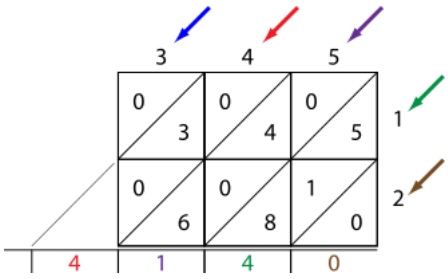
Wallace tree

A **Wallace tree** is an efficient hardware implementation of a digital circuit that multiplies two integers, devised by Australian Computer Scientist Chris Wallace in 1964.[1]



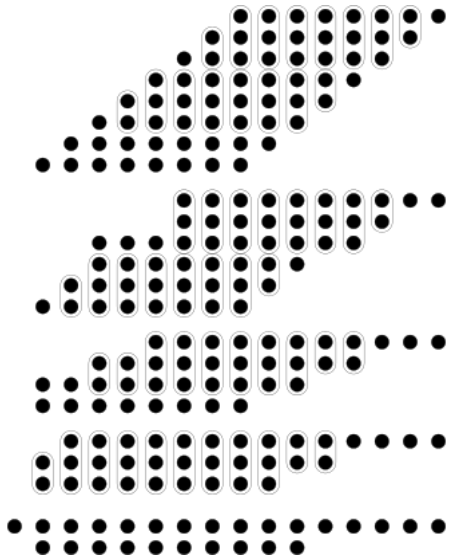
basic principle known from manual multiplication

The Wallace tree has three steps:

1. Multiply (that is – AND) each bit of one of the arguments, by each bit of the other, yielding n^2 results. Depending on position of the multiplied bits, the wires carry different weights, for example wire of bit carrying result of $a_4 b_3$ is 128 (see explanation of weights below).
2. Reduce the number of partial products to two by layers of full and half adders.
3. Group the wires in two numbers, and add them with a conventional adder.[2]

The second step works as follows. As long as there are three or more wires with the same weight add a following layer:-

- Take any three wires with the same weights and input them into a full adder. The result will be an output wire of the same weight and an output wire with a higher weight for each three input wires.
- If there are two wires of the same weight left, input them into a half adder.
- If there is just one wire left, connect it to the next layer.


Example of reduction on an 8x8 multiplier.

The benefit of the Wallace tree is that there are only $O(\log n)$ reduction layers, and each layer has $O(1)$ propagation delay. As making the partial products is $O(1)$ and the final addition is $O(\log n)$, the multiplication is only $O(\log n)$, not much slower than addition (however, much more expensive in the gate count). Naively adding partial products with regular adders would require $O(\log^2 n)$ time. From a complexity theoretic perspective, the Wallace tree algorithm puts multiplication in the class NC^1.

These computations only consider gate delays and don't deal with wire delays, which can also be very substantial.

The Wallace tree can be also represented by a tree of 3/2 or 4/2 adders.

It is sometimes combined with Booth encoding.[3][4]

Contents

Weights explained

Example

See also

References

Further reading

External links

Weights explained

The weight of a wire is the radix (to base 2) of the digit that the wire carries. In general, $a_n b_m$ – have indexes of n and m; and since $2^n 2^m = 2^{n+m}$ the weight of $a_n b_m$ is 2^{n+m}.

Example

$n = 4$, multiplying $a_3 a_2 a_1 a_0$ by $b_3 b_2 b_1 b_0$:

1. First we multiply every bit by every bit:
   - weight 1 – $a_0 b_0$
   - weight 2 – $a_0 b_1$, $a_1 b_0$
   - weight 4 – $a_0 b_2$, $a_1 b_1$, $a_2 b_0$
   - weight 8 – $a_0 b_3$, $a_1 b_2$, $a_2 b_1$, $a_3 b_0$
   - weight 16 – $a_1 b_3$, $a_2 b_2$, $a_3 b_1$
   - weight 32 – $a_2 b_3$, $a_3 b_2$
   - weight 64 – $a_3 b_3$
2. Reduction layer 1:
   - Pass the only weight-1 wire through, output: 1 weight-1 wire
   - Add a half adder for weight 2, outputs: 1 weight-2 wire, 1 weight-4 wire
   - Add a full adder for weight 4, outputs: 1 weight-4 wire, 1 weight-8 wire
   - Add a full adder for weight 8, and pass the remaining wire through, outputs: 2 weight-8 wires, 1 weight-16 wire
   - Add a full adder for weight 16, outputs: 1 weight-16 wire, 1 weight-32 wire
   - Add a half adder for weight 32, outputs: 1 weight-32 wire, 1 weight-64 wire
   - Pass the only weight-64 wire through, output: 1 weight-64 wire
3. Wires at the output of reduction layer 1:
   - weight 1 – 1
   - weight 2 – 1
   - weight 4 – 2
   - weight 8 – 3
   - weight 16 – 2
   - weight 32 – 2
   - weight 64 – 2
4. Reduction layer 2:
   - Add a full adder for weight 8, and half adders for weights 4, 16, 32, 64
5. Outputs:
   - weight 1 – 1
   - weight 2 – 1
   - weight 4 – 1
   - weight 8 – 2
   - weight 16 – 2
   - weight 32 – 2
   - weight 64 – 2
   - weight 128 – 1

6. Group the wires into a pair of integers and an adder to add them.

See also

- Dadda tree

References

1. Wallace, Christopher Stewart (February 1964). "A suggestion for a fast multiplier". *IEEE Transactions on Electronic Computers*. EC-13 (1): 14–17.
2. Bohsali, Mounir; Doan, Michael (2010). "Rectangular Styled Wallace Tree Multipliers" (https://web.archive.org/web/20100215152142/http://www.veech.com/index_files/Wallace%20Tree.pdf) (PDF). Archived from the original (http://www.veech.com/index_files/Wallace%20Tree.pdf) (PDF) on 2010-02-15.
3. "Introduction" (http://www.eecs.tufts.edu/~ryun01/vlsi/design.htm). *8x8 Booth Encoded Wallace-tree multiplier*. Tufts university. 2007. Archived (https://web.archive.org/web/20100617044555/http://www.eecs.tufts.edu/~ryun01/vlsi/design.htm) from the original on 2010-06-17.
4. Weems Jr., Charles C. (2001) [1995]. "CmpSci 535 Discussion 7: Number Representations" (https://web.archive.org/web/20110206142109/http://www.cs.umass.edu/~weems/CmpSci535/Discussion7.html). Amherst: University of Massachusetts. Archived from the original (http://www.cs.umass.edu/~weems/CmpSci535/Discussion7.html) on 2011-02-06.

Further reading

- Savard, John J. G. (2018) [2006]. "Advanced Arithmetic Techniques" (http://www.quadibloc.com/comp/cp0202.htm). *quadibloc*. Archived (https://web.archive.org/web/20180703001722/http://www.quadibloc.com/comp/cp0202.htm) from the original on 2018-07-03. Retrieved 2018-07-16.

External links

- Generic VHDL Implementation of Wallace Tree Multiplier (https://web.archive.org/web/20120306195308/http://www.openhdl.com/vhdl/655-vhdl-component-wallace-tree-multiplier-generic.html).
