ECSE 548 Project: 8-bit Booth Multiplier Design specifications

Marco Kassis, Aryan Mojtahedi, Dimitrios Stamoulis and Louis-Charles Trudeau
Department of Electrical and Computer Engineering
McGill University, Montreal, Canada

I. Introduction to Booth's algorithm

The traditional way to compute a signed $M \times N$ multiplication is performed by properly adding M sign-extended and shifted partial products. A signed multiplication therefore implies the summation of M partial products and is expected to be slow when M contains many bits. Each columns sums M bits and gives one or more carry bits to the following, leaving the multiplication result's MSB waiting for $2 \times M - 2$ carry signals to be generated. In order to increase the multiplier's speed and reduce its area, a modified Booth algorithm can be used. For a Radix-4 implementation, this reduces the amount of partial products by a factor of 2.

The Booth algorithm is quite simple; it encodes the multiplier N in overlapping groups of three bits to generate three control signals, SINGLE, DOUBLE and NEGATE. The decoder can compute the partial product according to the latter in five different ways:

1) PP=0's (multiplication by 0)
2) PP=M (multiplication by 1)
3) PP=-M (multiplication by -1)
4) PP=2M (multiplication by 2)
5) PP=-2M (multiplication by -2)

All these partial products are then sign extended appropriately and shifted left by 2 from each other. The two last stages of the multiplication compute the sum of these partial products. The result is (M+N) bits wide.

II. Booth Multiplier Architecture

Since our application is an 8x8 Booth multiplier, we have broken down the design in four distinct parts: 1) the encoder, 2) the selector (decoder), 3) the compressor tree and 4) the final addition. A schematic was created for each wordslice module in Electric and was tested for DRCs. They were then simulated in ModelSim for verification of correctness.

A. Radix-4 Modified Booth Encoder

For an 8x8 Booth multiplier, the wordslice encoder is made of 4 bitslice encoder. The latter encodes the multiplier using N[7:5], N[5:3], N[3:1] and N[1:0]&0 to generate SINGLE[3:0], DOUBLE[3:0] and NEGATE[3:0].

B. Booth Selector

The Booth selector uses the three encoded vectors to generate four different 9bits wide partial products. Its bitslice design is quite simple; you either shift left by one (multiply by 2) or invert the bit or both. The wordslice's first subtlety is to remember to add '1' when you negate the signal, to satisfy the two's complement representation. For that matter, 9 half-adders were used to sum the NEGATE signal (0 when positive, 1 when negative; propagate). The second subtlety was in the sign generation. The selector and half-adder would cover almost every possible case, except when Y= -128 and PP= -2M. The correct answer is 100000000 and sign=0. A simple 3-stage NAND8 reducer was used to cover this special case.

C. Partial Products Compressor Tree

The compressor tree uses thirteen [4:2] compressors optimized for area to sum the columns. Each compressor has five inputs: A, B, C, D and C_{int-1} and three outputs: Sum, C_{int} and C_{ext}. The internal carry C_{int} is propagated to the next column and the external carry C_{ext} is given to the final addition, thus balancing the propagation delay.

D. Final Addition

The final addition is done using a 13bits carry-ripple adder. This type of adder is optimized for area and can easily be implemented in layout, but has the longest propagation delay of all adders.

III. 8x8 Booth Multiplier Simulation

An additional simulation was done in Quartus II to test the overall functionality of our Booth multiplier. The integrated ModelSim simulator was found to be very useful to quickly generate stimuli vectors. Our modules result is compared to the output of a generic 8x8 multiplier using a XOR-tree and is tested extensively.