TSTE18 Digital Arithmetic
Seminar 3

Oscar Gustafsson

Addition and subtraction

- Ripple-carry addition
- Solutions to slow ripple-carry addition
  - Redundant number systems
  - **Carry-acceleration**
- **Multi-operand addition**

Carry look-ahead addition

- The carry can either take on a value independent on the carry input or the exact value of the carry input
- These cases are

a_i	b_i	c_{i+1}	Case
0	0	0	No carry-propagation (kill)
0	1	c_i	Carry-propagation (propagate)
1	0	c_i	Carry-propagation (propagate)
1	1	1	Carry-generation (generate)

- We can define propagate, p_i, and generate, g_i at bit-level as

$$p_i = a_i \oplus b_i \text{ and } g_i = a_i b_i \tag{1}$$

Carry look-ahead addition

- The carry output can then be expressed as

$$c_{i+1} = g_i + p_i c_i \tag{2}$$

- For the next stage the expression becomes

$$\begin{aligned} c_{i+2} &= g_{i+1} + p_{i+1}c_{i+1} = g_{i+1} + p_{i+1}(g_i + p_i c_i) \\ &= g_{i+1} + p_{i+1}g_i + p_{i+1}p_i c_i \end{aligned} \tag{3}$$

- For $N+1$:th stage we have

$$\begin{aligned} c_{i+(N+1)} &= g_{i+N} + p_{i+N}g_{i+(N-1)} + p_{i+N}p_{i+N-1}g_{i+(N-2)} + \\ &\quad + \cdots + p_{i+(N-1)} \cdots p_{i+1}p_i c_i \end{aligned} \tag{4}$$

Carry look-ahead addition

- Hence, it is possible to compute all carries in parallel

$$
\begin{aligned}
c_{i+1} &= g_i + p_i c_i \\
c_{i+2} &= g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i \\
c_{i+3} &= g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i \\
c_{i+4} &= g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i + \\
&\quad p_{i+2} p_{i+1} p_i c_i \\
&\ldots
\end{aligned}
$$

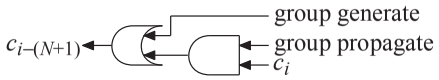
- This will lead to high fan-out for the signals p_j and g_j and high fan-in gates for higher order terms

Carry look-ahead addition

- These terms can be grouped as

$$
\begin{aligned}
c_{i+(N+1)} &= \underbrace{g_{i+N} + p_{i+N} g_{i+(N-1)} + p_{i+N} p_{i+N-1} g_{i+(N-2)} + \cdots +}_{G_{i+N:i}} \\
&\quad \underbrace{+ p_{i+N} p_{i+(N-1)} \cdots p_{i+1} p_i}_{P_{i+N:i}} c_i \\
&= G_{i+N:i} + P_{i+N:i} c_i
\end{aligned}
$$

  where $G_{i+N:i}$ is called group generate and $P_{i+N:i}$ group propagate as they correspond to the generate and propagate signals for the group of bits $i+N$ to i
- The group generate and propagate signals can be computed before the incoming carry is known to speed up the output carry generation



Prefix problem

- The prefix problem is that of computing the results

$$b_k = a_1 \circ a_2 \circ \ldots a_k, \ 1 \leq k \leq n \tag{5}$$

  for all k where $\circ$ is an associative operator
- As can be seen, each result b_k depends on all previous inputs $a_i, \ i \leq k$ as for the carry problem
- This also has applications in e.g. sorting

Parallel prefix addition

- The carry signal in position k can be written as

$$c_{k+1} = G_{k:l} + P_{k:l} c_l \tag{6}$$

- Similarly, the sum signal in position k for an adder using N bits is then expressed as

$$s_k = a_k \oplus b_k \oplus c_k = p_k \oplus (G_{(k-1):0} + P_{(k-1):0} c_{\text{in}}) = p_k \oplus c_k \tag{7}$$

- Of interest to compute all group generate and group propagate originating from the LSB position, i.e., $G_{k:0}$ and $P_{k:0}$ for $1 \leq k \leq W \Rightarrow$ parallel prefix.

Parallel prefix addition

- For parallel prefix adders, we introduce the dot operator $\bullet$ as

$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \begin{bmatrix} g_j \\ p_j \end{bmatrix} \triangleq \begin{bmatrix} g_i + p_i g_j \\ p_i p_j \end{bmatrix} \tag{8}$$

- Adders based on dot operators are commonly referred to as parallel prefix adders
- The group generate and propagate for bits k to l can be written as

$$\begin{bmatrix} G_{k:l} \\ P_{k:l} \end{bmatrix} \triangleq \begin{bmatrix} g_k \\ p_k \end{bmatrix} \bullet \begin{bmatrix} g_{k+1} \\ p_{k+1} \end{bmatrix} \bullet \cdots \bullet \begin{bmatrix} g_l \\ p_l \end{bmatrix} \tag{9}$$

Parallel prefix addition

- The dot-operator is associative

$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \left(\begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \begin{bmatrix} g_j \\ p_j \end{bmatrix} \right) \bullet \begin{bmatrix} g_l \\ p_l \end{bmatrix} = \begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \left(\begin{bmatrix} g_j \\ p_j \end{bmatrix} \bullet \begin{bmatrix} g_l \\ p_l \end{bmatrix} \right)$$

- Idempotent

$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \begin{bmatrix} g_k \\ p_k \end{bmatrix} \bullet \begin{bmatrix} g_k \\ p_k \end{bmatrix}$$

- But not commutative

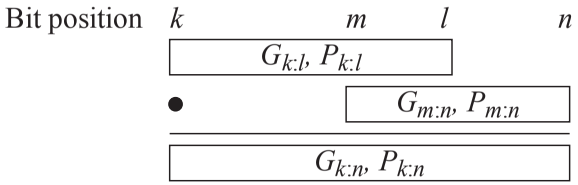
$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \begin{bmatrix} g_j \\ p_j \end{bmatrix} \neq \begin{bmatrix} g_j \\ p_j \end{bmatrix} \bullet \begin{bmatrix} g_i \\ p_i \end{bmatrix}$$

Parallel prefix addition

- The idempotency leads to the very useful property that overlapping groups does not cause any problems

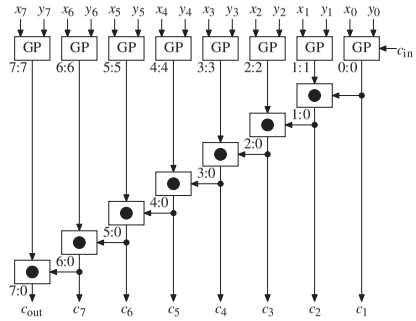
$$\begin{bmatrix} G_{k:n} \\ P_{k:n} \end{bmatrix} = \begin{bmatrix} G_{k:l} \\ P_{k:l} \end{bmatrix} \bullet \begin{bmatrix} G_{m:n} \\ P_{m:n} \end{bmatrix}, \; k \geq l, m \geq n, m \geq l-1. \tag{10}$$

which can be illustrated as below



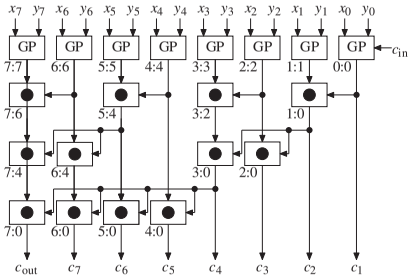
Parallel prefix addition

- Of interest to compute all group generate and group propagate originating from the LSB position, i.e., $G_{k:0}$ and $P_{k:0}$ for $1 \leq k \leq W$
- Can be computed sequentially



Parallel prefix addition

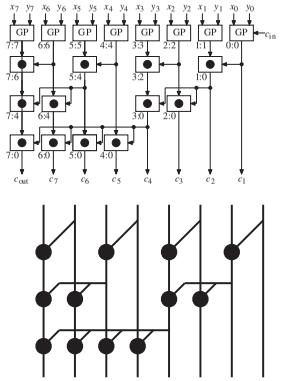
- Can also use the properties of the dot product and compute it in parallel



- Many different structures for computing the dot product trees have been proposed trading dot operators, fan-out, wire length, etc.

Parallel prefix addition

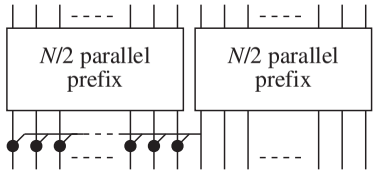
- Often a simplified view is used to represent the dot-products



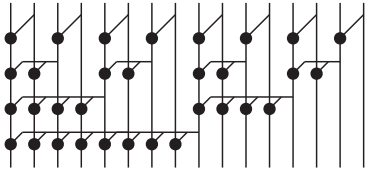
- This is the so called Ladner-Fischer parallel prefix structure

Parallel prefix addition

- The Ladner-Fischer approach is optimal in the number of stages and is composed as



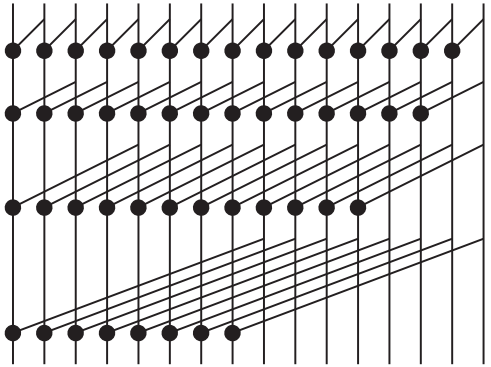
- For size 16 this gives



- However, the worst fan-out is $N/2$ leading to problems in VLSI implementations

Parallel prefix addition

- An alternative depth-optimal structure was proposed by Kogge and Stone as

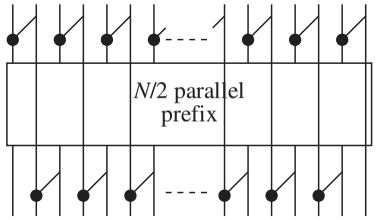


for size 16

- Here, the fan-out is decreased to 4 (2 with buffers) at the cost of more dot operators (49 compared to 32 for Ladner-Fischer)

Parallel prefix addition

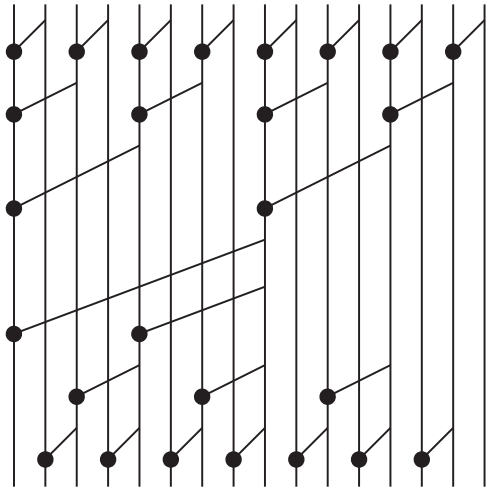
- Another useful structure is the Brent-Kung approach as



- For this, the depth is increased but the fan-out and operator count is low

Parallel prefix addition

- A 16-input Brent-Kung parallel prefix network



Parallel prefix addition

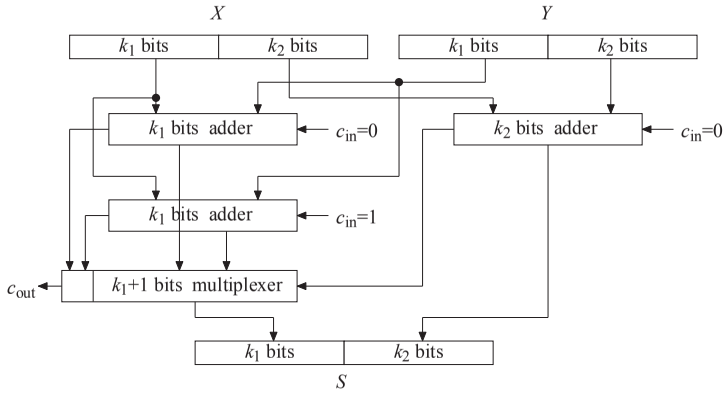
- Clearly, there are other trade-off available in terms of operators, depth, and maximum fan-out
- For those considered here with 16 inputs

Structure	Depth	Operators	Fan-out
Serial	15	15	1
Ladner-Fischer	4	32	8
Kogge-Stone	4	49	4 (2 with buffers)
Brent-Kung	6	26	3 (1 with buffers)

- Many more available with different trade-offs between these factors

Carry-select and conditional sum addition

- At any stage of the adder, the carry signal is either zero or one(!)
- Precompute both results and select the correct result once the correct carry arrives
- Carry-select adder



Carry-select and conditional sum addition

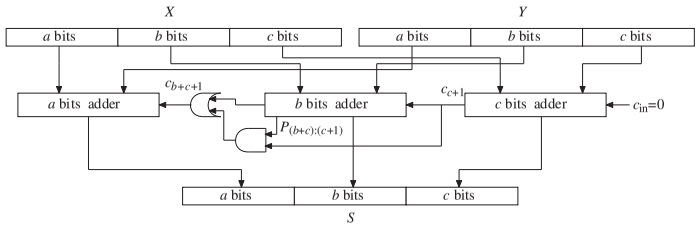
- The carry-select adder can of course be separate into several stages
- Breaking a word of length N into N/w stages of size w where $w \approx \sqrt{N}$ gives a good initial estimate of a suitable partitioning with delay $\propto \sqrt{N}$
- Best selection depends on the gate delays of the adders and the multiplexers
- Typically an increasing length provides slightly better results since the later adders have more time to finish because of the additional multiplexer delays for the incoming carry
- High fan-out for the carry signal controlling the multiplexers

Carry-select and conditional sum addition

- Each precomputation adder can in turn be split into a carry-select adder
- If this is done until the wordlength of each adder is one a conditional sum adder is obtained
- For each bit segment there are only two alternatives, $c_{in} = 0$ or $c_{in} = 1$
- Hence, the number of adders does not increase
- Can be seen as restructuring the multiplexer tree
- Example: derive an eight-bit conditional sum adder iteratively

Carry-skip addition

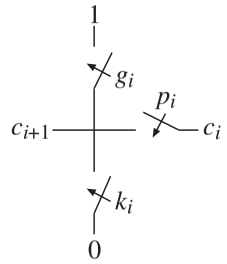
- Utilize the carry propagation information to skip parts of the carry-chain



- Breaking a word of length N into N/w parts of size w where $w \approx \sqrt{N}$ gives a good initial estimate of a suitable partitioning with delay $\propto \sqrt{N}$
- Note that $p_i = x_i \oplus y_i$ is often computed in the full adder anyway
- High-fan in for and-gate computing $P_{(b+c):c}$ (or $c_{c+1}P_{(b+c):c}$)

Manchester carry-chains

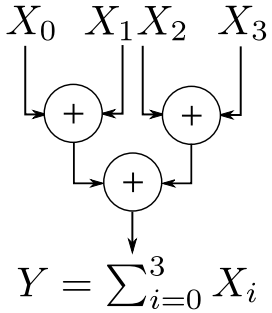
- Utilizing the ideas of the generate, propagate, and kill signals earlier defined as simple network of switches can be used for carry propagation as



- This is referred to as a Manchester carry-chain
- Each switch can be replaced with a transistor (plus buffers or inverters at appropriate places) to obtain simple and fast carry-propagation
- This is an efficient way to realize short ripple-carry adders in e.g. carry-select and carry-skip adders

Multi-operand addition

- When adding more than two values, one can of course simply connect carry-propagation adders



- However, for high-speed realization it is not efficient to have multiple accelerated carry-propagation adders

Multi-operand addition

- Instead, carry-save adders are normally used and an accelerated carry-propagation adder is used to accumulate the sum and carry vectors at the end