EMERGING TECH CONFERENCE – Edge Intelligence

Volume 02, 2023, Page 64 – 70

Proceedings of Emerging Tech Conference:
Edge Intelligence 2023

High-Performance Design of SRT IP Blocks

Aristotelis Tsekouras[1], Giorgos Stagakis[1], Anastasis Avgoustidis[2], Grigoris Kokkonis[1], Konstantinos Gkekas[2], Vasilis F. Pavlidis[1], Thomas Noulis[2] and Georgios Keramidas[3,4]

[1] *Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Greece*
[2] *Department of Physics, Aristotle University of Thessaloniki, Thessaloniki, Greece*
[3] *Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece*
[4] *Think Silicon, S.A. An Applied Materials Company, Patras, Greece*
{aristotet gstagakis}@ece.auth.gr, anavgous@physics.auth.gr, gkokkonis@ece.auth.gr, kogkekas@physics.auth.gr, vpavlid@ece.auth.gr, tnoul@physics.auth.gr, gkeramidas@csd.auth.gr

Abstract

The design and verification of three floating-point components (division, reciprocal, inverse square root), based on the Sweeney, Robertson and Tocher (SRT) algorithm and implemented in SystemVerilog, that operate with up to three pipeline stages are presented. The design flow for these blocks utilizes the Genus tool from Cadence for synthesis. To ensure proper functionality, Chipware components provided by Cadence serve as a golden reference and a means of validating the correctness of the blocks. A SystemVerilog testbench is developed, instantiating the designed SRT components as well as the respective Chipware components, enabling comprehensive functional testing across various rounding modes and precisions. While the components are specifically designed to support half, single, and bfloat16 precisions, they are also adaptable to custom precisions. The functional correctness of the components is evaluated through coverage analysis using the Integrated Metrics Center (IMC) of the Xcelium tool, achieving industry-level toggle, block, and expression coverage for all parameter combinations. Furthermore, equivalency checking is performed using the Conformal tool during different synthesis stages. Power consumption results are obtained using Joules.

1 Introduction

The motivation behind this work stems from the increasing demand for these components in scientific and data-intensive applications, where accurate and efficient computations are crucial. This work aims to enhance computational capabilities, improve numerical accuracy, and enable complex mathematical operations with low power in relevant application domains.

The floating-point components reciprocal (1/a), division (a/b) and inverse square root ($1/\sqrt{aa}$) have been developed in SystemVerilog, synthesized, and verified using Cadence tools. The block diagrams at register transfer level (RTL) for the three components are depicted in ***Figure 1***. The mantissa width, the exponent width, the rounding modes, and the number of pipeline stages are all parameterized. Following the RTL design of the components, a testbench was created to facilitate verification and coverage analysis. Subsequently, the designs undergo formal verification, synthesis, and logic equivalency checks. Finally, the power of the components is measured through Joules, and the components can be used as soft IPs.

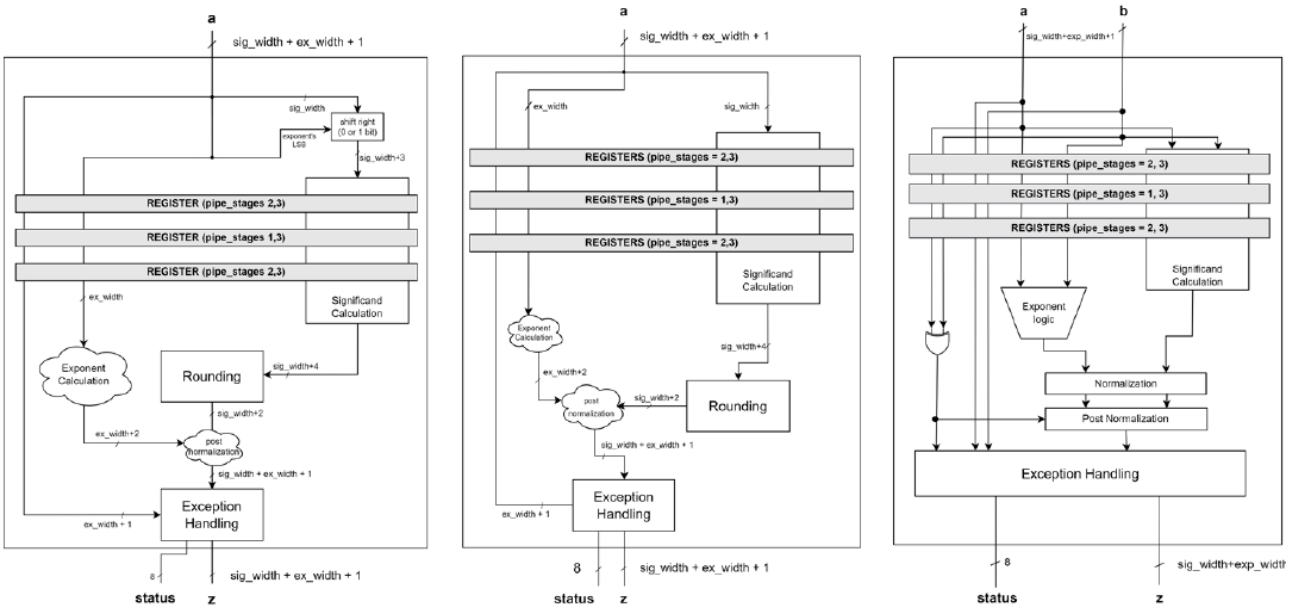

***Figure 1**: Block diagram of the inverse sqrt, reciprocal, and division components respectively*

Despite the undisputed advantages that IoT technologies bring to the Sports domain, currently their large

2 Design Description

The three components have many similar characteristics since all of these components use the SRT algorithm to determine the significant part of the floating-point number. Each component is divided into four parts, which are responsible for the exponent calculation, the significand calculation, the rounding and post-normalization, and the exception handling. The input number of a floating-point root component is a normalized number represented as

$$\alpha = (-1)^s \cdot 2^{e-bias} \cdot (1.f) \tag{1}$$

where *s* is the sign, *e* is the biased exponent and *f* is the fraction of number *a*.

2.1. Exponent Calculation

The exponent of the result (*z*) can be determined through the following equations depending on the specific operation of the component.

Division: $e_x = (e_a - bias) - (e_b - bias) + bias = e_a - e_b + bias$ (2)

Reciprocal: $e_z = -(e_a - bias) + bias = 2 \cdot (bias - e_a)$ (3)

Inverse square root: $e_z = -\frac{e_a - bias}{2} + bias = \frac{3}{2} bias - \frac{e_a}{2} - e_a[0]$ (4)

where e_a (and e_b) are the unbiased exponents of inputs *a* (and *b*). Note that in the case of the inverse square root component, if the biased exponent of the input is odd, the significand must be shifted 1-bit left such that the division by 2 is exact.

2.2. Significand Calculation

The SRT algorithm is a digit-recurrence method producing an approximation of the result. More precisely a radix-4 (r = 4), carry-save version of the algorithm is implemented using redundant representation. This redundancy implies that each quotient digit is chosen from five possibilities: {−2, −1, 0, +1, +2}, and is encoded using One-Hot encoding. Consequently, these possibilities are encoded as: {1000, 0100, 0000, 0001, 0010} respectively.

In order to avoid the normalization step after calculating the significand of z, the inputs are shifted to the right. In this way, the inverse square-root component $\alpha \in \left[\frac{1}{4}, \frac{1}{2}\right)$ results to $z = \frac{1}{\sqrt{a}} \in (1,2]$ and the reciprocal component $\alpha \in \left[\frac{1}{2}, 1\right)$ results to $z = \frac{1}{a} \in (1,2]$. On the contrary, for the division component in order to use the same selection function with the reciprocal component $\alpha \in \left[\frac{1}{8}, \frac{1}{4}\right)$ and $b \in \left[\frac{1}{2}, 1\right)$ results to $z = \frac{a}{b} \in \left[\frac{1}{8}, \frac{1}{2}\right)$ thus the result must be shifted 2 or 3 bits to the left.

For computing the result, a residual is defined for each operation as:

- Inverse square-root $\left({}^{1}/_{\sqrt{d}}\right)$: $w[j] = 2^{-1} \cdot 4^{j} \cdot (1\text{-}dQ[j]^2)$ (5)
- Division $\left({}^{x}/_{d}\right)$: $w[j] = 4^{j} \cdot \left(\frac{x}{8} - \left(\frac{d}{2}\right) Q[j]\right)$ (6)
- Reciprocal $\left({}^{1}/_{d}\right)$: $w[j] = 4^{j} \cdot (1 - dQ[j])$ (7)

where $Q[j] = Q[0] + \sum_{i=1}^{j} q_i \, 4^{-i}$ is the partial result up to iteration j.

From the above residual definitions, the recurrences are:

- Inverse square-root $\left({}^{1}/_{\sqrt{d}}\right)$: $w[j+1] = 4w[j] + q_{i+1}Q[j]d \text{ -}2^{-1} \, q_{j+1}^2 4^{-(j+1)d}$ (8)
- Division $\left({}^{x}/_{d}\right)$: $w[j+1] =\cdot\, 4w[j] \text{ -}q_{j+1^d}$ (9)
- Reciprocal $\left({}^{1}/_{d}\right)$: $w[j+1] =\cdot\, 4w[j] \text{ -}q_{j+1^d}$ (10)

To simplify the implementation of the inverse square-root recurrence steps, two variables are defined:

$$D[j] = dQ[j] \text{ and } C[j] = 2^{-1} \cdot 4 \text{ - } [j + 1] \quad (11)$$

Using these variables, we can obtain the recurrences:

$$w[j+1] = 4w[j] \text{ - } q_{j+1}D[j] \text{ - } q_{j+1}^2 C[j] \;\; D[j+1] = D[j] + 2q_{j+1}C[j], \; C[j+1] = 4^{-1}C[j] \quad (12)$$

Since q_{j+1} is an encoded digit, MUXs can be used for the multiplications of $q_{j+1}d$, $q_{j+1}D[j]$ *etc*. The q_{j+1}digits are selected in a way to bound the absolute error of $Q[j]$ ([1], [2]). The selection function uses an estimation of the residual w and an estimation of d (D for the inverse square root component).

D_L	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
m_{-1}	12	13	14	14	15	15	16	17	17	18	19	19	20	21	21	22
m_0	3	4	4	4	4	4	4	5	5	7	7	6	5	6	8	8
m_1	-5	-5	-5	-6	-6	-6	-7	-7	-7	-8	-8	-8	-9	-9	-9	-10
m_2	-13	-14	-14	-15	-16	-17	-18	-18	-19	-19	-20	-21	-23	-23	-24	-25

D_L is scaled by 32 and m_k is scaled by 16

Table 1: *Selection function for division and inverse square root components*

D_L	8	9	10	11	12	13	14	15
m_{-1}	12	14	15	16	18	20	20	24
m_0	4	4	4	4	6	6	8	8
m_1	-4	-6	-6	-6	-8	-8	-8	-8
m_2	-13	-15	-16	-18	-20	-20	-22	-24

D_L is scaled by 8 and m_k is scaled by 16

Table 2*: Selection function for reciprocal component*

The selection is performed using ***Table 1*** and ***Table 2*** as follows: for $\tilde{d} = D_1(i)$ ($\tilde{D} = D_1(i)$ in case of inverse sqrt),

$$q_{j+1} = -2 \; if \; 4\tilde{w} < m_{-1} \tag{13}$$

$$q_{j+1} = k(-1 \le k \le 1) \text{ if } m_k \le 4\tilde{w} \le m_{k+1} \tag{14}$$

$$q_{j+1} = 2 \; if \; 4\tilde{w} \ge m_2 \tag{15}$$

Where $\tilde{d}$: 4 bits after the MSB of (the MSB of *d* is not an input as it is always 1), $\tilde{D}$: 5 MSBs of D and $4\tilde{w}$: 7 MSBs of *w*.

Since the digits *q* are encoded, there is a need to convert these digits on the fly from redundant signed-digit representation to conventional (Q). This conversion is performed in each step of the recurrence and the result depends on the encoded digit *q* as well as the decoded quotient Q of the previous step (***Table 3***). For a detailed explanation see [3, p. 256].

q_{j+1}	Q[j+1]	QM[j+1]
+2	{Q[j], 2'b10}	{Q[j], 2'b 01}
+1	{Q[j], 2'b 01}	{Q[j], 2'b 00}
0	{Q[j], 2'b 00}	{QM[j], 2'b 11}
-1	{QM[j], 2'b 11}	{QM[j], 2'b 10}
-2	{QM[j], 2'b 10}	{QM[j], 2'b 01}

Table 3*: On the fly conversion*

The SRT algorithm produces two bits of the quotient in each step meaning that in order to produce the full *sig*_width width significand plus the guard, round, and sticky bit the required number of steps are $\left\lceil \frac{\text{sigwith} + 3}{2} \right\rceil$. Finally, a full addition is needed to calculate the final residual *w* and determine the correct quotient depending on the sign of *w* (if $w \ge 0$: Q *else* QM).

2.3. Rounding

Since the result is already normalized, there is only a need for rounding and, if needed, re-normalization, in case the rounding leads to an overflow. The rounding can be implemented by assigning to the *round* parameter 1 out of 6 rounding modes as reported in ***Table 4***. For the rounding, three extra bits (guard, round, and sticky) after the mantissa are used to determine whether the result should be incremented. The guard bit determines whether the unrounded result is above or below the middle of the two nearest representable values, while the round bit acts as a tiebreaker, and the sticky bit as an indication of what is contained in the lesser significant bits that are not kept.

Value	Description
"IEEE_near"	Round to the nearest representable value, if both are equally near, output the result with an even significand
"IEEE_zero"	IEEE round towards zero
"IEEE_pinf"	IEEE round to +Infinity
"IEEE_ninf"	IEEE round to -Infinity
"near_up"	Round to the nearest representable value, if both are equally near, output the result closer to +Infinity
"away_zero"	Round away from zero

Table 4: *Supported rounding modes*

2.4. Exception Handling

Special case values, listed in ***Table 5*** for input *a* (and *b* in case of division), are handled separately by flushing the result *z*. For example, $\frac{1}{0} = +\infty, \frac{1}{\sqrt{-0}} = \infty, \frac{+\infty}{-0} =$ -∞Denormals are flushed to sign preserved ± Zero or ± MinNorm, depending on the rounding mode, where Denormals are defined to be nearer to Zero than MinNorm.

Name	Sign	Exponent	Significand	Value
±Zero	±	0	0	±0
±Inf	±	11…11	0	±Inf
Denormal	S	0	≠0	$(-1)^s \cdot 2^{1-bias} \cdot 0.sig$
Normal	S	0<exp<11…11	sig	$(-1)^s \cdot 2^{exp-bias} \cdot 1.sig$
±MinNorm	±	1	0	$(-1)^s \cdot 2^{1-bias}$

Table 5: *Special case values*

3 Verification

To verify the correctness of the designs, three comprehensive testbenches compare the behavior of the division, reciprocal, and inverse square root components with the output from the respective Chipware/Designware components. The testbenches cover all possible parameter combinations and utilized covergroups to capture inputs and outputs. Cross-coverage is also employed for precise analysis. Special consideration is given to denormals and NaNs with illegal bins designated for these cases at the outputs, as well as for the unreachable status flags.

Regarding the covergroups, directed bins are defined differently for each of the components. For

reciprocal and inverse square root, we use 64 bins for inputs and outputs, while for division we use two covergroups, where the first group is used to employ cross coverage exclusively on normal numbers with 32 bins, and the second group considers the normal numbers as one bin and uses 8 bins for the corner cases. We use twelve corner cases during the testing of all three components, which are the following: One specific positive/negative signaling NaN, one specific positive/negative quiet NaN, one specific positive/negative normal, one specific positive/negative denormal, positive/negative Infinity, and positive/negative Zero.

The coverage analysis for division involved comparing the outputs of 10 million inputs and the corner cases as previously mentioned. The inverse square root and reciprocal blocks are covered by 5 million inputs as they have only one input. The coverage results for all the three implemented circuits are reported in ***Table 6***.

Coverage Type	Division	Reciprocal	Invs Sqrt
Block Coverage	100%	100%	100%
Expression Coverage	100%	100%	100%
Toggle Coverage	88%	94%	92%
Covergroup Coverage	100%	100%	100%

Table 6: *Percentage of each coverage type for the three designed circuits*

4 Results

The synthesis results are produced using the 45 nm fast library gsclib045 provided by Cadence. The default Genus synthesis flow [4] is used in addition to an incremental optimization at the end of the synthesis stage. The results are summarized and compared with the respective Chipware in ***Table 7***. All the results have been produced after Synthesis. Area and timing results are produced by Genus, while power results are produced by Innovus.

Note that our designs have a disadvantage in terms of area when compared to their respective Chipware counterpart. However, the power consumption of our SRT blocks are comparable or lower to Chipware, enabling their use for low power applications.

Component	Area (μm^2)	TNS (ps)	Power (mW)
Division @ 6 ns	9956.17	0	2.238
Division CW @ 6 ns	6634.87	0	1.667
Reciprocal @ 6 ns	5555.24	0	1.435
Reciprocal CW @ 6 ns	5746.01	1	1.47
Inverse Sqrt @ 10 ns	23831.12	0	3.792
Inverse Sqrt CW @ 10 ns	14823.24	0	2.243

Table 7: *Power, performance, and area (PPA) results for the SRT components for zero pipeline stages compared to the Chipware components*

5 Conclusion

Three floating-point SRT components, demonstrating the accuracy, functionality, and power efficiency are

designed and verified. These components contribute to the advancement of computational capabilities and numerical accuracy in a range of applications. The presented blocks exhibit comparable performance with the commercial IP blocks offered by Cadence while offering higher versatility through a customizable range of pipeline stages to suitably adapt to the requirements of the target application.

6 References

[1] E. Antelo, T. Lang, P. Montuschi and A. Nannarelli, "Low latency digit-recurrence reciprocal and square-root reciprocal algorithm and architecture," 17th IEEE Symposium on Computer Arithmetic (ARITH'05), Cape Cod, MA, USA, 2005, pp. 147-154, doi: 10.1109/ARITH.2005.29.

[2] T. Lang and E. Antelo, "Radix-4 reciprocal square-root and its combination with division and square root," in IEEE Transactions on Computers, vol. 52, no. 9, pp. 1100-1114, Sept. 2003, doi: 10.1109/TC.2003.1228508.

[3] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, 1st ed., Morgan Kaufmann, 2003, pp. 247-330.

[4] *All Products | Cadence - Cadence Design Systems*. Available at: www.cadence.com/en_US/home/tools/tools-a-z.html (Accessed: 01-10-2023)