More Fast Adders
Ivor Page[1]

8.1 Carry Skip Adders

Consider a k bit adder, where k is a multiple of 4. Each set of 4 neighboring stages can be considered a block that can generate, propagate, or absorb a carry. Divide the adder into $k/4$ such blocks and, for each block, add a 4-input AND gate to produce a group propagate signal. A carry signal entering a certain block can be propagated past the block without waiting for the signal to propagate through the 4 individual stages of the block. If all $k/4$ blocks propagate, a carry entering the least significant stage will pass to the most significant carry-out in time $k/4$ times the delay through the carry-skip unit. If any block generates a carry, that carry will propagate through the remaining stages of the block, and then through the *carry-skip* gates to the final block, where it may have to propagate through 3 stages to reach the most significant adder. See Figure 1 for the logic.

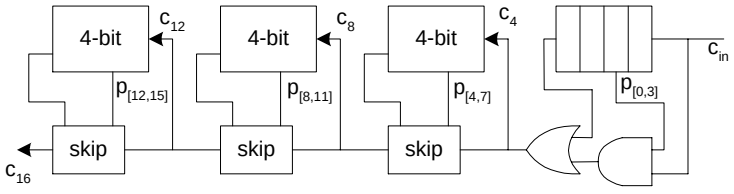

Figure 1: 16 bit Carry-Skip Adder

The longest delay path begins with a carry generated in stage 0 in the least significant block, propagates through 3 stages in that block, then through the OR gate, then through $k-2$ carry-skip units, and then through 3 of the 4 stages in the most significant block, to the c_{k-1} signal. We can generalize these results for a block size of b in a k bit adder as follows:

$$T_{fixed-skip-add} = (b-1)T_p + D + (k/b-2)T_s + (b-1)T_p$$

[1]University of Texas at Dallas

$$= \text{First block} \quad \text{OR gate} \quad \text{middle blocks} \quad \text{Last block}$$

T_p is the time to propagate a carry through one stage of the adder (from c_i to c_{i+1}), and T_s is the delay through one carry-skip stage

Recall that $T_p = 2D$ in the standard ripple-carry adder based on two half-adders. The delay $T_s = 2D$ since there is an AND gate and an OR gate in series in the carry-skip unit.

$$T_{fixed-skip-add} = 4Db + 2kD/b - 7D$$

The optimum block size, b_{opt}, is found by differentiating the right-hand side with respect to b and equating the result to zero.

$$\begin{aligned} \frac{dT_{fixed-skip-add}}{db} &= 4D - \frac{2kD}{b^2} = 0 \\ b^{opt} &= \sqrt{k/2} \end{aligned}$$

The corresponding adder delay is:

$$\begin{aligned} T^{opt}_{fixed-skip-add} &= 4Db^{opt} + 2kD/b^{opt} - 7D \\ &= 4D\sqrt{k} + 2D\sqrt{2k} - 7D \\ &= 6.8D\sqrt{k} - 7D \end{aligned}$$

For example, in a 32 bit adder, $b^{opt} = 4$ and the delay is approximately $25D$. Compare this value with the delay of a ripple-carry system, $64D$.

In a 64 bit adder, $b^{opt} = \sqrt{32} = 5.657$. If we use $b = 4$, the delay is $41D$. If $b = 8$ the delay is again $41D$. An in-between solution is possible with $b = 6$. Then there are 10 blocks of 6 and 1 block of 4 (at the most significant end). The corresponding delay is $35D$. The 64 bit ripple-carry adder has delay $128D$.

8.1.1 Variable Block Size

In the next development there are t carry-skip blocks with sizes $b_{t-1}, \cdots, b_1, b_0$ going from left to right. See Figure 2:

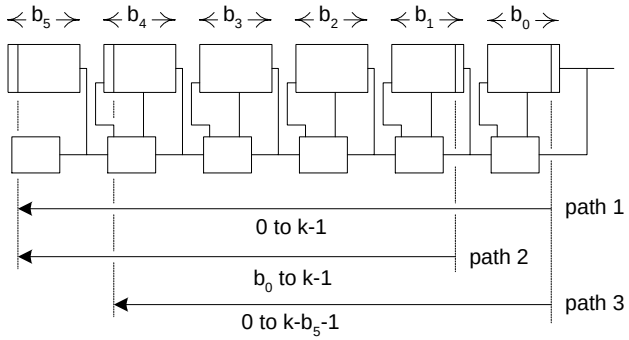

Figure 2: Variable Block Size Carry-Skip Adder

Consider the equation for the worst case delay from stage 0 to stage $k-1$, corresponding to path 1 in the diagram:

$$\begin{aligned} T_{var-carry-skip} = T_{path\ 1} &= (b_{t-1}-1)T_p \ + \ (t-2)T_s \ + \ D \ + \ (b_0-1)T_p \\ &= \text{last block} \quad \text{middle blocks} \quad \text{OR gate} \quad \text{first block} \end{aligned}$$

This equation is based on a carry being generated by stage 0 in block 0 propagating through the (b_0-1) remaining stages of block zero, then through $(t-2)$ carry skip units, then through $(b_{t-1}-1)$ stages of the left-most adder block.

Consider a carry being generated by stage b_0, the right-most stage of block 1, and following path 2 to the left-most stage of the adder. It's delay would be:

$$T_{path\ 2} = (b_{t-1}-1)T_p \ + \ (t-3)T_s \ + \ D \ + \ (b_1-1)T_p$$

If $T_s = T_p$ then b_1 can be 1 larger than b_0 without making this delay path worse than path 1. Similarly, if $b_2 = b_1 + 1$, the worst case delay from stage

$b_0 + b_1$ to stage $k - 1$ will be no larger than the delay for path 1. Blocks to the right of the center of the adder may therefore have sizes that form a simple incremental sequence. Now consider a carry being generated in stage 0 and used (absorbed) in the left-most stage of the penultimate block of the adder, stage $k - b_{t-1} - 1$. This corresponds to path 3 in the diagram. Its delay is:

$$T_{path\,3} \;=\; (b_{t-2} - 1)T_p \;+\; (t-3)T_s \;+\; D \;+\; (b_0 - 1)T_p$$

Compare this with the delay for path 1, the longest carry-propagation path. Again, if $T_p = T_s$, block size b_{t-2} can be one larger than block size b_{t-1} without making this delay path worse than path 1. Blocks to the left of the center of the adder may also have sizes that form a simple incremental sequence.

This analysis suggests an organ-pipe structure for the block sizes,

$$b, \quad b+1, \quad \cdots, \quad \frac{b+t}{2} - 1, \quad \frac{b+t}{2} - 1, \quad \cdots, \quad b+1, \quad b$$

We examine the effects of such a structure with an example. Consider a 28 bit adder with carry-skip block sizes 2,3,4,5,5,4,3,2. The following tables show the worst case delay paths for a carry generated in stage 0 and absorbed in stage i, and for a carry generated in stage j and absorbed in stage 27.

Delay from stage 0 to stage i							
$i =$	4	8	13	18	22	25	27
	$7D$	$11D$	$15D$	$17D$	$17D$	$17D$	$17D$

Delay from stage j to stage 27							
$j =$	0	2	5	9	14	19	23
	$17D$	$17D$	$17D$	$17D$	$15D$	$11D$	$7D$

The worst case delays are from carries generated in stage zero of the adder and absorbed anywhere in the left-hand half, and from carries generated anywhere in the right-hand half and absorbed in stage 27. These eight delays of $17D$ are made equal by making the block sizes vary in the organ-pipe fashion described above.

The total number of bits in the t blocks is then:

$$2[b + (b+1) + \cdots + (b + t/2 - 1)] = t(b + t/4 - 1/2)$$

which gives:

$$b = k/t - t/4 + 1/2$$

The worst-case delay through the adder with variable block sizes is then:

$$\begin{aligned} T_{var-skip-add} &= (b-1)T_p \;+\; (t-2)T_s \;+\; D \;+\; (b-1)T_p \\ &= \text{last block} \quad \text{middle blocks} \quad \text{OR gate} \quad \text{first block} \\ &= 4bD + 2tD - 7D \end{aligned}$$

The optimal number of blocks is calculated as follows:

$$\begin{aligned} \frac{dT_{var-skip-add}}{dt} &= -4kD/t^2 + D = 0 \\ t^{opt} &= 2\sqrt{k} \end{aligned}$$

$$T^{opt}_{var-skip-add} = 4D\sqrt{k} - 5D$$

which is approximately $\sqrt{2}$ smaller than with fixed block size. **Example:** Continuing with our 32 bit adder example of the previous section,

$$t^{opt} = 2\sqrt{32} = 11.3$$

If we choose $t = 10$, then $b = 32/t - t/4 + 1/2 = 1.2$. Say we choose $b = 1$. The block sizes are then 1,2,3,4,5,5,4,3,2,1, which only covers 30 bits. Its delay is $17D$.

The adder with block sizes 1,1,2,3,4,5,5,4,3,2,1,1 has delay $1 \times 2D + D + 10 \times 2D + 1 \times 2D = 25D$.

The adder with block sizes 2,2,3,4,5,5,4,3,2,2 has delay $1 \times 2D + D + 8 \times 2D + 1 \times 2D = 21D$.

The adder with block sizes 1,4,5,6,6,5,4,1 also has delay $21D$.

Notice here that the worst case delay corresponds to a delay path from the right-most stage of the right block of 6 to the left-most stage of the left block

of 6. As we discovered in the analysis, there is a balance between the largest (or smallest) block size and the number of blocks. Compare these results with $35D$ obtained with fixed block sizes and $128D$ obtained with a ripple-carry adder. Our analysis gives optimal delay of $17.6D$, but we were only able to achieve $21D$.

For a 64 bit adder, the best sequence of block sizes appears to be: 2,4,5,6,7,8,8,7,6,5,4,2 and it has delay $29D$ which is close to the optimum of $27D$ for this type of adder. Compare with $35D$ for the fixed block size adder and $128D$ for the ripple-carry adder.

Further developments are possible with the carry-skip idea if more than one level of skip units is employed. We shall not study these developments. The text has an example of a 30 bit adder with two levels of carry-skip units and a delay of $17D$ according to my calculations, which is no better than the single layer scheme for a 30 bit adder developed above.

8.2 Carry-Select Adder

The carry-select adder is based on a simple idea. For the left-most $k/2$ stages of the adder, we supply two $k/2$ bit adder units. In the first, we apply a carry-in of 0 and in the second we apply a carry-in of 1. The two sets of outputs are ready much more quickly than if the carry-out of the right-half of the adder had been used. Once that carry-out is available, it is used as a select signal to choose the correct set of outputs for the left-hand half of the adder. Figure 3 illustrates the idea:

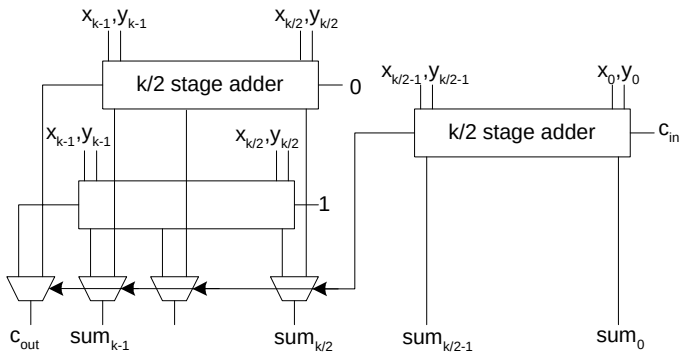

Figure 3: Simplest Carry-Select Adder

The delay through this adder, assuming ripple-carry units are used for the three $k/2$ bit adders, is $kD + T_m$, where T_m is the delay through the layer of 2-input multiplexers. This is approximately 1/2 the delay of a ripple carry adder, but it requires a large increase in the number of transistors. Note that some sharing of the logic between two adders in the left-half of the unit is possible. In position $j \geq k/2$ with inputs x_j and y_j, the signals $x_j \oplus y_j$ and $x_j.y_j$ are needed in both adders for that position.

The carry-select technique could be applied recursively to each of the three adder blocks in Figure 3. Just one recursive application results in the circuit of Figure 4.

Each of the three previous $k/2$ bit adders has been replaced by a carry-select adder using three $k/4$ bit adders. The result is wasteful since the most significant $k/4$ bit adders are unnecessarily repeated. We could use two adders in place of four. The result is shown in Figure 5.

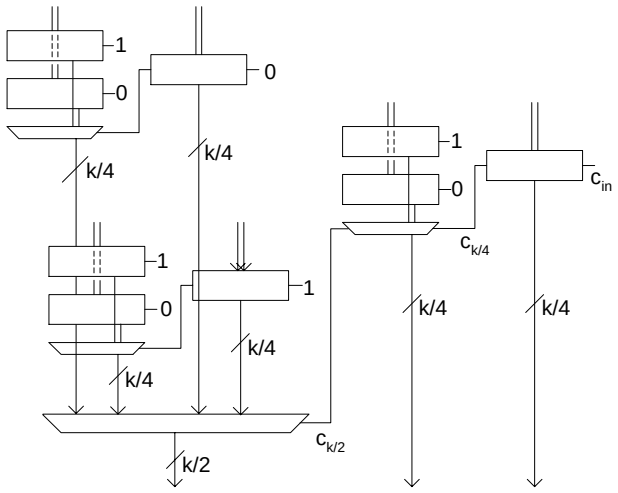

Figure 4: 2-level Wasteful Carry-Select Adder

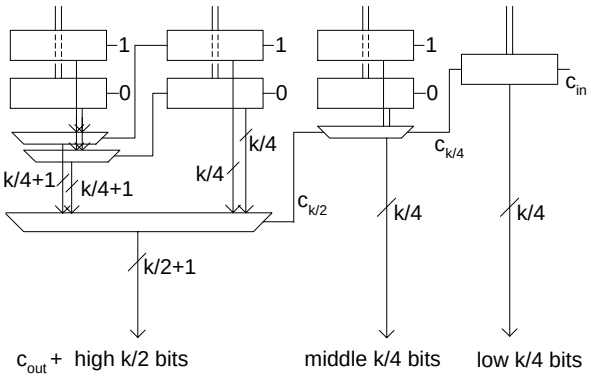

Figure 5: 2-level Carry-Select Adder

8.3 Conditional Sum Adder

If the recursive procedure is repeatedly applied to the units of the adder, the delay becomes logarithmic in the number of bits:

$$\begin{aligned} T(k/2) &= T(k)/2 + 2T_m \\ T(k/4) &= T(k)/4 + 3T_m \end{aligned}$$

$$\begin{aligned} T(k/2^i) &= T(k)/2^i + iT_m \\ &\ldots \\ T(1) &= T_{FA} + T_m log_2 k \end{aligned}$$

where $T(i)$ is the delay of the k bit conditional-sum adder using adder blocks of size i and T_{FA} is the delay of a single full adder.

The number of full-adders is $2k - i$ and the number of multiplexers is $(k - 1)(log_2 k + 1)$.

Note again that some sharing of the logic between the two adders with identical inputs is possible. The number of full-adders given above is therefore an overestimate of the circuit needs. The text suggests an equivalent of only k full adders.

A 32 bit fully recursive conditional-sum adder would have delay $6D + 5T_m$, which could realistically become $16D$ or so, and would require 32 full adders and 124 2-input multiplexers. We would probably implement the multiplexers using pass transistor logic. Inverters would be needed to avoid long chains of pass transistors.

A 64 bit fully recursive carry-select adder would have delay $6D + 6T_m$, which could realistically become $18D$ or so, and would require 64 full adders and 315 2-input multiplexers. The use of compound CMOS EX-OR gates could shave a few more gate delays from these estimates, although the assumption of $2D$ for the multiplexer delay could be an underestimate since at least one inverter per multiplexer would be present in the worst-case delay path.

These delay times are not much larger than for the best carry-prefix networks and CLA implementations, and the numbers of transistors are also comparable.

Assuming a 2-input multiplexer requires 8 transistors (including an inverter for its control input and another for its output), and each adder stage can be implemented in 32 transistors, a 64 bit conditional-sum adder would require approximately $32 \times 64 + 8 \times 315 = 3,608$ transistors. Recall that the CLA-4 based 64 bit adder had delay $15D$ and required 4076 transistors, while the Kogge-Stone carry-prefix network adder had delay $10D$ are required 6670 transistors.

The conditional-sum of 16 decimal digits is illustrated in the following table.

The sum is:

```
    2  6  7  7  4  1  0  0  2  6  9  2  4  3  5  8
+   5  6  0  4  9  7  9  4  1  5  1  7  1  6  4  5
```

At t_0 the sum in the righ-most column is finalized and in the other 15 columns, two sums are present. These are combined in pairs to form 2 sums for each group of two columns, then four columns, and so on. For the pair of columns with inputs 92+17, the two pairs of sums are 10/11 and 9/10. The logic combines these to form the two sums 109/110 for that group of two columns. These are then combined with sums 041/042 to the left to form sums 04209/04210 for 4 columns. The same process occurs at each level. At the same time, the resolved carry from the right hand end of the sum makes its way across the columns stradling 1,2,4,8, then 16 columns.

2 5	6 6	7 0	7 4	4 9	1 7	0 9	0 4	2 1	6 5	9 1	2 7	4 1	3 6	5 4	8 5	
$\frac{07}{08}$	$\frac{12}{13}$	$\frac{07}{08}$	$\frac{11}{12}$	$\frac{13}{14}$	$\frac{08}{09}$	$\frac{09}{10}$	$\frac{04}{05}$	$\frac{03}{04}$	$\frac{11}{12}$	$\frac{10}{11}$	$\frac{09}{10}$	$\frac{05}{06}$	$\frac{09}{10}$	$\frac{09}{10}$	13	t_0
$\frac{082}{083}$		$\frac{081}{082}$		$\frac{138}{139}$		$\frac{094}{095}$		$\frac{041}{042}$		$\frac{109}{110}$		$\frac{059}{060}$		103		t_1
$\frac{08281}{08282}$				$\frac{13894}{13895}$				$\frac{04209}{04210}$				06003				t_2
$\frac{082823894}{082823895}$								042096003								t_3
08282389442096003																t_4