

Radix-4 SRT Divider Verification Spec

|  |  |
| --- | --- |
| Document Number | VSP-00019 |
| Date Issued | 2015-03-16 |
| Status | Final Release |



**CONFIDENTIAL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision History | | | | |
| Rev. | Revision Date | Revised Chapter-Section | Revised Content | Author |
| 1.0 | 2015-03-24 | all | Initial revision | Feng |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Review History | | | | |
| Rev. | Review Date | Reviewed Chapter-Section | Comments | Reviewer |
| 1.0 | 2015-03-24 | All | Approved for final release | Justin, Yung-Ching, Taco |

|  |  |  |  |
| --- | --- | --- | --- |
| Document Sign-off List | | | |
| Department | Name | Date | Comment |
| VLSI | Yung-Ching | 2015-04-24 | Approved for final release |

Table of Contents

[1. Overview 1](#_Toc415576474)

[1.1. Introduction 1](#_Toc415576475)

[1.2. Features 1](#_Toc415576476)

[1.3. Block Diagram 2](#_Toc415576477)

[1.4. Function Description 2](#_Toc415576478)

[1.4.1. Quotient Lookup Table 3](#_Toc415576479)

[2. Verification Plan 5](#_Toc415576480)

[2.1. Verification Methodology 5](#_Toc415576481)

[2.2. Singularities of the SRT Divider 6](#_Toc415576482)

[2.3. Verify Overlap Region Boundaries 6](#_Toc415576483)

[2.4. Hitting corner cases in the middle of division 7](#_Toc415576484)

[3. Test Patterns 9](#_Toc415576485)

List of Tables

[Table 1. Test Patterns for Division Testing 9](#_Toc415576378)

List of Figures

[Figure 1. Block Diagram of Radix-4 SRT Divider related blocks for signed division 2](#_Toc415576373)

[Figure 2. p-d plot for Radix-4 Division with q ∈[-2,2] 3](#_Toc415576374)

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

This document describes the features and verification plan for Radix-4 SRT Divider.

## Introduction

The Radix-4 SRT Divider speeds up the division process by generating more bits of the quotient during each iteration. The quotient selection process for high radix divider is more prone to error, and this verification spec proposes a verification plan for complete verification of the Radix-4 SRT Divider.

The verification of high radix SRT divider is done by using a modified version of SRTEST program, which tests the corner cases of the quotient lookup table.

## Features

The Radix-4 SRT Divider handles two types of division, unsigned division and singed division. Both unsigned and signed division divides 2 32-bit numbers and generates a 32-bit quotient and 32-bit remainder; with signed division the numbers are in 2’s complement.

Before division, the dividend and divisor will be converted to absolute value, and scaled to appropriate magnitude. The division process will then take place to generate the quotient and remainder.

During the division process, each quotient digit is selected from a table of eligible digits. A few leading bits of the divisor determine which column of the look-up table will supply all the quotient digits for any given instance of division. A few leading bits of the current remainder (initially it is the dividend) determine the row. The quotient digit extracted from the selected row and column is then multiplied by the divisor and their product subtracted from the current remainder; the difference is the new remainder that will determine the next quotient digit.

After sufficiently many quotient digits have been extracted by the foregoing process, they are assimilated into a conventional quotient to produce the result of the division operation. The polarity of the remainder will then be converted to match the polarity of the dividend; the polarity of quotient will changed to be positive if the polarity of the dividend and the divisor are the same, and be negative otherwise.

## Block Diagram

The Radix-4 SRT Divider related blocks are shown in Figure 1.

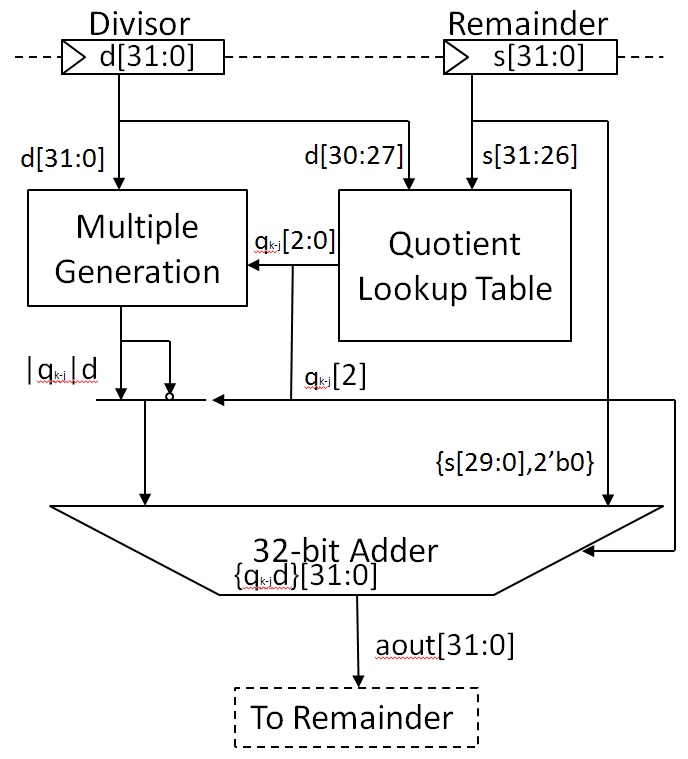


Figure 1. Block Diagram of Radix-4 SRT Divider related blocks for signed division

## Function Description

* Multiple Generation
* Generates multiples of d based on the selected quotient digit
* Generates 0 if q=0, generates 1 if q=1 or q=-1, generates 2 if q=2 or q=-2
* Quotient Lookup Table
* The quotient lookup table selects the next quotient digit based on the MSB 5 bits of remainder and MSB 4 bits of divisor
* Details elaborated in section 1.4.1
* 32-bit Adder
* The adder reuses the original ALU in the processor
* Carry-save adder is not needed because the divider is not on the critical path

### Quotient Lookup Table

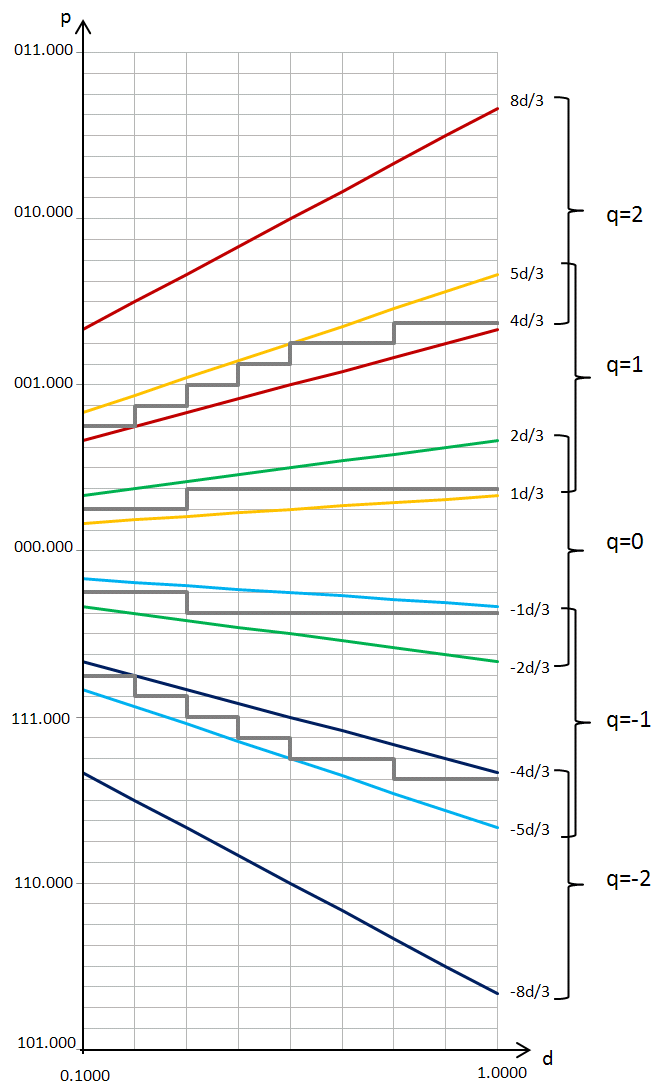


Figure . p-d plot for Radix-4 Division with q ∈[-2,2]

The p-d plot shown in Figure 2 shows how the quotient is selected form the quotient lookup table, the selection of quotient digits depends on the MSB few bits of divisor and remainder. Feasible quotients are shows on the right of the graph. For example, “q=2” is feasible for the range between 8d/3 and 4d/3, if a divisor, remainder pair lies between these two boundaries, q can be selected as 2. In this design we choose 5 bits of remainder and 4 bits of divisor.

The grey lines in the p-d plot is the region boundary, if the value of (d, p) is above or lands on the region boundary, the lookup table chooses the larger possible quotient from the right, otherwise it chooses the lesser possible quotient. If the region boundary is inappropriately set, it will cause the remainder to overflow, and lead to incorrect division results.

# Verification Plan

There are several characteristics of the divider that needs to be verified

* Corner cases
* Divide by zero
* Quotient overflow
* Dividend lies in/near infeasible region
* Quotient selection table corner cases
* Operations
* Sign conversion
* Prescaling
* Random tests
* Division with random number

Among the characteristics, the quotient selection table is most complicated to verify, in the following chapters we’ll discuss the verification methodology and algorithm for verifying this unit.

## Verification Methodology

In the theory of complex variables, the singularities of an analytic function are the boundary points of its domain of analyticity. Inside its domain an analytic function is smooth, infinitely differentiable.

Logical functions of numerical arguments are piecewise constant; their singularities are the arguments where the functions change value. To know a logical function at and near its every singularity is to know it everywhere. To prove the logical function of the quotient selection table correct our program need check it only at and near every numerical input where either the state of the quotient selection table or the desired result changes.

## Singularities of the SRT Divider

We select the singularities with the following algorithm

N = the number of leading bits of divisors that go to the lookup table.

P = the number of leading bits of reminders that go to the lookup table

L = Log2( Radix ) ; i.e., Radix = 2^L .

M = maximum bit-width preassigned for divisors.

The following formula will generate (2\*L + 1)\*2^(N-1) divisors, each an M-bit integer of the form

2.0^M - j\*2.0^(M-N) + S for j = 0, 1, 2, ..., 2^(N-1)

Where S runs through the set

{ -2^(L-1), ..., -4, -2, -1, 0, 1, 2, 4, ..., 2^(L-1) } .

These perturbations S provide divisors on and near the boundaries of vertical strips scaled up by 2^(M-1) .

The original formula in SRTEST is designed for divider with carry save adder and uses numerous dividends to generate as much combination of carry/sum pairs as possible. But since we use carry ripple adder to generate our remainder, there is no error in the partial remainder and we only need to test the values near the singularities to verify the quotient selection table.

The following formula will generate (2\*L + 1)\*2^P dividends, each an M-bit integer of the form

2.0^M - j\*2.0^(M-P) + S for j = 0, 1, 2, ..., 2^(P)

Where S runs through the set

{ -2^(L-1), ..., -4, -2, -1, 0, 1, 2, 4, ..., 2^(L-1) } .

These perturbations S provide dividends on and near the boundaries of horizontal strips scaled up by 2^(M-1) .

## Verify Overlap Region Boundaries

An overlap region is the region where the quotient digit can be either one of two values without generating unrecoverable remainder. Remainder/divisor values are more likely to cause a quotient lookup error if the value pair is near the overlap region boundaries. Based on this assumption, we use the following method to verify the values near overlap region boundaries.

The boundary of overlap regions can be represented with the following equation.

Z = D\*X

Where

Z is the dividend

X belongs to the set {5/3, 4/3, 2/3, 1/3, -1/3, -2/3, -4/3, -5/3} if quotient uses digit set [-2,2]

We scale up the dividend to the range of 2^(M-1) to 2^M and add small offset, to generate more digits of quotient and verify near-corner cases.

## Hitting corner cases in the middle of division

Aside from hitting the singularities or overlap region boundaries with the (divisor, dividend) pair, we can also intentionally hit these corner cases in the middle of division to verify if the lookup mechanism might fail under such circumstances. The method is shown below.

The relation of dividend, divisor, quotient, and remainder can be written as the following

Z: dividend, D: divisor, r: radix, R: remainder

Z = D\*q[31:30]\*r^30 + D\*q[29:28]\*r^28 + D\*q[27:26]\*r^26 + … + D\*q[1:0]\*r^0 + R

Each iteration of the division process, we generate 1 digit of q, and subtract it from the remainder. The remainder at the Nth iteration can be calculated with the following equation.

R(n) = Z - D\*q[31:30]\*r^30 - D\*q[29:28]\*r^28 - … - D\*q[33-2n:32-2n]\*r^(32-2n)

To hit corner cases at the Nth iteration, we use formulas in section 2.2 and 2.3 to generate corner case values for R(n). R(n) should be scaled to magnitude smaller than r^(33-2n), and add the previously subtracted products to generate the dividend.

Z = R(n) + D\*q[31:30]\*r^30 + D\*q[29:28]\*r^28 + … + D\*q[33-2n:32-2n]\*r^(32-2)

Where the value of q[31:32-2n] is randomly selected.

# Test Patterns

Table . Test Patterns for Division Testing

|  |  |
| --- | --- |
| Pattern Name | Description |
| **Existing Patterns in Vericore Environment** | |
| avp/mul\_div\_random | A self-checking test case to verify random DIV and MUL operations   * Random Signed/unsigned 32bit number as dividend and divisor * Tests div/divs/divr/divsr instructions * Checks results with calculation in Perl * Checks results when arithmetic exception occur |
| avp/mul\_div\_seq | Randomly mix MUL, DIV and other ALU operations   * Random multiplication/division instructions * No self-check |
| avp/mul\_div\_special\_operand | A test case including the combination of special operands as dividend/divisor   * Walking one * Walking zero * Random leading zero * Random leading one * 0/1/2/3 * 2^31 + 0/1/2/3 * 2^31 – 0/1/2/3 * 2^32 – 1/2/3/4 * 0x55555555, 0xaaaaaaaa   Result self-checked with calculation in Perl |
| avp/mul\_div\_special\_operand\_seq | Sequential MUL/DIV operations with special operands.   * Initialize operands with special values * Sequential multiplication/division * No self-check |
| avp/mul\_div\_stalled | A test case mixing syscall and load/store instruction to verify MUL/DIV when pipeline is stalled   * Stall pipeline with load, division will continue at EX stage |
| **Patterns to be added** | |
| avp/div\_srtest | Test the singularities of divider   * Test with divr/divsr instructions * Use near singularity values as divisor and dividend * Test all near singularity values with all possible polarities * Randomly scale magnitude of divisor * Tests dividend in infeasible region * Self-check result with Perl calculation |
| avp/div\_simple1 | Tests the overlap region boundaries of SRT divider   * Test with divr/divsr instructions * Control dividend value to be near singularity value at random cycle * Randomly scale magnitude of divisor * Randomly set polarity of operators * Self-check result with Perl calculation |
| avp/div\_simple2 | Tests the case where division hits corner case value in the middle of division process   * Test with divr/divsr instructions * Control temporary remainder value to be near singularity or overlap region boundary value at random cycle * Randomly set polarity of operators * Self-check result with Perl calculation |