

Booth Multiplier for DSP Instructions

|  |  |
| --- | --- |
| Document Number | DSP-xxxxx |
| Date Issued | 2015-05-14 |
| Status | Final Release |



**CONFIDENTIAL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision History | | | | |
| Rev. | Revision Date | Revised Chapter-Section | Revised Content | Author |
| 1.0 | 2015-05-14 | All |  | Yung-ching |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Review History | | | | |
| Rev. | Review Date | Reviewed Chapter-Section | Comments | Reviewer |
| 0.1 | 2014-mm-dd |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Document Sign-off List | | | |
| Department | Name | Date | Comment |
| VLSI |  | 2014-mm-dd |  |

Table of Contents

[1. Overview 1](#_Toc378093261)

[1.1. Introduction 1](#_Toc378093262)

[1.2. Features 1](#_Toc378093263)

[1.3. Block Diagram 1](#_Toc378093264)

[1.4. Function Description 2](#_Toc378093265)

[2. Signal Description 3](#_Toc378093266)

[3. Micro-Architecture 5](#_Toc378093267)

[4. Design Configuration 6](#_Toc378093268)

[5. Programming Model 7](#_Toc378093269)

[5.1. Summary of Registers 7](#_Toc378093270)

[5.2. Register Description 7](#_Toc378093271)

[5.3. Programming Sequence 7](#_Toc378093272)

List of Tables

[Table 1. Signal Definition 3](#_Toc378093273)

[Table 2. Register Summary 7](#_Toc378093274)

List of Figures

[Figure 1. Block Diagram 1](#_Toc378093275)

[Figure 2. Logic Symbol 3](#_Toc378093276)

Typographical Convention Index

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Document Element | Font | Font Style | Size | Color |
| Normal | Candara | Normal | 11 | Black |
| Code | Courier New | Normal | 11 | Indigo |
| USER VARIABLE | COURIER NEW | ALL-CAPS | 11 | INDIGO |
| Note/Warning | Candara | Normal | 11 | Red |
| Hyperlink | Candara | Bold + Underlined | 11 | Blue |

Overview

The N1068A-S multiplier is a booth algorithm based multiplier. The multiplier implements signed/unsigned 32-bit x 32-bit to 64-bit multiplications. The DSP instruction for D1068-S additionally requires dual 16-bit x 16-bit to 32-bit multiplications. This article describes the multiplier design that supports the new requirements with minimal timing overhead.

## Introduction

The multiplier support the following instructions:

* Signed 32-bit x 32-bit = 64-bit multiplication
* Unsigned 32-bit x 32-bit = 64-bit multiplication
* Signed dual 16-bit x 16-bit = 32-bit multiplication
* Unsigned dual 16-bit x 16-bit = 32-bit multiplication

## Features

* See Introduction.

## Block Diagram

N/A

## Function Description

N/A

# Signal Description

ieu\_mdu\_mdul

mul\_en

mul\_en\_32b

mdu\_signed\_ex

mul\_in0[31:0]

mul\_in1[31:0]

mul\_out[63:0]

Figure 2. Multiplier Symbol

Table 1 gives the detailed descriptions of I/O signals.

Table 1. Signal Definition

| Signal Name | I/O Type | Description |
| --- | --- | --- |
| mul\_en | I | Enable the multiplier (for logic gating) |
| mul\_en\_32b | I | 1: 32-bit multiplication 0: dual 16-bit multiplication |
| mdu\_signed\_ex | I | 1: signed multiplication |
| mul\_in0[31:0] | I | Multiplicant |
| mul\_in1[31:0] | I | Multiplier |
| mul\_out[63:0] | O | Multiplication result |

Micro-Architecture

## Main micro-arch

Use Radix-4 Booth Encoding to create partial products and sum them together by CSA adders.

Radix-4 Booth Encoding: X is the multiplicand (mul\_in0). Y is the multiplier (mul\_in1).

|  |  |  |
| --- | --- | --- |
| Y[2i+1,2i,2i-1] | Partial Product | comment |
| 3’b00\_0 | 0 |  |
| 3’b00\_1 | X | Carryout from previous partial product |
| 3’b01\_0 | X |  |
| 3’b01\_1 | 2X | X + carry out of previous partial product |
| 3’b10\_0 | -2X | Next partial product will assume a carry out, so we need to deduct the result by -2X |
| 3’b10\_1 | -X | 3’b10\_0 case + carry out from previous partial product |
| 3’b11\_0 | -X | Next partial product will assume a carry out, so we need to deduct the result by -X |
| 3’b11\_1 | 0 | 3’b11\_0 case + carry out from previous partial product |

2X is shift operation. -2X & -X requires finding two’s complement number of the multiplier:

-2X = ~(2X) + 1,

-X = ~X + 1,

where 1 is the carry in.

Recode\_pp\*() function deals with the main partial product from the multiplicand, Recode\_cin() function computes the required carry in of the corresponding partial product.

The recode\_pp\*() function has two versions to support 16-bit multiplications: recode\_pp\_l() and recode\_pp\_h(). Recode\_pp\_l() is modified to support the bottom 16-bit multiplications and recode\_pp\_h() for the top 16-bit multiplications.

## Dealing with both signed & unsigned numbers

The Booth algorithm only works for signed numbers. Some adjustments are needed for unsigned numbers:

1. Treat 32-bit unsigned number as 33-bit signed number. But since we are working with Radix-4 algorithm, this has to be sign-extended by one more bit to 34-bit.
2. The 32-bit signed numbers become signed-extended 34-bit multiplications.

Therefore our booth multiplier will generate 17 partial products: pp\_x0 pp\_x2 .. pp\_x32.

## Simplify the Sign-Extension Fan-outs

Our 34-bit x 34-bit multiplier needs to product 64-bit results. This means the partial product must be sign-extended to 64-bit. This creates extra loadings for the leading bits. Observe 0x000000001 x 0xaaaaaaab – the yellow background digits are sign extensions for supporting signed/unsigned multiplication. The grey background digits are sign extensions required for computing the partial products in the Booth algorithm. The red digits are the actual sign digits (mul\_in0[31]), and the diagram below shows the loading of the sign digits.



Work around: invert the msb of sign-extension bit of the multiplicand and rewrite the partial product into the grey background part + our new partial product:



The carry chain will be the same. But with the greyed out constants for sign-extension, we can add them together:



## Dealing with 16-bit multiplications

DualMul16({mul\_in0[31:16], mul\_in[15:0]} \* {mul\_in1[31:16], mul\_in1[15:0]}) = Mul16(mul\_in0[31:16]\*mul\_in1[31:16]) and mul16(mul\_in0[15:0]\*mul\_in1[15:0])

For partial product pp\_x0 – pp\_x14, we care only about bit [31:0]. They are produced by mul\_in0[15:0] and mul\_in1[15:0].

For partial product pp\_x16 – pp\_x32, we care only about bit[63:32]. They are produced by mul\_in0[31:16] and mul\_in[31:16].

### Split the multiplier to two halves

For encoding Radix-4 partial products, these means mul\_in1[17:16] do not need to look at mul\_in1[15] under 16-bit multiplication mode.

### Handling both signed/unsigned 16-bit multiplications

For the bottom half product, to support both signed and unsigned multiplication, we should in fact perform 18-bit x 18-bit multiplication. This means we need one additional partial product: pp\_x1616 (same reason & equations for pp\_x32 under 32-bit multiplication)

### Dealing with unwanted bits of partial products

Unwanted sign extension bits, Unwanted trailing bits when negated, Unwanted carry propagations:

Observe 0x000010001 x 0xaaabaaab below.

* + For pp\_x0 .. pp\_x1616, there are sign extension bits. The greyed out part should be masked out so that they do not interfere with the upper half result.
  + For pp\_x16 .. pp\_x32, the only part we are interested is the double-underlined part. The underlined part should be zero out (so that negation will not create 1 for the bottom half result.)
  + When summing the partial products, the carry chain should not propagate across 32-bit boundaries.



### Sign-extension optimization



# Design Configuration

N/A

Programming Model

## Summary of Registers

N/A