HB0818

Handbook
CoreLNSQRT v2.0

Microsemi
a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

1.1 Revision 1.0

Revision 1.0 is the first publication of this document. Created for CoreLNSQRT v2.0.

2 Introduction

The Coordinate Rotation Digital Computer (CORDIC) algorithm is an iterative technique, which uses shift and add operations to evaluate the basic arithmetic operations and mathematical functions. CORDIC can be implemented in three basic forms—trigonometric, linear, or hyperbolic, enabled in two modes—rotational mode and vector mode. In rotation mode, CORDIC rotates the inputs by a given angle, while in vector mode, one of the inputs is forced to zero.

In vector mode, CoreLNSQRT is used to compute natural logarithm and square root of a fixed point positive number. The general implementation of CORDIC in hyperbolic mode has limited range of inputs. An expanded CORDIC is used to extend the range of input up to 64 bits.

2.1 Features

The CoreLNSQRT provides the following features:

- Expanded hyperbolic function for wide input range (up to 64 bits)
- Selectable Sequential or pipelined architecture
- Square root and natural logarithm computation

2.2 Core Version

This handbook is for CoreLNSQRT version 2.0.

2.3 Supported Families

- PolarFire®
- RTG4™
- IGLOO®2
- SmartFusion®2

2.4 Utilization and Performance

Table 1 lists the resources utilized for CoreLNSQRT based on various combinations of generic parameters on SmartFusion2, IGLOO2, and RTG4. The G_INPUT_WIDTH and G_OUTPUT_WIDTH values are taken to be equal and G_NO_OF_ITERATIONS is taken as 25.

Table 1 • **Resource Utilization for CoreLNSQRT Block on SmartFusion2, IGLOO2, and RTG4**

Architecture	Input and Output Width	LUT	DFF
Sequential	64	4400	380
Sequential	32	2100	250
Sequential	24	1640	210
Pipeline	64	13890	5540
Pipeline	32	5650	2840
Pipeline	24	3570	2200

Table 2 lists the resources utilized for CoreLNSQRT based on various combinations of generic parameters on PolarFire. The G_INPUT_WIDTH and G_OUTPUT_WIDTH values are taken to be equal and G_NO_OF_ITERATIONS is taken as 25.

Table 2 • **Resource Utilization for CoreLNSQRT Block on PolarFire**

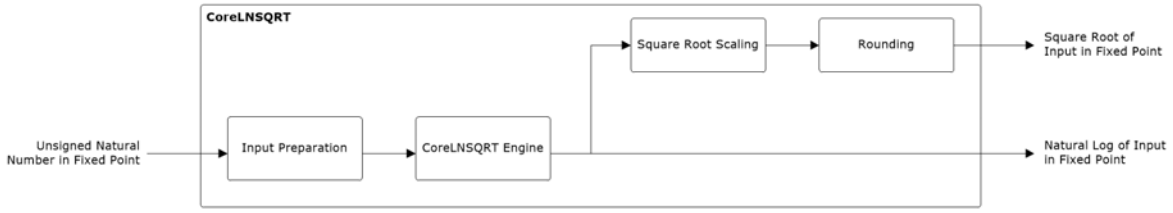
Architecture	Input and Output Width	LUT	DFF
Sequential	64	4510	370
Sequential	32	2170	250
Sequential	24	1630	210
Pipeline	64	13890	5540
Pipeline	32	5650	2840
Pipeline	24	3570	2200

3 Functional Description

3.1 Hardware Implementation

Figure 1 shows the block diagram of the CoreLNSQRT block.

Figure 1 • **CoreLNSQRT IP Block Diagram**



The CoreLNSQRT computes the natural log and square root value of a fixed point positive integer that can range from 16 bits to 64 bits. The square root output is an absolute number after truncation or rounding based on the configuration. The natural log output is scaled up by 16384.

The START_I and DONE_O signals are a pulse of one clock cycle width when the block is configured in sequential architecture, and will remain high when configured in pipeline architecture.

3.2 Types of Architecture

CoreLNSQRT operates in iterations. The actual number of iterations can be calculated as follows:

Number of actual iterations to be executed (n) = N - I ……equation 1

Where,

N = G_NO_OF_ITERATIONS, which is a configuration parameter ranging from 13 to 25 and can be configured by the user.

I = Initial index = 1, If 54 < input bit width < 65

= 2, If 44 < input bit width < 55

= 3, If 35 < input bit width < 45

= 4, If 27 < input bit width < 36

= 5, If 20 < input bit width < 28

= 6, If 14 < input bit width < 21

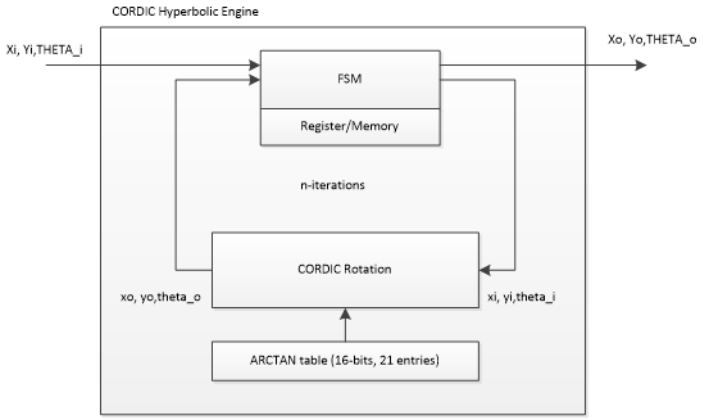
= 1, Otherwise

The CoreLNSQRT can be used with one of the following two architectures. The difference in these two architectures are in the CoreLNSQRT engine.

3.2.1 Sequential Architecture

Figure 2 shows the block diagram for sequential architecture.

Figure 2 • **Sequential Architecture Block Diagram**



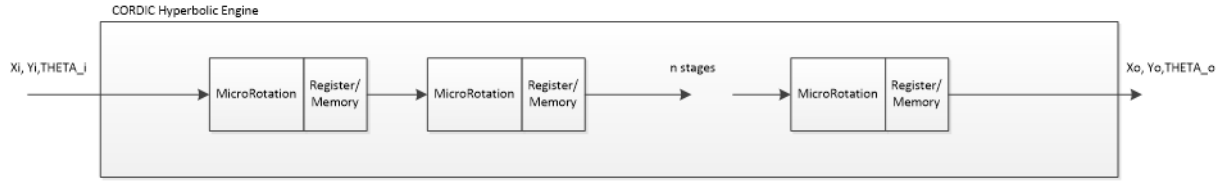
In this architecture one input can be processed at a time. The operation starts as soon as the START_I signal goes high for one clock cycle. The second input will not be processed until the first input is processed completely and we get the DONE_O signal as high for one clock cycle. Since we are using same micro rotation block for n iterations, a lesser area is achieved with the cost of decrease in throughput.

Total delay for processing an input = n clock cycles ≤ N

3.2.2 Pipeline Architecture

Figure 3 shows the block diagram for pipeline architecture.

Figure 3 • **Pipeline architecture Block Diagram**



In this architecture multiple inputs can be applied at every rising edge of the clock provided the START_I signal is high throughout. Here n iterations are broken down to n stages. The output will be available once the DONE_O signal goes high. An increase in throughput is achieved at the cost of increase in area.

The flow of inputs among different stages is shown in Table 3:

Table 3 • **Pipeline Architecture Inputs Flow**

Input Sequence	No of Clock Periods	Microrotation (I)	Microrotation (I + 1)	.....	Microrotation (N)
Input_1	Clockperiod_1	Input_1			
Input_2	Clockperiod_2	Input_2	Input_1		
Input_3	Clockperiod_3	Input_3	Input_2	Input_1	
.					
.					
Input_n	Clockperiod_n	Input_n	Input_(n-1)		Input_1

Here I, n and N are same as in equation 1.

Output corresponding to input_1 will be available in nth clock cycle, Output corresponding to input_2 will be available in (n+1)th clock cycle, Output corresponding to input_3 will be available in (n+2)th clock cycle and so on.

Depending on the requirements of application we can use any one of the above architectures.

4 Interface

4.1 Ports

The port signals for CoreLNSQRT are described in Table 4 and as shown in Figure 4.

Figure 4 • **CoreLNSQRT I/O Signals**

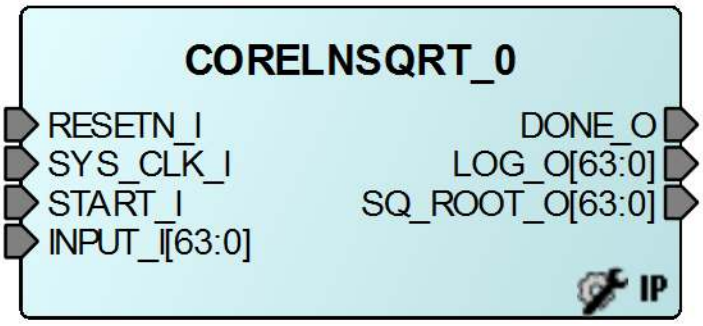


Table 4 • **I/O Signals Description**

Port Name	Type	Width	Description
RESETN_I	Input	1bit	Active low asynchronous reset signal to design
SYS_CLK_I	Input	1bit	System clock
START_I	Input	1bit	Starts square root or log computation in sequential mode, and acts as an enable signal in pipeline mode.
INPUT_I	Input	16 to 64 bit	Input number whose square root or log is to be computed
DONE_O	Output	1bit	Asserted as a pulse of one clock cycle width in sequential mode, and indicates results ready in pipeline mode.
LOG_O	Output	24 to 64 bit	Natural logarithm output (scaled up by 16384)
SQ_ROOT_O	Output	24 to 64 bit	Square root output (no scaling)

4.2 Configuration GUI Parameters

Table 5 lists the description of generic configuration parameters used in the hardware implementation of CoreLNSQRT, which can vary based on the application requirements.

Table 5 • **Configuration Parameters**

Name	Description
G_ARCHITECTURE	If 0, sequential architecture If 1, pipeline architecture
G_INPUT_WIDTH	Defines the bit-width of the input signal
G_OUTPUT_WIDTH	Defines the bit-width of the output signal
G_NO_OF_ITERATIONS	Number of CORDIC iterations to be executed

5 Timing Diagrams

Figure 5 shows the timing diagram of the CoreLNSQRT block implemented as a sequential architecture.

Figure 5 • **Sequential Architecture Timing Diagram**

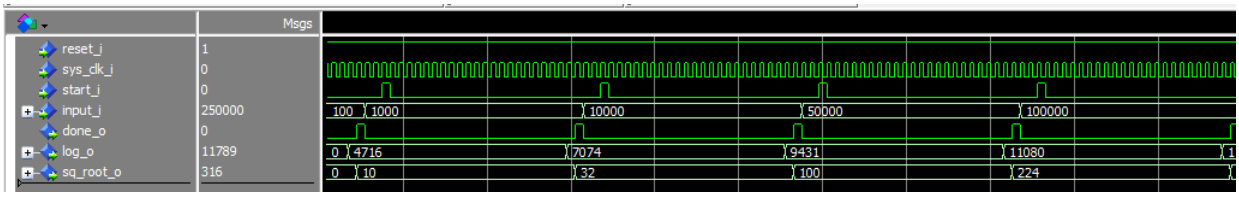
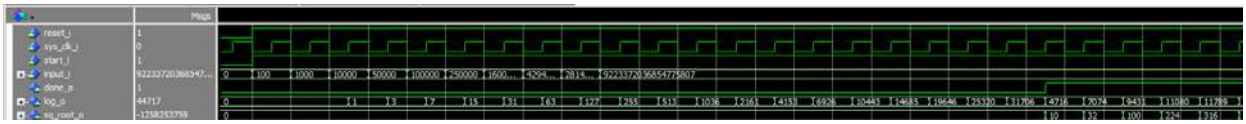


Figure 6 shows the timing diagram of the CoreLNSQRT implemented as a pipelined architecture.

Figure 6 • **Pipelined Architecture Timing Diagram**



6 Tool Flow

6.1 License

CoreLNSQRT clear RTL is license locked and the encrypted RTL is freely available.

6.1.1 Encrypted

- Complete RTL code is provided for the core, enabling the core to be instantiated with SmartDesign.
- Simulation, Synthesis, and Layout can be performed with Libero software. The RTL code for the core is encrypted using the IP encryption (encryptP1735.pl) solution.

6.1.2 RTL

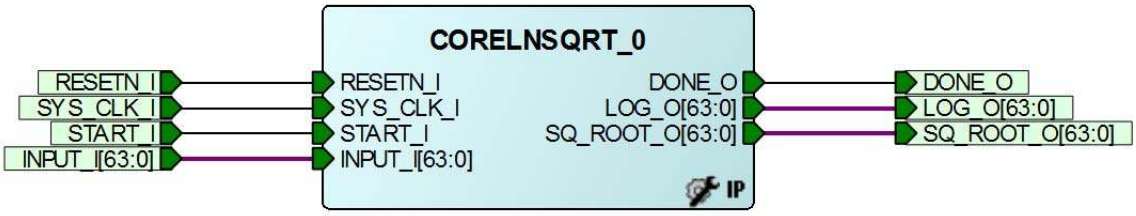
Complete RTL source code is provided for the core.

6.2 SmartDesign

CoreLNSQRT is available for download in the Libero IP catalog through web repository. Once it is listed in the catalog, the core can be instantiated using the SmartDesign flow. For information on using SmartDesign to configure, connect, and generate cores, see Libero SoC online help. An example instantiated view is shown in Figure 7.

After configuring and generating the core instance, the basic functionality can be simulated using the test-bench supplied with the CoreLNSQRT. The CoreLNSQRT can be instantiated as a component of a larger design.

Figure 7 • **CoreLNSQRT Instance View**



6.3 Simulation Flows

The user testbench for CoreLNSQRT is included in all releases. After instantiating the IP in the SmartDesign, click Generate Design under the SmartDesign menu. To run the simulation, Set the CoreLNSQRT in Design Hierarchy window as design root and then set the CoreLNSQRT_tb in Stimulus Hierarchy window as the active stimulus. Double click the Pre-Synthesis simulation in the Design Flow window. This will invoke the ModelSim and automatically runs the simulation.

6.4 Synthesis in Libero

To run the synthesis, Set the CoreLNSQRT IP in the Design Hierarchy window as design root and then double click Synthesis in Design Flow window.

6.5 Place-and-Route in Libero

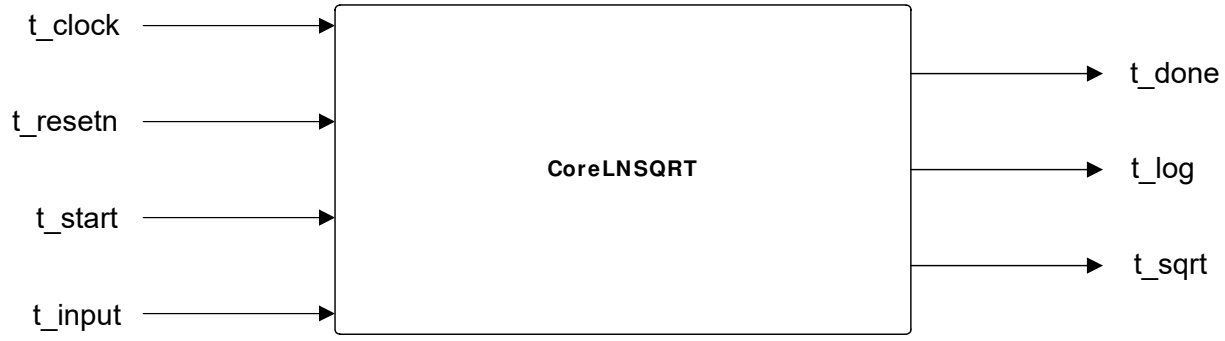
After the design is synthesized, run the compilation and the place-and-route the tools. CoreLNSQRT requires no place-and-route settings.

7 Testbench

7.1 User Testbench

Figure 8 shows the block diagram of the user testbench instantiating the CoreLNSQRT.

Figure 8 • **User Testbench**



The user testbench supports both sequential and pipeline architecture(G_ARCHITECTURE = 0 for sequential and G_ARCHITECTURE = 1 for pipeline) for some fixed set of input and output widths.

A particular input output width combination can be chosen by choosing the G_IO_COMBINATION value in the testbench as follows

If G_IO_COMBINATION = 1, then input width = 64 and output width = 32

If G_IO_COMBINATION = 2, then input width = 48 and output width = 32

If G_IO_COMBINATION = 3, then input width = 36 and output width = 32

If G_IO_COMBINATION = 4, then input width = 32 and output width = 24

If G_IO_COMBINATION = 5, then input width = 24 and output width = 24

If G_IO_COMBINATION = 6, then input width = 18 and output width = 24

These input output width combinations are applicable only for the user testbench. But, the IP supports all input widths(16 to 64) and all output widths (24 to 64) as mentioned in Table4.

7.2 Sample Inputs and Outputs

Table 6 lists a set of sample inputs and outputs. The natural log output is scaled by 16384.

Table 6 • **Sample Inputs and Outputs**

Input	Square Root	Natural Log
100	10	75451
10000	100	150902
100000	316	188628
16000000	4000	271779

8 Ordering Information

8.1 Ordering Codes

CoreLNSQRT can be ordered through your local Microsemi sales representative. It should be ordered using the following number scheme: CoreLNSQRT -XX, where XX is listed in Table 7.

Table 7 • **Ordering Codes**

XX	Description
RM	RTL multi-use multiple-site license