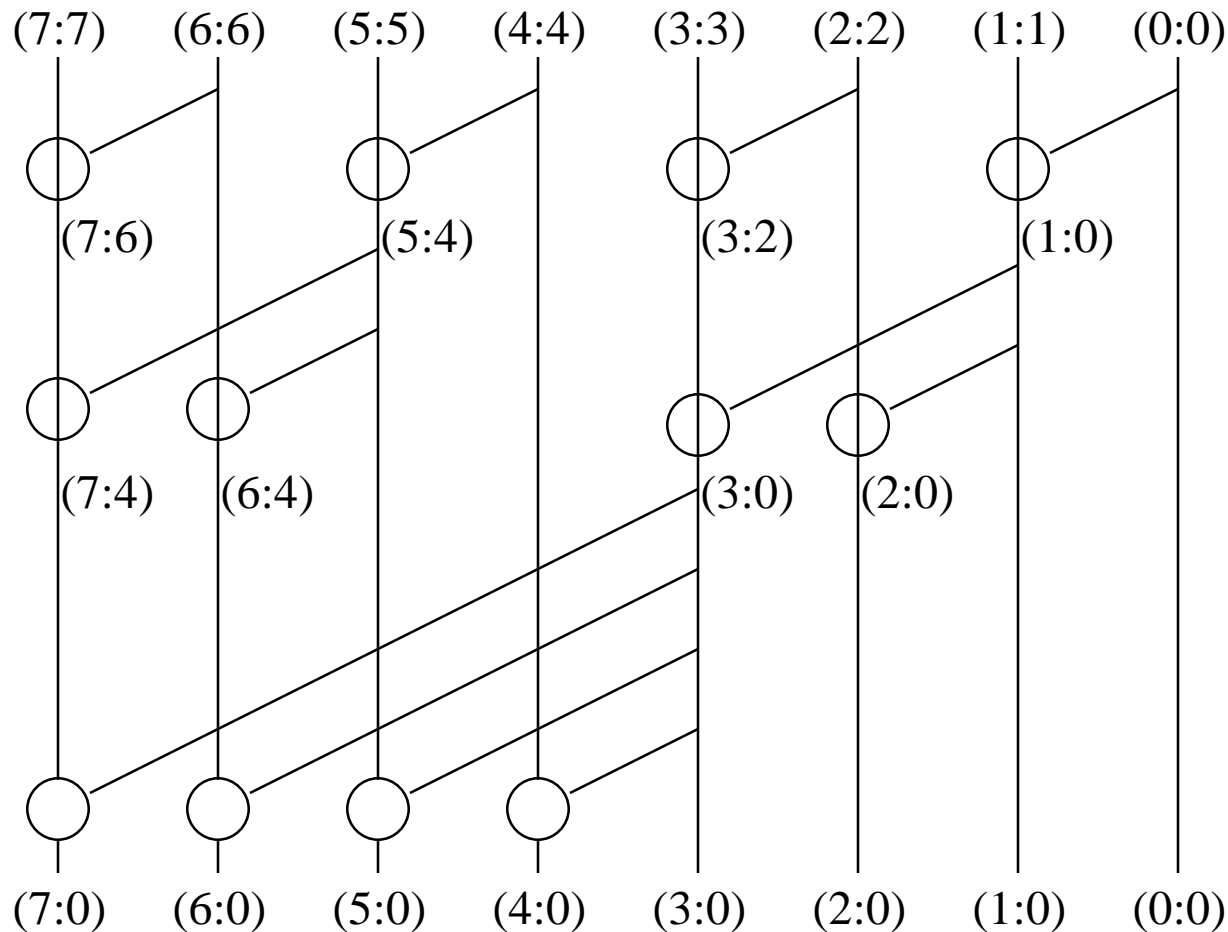
## Kogge-Stone (KS) Architecture

- The  $n$ -bit KS design achieves the minimum logical depth of  $\log_2 n$  levels. It combines values at spans of distance  $2^0, 2^1, 2^2, \dots, 2^{n-1}$ . The 8-bit KS design is:



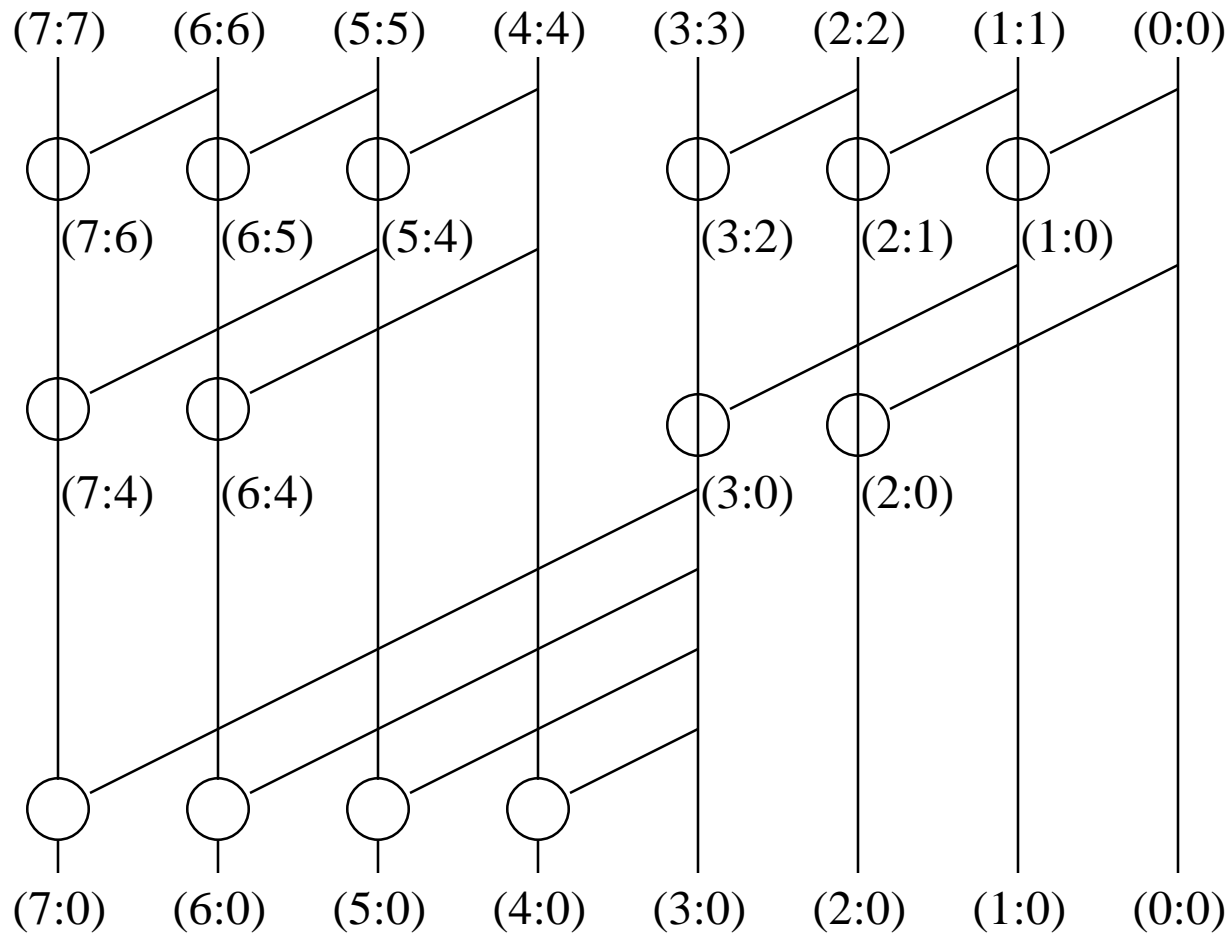
## Ladner-Fischer (LF) Architecture

- The n-bit LF design also achieves the minimum logical depth of  $\log_2 n$  levels. It uses higher fan-out configurations. The 8-bit LF design is shown below:



## **Knowles Architecture**

- Knowles has shown that the KS and LF designs are limiting cases of a set of minimum-depth designs. There are 3 other possible 8-bit designs, such as:



## Han-Carlson (HC) Architecture

- One additional stage at the end is used, but it has low fanout and few devices.

