THE WALLACE TREE SIMULATOR

John D. Carpinelli, Michael Dokachev
New Jersey Institute of Technology

ABSTRACT

Wallace Trees are combinatorial logic circuits used to multiply binary integers. Constructed using carry-save adders, they are a fast, efficient method to implement multiplication. Since these adders do not propagate carry values between bits, they are faster than parallel adders and can produce multiplication products faster than other multiplication hardware.

This paper presents the Wallace Tree Simulator, an instructional aid for students studying computer architecture and CPU design, typically at the junior or senior level. It simulates 4-, 6-, and 8-bit Wallace Tree multipliers, as presented in the textbook *Computer Systems Organization and Architecture*. Students select the size of the Wallace Tree to be simulated and enter values for the operands to be multiplied. The simulator shows the partial products generated by the Wallace Tree, and the results generated by each carry-save adder in the tree, as well as the final product. Students can also examine the internal organization of the carry-save adders to see how they generate their results within the tree.

The Wallace Tree Simulator is coded as a platform-independent Java applet that can be executed within any Java-enabled web browser. The simulator and its source code are freely available under the terms of the GNU Public License.

INTRODUCTION

In order to perform useful work, a CPU must be able to perform arithmetic and logical operations. Although copying data from one location to another is the most frequently performed operation within a processor, a CPU that cannot modify its data would not be useful for practical applications. All processors include circuitry to perform some arithmetic and logical operations, and this topic is typically covered in computer organization and architecture courses.

Integer multiplication can be performed using any of several methods. The traditional shift-add approach and ROM lookup tables are two methods used to implement multiplication, but each has its drawbacks. The time needed to calculate products using the shift-add method increases linearly as the number of bits in the operands increases, and the size of the lookup ROM increases exponentially with increases in the size of the operands. Wallace Trees,[1] which use fast carry-save adders to calculate partial results, resolve much of these problems without unduly increasing hardware requirements.

The Wallace Tree Simulator allows students to view the internal organization of 4-, 6-, and 8-bit Wallace Tree multipliers. The user enters binary values to be multiplied and the simulator shows the results generated at each point in the tree. Users can see how partial products are generated, and the internal structure and computations of each carry-save adder.

This simulator was developed to work with designs presented in the textbook *Computer Systems Organization and Architecture*.[2] It is written in Java and is executed as an applet within any Java-enabled web browser. The primary reason for doing this was to provide platform independence. The Wallace Tree Simulator can be executed on any Java-enabled web browser, regardless of the type of computer used. By excluding proprietary

extensions in the source code, the simulator realizes the "write once run anywhere" mantra of Java developers. This is the most recent in a series of simulators developed for computer architecture education. Past simulators focus on CPUs,[3,4] computer systems,[5] and programmable logic devices.[5] The source and executable code for the Wallace Tree Simulator, as well as the other simulators developed by this research group, are freely available online[6,7] under the terms of the GNU Public License.

The rest of this paper is organized as follows. The next section gives some background on Wallace Trees and carry-save adders. The functions of the simulator are given in the following section; finally, concluding remarks are presented.

WALLACE TREES

A Wallace Tree is a combinatorial circuit used to multiply two binary numbers. Although it requires more hardware than shift-add multipliers, it produces a product in far less time. Instead of performing additions using standard parallel adders, Wallace Trees use carry-save adders to calculate intermediate results and one parallel adder to calculate the final product.

A carry-save adder can add three values simultaneously, instead of just two. However, it does not output a single result. Instead, it outputs both a sum and a set of carry bits. The carry-save adder is essentially a group of full adders, each of which adds the bits in the same position of its three input operands. The full adder that adds bit *i* of its operands outputs sum bit S_i and carry bit C_{i+1}. Because the carry bits do not propagate through the adder, it is faster than parallel adders. Unlike the parallel adder, though, it does not produce a final sum of its inputs.

To use a carry-save adder to perform multiplication, we first calculate the partial products of the multiplication, and then input them to the carry-save adder. For example, consider the multiplication of binary values 011010 and 110101. It generates partial products as shown below.

```
x  =        011010
y  =        110101
            011010    ← PP0
           000000     ← PP1
          011010      ← PP2
         000000       ← PP3
        011010        ← PP4
       011010_____    ← PP5
       10101100010    ←final sum calculated
```

The partial products can be calculated by several methods, one of which uses the bits of y to select either the value x or 0, shifting the result to the left to properly align the partial products. Figure 1 shows one method using multiplexers that can be used to calculate the partial products for this example.

The partial products are input to two carry-save adders at the top of the Wallace Tree, each of which generates sum and carry outputs. These outputs are combined using additional carry-save adders until only two outputs are left. These values are added using a parallel adder to produce the final product. The 6-bit Wallace Tree multiplier, with partial and final results for this example, is shown in Figure 2.

SIMULATOR OPERATIONS

To use the simulator, the user first starts the simulator from within any Java-enabled web browser. After selecting the size of the Wallace Tree, 4-bit, 6-bit, or 8-bit, the simulator displays the Wallace Tree and data entry fields, as shown in Figure 3. The user then enters binary values for the multiplier and multiplicand and clicks the Calculate button. The simulator checks for valid binary operands and then calculates and displays all

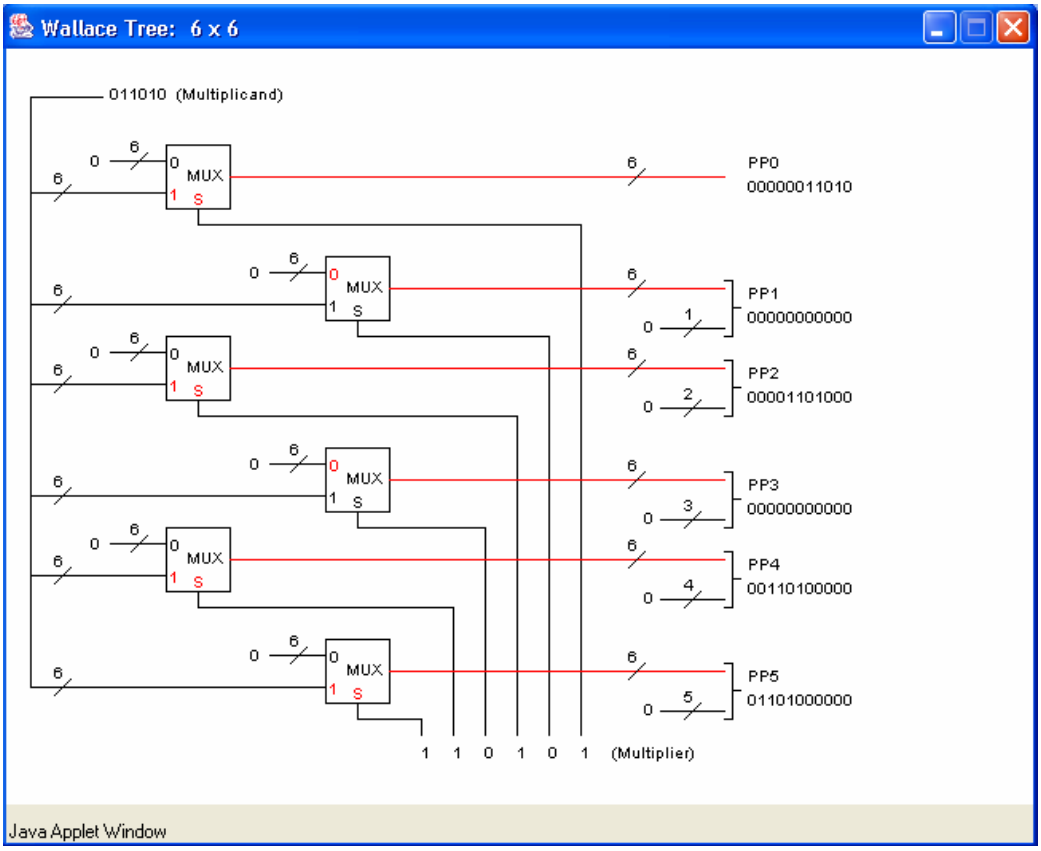

Figure 1: Partial product generation using multiplexers

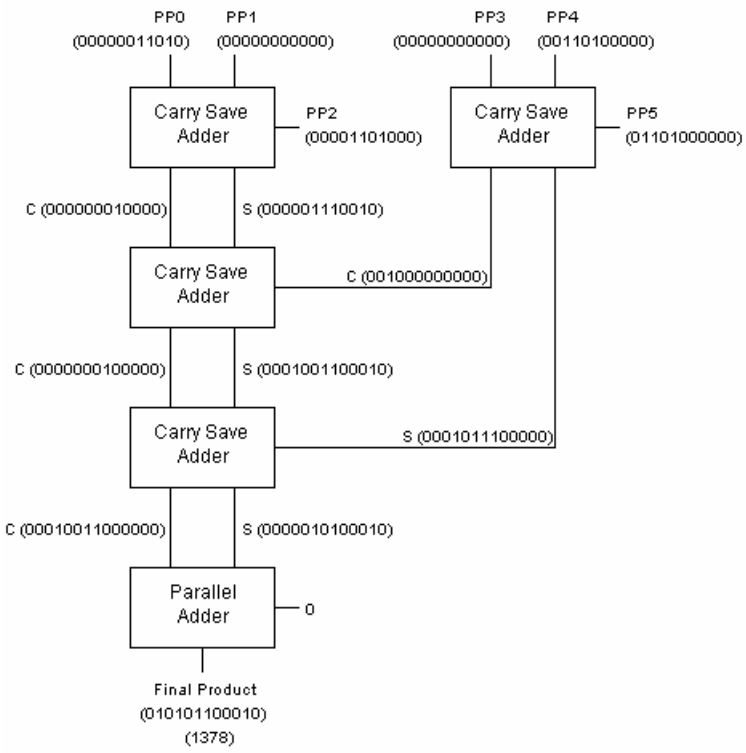

Figure 2: 6-bit Wallace Tree

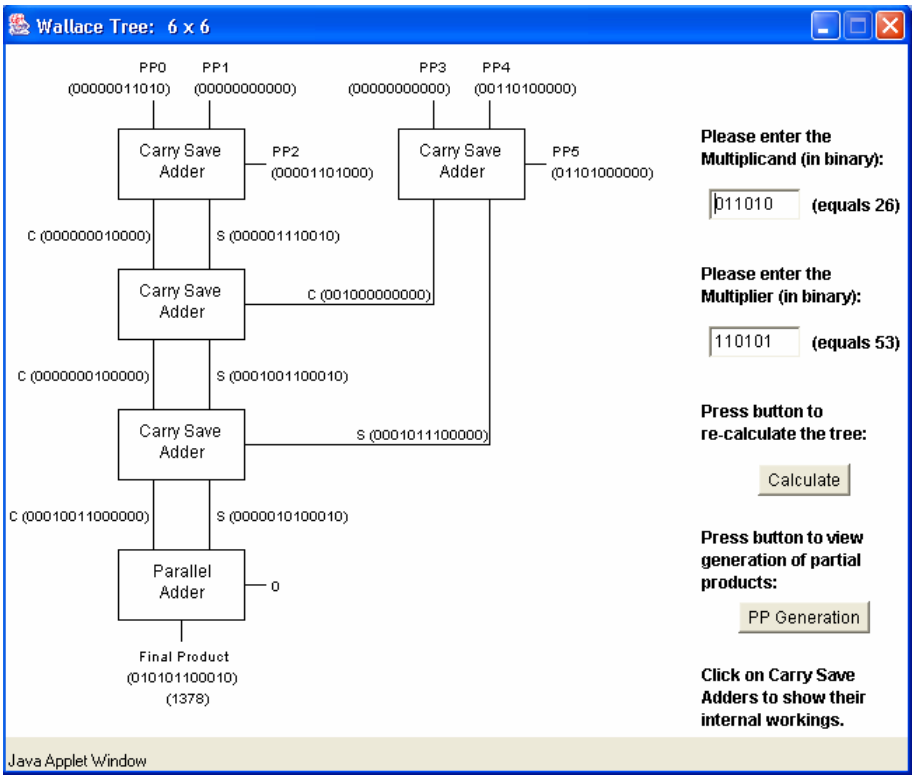

Figure 3: Wallace Tree and data entry window

partial products and intermediate and final results, as shown in the figure.

The user may click the *PP Generation* button to open a new window displaying the hardware used to generate the partial products, along with the values for the present computation. This window was shown previously in Figure 1 for this example.

One additional function supplied by this simulator is the visual display of the internal organization and function of the carry-save adders. The user may click on any carry-save adder in the Wallace Tree to open a new window showing its hardware, basically a sequence of full adders, and its data input and output values for the present computation. The simulator does not display the internal organization of the parallel adder at the bottom of the tree. Figure 4 shows the top-right carry-save adder for this example.

SUMMARY

The Wallace Tree Simulator simulates the internal functions of 4-, 6-, and 8-bit Wallace tree multipliers. By displaying partial product generation and intermediate results, as well as the internal functions of the carry-save adders, the simulator provides students with a more intuitive understanding of how Wallace Trees work. It serves as a useful adjunct for students using the textbook *Computer Systems Organization and Architecture*. Both the executable and source code for this simulator are available at the book's companion web sites.[6,7] The source code is available without cost under the terms of the GNU Public License.

ACKNOWLEDGMENTS

Partial funding for this project was provided by the National Science Foundation through the Gateway Engineering Education Coalition.

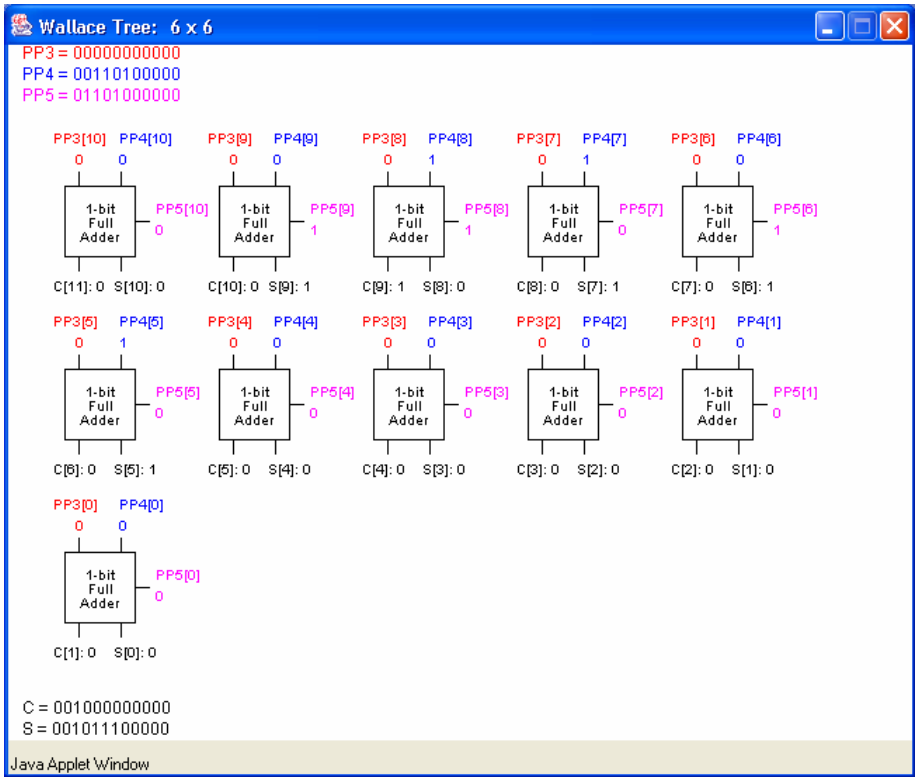

Figure 4: Carry-save adder detail

BIBLIOGRAPHY

1. **Wallace, C. S. "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computing, vol. EC-13, no. 2, Feb. 1964, pp. 14-17.**

2. **Carpinelli, John D. Computer Systems Organization and Architecture. Boston, MA: Addison-Wesley (2001).**

3. **Carpinelli, John D. "The Relatively Simple CPU Simulator," ASEE Computers in Education Journal, April-June 2002, pp. 20-26.**

4. **Carpinelli, John D. "The Very Simple CPU Simulator," Proceedings of the IEEE/ASEE 2002 Frontiers in Education Conference, Boston, MA, November 2002.**

5. **Carpinelli, John D. and Fabio Jaramillo, "Simulation Tools for Digital Design and Computer Organization and Architecture," Proceedings of the 2001 Frontiers in Education Conference, Reno, NV, October, 2001, pp. S3C.1-5.**

6. **URL: www.awl.com/carpinelli; Companion web site for Computer Systems Organization and Architecture**

7. **URL:www.awl.com/info/carpinelli; Companion web site for Computer Systems Organization and Architecture**

BIOGRAPHICAL INFORMATION

John D. Carpinelli is an associate professor of Electrical and Computer Engineering, and Computer and Information Sciences, at New Jersey Institute of Technology. His research interests include interconnection networks, computer architecture, parallel processing, distance learning, and computer simulation. He is the author of the textbook *Computer Systems Organization and Architecture* (Addison-Wesley, 2001).

Michael Dokachev received the B.S. in Computer Engineering from the New Jersey Institute of Technology in 2003. He is currently working for the U.S. Army at Picatinny Arsenal, NJ.