Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

A reduced-complexity lookup table approach to solving mathematical functions in FPGAs

Michael Low, Jim P. Y. Lee

Defence R&D Canada – Ottawa
Technical Memorandum
DRDC Ottawa TM 2008-261
December 2008

Canada

A reduced-complexity lookup table approach to solving mathematical functions in FPGAs

Michael Low
Jim P. Y. Lee
DRDC Ottawa

Defence R&D Canada – Ottawa

Technical Memorandum
DRDC Ottawa TM 2008-261
December 2008

Principal Author

Original signed by Michael Low

Michael Low

Defence Scientist, Radar Electronic Warfare

Approved by

Original signed by Jean-F. Rivest

Jean-F. Rivest

Head, Radar Electronic Warfare

Approved for release by

Original signed by Pierre Lavoie

Pierre Lavoie

Chief Scientist, DRDC Ottawa

Abstract

Certain mathematical functions, such as the inverse, log, and arctangent, have traditionally been implemented in the digital domain using the Coordinate Rotation Digital Computer (CORDIC) algorithm. In this study, it is shown that it is possible to achieve a similar degree of numerical accuracy using a reduced-complexity lookup table (RCLT) algorithm, which is much less computationally-intensive than the CORDIC. On programmable digital signal processor (DSP) chips, this reduces computation time. On field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs), this reduces the consumption of hardware resources.

This paper presents the results of a study in which three commonly-used functions, i.e. inverse, log, and arctangent functions, have been synthesized for the Xilinx Virtex-II family of FPGAs, using both the RCLT and CORDIC implementations. Each implementation is compared for numerical accuracy of the output, the amount of hardware resources consumed, and throughput latency. In each case, it is shown that an RCLT implementation can be designed to achieve similar mathematical accuracy to the corresponding CORDIC implementation while consuming significantly less programmable logic and with reduced computational delay.

Résumé

Certaines fonctions mathématiques, comme logarithme, réciproque (inverse) et tangente inverse (arc Tg), sont habituellement calculées dans le domaine numérique au moyen de l'algorithme CORDIC (calculateur numérique de rotation de coordonnées). La présente étude montre qu'il est possible d'obtenir un degré comparable de précision numérique en utilisant un algorithme de recherche dans une table de recherche à complexité réduite (RCLT), qui exige beaucoup moins de puissance de calcul que CORDIC. Dans les processeurs de signal programmables (DSP), cela réduit le temps de calcul. Dans les matrices logiques programmables par l'utilisateur (FPGA) et les circuits intégrés à application spécifique (ASIC), cela réduit la consommation de ressources matérielles.

Ce document présente les résultats d'une étude au cours de laquelle trois fonctions largement utilisées, logarithme, réciproque et tangente inverse, ont été synthétisées pour la famille de FPGA Xilinx Virtex-II, en mettant en oeuvre les algorithmes RCLT et CORDIC. Chaque mise en oeuvre est comparée sur le plan de la précision mathématique des résultats, de la quantité de ressources matérielles utilisées et du temps d'attente du résultat. Dans chaque cas, on a démontré qu'une mise en oeuvre de RCLT peut être conçue de manière à obtenir une précision mathématique semblable à celle des mises en oeuvre correspondantes de CORDIC, tout en réduisant considérablement la consommation de logique programmable et en réduisant les retards liés aux calculs.

This page intentionally left blank.

Executive summary

A reduced-complexity lookup table approach to solving mathematical functions in FPGAs

Michael Low; Jim P. Y. Lee; DRDC Ottawa TM 2008-261; Defence R&D Canada – Ottawa; December 2008.

Background: Certain common mathematical functions, such as the inverse, log, and arctangent, have traditionally been implemented in the digital domain using the Coordinate Rotation Digital Computer (CORDIC) algorithm. This implementation can be quite complex and consume a large amount of hardware or software resources, depending on the required accuracy of the output.

Results: This study shows that it is possible to achieve a similar degree of numerical accuracy using a reduced-complexity lookup table (RCLT) algorithm, which is much less computationally-intensive than the CORDIC. On programmable digital signal processor (DSP) chips, this reduces computation time. On field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs), this reduces the consumption of hardware resources and reduces computation time in some cases.

This paper presents the results of a study in which the inverse, log, and arctangent functions have been synthesized for the Xilinx Virtex-II family of FPGAs, using both the RCLT and CORDIC implementations. Each implementation is compared for numerical accuracy of the output, the amount of hardware resources consumed, and throughput latency. In each case, it is shown that an RCLT implementation can be designed to achieve similar mathematical accuracy to the corresponding CORDIC implementation while consuming significantly less programmable logic and with reduced computational delay.

Significance: In future work to be performed by the Electronic Surveillance group of the Radar Electronic Warfare section at DRDC Ottawa, certain radar signal processing algorithms will be migrated from the software domain to the FPGA hardware domain in order to reduce computation time and achieve near-real-time processing. These algorithms are related to the task of specific emitter identification (SEI) and the detection of low probability of intercept (LPI) radar signals. The RCLT algorithm presented in this report will be used in the FPGA implementation since it consumes fewer resources than the CORDIC algorithm in cases of practical interest.

Sommaire

A reduced-complexity lookup table approach to solving mathematical functions in FPGAs

Michael Low; Jim P. Y. Lee; DRDC Ottawa TM 2008-261; R & D pour la défense Canada – Ottawa; décembre 2008.

Contexte : Certaines fonctions mathématiques communes comme logarithme, réciproque (inverse) et tangente inverse (arc Tg) sont habituellement réalisées dans le domaine numérique au moyen de l'algorithme CORDIC (calculateur numérique de rotation de coordonnées). Une telle mise en oeuvre peut être très complexe et elle peut consommer de grandes quantités de ressources matérielles et logicielles, selon le degré de précision requis de la sortie.

Résultats : La présente étude montre qu'il est possible d'obtenir un degré comparable de précision numérique en utilisant un algorithme de recherche dans une table de recherche à complexité réduite (RCLT), qui exige beaucoup moins de puissance de calcul que CORDIC. Dans les processeurs de signal programmables (DSP), cela réduit le temps de calcul. Dans les matrices logiques programmables par l'utilisateur (FPGA) et les circuits intégrés à application spécifique (ASIC), cela réduit la consommation de ressources matérielles et, parfois, le temps de calcul.

Ce document présente les résultats d'une étude au cours de laquelle les fonctions logarithme, réciproque et tangente inverse ont été synthétisées pour la famille de FPGA Xilinx Virtex-II, en mettant en oeuvre les algorithmes RCLT et CORDIC. Chaque mise en oeuvre a été comparée sur le plan de la précision mathématique des résultats, de la quantité de ressources matérielles utilisées et du temps d'attente du résultat. Dans chaque cas, on a démontré qu'une mise en oeuvre de RCLT peut être conçue de manière à obtenir une précision mathématique semblable à celle des mises en oeuvre correspondantes de CORDIC, tout en réduisant considérablement la consommation de logique programmable et en réduisant les retards liés aux calculs.

Importance : Au cours de travaux futurs qui seront effectués par le Groupe de surveillance électronique (GSE) de la Section de la guerre électronique par radar de R & D pour la défense Canada – Ottawa, à Ottawa, on fera migrer les algorithmes de traitement du signal radar du domaine logiciel au domaine matériel, en les intégrant en circuits FPGA, afin de réduire les temps de calcul et de réaliser le traitement du signal en temps quasi-réel. Ces algorithmes sont liés à la tâche précise d'identification d'émetteurs spécifiques (SEI) et de détection de signaux radar à faible probabilité d'interception (LPI). L'algorithme RCLT présenté dans ce rapport sera utilisé pour l'intégration en FPGA, car il consomme moins de ressources que l'algorithme CORDIC dans les cas d'intérêt pratique.

Table of contents

List of tables

1 Introduction

Certain common mathematical functions, such as the inverse, log, and arctangent, have traditionally been implemented in the digital domain using the Coordinate Rotation Digital Computer (CORDIC) algorithm (see [1], [2], and [3]). This implementation can be quite complex and consume a large amount of hardware or software resources, depending on the required accuracy of the output.

Lookup table-based methods have also been proposed as an approach to solving such functions (see [4] and [5]). However, in these cases, it is felt that the proposed solution in unnecessarily complicated by resorting to storing in the lookup table the results of various polynomial approximations to the desired function.

In this study, it is shown that it is possible to achieve comparable numerical accuracy using a reduced-complexity lookup table (RCLT) algorithm, which is introduced here. It is much less computationally-intensive than the CORDIC, and much simpler to construct compared to previous lookup table-based approaches. On programmable digital signal processor (DSP) chips, this reduces computation time. On field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs), this reduces the consumption of hardware resources.

This paper presents the results of a study in which the inverse, log, and arctangent functions have been synthesized using both the RCLT and CORDIC implementations for the Xilinx Virtex-II family of FPGAs. Each implementation will be compared for numerical accuracy of the output, the amount of hardware resources consumed, and throughput latency.

This paper is organized as follows:

Section 2 describes the theory behind the CORDIC algorithm and the RCLT algorithm.

Section 3 summarizes the performance of each algorithm in terms of numerical accuracy, and shows the corresponding size of each implementation in FPGA hardware.

Section 4 reviews the conclusions.

2 Algorithms

This section presents an overview of the CORDIC algorithm before examining the development of the RCLT algorithm. Since the focus is on implementation in a fixed-point environment, it is helpful to define some convenient terminology. The notation Fix_*M*_*N* is used to denote an *M*-bit signed number of which there are *N* fractional bits. For an unsigned number, the prefix UFix is used instead of Fix. This notation is practical because it is the same as that used in Matlab Simulink System Generator.

In this paper, a number of fixed-point dimensions are of particular interest:

Fix_14_13 – The ADC on-board Nallatech's XtremeDSP Development Kit II, which was used to generate the results in section 3, is an Analog Devices AD6644 which operates at up to 65 MSPS and returns a 14-bit signed value. Note that this product, purchased from Nallatech around 2003, is now obsolete, but is a convenient platform on which to test small- to medium-sized designs.

Fix_18_17 – The hardware multipliers in the Xilinx Virtex-II family of FPGAs operate on signed 18-bit values. Operands of other lengths are accommodated either by sign-extending the value to 18 bits, or by chaining multiple hardware multipliers together to accommodate shorter and longer operands, respectively.

Fix_24_23 – This value is arbitrarily chosen as representative of a fixed-point number with a high degree of precision. Note that $2^{-23} = 1.192 \times 10^{-7}$.

The examples presented in section 3 are implemented for unsigned values only, so for practical purposes, the sign bit would be discarded, leaving UFix_13_13, UFix_17_17, and UFix_23_23, respectively, as the operand sizes.

2.1 CORDIC

The CORDIC is an algorithm which asymptotically converges to the final value as a result of successive iterations [3]. It can be used to solve a wide range of trigonometric functions and linear equations. An arbitrarily high degree of accuracy can be obtained by increasing the number of iterations and, if the computation is being performed in fixed-point arithmetic, the number of operand bits. One significant feature is that it can be implemented in such a way that multiplications are performed by a series of bit shifts and additions, thereby saving on computing resources. A more detailed treatment is beyond the scope of this paper, but a practical overview, with emphasis on FPGA implementation, is given in [1] and [2].

2.2 Reduced-Complexity Lookup Table (RCLT)

Suppose we have a UFix_*M*_*N* value x on which we would like to perform the function $f(x)$. In a pure lookup table method, this would require a table with 2^M entries. If the output is required to have K bits of accuracy, then the total amount of memory required is $K2^M$ bits. For example, suppose $f(x) = log(x)$ where x is a UFix_23_23 number, and the result is required to have 23 bits of precision. In this case, the lookup table would have to be almost 193 Mbits in size!

For practical purposes, however, the lookup table can be reduced in size by exploiting the properties of $f(x)$. For example it will be shown that if $f(x)$ is an exponentially increasing or decreasing function, a considerable reduction can be made. Further savings may be obtained by taking into account the number of output bits K, and the error, relative to the true value, that can be tolerated. These properties are best illustrated by example.

2.2.1 Inverse Function

Consider the case of computing the inverse of a UFix_23_23 number x. Some representative values are shown in Table 1. Note that all unmarked bit positions are treated as containing zero. With reference to rows 1 through 3, let us consider all values of x which truncate to the binary value $\hat{x}$ = 0.1111 after retaining only the first four MSBs. This range includes the numbers 0.1111 0000 0000 0000 0000 000 (decimal 0.9375) through 0.1111 1111 1111 1111 1111 111 (decimal $1\text{-}2^{-23}$), with the mean being $\bar{x}$ = 0.1111 1 (decimal 0.96875). If we approximate the inverse of this entire range with $1/\bar{x}$, then the maximum relative error is incurred when $1/x\big|_{x=0.1111\_0000\_0000\_0000\_0000\_000} = 1.066667$ is approximated by $1/\bar{x} = 1.032258$. In this case, the relative error is -29.8 dB. Note that relative error is defined as the ratio in dB of the absolute value of the error to the true value. If this degree of error is acceptable, then the inverse of all numbers between 0.9375 and $1\text{-}2^{-23}$ may be approximated by a single value, thus reducing 2^{20} lookup table values to just one value. Note that because the output value spans many orders of magnitude, it is more appropriate to use units of dB to compare the magnitude of the error relative to the output value. In the case of log(x) in section 2.2.2, the output value only spans one order of magnitude, so it is more important to examine the absolute value of the error in that case.

In a similar fashion, we can trace the error in the approximation of 1/x for various other values of $\hat{x}$. An exhaustive investigation shows the maximum relative error to be -24.6 dB. If this degree of approximation can be tolerated, then the lookup table, which originally containing 2^{24} entries, can be reduced to one containing just 23×2^3 entries.

However, there are still further savings to be gained when one examines some properties of the binary representation of $1/\bar{x}$ in detail. Table 2 shows a subset of 3-bit RCLT for a UFix_23_23 input value.

As an example, consider row 9, where $1/\bar{x}\big|_{\bar{x}=0.0111\_11} = 2.064516$. Now note that this value can be obtained by doubling the value in row 1, where $1/\bar{x}\big|_{\bar{x}=0.1111\_1} = 1.032258$.

A careful investigation of each of the values of $1/\bar{x}$ in rows 9 through 16 shows that each can be obtained by multiplying the corresponding value in rows 1 through 8 by 2^1. Similarly, the values in rows 17 through 24 can be obtained by multiplying the corresponding value in rows 1 through 8 by 2^{18}.

These relationships allow us to further reduce the already-reduced lookup table consisting of 23×2^3 entries, to a table consisting of just 2^3 entries. The output of the twice-reduced lookup table is then multiplied by the appropriate power of two to obtain the final value. Further note that multiplication by a power of two reduces to a simple bitwise shift to the left.

To generalize the main features of the RCLT for the inverse function:

For some UFix_N_N input value x, it is possible to design an L-bit RCLT with K bits of output accuracy.

The 2^L-entry lookup table consists of the inverses of the binary numbers $0.1X_{L-1}X_{L-2}\ldots X_1X_0 1$ where the $X_{L-1}X_{L-2}\ldots X_1X_0$ represent all 2^L possible bit combinations. The inverse of $0.1X_{L-1}X_{L-2}\ldots X_1X_0 1$ is stored as the representative "mean" value of the inverses of all the values from $0.1X_{L-1}X_{L-2}\ldots X_1X_0 0\ldots 0$ through $0.1X_{L-1}X_{L-2}\ldots X_1X_0 1\ldots 1$.

The output of the 2^L-entry lookup table must be left-shifted by a certain number of bits to obtain the final value. This is described in more detail below.

The RCLT for the inverse function operates as follows:

Treat the UFix_N_N input value x as being composed of the bits $b_1 b_2 b_3,\ldots,b_{\tilde{N}-1} b_{\tilde{N}} b_{\tilde{N}+1} b_{\tilde{N}+2},\ldots,b_{\tilde{N}+L-1} b_{\tilde{N}+L},\ldots,b_N$, where b_1 is the 2^{-1} bit value and b_N is the 2^{-N} bit value. All the b_1 through $b_{\tilde{N}-1}$ are 0 and so $b_{\tilde{N}}$ is the largest non-zero bit value.

From the input value x, extract bits $b_{\tilde{N}+1} b_{\tilde{N}+2} b_{\tilde{N}+L-1} b_{\tilde{N}+L}$ to form the address $X_{L-1}X_{L-2}\ldots X_1X_0$ into the lookup table. This address stores the value $1/(0.1X_{L-1}X_{L-2}\ldots X_1X_0 1)$. This value is then left-shifted by $\tilde{N}$-1 bits to deliver the final result.

In the extreme cases, $\tilde{N} > N - L + 1$. For example, x = 0.0000 0000 0000 0000 0000 010, where N = 23, L = 3, $\tilde{N}$ = 22 and the only two significant bits are "10". In this case, the address into the L-bit lookup table is formed by appending zeros to the significant bits until a valid L-bit address is obtained. In this example, one "0" is appended to "10" to obtain the address "100". It is expected that the relative error of the output will be greater than that which results when a full L bits are naturally available to form the lookup table address, but since these are unavoidable extreme cases for both the RCLT and CORDIC algorithms, they will not be considered further when calculating the accuracy of a particular algorithm.

Table 1: Examples of approximations to 1/x where x is a UFix_23_23 value.

Row	x as a sum of powers of 2																							x	1/x	Relative error of approximation to 1/x in dB
	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23			
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.0000E+00	1.000000	-29.8
2	1	1	1	1	1																			9.6875E-01	1.032258	
3	1	1	1	1	0																			9.3750E-01	1.066667	-29.8
4	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5.6250E-01	1.777778	-24.6
5	1	0	0	0	1																			5.3125E-01	1.882353	
6	1	0	0	0	0																			5.0000E-01	2.000000	-24.6
7																										
8		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5.0000E-01	2.000000	-29.8
9		1	1	1	1	1																		4.8438E-01	2.064516	
10		1	1	1	1	0																		4.6875E-01	2.133333	-29.8
11		1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.8125E-01	3.555557	-24.6
12		1	0	0	0	1																		2.6563E-01	3.764706	
13		1	0	0	0	0																		2.5000E-01	4.000000	-24.6
14																										
15			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.5000E-01	4.000002	-29.8
16			1	1	1	1	1																	2.4219E-01	4.129032	
17			1	1	1	1	0																	2.3438E-01	4.266667	-29.8
18			1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.4062E-01	7.111117	-24.6
19			1	0	0	0	1																	1.3281E-01	7.529412	
20			1	0	0	0	0																	1.2500E-01	8.000000	-24.6
21																										
22																			1	1	1	1	1	3.6955E-06	270600.258065	
23																			1	1	1	1	0	3.5763E-06	279620.266667	-29.8
24																			1	0	0	0	1	2.0266E-06	493447.529412	
25																			1	0	0	0	0	1.9073E-06	524288.000000	-24.6

Table 2: Examples of a subset of a 3-bit RCLT for 1/x.

Row	x as a sum of powers of 2																							x	1/x
	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23		
1	1	1	1	1	1																			9.6875E-01	1.032258
2	1	1	1	0	1																			9.0625E-01	1.103448
3	1	1	0	1	1																			8.4375E-01	1.185185
4	1	1	0	0	1																			7.8125E-01	1.280000
5	1	0	1	1	1																			7.1875E-01	1.391304
6	1	0	1	0	1																			6.5625E-01	1.523810
7	1	0	0	1	1																			5.9375E-01	1.684211
8	1	0	0	0	1																			5.3125E-01	1.882353
9		1	1	1	1	1																		4.8438E-01	2.064516
10		1	1	1	0	1																		4.5313E-01	2.206897
11		1	1	0	1	1																		4.2188E-01	2.370370
12		1	1	0	0	1																		3.9063E-01	2.560000
13		1	0	1	1	1																		3.5938E-01	2.782609
14		1	0	1	0	1																		3.2813E-01	3.047619
15		1	0	0	1	1																		2.9688E-01	3.368421
16		1	0	0	0	1																		2.6563E-01	3.764706
17																				1	1	1	1	1.7881E-06	559240.533333
18																				1	1	1	0	1.6689E-06	599186.285714
19																				1	1	0	1	1.5497E-06	645277.538462
20																				1	1	0	0	1.4305E-06	699050.666667
21																				1	0	1	1	1.3113E-06	762600.727273
22																				1	0	1	0	1.1921E-06	838860.800000
23																				1	0	0	1	1.0729E-06	932067.555556
24																				1	0	0	0	9.5367E-07	1048576.000000

2.2.2 log Function

Consider the case of computing the natural log of *x*, which is a UFix_23_23 number. Some representative values are shown in Table 3. Note that all unmarked bit positions are treated as containing zero. With reference to rows 4 through 6, let us consider all values of *x* which truncate to the binary value $\hat{x}$ = 0.1000 after retaining only the first four MSBs. This range includes the numbers 0.1000 0000 0000 0000 0000 000 (decimal 0.5) through 0.1000 1111 1111 1111 1111 111 (decimal 0.5625-2^{-23}), with the mean being $\bar{x}$ = 0.1000 1 (decimal 0.53125). If we approximate the log of this entire range with $\log(\bar{x})$, then the maximum error is incurred when $\log(x)\big|_{x=0.1111\_1111\_1111\_1111\_1111\_111} = -0.575364$ is approximated by $\log(\bar{x}) = -0.632523$. In this case, the absolute value of the error is 5.7e-2, and the relative error is -20.1 dB. If this degree of error is tolerable, then the log of all UFix_23_23 numbers between 0.5 and 0.5625-2^{-23} may be approximated by a single value, thus reducing 2^{20} lookup table values to just one value.

A few words must be said about the exceptional cases on lines 1 through 3 where $\bar{x}$ = 0.1111 1 (decimal 0.96875) is used to approximate numbers ranging from 0.1111 0000 0000 0000 0000 000 (decimal 0.9375) through 0.1111 1111 1111 1111 1111 111 (decimal 1-2^{-23}). The absolute value of the error is 3.3e-2, which is in the same order of magnitude as absolute errors for other ranges in Table 3; however the relative error is extremely high because the true value is very close to zero. Note that the CORDIC log algorithm suffers from a similar degree of numerical error for *x* values close to unity, as is shown in section 3.

Continuing with the previous discussion, we can trace the error in the approximation of log(*x*) for $\hat{x}$ = 0.0111 1 on rows 8 through 10, and log(*x*) for $\hat{x}$ = 0.0100 0 on rows 11 through 13, and so on. The error is highest on the approximation of 0.1000, and on all single-bit right-shifts of this number down to 0.0000 0000 0000 0000 0001 000. If the maximum absolute error in all cases is tolerable, then the lookup table, which originally contained 2^{24} entries, can be reduced to one containing just 23×2^3 entries.

However, there are still further savings to be gained when one examines some properties of the binary representation of $\log(\bar{x})$ in detail. Table 4 shows a subset of the 3-bit RCLT for a UFix_23_23 input value. It is considered to be a 3-bit table because the three bits following the first non-zero MSB, shown shaded in grey, are of significance.

As an example, consider row 9, where $\log(\bar{x})\big|_{\bar{x}=0.0111\_11} = -0.724896$. Now note the following relationship with $\log(\bar{x})\big|_{\bar{x}=0.1111\_1} = -0.031749$ in row 1:

$$\begin{aligned} \log(2^{-1}) + \log(\bar{x})\big|_{\bar{x}=0.1111\_1} &= -0.693147 - 0.031749 \\ &= -0.724896 \\ &= \log(\bar{x})\big|_{\bar{x}=0.0111\_11} \end{aligned}$$

A careful investigation of $\log(\bar{x})$ in rows 9 through 16 shows that each value can be expressed as the sum of $\log(2^{-1})$ and the corresponding value in rows 1 through 8. Similarly, the values in rows 17 through 24 can be obtained by adding $\log(2^{-18})$ to the corresponding value in rows 1 through 8.

These relationships allow us to further reduce the already-reduced lookup table consisting of 23×2^3 entries, to two lookup tables, where the first contains 2^3 entries, such as the example shown in Table 4, and the second contains 23 entries, namely the values of $\log(2^{-1})$ through $\log(2^{-23})$.

To generalize the main features of the RCLT for the log function:

> When the input x is a UFix_N_N value, and it is desired to create an RCLT with K bits of output accuracy, two lookup tables are required. The first is an L-bit lookup table containing 2^L entries of K bits each. The second table contains N entries of K bits each.
>
> The 2^L-entry lookup table consists of the logs of the binary numbers $0.1X_{L-1}X_{L-2}\ldots X_1X_0 1$ where the $X_{L-1}X_{L-2}\ldots X_1X_0$ represent all 2^L possible bit combinations. The log of $0.1X_{L-1}X_{L-2}\ldots X_1X_0 1$ is stored as the representative "mean" value of the logs of all the values from $0.1X_{L-1}X_{L-2}\ldots X_1X_0 0\ldots0$ through $0.1X_{L-1}X_{L-2}\ldots X_1X_0 1\ldots1$.
>
> The N-entry lookup table consists of all the values of $\log(2^{-1})$ through $\log(2^{-N})$.

The RCLT for the log function operates as follows:

> Treat the UFix_N_N input value x as being composed of the bits $b_1b_2b_3,\ldots,b_{\tilde{N}-1}b_{\tilde{N}}b_{\tilde{N}+1}b_{\tilde{N}+2},\ldots,b_{\tilde{N}+L-1}b_{\tilde{N}+L},\ldots,b_N$, where b_1 is the 2^{-1} bit value and b_N is the 2^{-N} bit value. All the b_1 through $b_{\tilde{N}-1}$ are 0 and so $b_{\tilde{N}}$ is the largest non-zero bit value.
>
> From the input value x, extract bits $b_{\tilde{N}+1}b_{\tilde{N}+2}b_{\tilde{N}+L-1}b_{\tilde{N}+L}$ to form the address $X_{L-1}X_{L-2}\ldots X_1X_0$ into the lookup table. This address stores the value $\log(0.1X_{L-1}X_{L-2}\ldots X_1X_0 1)$. This value is then added to the value $\log(\tilde{N}\text{-}1)$ to deliver the final result.
>
> In the extreme cases, $\tilde{N} > N - L + 1$. For example, x = 0.0000 0000 0000 0000 0000 010, where N = 23, L = 3, $\tilde{N}$ = 22 and the only two significant bits are "10". In this case, the address into the L-bit lookup table is formed by appending zeros to the significant bits until a valid L-bit address is obtained. In this example, one "0" is appended to "10" to obtain the address "100". It is expected that the relative error of the output will be greater than that which results when a full L bits are naturally available to form the lookup table address, but since these are unavoidable extreme cases for both the RCLT and CORDIC algorithms, they will not be considered further when calculating the accuracy of a particular algorithm.

Table 3: Examples of approximations to log(x), where x is a UFix_23_23 value.

Row	x as a sum of powers of 2																							x	log(x)	Relative error of approxi-mation to log(x)	Absolute error of approxi-mation to log(x)
	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23				
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.0000E+00	0.000000	108.5	3.2E-02
2	1	1	1	1	1																			9.6875E-01	-0.031749		
3	1	1	1	1	0																			9.3750E-01	-0.064539	-5.9	3.3E-02
4	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5.6250E-01	-0.575364	-20.1	5.7E-02
5	1	0	0	0	1																			5.3125E-01	-0.632523		
6	1	0	0	0	0																			5.0000E-01	-0.693147	-21.2	6.1E-02
7																											
8		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	5.0000E-01	-0.693147	-26.8	3.2E-02
9		1	1	1	1	1																		4.8438E-01	-0.724896		
10		1	1	1	1	0																		4.6875E-01	-0.757686	-27.3	3.3E-02
11		1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.8125E-01	-1.268512	-26.9	5.7E-02
12		1	0	0	0	1																		2.6563E-01	-1.325670		
13		1	0	0	0	0																		2.5000E-01	-1.386294	-27.2	6.1E-02
14																											
15			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.5000E-01	-1.386295	-32.8	3.2E-02
16			1	1	1	1	1																	2.4219E-01	-1.418043		
17			1	1	1	1	0																	2.3438E-01	-1.450833	-32.9	3.3E-02
18			1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.4062E-01	-1.961659	-30.7	5.7E-02
19			1	0	0	0	1																	1.3281E-01	-2.018817		
20			1	0	0	0	0																	1.2500E-01	-2.079442	-30.7	6.1E-02
21																											
22																			1	1	1	1	1	3.6955E-06	-12.508398		
23																			1	1	1	1	0	3.5763E-06	-12.541188	-51.7	3.3E-02
24																			1	0	0	0	1	2.0266E-06	-13.109172		
25																			1	0	0	0	0	1.9073E-06	-13.169796	-46.7	6.1E-02

Table 4: Examples of a subset of a 3-bit RCLT for log(x).

Row	x as a sum of powers of 2																							x	log(x)
	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23		
1	1	1	1	1	1																			9.6875E-01	-0.031749
2	1	1	1	0	1																			9.0625E-01	-0.098440
3	1	1	0	1	1																			8.4375E-01	-0.169899
4	1	1	0	0	1																			7.8125E-01	-0.246860
5	1	0	1	1	1																			7.1875E-01	-0.330242
6	1	0	1	0	1																			6.5625E-01	-0.421213
7	1	0	0	1	1																			5.9375E-01	-0.521297
8	1	0	0	0	1																			5.3125E-01	-0.632523
9		1	1	1	1	1																		4.8438E-01	-0.724896
10		1	1	1	0	1																		4.5313E-01	-0.791587
11		1	1	0	1	1																		4.2188E-01	-0.863046
12		1	1	0	0	1																		3.9063E-01	-0.940007
13		1	0	1	1	1																		3.5938E-01	-1.023389
14		1	0	1	0	1																		3.2813E-01	-1.114361
15		1	0	0	1	1																		2.9688E-01	-1.214444
16		1	0	0	0	1																		2.6563E-01	-1.325670
17																				1	1	1	1	1.7881E-06	-13.234335
18																				1	1	1	0	1.6689E-06	-13.303328
19																				1	1	0	1	1.5497E-06	-13.377436
20																				1	1	0	0	1.4305E-06	-13.457479
21																				1	0	1	1	1.3113E-06	-13.544490
22																				1	0	1	0	1.1921E-06	-13.639800
23																				1	0	0	1	1.0729E-06	-13.745161
24																				1	0	0	0	9.5367E-07	-13.862944

2.2.3 arctan Function

Consider the case of computing the arctangent of x, which is a UFix_23_15 number. Note that, unlike in the previous examples, there are a eight integer bits to accommodate input values of interest that are greater than one. Some representative values are shown in Table 5. Note that all unmarked bit positions are treated as containing zero. First examine rows 1 through 3. Let us consider all values of x which truncate to the binary value $\hat{x}$ = 1111 0000.0 after retaining only the first four MSBs. This range includes the numbers 1111 0000.0000 0000 0000 000 (decimal 240) through 1111 1111.1111 1111 1111 111 (approximately decimal 256), with the mean being $\bar{x}$ = 1111 1000.0 (decimal 248). If we approximate the arctangent of this entire range with $\arctan(\bar{x})$, then the maximum error is incurred when $\arctan(x)\big|_{x=0.1111\_0000.0000\_0000\_0000\_000} = 1.566630$ is approximated by $\arctan(\bar{x}) = 1.566764$. In this case, the error, relative to the true value, has a magnitude of -81.3 dB. If this degree of error is acceptable, then the arctangent of all numbers between 240 and 256 may be approximated by a single value, thus reducing 2^{20} lookup table values to just one value.

In a similar fashion, we can trace the error in the approximation of arctan(x) for various other values of $\hat{x}$. An exhaustive investigation shows the maximum error to be -24.1 dB. If this degree of approximation can be tolerated, then the lookup table, which originally containing 2^{24} entries, can be reduced to one containing just 23×2^3 entries.

Unlike the case for log(x) and 1/x, there is no convenient relationship which exists for all x which allows for the RCLT to be further reduced. It is worth noting that arctan(x) ≈ x for small values of x, however this idea is not explored further here in order to demonstrate that significant savings can still be achieved.

To generalize the main features of the RCLT for the arctangent function:

> For some UFix_M_N input value x, it is possible to create an L-bit RCLT containing $M2^L$ entries, each with K bits of output accuracy.
>
> The first 2^L rows of the RCLT contain of the arctangent of all possible binary numbers $0.1X_{L-1}X_{L-2}\ldots X_1X_01$. The arctangent of $0.1X_{L-1}X_{L-2}\ldots X_1X_01$ is stored as the representative "mean" value of the arctangent of all the values $0.1X_{L-1}X_{L-2}\ldots X_1X_00\ldots0$ through $0.1X_{L-1}X_{L-2}\ldots X_1X_01\ldots1$. The next set of 2^L rows contains the arctangent of $0.01X_{L-1}X_{L-2}\ldots X_1X_01$, and so on for a total of N sets of 2^L entries.

The RCLT for the arctangent function is used as follows:

> Treat the UFix_M_N input value x as being composed of the bits $b_1b_2b_3,\ldots,b_{\tilde{N}-1}b_{\tilde{N}}b_{\tilde{N}+1}b_{\tilde{N}+2},\ldots,b_{\tilde{N}+L-1}b_{\tilde{N}+L},\ldots,b_N$, where b_1 is the $2^{M-\tilde{N}-1}$ bit value and b_N is the $2^{-(M-N)}$ bit value. All the b_1 through $b_{\tilde{N}-1}$ are 0 and so $b_{\tilde{N}}$ is the largest non-zero bit value.
>
> From the input value x, extract bits $b_{\tilde{N}+1}b_{\tilde{N}+2}b_{\tilde{N}+L-1}b_{\tilde{N}+L}$ to form the address $X_{L-1}X_{L-2}\ldots X_1X_0$ into the lookup table. This address stores the value arctan($0.0,\ldots,01b_{\tilde{N}+1}b_{\tilde{N}+2}b_{\tilde{N}+L-1}b_{\tilde{N}+L}1$), which is the final result.

In the extreme cases, $\tilde{N} > N - L + 1$. For example, x = 0.0000 0000 0000 0000 0000 010, where N = 23, L = 3, $\tilde{N}$ = 22 and the only two significant bits are "10". In this case, the address into the L-bit lookup table is formed by appending zeros to the significant bits until a valid L-bit address is obtained. In this example, one "0" is appended to "10" to obtain the address "100". It is expected that the relative error of the output will be greater than that which results when a full L bits are naturally available to form the lookup table address, but since these are unavoidable extreme cases for both the RCLT and CORDIC algorithms, they will not be considered further when calculating the accuracy of a particular algorithm.

Table 5: Examples of approximations to arctan(x) where x is a UFix_23_15 value.

Row	x as a sum of powers of 2																							x	arctan(x)	Absolute error of approxi-mation to arctan(x)	Relative error of approxi-mation to arctan(x) in dB
	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15				
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.5600E+02	1.566890	1.3E-04	-81.9
2	1	1	1	1	1																			2.4800E+02	1.566764		
3	1	1	1	1	0																			2.4000E+02	1.566630	1.3E-04	-81.3
4	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.4400E+02	1.563852	4.1E-04	-71.7
5	1	0	0	0	1																			1.3600E+02	1.563444		
6	1	0	0	0	0																			1.2800E+02	1.562984	4.6E-04	-70.6
7																											
8		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1.2800E+02	1.562984	2.5E-04	-75.9
9		1	1	1	1	1																		1.2400E+02	1.562732		
10		1	1	1	1	0																		1.2000E+02	1.562463	2.7E-04	-75.3
11		1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7.2000E+01	1.556908	8.2E-04	-65.6
12		1	0	0	0	1																		6.8000E+01	1.556092		
13		1	0	0	0	0																		6.4000E+01	1.555173	9.2E-04	-64.6
14																											
15								1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2.0000E+00	1.107143	1.3E-02	-38.7
16								1	1	1	1	1												1.9375E+00	1.094329		
17								1	1	1	1	0												1.8750E+00	1.080839	1.3E-02	-38.1
18								1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1.1250E+00	0.844141	2.8E-02	-29.4
19								1	0	0	0	1												1.0625E+00	0.815692		
20								1	0	0	0	0												1.0000E+00	0.785398	3.0E-02	-28.3
21																											
22																			1	1	1	1	1	9.4604E-04	0.000946		
23																			1	1	1	1	0	9.1553E-04	0.000916	3.1E-05	-29.5
24																			1	0	0	0	1	5.1880E-04	0.000519		
25																			1	0	0	0	0	4.8828E-04	0.000488	3.1E-05	-24.1

Table 6: Examples of a subset of a 3-bit RCLT for arctan(x).

Row	x as a sum of powers of 2																							x	arctan(x)
	7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15		
1	1	1	1	1	1																			2.4800E+02	1.566764
2	1	1	1	0	1																			2.3200E+02	1.566486
3	1	1	0	1	1																			2.1600E+02	1.566167
4	1	1	0	0	1																			2.0000E+02	1.565796
5	1	0	1	1	1																			1.8400E+02	1.565362
6	1	0	1	0	1																			1.6800E+02	1.564844
7	1	0	0	1	1																			1.5200E+02	1.564217
8	1	0	0	0	1																			1.3600E+02	1.563444
9		1	1	1	1	1																		1.2400E+02	1.562732
10		1	1	1	0	1																		1.1600E+02	1.562176
11		1	1	0	1	1																		1.0800E+02	1.561537
12		1	1	0	0	1																		1.0000E+02	1.560797
13		1	0	1	1	1																		9.2000E+01	1.559927
14		1	0	1	0	1																		8.4000E+01	1.558892
15		1	0	0	1	1																		7.6000E+01	1.557639
16		1	0	0	0	1																		6.8000E+01	1.556092
17																				1	1	1	1	4.5776E-04	0.000458
18																				1	1	1	0	4.2725E-04	0.000427
19																				1	1	0	1	3.9673E-04	0.000397
20																				1	1	0	0	3.6621E-04	0.000366
21																				1	0	1	1	3.3569E-04	0.000336
22																				1	0	1	0	3.0518E-04	0.000305
23																				1	0	0	1	2.7466E-04	0.000275
24																				1	0	0	0	2.4414E-04	0.000244

This page intentionally left blank.

3 Results

This section presents the results obtained from comparing the RCLT and CORDIC implementations of the inverse, log, and arctangent functions. As explained in section 2, operand sizes of practical interest are UFix_13_13, UFix_17_17, and UFix_23_23. Therefore, the RCLT algorithms presented in this section were implemented using each of these input operand sizes.

The CORDIC algorithms, against which the RCLT implementations are compared, are pre-packaged cores taken from the Math library of the Xilinx Reference Blockset, which is a Matlab Simulink blockset available from Xilinx [7]. The number of CORDIC processing elements is programmable, which permits control over the accuracy of the final result; that is, the greater the number of elements, the greater the output accuracy. Note, however, that there is a practical limit to the degree of output accuracy that can be achieved by a fixed-point CORDIC. This is because the use of fixed-point numbers in the calculations creates a numerical granularity that, eventually, cannot be overcome by simply performing more iterations.

In choosing a core against which to perform a fair comparison with the RCLT version of the inverse and arctangent functions, the number of CORDIC elements was chosen such that both implementations achieved similar relative accuracy of the output value. The relative error of a fixed-point algorithm is defined here as the ratio in dB of the absolute value of the error divided by the true value. The true value is taken to be the result of the corresponding floating-point calculation, and the error is taken to be the true value minus the output of the fixed-point algorithm.

In the case of the log function, the absolute value of the error achieved by the RCLT and CORDIC implementations was considered instead of the relative error. Because some true values are close to zero, division by the true value to obtain the relative error results in some apparently large relative errors which do not reflect the actual usability of the fixed-point result. As discussed in section 2.2.2, both algorithms suffer from this phenomenon.

Both the RCLT and CORDIC algorithms were implemented for the Xilinx Virtex-II series of FPGAs. The fundamental building block of programmable logic in this FPGA family is the "slice". One slice consists primarily of two flip-flop (FF) registers and two 4-input function generators, as well as some glue logic. Each function generator can be configured as either a 4-input lookup table (LUT), a 16-bit configurable RAM, or a 16-bit shift register. It is from the slice that more complex functions, such as adders or larger-sized LUTs, can be realized. In addition to slices, there are dedicated segments of block RAM (BRAM) of 18 kbits each. It is often more efficient to construct large-size LUTs from BRAM instead of slices. Finally, the Virtex-II FPGAs contain a limited number of dedicated hardware multipliers. It is more resource-efficient to use an embedded multiplier than to build it out of slices. Table 7 provides a comparison of available resources for some selected devices in the Virtex-II family. For more details, the reader is referred to the manufacturer's documentation [6].

The amount of post-synthesis resources consumed by a particular implementation was obtained by using the Resource Estimator block, which comes from the Tools library of the Xilinx blockset. This blockset is available as part of System Generator from Xilinx, and runs as a plug-in under Matlab Simulink.

Table 7: Summary of resources available in selected Xilinx Virtex-II FPGAs.

Device	Slices	Embedded multipliers	BRAM blocks (18 kbits each)	Max I/O pads
XC2V40 (smallest)	256	4	4	88
XC2V1000	5120	40	40	432
XC2V3000 (used here)	14336	96	96	720
XC2V6000	33792	144	144	1104
XC2V8000 (largest)	46592	168	168	1108

Table 8 compares the resources consumed by the RCLT and CORDIC algorithms, as implemented to perform the inverse function. The maximum relative error produced by the RCLT decreases noticeably as the number of table bits *L* increases. The number of output fraction bits was set to 16. The number of output integer bits was set sufficiently large to accommodate the largest meaningful output value. In the extreme case of a UFix_23_23 input dimension and a 4-bit RCLT, the output dimension was set to UFix_40_16. When the RCLT lookup table was built using one BRAM, the RCLT implementation consumes between 2.2 and 3.0 times fewer slices than the corresponding CORDIC to achieve similar output accuracy. When the lookup table was built using slices (i.e. "distributed RAM" in Xilinx terminology), the RCLT implementation consumes between 2.2 and 2.9 times fewer slices. Building the lookup table out of slices does not greatly increase the overall consumption of slices because of the small size of the lookup table in this case. The RCLT also does not use any of the embedded multipliers, of which between 1 and 4 are used by the CORDIC implementation. The throughput latency of the RCLT implementation is also noticeably lower than that of the CORDIC.

Table 9 compares the resources consumed by the RCLT and CORDIC algorithms, as implemented to perform the log function. The maximum absolute error produced by the RCLT decreases slightly as the number of table bits *L* increases. In all cases, for an input dimension of UFix_*N*_*N*, the output dimension was set to Fix_(*N*+5)_*N* to accommodate the range of meaningful output values. When the RCLT lookup tables were built using two BRAMs (one for each lookup table), the RCLT implementation consumes between 4.2 and 4.8 times fewer slices than the corresponding CORDIC to achieve similar output accuracy. When the lookup tables were built using slices, the RCLT implementation consumes between 3.5 and 4.1 times fewer slices. The throughput latency of the RCLT implementation is also noticeably lower than that of the CORDIC.

Table 10 compares the resources consumed by the RCLT and CORDIC algorithms, as implemented to perform the arctangent function. The maximum relative error produced by the RCLT decreases noticeably as the number of table bits *L* increases. In all cases, for an input value with dimension UFix_*M*_*N*, the dimension of the output value was set to UFix_(*N*+1)_*N*. Although the arctangent is defined for input values ranging from negative to positive infinity, this RCLT implementation was restricted to input values from zero to positive infinity; therefore the

output value ranges from zero to /2, and is a UFix instead of a Fix number. Note that extending the capability to handle negative-valued input would only have consumed a small amount of extra slices because it is simply a matter of assigning the sign of the input to the sign of the output. When the RCLT lookup table was built using one BRAM, the RCLT implementation consumes between 2.8 and 4.8 times fewer slices than the corresponding CORDIC to achieve similar output accuracy. When the lookup table was built using slices, the RCLT implementation consumes between 2.4 and 3.9 times fewer slices. Unlike for the other functions, the throughput latency of the RCLT implementation of the arctangent is noticeably higher than that of the CORDIC. This is because the RCLT latency for all functions is equal to the number of input bits, *N*, plus the internal lookup table latency, whereas the CORDIC latency tends to be dependent on the complexity of the iterations to be performed, which varies greatly from function to function. In the case of the arctangent, the iterations are not very complex compared to those required for the inverse and log.

Table 8: Comparison of resource consumption by RCLT and CORDIC for inverse function.

Input dimension	RCLT								CORDIC								Relative size of CORDIC vs. RCLT
	Table size (bits)	Through-put latency (clock cycles)	Slices	FF	BRAM	LUT	Em-bedded Multi-pliers	Max. rela-tive error (dB)	Num-ber of ele-ments	Through-put latency (clock cycles)	Slices	FF	BRAM	LUT	Em-bedded Multi-pliers	Max. rela-tive error (dB)	
UFix_13_13	2	16	188 196	110 125	1 0	349 362	0 0	-19.1	4	22	570	538	0	735	1	-18.4	3.0 2.9
	3	16	198 206	112 128	1 0	367 383	0 0	-24.6	5	23	592	552	0	761	1	-24.4	3.0 2.9
	4	16	207 216	114 131	1 0	385 401	0 0	-30.4	6	24	611	566	0	789	1	-30.4	3.0 2.8
UFix_17_17	2	20	392 400	156 171	1 0	733 746	0 0	-19.1	4	26	896	910	0	1176	4	-18.4	2.3 2.2
	3	20	410 418	158 174	1 0	769 785	0 0	-24.6	5	27	922	928	0	1206	4	-24.4	2.2 2.2
	4	20	428 437	160 177	1 0	805 821	0 0	-30.4	6	28	945	946	0	1238	4	-30.4	2.2 2.2
UFix_23_23	2	26	487 495	215 230	1 0	892 905	0 0	-19.1	4	32	1406	1498	0	1927	4	-18.4	2.9 2.8
	3	26	505 513	217 233	1 0	928 944	0 0	-24.6	5	33	1438	1522	0	1963	4	-24.4	2.8 2.8
	4	26	523 532	219 236	1 0	964 980	0 0	-30.4	6	34	1467	1546	0	2001	4	-30.4	2.8 2.8

Table 9: Comparison of resource consumption by RCLT and CORDIC for log function.

Input dimension	RCLT								CORDIC								Relative size of CORDIC vs. RCLT
	Table size (bits)	Through-put latency (clock cycles)	Slices	FF	BRAM	LUT	Em-bedded Multi-pliers	Max. abso-lute error	Num-ber of ele-ments	Through-put latency (clock cycles)	Slices	FF	BRAM	LUT	Em-bedded Multi-pliers	Max. abso-lute error	
UFix_13_13	3	16	107 125	162 198	2 0	120 156	0 0	6.1e-2	6	25	517	333	0	828	0	6.3e-2	4.8 4.1
	4	16	112 130	164 200	2 0	129 160	0 0	3.1e-2	7	26	544	333	0	882	0	3.2e-2	4.9 4.2
	5	16	118 140	166 202	2 0	138 181	0 0	1.6e-2	8	27	571	333	0	936	0	1.7e-2	4.8 4.1
UFix_17_17	3	20	162 190	221 265	2 0	192 248	0 0	6.1e-2	6	29	692	495	0	1110	0	6.4e-2	4.3 3.6
	4	20	171 199	223 267	2 0	210 261	0 0	3.1e-2	7	30	725	495	0	1176	0	3.3e-2	4.2 3.6
	5	20	181 215	225 269	2 0	228 295	0 0	1.6e-2	8	31	758	495	0	1242	0	1.8e-2	4.2 3.5
UFix_23_23	3	26	219 255	302 354	2 0	250 327	0 0	6.1e-2	6	35	996	795	0	1605	0	6.5e-2	4.5 3.9
	4	26	228 264	304 356	2 0	268 341	0 0	3.1e-2	7	36	1038	795	0	1689	0	3.5e-2	4.6 3.9
	5	26	238 283	306 358	2 0	286 381	0 0	1.6e-2	8	37	1080	795	0	1773	0	1.9e-2	4.5 3.8

Table 10: Comparison of resource consumption by RCLT and CORDIC for arctangent function.

Input dimension	RCLT									CORDIC									Relative size of CORDIC vs. RCLT
	Table size (bits)	Through-put latency (clock cycles)	Slices	FF	BRAM	LUT	Em-bedded Multi-pliers	Max. abso-lute error	Max. rela-tive error (dB)	Num-ber of ele-ments	Through-put latency (clock cycles)	Slices	FF	BRAM	LUT	Em-bedded Multi-pliers	Max. abso-lute error	Max. rela-tive error (dB)	
UFix_13_5	2	16	68 75	100 106	1 0	91 104	0 0	5.8e-2	-16.6	5	8	205	87	0	376	0	6.5e-2	-16.8	3.0 2.7
	3	16	73 85	100 106	1 0	97 121	0 0	3.6e-2	-22.1	5	8	205	87	0	376	0	6.5e-2	-20.2	2.8 2.4
	4	16	76 91	100 106	1 0	103 133	0 0	2.8e-2	-29.2	6	9	232	87	0	430	0	3.9e-2	-28.5	3.1 2.5
UFix_17_9	2	20	84 99	124 134	1 0	115 145	0 0	5.8e-2	-17.7	9	12	383	107	0	724	0	4.2e-3	-13.2	4.6 3.9
	3	20	89 117	124 134	1 0	121 177	0 0	3.1e-2	-23.3	9	12	383	107	0	724	0	4.5e-3	-19.2	4.3 3.3
	4	20	92 135	124 134	1 0	127 212	0 0	1.5e-2	-28.7	9	12	383	107	0	724	0	4.5e-3	-25.0	4.2 2.8
UFix_23_15	2	26	153 188	198 214	1 0	200 270	0 0	5.9e-2	-18.0	15	18	740	137	0	1426	0	6.6e-5	-16.5	4.8 3.9
	3	26	159 217	197 213	1 0	214 330	0 0	3.0e-2	-24.0	15	18	740	137	0	1426	0	6.8e-5	-21.6	4.7 3.4
	4	26	167 268	196 212	1 0	228 430	0 0	1.5e-2	-30.0	15	18	740	137	0	1426	0	6.9e-5	-27.1	4.4 2.8

This page intentionally left blank.

4 Conclusions

In this paper, the RCLT algorithm has been developed and proposed as an alternative to the CORDIC algorithm for performing certain numerical calculations on DSPs, FPGAs and ASICs. As practical examples, the inverse, log, and arctangent functions have been implemented using both algorithms for the Xilinx Virtex-II family of FPGAs. In each case, it has been shown that an RCLT algorithm can be designed to achieve similar numerical accuracy to the corresponding CORDIC algorithm while consuming less general-purpose programmable logic and, in many cases, with less throughput latency.

The RCLT approach is not limited to these particular functions, but can be applied to any function $y = f(x)$. In some cases, such as that of the inverse and log, it is possible to exploit the characteristics of the function to shrink the lookup table to a very small size. In other cases, such as that of the arctangent, such reductions are not possible, but a significant amount of resources can still be saved compared to the CORDIC approach.

References

[1] Andraka, R, "A survey of CORDIC algorithms for FPGA based computers," International Symposium on Field Programmable Gate Arrays, Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Monterey, California, United States, 1998, pp. 191 – 200.

[2] Vadlamani, S., Mahmoud, W., "Comparison of CORDIC Algorithm Implementations on FPGA families," Proceedings of the Thrity-Fourth Southeastern Symposium on System Theory, 2002, 18-19 March 2002, pp. 192 – 196.

[3] Volder, J. E., "The CORDIC Trigonometric Computing Technique," IRE Transactions on Electronic Computers, vol. 8, pp. 330 – 334, September 1959.

[4] Tang, P. T. P., "Table-lookup algorithms for elementary functions and their error analysis," in Proceedings of the 10th IEEE Symposium on Computer Arithmetic, June 1991, pp. 232 – 236.

[5] Seidner, D., "Efficient implementation of log10 lookup table in FPGA," IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems, 2008 (COMCAS 2008), 13-14 May 2008, pp. 1-9.

[6] Xilinx, "Virtex-II Platform FPGAs: Complete Data Sheet," DS031, v3.5, 5 November 2007.

[7] Xilinx, "System Generator for DSP: Reference Guide," Release 10.1, March 2008.

List of symbols/abbreviations/acronyms/initialisms

ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integration Circuit
BRAM	Block Random Access Memory
CORDIC	Coordinate Rotation Digital Computer
DRDC	Defence Research and Development Canada
DSP	Digital Signal Processor
FF	Flip-Flop
Fix	A signed, fixed-point number
FPGA	Field-Programmable Gate Array
I/O	Input / Output
LPI	Low Probability of Intercept
LUT	Look-Up Table
MSB	Most Significant Bit
MSPS	Million Samples Per Second
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
RCLT	Reduced-Complexity Lookup Table
SEI	Specific Emitter Identification
UFix	An unsigned, fixed-point number

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – Ottawa
3701 Carling Avenue
Ottawa, Ontario K1A 0Z4

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

A reduced-complexity lookup table approach to solving mathematical functions in FPGAs

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Low, M.; Lee, J. P. Y.

5. DATE OF PUBLICATION (Month and year of publication of document.)

December 2008

6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)

36

6b. NO. OF REFS (Total cited in document.)

7

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Ottawa
3701 Carling Avenue
Ottawa, Ontario K1A 0Z4

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

15df02

9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Ottawa TM 2008-261

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Certain mathematical functions, such as the inverse, log, and arctangent, have traditionally been implemented in the digital domain using the Coordinate Rotation Digital Computer (CORDIC) algorithm. In this study, it is shown that it is possible to achieve a similar degree of numerical accuracy using a reduced-complexity lookup table (RCLT) algorithm, which is much less computationally-intensive than the CORDIC. On programmable digital signal processor (DSP) chips, this reduces computation time. On field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs), this reduces the consumption of hardware resources.

This paper presents the results of a study in which three commonly-used functions, i.e. inverse, log, and arctangent functions, have been synthesized for the Xilinx Virtex-II family of FPGAs, using both the RCLT and CORDIC implementations. Each implementation is compared for numerical accuracy of the output, the amount of hardware resources consumed, and throughput latency. In each case, it is shown that an RCLT implementation can be designed to achieve similar mathematical accuracy to the corresponding CORDIC implementation while consuming significantly less programmable logic

Certaines fonctions mathématiques, **comme logarithme,** réciproque (inverse) et tangente inverse (arc Tg), sont habituellement calculées dans le domaine numérique au moyen de l'algorithme CORDIC (calculateur numérique de rotation de coordonnées). La présente étude montre qu'il est possible d'obtenir un degré comparable de précision numérique en utilisant un algorithme de recherche dans une table de recherche à complexité réduite (RCLT), qui exige beaucoup moins de puissance de calcul que CORDIC. Dans les processeurs de signal programmables (DSP), cela réduit le temps de calcul. Dans les matrices logiques programmables par l'utilisateur (FPGA) et les circuits intégrés à application spécifique (ASIC), cela réduit la consommation de ressources matérielles.

Ce document présente les résultats d'une étude au cours de laquelle trois fonctions largement utilisées, logarithme, réciproque et tangente inverse, ont été synthétisées pour la famille de FPGA Xilinx Virtex-II, en mettant en oeuvre les algorithmes RCLT et CORDIC. Chaque mise en oeuvre est comparée sur le plan de la précision mathématique des résultats, de la quantité de ressources matérielles utilisées et du temps d'attente du résultat. Dans chaque cas, on a démontré qu'une mise en oeuvre de RCLT peut être conçue de manière à obtenir une précision mathématique semblable à celle des mises en oeuvre correspondantes de CORDIC, tout en réduisant considérablement la consommation de logique programmable et en réduisant les retards liés aux calculs.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

FPGA; LPI; SEI; CORDIC; reduced-complexity; lookup; table; inverse; arctan; log

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca