Booth multiplier algorithm specification and FPGA implementation

Luis M. Gallegos C.

September 2014

Abstract

This work presents an FPGA implementation of a Booth multiplier, tested in an Atlys Spartan-6 FPGA Trainer Board. The work was originally done for an embedded systems class project.

1 Brief introduction

The algorithm was developed by Andrew D. Booth and published for the first time in an article in *The Quarterly Journal of Mechanics and Applied Mathematics* in 1951 [1]. The algorithm was developed in a time where calculators were faster doing shift operations than sums.

2 Algorithm description

The main idea behind the algorithm was to reduce the number of sums, considering a sequence of 1s can be represented as a subtraction, as show in (1).

$$1111 = 10000 - 1 \tag{1}$$

Considering this, a multiplication by a sequence of n 1s can be performed with a subtraction, instead of a series of n sums, one for each 1 in the sequence. See (2)

$$a \times 1111 = a \times (10000 - 1) = a \times 10000 - a \tag{2}$$

To illustrate the algorithm, we consider a and x the multiplicand and multiplier, respectively, and p the result, all represented in two's complement. Both a and x are $n = 4$-bit long, with p being 8-bit, all including the sign bit.

The algorithm goes through each bit in the multiplier, x_i, from LSB to MSB, analyzing each bit and its adjacent x_{i-1}, where i goes from 0 to $n-1$. For the case of x_0, $x_{i-1} = 0$. Partial sums are performed and their results stored in p, where the final result is also stored in the end. In each iteration, one of four operations is performed, depending on the pair of bits being analyzed:

a) If $x_i = 0$, $x_{i-1} = 0$
   - Perform an arithmetic right shift on p.

b) If $x_i = 1$, $x_{i-1} = 1$
   - Perform an arithmetic right shift on p.

c) If $x_i = 0$, $x_{i-1} = 1$
   - Add a to p, in its most significant half.
   - Perform an arithmetic right shift on p.

d) If $x_i = 1$, $x_{i-1} = 0$
   - Subtract a from p, in its most significant half.
   - Perform an arithmetic right shift on p.

To make operations easier, the following values are defined.

Table 1: Additional values for the procedure.

Values	Description
$\bar{a}$	a's two's complement
A	$\{a, 0000\}$ (a concatenated with n zeros)
$\bar{A}$	$\{\bar{a}, 0000\}$ ($\bar{a}$ concatenated with n zeros)

The reason for A is to be able to add a to p in its most significant half. Likewise, the reason for $\bar{A}$ is to subtract a from p in its most significant half, through a sum with the two's complement of a (opposite sign).

Algorithm 1 shows the algorithm for the implementation.

Algorithm 1 Booth implementation algorithm

```
$\bar{a} \leftarrow -a$
$A \leftarrow \{a, 0000\}$
$\bar{A} \leftarrow \{-a, 0000\}$
$p \leftarrow 0000\ 0000$
$i \leftarrow 0$
**for** $i = 0$ **to** $n - 1$ **do**
    **switch** $x_i$, $x_{i-1}$ **do**                ▷ Analyze $x_i$ and $x_{i-1}$
        **case** $0, 0$
            $p \leftarrow p >>> 1$
        **end case**
        **case** $1, 1$
            $p \leftarrow p >>> 1$
        **end case**
        **case** $0, 1$
            $p \leftarrow p + A$                      ▷ Ignore overflow
            $p \leftarrow p >>> 1$
        **end case**
        **case** $1, 0$
            $p \leftarrow p + \bar{A}$                ▷ Ignore overflow
            $p \leftarrow p >>> 1$
        **end case**
    **end switch**
**end for**
```

3 Example

For this example, we have the following values for multiplicand a and multiplier x.

Variable	Binary value	Decimal value
a	1011	-5
x	1010	-6

From a and x, we get:

$$\bar{a} = 0101$$
$$A = 10110000$$
$$\bar{A} = 01010000$$

Additionally, p is initialized, $p = 0000\ 0000$.
The bits from x can now be analyzed. The following is a step by step execution of the algorithm.

$$\mathbf{x_0 = 0},\ \mathbf{x_{-1} = 0}$$

$$p = p >>> 1 = \quad 0000\ 0000$$

$$\mathbf{x_1 = 1},\ \mathbf{x_0 = 0}$$

$$\begin{array}{rrl} & 0000\ 0000 & p \\ & \underline{+0101\ 0000} & \bar{A} \\ p = p + \bar{A} = & 0101\ 0000 & \\ & & \\ p = p >>> 1 = & 0010\ 1000 & \end{array}$$

$$\mathbf{x_2 = 0},\ \mathbf{x_1 = 1}$$

$$p = p + A$$

$$\begin{array}{rrl} & 0010\ 1000 & p \\ & \underline{+1011\ 0000} & A \\ p = p + A = & 1101\ 1000 & \\ & & \\ p = p >>> 1 = & 1110\ 1100 & \end{array}$$

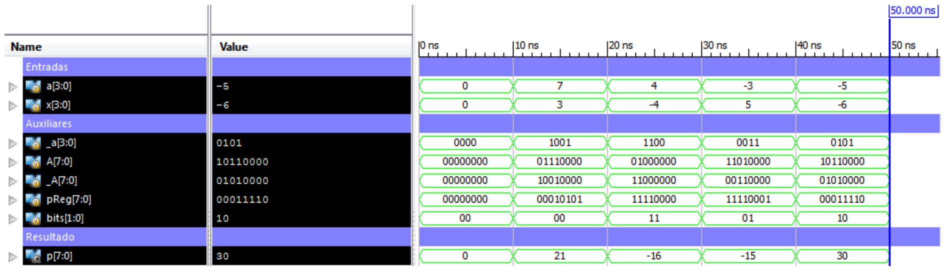
$$\mathbf{x_3 = 1},\ \mathbf{x_2 = 0}$$

$$p = p + \bar{A}$$

$$\begin{array}{rrl} & 1110\ 1100 & p \\ & \underline{+0101\ 0000} & \bar{A} \\ p = p + \bar{A} = & (1)0011\ 1100 & \\ & \text{ignore overflow (1)} & \\ & & \\ p = p >>> 1 = & 0001\ 1110 & \end{array}$$

At the end of the operations, we have a final value of $p = 0001\ 1110$, which is 30 in decimal notation, the correct answer for the multiplication $-5 \times -6 = 30$.

Figure 1 shows the result of this same operation in the ISE simulator.



Name	Value	0 ns	10 ns	20 ns	30 ns	40 ns
Entradas						
a[3:0]	-5	0	7	4	-3	-5
x[3:0]	-6	0	3	-4	5	-6
Auxiliares						
_a[3:0]	0101	0000	1001	1100	0011	0101
A[7:0]	10110000	00000000	01110000	01000000	11010000	10110000
_A[7:0]	01010000	00000000	10010000	11000000	00110000	01010000
pReg[7:0]	00011110	00000000	00010101	11110000	11110001	00011110
bits[1:0]	10	00	00	11	01	10
Resultado						
p[7:0]	30	0	21	-16	-15	30

Figure 1: Booth multiplier simulation.

4 Implementation

The algorithm was implemented in an Atlys Spartan-6 FPGA Trainer Board [2]. The 4-bit multiplier x is input using switches SW7-SW4, where SW4 is the LSB. The 4-bit multiplicand a is input using switches SW3-SW0, where SW0 is the LSB. The 8-bit product p is displayed with the LEDs LD7-LD0, where LD0 is the LSB. These specifications can be seen in the *ucf* implementation file.

A short video showing the multiplier functioning on the Atlys board can be seen in the link `http://youtu.be/RQSdaQeM46E` [3].

References

[1] A. D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[2] Digilent. (2016) Atlys Spartan-6 FPGA Trainer Board - Digilent. [Online]. Available: http://store.digilentinc.com/atlys-spartan-6-fpga-trainer-board-limited-time-see-nexys-video/

[3] L. Gallegos. (2014) Booth multiplier, 4 bits x 4 bits = 8 bits - YouTube. [Online]. Available: https://www.youtube.com/watch?v=RQSdaQeM46E