Reviewing 4-to-2 Adders for Multi-Operand Addition

Peter Kornerup

Dept. of Mathematics and Computer Science

Odense, Denmark

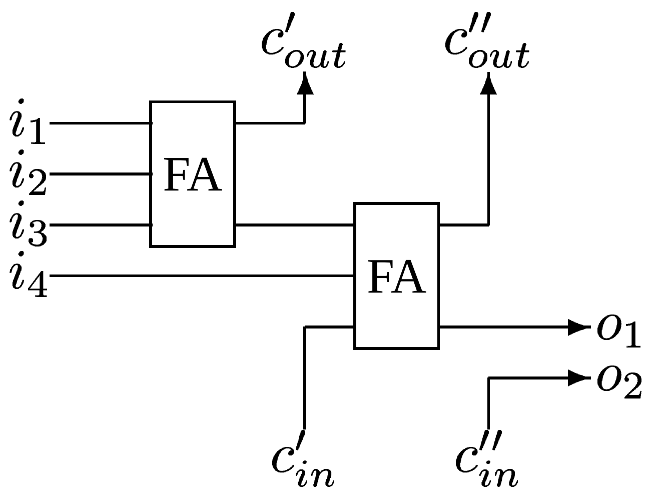
Introduction:

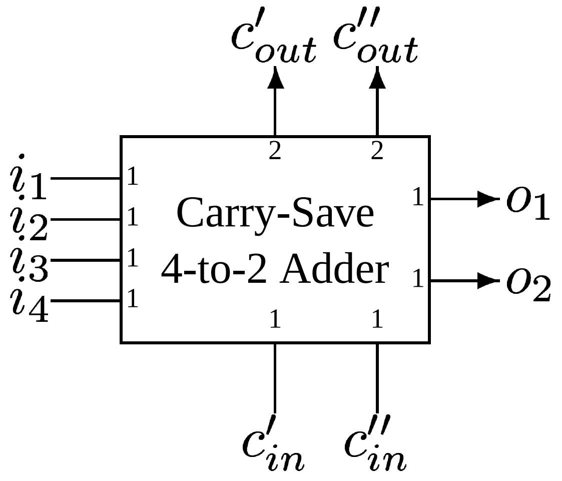
4-to-2 adders:

- allow binary tree structured multipliers.
- usually used with redundant digit set $\{0,1,2\}$ or $\{-1,0,1\}$.
- recently digit set $\{0,1,2,3\}$ was proposed by Ercegovac and Lang [EL97] using a 3-bit encoding.
- Phatak, Goff and Koren [PGK01] proposed various digit sets with encodings $d = \pm 2d^h \pm d^l$ and *equal-weight grouping*, or EWG.

It is shown that they all can be implemented with any standard 4-to-2 carry-save adder, using alternative wiring and possibly a few inverters.

4-to-2 Carry-Save Adder from Full-Adders:

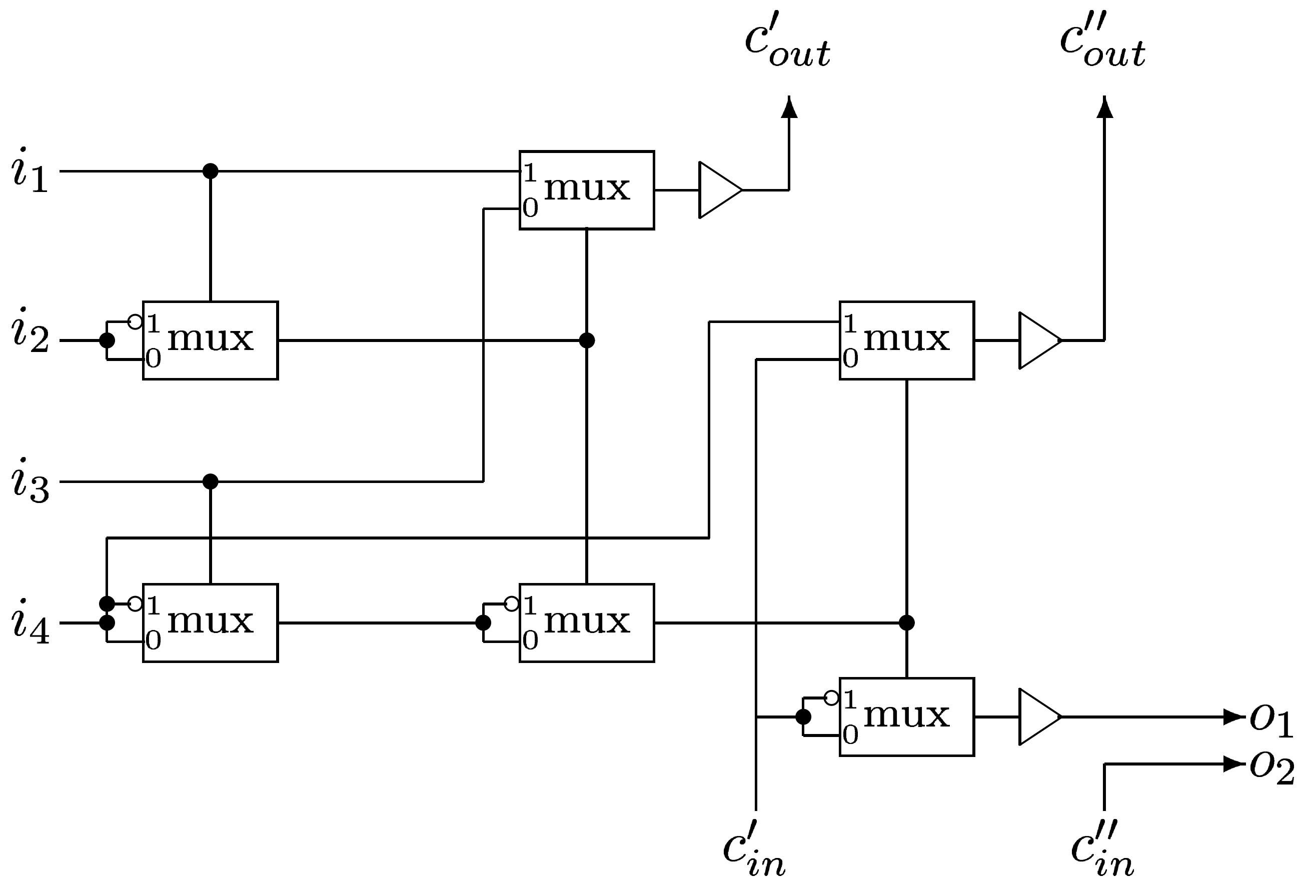




A pair of bits can be interpreted as a digit in the set $\{0, 1, 2\}$, employing the *carry-save encoding*:

$$\begin{array}{ccccc} 0 & \sim & 00 & & \\ 1 & \sim & 01 & \text{or} & 10 \\ 2 & \sim & 11 & & \end{array}$$

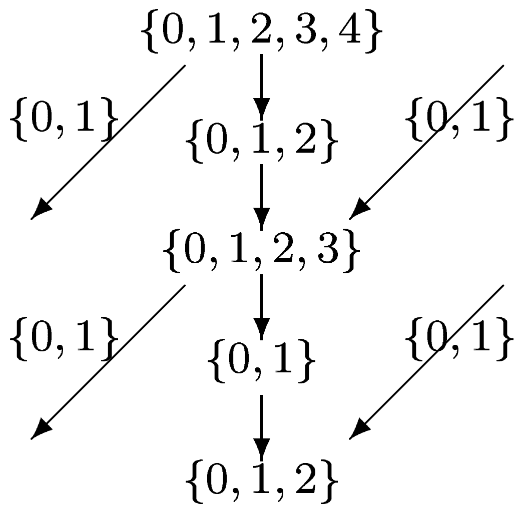
4-to-2 Carry-Save Adder from Multiplexors:



From [OSYSSN95], using pass transistors and a dual-rail coding.

Addition as a Digit Set Conversion:

Carry-Save addition is equivalent to a digit-set conversion [Kor94] from the set $\{0,1,2\}+\{0,1,2\}=\{0,1,2,3,4\}$ into the set $\{0,1,2\}$:



Signal Weights:

Defining equation:

$$2(c'_{out} + c''_{out}) + (o_1 + o_2) = i_1 + i_2 + i_3 + i_4 + c'_{in} + c''_{in}.$$

Assume $o_2 = c''_{in}$, this allows c''_{in} to be added in at no cost. Hence:

$$\begin{aligned} o_1 &= (i_1 + i_2 + i_3 + i_4 + c'_{in}) \bmod 2 \\ o_2 &= c''_{in} \\ c'_{out} + c''_{out} &= (i_1 + i_2 + i_3 + i_4 + c'_{in}) \text{ div } 2, \end{aligned}$$

(c'_{out}, c''_{out}) is a carry-save encoding of the combined carry in $\{0, 1, 2\}$.

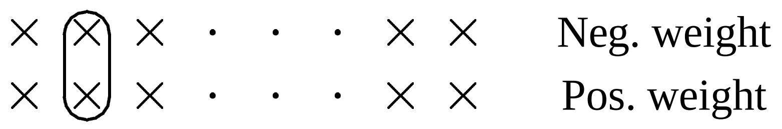
Signed and Weighted Signals:

Theorem 1 *Let a binary signal $b \in \{0,1\}$ have associated weight w, so that the value represented by the signal is $v = w\,b$. Inverting the signal into $1-b$, while at the same time negating the sign of the weight, changes its value into $v' = v - w$, i.e., the value is being biased by the amount $-w$.*

Proof: Trivial since $v' = (-w)(1-b) = wb - w = v - w$. $\square$

Signed-Digit / Borrow-Save

Consider the base 2 digit-set $\{-1, 0, 1\}$ using the *signed-digit* encoding (also denoted *borrow-save*), where two bit strings are considered a string of digits:



obtained by pairing bits using the digit encoding:

$$\begin{array}{rcccc} -1 & \sim & 10 & & \\ 0 & \sim & 00 & \text{or} & 11 \\ 1 & \sim & 01, & & \end{array}$$

where the left-most bit has negative weight.

Signed-Digit Addition:

Addition of two signed-digit operands is a conversion of digit set $\{-1,0,1\}+\{-1,0,1\}=\{-2,-1,0,1,2\}$ into $\{-1,0,1\}$ with conversion diagram:

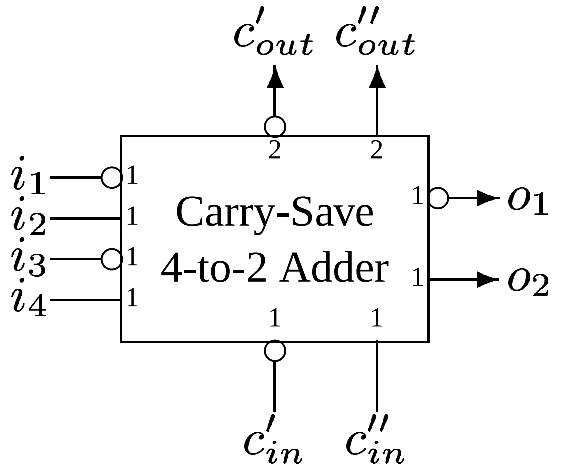
$$\begin{tikzcd}
 & \{-2,-1,0,1,2\} \arrow[d, "\{0,1,2\}"] \arrow[dl, "\{-1,0\}"'] & {} \arrow[dl, "\{-1,0\}"'] \\
{} & \{-1,0,1,2\} \arrow[d, "\{-1,0\}"] \arrow[dl, "\{0,1\}"'] & {} \arrow[dl, "\{0,1\}"'] \\
{} & \{-1,0,1\} &
\end{tikzcd}$$

and defining equation:

$$2(-c'_{out}+c''_{out})+(-o_1+o_2)=-i_1+i_2-i_3+i_4-c'_{in}+c''_{in}$$

Signed-Digit Adder from Carry-Save Adder:

A signed-digit adder:



Observation 2 *In a computational model where inversion is without cost in area and time, radix 2 signed-digit addition can be realized at exactly the same cost as carry-save addition.*

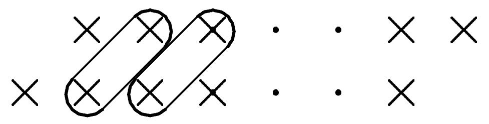
Observation 3 *Multi-operand addition of radix 2 signed-digit operands can be implemented by a tree of carry-save adders, by inverting all negatively weighted signals on input as well as on output external to the tree,*
but with no changes internally in the tree.

Codings of the Form $\pm 2d^h \pm d^l$:

Phatak, Goff and Koren [PGK01] suggested:

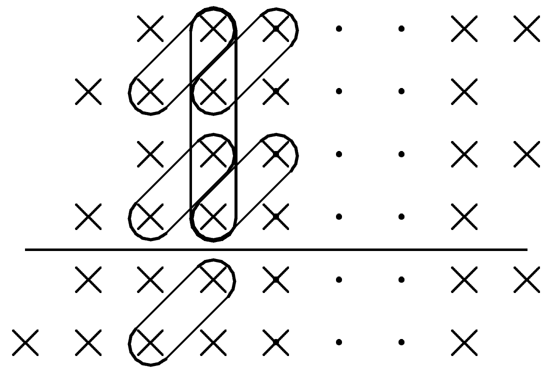
$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{D}^{(SD)} & = & \{-1,0,1\} & \mathcal{D}^{(CS2)} & = & \{0,1,2\} \\
\mathcal{D}^{(SD3^{(-)})} & = & \{-2,-1,0,1\} & \mathcal{D}^{(CS3)} & = & \{0,1,2,3\} \\
\mathcal{D}^{(SD3^{(+)})} & = & \{-1,0,1,2\}. & & &
\end{array}$$

Since the encodings employ weights differing by a factor of 2, neighboring digits d_i and d_{i-1} in a radix representation overlap one another, i.e., d_i^l has the same weight as d_{i-1}^h,



EWG: Equal-Weight Grouping:

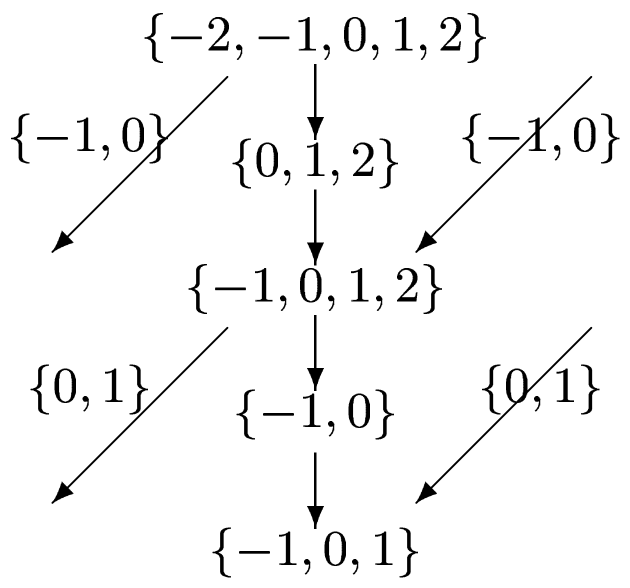
With $\mathcal{D}^{(CS3)} = \{0,1,2,3\}$ addition is conversion from $\mathcal{D}^{(CS3)} + \mathcal{D}^{(CS3)} = \{0,1,2,3,4,5,6\}$, to $\{0,1,2,3\}$, but with EWG:

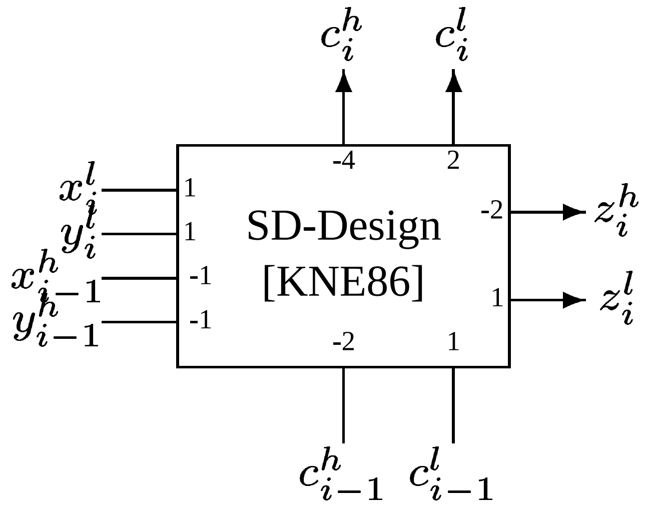


Thus with coding $d = 2d^h + d^l$ for $\mathcal{D}^{(CS3)}$ EWG it is a conversion

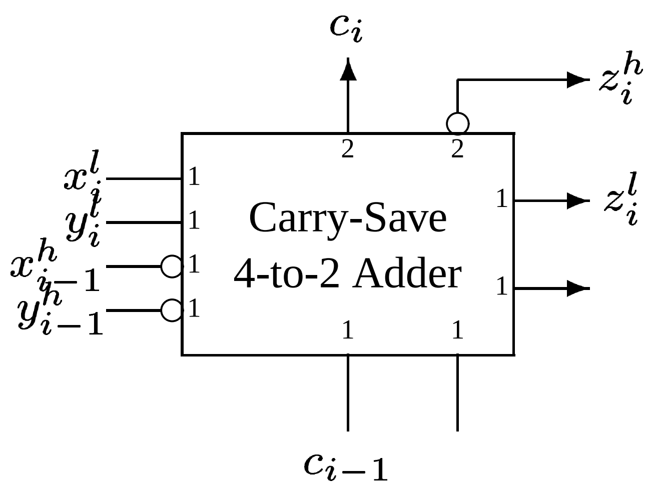
$$\begin{tikzcd}
 & \{0,1,2,3,4\} \arrow[dl, "\{0,1\}"'] \arrow[d] \arrow[dr, phantom] & \\
{} & \{0,1,2\} \arrow[d] & \{0,1\} \arrow[dl] \\
 & \{0,1,2,3\} &
\end{tikzcd}$$

EWG Design, $SD + SD \to SD$:



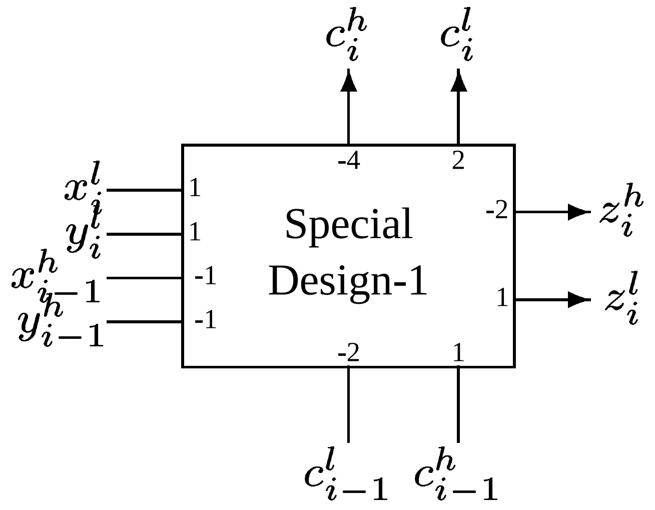


An alternative implementation:



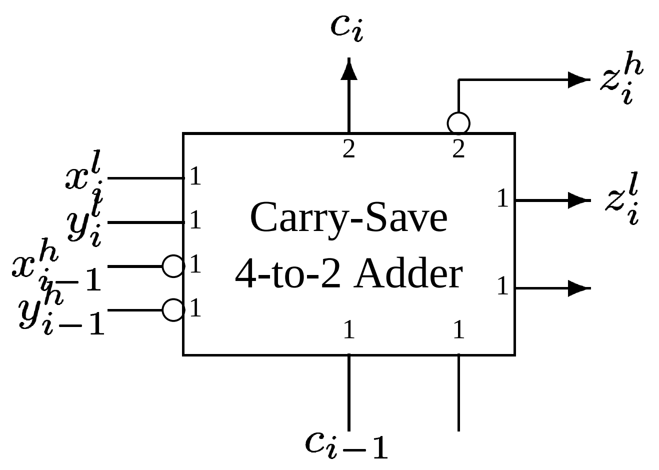
EWG Design, $SD3^{(-)} + SD3^{(-)} \to SD3^{(-)}$:

$$\begin{tikzcd}
 & \{-2,-1,0,1,2\} \arrow[d] \arrow[ddl, "\{-1,0\}"'] & {} \arrow[ddl, "\{-1,0\}"] \\
 & \{0,1,2\} \arrow[d] & \\
{} & \{-1,0,1,2\} \arrow[d] \arrow[ddl, "\{0,1\}"'] & {} \arrow[ddl, "\{0,1\}"] \\
 & \{-1,0\} \arrow[d] & \\
{} & \{-1,0,1\} &
\end{tikzcd}$$

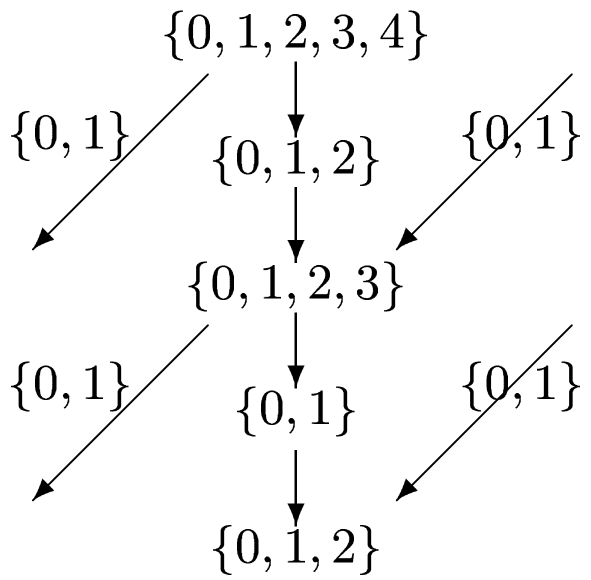


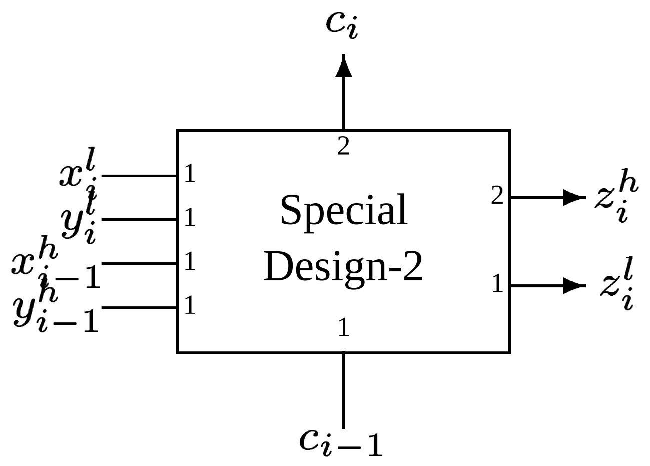
An alternative design:

$$\begin{tikzcd}
 & \{-2,-1,0,1,2\} \arrow[d] \arrow[ddl, "\{0,1\}"'] & {} \arrow[ddl, "\{0,1\}"] \\
 & \{-2,-1,0\} \arrow[d] & \\
{} & \{-2,-1,0,1\} &
\end{tikzcd}$$

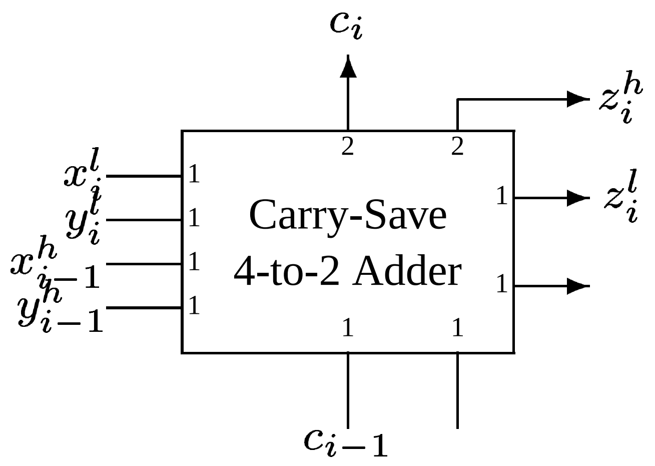


EWG Design, $CS2 + CS2 \rightarrow CS2$:



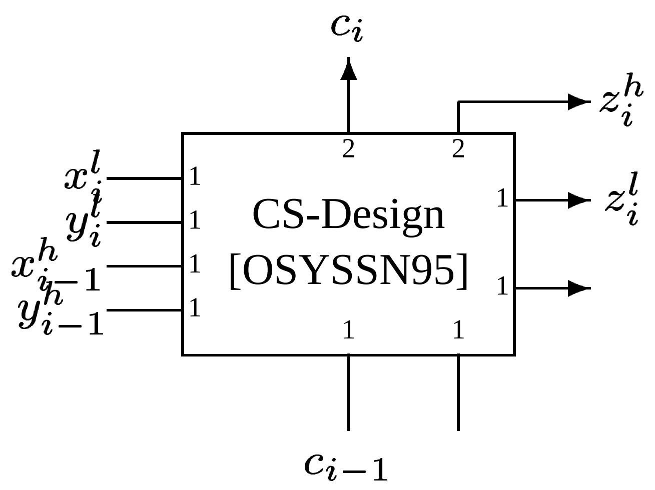


An alternative implementation:

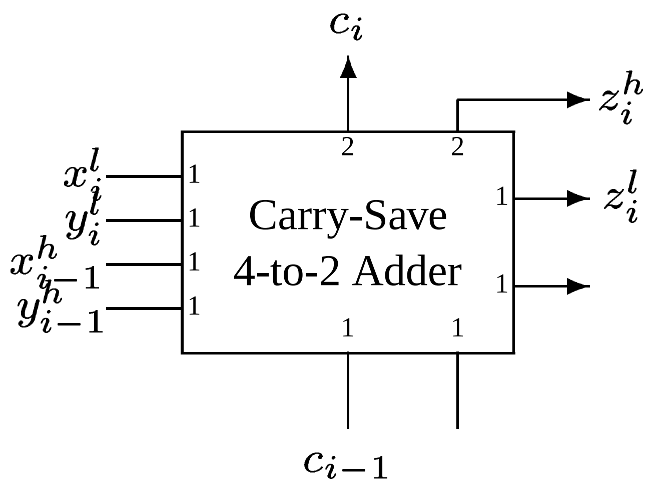


EWG Design: $CS3 + CS3 \rightarrow CS3$:

$$\begin{tikzcd}
 & \{0,1,2,3,4\} \arrow[d] \arrow[ddl, "\{0,1\}"'] & \\
 & \{0,1,2\} \arrow[d] & {} \arrow[dl, "\{0,1\}"'] \\
{} & \{0,1,2,3\} &
\end{tikzcd}$$



But this is the same as:



EWG Timings:

In [PGK01] the following timings were found:

Adder Cell	Critical Path Delay (ns)
SD	0.78750
$SD3^{(-)}$	0.96025
$CS2$	0.66100
$CS3$	0.46580

In [PGK01] it is claimed that:

“multipliers based on $CS3$ can be expected to outperform multipliers based on other redundant representations”

and furthermore using arguments on digit-set cardinalities:

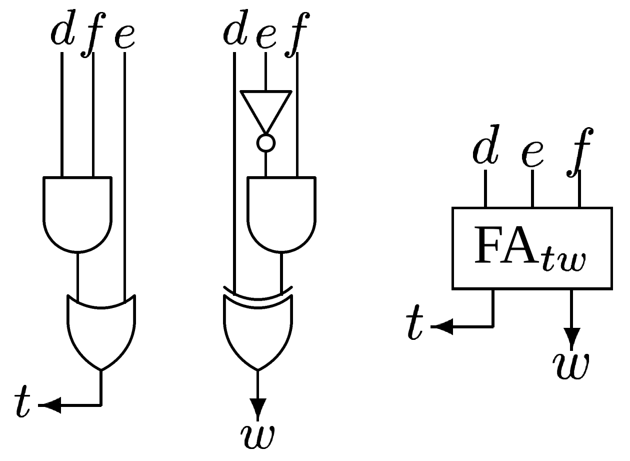
“Therefore, cells such as [SD and $CS2$ above] are fundamentally more complex, hence, bigger and slower.”

Another Example using $\{0, 1, 2, 3\}$:

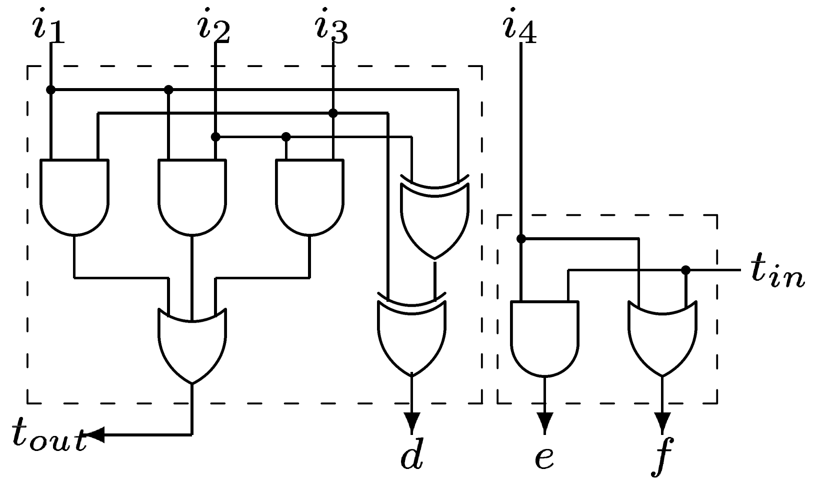
Ercegovac and Lang [EL97] proposed this digit set using (d, e, f) encoding, $(e, f) \neq (1, 0)$, with value $v = d + e + f$. Internally also using (t, w) with $v = 2t + w$:

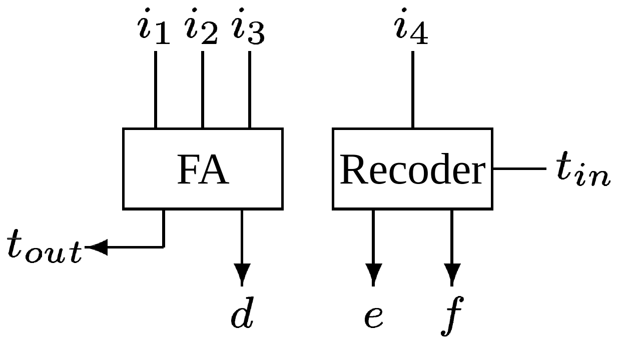
Value	0	1	–	2	1	2	–	3
d, e, f	000	001	010	011	100	101	110	111
t, w	00	01	–	10	01	10	–	11

Using conversion:



Binary Signals into (d, e, f)-Encoding:





Note the recoding taking place at the right to assure $(e, f) \neq (1, 0)$

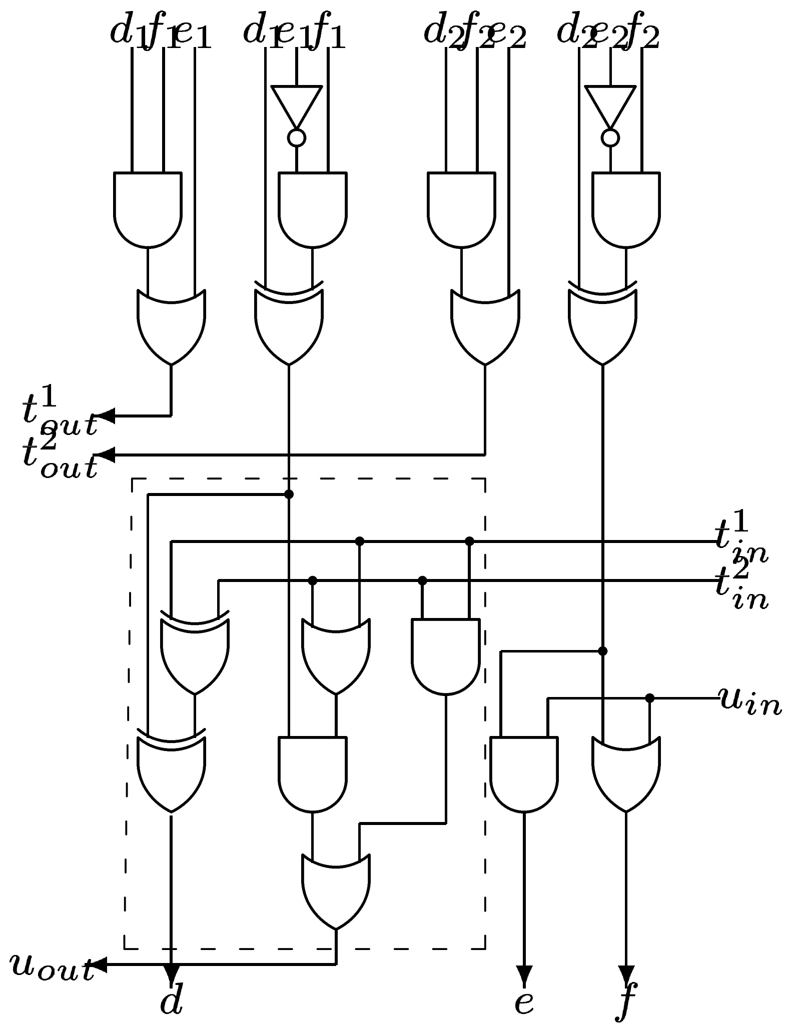
Addition as Digit Set Conversion:

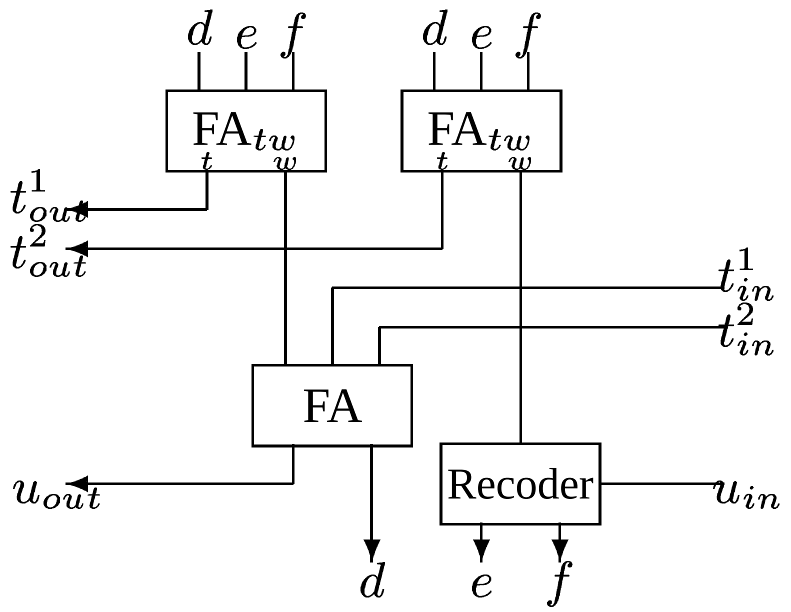
With digit set $\{0,1,2,3\}$, addition is here a conversion from $\{0,1,2,3\}+\{0,1,2,3\}=\{0,1,2,3,4,5,6\}$ into $\{0,1,2,3\}$:

$$\begin{tikzcd}
{} & \{0,1,2,3,4,5,6\} \arrow[dl, "\{0,1,2\}"] \arrow[d, "\{0,1,2\}"] & {} \arrow[dl, "\{0,1,2\}"'] \\
{} & \{0,1,2,3,4\} \arrow[dl, "\{0,1\}"] \arrow[d, "\{0,1,2\}"] & {} \arrow[dl, "\{0,1\}"'] \\
{} & \{0,1,2,3\} & {}
\end{tikzcd}$$

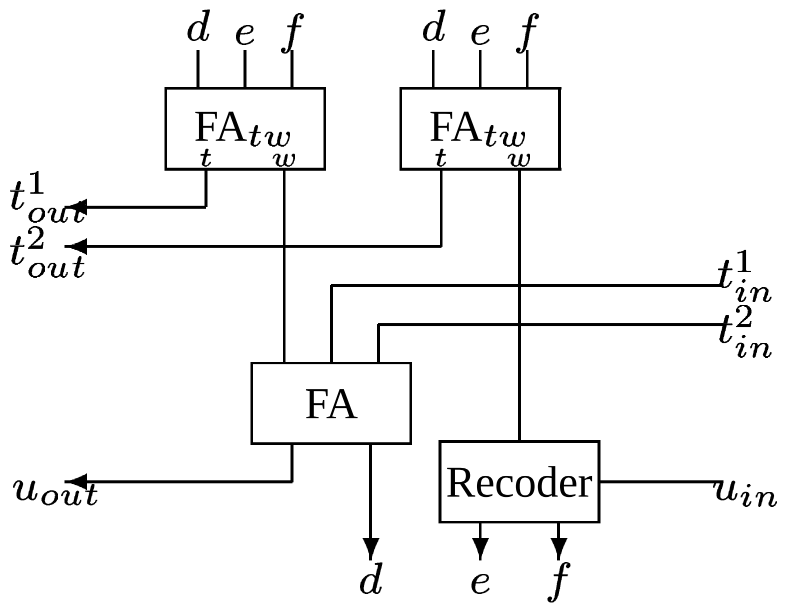
Note that the carry set is $\{0,1,2\}+\{0,1\}=\{0,1,2,3\}$.

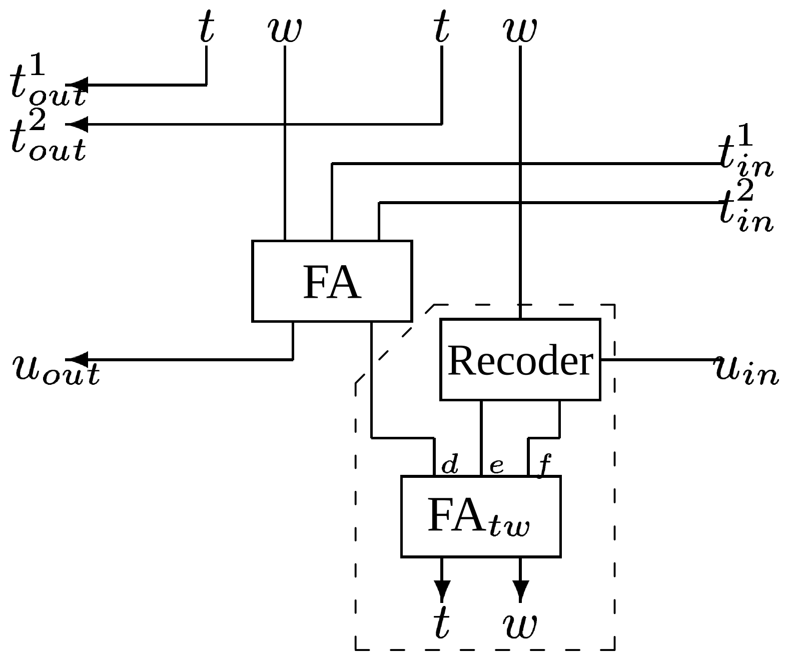
6-to-3 Adder with (d, e, f) Encoding:





Reorganizing the 6-to-3 into 4-to-2 Adder:

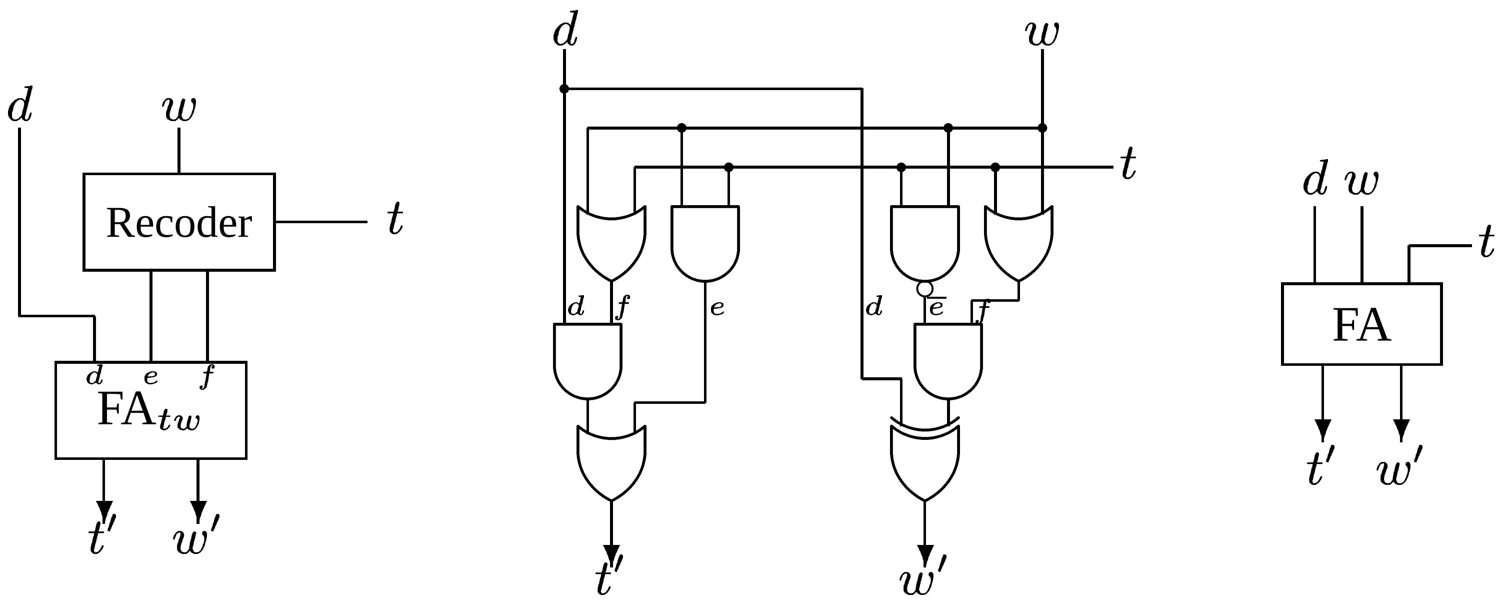




Saves wiring in a multiplier tree, but has the same critical path.

The difference is only in interpreting where the boundaries between layers of the tree are located.

But this is just a Full-Adder



Hence using 6-to-3 adders in a multiplier tree is equivalent to using 4-to-2 adders!

Conclusions:

On multiplier and other multioperand addition trees:

- The difference between the 6-to-3 (d, e, f)-tree and the 4-to-2 (t, w)-tree, is only a question of interpreting where the boundaries between layers of the tree are positioned.
- The difference between the 4-to-2 (t, w)-tree and the standard carry-save (c, s)-tree, is only a question of interpreting which signal pairs constitute digit encodings.
- The difference between the standard 4-to-2 carry-save (c, s)-tree and the signed-digit (or borrow-save) tree, is only a question of placing inverters appropriately on input- and output-signals at the external boundary of the tree, but no differences internally in the tree.

All these trees can be implemented employing the same 4-to-2 carry-save adder as the fundamental building block.

References:

[EL97] M.D. Ercegovac and T. Lang. *Effective Coding for Fast Redundant Adders using the Radix-2 Digit Set* $\{0,1,2,3\}$. In *Proc. 31st Asilomar Conf. Signals Systems and Computers*, pages 1163–1167, 1997.

[Kor94] P. Kornerup. *Digit-Set Conversions: Generalizations and Applications*. *IEEE Transactions on Computers*, C-43(6):622–629, May 1994.

[KNE87] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi. *Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation*. In *Proc. 8th IEEE Symposium on Computer Arithmetic*, pages 80–86. IEEE Computer Society, 1987.

[OSYSSN95] N. Ohkubo, M. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, and Y. Nakagome. *A 4.4 ns CMOS 54 $\times$ 54-b Multiplier Using Pass Transistor Multiplexers*. *IEEE Journal of Solid State Circuits*, 30(3):251–257, 1995.

[PGK01] D.S. Phatak, T. Goff, and I. Koren. *Constant-Time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations*. *IEEE Transactions on Computers*, 50(11):1267–1278, 2001.