

MATLAB for Synthesis

Style Guide

UG637 (v11.1) April 27, 2009





Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS” WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2002-2009 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

Table of Contents

Preface: About This Guide

Guide Contents	15
Additional Resources	15
Conventions	15
Typographical.....	15
Online Document.....	16

Chapter 1: Design File Structure

A Synthesizable MATLAB Design	20
The Streaming Loop	21
The Top-Level Design Function Call	21
Input and Output Data Types	21
The Function M-File	21
Inputs and Outputs to Function Calls within the Design Body	22
How the Design is Mapped to Hardware	23

Chapter 2: Data Types

MATLAB Data Types	25
Constant and Variable Data	25
Initializing Constant Variables.....	26
Persistent Variables	26
Global Variables	26
Data Dimensions	26
'double' Data Type	26
'structure' Data Type	26
Supported MATLAB Coding Styles for Structure Arrays	27
Unsupported MATLAB Coding Styles for Structures	27
Supported Forms of structure-related Functions	28
Unsupported Forms of structure-related Functions.....	28
'char' Data Type.....	28
Supported MATLAB Coding Styles for String Constants	28
Unsupported MATLAB Coding Styles for Constant Strings	29
Supported Forms of string-related Functions	29
Unsupported Forms of string-related Functions.....	29
'gf' Data Type	29
Galois Field Basics	29
Creating a Galois Field in MATLAB	30
Primitive Polynomials	30
Galois Field Type Propagation.....	30
Using the gf Class in AccelDSP Synthesis	31
Data Types Not Supported	32

Chapter 3: MATLAB Construct Reference

Introduction	33
Setting Implementation Parameters	33
Bipartite Tables	34
Linearly Interpolated Lookup Tables	35
InputType	36
AccelDSP Bitwise Functions	39
+ (addition)	40
- (subtraction)	40
* (multiplication)	40
/ (right division)	40
\ (left division)	41
^ (matrix power)	41
.^ (array power)	41
./ (array right divide)	41
.\ (array left divide)	41
< (Less than)	41
<= (Less than or equal)	41
> (Greater than)	41
>= (Greater than or equal)	41
==, eq (equal)	42
~=, ne (Not equal)	42
&& (Logical AND)	42
 (Logical OR)	42
& (Logical AND for arrays)	42
 (Logical OR for arrays)	42
~ (Logical NOT)	42
false	42
true	42
% (comment)	42
a_dsinccompensation	42
abs/accel_abs	43
Description	43
Related MATLAB Functions	43
Differences with the MATLAB Function	43
Features of the accel_abs() Function	43
Syntax	43
Parameters	44
Inputs / Outputs	45
accel_bitand	47
accel_bitcmp	48
accel_bitmerge	49
accel_bitnand	50
accel_bitnor	51
accel_bitor	52

accel_bitpack	53
accel_bitrev2	54
accel_bitrev	55
accel_bitshl	56
accel_bitshr	57
accel_bitsplit	58
accel_bitunpack	59
accel_bitunpackselect	60
accel_bitxor	61
accel_chol_factor/accel_complex_chol	62
Description	62
Related MATLAB Functions	62
Difference in Operation to MATLAB Functions	62
Syntax	62
Parameters	63
Inputs / Outputs	64
Expected Quality of Results	64
accel_cmplxrot	67
accel_givens_rotation	68
Description	68
Related MATLAB Functions	68
Syntax	68
Parameters	69
Inputs / Outputs	70
Expected Quality of Results	71
accel_qr_factor/accel_complex_qr_factor	73
Description	73
Related MATLAB Functions	73
Differences with the MATLAB Function	73
Syntax	73
Parameters	74
Inputs / Outputs	76
Expected Quality of Results	78
accel_qr_inverse/accel_complex_qr_inverse	81
Description	81
Related MATLAB Functions	81
Differences with the MATLAB Function	81
Syntax	81
Parameters	82
Inputs / Outputs	84
Expected Quality of Results	86
accel_triang_inverse/accel_complex_triang_inverse	89
Description	89
Related MATLAB Functions	89
Differences with the MATLAB Function	89
Syntax	89
Parameters	90
Inputs / Outputs	92
Expected Quality of Results	94
accel_triang_solver	97

Description	97
Related MATLAB Functions	97
Syntax	97
Parameters	98
Inputs / Outputs	101
Memory Mapping	103
Quantization	103
Expected Quality of Results	104
acos/accel_cos	107
Description	107
Related MATLAB Functions	107
Differences with the MATLAB Function	107
Extended Features of accel_acos()	107
Syntax	107
Parameters	108
Inputs / Outputs	109
all	110
angle/accel_angle	110
Description	110
Related MATLAB Functions	110
Differences with the MATLAB Function	111
Features of the accel_angle() Function	111
Syntax	111
Parameters	111
Inputs / Outputs	112
any	113
asin/accel_asin	114
Description	114
Related MATLAB Functions	114
Differences with the MATLAB Function	114
Extended Features of accel_asin()	114
Syntax	114
Parameters	115
Inputs / Outputs	116
atan/accel_atan	117
Description	117
Related MATLAB Functions	117
Differences with the MATLAB Function	117
Extended Features of accel_atan()	118
Syntax	118
Parameters	118
Inputs / Outputs	120
atan2/accel_atan2	120
Description	120
Related MATLAB Functions	120
Differences with the MATLAB Function	121
Extended Features of accel_atan2()	121
Syntax	121
Parameters	121
Inputs / Outputs	123
bchdec	123

bchenc	123
bi2de	123
bitand	124
bitcmp	124
bitget	124
bitor	124
bitset	124
bitshift	124
bitxor	124
cart2pol	124
case	124
cat	125
ceil	125
char	125
chol	125
chol_inverse/accel_complex_chol_inverse	126
Description	126
Related MATLAB Functions	126
Differences in Operation to MATLAB Functions	126
Syntax	126
Parameters	127
Inputs / Outputs	128
Expected Quality of Results	129
cicdecim	131
cicinter	131
complex	132
Complex Normalization	132
Description	132
Related MATLAB Functions	132
Differences in Operation to MATLAB Functions	132
Syntax	132
Parameters	133
Inputs / Outputs	134
conj	134
convenc	134
convergent	134
convdenintlv	134
convintlv	134
colon	135
cos/accel_cos	136
Description	136
Related MATLAB Functions	136
Differences with the MATLAB Function	136
Extended Features of accel_cos()	136
Syntax	136
Parameters	137
Inputs / Outputs	138

cumprod	139
cumsum	139
de2bi	140
dfilt	140
diff	140
dot	140
else	140
elseif	140
end	141
eps	141
eq	141
exp/accel_exp	141
Description	141
Related MATLAB Functions	141
Differences with the MATLAB Function	141
Limitations	141
Syntax.....	141
Parameters.....	142
Inputs / Outputs	143
eye	144
factorial	145
false	145
fft	145
filter	145
firhalfband	145
fix	145
fliplr	145
fliprl	145
flipud	145
floor	146
for	146
function	147
gf	147
Givens Array Rotation	147
global	147
hypot	147
if	147
ifft	148
imag	149
inv	149
Inverse Square Root	149
isempty	149
ldivide	149
length	149

load	149
log	149
Description	149
Related MATLAB Functions	150
Differences with the MATLAB Function	150
Syntax	150
Parameters	150
Inputs / Outputs	151
log10	153
Description	153
Related MATLAB Functions	153
Differences with the MATLAB Function	153
Syntax	153
Parameters	153
Inputs / Outputs	154
log2	156
Description	156
Related MATLAB Functions	156
Differences with the MATLAB Function	156
Syntax	156
Parameters	157
Inputs / Outputs	158
max	159
mean	159
mfilt.firdecim	159
mfilt.firtdecim	159
min	159
minus/accel_complex_minus	160
Description	160
Related MATLAB Functions	160
Syntax	160
Parameters	160
Inputs / Outputs	161
mod	162
Description	162
Related MATLAB Functions	162
Differences with the MATLAB Function	162
Syntax	162
Parameters	162
Inputs / Outputs	164
mpower	165
mtimes/accel_complex_mtimes	166
Description	166
Related MATLAB Functions	166
Syntax	166
Parameters	166
Inputs / Outputs	166
ndims	167
ne	167
nexpow2	167

norm	167
ones	168
otherwise	168
pause	168
persistent	168
pi	168
plus/accel_complex_plus	169
Description	169
Related MATLAB Functions	169
Syntax	169
Parameters	169
Inputs / Outputs	170
pol2cart	170
poly2trellis	170
polyval	171
Description	171
Related MATLAB Functions	171
Differences with the MATLAB Function	171
Syntax	171
Parameters	172
Inputs / Outputs	173
pow2	175
power/accel_power	175
Description	175
Related MATLAB Functions	175
Differences with the MATLAB Function	175
Limitations	175
Syntax	175
Parameters	176
Inputs / Outputs	176
prod	177
qr	177
qdrsls_spatial	178
Description	178
Related MATLAB Functions	178
Differences with the MATLAB Function	178
Syntax	179
Parameters	179
Inputs / Outputs	180
quantize	182
quantizer	182
rcosflt	182
rdivide	182
Description	182
Related MATLAB Functions	182
Differences with the MATLAB Function	182
Syntax	182
Parameters	183
Inputs / Outputs	186

reallog	186
real	187
realpow	187
realsqrt	187
rem	187
Description	187
Related MATLAB Functions	187
Differences with the MATLAB Function	187
Syntax	187
Parameters	188
Inputs / Outputs	190
reshape	191
rot90	191
round	191
rsdec	191
rsenc	191
sign	192
sin/accel_sin	192
Description	192
Related MATLAB Functions	192
Differences with the MATLAB Function	192
Extended Features of accel_sin()	192
Syntax	192
Parameters	193
Inputs / Outputs	194
size	195
sqrt	197
Description	197
Related MATLAB Functions	197
Differences with the MATLAB Function	197
Syntax	197
Parameters	197
Inputs / Outputs	199
std	199
structure	199
sum	200
svd/accel_svd	201
Description	201
Related MATLAB Functions	201
Differences with the MATLAB Function	201
Syntax	201
Parameters	202
Inputs / Outputs	203
Expected Quality of Results	204
switch	207
tan	207
Description	207
Related MATLAB Functions	207
Differences with the MATLAB Function	207

Extended Features of accel_sin()	207
Syntax	207
Parameters	208
Inputs / Outputs	209
times/accel_complex_times	209
Description	209
Related MATLAB Functions	209
Syntax	209
Parameters	210
Inputs / Outputs	210
true	211
var	212
Description	212
Related MATLAB Functions	212
Differences with the MATLAB Function	212
Syntax	212
Parameters	212
Inputs / Outputs	214
while	214
zeros	214

Chapter 4: Table of Synthesizable MATLAB Constructs

Programming with MATLAB	217
Data Types and Quantize Functions	217
Flow Control	218
Scripts and Functions	218
Basic Information	218
Array Operations and Manipulations	219
Elementary Matrices and Arrays	219
Opening, Loading, Saving Files	220
Mathematics	220
Mathematical Operators	220
Relational Operators	220
Logical Operators	221
MATLAB Bit-wise Operators	221
AccelDSP Bit-wise Operators	222
Linear Algebra	223
Statistics	223
Trigonometric Functions	223
Polynomials	223
Exponential Functions	224
Complex Numbers	224
Rounding and Remainder	225
Discrete Math	225
Math Constants	225
Signal Processing Library	225
Filters - FIR - General Purpose	225
Filters - Multirate	226
Filters - Other	226
Transformations	226
Communications Library	227

Direct Digital Synthesizers	227
Encoders/Decoders	227
Scramblers / Descramblers	227
Index	229

About This Guide

The AccelDSP™ Synthesis Tool is a product that allows you to transform a MATLAB floating-point design into a hardware module that can be implemented in a Xilinx FPGA. This document describes the AccelDSP coding style guidelines for the MATLAB floating-point model.

Guide Contents

This manual contains the following chapters:

- [“Chapter, comma” cross-reference to chapter 1] [explain chapter content here]
- [List each additional chapter in a separate bulleted item.]
- [“Appendix, comma” cross-reference to the first appendix], [explain appendix content here]
- [List each additional appendix in a separate bulleted item.]
- [Do not list the glossary in this bulleted list.]

Additional Resources

To find additional documentation, see the Xilinx website at:

<http://www.xilinx.com/literature>.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

<http://www.xilinx.com/support>.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
Courier bold	Literal commands that you enter in a syntactical statement	ngdbuild <i>design_name</i>
Helvetica bold	Commands that you select from a menu	File →Open
	Keyboard shortcuts	Ctrl+C
Italic font	Variables in a syntax statement for which you must supply values	ngdbuild <i>design_name</i>
	References to other manuals	See the <i>Development System Reference Guide</i> for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus [7:0] , they are required.	ngdbuild [<i>option_name</i>] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	lowpwr = { on off }
Vertical bar	Separates items in a list of choices	lowpwr = { on off }
Vertical ellipsis .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...		allow block <i>block_name loc1 loc2 ... locn</i> ;

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section “ Additional Resources ” for details. Refer to “ Title Formats ” in Chapter 1 for details.

Convention	Meaning or Use	Example
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the <i>Virtex-II Platform FPGA User Guide</i> .
<u>Blue, underlined text</u>	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Design File Structure

The following information provides guidelines for structuring the constructs within your MATLAB floating-point files.

A Synthesizable MATLAB Design

Figure 1-1 illustrates the basic structure of a MATLAB design that can be synthesized by the AccelDSP™ Synthesis Tool. The design must be represented by a minimum of two M-files, a *script* M-file and a *function* M-file. These two basic files may also reference other related function files.

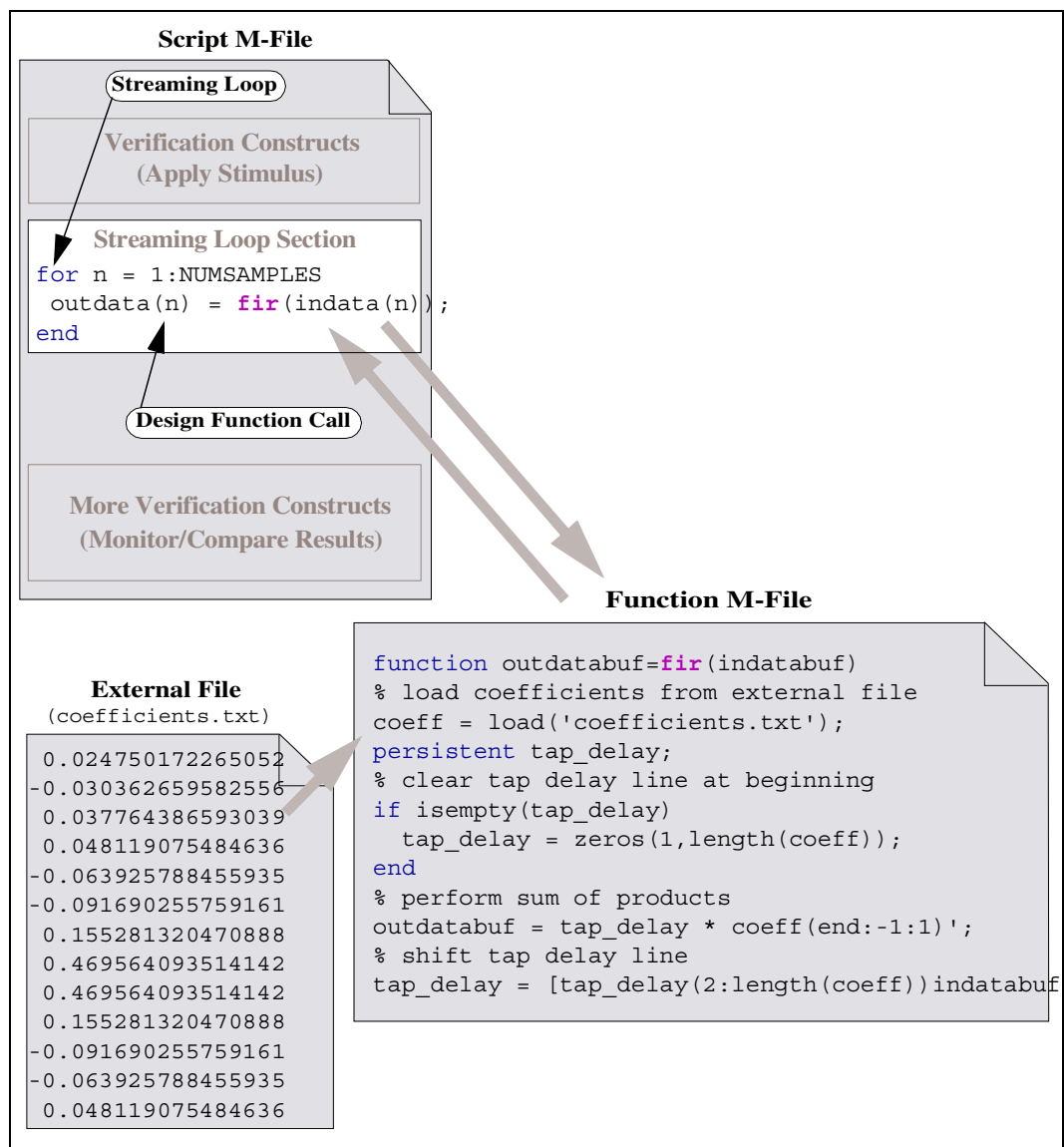


Figure 1-1: The Basic Files of a Synthesizable MATLAB Design

The two main elements of the script file are (1) the streaming loop and (2) the top-level design function call. Other constructs may be intermixed with these basic elements, but are not recognized by the AccelDSP Synthesis Tool for hardware synthesis. These secondary constructs are typically used for design verification.

The Streaming Loop

For hardware synthesis, the infinite streams of data entering and leaving the design must be partitioned into manageable groups or “slices” in order to properly process the data. A streaming loop (a **for** loop or **while** loop) is the construct that performs this task. A MATLAB script file must have a streaming loop.

The Top-Level Design Function Call

The top-level design function call represents the hardware to be synthesized. This function must reside inside the streaming loop. During the pre-analyze phase, the AccelDSP synthesis tool attempts to automatically identify the top-level design function. If it can not, the tool prompts you to manually identify the design function call in the AccelDSP Project Explorer window.

Input and Output Data Types

Each input and output to the top-level design function must be a variable that represents a **scalar**, a **row vector**, or a **column vector**. *All other argument types are not supported.* For example, array elements $a(i)$, sub-expressions $(a+b)$, and function calls like $(\text{abs}(x)*2)$ are not allowed in the top-level design function argument list.

The Function M-File

Figure 1-2 shows a top-level design function file for a floating-point low-pass FIR filter. This function will be synthesized into the hardware block. In this design, the coefficients are maintained in a separate external file that is called within the body of the function. A basic rule for defining a design function is that that a variable in the function input list cannot be specified in the function output list.

You may include calls to other functions within the body of this block, as long as the constructs of the other functions are supported by AccelDSP Synthesis. Each call to a sub-function must be on its own line. For example, the following line is not allowed:

```
a = my_func1(indata1,indata2) + my_func2(indata1,indata2);
```

The equivalent code is allowed:

```
tmp1 = my_func1(indata1,indata2);
tmp2 = my_func2(indata1,indata2);
a = tmp1 + tmp2;
```

Inputs and Outputs to Function Calls within the Design Body

The I/O restrictions that apply to the top-level design function do not apply to function calls within the design body. For example, an argument may be an array element $a(i)$, a sub-expression $(a+b)$, or include a function call like $(\text{abs}(x)^2)$. For example, the following line is allowed within the body of the top-level design function:

```
tmp_out(i) = abs(5*indatabuf(i))
```

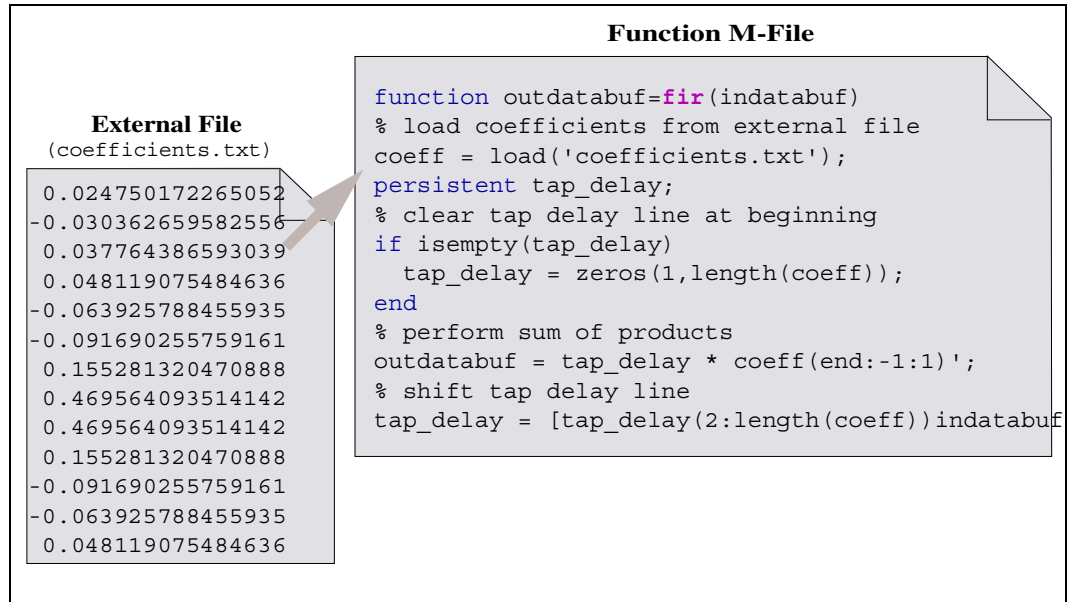


Figure 1-2: A Design Function File

How the Design is Mapped to Hardware

Figure 1-3 shows how the constructs surrounding the top-level design function are synthesized into hardware. The name of the hardware block matches the name of the *design function* call. The names of the input and output ports are derived from the MATLAB variables used to move data into and out of the design function.

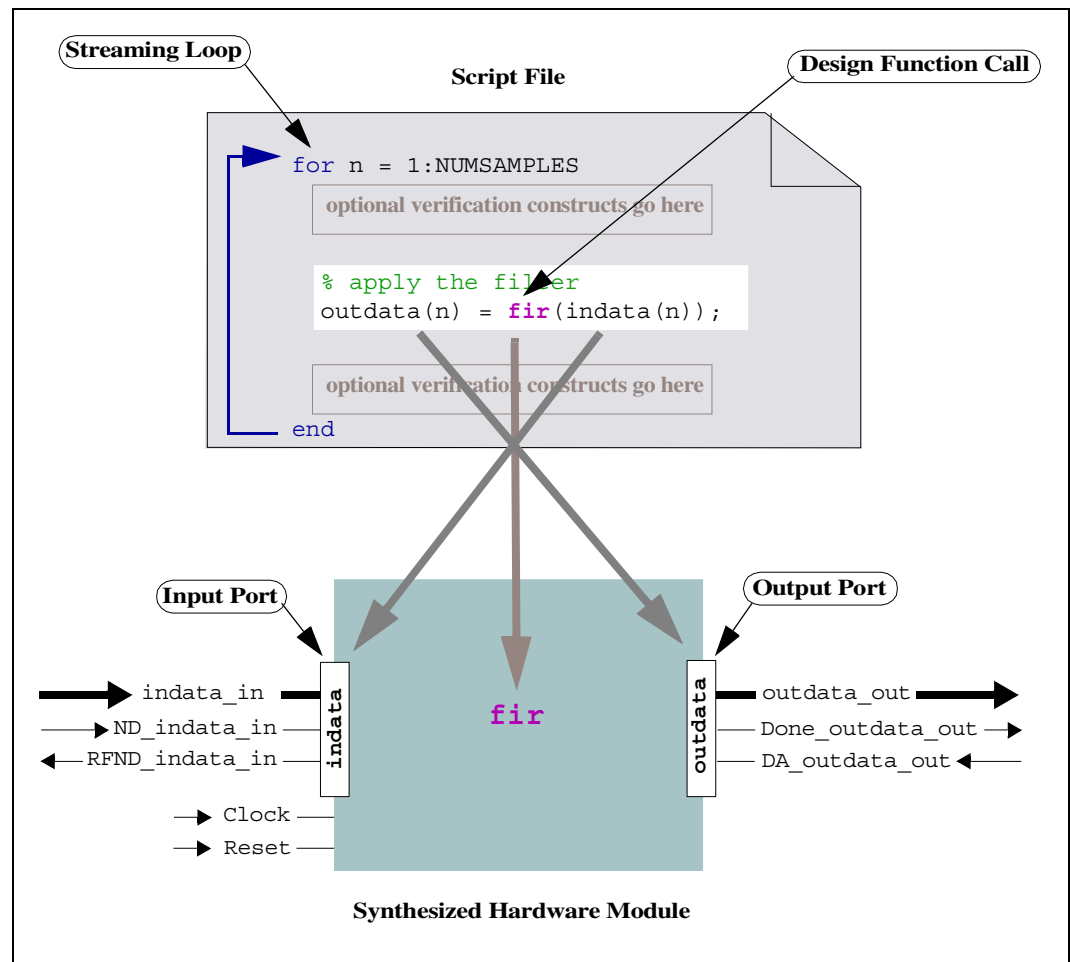


Figure 1-3: Mapping MATLAB Constructs to Hardware Elements

Data Types

This chapter presents the data types and the associated MATLAB coding styles that are supported by the AccelDSP synthesis tool.

MATLAB Data Types

The MATLAB programming language supports the processing of several types of data. In terms of precision, MATLAB supports different data types: double precision floating-point (the default) both real and complex, integers, fixed-point quantized data (as provided with the Filter Design Tool Box), and others. In terms of simple and aggregate data, MATLAB supports: scalars, arrays, cells, structures, and combinations of these. The following sections describe the data types that AccelDSP Synthesis supports for effective hardware synthesis.

Constant and Variable Data

In the MATLAB programming language there is no explicit differentiation between data that is constant (constant variable) and data that is variable (non-constant variable) in a program. Scalars, vectors, and matrices can be assigned single or multiple values throughout the program with no need to differentiate between these cases. However, when a MATLAB program is used for hardware synthesis, it is important to differentiate between these types of data because of the distinct impact on the generated hardware.

The AccelDSP synthesis tool differentiates between constant and non-constant variable data by analyzing the number of times a variable is assigned values throughout the MATLAB program. Specifically, these are the rules followed by the AccelDSP synthesis tool to determine the type of variables present in a MATLAB function:

- If a variable is assigned a value only once in the program, the variable is recognized as a constant variable. A constant variable translates into a constant when the MATLAB program is synthesized into VHDL, and it translates into a parameter when synthesized to Verilog.
- If a variable has multiple assignments throughout the synthesizable constructs, and the first value assigned can be determined at analysis time, then the variable is recognized as a non-constant variable. The determined first assigned value of the variable is treated as an initialization value in the generated HDL RTL Model.
- If a variable has multiple assignments and no initial value is determined at analysis time, then the variable is recognized as a non-constant variable with no initial value in the HDL RTL Model.

Initializing Constant Variables

You can initialize a constant variable by assigning explicit values in the M-file or you can load the values from an external text file.

In the synthesizable code segment below, the constant variable `coeff` is initialized by the six numeric values on the right side of the assignment statement.

```
% define filter coefficients - numerator and denominator polynomials
coeff = [-2.4750172265052; -2.4750172265052; -3.0362659582556; ...
        3.7764386593039; 4.8119075484636; -6.3925788455935];
```

In the synthesizable code segment below, the constant variable `coeff` is initialized with values read from the external text file `coefficients.txt`:

```
% define filter coefficients
coeff = load('coefficients.txt');
```

Persistent Variables

MATLAB persistent variables are supported by AccelDSP.

Global Variables

MATLAB global variables are not currently supported by AccelDSP.

Data Dimensions

The AccelDSP synthesis tool supports operands with fixed-point quantization parameters as described earlier for hardware synthesis. Only real-valued operands are supported of the following dimensions

- Scalar operands
- Single dimensional (row or column) vectors operands
- Two-dimensional matrix operands
- Data structures

'double' Data Type

The AccelDSP synthesis tool provides support for the MATLAB built-in data type **'double'** which is a 64-bit double-precision signed number with a fractional part. Both real and native complex numbers are supported.

'structure' Data Type

In MATLAB, a structure array allows you to create a single variable that holds any number of other variables via named “fields”. These fields can be any type of MATLAB variable that is supported by AccelDSP including a structure which yields a nested structure.

The AccelDSP synthesis tool provides the following support for the **structure** data type:

- Structures can be used in any way in the script file.
- Passing a whole structure into or out of a design function port is not allowed. However, an individual element of a structure can be passed into or out of a design function port as long as a real-valued scalar or vector is assigned to the element.

- Inside the design function and its sub-functions, structures are supported as described below.

Supported MATLAB Coding Styles for Structure Arrays

- The fields of a structure can be any data type that is supported for regular variables (e.g. double, gf, quantizer, structure).
- Structure fields may be constant or non-constant.
- Structure fields can be any shape that is supported for the structure field's type (e.g. for a double structure field, that structure field can be a scalar, vector, or matrix).
- All operations on structure fields should be supported as per the structure field's type (e.g. arithmetic operations are supported for structure fields that are of type double or gf).
- Structure fields that are the same name as other non-structure variables are allowed.
- A structure field that is named the same as its "parent" (or other "ancestor") structure object is allowed (e.g. "a.a.b.a").
- Structure objects may be declared persistent.
- Persistent structure objects may be initialized (within an "if isempty" statement) via:
 - ♦ A series of constant structure field assignments.
 - ♦ A struct function call that returns a structure object whose structure fields are all constant.
 - ♦ A combination of the above two assignments.
- Structure objects may be passed into and out of sub-functions.
- A structure object input argument may be used to auto-infer a function.
- Assigning a structure object to a variable (e.g. "a = b" where b is a structure object) is allowed if one of the following are true:
 - ♦ The left-hand side variable is previously undefined
 - ♦ The left-hand side variable is also a structure object and all structure fields that are common (e.g. the same name) between the left-hand side and right-hand side of the structure objects have the same type and shape. The left-hand side and right-hand side of a quantizer object must be exactly the same. Note that this is a recursive requirement for structure fields that are also structure objects.
 - ♦ The left-hand side and right-hand side variable names are the same (e.g. "a = a").
- A new structure field may be added to a structure object at any time (including in a loop or conditional statement).
- Assigning a structure object to another structure object will cause every structure field that is common to both the left-hand side and right-hand side to be AutoQuantized as if they were directly assigned to each other (this includes if any structure field is in a feedback loop).
- Passing a structure object into or out of a sub-function will cause all the structure object's fields to be AutoQuantized as if they were passed individually into or out-of the sub-function (this includes if any structure field is in a feedback loop).

Unsupported MATLAB Coding Styles for Structures

- Structure objects that are non-scalar will cause an error in Analyze (e.g. "a(2).b").

- Any operations on a whole structure (e.g. arithmetic, logical, etc.) will cause an error in Analyze.
- Dynamic field names are not supported (e.g. using parenthesis around a field name "a.(b)") and will cause an error in Analyze.
- Assigning a structure object to another structure object will result in an error if the left-hand side and right-hand side structure object have a common structure field (or nested structure field) whose left-hand side type or shape is different than the corresponding right-hand side structure field's type or shape. Also, the left-hand side and right-hand side of a quantizer object must be exactly the same or an error will result.

Supported Forms of structure-related Functions

- **struct** returns a scalar structure with any number of fields and initial values. The field names must be constant strings.
- **accel_probe** allows a structure field argument if it is a class 'double'.
- **quantize** allows a structure field argument if it is a class 'double' or 'gf'.

Unsupported Forms of structure-related Functions

- **struct** will error if a field name argument is not a constant string.
- **accel_probe** will error if it is passed a structure object.
- **quantize** will error if it is passed a structure object.
- **structfun** is not supported.

'char' Data Type

The AccelDSP synthesis tool provides native support for MATLAB string constants. String constants are declared using single quote marks surrounding a list of characters. For example:

```
a = 'This is a string.';
```

A MATLAB string is of type 'char'. The AccelDSP synthesis tool provides the following support for **char**:

- char data can be used in any way in the script file.
- Passing data of type char into or out of a design function port is not allowed.
- Inside the design function and its sub-functions, constant strings are supported as described below.

Supported MATLAB Coding Styles for String Constants

- You can pass constant strings into and out of sub-functions.
- A constant string can be assigned to a symbol (e.g. "a = 'my string';") which then becomes a char type.
- A symbol with char type can only be assigned a value once.
- The right-hand side of a char symbol assignment must be a constant string.
- The left-hand side of a char symbol assignment may be a structure field.
- A char type symbol must be used as a whole (e.g. not indexed into).

- The equals and not equals (== and ~=) operators are support for char types if both operands are constant strings.
- Row-vector shaped char types are supported.
- Scalar shaped char types are supported.

Unsupported MATLAB Coding Styles for Constant Strings

- All operators other than equals and not equals that have an operand of char type.
- The equals and not equals operators that have one char type operand and one non-char type operand.
- Square brackets cannot be used around char types (this performs a string concatenation).
- You cannot index into a char type symbol (e.g. "mystringvar(1)").
- Multi-dimensionally shaped char types are not supported.

Supported Forms of string-related Functions

- **length** allows a char type as the first input argument.
- **size** allows a char type as the first input argument.
- **struct** allows a char type for field name and value input arguments.

Unsupported Forms of string-related Functions

- **accel_probe** will error if passed a char type.
- **quantize** will error if passed a char type.
- **str* functions** All string manipulation functions are not supported.

'gf' Data Type

The AccelDSP synthesis tool provides native support for the MATLAB Galois Field class 'gf' which is part of the MATLAB Communications Toolbox. The gf class is a special structural class that is a matrix (representing the Galois Field) and a .x structure field that holds the current gf value as **uint16**. Galois Fields have applications in Forward Error Correction (FEC) - Encoders / Decoders and Encryption

Galois Field Basics

A Galois field is a finite field with p^n elements where p is a prime integer. The set of nonzero elements in the field is a cyclic group under multiplication. A generator of this cyclic group is called a primitive element of the field. The Galois field can be generated as the set of polynomials with coefficients in \mathbb{Z}_p modulo an irreducible polynomial of degree n .

1. A Galois field has 2^m numbers in it.
2. Any number operated on by any other number in a Galois field results in a number in that field.

Creating a Galois Field in MATLAB

Galois field functions in MATLAB are part of the Communications Toolbox. You will need this toolbox to take full advantage of the AccelDSP synthesis tool's support of these functions.

Syntax:

```
<variable> = gf(<input>, <order>, [primitive-polynomial])
```

Primitive Polynomials

The optional primitive polynomial (the third argument) is always represented with an integer.

```
C = gf(indata,4,25); % Use D^4+D^3+1
```

In a Galois Field of order 'm' the primitive polynomial will be represented by an integer having m +1 bits.

The integer is converted to a polynomial as follows:

$$25 = 11001b = 1 \cdot D^4 + 1 \cdot D^3 + 0 \cdot D^2 + 0 \cdot D^1 + 1 \cdot D^0$$

In MATLAB, `primpoly(4)` will list the default primitive polynomials for a Galois Field with an order of 4. `primpoly(4,'all')` will list all primitive polynomials for a Galois Field with m=4.

Galois Field Type Propagation

Data must have the same order and primitive polynomial to be operated on in Galois math. MATLAB assumes this in many cases to avoid the need to repeatedly use the `gf()` function.

```
A = gf(indata,4);
B = A + 1; % B = A + gf(1,4);
C = double(B.x);
```

In the example above, the constant 1 is typecast to the same Galois Field as A. B is also of the same Galois Field type.

Using the gf Class in AccelDSP Synthesis

Figure 2-1 illustrates how the gf class can be used in the design function.

```
function [outdata] = SimpleGalois (indata1, indata2)

% Store indata1 in a Galois Field of order 8 using the primitive
% polynomial D^8+D^7+D^6+D^5+D^4+D^2+1 which is represented by
% the integer 501 from the coefficients of the primitive
% polynomial 111110101b = 501. See the MATLAB function primpoly() for
% more information.
A = gf(indata1,8,501);

% Store indata2 in the same Galois Field. Data must be in the same field
% to operate on each other.
B = gf(indata2,8,501);

% Since the constant 3 is not defined to be in any Galois field, it is
% assumed to be in the same field so the operation can be performed.
C = A * 3; % This is the same as C = A * gf(3,8,501);

% AccelDSP fully supports +, -, *, .* , /
D = (B + 2) / 5;
E = C .* D

% Extract the .x data field from the Galois array and
% convert it from UINT16 to type double. This conversion
% is required.
outdata = double(E.x);
```

Figure 2-1: Mapping MATLAB Constructs to Hardware Elements

Rules for using the 'gf' class in AccelDSP Synthesis are as follows:

- Data entering and leaving the design function cannot be of class gf
- After double precision data enters the design function, it must be converted to class 'gf' using the gf() function in a manner similar to the following: A = gf(indata1,8,501). Class 'gf' data leaving the design function must first be converted to double precision data in a manner similar to the following: outdata = double(E.x);
- Data used in the gf function or used in gf operations cannot have a fractional part.
- If specified, the optional primitive polynomial (the third argument) must be represented by an integer.
- The supported gf operators are: add, subtract, divide, multiply, constant power (i.e. x^3), equals, and not equals. (x^y is only supported when y is a constant.)

Data Types Not Supported

The current version of the AccelDSP synthesis tool does not support hardware synthesis for operands of the following data types:

- Sparse matrices
- Cell arrays
- Empty arrays

MATLAB Construct Reference

Introduction

Setting Implementation Parameters

Some MATLAB functions within the AccelDSP synthesis tool have associated implementation parameters that you can change to customize the resulting hardware. [Figure 3-1](#) illustrates how you can select a function in the Project Explorer window and change the associated parameters in the Properties Viewer window.

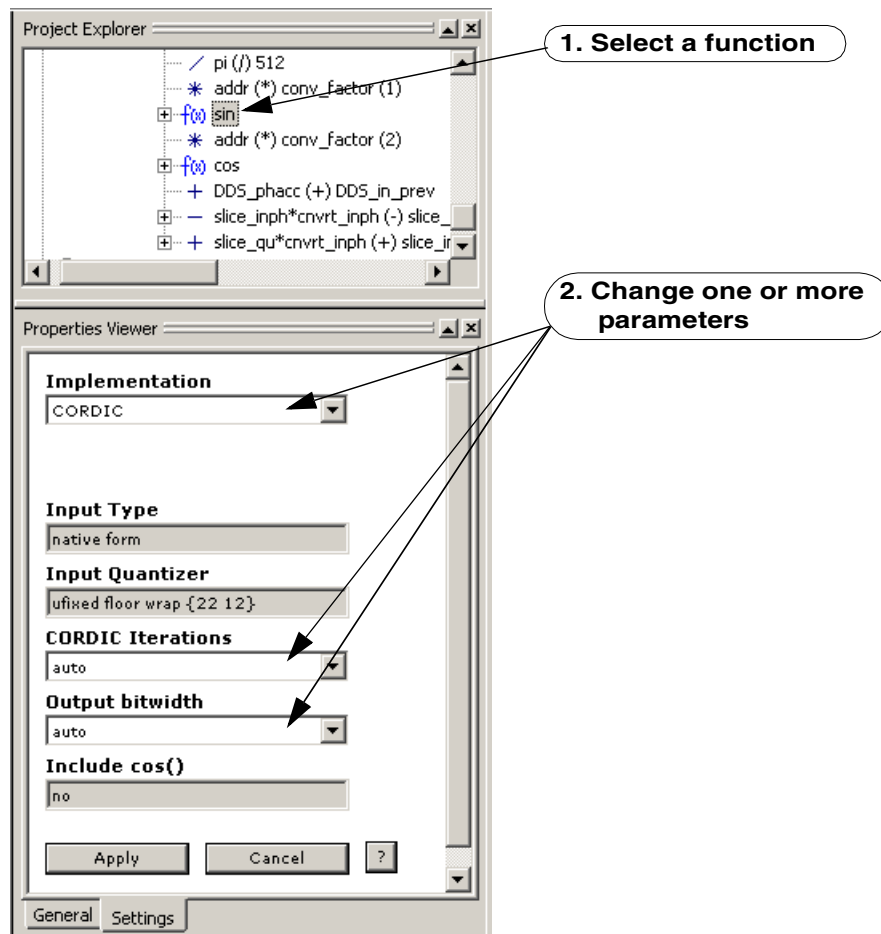


Figure 3-1: Project Explorer window and Properties View window

In Figure 3-1, the `sin` function is selected in the Project Explorer window and the associated parameters are displayed in the Properties Viewer window. Initially, heuristics are used to select the optimal hardware architecture for the function. As the designer, you can make manual adjustments to the parameter if necessary.

Parameters that you can change are displayed in white; parameters that you can't change are displayed in gray. Usually, these non-changeable parameters are inferred directly from the surrounding design environment. In this example, you can change the hardware implementation to either Bipartite Tables or Linear Interpolated LUT, or leave it as CORDIC. You can also choose adjust the number of CORDIC iterations and adjust the Output bitwidth.

Bipartite Tables

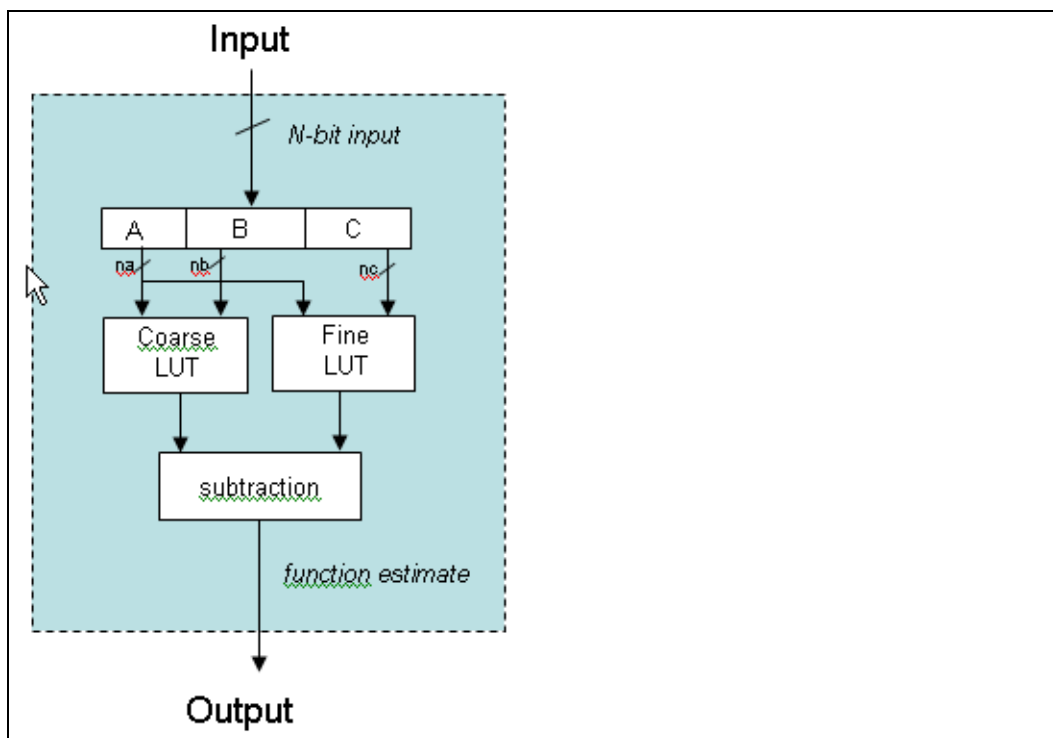


Figure 3-2: Bipartite Table

- N-bit Input word is divided into two smaller words [AB] and [AC].
- MSBs [AB] are used to look-up a coarse estimate of the function value.
- MSBs and LSBs [AC] are used to look-up a “correction” to the coarse estimate.
- Correction factors are constant for each “region” as defined by the MSB [A].
- Regions should be selected such that the function within a region is as linear as possible.

Over-riding Default Values

You may over-ride the default setting while creating a bipartite table implementation by setting the LUT parameters as follows:

1. To fully specify the bipartite parameters set LUT parameters to {A B C}
 - a. $A+B+C$ must be equal to the number of input bits.
2. To partially specify the table the user may simply set the A value. This will determine the number of regions that will be used in the table (see detailed description below).
 - a. C will be set via $C = \text{ceil}((\text{IBITS}-A)/2)$
 - b. B will be set via $B = \text{IBITS}-(A+C)$

The parameters B and C will be alternately reduced (starting with B) when AccelDSP recognizes that function symmetry may be used to the table size.

Bipartite Table Performance Summary

- Good performance for larger input word length with NO multipliers.
- Table Compression ratio (uncompressed table size/bipartite table size)
 - ♦ Increases with input bit width.
 - ♦ Increases as the number of regions decreases (A)

Linearly Interpolated Lookup Tables

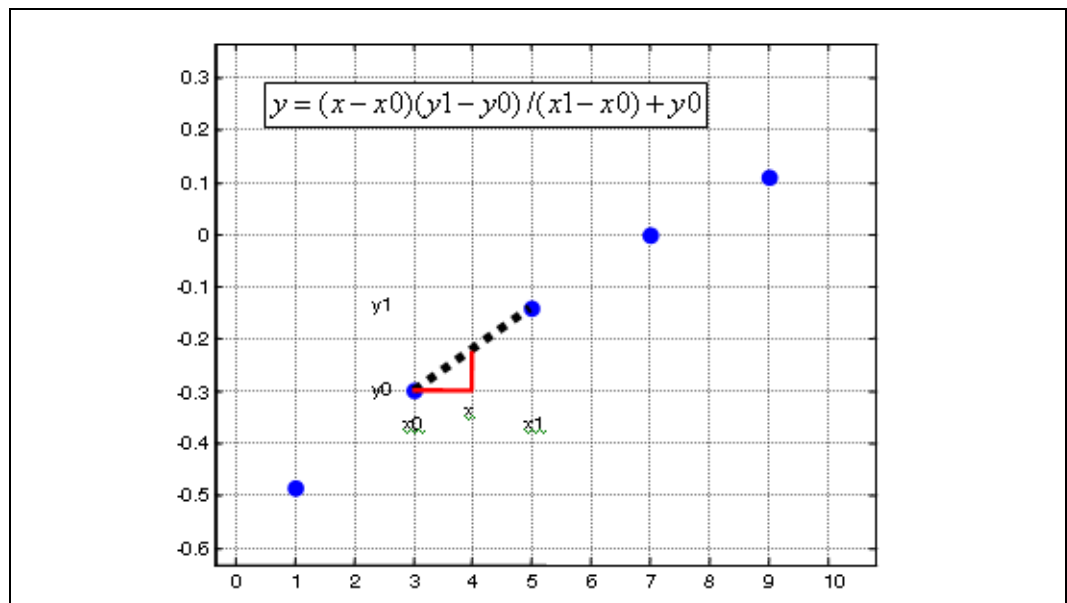


Figure 3-3: Linearly Interpolated Lookup Tables

You may over-ride the default setting while creating a LILT implementation by setting the LUT parameters to the desired compression factor:

1. To create an uncompressed table, set to '1'
2. Compression factor must be power of 2
3. Compression factor must be less than $2^{\text{IBITS}-1}$

InputType

One of the possible implementation parameters is called the 'input type' which can be specified as either 'native form' or 'scaled form'. The following text explains the meaning of these terms.

native form

Native form refers to the input type specified in MATLAB. The native form for sine and cosine is radians. Scaled form is a unit of measure defined by the number of bits in the input word. If the input word is 10 bits, the range of the input is 0 to 1024 (210) integer values. Figure 3-4 shows the relationship between the native form and scaled form.

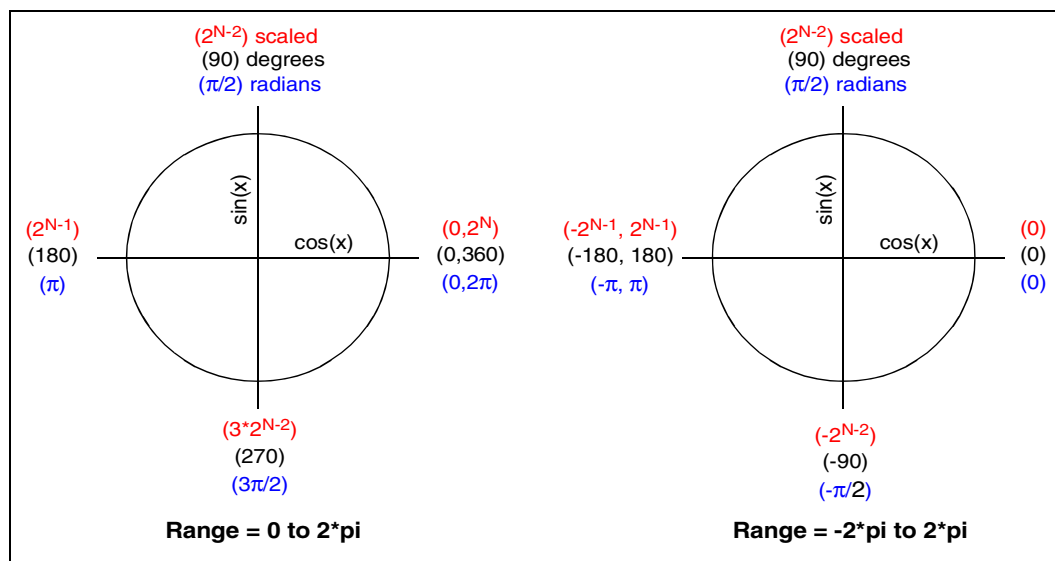


Figure 3-4: InputType - Scaled Form vs. Native Form

For sine and cosine, the native input form is assumed to be radians. If the Input Quantizer Sign Mode is 'ufixed', input range is inferred to be 0 to 2p. If the Input Quantizer Sign Mode is 'fixed', input range is inferred to be 0 to 2p.

scaled form

The input data must be integer values only with no fractional part. The unit of measure is defined by the number of bits in the input word. If the Input Quantizer Word Length is 10 and the Sign Mode is 'ufixed', then the range 0 to 2*p is inferred and it is assumed to be integers between 0 and 210. The only valid value for Fractional Length is 0. The input range is 0 to 1024. As shown in Figure 3-4, the unit circle is divided into 1024 units. If the Input Quantizer Word Length is 10 and the Sign Mode is 'fixed', then the range -p to p is inferred and the input range is -512 to +511. The scale naturally wraps, so the two's complement number +512 is the same as -512.

A scaled input is ideal for algorithms that are constructed in the “scaled” domain from the beginning. There is no need to convert the input data to radians. Also, since a scaled input is integer values only and naturally wraps, there is no need for extra rounding hardware.

Comparing Native Form to Scaled Form Plots in Range 0 to 2π

The sine and cosine plots in Figure 3-5 illustrate the difference between the native form and the scaled form input type when range 0 to 2π is inferred. In the native form plot on the right, the X axis scale is 0 to 2π radians (6.28). Notice that the quantization is ufixed [10 7], meaning three integer bits and 7 fractional bits.

In the scaled form plot on the right, the quantization is ufixed [10 0] so the X axis is a scale of integers from 0 to 2^{10} (1024). The scale naturally wraps so the 1024 point on the scale is the same as 0.

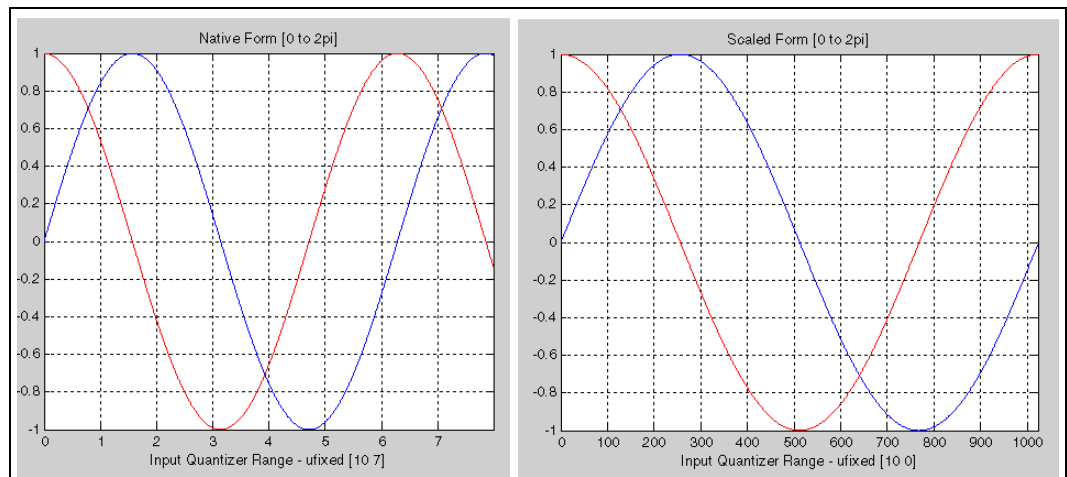


Figure 3-5: Comparing Native Form to Scaled Form Plots in Range 0 to 2π

Comparing Native Form to Scaled Form Plots in Range -2π to 2π

The sine and cosine plots in the Figure 3-6 illustrate the difference between the native form and the scaled form input type when range -2π to 2π is inferred. In the native form plot on the right, the X axis scale is -2π (-3.14) to 2π (3.14) radians. Notice that the quantization is fixed [10 7], meaning a sign bit, two integer bits and 7 fractional bits. Notice also that the native form does not naturally wrap.

In the scaled form plot on the right, the quantization is fixed [10 0], meaning a sign bit and 9 integer bits. The X axis is a scale of integers from -2^9 (-512) to 2^9 (+512). The scale naturally wraps so the +512 point on the scale is the same as -512 point.

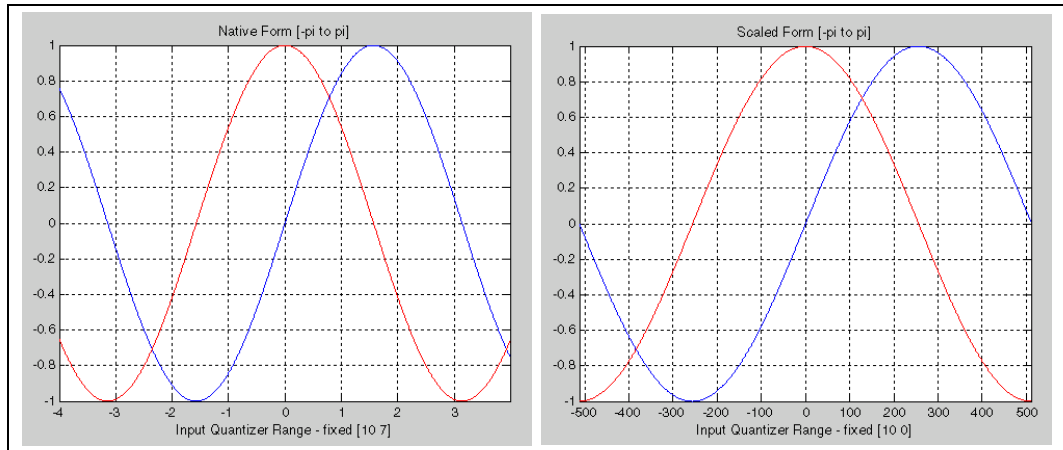


Figure 3-6: Comparing Native Form to Scaled Form Plots in Range -2π to 2π

InputType Conversion Factors

$$\text{radians} = \text{scaled} * p / 2^{(N-1)}$$

$$\text{radians} = \text{degrees} * \pi / 180$$

$$\text{scaled} = \text{radians} * 2^{(N-1)} / \pi$$

$$\text{scaled} = \text{degrees} * 2^{(N-1)} / 180$$

$$\text{degrees} = \text{scaled} * 180 / 2^{(N-1)}$$

$$\text{degrees} = \text{radians} * 180 / \pi$$

AccelDSP Bitwise Functions

The AccelDSP synthesis tool provides specialized bitwise functions that you can use for advanced algorithmic design. These functions allow you to manipulate data at the bit level and are typically used to develop hardware addressing schemes. In general, each bitwise function provides extended capability beyond its equivalent MATLAB function. For example, the `accel_bitand` function allows you to input negative as well as positive integers and allows you to specify the quantization of each input and output port.

Before you can use the specialized AccelDSP functions, you must first add the AccelDSP .../lib/matlab folder (and subfolders) to the MATLAB search path. The folders should be added at a priority level that is higher than any of the MATLAB Tool Boxes. The procedure for adding the folder is described in the AccelDSP User Guide under the topic “Adding AccelDSP MATLAB Functions to the MATLAB Search Path”.

The following table provides a summary and a link to each base function:

Table 3-1: Alphabetical Function Summary

Base Function	Description
accel_bitand	Returns the unsigned bit-wise AND of two integers.
accel_bitcmp	Returns the unsigned bit-wise complement of an integer.
accel_bitmerge	Concatenates two components, MSB (most-significant bits) and LSB (least-significant bits) into one output word.
accel_bitnand	Returns the unsigned bit-wise NAND of two integers.
accel_bitnor	Returns the unsigned bit-wise NOR of two integers.
accel_bitor	Returns the unsigned bit-wise OR of two integers.
accel_bitpack	Returns a single value-representation of the bit-vector argument <code>x</code> quantized to quantizer <code>q</code> .
accel_bitrev2	Returns the unsigned digit reversal of the argument <code>IN</code> .
accel_bitrev	Returns the unsigned bit-wise reversal of the argument <code>IN</code> .
accel_bitshl	Returns the unsigned bit-wise shift-left of the argument <code>IN</code> shifted left by <code>N</code> bits.
accel_bitshr	Returns the unsigned bit-wise shift-right of the argument <code>IN</code> shifted right by <code>N</code> bits.
accel_bitsplit	Splits the input value into MSB (most-significant bits) and LSB (least-significant bits) components according to the bitwidths specified in the specified quantizers.
accel_bitunpack	Returns a row-vector bit-representation of the scalar argument <code>x</code> quantized to quantizer <code>qout</code> .
accel_bitunpackselect	Returns the <code>i</code> 'th element from the output of <code>accel_bitunpack(x, q)</code> .
accel_bitxor	Returns the unsigned bit-wise XOR of two integers.

+ (addition)

Addition is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

Generated Hardware

An RTL adder is generated.

Example 1

```
c = a + b;
```

- (subtraction)

Subtraction is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

Generated Hardware

An RTL subtractor is generated

Example 1

```
c = a - b;
```

***** (multiplication)

Multiplication is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

Generated Hardware

An RTL shift register is generated if multiplying by a constant that is equivalent to 2^n (where n is a real integer). Otherwise, an RTL multiplier is generated.

Example 1

```
c = a * 4; % This will generate a RTL shift register
```

Example 2

```
c = a * b; % This will generate a RTL multiplier
```

/ (right division)

Right MatrixDivision.

Example

```
C = a / 4; % This will generate an RTL shift register
```


\ (left division)

Left MatrixDivision.

^ (matrix power)

$X \wedge Y$ is supported when Y is a scalar integer. Same as the `mpower()` function.

Example

$Z = X \wedge Y;$

Generated Hardware

$X \wedge 3$ turns into $X * X * X$ in hardware.

.^ (array power)

Power is computed element-by-element. X and Y must have the same dimensions unless one is a scalar. A scalar is expanded to an array of the same size as the other input. Same as the `power()` function.

Example

$Z = X \wedge Y;$

./ (array right divide)

Array right divide.

.\ (array left divide)

Array left divide.

< (Less than)

Less than.

<= (Less than or equal)

Less than or equal.

> (Greater than)

Greater than.

>= (Greater than or equal)

Greater than or equal.

==, eq (equal)

Test for equality.

~=, ne (Not equal)

Not equal.

&& (Logical AND)

Logical AND.

|| (Logical OR)

Logical OR.

& (Logical AND for arrays)

Logical AND for arrays.

| (Logical OR for arrays)

Logical OR for arrays.

~ (Logical NOT)

Logical NOT.

false

False array.

true

True array.

% (comment)

Comment.

a_dsinccompensation

A/D Sinc Compensation filter.

abs/accel_abs

Description

Returns the absolute value of the input matrix.

Related MATLAB Functions

[abs](#)

Differences with the MATLAB Function

The shape of the input is limited to scalars, vectors and 2-D matrices. This AccelDSP function does not support native complex numbers.

Features of the accel_abs() Function

The accel_abs() function provides an abs() function with separated complex inputs:

1. Accepts two values instead of a single complex value ($x=a+jb$).
2. Offers an optional angle() output with little additional hardware.
3. Offers a normalized output with range $\{-1 \text{ to } 1\}$ instead of $\{-\pi \text{ to } \pi\}$.

Syntax

AccelDSP abs() Function Call	Supported MATLAB Syntax
<code>y = abs(x);</code>	<code>y = abs(x);</code>
AccelDSP accel_abs() Function Calls	Supported MATLAB Syntax
<code>y = accel_abs(a,b);</code> <code>[y,z] = accel_abs(a,b);</code>	<code>y = abs(x);</code> <code>z = angle(x);</code>

Parameters

Input quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	{CORDIC}
CORDIC Scaling	When a relative measure of the absolute value are sufficient, the CORDIC scaling may be removed to create a multiplier-free implementation.	String	{yes no} Default = {yes}
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB.	Integer	[4 to 100] Default = [auto]

Implementation Parameters			
Name	Description	Type	Range
Output bitwidth	Number of bits used to represent the abs() output y.	Positive Integer	Range = [4 to 32] Default = auto
Include angle() output	A angle() output can be created with little additional hardware. The angle can be returned in radians $[-\pi$ to $\pi]$ or normalized to $[-1$ to $1]$.	String	{no Radians Normalized} Default = {no}

Inputs / Outputs

abs() Input			
Name	Description	Type	Range
x	An input matrix representing real(x) each IBITS-bits wide with IFBITS-bits representing the fractional portion.	Real	Range = $[-2^{\text{IBITS}-1}$ to $2^{\text{IBITS}-1-1}]/2^{\text{IFBITS}}$

accel_abs() Input(s)			
Name	Description	Type	Range
a	An input matrix representing real(x) each IBITS-bits wide with IFBITS-bits representing the fractional portion.	Real	Range = $[-2^{\text{IBITS}-1}$ to $2^{\text{IBITS}-1-1}]/2^{\text{IFBITS}}$
b	An input matrix representing imag(x) each IBITS-bits wide with IFBITS-bits representing the fractional portion.	Real	Range = $[-2^{\text{IBITS}-1}$ to $2^{\text{IBITS}-1-1}]/2^{\text{IFBITS}}$

abs() Output			
Name	Description	Type	Range
y	An output matrix with each element OBITS-bits wide.	Real	Function of the SigInQ parameters

Additional accel_abs() Output			
Name	Description	Type	Range
z	An optional output Angle OBITS-bits wide with (Angle OBITS-3)-bits representing the fractional portion.	Real	Range = $[-\pi$ to $\pi]$ or $[-1$ to $1]$ depending on the value of the IncludeAngle parameter.

accel_bitand

Returns the unsigned bit-wise AND of two integers.

Syntax

```
out = accel_bitand(a , b , qa, qb, qout )
out = accel_bitand(a , b , q, qout ) where qa=qb=q
out = accel_bitand(a , b , q ) where qa=qb=qout=q
```

Related MATLAB Function

[bitand](#)

Description

Returns the bit-wise AND of two integer arguments. To ensure that the arguments are integers, you can use the `ceil`, `fix`, `floor`, and `round` functions.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitand(13,11,q)
y =
    9
```

The four-bit binary representations of the integers 13 and 11 are 1101 and 1011, respectively. Doing a bit-wise AND on these two integers yields 1001, or 9.

	2^3	2^2	2^1	2^0	
	1	1	0	1	(13)
	1	0	1	1	(11)
AND	↓	↓	↓	↓	
	1	0	0	1	(9)

Related Commands

[accel_bitcmp](#)
[accel_bitnand](#)
[accel_bitnor](#)
[accel_bitor](#)
[accel_bitxor](#)

accel_bitcmp

Returns the unsigned bit-wise complement of an integer.

Syntax

```
out = accel_bitcmp( in, qin, qout)
```

Related MATLAB Function

[bitcmp](#)

Description

Returns the unsigned bit-wise complement of the argument IN.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitcmp(13,q)
y =
     2
```

The four-bit binary representation of the integer 13 is 1101. Doing a bit-wise CMP on this integer yields 0010, or 2.

	2^3	2^2	2^1	2^0	
	1	1	0	1	(13)
	1	0	1	1	(11)
AND	↓	↓	↓	↓	
	1	0	0	1	(9)

Related Commands

[accel_bitand](#)
[accel_bitnand](#)
[accel_bitnor](#)
[accel_bitor](#)
[accel_bitxor](#)

accel_bitmerge

Concatenates two components, MSB (most-significant bits) and LSB (least-significant bits) into one output word.

Syntax

```
out = accel_bitmerge(msb, lsb, qmsb, qlsb, qout)
```

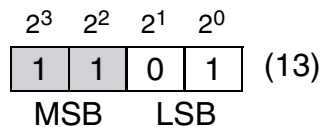
Description

This merge function concatenates two components, MSB (most-significant bits) and LSB (least-significant bits) into one output word.

Example

```
qmsb = quantizer('ufixed','floor','wrap',[2,0]);
qlsb = quantizer('ufixed','floor','wrap',[2,0]);
qout = quantizer('ufixed','floor','wrap',[4,0]);
out = accel_bitmerge(3,1,qmsb,qlsb,qout)
out =
    13
```

The quantizer `qmsb` defines the most significant bits as the upper two bits. The quantizer `qlsb` defines the least significant bits as the lower two bits. The MSB and LSB total must match the word width of the output quantizer `qout` (4 bits). In this case, the binary bits of the MSB (integer 3) concatenated to the binary bits of the LSB (integer 1) are 1101, which is equivalent to the integer 13.



Related Commands

[accel_bitsplit](#)

accel_bitnand

Returns the unsigned bit-wise NAND of two integers.

Syntax

```
out = accel_bitnand(a , b , qa, qb, qout )
out = accel_bitnand(a , b , q, qout ) where qa=qb=q
out = accel_bitnand(a , b , q ) where qa=qb=qout=q
```

Description

Returns the unsigned bit-wise NAND of two integers.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitnand(13,11,q)
y =
    6
```

The four-bit binary representations of the integers 13 and 11 are 1101 and 1011, respectively. Doing a bit-wise NAND on these two integers yields 0110, or 6.

	2 ³	2 ²	2 ¹	2 ⁰	
	1	1	0	1	(13)
	1	0	1	1	(11)
NAND	↓	↓	↓	↓	
	0	1	1	0	(6)

Related Commands

[accel_bitand](#)
[accel_bitcmp](#)
[accel_bitnor](#)
[accel_bitor](#)
[accel_bitxor](#)

accel_bitnor

Returns the unsigned bit-wise NOR of two integers.

Syntax

```
out = accel_bitnor(a , b , qa, qb, qout )
out = accel_bitnor(a , b , q, qout ) where qa=qb=q
out = accel_bitnor(a , b , q ) where qa=qb=qout=q
```

Description

Returns the unsigned bit-wise NOR of two integers.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitnor(13,11,q)
y =
    0
```

The four-bit binary representations of the integers 13 and 11 are 1101 and 1011, respectively. Doing a bit-wise NOR on these two integers yields 0000, or 0.

	2^3	2^2	2^1	2^0	
	1	1	0	1	(13)
	1	0	1	1	(11)
NOR	↓	↓	↓	↓	
	0	0	0	0	(0)

Related Commands

[accel_bitand](#)
[accel_bitcmp](#)
[accel_bitnand](#)
[accel_bitor](#)
[accel_bitxor](#)

accel_bitor

Returns the unsigned bit-wise OR of two integers.

Syntax

```
out = accel_bitor(a , b , qa, qb, qout )
out = accel_bitor(a , b , q, qout ) where qa=qb=q
out = accel_bitor(a , b , q ) where qa=qb=qout=q
```

Related MATLAB Function

[bitor](#)

Description

Returns the unsigned bit-wise OR of two integers.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitor(13,11,q)
y =
    15
```

The four-bit binary representations of the integers 13 and 11 are 1101 and 1011, respectively. Doing a bit-wise OR on these two integers yields 1111, or 15.

	2 ³	2 ²	2 ¹	2 ⁰	
	1	1	0	1	(13)
	1	0	1	1	(11)
OR	↓	↓	↓	↓	
	1	1	1	1	(15)

Related Commands

[accel_bitand](#)
[accel_bitcmp](#)
[accel_bitnand](#)
[accel_bitnor](#)
[accel_bitxor](#)

accel_bitpack

Returns a single value-representation of the bit-vector argument x quantized to quantizer q .

Syntax

```
out = accel_bitpack( x, q)
```

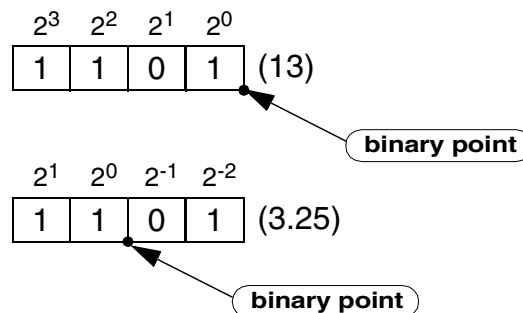
Description

Returns a single value-representation of the bit-vector argument x quantized to quantizer q .

Example

```
x = [1 1 0 1];
qout = quantizer('ufixed','floor','wrap',[4,0]);
out = accel_bitpack(x,qout)
out =
    13
qout = quantizer('ufixed','floor','wrap',[4,2]);
out = accel_bitpack(x,qout)
out =
    3.25
```

The quantizer $qout$ must specify a word width that matches the number of elements in the row vector. In the second example, the quantizer is changed so that the last 2 bits are defined as the fractional part. The equivalent integer is 3.25.



Related Commands

[accel_bitunpack](#)
[accel_bitunpackselect](#)

accel_bitrev2

Returns the unsigned digit reversal of the argument IN.

Syntax

```
out = accel_bitrev2( in, q)
```

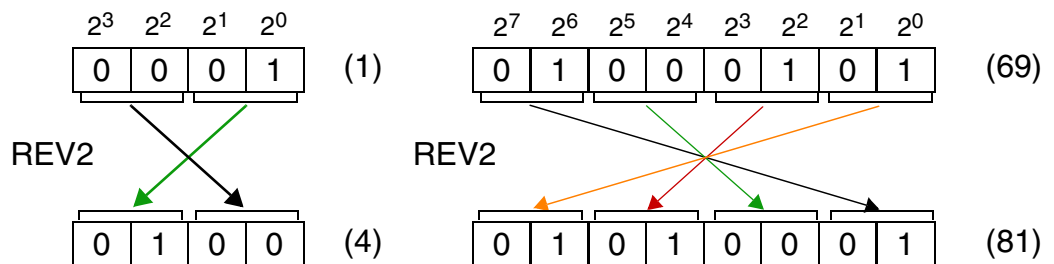
Description

Returns the unsigned digit reversal of the argument IN. Currently, the AccelDSP synthesis tool only supports 2-bit pairs.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
out = accel_bitrev2(1,q)
out =
    4
q = quantizer('ufixed','floor','wrap',[8,0]);
out = accel_bitrev2(69,q)
out =
    81
```

In the examples below, the bit reversals are done in pairs of two.



Related Commands

[accel_bitrev](#)
[accel_bitshl](#)

accel_bitrev

Returns the unsigned bit-wise reversal of the argument IN.

Syntax

```
out = accel_bitrev( in, q)
```

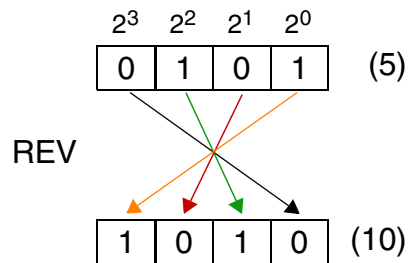
Description

Returns the unsigned bit-wise reversal of the argument IN.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);  
out = accel_bitrev(5,q)  
out =  
    10
```

The illustration below shows how the bits are reversed by the rev function.



Related Commands

[accel_bitrev2](#)

[accel_bitshl](#)

accel_bitshl

Returns the unsigned bit-wise shift-left of the argument IN shifted left by N bits.

Syntax

```
C = accel_bitshl( in, n, qin, qout)
C = accel_bitshl( in, n, q ) where qin=qout=q
```

Related MATLAB Function

[bitshift](#)

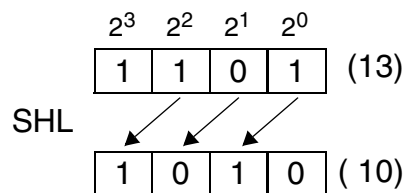
Description

Returns the unsigned bit shift left of the argument IN shifted left by N bits.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitshl(13,1,q)
y =
    10
```

The four-bit binary representation of the integer 13 is 1101. Doing a 1-bit SHL on this integer yields 1010, or 10.



Related Commands

[accel_bitshr](#)

accel_bitshr

Returns the unsigned bit-wise shift-right of the argument IN shifted right by N bits.

Syntax

```
y = accel_bitshr( in, n, qin, qout)
y = accel_bitshr( in, n, q ) where qin=qout=q
```

Related MATLAB Function

[bitshift](#)

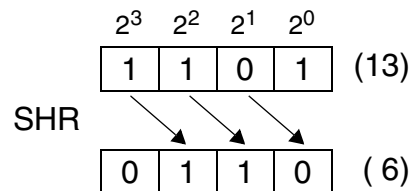
Description

Returns the unsigned bit shift right of the argument IN shifted left by N bits.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitshr(13,1,q)
y =
    6
```

The four-bit binary representation of the integer 13 is 1101. Doing a 1-bit SHR on this integer yields 0110, or 6.



Related Commands

[accel_bitshl](#)

accel_bitsplit

Splits the input value into MSB (most-significant bits) and LSB (least-significant bits) components according to the bitwidths specified in the specified quantizers.

Syntax

```
[msb,lsb] = accel_bitsplit(in, qin, qmsb, qlsb)
```

Description

The function splits the input value into two components, MSB (most-significant bits) and LSB (least-significant bits) according to the bitwidths specified in the output quantizers. The word width of the MSB plus the word width of the LSB must match the word width of the unsigned input integer.

Example

```
qin      = quantizer('ufixed','floor','wrap',[4,0]);
qmsb     = quantizer('ufixed','floor','wrap',[2,0]);
qlsb     = quantizer('ufixed','floor','wrap',[2,0]);
[msb,lsb] = accel_bitsplit(13,qin,qmsb,qlsb)
msb =
     3
lsb =
     1
```

The four-bit binary representation of unsigned integer 13 is 1101. The quantizer qmsb defines the most significant bits as the upper two bits. The quantizer qlsb defines the least significant bits as the lower two bits. The MSB and LSB total must match the word width of the integer (4 bits). In this case, the most significant bits are equivalent to the integer 3 and the least significant bits are equivalent to the integer 1.

2^3	2^2	2^1	2^0	
1	1	0	1	(13)
MSB		LSB		

Related Commands

[accel_bitmerge](#)

accel_bitunpack

Returns a row-vector bit-representation of the scalar argument x quantized to quantizer $qout$.

Syntax

```
out = accel_bitunpack( x, qout )
```

Description

Returns a row-vector bit-representation of the scalar argument x quantized to quantizer $qout$.

Example

```
qout = quantizer('ufixed','floor','wrap',[4,0]);
out = accel_bitunpack(13,qout)
out =
    1 1 0 1
qout = quantizer('ufixed','floor','wrap',[8,0]);
out = accel_bitunpack(13,qout)
out =
    0 0 0 0 1 1 0 1
```

The quantizer $qout$ specifies the word width of the row-vector. Notice that in each case, the binary representation is the equivalent to the integer 13, regardless of the word width.

2^3	2^2	2^1	2^0	
1	1	0	1	(13 quantized as a 4-bit binary word)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	1	1	0	1	(13 quantized as an 8-bit binary word)

Related Commands

[accel_bitpack](#)
[accel_bitunpackselect](#)

accel_bitunpackselect

Returns the i 'th element from the output of `accel_bitunpack(x, q)`.

Syntax

```
out = accel_bitunpackselect( i, x, qout )
```

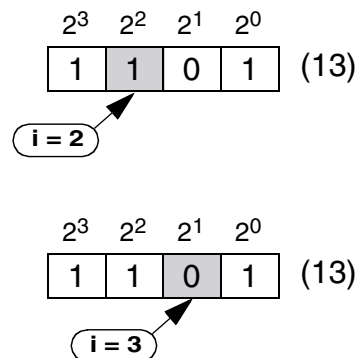
Description

Returns the i 'th element (from the MSB) from the output of `accel_bitunpack(x, qout)`.

Example

```
i = 2;  
qout = quantizer('ufixed','floor','wrap',[4,0]);  
out = accel_bitunpackselect(i, 13,qout)  
out =  
    1  
i = 3;  
out = accel_bitunpackselect(i, 13,qout)  
out =  
    0
```

After the bit unpacking, the value returned is the bit position that is specified by the input.



Related Commands

[accel_bitpack](#)
[accel_bitunpack](#)

accel_bitxor

Returns the unsigned bit-wise XOR of two integers.

Syntax

```
out = accel_bitxor(a , b , qa, qb, qout )
out = accel_bitxor(a , b , q, qout ) where qa=qb=q
out = accel_bitxor(a , b , q ) where qa=qb=qout=q
```

Related MATLAB Function

[bitxor](#)

Description

Returns the unsigned bit-wise XOR of two integers.

Example

```
q = quantizer('ufixed','floor','wrap',[4,0]);
y = accel_bitxor(13,11,q)
y =
    6
```

The for-bit binary representations of the integers 13 and 11 are 1101 and 1011, respectively. Doing a bit-wise XOR on these two integers yields 0110, or 6.

	2^3	2^2	2^1	2^0	
	1	1	0	1	(13)
	1	0	1	1	(11)
XOR	↓	↓	↓	↓	
	0	1	1	0	(6)

Related Commands

[accel_bitand](#)
[accel_bitcmp](#)
[accel_bitnand](#)
[accel_bitnor](#)
[accel_bitor](#)

accel_chol_factor/accel_complex_chol

Description

Operates on a symmetric, positive definite input matrix X to produce an upper triangular output matrix R such that

$$R' * R = X$$

The R is called a Cholesky factor of X .

Related MATLAB Functions

[chol](#)

[Cholesky Factorization Block \(Simulink\)](#)

Difference in Operation to MATLAB Functions

The MATLAB `chol` function can operate on a complex input matrix. The AccelDSP `accel_complex_chol()` function accepts two separate values (real, imag) instead of a single complex value ($x=a+jb$).

The MATLAB `chol` function detects when the input matrix is not positive definite and produces an error indication via a second output p with a non-zero value. The AccelDSP `chol_factor` function produces a second output with a value of 1 that indicates the detection of a non-positive definite input matrix.

Syntax

Function Call	Supported MATLAB Syntax	Notes
<code>R = accel_chol(X);</code>	<code>R = chol(X);</code>	
<code>[R,p] = accel_chol(X);</code>	<code>[R,p] = chol(X);</code>	The output parameter p with a value of 1 indicates that the input matrix is not positive definite.
accel_complex_chol() Func Call	Supported MATLAB Syntax	Notes
<code>[R_real,R_imag] = accel_complex_chol(X_real, X_imag);</code>		
<code>[R_real,R_imag,p] = accel_complex_chol(X_real, X_imag);</code>		

Parameters

Input quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Range = [2 to 20] Default = 12
Fractional Length	The number of fractional bits for the input word(s).	String	Range = [0 to Word Length as per Mode parameter] Default = 11

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	Multiplier
Type	Input matrix data type.	String	double {complex: separate real and imag IO}

Implementation Parameters			
Name	Description	Type	Range
Matrix Size (3:32)	Input matrix size.	Integer	Range = [2 to 32] Default = 4
Output Precision	Number of bits for output matrix.	Integer	Default = auto auto sets value based on input matrix word-length

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
X	Input matrix to be factored. XBITS, XFBITS are the input quantization wordlength and fractional length as per A quantization parameters.	Real	Range = $[-2^{XBITS-1}$ to $2^{XBITS-1}-1]/2^{XFBITS}$

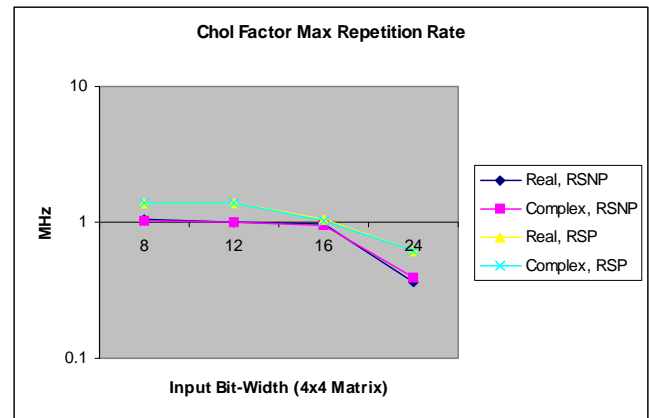
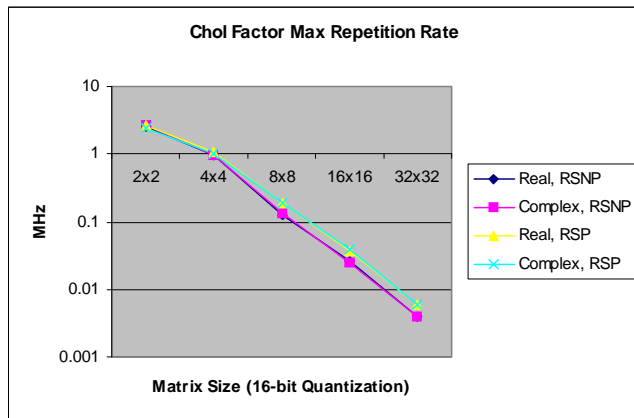
Output(s)			
Name	Description	Type	Range
R	Triangular factor matrix. RBITS, RFBITS are the R matrix output wordlength and fractional length.	Real	Range = $[0$ to $2^{RBITS-1}] / 2^{RFBITS}$
p	0 - indicates that the input matrix is positive definite 1 - indicates that the input matrix is non-positive definite. The output R in this case is not a valid Cholesky factor.	Positive Integer	Range = [0 1]

Expected Quality of Results

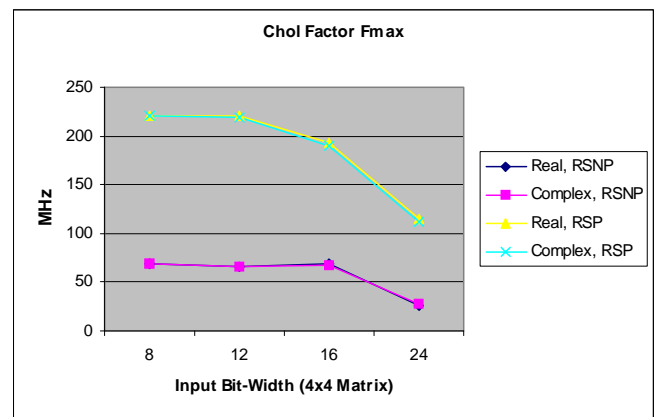
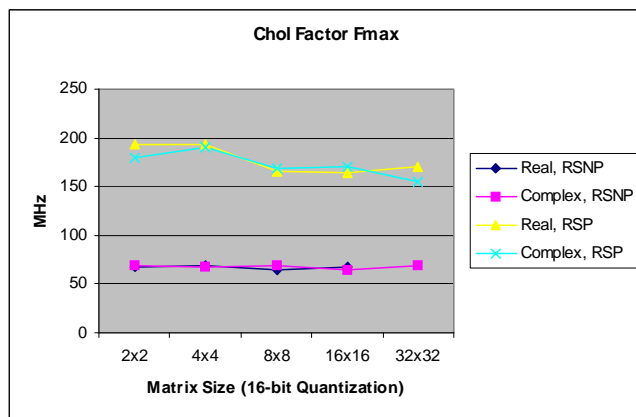
The following plots show typical speed and utilization of the chol_factor core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and RSP refer to resource-shared, non-pipelined architectures and resource-shared pipelined architectures respectively.

Speed of Operation

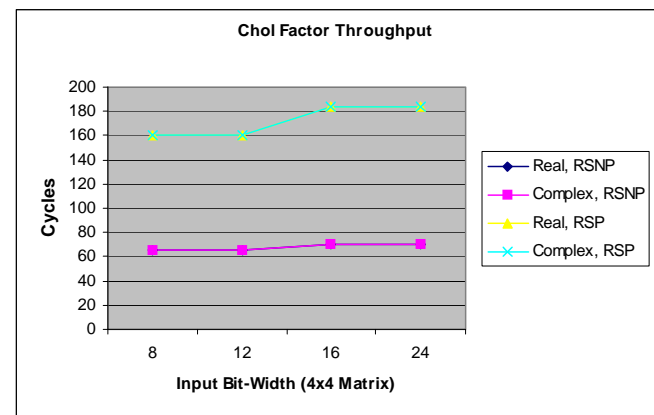
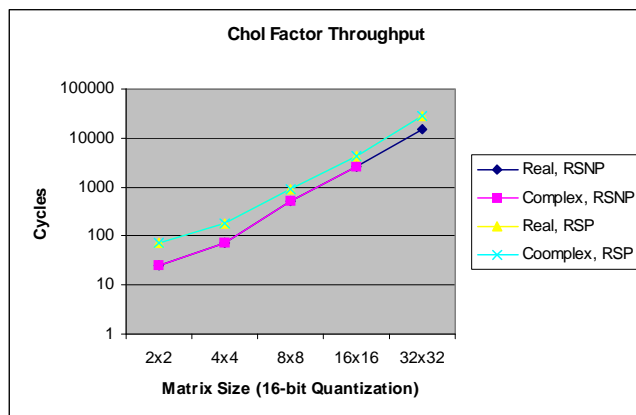
Factorization Repetition Rate



Maximum Operating Frequency

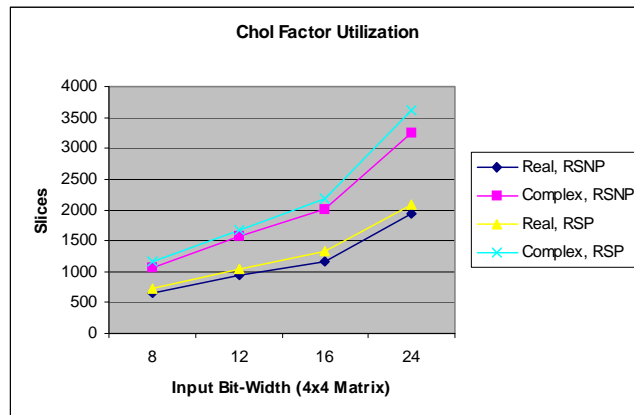
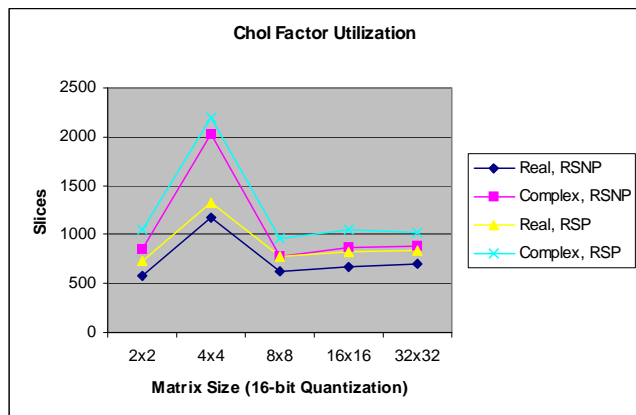


Hardware Clock Cycles per Design Function Call

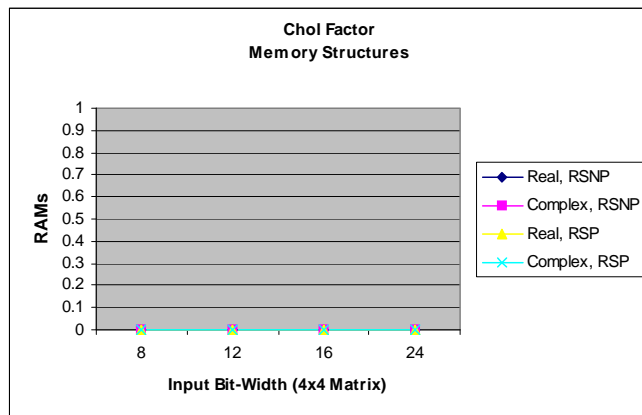
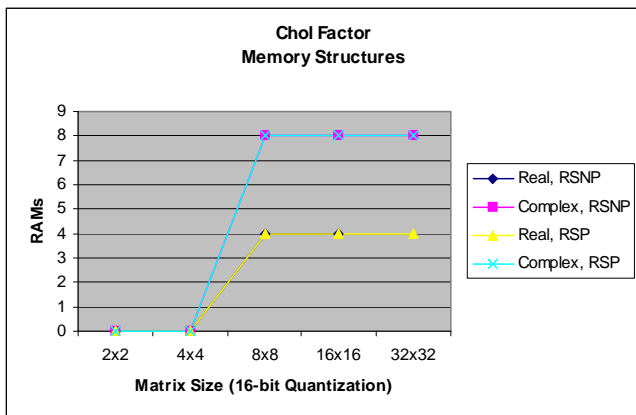


Utilization

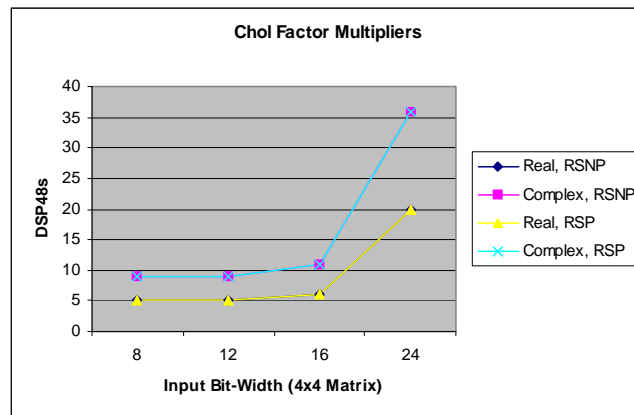
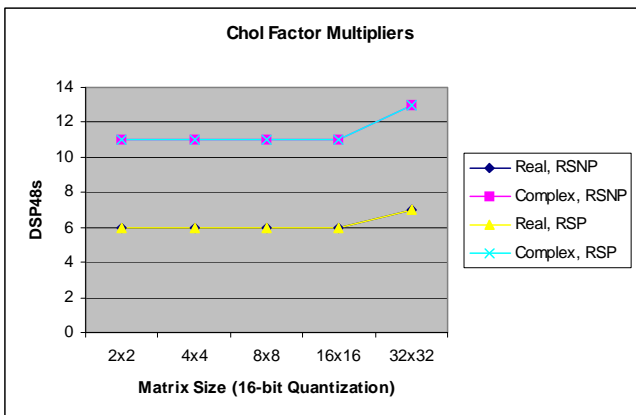
Number of Slices



Number of Memories



Number of Multipliers



accel_cmplxrot

Used to transform polar or cylindrical coordinates to Cartesian coordinates.

Syntax

```
[X,Y] = accel_cmplxrot(A,B,TH)
[X,Y] = accel_cmplxrot(A,B,TH,N)
```

Related MATLAB Function

There is no equivalent MATLAB function.

Description

Form 1: $[X,Y] = \text{accel_cmplxrot}(A,B,TH)$

Equivalent to $Z = C \cdot \exp(j \cdot TH)$ where $Z = (X+jY)$, $C = (A+jB)$ and TH is a real value. A,B and TH can be scalar, vector, or two-dimensional matrix, BUT all must be the same shape.

Form 2: $[X,Y] = \text{accel_cmplxrot}(A,B,TH,N)$

Same as Form 1, but the angle is specified as a scaled value $-2^{(N-1)} \leq TH \leq 2^{(N-1)}-1$ or $0 \leq TH \leq 2^N-1$. $Z = C \cdot \exp(j \cdot TH \cdot \pi / 2^{(N-1)})$. A,B and TH can be scalar, vector, or two-dimensional matrix, BUT all must be the same shape.

accel_givens_rotation

Description

The givens_rotation reference design rotates the elements of a pair of 1-D input arrays by a specified angle of rotation. The angle of rotation is specified as the angle, relative to the positive side of the x-axis, of a Cartesian coordinate pair.

Related MATLAB Functions

[atan2](#)

[cos](#)

[sin](#)

Syntax

Function Call	Supported MATLAB Syntax	Notes
[v,w] = accel_givens_rotation(x,y);	phi = atan2(b,a); v = x*cos(phi) + y*sin(phi); w = y*cos(phi) + x*sin(phi);	The angle of rotation is defined by the pair formed by the first elements of the input x and y arrays taken as the coordinates of a point in the Cartesian plane.

Parameters

Input quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is $-PI$ to PI . ufixed : unsigned fixed-point mode. Infers that the input range is 0 to $2*PI$.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Range = [4 to 16] Default = 12
Fractional Length	The number of fractional bits for the input word(s).	String	Range = [0 to 15] Default = 11

Implementation Parameters			
Name	Description	Type	Range
Implementation	Implementation computation style.	String	CORDIC Multiplier Default = CORDIC
Input Array Size	Size of the input matrix.	Integer	range = [1 to 64] Default = [4]

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
x,y	Input arrays. "word" and "fractional" are the input quantization wordlength and fractional length as per the Input Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{-\text{fractional}}$

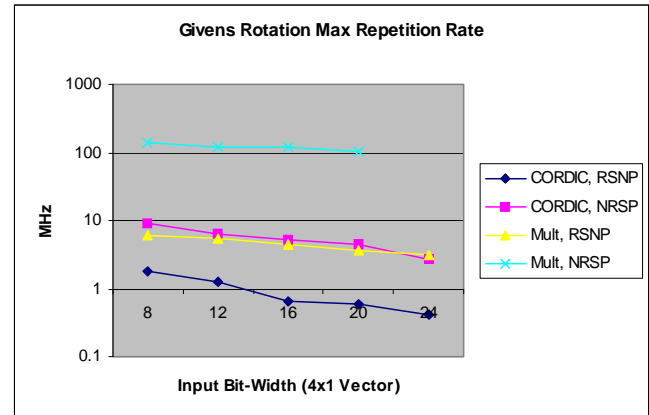
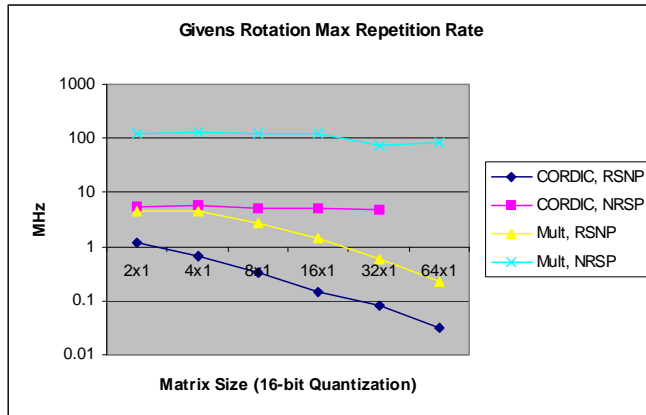
Output(s)			
Name	Description	Type	Range
v,w	Output arrays. OBITS, OFBITS are the output wordlength and fractional length automatically determined by the AccelDSP generator.	Real	Range = $[-2^{\text{OBITS}-1}$ to $2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

Expected Quality of Results

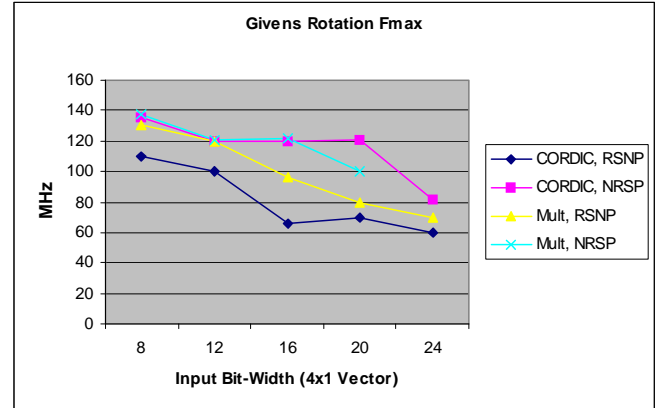
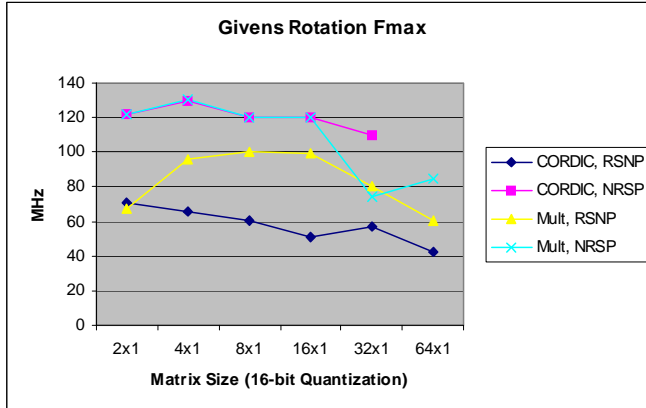
The following plots show typical speed and utilization of the givens_rotation core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and NRSP refer to resource-shared, non-pipelined architectures and non-resource-shared, pipelined architectures respectively.

Speed of Operation

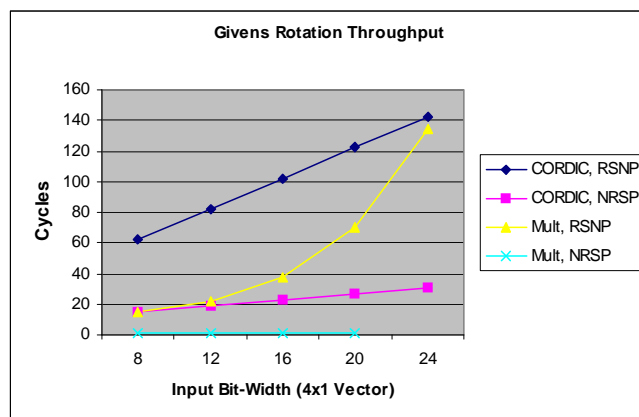
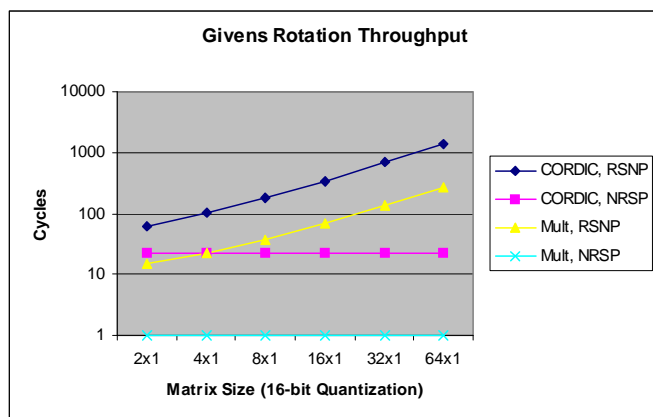
Rotation Repetition Rate



Maximum Operating Frequency

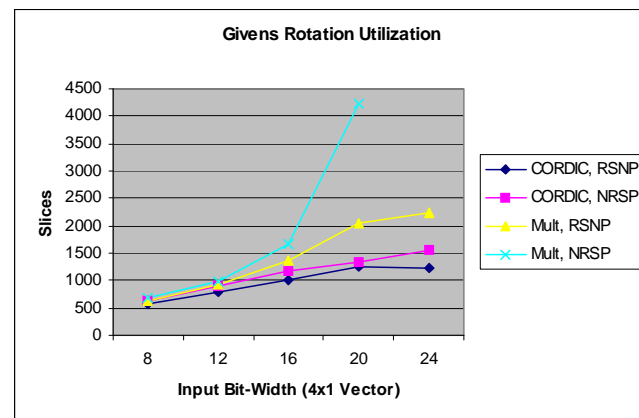
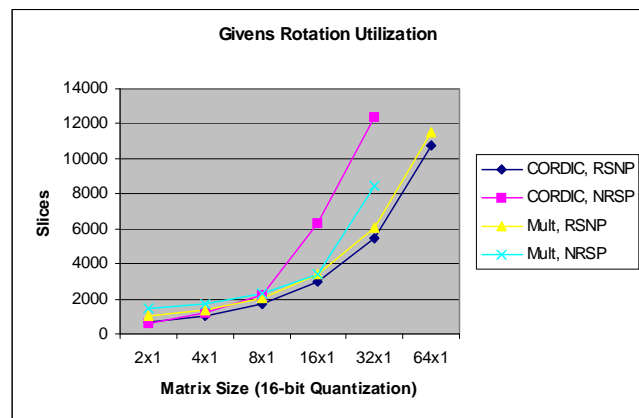


Hardware Clock Cycles per Design Function Call

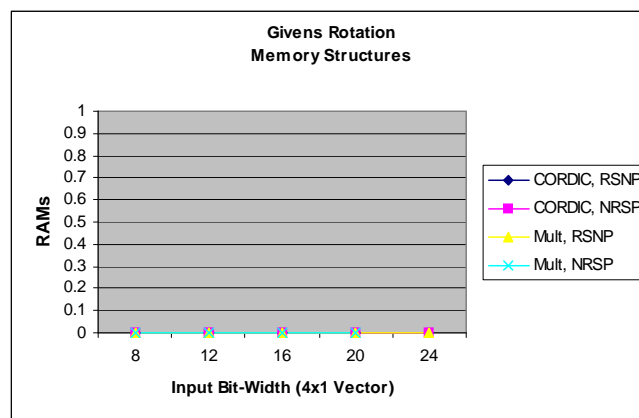
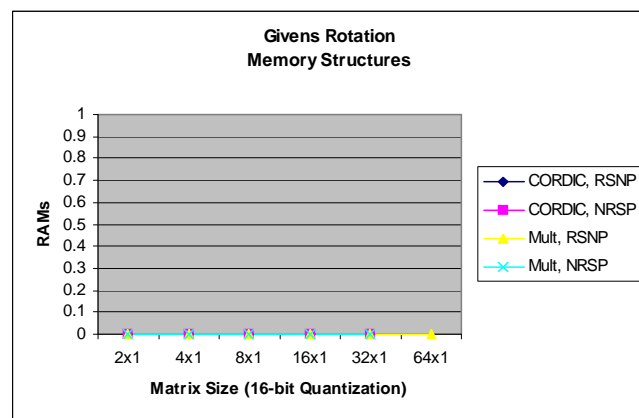


Utilization

Number of Slices



Number of Memories



accel_qr_factor/accel_complex_qr_factor

Description

Orthogonal-triangular decomposition. Decomposes an input matrix A into an upper triangular R matrix and an orthogonal/unitary matrix Q such that $A = Q \cdot R$.

Related MATLAB Functions

[qr](#)

Differences with the MATLAB Function

The MATLAB qr function can operate on full and sparse matrices. The AccelDSP accel_qr_factor function operates on full matrices only.

Syntax

Function Call	Supported MATLAB Syntax	Notes
<code>[Q,R] = accel_qr_factor(A);</code>	<code>[Q,R] = qr(A);</code>	Syntax for real-valued input and output matrices
<code>[Q_real,Q_imag,R_real,R_imag] = accel_complex_qr_factor(A_real, A_imag)</code>	<code>[Q,R] = qr(A);</code>	Syntax for complex-valued input and output matrices. The real and imaginary components are passed as separate matrices.

Unsupported MATLAB Syntax	Notes
<code>[Q,R] = qr(A,O);</code> <code>[Q,R,E] = qr(A,O);</code>	No 'economy' size decomposition is supported.
<code>[Q,R,E] = qr(A);</code>	No generation of permutation matrix E is supported.

Parameters

Input quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Range = [fixed ufixed] Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Range = [4 to 24] for real-only data Default = 16 Range = [4 to 20] for complex data Default = 12
Fractional Length	The number of fractional bits for the input word(s).	String	Range = [0 to Word Length as per Mode parameter]

Implementation Parameters			
Name	Description	Type	Range
Implementation	Algorithm used for factorization implementation	String	Conventional Givens Rotations
Type	Input matrix data type.	String	{double complex} Default = double

Implementation Parameters			
Name	Description	Type	Range
Matrix Rows (1)	Input matrix number of rows.	Integer	Range = [2 to 64] for real-only data Range = [2 to 32] for complex data
Matrix Columns (1)	Input matrix number of columns.	Integer	Range = [2 to 64] for real-only data Range = [2 to 32] for complex data
Data IO Format (2)	Selects the format for the IO streaming of data (array or sample per invocation);it affects the test design function wrapper only.	String	Range = [array sample] Default = [array]
Output Precision	Number of bits for output matrix.	Integer	Default = auto auto sets value based on input matrix word-length
Implementation	Algorithm used for factorization implementation	String	Conventional Givens Rotations

Notes on Parameters:

1. Cores with matrix size of 8x8 or larger are automatically generated with memory mapping directives.
2. Sample-based IO streaming is more efficient in terms of the number of IO signals but there is a cost in the number of clock cycles used for the data transfers. The wrapper generated for sample-based streaming 'streams' data in and out at the boundaries of the design function automatically generated for a core. For cores with matrix size of 8x8 or larger, the generated wrapper includes memory mapping of the input and output matrix arrays for efficient resource utilization. The following two recommendations are for the user:
 - a. When generating cores of matrix size 8x8 or larger, use sample-based IO streaming to support efficient synthesis and mapping to FPGA devices. Not using this option can cause RTL synthesis to take a very long run time to complete or to fail due to an excessive number of logic resources consumed by the large signal arrays for input/output matrices.
 - b. To use a generated core in a System Generator design, use array-based IO. This option provides a constant number of cycles of execution – 'constant throughput' – that is necessary for integration into a System Generator design.

Inputs / Outputs

Real-Only Data

Input(s)			
Name	Description	Type	Range
A	Input matrix to be factored. ABITS and AFBITS are the input quantization wordlength and fractional length as per A quantization parameters.	Real	Range = $[-2^{ABITS-1}$ to $2^{ABITS-1-1}]/2^{AFBITS}$

Output(s)			
Name	Description	Type	Range
Q	Output orthogonal factor matrix. QBITS and QFBITS are the Q matrix output wordlength and fractional length.	Real	Range = $[-2^{QBITS-1}$ to $2^{QBITS-1-1}]/2^{QFBITS}$
R	Output triangular factor matrix. RBITS and RFBITS are the R matrix output wordlength and fractional length.	Real	Range = $[-2^{RBITS-1}$ to $2^{RBITS-1-1}]/2^{RFBITS}$

Complex Data

Input(s)			
Name	Description	Type	Range
A_real, A_imag	Input matrix to be factored; separate real and imaginary parts of complex input matrix. ABITS and AFBITS are the input quantization wordlength and fractional length as per A quantization parameters.	Real	Range = $[-2^{ABITS-1}$ to $2^{ABITS-1}-1]/2^{AFBITS}$

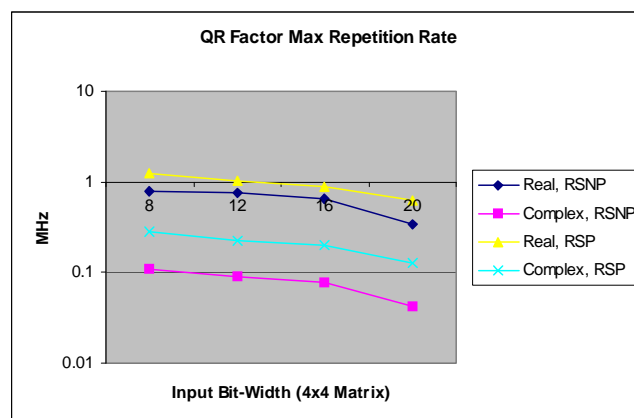
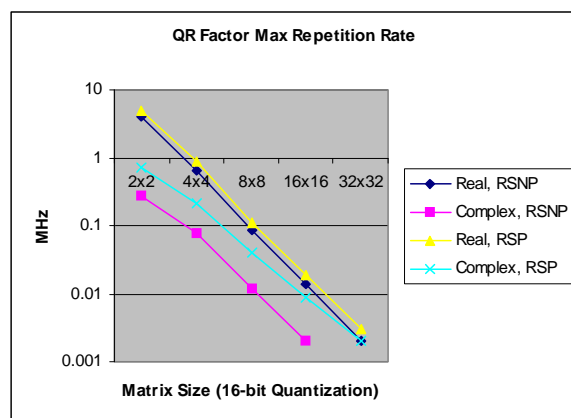
Output(s)			
Name	Description	Type	Range
Q_real, Q_imag	Output orthogonal factor matrix;separate real and imaginary parts of complex input matrix. QBITS and QFBITS are the Q matrix output wordlength and fractional length.	Real	Range = $[-2^{QBITS-1}$ to $2^{QBITS-1}-1]/2^{QFBITS}$
R_real, R_imag	Output triangular factor matrix;separate real and imaginary parts of complex input matrix. RBITS and RFBITS are the R matrix output wordlength and fractional length.	Real	Range = $[-2^{RBITS-1}$ to $2^{RBITS-1}-1]/2^{RFBITS}$

Expected Quality of Results

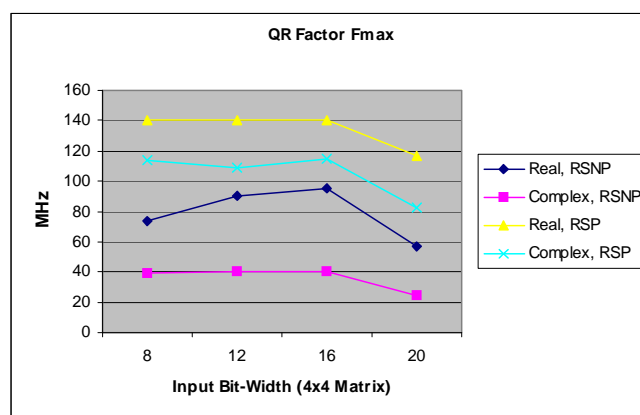
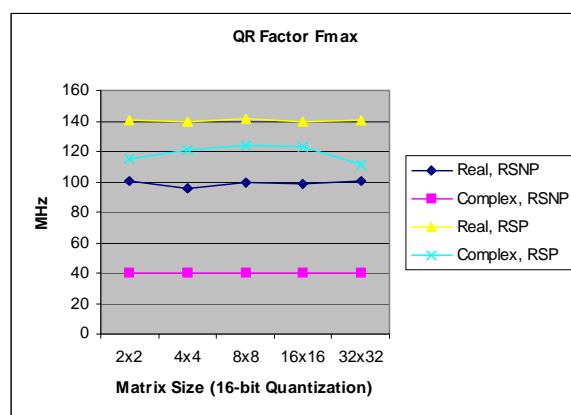
The following plots show typical speed and utilization of the qr_factor core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and NRSP refer to resource-shared, non-pipelined architectures and non-resource-shared, pipelined architectures respectively.

Speed of Operation

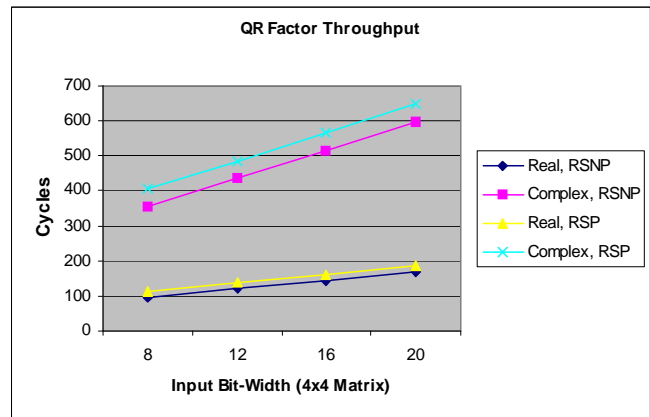
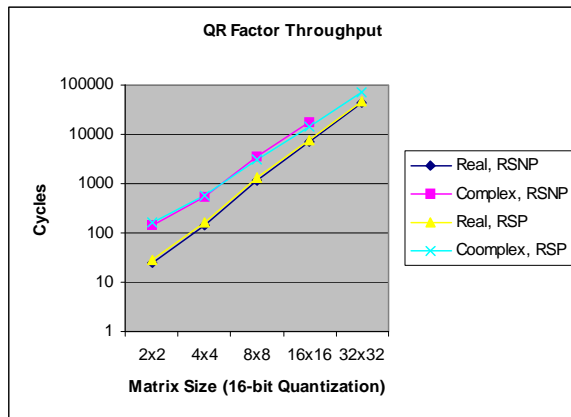
Factorization Repetition Rate



Maximum Operating Frequency

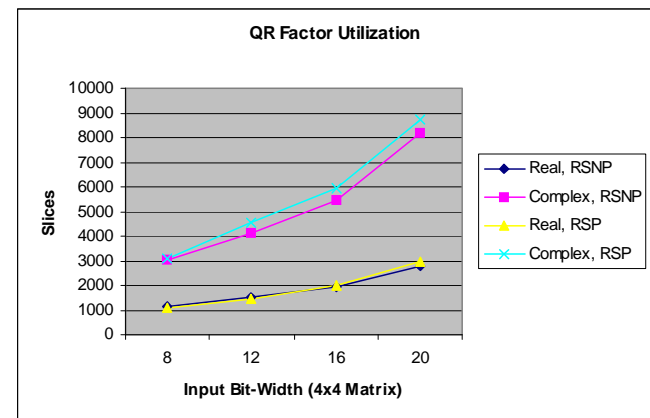
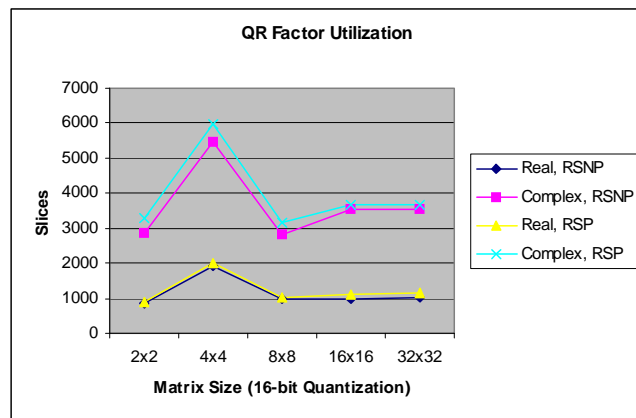


Hardware Clock Cycles per Design Function Call

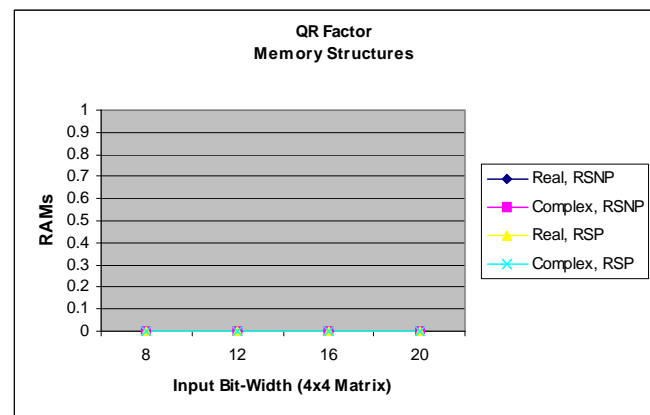
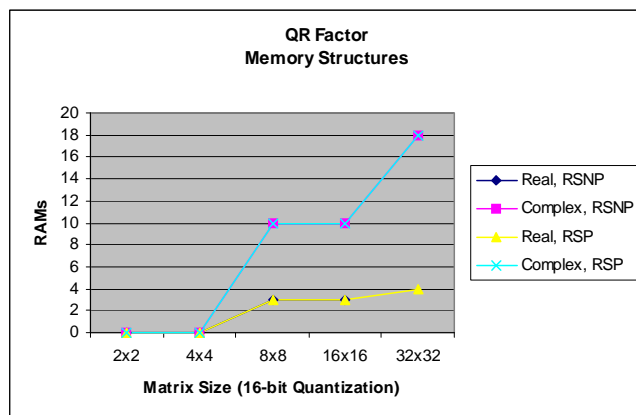


Utilization

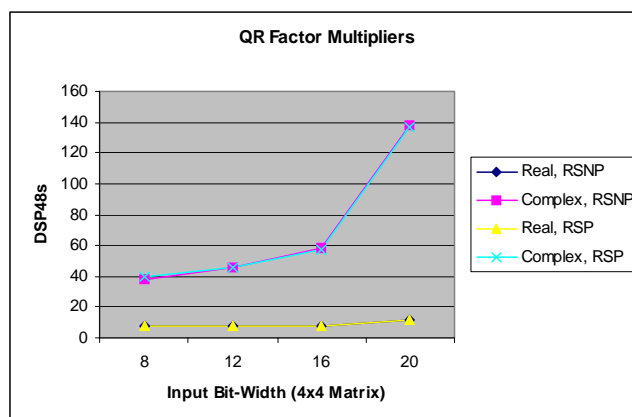
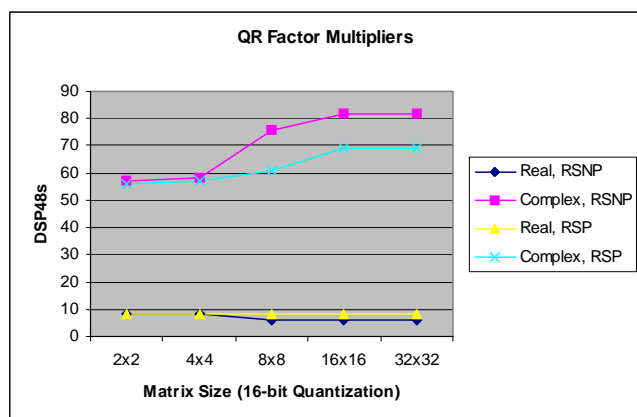
Number of Slices



Number of Memories



Number of Multipliers



accel_qr_inverse/accel_complex_qr_inverse

Description

Matrix inverse. Produces the inverse of a square input matrix. The implementation of the inverse computation is based on the triangular-orthogonal (QR) factorization of the input matrix followed by a product of the inverse of the matrix factors.

Related MATLAB Functions

[inv](#)

Differences with the MATLAB Function

The MATLAB `inv` function generates an indication when the input matrix is badly conditioned. The AccelDSP `qr_inverse` function produces an output that indicates when the input matrix is detected to be singular (non-invertible).

Syntax

Function Call	Supported MATLAB Syntax	Notes
<code>[Y, s] = accel_qr_inverse(X);</code>	<code>Y = inv(X);</code>	Syntax for real-valued input and output matrices; the output 's' indicates when the input matrix is singular (non-invertible)
<code>[Y_real, Y_imag, s] = accel_complex_qr_inverse(X_real, X_imag)</code>	<code>Y = inv(X);</code>	Syntax for complex-valued input and output matrices. The real and imaginary components are passed as separate matrices. ; the output 's' indicates when the input matrix is singular (non-invertible)

Parameters

Input quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is $-\pi$ to π . ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2π .	String	Range = [fixed ufixed] Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Range = [4 to 24] for real-only data Range = [4 to 16] for complex data
Fractional Length	The number of fractional bits for the input word(s).	String	Range = [0 to Word Length as per Mode parameter]

Implementation Parameters			
Name	Description	Type	Range
Implementation	Algorithm used for factorization implementation	String	Conventional Givens Rotations
Type	Input matrix data type.	String	Range = [double [complex: real and imag IO]

Implementation Parameters			
Name	Description	Type	Range
Matrix Size (1)	Input matrix size.	Integer	Range = [2 to 64] for real-only data Range = [2 to 32] for complex data
Data IO Format (2)	Selects the format for the IO streaming of data (array or sample per invocation);it affects the test design function wrapper only.	String	Range = [array sample] Default = [array]
Output Precision	Number of bits for output matrix.	Integer	Default = auto auto sets value based on input matrix word-length

Notes on Parameters:

1. Cores with matrix size of 8x8 or larger are automatically generated with memory mapping directives.
2. Sample-based IO streaming is more efficient in terms of the number of IO signals but there is a cost in the number of clock cycles used for the data transfers. The wrapper generated for sample-based streaming 'streams' data in and out at the boundaries of the design function automatically generated for a core. For cores with matrix size of 8x8 or larger, the generated wrapper includes memory mapping of the input and output matrix arrays for efficient resource utilization. The following two recommendations are for the user:
 - a. When generating cores of matrix size 8x8 or larger, use sample-based IO streaming to support efficient synthesis and mapping to FPGA devices. Not using this option can cause RTL synthesis to take a very long run time to complete or to fail due to an excessive number of logic resources consumed by the large signal arrays for input/output matrices.
 - b. To use a generated core in a System Generator design, use array-based IO. This option provides a constant number of cycles of execution – 'constant throughput' – that is necessary for integration into a System Generator design.

Inputs / Outputs

Real-Only Data

Input(s)			
Name	Description	Type	Range
X	Input matrix to be inverted. "word" and "fractional" are the input quantization wordlength and fractional length as per the Input Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$

Output(s)			
Name	Description	Type	Range
Y	Matrix Inverse. YBITS, YFBITS are the Y matrix output wordlength and fractional length automatically determined by the AccelDSP generator.	Real	Range = $[-2^{\text{YBITS}-1}$ to $2^{\text{YBITS}-1}-1]/2^{\text{YFBITS}}$
s	Singular matrix indication	Boolean	Range { 1 0 } 0 = (invertable matrix) 1 = (singular, non-invertable matrix)

Complex Data

Input(s)			
Name	Description	Type	Range
X_real, X_imag	Input matrix to be inverted. "word" and "fractional" are the input quantization wordlength and fractional length as per the Input Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$

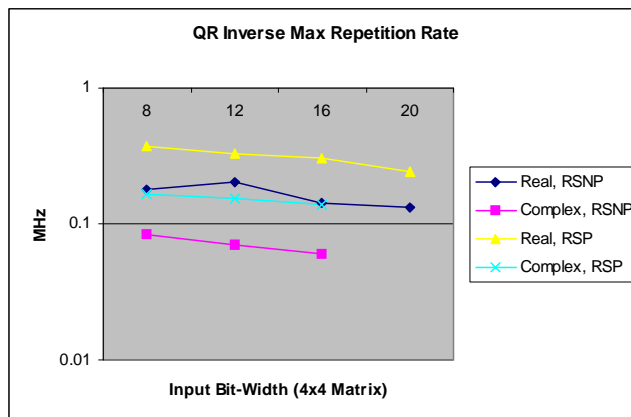
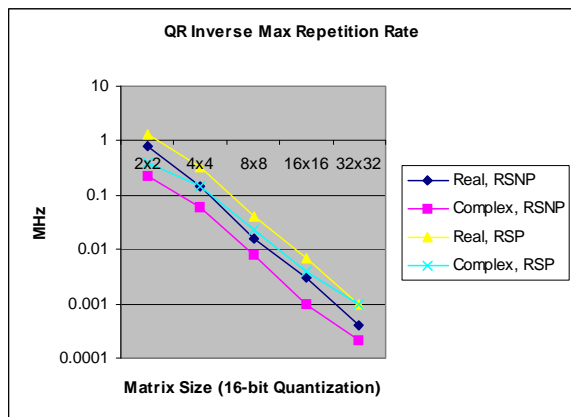
Output(s)			
Name	Description	Type	Range
Y_real, Y_imag	Matrix Inverse. YBITS, YFBITS are the Y matrix output wordlength and fractional length automatically determined by the AccelDSPgenerator.	Real	Range = $[-2^{YBITS-1}$ to $2^{YBITS-1}-1]/2^{YFBITS}$
s	Singular matrix indication	Boolean	Range { 1 0} 0 = (invertable matrix) 1 = (singular, non-invertable matrix)

Expected Quality of Results

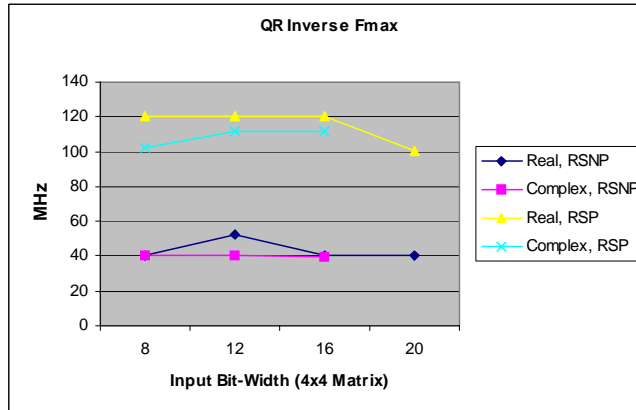
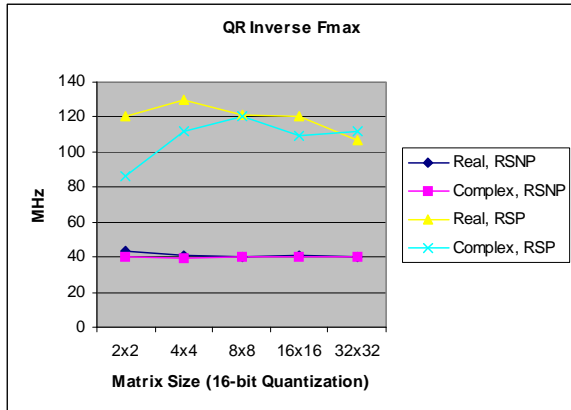
The following plots show typical speed and utilization of the `qr_inverse` core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and NRSP refer to resource-shared, non-pipelined architectures and non-resource-shared, pipelined architectures respectively.

Speed of Operation

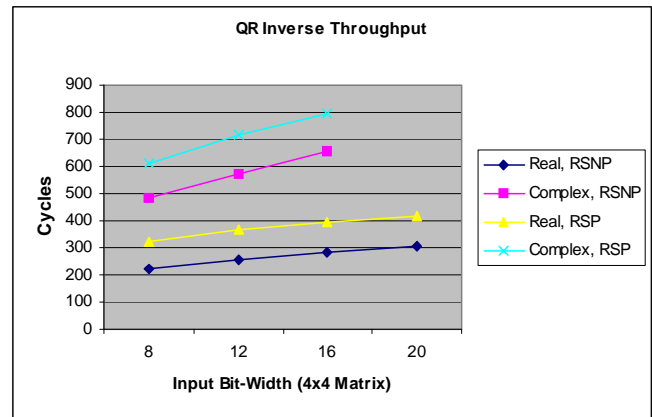
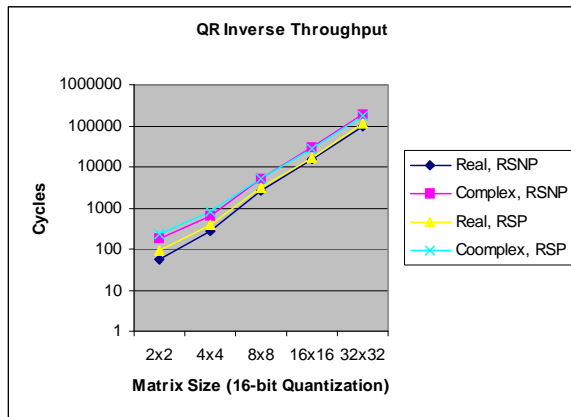
Inversion Repetition Rate



Maximum Operating Frequency

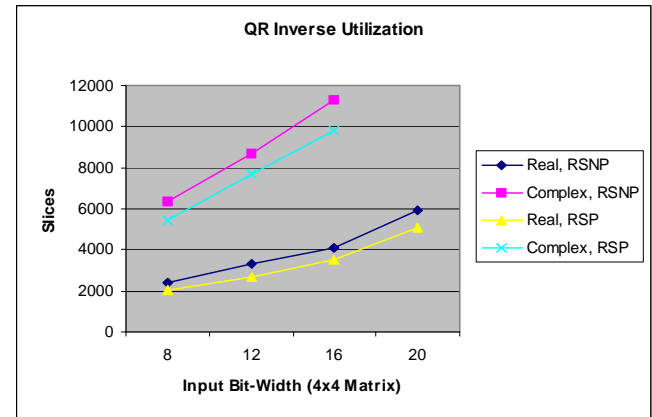
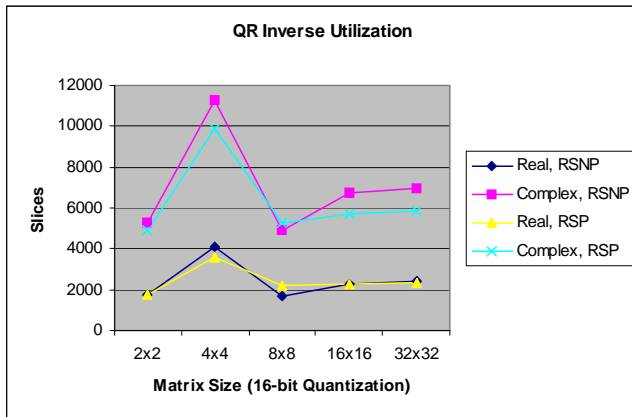


Hardware Clock Cycles per Design Function Call

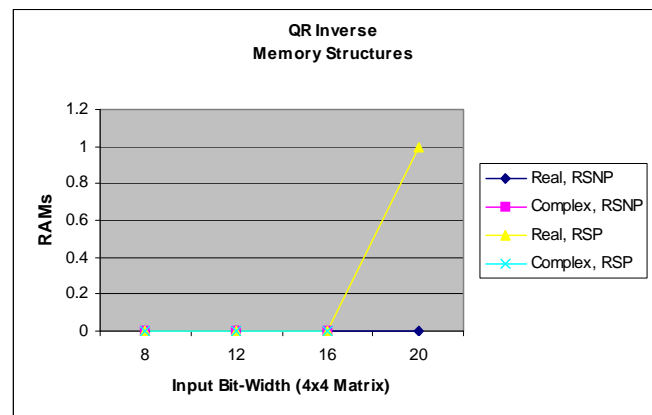
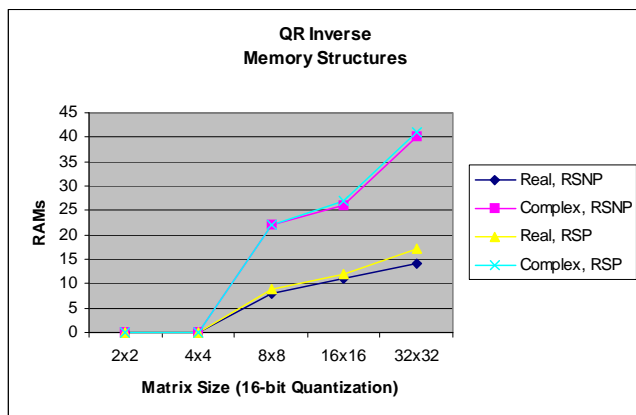


Utilization

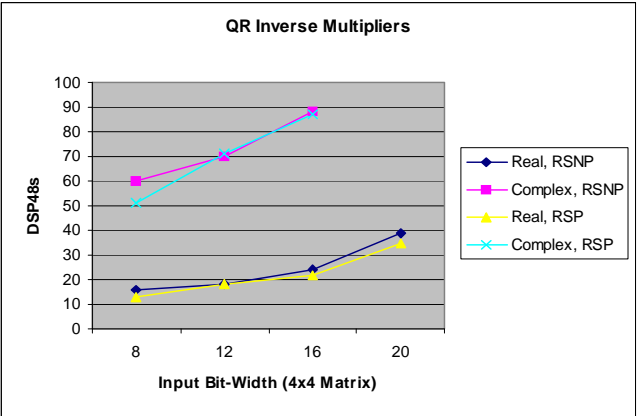
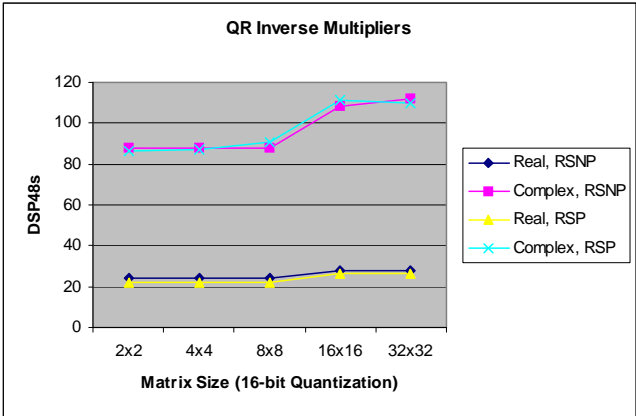
Number of Slices



Number of Memories



Number of Multipliers



accel_triang_inverse/accel_complex_triang_inverse

Description

The accel_triang_inverse reference design computes the inverse of a square, triangular matrix

Related MATLAB Functions

[inv](#)

Differences with the MATLAB Function

The MATLAB inv() function can invert triangular and non-triangular matrices. The AccelDSP accel_triang_inverse function is explicitly designed for operation on triangular matrices only.

The MATLAB inv() function generates a warning when the input matrix is badly scaled or nearly singular. The AccelDSP accel_triang_inverse function produces an output signal that indicates when the input matrix is detected to be singular (non-invertible).

Syntax

Function Call	Supported MATLAB Syntax	Notes
[Y, s] = accel_triang_inverse(X);	Y = inv(X);	Syntax for real-valued input and output matrices; the output 's' indicates when the input matrix is singular (non-invertible)
[Y_real, Y_imag, s] = accel_complex_triang_inverse(X_real, X_imag)	Y = inv(X);	Syntax for complex-valued input and output matrices. The real and imaginary components are passed as separate matrices. ; the output 's' indicates when the input matrix is singular (non-invertible)

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is $-PI$ to PI . ufixed : unsigned fixed-point mode. Infers that the input range is 0 to $2*PI$.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length(3)	The number of bits for the input word.	String	Range = [4 to 24] Default = 12
Fractional Length	The number of fractional bits for the input word(s).	String	Range = [0 to Word Length as per Mode parameter]

Implementation Parameters			
Name	Description	Type	Range
Implementation	Implementation type based on the triangular matrix orientation.	String	{upper-triangular lower-triangular} Default = upper-triangular
Type	Input data type	real	Range = [double [complex: real and imag IO] Default = double
Matrix Size (1)	Size of the input matrix.	Integer	Range = [2 to 32]
Data IO Format (2)	Selects the format for the IO streaming of data (array or sample per invocation);it affects the test design function wrapper only.	String	Range = [array sample] Default = [array]
Output Precision	Number of bits for the output matrix word-length.	Integer	Default = auto auto sets value based on input matrix word-length

Notes on Parameters:

1. Cores with matrix size of 8x8 or larger are automatically generated with memory mapping directives.
2. Sample-based IO streaming is more efficient in terms of the number of IO signals but there is a cost in the number of clock cycles used for the data transfers. The wrapper generated for sample-based streaming 'streams' data in and out at the boundaries of the design function automatically generated for a core. For cores with matrix size of 8x8 or larger, the generated wrapper includes memory mapping of the input and output matrix arrays for efficient resource utilization. The following two recommendations are for the user:
 - a. When generating cores of matrix size 8x8 or larger, use sample-based IO streaming to support efficient synthesis and mapping to FPGA devices. Not using this option can cause RTL synthesis to take a very long run time to complete or to fail due to an excessive number of logic resources consumed by the large signal arrays for input/output matrices.
 - b. To use a generated core in a System Generator design, use array-based IO. This option provides a constant number of cycles of execution – 'constant throughput' – that is necessary for integration into a System Generator design.
3. The triang_inverse core is generated with a top-level script for verification against the MATLAB 'inv' function using random matrices. It is possible that for some combination of small quantization word-length and large matrix size, verification will

fail because of singular matrices resulting from coarse quantization are used in the script. This can be avoided by using a larger input word-length.

Inputs / Outputs

Real-Only Data

Input(s)			
Name	Description	Type	Range
X	Input matrix to be inverted. "word" and "fractional" are the input quantization wordlength and fractional length as per the Input Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$

Output(s)			
Name	Description	Type	Range
Y	Matrix Inverse. YBITS, YFBITS are the Y matrix output wordlength and fractional length automatically determined by the AccelDSP generator.	Real	Range = $[-2^{\text{YBITS}-1}$ to $2^{\text{YBITS}-1}-1]/2^{\text{YFBITS}}$
s	Singular matrix indication	Boolean	Range { 1 0 } 0 = (invertable matrix) 1 = (singular, non-invertable matrix)

Complex Data

Input(s)			
Name	Description	Type	Range
X_real, X_imag	Input matrix to be inverted. "word" and "fractional" are the input quantization wordlength and fractional length as per the Input Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$

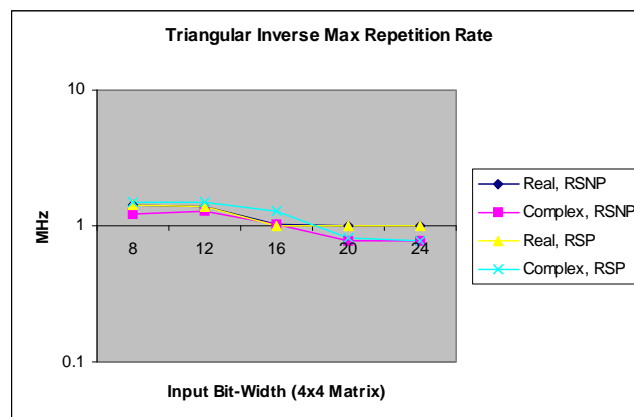
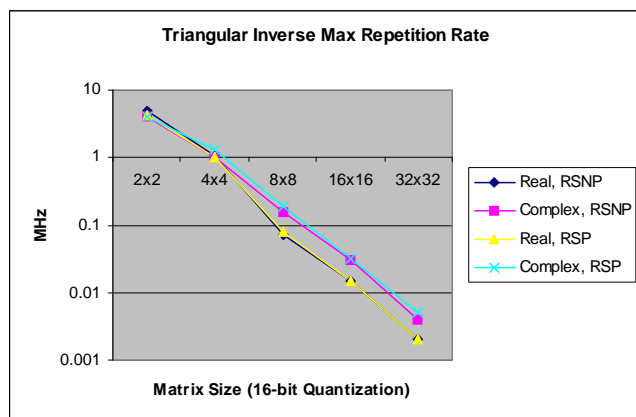
Output(s)			
Name	Description	Type	Range
Y_real, Y_imag	Matrix Inverse. YBITS, YFBITS are the Y matrix output wordlength and fractional length automatically determined by the AccelDSP generator.	Real	Range = $[-2^{YBITS-1}$ to $2^{YBITS-1}-1]/2^{YFBITS}$
s	Singular matrix indication	Boolean	Range { 1 0} 0 = (invertable matrix) 1 = (singular, non-invertable matrix)

Expected Quality of Results

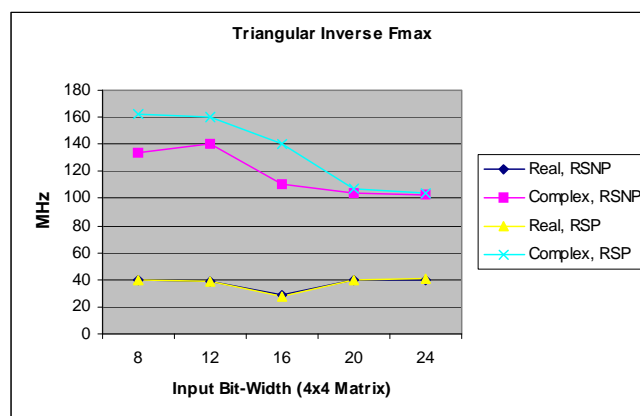
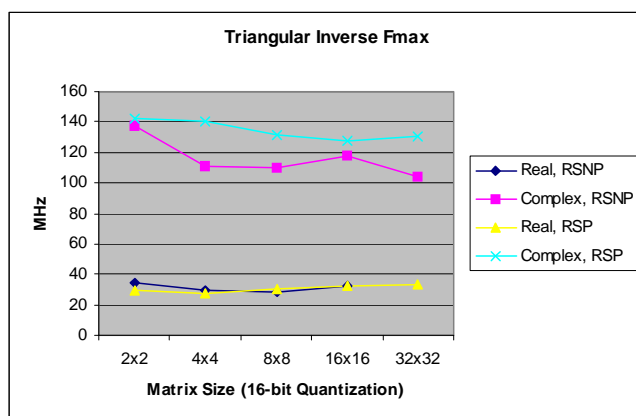
The following plots show typical speed and utilization of the triang_inverse core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and NRSP refer to resource-shared, non-pipelined architectures and non-resource-shared, pipelined architectures respectively.

Speed of Operation

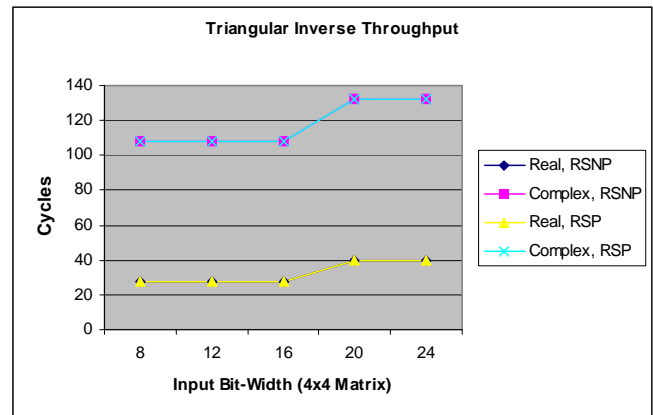
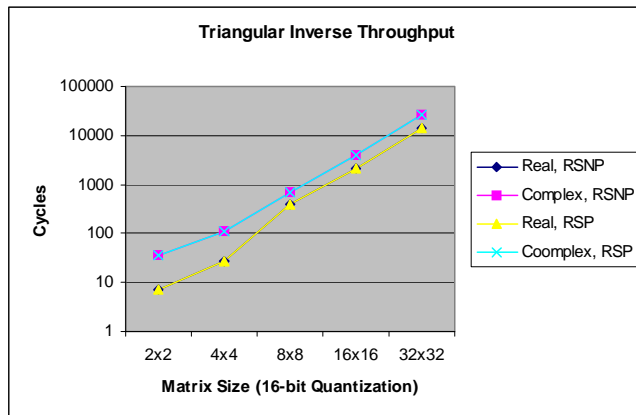
Inversion Repetition Rate



Maximum Operating Frequency

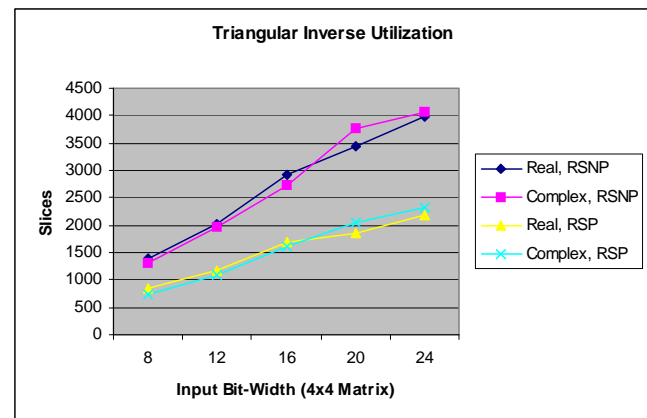
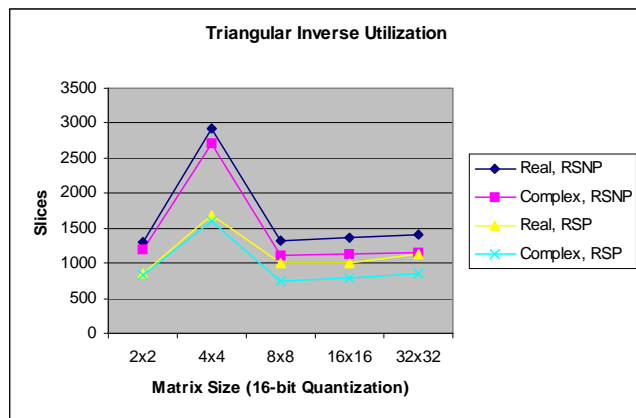


Hardware Clock Cycles per Design Function Call

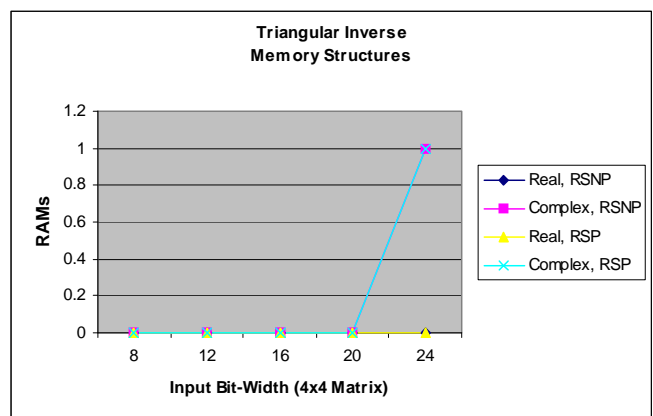
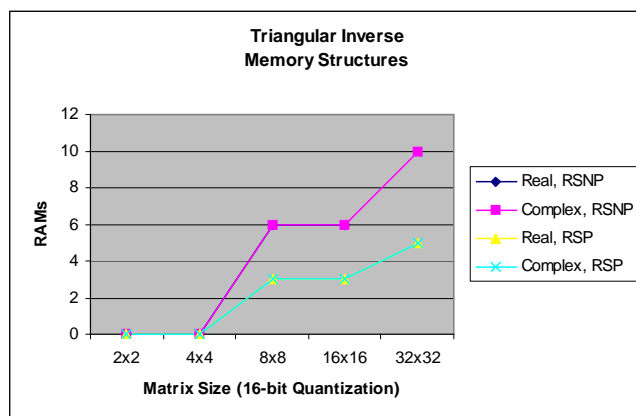


Utilization

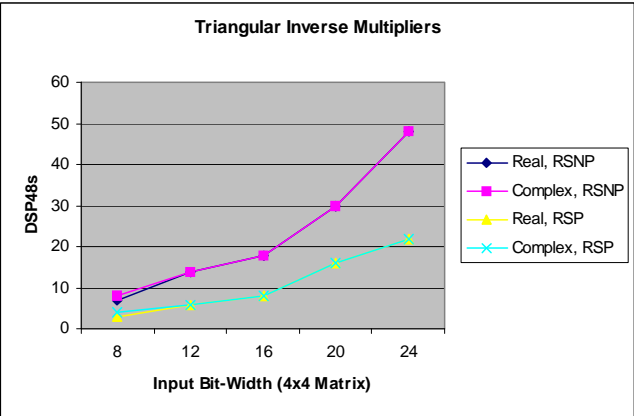
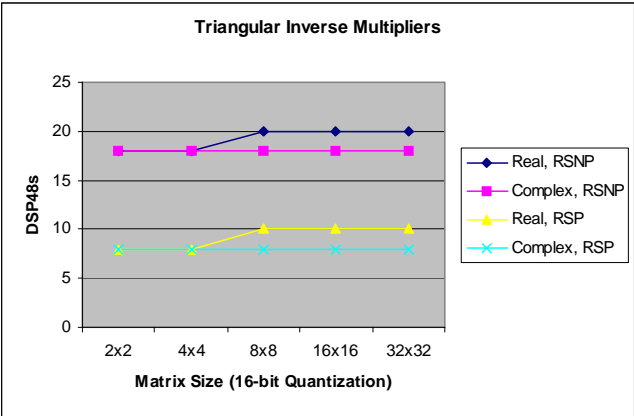
Number of Slices



Number of Memories



Number of Multipliers



accel_triang_solver

Description

The AccelDSP triang_solver computes the solution to an upper or lower triangular system of equations using backward or forward substitution, respectively. The input-output relationship can be stated as follows.

Given the systems of equations

$$\mathbf{Ax} = \mathbf{b}$$

where the coefficient matrix \mathbf{A} is square, triangular, and with real-valued diagonal elements. The triang_solver computes the solution

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

using backward or forward substitution.

Related MATLAB Functions

none

Syntax

Function Call	Notes
[x,s] = accel_triang_solver(A,b); A is real-valued.	The core is auto-inferred when the MATLAB signature is detected. A is assumed to be triangular (upper or lower) with <u>real-valued diagonal elements</u> . The output 's' is set to 1 when the input matrix is detected to be singular. In this case, the output x is not valid. 's' is 0 otherwise.
[x] = accel_triang_solver(A,b); A is real-valued.	The core is auto-inferred when the MATLAB signature is detected. A is assumed to be triangular (upper or lower) The output x is not valid when the input matrix A is singular.

Function Call	Notes
<pre>[x_real, x_imag, s] = accel_complex_triangular_solver(A_real, A_imag, b_real, b_imag);</pre> <p>The input matrix and the input and output vectors are complex with separate real and imaginary parts.</p>	<p>The core is auto-inferred when the MATLAB signature is detected.</p> <p>The input matrix is assumed to be triangular (upper or lower) with <u>real-valued diagonal elements</u>.</p> <p>The output 's' is set to 1 when the input matrix is detected to be singular. In this case, the output x is not valid. 's' is 0 otherwise.</p>
<pre>[x_real, x_imag] = accel_complex_triangular_solver(A_real, A_imag, b_real, b_imag);</pre> <p>The input matrix and the input and output vectors are complex with separate real and imaginary parts.</p>	<p>The core is auto-inferred when the MATLAB signature is detected.</p> <p>The input matrix is assumed to be triangular (upper or lower).</p> <p>The output x is not valid when the input matrix A is singular.</p>

Parameters

Coefficient Matrix Quantizer		
Name	Description	Range
Sign Mode	<p>fixed: signed fixed-point model. Infers that the input range is -PI to PI.</p> <p>ufixed: unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.</p>	Mode = fixed ufixed Default = fixed
Round Mode	<p>floor: round to the closest representable number in the direction of negative infinity</p> <p>nearest: round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.</p>	String
Overflow Mode	<p>wrap: wrap on overflow.</p> <p>saturate: saturate a maximum value on overflow.</p>	OverflowMode = [wrap saturate] Default = wrap

Coefficient Matrix Quantizer		
Name	Description	Range
Word Length	The number of bits for the input word.	Range = [-4 to 24] Default = 12
Fractional Length	The number of fractional bits for the input word(s).	Range = [0 to Word Length as per Mode parameter] Default = 11

RHS Vector Quantizer		
Name	Description	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	Mode = fixed ufixed Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	Range = [-4 to 24] Default = 12
Fractional Length	The number of fractional bits for the input word(s).	Range = [0 to Word Length as per Mode parameter] Default = 11

Implementation Parameters		
Name	Description	Range
Implementation	Implementation type.	[upper-triangular lower-triangular] Default = [upper-triangular]
Type	Input data type.	Range={double complex: separate real / imag I/O} Default = double
Matrix Size (2:32)	Input matrix size.	range = [2 to 32] Default = [4]
Data IO Format	Selects the format for the IO streaming of data (array or sample per invocation);it affects the test design function wrapper only.	Range = [array sample] Default = [array]
Output Precision	Number of bits for the output matrix.	Default = [auto] auto sets the value based on the input matrix word-length

Notes on Parameters

1. Cores with a matrix size of 8x8 or larger are automatically generated with memory mapping directives.
2. Sample-based IO streaming is more efficient in terms of the number of IO signals, but there is a cost in the number of clock cycles used for the data transfers. A wrapper is automatically generated for sample-based streaming and 'streams' data in and out at the boundaries of the design function. For cores with matrix size of 8x8 or larger, the generated wrapper includes memory mapping of the input and output matrix arrays for efficient resource utilization. There are two recommendations to the user
 - a. When you generate cores of matrix size 8x8 or larger, you should use sample-based IO streaming to support efficient synthesis and mapping to FPGA devices. Not using this option can cause RTL synthesis to take a very long time to complete or to fail due to the excessive number of logic resources consumed by the large signal arrays for input/output matrices.
 - b. To use a generated core in a System Generator design, use array-based IO. This option provides a constant number of cycles of execution – 'constant throughput' – that is necessary for the integration into a System Generator design.
3. The triang_solver core is generated with a top-level script for verification against the MATLAB '\ ' operator. This script uses the gallery() function to generate triangular matrices with random singular values with a predetermined condition number for testing. Quantization of these matrices can render them singular especially when using small word-lengths combined with larger matrix sizes. It is possible that for some combination of small quantization word-length and large matrix size verification will fail because non-positive matrices are used in the script. This can be avoided by using a larger input word-length. Word-lengths of 14 or more bits should not cause such a failure even with 8x8 or larger matrices.

Inputs / Outputs

Real-Only Data

accel_triang_solver Input(s)			
Name	Description	Type	Range
A	Coefficient Input matrix "word" and "fractional" are the input quantization wordlength and fractional length as per the Coefficient Matrix Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$
b	The Right-Hand Side vector "word" and "fractional" are the input quantization wordlength and fractional length as per the RHS Vector Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$

accel_triang_solver Output(s)			
Name	Description	Type	Range
x	The solution. xBITS, xFBITS are the x vector output wordlength and fractional length automatically determined.	Real	Range = $[-2^{\text{xBITS}-1}$ to $2^{\text{xBITS}-1}-1]/2^{\text{xFBITS}}$
s	Singular matrix indication	Boolean	Range { 1 0 } 0 = (invertable matrix) 1 = (singular, non-invertable matrix)

accel_complex_triang_solver Input(s)			
Name	Description	Type	Range
A_real	Coefficient Input matrix “word’ and “fractional” are the input quantization wordlength and fractional length as per the Coefficient Matrix Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{-\text{fractional}}$
A_imag	Coefficient Input matrix “word’ and “fractional” are the input quantization wordlength and fractional length as per the Coefficient Matrix Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{-\text{fractional}}$
b_real	The Right-Hand Side vector “word’ and “fractional” are the input quantization wordlength and fractional length as per the RHS Vector Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{-\text{fractional}}$
b_imag	The Right-Hand Side vector “word’ and “fractional” are the input quantization wordlength and fractional length as per the RHS Vector Quantizer parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{-\text{fractional}}$

accel_complex_triang_solver Output(s)			
Name	Description	Type	Range
x_real	The solution. xBITS, xFBITS are the x vector output wordlength and fractional length automatically determined.	Real	Range = $[-2^{\text{xBITS}-1}$ to $2^{\text{xBITS}-1}-1]/2^{\text{xFBITS}}$
x_imag	The solution. xBITS, xFBITS are the x vector output wordlength and fractional length automatically determined.	Real	Range = $[-2^{\text{xBITS}-1}$ to $2^{\text{xBITS}-1}-1]/2^{\text{xFBITS}}$
s	Singular matrix indication	Boolean	Range { 1 0} 0 = (invertable matrix) 1 = (singular, non-invertable matrix)

Memory Mapping

In general, the triang_solver core will use memory mapping when the matrix dimension is equal to or greater than 8x8. See note on 'Data IO Format' parameter for effective synthesis of cores for matrices 8x8 or larger.

Quantization

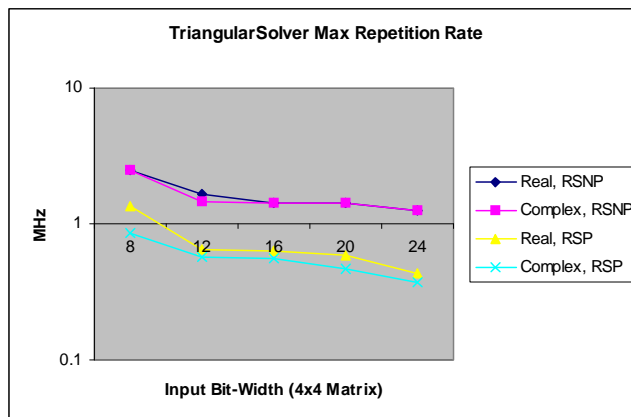
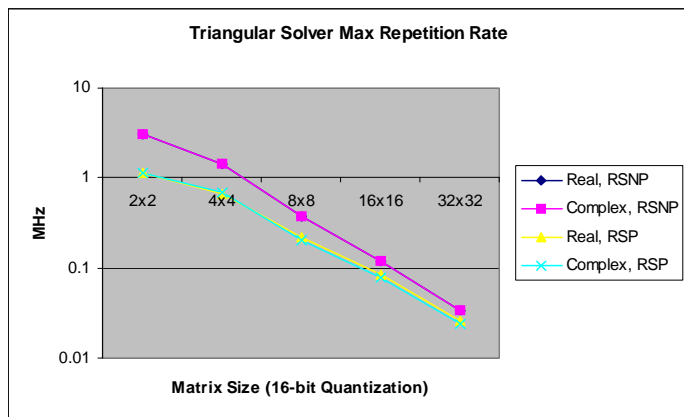
The triang_solver core generator calculates the quantization parameters for key variables in the factorization algorithm to insure proper numerical precision. These parameters are applied with quantization directives in a hierarchical directives file.

Expected Quality of Results

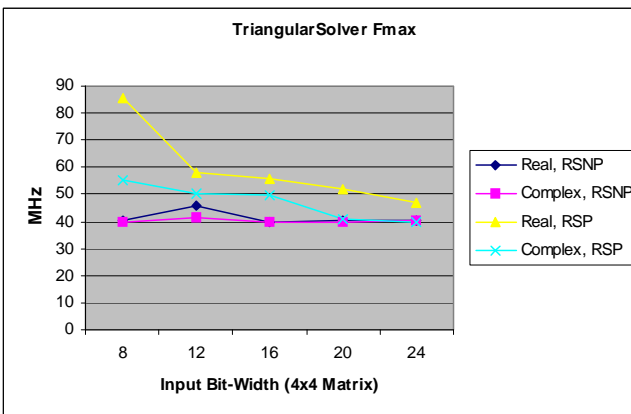
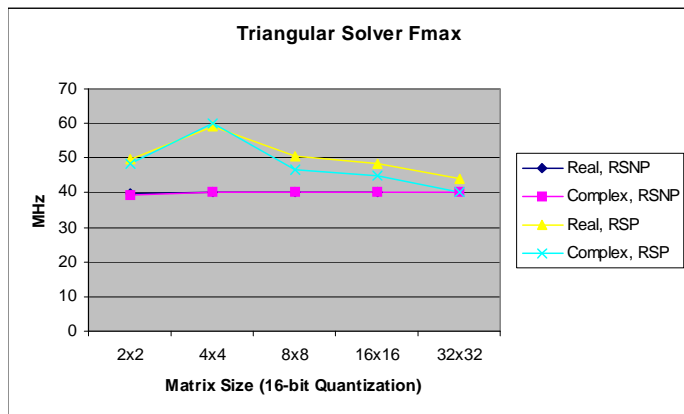
The following plots show typical speed and utilization of the triang_inverse core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and NRSP refer to resource-shared, non-pipelined architectures and non-resource-shared, pipelined architectures respectively.

Speed of Operation

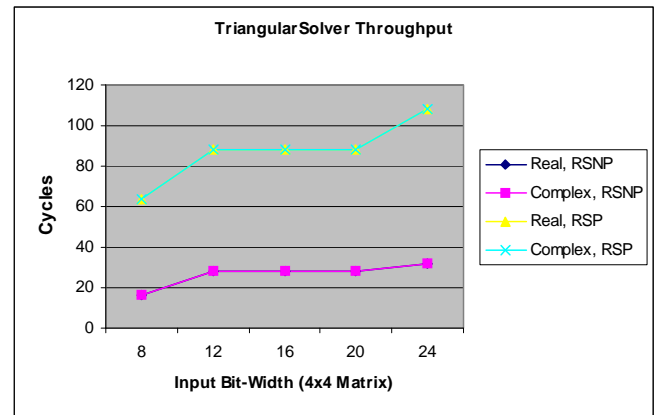
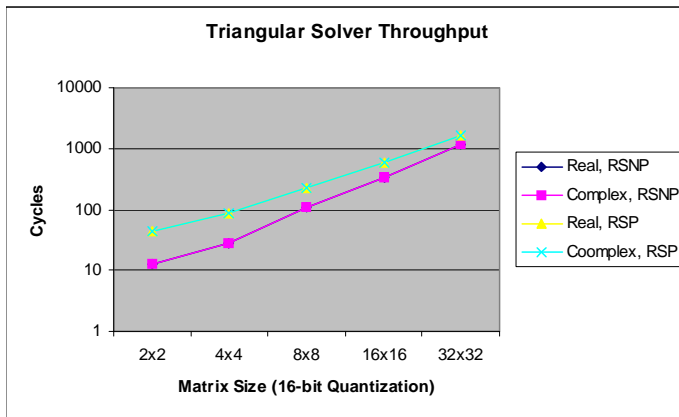
Solution Repetition Rate



Maximum Operating Frequency

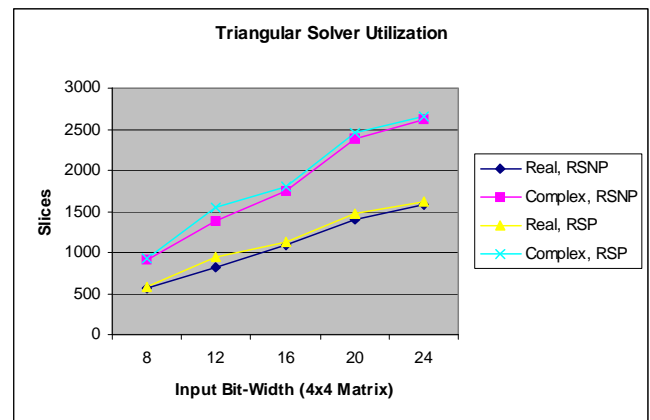
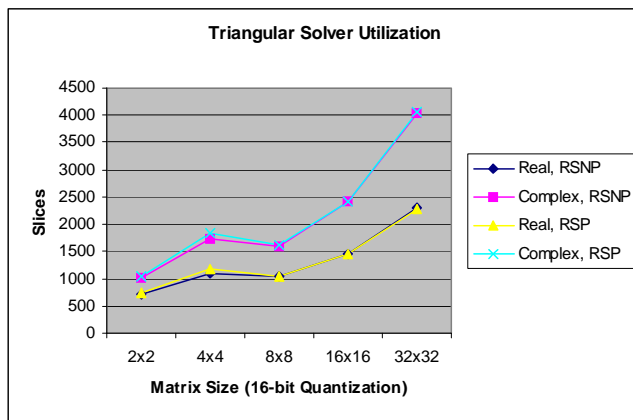


Hardware Clock Cycles per Design Function Call

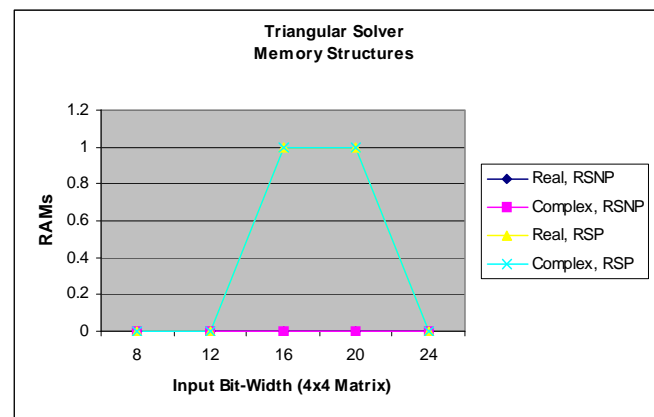
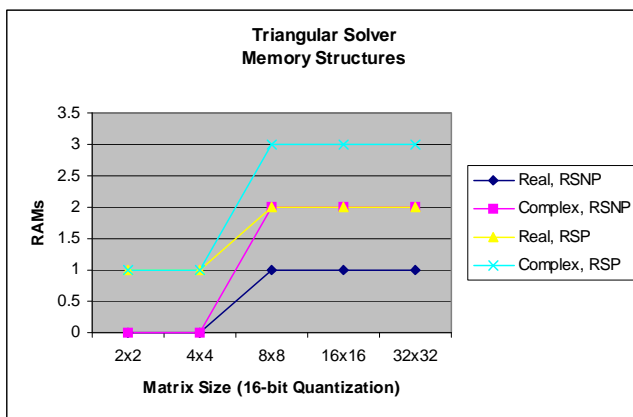


Utilization

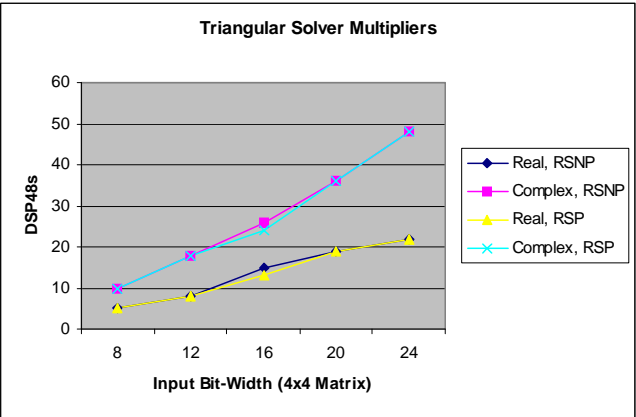
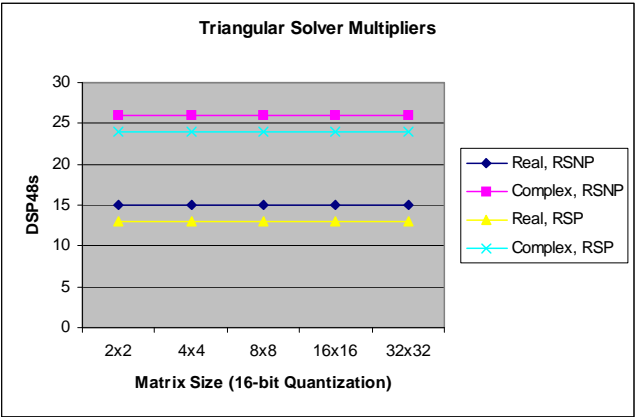
Number of Slices



Number of Memories



Number of Multipliers



acos/accel_cos

Description

Returns the Inverse Cosine for each element of the input.

Related MATLAB Functions

[acos](#)

Differences with the MATLAB Function

The shape of the input is limited to a scalar, vector or 2-D matrix.

Extended Features of accel_acos()

The accel_acos() function provides the same functionality as the cos() function with the following extended features:

1. The accel_acos() function can create an optional asin() output with little additional hardware.
2. The accel_acos() function can specify input units of measure ([InputType](#)) in the following formats:
 - ♦ [native form](#)
 - ♦ [scaled form](#)

Syntax

AcceIDSP acos() Function Call	Supported MATLAB Syntax
y = acos(x);	y = acos(x);
AcceIDSP accel_acos() Function Call(s)	Supported MATLAB Syntax
y = accel_acos(x); [y,z] = accel_acos(x);	y = acos(x); z = asin(x);

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 7

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
InputType	Specify the representation of the input word.	String	{ native form scaled form } Default = native form

Implementation Parameters			
Name	Description	Type	Range
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 100] Default = [auto]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.		Range = 1 to 14 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 1 to 16 Default = auto
Output bitwidth	The number of bits in the output word(s).		Range 4 to 32 Default = auto
Include asin()	A asin() output can be created with little additional hardware.	String	{yes no} Default = {no}

Inputs / Outputs

acos() Input(s)			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix.	Real	Range = [0 to 2π] or [$-\pi$ to π] depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

accel_acos() Input(s)			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix. (For InputType = native form).	Real	Range = $[0 \text{ to } 2\pi]$ or $[-\pi \text{ to } \pi]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.
	May be a scalar, vector or 2-D matrix. (For InputType = scaled form).	Positive Integer	Range = $[0 \text{ to } 2^{\text{IBITS}-1}]$ or $[-2^{\text{IBITS}-1} \dots 2^{\text{IBITS}-1}-1]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

acos() Output(s)			
Name	Description	Type	Range
y	May be a scalar, vector, or 2-D matrix representing the acos of the corresponding input element. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OBITS}}$

Additional accel_acos() Output			
Name	Description	Type	Range
z	An optional output scalar, vector, or 2-D matrix representing the asin of the corresponding input. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OBITS}}$

all

Supported for scalars, vectors or two-dimensional matrices .

angle/accel_angle

Description

Computes the phase angle for a complex number.

Related MATLAB Functions

[angle](#)

Differences with the MATLAB Function

The shape of the input is limited to scalars, vectors and 2-D matrices.

Features of the accel_angle() Function

1. The accel_angle() function provides an angle() function with a complex input:
2. Accepts two values instead of a single complex value ($x=a+jb$).
3. Offers an optional abs() output with little additional hardware.
4. Offers a normalized output with range $\{-1 \text{ to } 1\}$ instead of $\{-\pi \text{ to } \pi\}$.

Syntax

AccelDSP angle() Function Call	Supported MATLAB Syntax
<code>y = angle(x);</code>	<code>y = angle(x);</code>
AccelDSP accel_angle() Function Calls	Supported MATLAB Syntax
<code>y = accel_angle(a,b);</code> <code>[y,z] = accel_angle(a,b);</code>	<code>y = angle(x);</code> <code>z = abs(x);</code>

Parameters

Input quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	{CORDIC}
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the larger of Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB.	Integer	[4 to 32] Default = [auto]
angle_output	The output can be returned in radians $[-\pi$ to π] or normalized to [-1 to 1].	String	{Radians Normalized} Default = {Radians}
Output bitwidth	Number of bits used to represent the angle() output y.	Positive Integer	Range = [4 to 32] Default = auto
Include abs() output	An abs() output can be created with little additional hardware.	String	{no yes} Default = {no}

Inputs / Outputs

angle() Input(s)			
Name	Description	Type	Range
x	An input matrix representing real(x) each IBITS-bits wide with IFBITS-bits representing the fractional portion.	Real	Range = $[-2^{IBITS-1}$ to $2^{IBITS-1}-1]/2^{IFBITS}$

accel_angle() Input(s)			
Name	Description	Type	Range
a	An input matrix representing real(x) each IBITS-bits wide with IFBITS-bits representing the fractional portion.	Real	Range = $[-2^{IBITS-1}$ to $2^{IBITS-1}-1]/2^{IFBITS}$
b	An input matrix representing imag(x) each IBITS-bits wide with IFBITS-bits representing the fractional portion.	Real	Range = $[-2^{IBITS-1}$ to $2^{IBITS-1}-1]/2^{IFBITS}$

angle() Output(s)			
Name	Description	Type	Range
y	An output matrix with each element OBITS-bits wide.	Real	Function of the Input Quantizer parameters.

Additional accel_angle() Output			
Name	Description	Type	Range
z	An optional Abs() output	Real	Function of the Input Quantizer parameters.

any

Supported for scalars, vectors or two-dimensional matrices .

asin/accel_asin

Description

Returns the inverse sine of the input matrix.

Related MATLAB Functions

[asin](#)

Differences with the MATLAB Function

1. The inputs must be pre-quantized to the accuracy specified by the input quantizer.
2. The shape of the input is limited to a scalar, vector or 2-D matrix.

Extended Features of accel_asin()

The `accel_asin()` function provides the same functionality as the `asin()` function plus the following extended features:

1. The `accel_asin()` function can create an optional `acos()` output with little additional hardware.
2. The `accel_asin()` function can specify input units of measure ([InputType](#)) in the following formats:
 - ♦ [native form](#)
 - ♦ [scaled form](#)

Syntax

AccelDSP asin() Function Call	Supported MATLAB Syntax
<code>y = asin(x);</code>	<code>y = asin(x);</code>
AccelDSP accel_asin() Function Call	Supported MATLAB Syntax
<code>y = accel_asin(x);</code> <code>[y,z] = accel_asin(x);</code>	<code>y = sin(x)</code> <code>z = cos(x);</code>

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 7

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
InputType	Specify the representation of the input word.	String	{ native form scaled form } Default = native form

Implementation Parameters			
Name	Description	Type	Range
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 100] Default = [auto]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.		Range = 1 to 14 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 1 to 16 Default = auto
Output bitwidth	The number of bits in the output word(s).		Range 4 to 32 Default = auto
Include acos()	A acos() output can be created with little additional hardware.	String	{yes no} Default = {no}

Inputs / Outputs

asin() Input(s)			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix.	Real	Range = [0 to 2π] or [$-\pi$ to π] depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

accel_asin() Input(s)			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix. (For InputType = native form).	Real	Range = $[0 \text{ to } 2\pi]$ or $[-\pi \text{ to } \pi]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.
	May be a scalar, vector or 2-D matrix. (For InputType = scaled form).	Positive Integer	Range = $[0 \text{ to } 2^{\text{IBITS}-1}]$ or $[-2^{\text{IBITS}-1} \dots 2^{\text{IBITS}-1}-1]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

asin() Output(s)			
Name	Description	Type	Range
y	May be a scalar, vector, or 2-D matrix representing the asin of the corresponding input element. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

Additional accel_asin() Output			
Name	Description	Type	Range
z	An optional output scalar, vector, or 2-D matrix representing the acos of the corresponding input. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

atan/accel_atan

Description

Returns the inverse tangent of the input matrix.

Related MATLAB Functions

[atan](#)

Differences with the MATLAB Function

1. The inputs must be pre-quantized to the accuracy specified by the input quantizer.

- The shape of the input is limited to a scalar, vector or 2-D matrix.

Extended Features of accel_atan()

The accel_atan() function provides the same functionality as the atan() function plus the following extended features:

- The accel_atan() function can specify input units of measure (**InputType**) in the following formats:
 - native form
 - scaled form

Syntax

AccelDSP atan() Function Call	Supported MATLAB Syntax
y = atan(x);	y = atan(x);
AccelDSP accel_atan() Function Call	Supported MATLAB Syntax
y = accel_atan(x);	y = atan(x);

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap

Input Quantization			
Name	Description	Type	Range
Word Length	The number of bits for the input word.	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 9

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
InputType	Specify the representation of the input word.	String	{ native form scaled form } Default = native form
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 100] Default = [auto]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimize the table sizes.		Range = 1 to 14 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 1 to 16 Default = auto
Output bitwidth	The number of bits in the output word(s).		Range 4 to 32 Default = auto

Inputs / Outputs

atan() Input			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix.	Real	Range = $[0 \text{ to } 2\pi]$ or $[-\pi \text{ to } \pi]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

accel_atan() Input			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix. (For InputType = native form).	Real	Range = $[0 \text{ to } 2\pi]$ or $[-\pi \text{ to } \pi]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.
	May be a scalar, vector or 2-D matrix. (For InputType = scaled form).	Positive Integer	Range = $[0 \text{ to } 2^{\text{IBITS}-1}]$ or $[-2^{\text{IBITS}-1} \dots 2^{\text{IBITS}-1}]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

atan() Output			
Name	Description	Type	Range
y	May be a scalar, vector, or 2-D matrix representing the sine of the corresponding input element. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1} - 1] / 2^{\text{OBITS}}$

atan2/accel_atan2

Description

Four-quadrant inverse tangent

Related MATLAB Functions

[atan2](#)

Differences with the MATLAB Function

1. The inputs must be pre-quantized to the accuracy specified by the Input Quantizer parameters.
2. The shape of the input is limited to scalars, vectors and 2-D matrices.

Extended Features of accel_atan2()

The accel_atan2() function provides the same functionality as the atan2() function plus the following extended features:

1. Offers a normalized output with range $\{-1 \text{ to } 1\}$ instead of $\{-\pi \text{ to } \pi\}$.
2. Offers an optional abs() output with little additional hardware.

Syntax

AccelDSP atan2() Function Call	Supported MATLAB Syntax
<code>p = atan2(y,x);</code>	<code>p = atan2(y,x);</code>

AccelDSP accel_atan2() Function Call(s)	Supported MATLAB Syntax
<code>p = accel_atan2(y,x);</code> <code>[p,z] = accel_atan2(y,x);</code>	<code>p = atan2(y,x);</code> <code>z = abs(x+jy);</code>

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap

Input Quantization			
Name	Description	Type	Range
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	{CORDIC}
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth and Angle OBITS to compute the number of iterations required to ensure the maximum error is ~1 LSB.	Integer	[4 to 32] Default = [auto]
angle_output	The output can be returned in radians $[-\pi$ to π] or normalized to [-1 to 1].	String	{Radians Normalized} Default = {Radians}
Output bitwidth	Number of bits used to represent the output.	Positive Integer	Range = [4 to 32] Default = auto
Include abs() output	A abs() output can be created with little additional hardware.	String	{no yes} Default = {no}

Inputs / Outputs

atan2() Input(s)			
Name	Description	Type	Range
y	An input matrix representing real(y) each IBITS-bits wide with IFBITS-bits representing the fractional portion. IBITS = Input Word Length and IFBITS = Input Fractional Length.	Real	Specified by the Input Quantizer.
x	An input matrix representing real(x) each IBITS-bits wide with IFBITS-bits representing the fractional portion. IBITS = Input Word Length and IFBITS = Input Fractional Length.	Real	Specified by the Input Quantizer.

atan2() Output			
Name	Description	Type	Range
p	An output matrix with each element OBITS-bits wide.	Real	Function of the SigInQ parameters

Additional atan2() Output			
Name	Description	Type	Range
z	An optional abs() output.	Real	Function of the SigInQ parameters

bchdec

Communications Toolbox function.

bchenc

Communications Toolbox function.

bi2de

Supported for synthesis. First argument can be a variable representing a scalar, vector or two-dimensional matrix. The second argument specifies the numeric base of the first argument and must be an unsigned constant integer. The second argument is currently

restricted to 2. The third parameter <flg> is supported where <flg> can be a string equal to 'right-msb' or 'left-msb'. The value 'right-msb' produces the default behavior.

bitand

Supported for scalars, vectors and two-dimensional matrices.

Extended capability is provided by calling the function [accel_bitand](#).

bitcmp

Overloaded with [accel_bitcmp](#).

Extended capability is provided by calling the function [accel_bitcmp](#).

bitget

Supported for scalars, vectors and two-dimensional matrices.

bitor

Supported for scalars, vectors and two-dimensional matrices.

Extended capability is provided by calling the function [accel_bitor](#).

bitset

Supported for scalars, vectors and two-dimensional matrices.

bitshift

Overloaded with [accel_bitshl](#) and [accel_bitshr](#).

bitxor

Supported for scalars, vectors and two-dimensional matrices.

Extended capability is provided by calling the function [accel_bitxor](#).

cart2pol

Supported for scalars, vectors and two-dimensional matrices containing both positive and negative numbers.

case

Supported as part of the switch statement.

Example

switch (x)

```
case (0)
    y = y + 1;
case (1)
    y = y + 2;
end
```

cat

Supported for synthesis.

ceil

Supported for synthesis.

Example 2

```
y = ceil(x);
```

char

Create a string constant inside the design function. Refer to [“char' Data Type,”](#) for details.

chol

Supported function.

chol_inverse/accel_complex_chol_inverse

Description

Generates an output matrix Y which is the inverse of a symmetric positive definite input matrix X using Cholesky factorization. The input-output relationship can be stated as follows.

$$Y = X^{-1} = (R' * R)^{-1}$$

Where R is a Cholesky factor of X.

Related MATLAB Functions

[inv](#)

[Cholesky Inversion Block \(Simulink\)](#)

Differences in Operation to MATLAB Functions

The MATLAB inv function can operate on input matrices with complex data. The AccelDSP chol_inverse() function accepts two separate values (real, imag) instead of a single complex value (x=a+jb) and output two separate values (real, imag) instead of a single complex value.

The MATLAB inv function generates an indication when the input matrix is badly conditioned. The AccelDSP chol_inverse function produces a second output which indicates the invertibility of the input matrix based on the detection of non-positive definiteness during Cholesky factorization.

Syntax

Function Call	Supported MATLAB Syntax
[Y,s] = chol_inverse(X);	Y = inv(X);
accel_complex_chol() Func Call	Supported MATLAB Syntax
[Y_real, Y_imag, s] = accel_complex_chol_inverse(X_real, X_imag)	Y = inv(X);

Parameters

Input quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Range = [2 to 20] Default = 16
Fractional Length	The number of fractional bits for the input word(s).	String	Range = [0 to Word Length as per Mode parameter] Default = 15

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	Multiplier
Type	Input matrix data type.	String	double
Matrix Size (2:32)	Input matrix size.	Integer	Range = [2 to 32] Default = 4

Implementation Parameters			
Name	Description	Type	Range
Data IO Format	Selects the format for the IO streaming of data (array or sample per invocation);it affects the test design function wrapper only.	String	Range = [array sample] Default = [array]
Output Precision	Number of bits for output matrix.	Integer	Default = auto auto sets value based on input matrix word-length

Inputs / Outputs

Real-Only Data

Input(s)			
Name	Description	Type	Range
X	Input matrix. XBITS, XFBITS are the input quantization wordlength and fractional length as per A quantization parameters.	Real	Range = $[-2^{XBITS-1}$ to $2^{XBITS-1}-1]/2^{XFBITS}$

Output(s)			
Name	Description	Type	Range
Y	Inverse ouput matrix. YBITS, YFBITS are the Y matrix output wordlength and fractional length.	Real	Range = $[0$ to $2^{YBITS-1}] / 2^{YFBITS}$

Complex Data

Input(s)			
Name	Description	Type	Range
X_real, X_imag	Input matrix. XBITS, XFBITS are the input quantization wordlength and fractional length as per A quantization parameters.	Real	Range = $[-2^{XBITS-1}$ to $2^{XBITS-1}-1]/2^{XFBITS}$

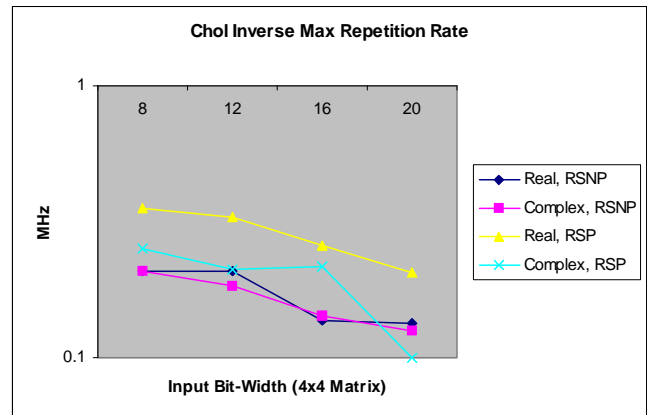
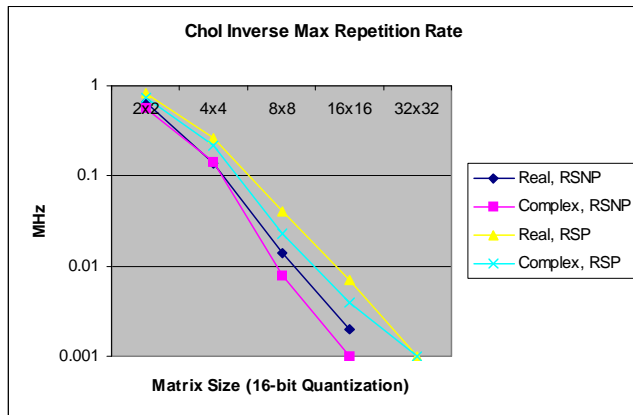
Output(s)			
Name	Description	Type	Range
Y_real, Y_imag	Inverse output matrix. YBITS, YFBITS are the Y matrix output wordlength and fractional length.	Real	Range = $[0 \text{ to } 2^{YBITS-1}] / 2^{YFBITS}$
s	Singular matrix indication	Boolean	Range { 1 0 } 0 = (invertable matrix) 1 = (singular, non-invertable matrix)

Expected Quality of Results

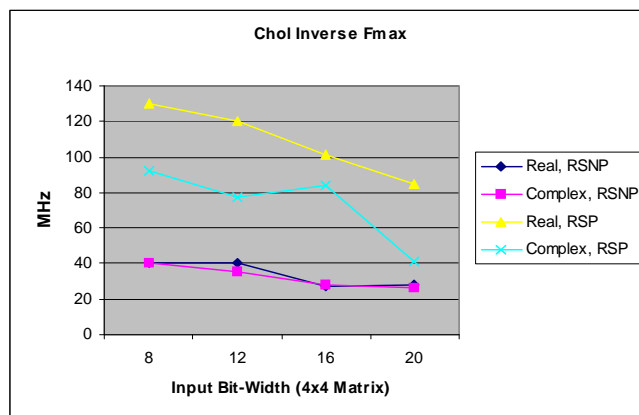
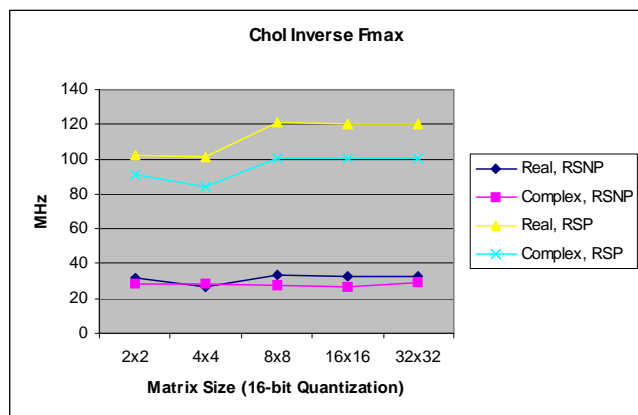
The following plots show typical speed and utilization of the chol_inverse core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and RSP refer to resource-shared, non-pipelined architectures and resource-shared pipelined architectures respectively.

Speed of Operation

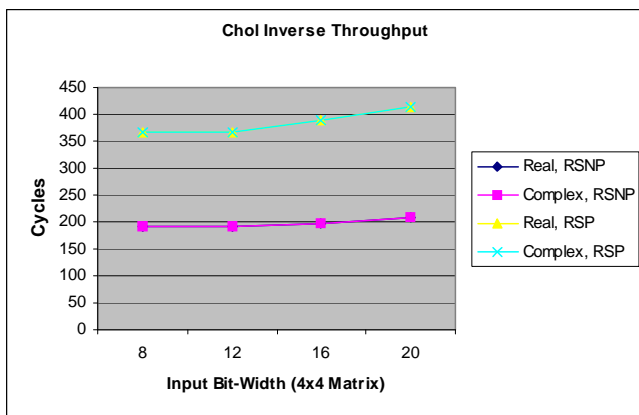
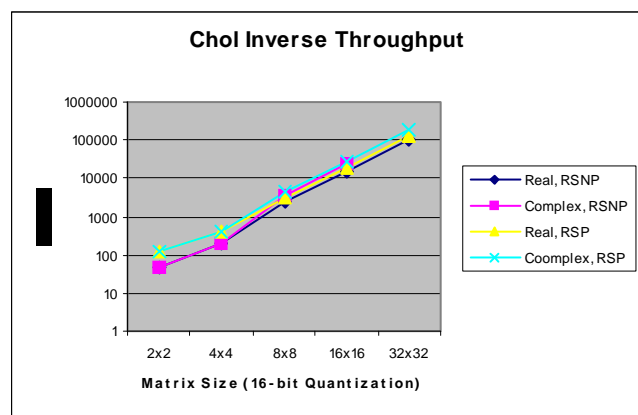
Inversion Repetition Rate



Maximum Operating Frequency



Hardware Clock Cycles per Design Function Call



cicdecim

Supported Filter Design Toolbox function.

cicinter

Supported Filter Design Toolbox function.

complex

Construct complex data from real and imaginary components. The complex function is fully supported for synthesis.

Complex Normalization

Description

Normalizes the magnitude of the complex number represented by the pair of inputs containing the real and imaginary values. Mathematically, the input/output relationship is as follows:

function

```
[u,v] = accel_cmplxnorm(x,y)
```

Where

```
abs(u + i*v) = 1.0;
angle(u + i*v) = angle(x + i*y);
```

abs() and angle() are the corresponding MATLAB functions that generate the complex modulus and angle() of the complex input pair.

Related MATLAB Functions

[atan2](#)

[sin](#)

Differences in Operation to MATLAB Functions

The cmplxnorm() function does not have a single equivalent function in MATLAB. The functionality of cmplxnorm(), however, is equivalent to the combination of the MATLAB atan2() and sin() as follows.

```
phi = atan2(x,y)
[u,v] = [sin(phi), cos(phi)]
```

Syntax

AccelDSP cmplxnorm() Function Call	Supported MATLAB Syntax
[u,v] = accel_cmplxnorm(x,y)	phi = atan2(x,y) [u,v] = [sin(phi) cos(phi)]

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	{CORDIC}
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB.	Integer	[4 to 32] Default = [auto]
Output bitwidth	Number of bits used to represent the output.	Positive Integer	Range = [4 to 32] Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
x	Input representing the real(x) part of the input complex number to normalize.	Real	Function of the Input Quantizer parameters
y	Input representing the imag(x) part of the input complex number to normalize.	Real	Function of the Input Quantizer parameters

Output(s)			
Name	Description	Type	Range
u	Output representing the real(x) part of the normalized complex number.	Real	Function of the Output bitwidth parameter.
v	Output representing the imag(x) part of the normalized complex number.	Real	Function of the Output bitwidth parameter.

conj

Complex conjugate function is fully supported for synthesis.

convenc

Supported Communications Toolbox function.

convergent

Convergent is supported for synthesis.

convdenintlv

Convolutional Deinterleaver is supported.

convinctlv

Convolutional Interleaver is supported.

colon

Supported for synthesis except in control constructs.

cos/accel_cos

Description

Returns the cosine of the input matrix.

Related MATLAB Functions

[cos](#)

Differences with the MATLAB Function

1. The inputs must be pre-quantized to the accuracy specified by the input quantizer.
2. The shape of the input is limited to a scalar, vector or 2-D matrix.

Extended Features of accel_cos()

The accel_cos() function provides the same functionality as the cos() function plus the following extended features:

1. The accel_cos() function can create an optional sin() output with little additional hardware.
2. The accel_cos() function can specify input units of measure ([InputType](#)) in the following formats:
 - ♦ [native form](#)
 - ♦ [scaled form](#)

Syntax

AccelDSP cos() Function Call	Supported MATLAB Syntax
y = cos(x);	y = cos(x);
AccelDSP accel_cos() Function Call(s)	Supported MATLAB Syntax
y = accel_cos(x); [y,z] = accel_cos(x);	y = cos(x); z = sin(x);

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 7

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
InputType	Specify the representation of the input word.	String	{ native form scaled form } Default = native form

Implementation Parameters			
Name	Description	Type	Range
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 100] Default = [auto]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.		Range = 1 to 6 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 1 to 32 Default = auto
Output bitwidth	The number of bits in the output word(s).		Range 4 to 32 Default = auto
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC

Inputs / Outputs

cos() Input			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix.	Real	Range = [0 to 2π] or [$-\pi$ to π] depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

accel_cos() Input			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix. (For InputType = native form).	Real	Range = $[0 \text{ to } 2\pi]$ or $[-\pi \text{ to } \pi]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.
	May be a scalar, vector or 2-D matrix. (For InputType = scaled form).	Positive Integer	Range = $[0 \text{ to } 2^{\text{IBITS}-1}]$ or $[-2^{\text{IBITS}-1} \dots 2^{\text{IBITS}-1}-1]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

cos() Output			
Name	Description	Type	Range
y	May be a scalar, vector, or 2-D matrix representing the cosine of the corresponding input element. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

Additional accel_cos() Output			
Name	Description	Type	Range
z	An optional output scalar, vector, or 2-D matrix representing the sine of the corresponding input. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

cumprod

Cumprod is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

cumsum

Cumsum is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

de2bi

The syntax `b = de2bi(d, n)` is supported for synthesis, where `d` can be a variable representing a scalar, vector or two-dimensional matrix. `n` must be a positive integer (constant) which specifies the output word width. The syntax `b = de2bi(d)` is not supported.

dfilt

Supported Filter Design Toolbox function.

diff

Supported for synthesis.

dot

else

Else is supported as part of an IF structure.

Example

```
if ( a > b)
    c = a;
else
    c = 0;
end
```

elseif

Elseif is supported as part of an IF structure.

Example

```
if ( a > b)
    c = a;
elseif (a == b)
    c = d;
else
    c = 0;
end
```

end

End is supported as part of IF, FOR, and WHILE statements.

eps

Supported for synthesis.

eq

Supported for synthesis.

A==B

tmp = eq(A,B)

exp/accel_exp

Description

Returns e (the base of natural logarithms) raised to a power. The exp function operates element-by-element on arrays. For example, Y = exp(X) returns the exponential for each element of X

Related MATLAB Functions

[exp](#)

Differences with the MATLAB Function

The shape of the input is limited to scalars, vectors, or 2-D matrices.

Limitations

Fixed point severely limits the dynamic range. Answers in most cases will not match MATLAB. When large exponent values are detected, AccelDSP displays a WARNING message about the accuracy of the results.

To allow more accurate results, a floating point engine that returns a mantissa and an exponent instead of a single number is used for computing exp() under all conditions: [f,e]=accel_exp(X) where $z=f*2^e$. Even for type="fixed point" in the exp() model, the accel_exp() model is called and floating point math is used. The fixed point type maintains the MATLAB format of a single number output.

Syntax

AccelDSP Function Call	Supported MATLAB Syntax
Y = exp(X);	Y = exp(X);

AccelDSP accel_exp() Function Call	Notes
$[f,e] = \text{accel_exp}(X);$ $Y = f \cdot 2^e$	

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 9

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	CORDIC Default = CORDIC
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 47] Default = [auto]
Type	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range [fixed point floating point auto] Default = floating point
Output bitwidth	The number of bits in the output word(s).		Range 4 to 31 Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
X	May be a scalar, vector or 2-D matrix. Each element is IBITS-bits wide.	Positive Integer	Specified by the Input Quantizer parameters

exp Output(s)			
Name	Description	Type	Range
Y	A scalar, vector or 2-D matrix with each element representing the exp() of the corresponding element in the input. Each element is OBITS-bits wide.	Real	Range = $[-2^{OBITS-1} \text{ to } -2^{OBITS-1-1}] / 2^{OFBITS}$. OFBITS is automatically computed as a function of the Input Quantizer.

accel_exp Output(s)			
Name	Description	Type	Range
f	Represents the mantissa of the floating-point output where $Z = f \cdot 2^e$	Real	
e	Represents the exponent of the floating-point output where $Z = f \cdot 2^e$	Real	

Example

```
Z = exp(X,Y);
```

Alternative Function accel_exp()

Fixed-point math severely limits the acceptable dynamic range of the `exp()` input. As an alternative function, you can use `accel_exp()` which computes the power using floating point math.

Example

```
[f,e] = accel_exp(X,Y);
```

```
Z = f*2^e
```

The results should be much closer to the MATLAB results.

eye

Eye is supported for generation of scalars, 1-D and 2-D matrices.

Example 1

```
A = eye;
```

Example 2

```
A = eye(1);
```

Example 3

```
A = eye(2);
```

Example 4

```
A = eye(2,3);
```

Limitations:

`eye(0)` is unsupported. Eye is not supported for 3-D and higher matrices.

factorial

Factorial of constants is supported when x in factorial(x) resolves to a positive constant integer. Factorial of non-constants is not supported for synthesis in this release as in factorial(x) where x is not a constant.

Example

```
A = factorial(3);
```

false

Supported for synthesis.

Example 1

```
A = false;
```

Example 2

```
A = false(1,5);
```

fft

Supported for synthesis.

filter

Supported for synthesis.

firhalfband

A supported Filter Design Toolbox function.

fix

Supported for synthesis.

fliplr

Supported for synthesis.

fliprl

Supported for synthesis.

flipud

Supported for synthesis.

floor

Supported for synthesis as a function. Supported in the “Roundmode” when defining a quantizer type.

Example 1

```
q = quantizer('fixed','floor','wrap',[24 8]); % This usage of floor is supported
```

Example 2

```
y = floor(x);
```

for

Supported for synthesis when start:stride:end resolve to integers. Arbitrary nested for loops are also supported.

Example 1

```
for i = start:stride:end
    Statement 1;
    Statement 2;
    ...
    Statement n;
end
```

Example 2

```
for i = start:end
    Statement 1;
    Statement 2;
    ...
    Statement n;
end
```

Example 3

```
NUMTAPS = 16;
for i = 1:NUMTAPS
    Statement 1;
    Statement 2;
    ...
    Statement n;
end
```

Example 4

```
for i = start:end
    for j = start:end
        Statement 1;
        Statement 2;
        ...
        Statement n;
    end % j
end % i
```

function

Arbitrary levels of function hierarchy are supported. Subfunctions and private functions are also supported.

gf

Create a Galois field array. Supported for synthesis as a function.

Givens Array Rotation

Performs a givens rotation on a vector pair.

global

Global variables are not currently supported by AccelDSP.

hypot

Supported for synthesis.

if

Supported as part of an if / end construct. Also supported with else and elseif constructs.

Example 1

```
if ( a > b)
    c = a;
else
    c = 0;
end
```

Example 2

```
if ( a > b)
    c = a;
elseif (a == b)
    c = d;
else
    c = 0;
end
```

ifft

Supported for synthesis.

imag

Imaginary part of complex number. Fully supported for synthesis.

inv

Supported for synthesis.

Inverse Square Root

Returns the inverse square root of the input matrix. Supported for synthesis.

isempty

Supported for synthesis when used to set an initial value of a [persistent](#) variable.

Example

```
persistent z1; % declare persistent filter state
if isempty(z1)
    z1 = 0.5; % initialize filter state
end
```

ldivide

Supported for synthesis.

length

Supported for synthesis when the argument is statically determinant.

Example

```
A = [2 4 6 8 10];
B = length(A);
```

load

Only supported for .txt files, not .mat files. The supported usage is as follows:

```
X = load('my_file.txt');
```

log

Description

Natural logarithm

Related MATLAB Functions

[log](#)

Differences with the MATLAB Function

The shape of the input is limited to scalars, vectors, or 2-D matrices.

Syntax

AccelDSP Function Call	Supported MATLAB Syntax
<code>y = log(x);</code>	<code>y = log(x);</code>

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 32] Default = [auto]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimize the table sizes.		Range = 2 to 32 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 2 to 32 Default = auto
Output bitwidth	The number of bits in the output word(s).		Range 4 to 31 Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
SigIn	May be a scalar, vector or 2-D matrix. Each element is IBITS-bits wide.	Positive Integer	Specified by the SigIn quantizer parameters

Output(s)			
Name	Description	Type	Range
SigOut	A scalar, vector or 2-D matrix with each element representing the natural logarithm of the corresponding element in the input. Each element is OBITS-bits wide.	Real	Range = $[-2^{\text{OBITS}-1}$ to $-2^{\text{OBITS}-1}-1] / 2^{\text{OFBITS}}$. OFBITS is automatically computed as a function of the input quantizer.

log10

Description

Base-10 logarithm

Related MATLAB Functions

[log10](#)

Differences with the MATLAB Function

The shape of the input is limited to scalars, vectors, or 2-D matrices.

Syntax

AccelDSP Function Call	Supported MATLAB Syntax
SigOut = log10(SigIn);	y = log10(x);

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 32] Default = [auto]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimize the table sizes.		Range = 2 to 32 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 2 to 32 Default = auto
Output bitwidth	The number of bits in the output word(s).		Range 4 to 31 Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
SigIn	May be a scalar, vector or 2-D matrix. Each element is IBITS-bits wide.	Positive Integer	Specified by the SigIn quantizer parameters

Output(s)			
Name	Description	Type	Range
SigOut	A scalar, vector or 2-D matrix with each element representing the $\log_{10}()$ of the corresponding element in the input. Each element is OFBITS-bits wide.	Real	Range = $[-2^{\text{OFBITS}-1}$ to $-2^{\text{OFBITS}-1}-1] / 2^{\text{OFBITS}}$. OFBITS is automatically computed as a function of the input quantizer.

log2

Description

Base-2 logarithm

Related MATLAB Functions

[log2](#)

Differences with the MATLAB Function

The shape of the input is limited to scalars, vectors, or 2-D matrices.

Syntax

AcceIDSP Function Call(s)	MATLAB Syntax	Notes
SigOut = log2(SigIn);	Y = log2(X);	
[F,E] = log2(SigIn);	[F,E] = log2(X);	Dissect the floating point number into F and E where $F \cdot 2^E$ represents the floating point number. This is the auto-inferable form.

Inferring the 'Dissect Floating Point Number' Form of the log2 Function

The code sample below shows how to infer the log2 function:

```
function [E,F] = log2_function(in1)

% infer the log2 function
[E,F] = log2(in1);
```

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = ufixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 32] Default = [auto]

Implementation Parameters			
Name	Description	Type	Range
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.		Range = 2 to 32 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 2 to 32 Default = auto
Function Type	Returns the Base-2 logarithm or F and E. In the latter, the floating point input X is dissected into the elements F and E where $X = F \cdot 2^E$.	String	{Base-2 logarithm dissect floating pint number}
Output bitwidth	The number of bits in the output word(s).		Range 4 to 31 Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
SigIn	May be a scalar, vector or 2-D matrix. Each element is IBITS-bits wide.	Positive Integer	Specified by the SigIn quantizer parameters

Output(s)			
Name	Description	Type	Range
SigOut	A scalar, vector or 2-D matrix with each element representing the $\log_2()$ of the corresponding element in the input. Each element is OBITS-bits wide.	Real	Range = $[-2^{OBITS-1}$ to $-2^{OBITS-1}] / 2^{OFBITS}$. OFBITS is automatically computed as a function of the input quantizer.

max

Supported for synthesis. When the argument is a constant, the max function will resolve to a constant in hardware. If the argument is not a constant, hardware will be generated to calculate the maximum.

Example

```
a = [1 2 3 4 7 4];
if (b > max(a))
    c = b;
else
    c = b + 2;
end
```

mean

Supported for synthesis.

mfilt.firdecim

Filter Design Toolbox function supported for synthesis.

mfilt.firtdecim

Filter Design Toolbox function supported for synthesis.

min

Supported for synthesis. When the argument is a constant, the min function will resolve to a constant in hardware. If the argument is not a constant, hardware will be generated to calculate the minimum.

Example

```
a = [1 2 3 4 7 4];
if (b > min(a))
    c = b;
else
    c = b + 2;
end
```

minus/accel_complex_minus

Description

Return the matrix difference.

Related MATLAB Functions

[minus](#)

Syntax

AccelDSP minus() Function	Supported MATLAB Syntax
<code>C = minus(A,B);</code>	<code>C = minus(A,B);</code>

AccelDSP accel_complex_minus() Function Call	Notes
<code>[C_real,C_imag] = accel_complex_minus(A_real,A_imag,B_real,B_imag);</code>	

Parameters

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the data type.	String	{double galoisfield} Default = double
Rows in A input	Specify the number of rows in the A input.	String	Range = [3 to 1024] Default = 3
Columns in A input	Specify the number of Columns in the A input.	String	Range = [3 to 1024] Default = 3
Rows in B input	Specify the number of rows in the B input.	String	Range = [3 to 1024] Default = 3
Columns in B input	Specify the number of Columns in the B input.	String	Range = [3 to 1024] Default = 3

Inputs / Outputs

minus() Inputs			
Name	Description	Type	Range
A	May be a scalar, vector or 2-D matrix.	Real	
B	May be a scalar, vector or 2-D matrix.	Real	

accel_complex_minus() Inputs			
Name	Description	Type	Range
A_real	May be a scalar, vector or 2-D matrix.	Real	
A_imag	May be a scalar, vector or 2-D matrix.	Real	
B_real	May be a scalar, vector or 2-D matrix.	Real	
B_imag	May be a scalar, vector or 2-D matrix.	Real	

minus() Output			
Name	Description	Type	Range
C	Result of matrix difference	Real	

accel_complex_minus() Outputs			
Name	Description	Type	Range
C_real	Result of matrix difference.	Real	
C_imag	Result of matrix difference.	Real	

mod

Description

Modulus after division

Related MATLAB Functions

[mod](#)

Differences with the MATLAB Function

None

Syntax

AccelDSP Function Call	Supported MATLAB Syntax
SigOut = mod(SigIn_N,SigIn_D);	Z = mod(X,Y);

ParametersParameters

Denominator Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = ufixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 0

Numerator Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Newton-Raphson Goldschmidt Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Newton-Raphson Goldschmidt Bipartite Tables Linearly Interpolated LUT Default = CORDIC
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Positive Integer	[4 to 32] Default = [auto]

Implementation Parameters			
Name	Description	Type	Range
Iterations	This parameter dictates the number of iterations to use in the Newton Raphson or Goldschmidt algorithm.	Positive Integer	Range = [1, 2 or 3] Default = [auto]
Init Lut Bits	Specifies the number of MSB bits to use for the initialization table when Newton Raphson or Goldschmidt implementations are specified.	Positive Integer	Range 2 to 32 Default = [default]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.	Positive Integer	Range = 2 to 32 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.	Positive Integer	Range 2 to 32 Default = auto
Output bitwidth	The number of bits in the output word(s).	Positive Integer	Range 4 to 32 Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
SigIn_N	An input stream of numbers that represent the numerator of the division operation.	Real	The range of the numbers is defined by the SigIn_N quantizer settings.
SigIn_D	An input stream of numbers that represent the denominator of the modulus operation.	Real	The range of the numbers is defined by the SigIn_D quantizer settings.

Output(s)			
Name	Description	Type	Range
SigOut	An output stream of numbers that represent the result of the modulus operation.	Real	<p>If either the numerator or the denominator is type 'fixed', the range of the output will be equal to $[-2^{\text{OBFITS}-1}, -2^{\text{OBFITS}-1}]/2^{\text{OBFITS}}$</p> <p>However, if both inputs are type 'ufixed', the range of the output will be equal to $[0, -2^{\text{OBFITS}-1}]/2^{\text{OBFITS}}$</p> <p>The number of fractional bits (OBFITS) is automatically selected to cover the maximum possible output value.</p>

mpower

Matrix power. Support includes X to the power y when y is an integer and a scalar.

Example

```
X = [1 2; 3 4];
y = 3;
Z = mpower(X,y);
```

mtimes/accel_complex_mtimes

Description

Matrix multiplication

Related MATLAB Functions

[mtimes](#)

Syntax

AccelDSP mtimes() Function	Supported MATLAB Syntax
<code>C = mtimes(A,B);</code>	<code>C = mtimes(A,B);</code>

AccelDSP accel_complex_mtimes() Function Call	Notes
<code>[C_real,C_imag] = accel_complex_mtimes(A_real,A_imag,B_real,B_imag);</code>	

Parameters

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	{array-array array-scalar scalar-arrayT} Default = array-array

Inputs / Outputs

mtimes() Inputs			
Name	Description	Type	Range
A	May be a scalar, vector or 2-D matrix. Matrix size is limited to 4096 rows by 4096 columns.	Real	Rows: [1-4096] Columns: [1-4096]
B	May be a scalar, vector or 2-D matrix.	Real	Rows: [1-4096] Columns: [1-4096]

accel_complex_mtimes() Inputs			
Name	Description	Type	Range
A_real	May be a scalar, vector or 2-D matrix.	Real	
A_imag	May be a scalar, vector or 2-D matrix.	Real	
B_real	May be a scalar, vector or 2-D matrix.	Real	
B_imag	May be a scalar, vector or 2-D matrix.	Real	

mtimes() Output			
Name	Description	Type	Range
C	Result of matrix multiplication.	Real	

accel_complex_mtimes() Outputs			
Name	Description	Type	Range
C_real	Result of matrix multiplication.	Real	
C_imag	Result of matrix multiplication.	Real	

ndims

Supported for synthesis.

ne

Supported for synthesis.

nexpow2

Supported for synthesis.

norm

Supported for synthesis.

ones

Supported for synthesis in this release for up to two dimensions.

Example 1

```
a = ones;
```

Example 2

```
a = ones(1,5);
```

otherwise

Supported as part of the switch statement.

Example

```
switch (x)
case (0)
    y = y + 1;
otherwise (1)
    y = y + 2;
end
```

pause

The pause statement is not supported and may cause AccelDSP to enter a suspended state.

persistent

Supported for synthesis if the declared variable(s) are initialized by an [isempty](#) statement.

Example

```
persistent z1; % declare persistent filter state
if isempty(z1)
    z1 = 0.5; % initialize filter state
end
```

pi

Supported for synthesis.

plus/accel_complex_plus

Description

Return the matrix sum.

Related MATLAB Functions

[plus](#)

Syntax

AccelDSP plus() Function	Supported MATLAB Syntax
<code>C = plus(A,B);</code>	<code>C = plus(A,B);</code>

AccelDSP accel_complex_plus() Function Call	Notes
<code>C_real,C_imag] = accel_complex_plus(A_real,A_imag,B_real,B_imag);</code>	

Parameters

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the data type.	String	{double galoisfield} Default = double
Rows in A input	Specify the number of rows in the A input.	String	Range = [3 to 1024] Default = 3
Columns in A input	Specify the number of Columns in the A input.	String	Range = [3 to 1024] Default = 3
Rows in B input	Specify the number of rows in the B input.	String	Range = [3 to 1024] Default = 3
Columns in B input	Specify the number of Columns in the B input.	String	Range = [3 to 1024] Default = 3

Inputs / Outputs

plus() Input(s)			
Name	Description	Type	Range
A	May be a scalar, vector or 2-D matrix.	Real	
B	May be a scalar, vector or 2-D matrix.	Real	

accel_plus_minus() Inputs			
Name	Description	Type	Range
A_real	May be a scalar, vector or 2-D matrix.	Real	
A_imag	May be a scalar, vector or 2-D matrix.	Real	
B_real	May be a scalar, vector or 2-D matrix.	Real	
B_imag	May be a scalar, vector or 2-D matrix.	Real	

plus() Output(s)			
Name	Description	Type	Range
C	Result of matrix difference	Real	

accel_complex_plus() Inputs			
Name	Description	Type	Range
C_real	Result of matrix difference.	Real	
C_imag	Result of matrix difference.	Real	

pol2cart

Supported for scalars, vectors and two-dimensional matrices containing both positive and negative numbers.

poly2trellis

Communications Toolbox function supported for synthesis.

polyval

Description

Polynomial Evaluation

Related MATLAB Functions

[polyval](#)

Differences with the MATLAB Function

In MATLAB, the input <x> is a vector of complex numbers passed to polyval() in parallel with the output <y> occurring as a vector of complex numbers in parallel as well. Since this is not practical in hardware, in the AccelDSP polyval() function, the input <x> occurs as a serial stream of IBITS-bit numbers and the output <y> occurs as serial stream of OBITS-bit numbers.

Syntax

Function Call	Supported MATLAB Syntax	Unsupported MATLAB Syntax
y = polyval(x);	y = polyval(x);	Complex and Negative inputs are not supported in this version due to the complex outputs generated by the polyval function

Parameters

Denominator quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = ufixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
polyvec	<p>A vector of length N+1 whose elements are the coefficients of a polynomial, is the value of the polynomial evaluated at X.</p> <p>Default is ([0.5 0 -4.1 3.4 2.32 1.5]).</p> <p>$Y = 0.5 \cdot X^5 - 4.1 \cdot X^3 + 3.4 \cdot X^2 + 2.32 \cdot X + 1.5$</p>		Default is ([0.5 0 -4.1 3.4 2.32 1.5])
Output bitwidth	Number of bits used to represent the output y.	Positive Integer	<p>Range = [4 to 32]</p> <p>Default = [auto]</p> <p>[auto] is used to automatically compute the output word widths.</p>
Implementation	Hardware implementation options.	String	<p>Options = [Logic LUT Interpolated LUT]</p> <p>Default = Interpolated Logic</p>
Coefficients Wordlength	Number of bits used to represent the polynomial coefficients p.	Positive Integer	Default = 0
Rolled	Resource share the iterative engine	String	<p>Range = [yes no]</p> <p>Default = yes</p>

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
x	An input stream of numbers each IBITS-bits wide.	Positive Integer	<p>Range = [0 to $2^{\text{IBITS}} - 1$]</p> <p>/2^{IFBITS}</p>

Output(s)			
Name	Description	Type	Range
y	An output stream of numbers representing the Polyval(x) each OBITS-bits wide.	Real	Range = $[0 \text{ to } 2^{\text{OBITS}} - 1]$ $/2^{\text{OFBITS}}$

pow2

Pow2 of constants is supported when x in pow2(x) resolves to a constant. Pow2 of non-constants is not supported for synthesis in this release as in pow2(x) where x is not a constant.

Example

```
A = pow2(4);
```

power/accel_power

Description

Return the result of the power.

Related MATLAB Functions

[power](#)

Differences with the MATLAB Function

The shape of the input is limited to scalars, vectors, or 2-D matrices.

Limitations

Fixed point severely limits the dynamic range. Answers in most cases will not match MATLAB. When large exponent values are detected, AccelDSP displays a WARNING message about the accuracy of the results.

To allow more accurate results, a floating point engine that returns a mantissa and an exponent instead of a single number is used for computing power() under all conditions: [f,e]=accel_power(X,Y) where $z=f*2^e$. Even for type="fixed point" in the power() model, the accel_exp() model is called and floating point math is used. The fixed point type maintains the MATLAB format of a single number output.

Syntax

AccelDSP Function Call	Supported MATLAB Syntax
Z = power(X,Y);	Z = power(X,Y);

AccelDSP accel_power() Function Call	Notes
[f,e] = accel_power(X,Y); Z = f*2^e	

Parameters

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	Range = {default}
Type	See Limitations above		Range [fixed point floating point galiosfield] Default = floating point
Output bitwidth	The number of bits in the output word(s).		Range [4 to 31] Default = auto

Inputs / Outputs

power Input(s)			
Name	Description	Type	Range
X	May be a scalar, vector or 2-D matrix.	Positive Integer	
Y	May be a scalar, vector or 2-D matrix.	Positive Integer	

power Output(s)			
Name	Description	Type	Range
Z	A scalar, vector or 2-D matrix with each element representing the exp() of the corresponding element in the input. Each element is OBITS-bits wide.	Real	Range = $[-2^{\text{OBITS}-1}$ to $-2^{\text{OBITS}-1}-1] / 2^{\text{OFBITS}}$. OFBITS is automatically computed.

accel_powerOutput(s)			
Name	Description	Type	Range
f	Represents the mantissa of the floating-point output where $Z = f \cdot 2^e$	Real	
e	Represents the exponent of the floating-point output where $Z = f \cdot 2^e$	Real	

prod

Prod is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

qr

Supported for synthesis.

qdrpls_spatial

Description

The *qdrpls_spatial* reference design implements a spatial filtering operation based on a recursive, least-squares (RLS) approximation of a desired input signal. The RLS operation in turn is based on recursive orthogonal-triangular decomposition (QRD) of the matrix formed by stacking samples of the input signal.

Related MATLAB Functions

[qdrpls](#)

Differences with the MATLAB Function

The functionality of this reference design is mathematically described as follows.

$$\mathbf{A} = [\mathbf{A}; \mathbf{x}]$$

$$\mathbf{b} = [\mathbf{b}; \mathbf{d}]$$

$$\mathbf{A} = \mathbf{Q} \mathbf{R}$$

$$\mathbf{w} = \mathbf{R}^{-1} \mathbf{Q}^T \mathbf{b}$$

$$\mathbf{e} = \mathbf{d} - \mathbf{x} \mathbf{w}$$

Where

\mathbf{A} is a matrix formed with the stacking of input samples \mathbf{x} as consecutive rows. The number of columns is fixed and equals the number of filter weights. The number of rows increases with every iteration of the algorithm.

\mathbf{b} is a column vector formed by stacking input samples of the desired input signal \mathbf{d} .

\mathbf{Q} , \mathbf{R} are the orthogonal-triangular factors of the matrix \mathbf{A} ; these are obtained with a recursive QR-decomposition algorithm.

\mathbf{w} is the a least-squares solution to the system of equations $\mathbf{A} \mathbf{w} = \mathbf{b}$. These values are the weights (or taps) of the QRD-RLS spatial filter.

\mathbf{e} is the scalar output of the filtering operations.

The *qdrpls_spatial* operates on real-valued input and output data only.

The MATLAB *qdrpls()* function is a constructor of an adaptive fitter object which can then be used in conjunction with the *filter()* function to apply the filtering. This object implements a single-input, transversal linear filter which differs from the AccelDSP *qdrpls_spatial* function. The AccelDSP *qdrpls_spatial* function takes multiple inputs (as a 1-D vector input \mathbf{x}), and it applies the recursive QR-decomposition treating \mathbf{x} as a new row of the input matrix.

Syntax

Function Call	Supported MATLAB Syntax	Notes
[e] = qdrils_spatial(x,d);	$A = [A;x];$ $b = [b;d];$ $[Q,R] = qr(A);$ $w = inv(R)*Q*b;$ $e=d-x*w;$	The x input is a real-valued 1-D array, d and e are real-valued scalars.

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Mode = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Range = [4 to 12] Default = 12
Fractional Length	The number of fractional bits for the input word(s).	String	Range = [0 to 10] Default = 10

Implementation Parameters			
Name	Description	Type	Range
Implementation	Implementation computation style.	String	CORDIC Multiplier Default = CORDIC
Type	Input data type.	string	{double complex} Default = double
Number of Weights	Number of weights (or taps) in filtering operation.		Range = [3 to 16] Default = [4]
Lambda-Square Factor	Lambda forgetting factor (squared value).		Range = [0.98 to 0.99] Default = [0.99]
Initial Correlation Matrix	Value to initialize correlation matrix; initialization is done with a diagonal matrix containing this value; auto sets the diagonal values based on input quantization LSB.		Range = [0.001 - 0.1] Default = [auto]

Inputs / Outputs

Real-Only Data

Input(s)			
Name	Description	Type	Range
x	Row vector of input samples. 'word' is the Input Quantizer Word Length and 'fractional' is the Input Quantizer Fractional Length.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$
d	Scalar input sample of desired signal. 'word' is the Input Quantizer Word Length and 'fractional' is the Input Quantizer Fractional Length.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$

Output(s)			
Name	Description	Type	Range
e	Scalar filter output. OBITS, OFBITS are the output wordlength and fractional length automatically determined by the AccelDSP generator.	Real	Range = $[-2^{\text{OBITS}-1}$ to $2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

Complex Data

Input(s)			
Name	Description	Type	Range
x_real, x_imag	Input matrix to be factored; separate real and imaginary parts of complex input matrix. ABITS and AFBITS are the input quantization wordlength and fractional length as per A quantization parameters.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$
d_real, d_imag	Scalar input sample of desired signal. 'word' is the Input Quantizer Word Length and 'fractional' is the Input Quantizer Fractional Length.	Real	Range = $[-2^{\text{word}-1}$ to $2^{\text{word}-1}-1]/2^{\text{fractional}}$

Output(s)			
Name	Description	Type	Range
e_real, e_imag	Scalar filter output; separate real and imaginary parts of complex input matrix. QBITS and QFBITS are the Q matrix output wordlength and fractional length.	Real	Range = $[-2^{\text{QBITS}-1}$ to $2^{\text{QBITS}-1}-1]/2^{\text{QFBITS}}$

quantize

Defines the fixed-point parameters of a variable.

quantizer

Applies the quantize object to a variable.

rcosflt

Communications Toolbox function supported for synthesis.

rdivide

Description

Right array division.

Related MATLAB Functions

[Array Right Division](#)

[rdivide](#)

Differences with the MATLAB Function

None

Syntax

AccelDSP rdivide() Function	Supported MATLAB Syntax
Y = rdivide(N,D); [Y,dz] = rdivide(N,D);	Y = N./D; y = rdivide(N,D);

AccelDSP accel_rdivide() Function Call	Supported MATLAB Syntax
Y = accel_rdivide(N,D); [Y,dz] = accel_rdivide(N,D);	Y = N./D; y = rdivide(N,D);

Parameters

Denominator Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = ufixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 0

Numerator Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = ufixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap

Numerator Quantization			
Name	Description	Type	Range
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Type	Select the data type	String	double galoisfield complex: separate real and imag IO Default = double
Implementation	Select the hardware architecture to create. CORDIC Newton-Raphson Goldschmidt Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Newton-Raphson Goldschmidt Bipartite Tables Linearly Interpolated LUT Default = CORDIC
Resource Sharing	Newton Raphson: This parameter determines the type of resource sharing to use in the Newton Raphson algorithm.	String	{single multiplier single N-R iteration (2 mults)}
additional mult. pipes(-5:5)	Newton Raphson/Goldschmidt: This parameter determines the number of additional pipe stages to insert in the multiplier chains of the Newton Raphson and Goldschmidt algorithm.	String	Range = [-5:5] Default = 0
additional adder pipes(-5:5)	Newton Raphson/Goldschmidt: This parameter determines the number of additional pipe stages to insert in the adder chains of the Newton Raphson algorithm.	String	Range = [-5:5] Default = 0

Implementation Parameters			
Name	Description	Type	Range
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Positive Integer	[4 to 32] Default = [auto]
Iterations	This parameter dictates the number of iterations to use in the Newton Raphson or Goldschmidt algorithm.	Positive Integer	Range = [1, 2 or 3] Default = [auto]
Init Lut Bits	Specifies the number of MSB bits to use for the initialization table when Newton Raphson or Goldschmidt implementations are specified.	Positive Integer	Range 2 to 32 Default = [default]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.	Positive Integer	Range = 2 to 5 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.	Positive Integer	Range 2 to 32 Default = auto
Output bitwidth	The number of bits in the output word(s).	Positive Integer	Range 4 to 32 Default = auto
Divide by Zero Flag	Include a divide-by-zero flag signal.	String	Range = [yes no] Default = no

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
Sig_N	An input stream of numbers that represent the numerator of the division operation.		The range of the numbers is defined by the SigIn_N quantizer settings.
D	An input stream of numbers that represent the denominator of the division operation.		The range of the numbers is defined by the SigIn_D quantizer settings.

Output(s)			
Name	Description	Type	Range
Y	An output stream of numbers that represent the result of the division operation.		<p>If either the numerator or the denominator is type 'fixed', the range of the output will be equal to $[-2^{\text{OBFITS}-1}, -2^{\text{OBFITS}-1}]/2^{\text{OBFITS}}$</p> <p>However, if both inputs are type 'ufixed', the range of the output will be equal to $[0, -2^{\text{OBFITS}-1}]/2^{\text{OBFITS}}$</p> <p>The number of fractional bits (OBFITS) is automatically selected to cover the maximum possible output value.</p>

Additional accel_rdivide() Output			
Name	Description	Type	Range
dz	Optional divide-by-zero flag.		A binary value that is set to '1' if the denominator is zero.

reallog

Supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

real

Real part of complex number. Fully supported for synthesis.

realpow

Supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

realsqrt

Supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

rem

Description

Remainder after division

Related MATLAB Functions

[rem](#)

Differences with the MATLAB Function

When $Y = 0$, MATLAB $\text{rem}(X,Y) = \text{NaN}$

When $Y = 0$, $\text{rem}(X,Y) = 0$ with “DivZero” output driven to 1

Syntax

AcceIDSP rem() Function Call	Supported MATLAB Syntax
$R = \text{rem}(X,Y);$	$R = \text{rem}(X,Y);$

ParametersParameters

Denominator Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = ufixed
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = ufixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10

Numerator Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor

Numerator Quantization			
Name	Description	Type	Range
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Newton-Raphson Goldschmidt Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Newton-Raphson Goldschmidt Bipartite Tables Linearly Interpolated LUT Default = CORDIC
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Positive Integer	[4 to 32] Default = [auto]
Iterations	This parameter dictates the number of iterations to use in the Newton Raphson or Goldschmidt algorithm.	Positive Integer	Range = [1, 2 or 3] Default = [auto]
Init Lut Bits	Specifies the number of MSB bits to use for the initialization table when Newton Raphson or Goldschmidt implementations are specified.	Positive Integer	Range 2 to 32 Default = [default]

Implementation Parameters			
Name	Description	Type	Range
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.	Positive Integer	Range = 2 to 32 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.	Positive Integer	Range 2 to 32 Default = auto
Output bitwidth	The number of bits in the output word(s).	Positive Integer	Range 4 to 32 Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
SigIn_N	The numerator of the division operation.	Real	The range of the numbers is defined by the SigIn_N quantizer settings.
SigIn_D	The denominator of the remainder operation.	Real	The range of the numbers is defined by the SigIn_D quantizer settings.

Output(s)			
Name	Description	Type	Range
SigOut	An output stream of numbers that represent the result of the remainder operation.	Real	<p>If either the numerator or the denominator is type 'fixed', the range of the output will be equal to $[-2^{\text{OBFITS}-1}, -2^{\text{OBFITS}-1}]/2^{\text{OBFITS}}$</p> <p>However, if both inputs are type 'ufixed', the range of the output will be equal to $[0, -2^{\text{OBFITS}-1}]/2^{\text{OBFITS}}$</p> <p>The number of fractional bits (OBFITS) is automatically selected to cover the maximum possible output value.</p>
DivZero	An output stream of numbers that will always be 0 unless y is zero, in which case DivZero = 1.	Integer	Range = [0 to 1]

reshape

Supported for synthesis in this release for up to 2 dimensions.

rot90

Supported for synthesis.

round

Supported for synthesis.

rsdec

Communications Toolbox function supported for synthesis.

rsenc

Communications Toolbox function supported for synthesis.

sign

Sign is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

sin/accel_sin

Description

Sine Function

Related MATLAB Functions

[sin](#)

Differences with the MATLAB Function

1. The inputs must be pre-quantized to the accuracy specified by the input quantizer.
2. The shape of the input is limited to a scalar, vector or 2-D matrix.

Extended Features of accel_sin()

The accel_sin() function provides the same functionality as the sin() function plus the following extended features:

1. The accel_sin() function can create an optional cos() output with little additional hardware.
2. The accel_sin() function can specify input units of measure ([InputType](#)) in the following formats:
 - ♦ [native form](#)
 - ♦ [scaled form](#)

Syntax

AccelDSP sin() Function	Supported MATLAB Syntax
<code>y = sin(x);</code>	<code>y = sin(x);</code>

AccelDSP accel_sin() Function Call	Notes
<code>y = accel_sin(x);</code>	<code>y = sin(x)</code>
<code>[y,z] = accel_sin(x);</code>	<code>z = cos(x);</code>

Parameters

Input Quantization			
Name	Description	Type	Range
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word.	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 7

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC
InputType	Specify the representation of the input word.	String	{ native form scaled form } Default = native form

Implementation Parameters			
Name	Description	Type	Range
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses OBITS to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 100] Default = [auto]
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.		Range = 1 to 6 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 1 to 32 Default = auto
Output bitwidth	The number of bits in the output word(s).		Range 4 to 32 Default = auto
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	CORDIC Bipartite Tables Linearly Interpolated LUT Default = CORDIC

Inputs / Outputs

sin() Input			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix.	Real	Range = [0 to 2π] or [$-\pi$ to π] depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

accel_sin() Input			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix. (For <code>InputType = native form</code>).	Real	Range = $[0 \text{ to } 2\pi]$ or $[-\pi \text{ to } \pi]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.
	May be a scalar, vector or 2-D matrix. (For <code>InputType = scaled form</code>).	Positive Integer	Range = $[0 \text{ to } 2^{\text{IBITS}-1}]$ or $[-2^{\text{IBITS}-1} \dots 2^{\text{IBITS}-1}-1]$ depending on the Sign Mode of the Input Quantizer, 'ufixed' and 'fixed', respectively.

sin() Output			
Name	Description	Type	Range
y	May be a scalar, vector, or 2-D matrix representing the sine of the corresponding input element. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

Additional accel_sin() Output			
Name	Description	Type	Range
z	An optional output scalar, vector, or 2-D matrix representing the cosine of the corresponding input. OBITS = Output bitwidth.	Real	Range = $[-2^{\text{OBITS}-1} \text{ to } 2^{\text{OBITS}-1}-1]/2^{\text{OFBITS}}$

size

Supported for synthesis in this release when the argument resolves to a constant.

Example 1

```
a = [4 9 7 ; -12 10 8];
b = size(a,1)
```

Example 2

```
a = [4 9 7 -12 10 8];
[b c] = size(a)
```

Example 3

```
a = [4 9 7 ; -12 10 8];  
b = size(a)
```

sqrt

Description

Square Root

Related MATLAB Functions

[sqrt](#)

Differences with the MATLAB Function

In MATLAB, the input <x> is a vector of complex numbers passed to sqrt() in parallel with the output <y> occurring as a vector of complex numbers in parallel as well. Since this is not practical in hardware, in the sqrt() function, the input <x> occurs as a serial stream of IBITS-bit real un-signed numbers and the output <y> occurs as serial stream of OBITS-bit real un-signed numbers.

Syntax

AccelDSP Function Call	Supported MATLAB Syntax
y = sqrt(x); [y inv_y] = sqrt(x);	y = sqrt(x);
	Unsupported MATLAB Syntax
	Complex and Negative inputs are not supported in this version due to the complex outputs generated by the sqrt function.

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point model. Infers that the input range is -PI to PI. ufixed : unsigned fixed-point mode. Infers that the input range is 0 to 2*PI.	String	Mode = fixed ufixed Default = fixed
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed

Input Quantization			
Name	Description	Type	Range
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Default = 10

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. CORDIC Bipartite Tables (more) Linearly Interpolated Lookup Tables (more) Newton-Raphson Newton-Raphson with Inverse	String	CORDIC Bipartite Tables Linearly Interpolated LUT Newton-Raphson Newton-Raphson with Inverse Default = CORDIC
CORDIC Iterations	Selects the number of CORDIC iterations to perform. 'auto' can be used to automatically select the number of iterations to perform. 'auto' uses the Output bitwidth to compute the number of iterations required to ensure the maximum error is ~1 LSB	Integer	[4 to 32] Default = [auto]
Bipartite Sections	Bipartite Tables : Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimized the table sizes.		Range = 2 to 5 Default = auto

Implementation Parameters			
Name	Description	Type	Range
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.		Range 2 to 32 Default = auto
Iterations	This parameter dictates the number of iterations to use in the Newton Raphson algorithm.	Positive Integer	Range = [1, 2 or 3] Default = [auto]
Output bitwidth	The number of bits in the output word(s).		Range 4 to 31 Default = auto

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix. Each element is IBITS-bits wide.	Positive Integer	Range = [0 to $2^{\text{IBITS}} - 1$] $/2^{\text{IFBITS}}$

Output(s)			
Name	Description	Type	Range
y	A scalar, vector or 2-D matrix with each element representing the sqrt(x) of the corresponding element in the input. Each element is OBITS-bits wide.	Real	Range = [0 to $2^{\text{OBITS}} - 1$] $/2^{\text{OFBITS}}$
inv_y	An optional output representing the 1/sqrt(x) of each element in the input. Each output element is OBITS-bits wide.	Real	Range = [0 to $2^{\text{OBITS}} - 1$] $/2^{\text{OFBITS}}$

std

Standard Deviation function supported for synthesis.

structure

Create a structure inside the design function. Refer to ["structure' Data Type,"](#) for details.

sum

Sum is supported for scalars, vectors or two-dimensional matrices containing both positive and negative numbers.

svd/accel_svd

Description

Computes the matrix singular value decomposition of the input matrix and returns a vector of singular values.

Related MATLAB Functions

[svd](#)

Differences with the MATLAB Function

The MATLAB svd function can accept a complex input matrix. The AccelDSP svd function accepts only real-valued, square input matrices.

The MATLAB svd function arranges the output singular values in descending order, and the singular vectors in U and V are also arranged to match the ordering of the singular values. The function does not sort the singular values in order. When you re-order or sort the singular values, you should take care to apply the same re-ordering to the corresponding singular vectors to maintain a one-to-one correspondence between singular values and vectors.

The MATLAB svd function can produce 'economy size' decompositions. The AccelDSP svd function always produces a full decomposition according to the input matrix size.

Syntax

Function Call	Supported MATLAB Syntax
[U,S, V] = svd(X);	[U,S,V] = svd(X); Produces a diagonal matrix S, of the same dimension as X and with nonnegative diagonal elements, and unitary matrices U and V so that $X = U*S*V'$
[U,S, V] = accel_svd(X);	Function signature recognized for the auto-interference of the AccelDSP SVD core. Produces a diagonal matrix S, of the same dimension as X and with nonnegative diagonal elements, and unitary matrices U and V so that $X = U*S*V'$ X must be a square, real-valued matrix.

Unsupported MATLAB Syntax	Notes
S = svd(X)	Returns a vector containing the singular values.
[U,S,V] = svd(X,0)	No 'economy' size decomposition is supported.
[U,S,V] = (X,'econ')	No 'economy' size decomposition is supported.

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round to the closest representable number in the direction of negative infinity nearest : round to the closest representable integer. In the case of a tie, it rounds positive numbers to the closest representable integer in the direction of positive infinity, and it rounds negative numbers to the closest representable integer in the direction of negative infinity.	String	RoundMode = [floor nearest] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap
Word Length	The number of bits for the input word(s).	String	Range [8 - 16] Default = 14
Fractional Length	The number of fractional bits for the input word(s).	String	Range [0 - 16] Default = 13

Implementation Parameters			
Name	Description	Type	Range
Implementation	Type of implementation	String	Jacobi
Type	Input matrix data type	String	double
Matrix Symmetry	Input matrix symmetry properties	String	[symmetric non-symmetric]
Matrix Size (2:16)	Input matrix size	Integer	Range = [2 to 16] Default = 4

Implementation Parameters			
Name	Description	Type	Range
Data IO Format	Selects the format for the IO streaming of data (array or sample per invocation);it affects the test design function wrapper only.	String	[array sample] Default = array
Sweeps	Number of sweeps of Jacobi rotations	Integer	Range = [2 to 16] Default = auto 'auto' selects sweeps based on matrix size

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
X	Input matrix to be factored. XBITS and XFBITS are the input quantization wordlength and fractional length as per A quantization parameters.	Real	Range = $[-2^{XBITS-1}$ to $2^{XBITS-1-1}]/2^{XFBITS}$

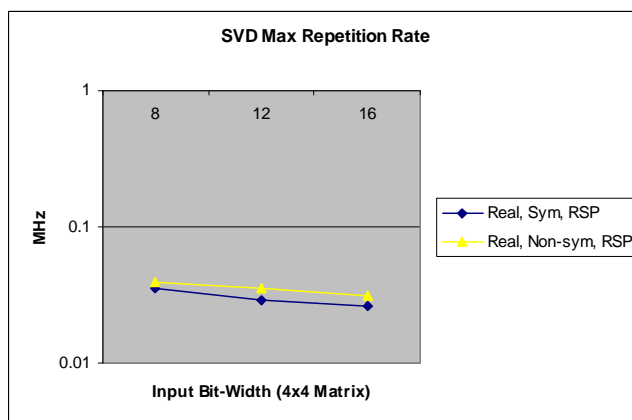
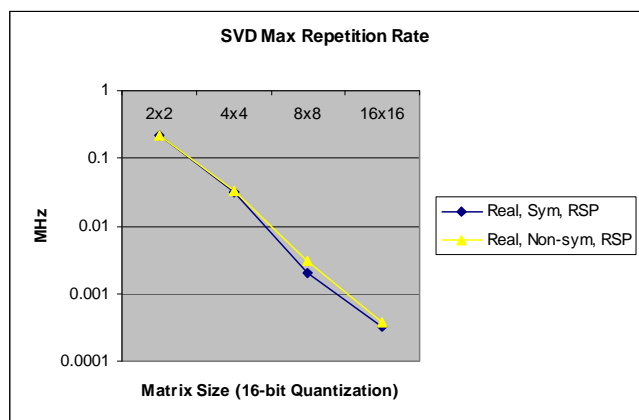
Output(s)			
Name	Description	Type	Range
U	Orthogonal matrix with left singular vectors. UBITS and UFBITS are the U matrix output wordlength and fractional length.	Real	Range = $[-2^{UBITS-1}$ to $2^{UBITS-1-1}]/2^{UFBITS}$
D	Diagonal matrix whose elements are the singular values of the original matrix. DBITS and DFBITS are the D matrix output wordlength and fractional length.	Real	Range = $[-2^{DBITS-1}$ to $2^{DBITS-1-1}]/2^{DFBITS}$
V	Orthogonal matrix with right singular vectors. VBITS and VFBITS are the V matrix output wordlength and fractional length.	Real	Range = $[-2^{VBITS-1}$ to $2^{VBITS-1-1}]/2^{VFBITS}$

Expected Quality of Results

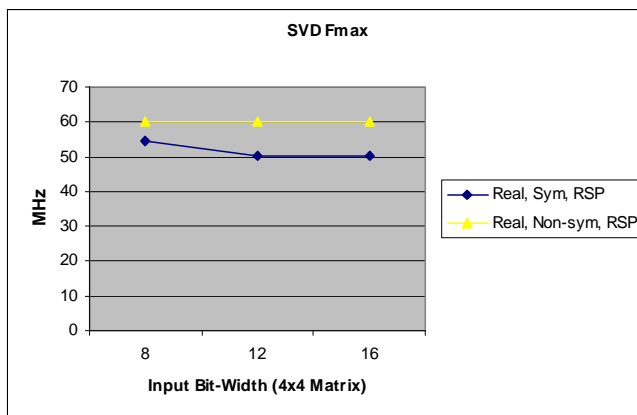
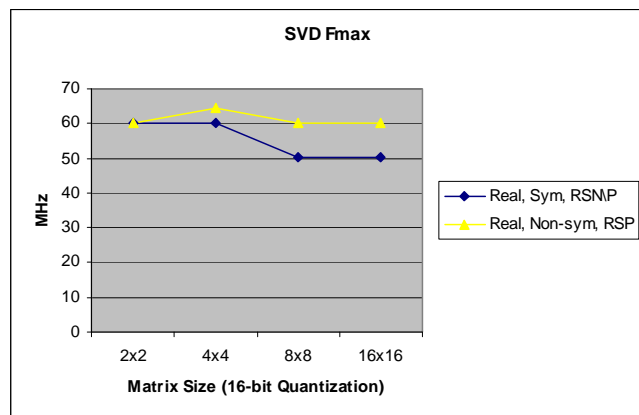
The following plots show typical speed and utilization of the SVD core when mapped to a Virtex4 XC4VSX55 device. The acronyms RSNP and NRSP refer to resource-shared, non-pipelined architectures and non-resource-shared, pipelined architectures respectively.

Speed of Operation

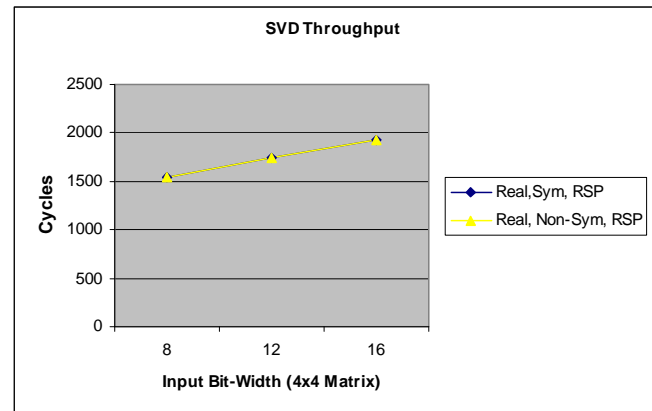
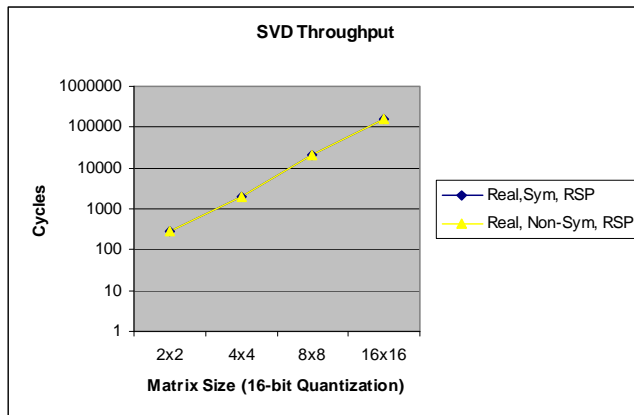
Decomposition Repetition Rate



Maximum Operating Frequency

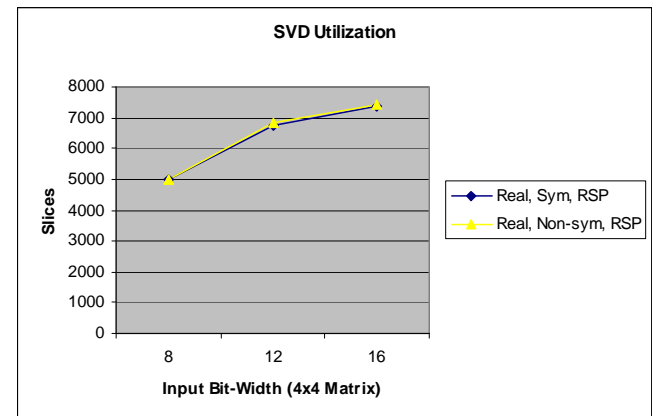
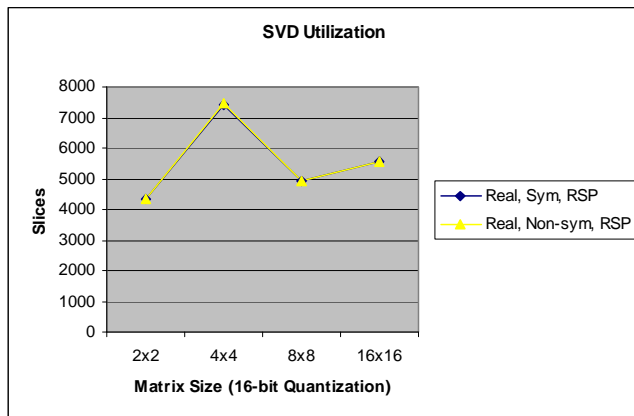


Hardware Clock Cycles per Design Function Call

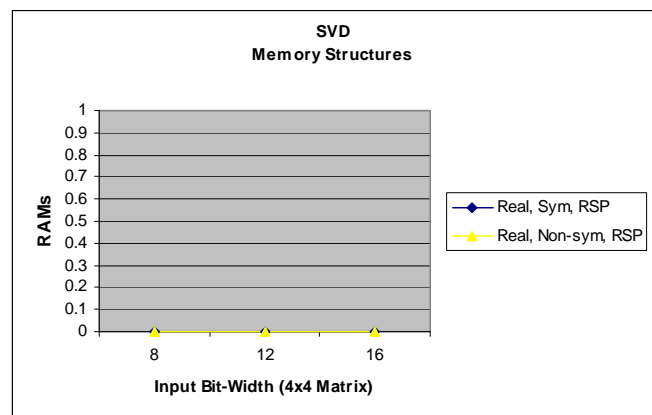
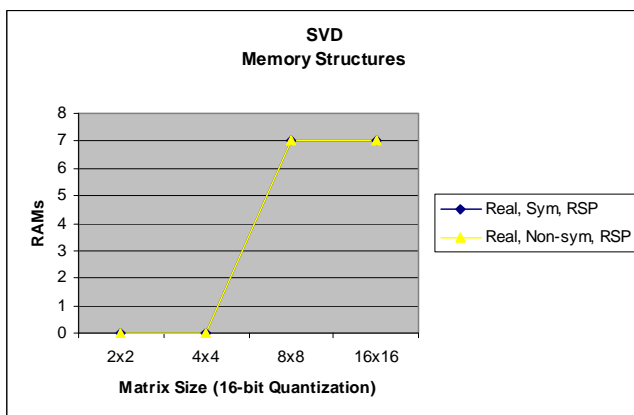


Utilization

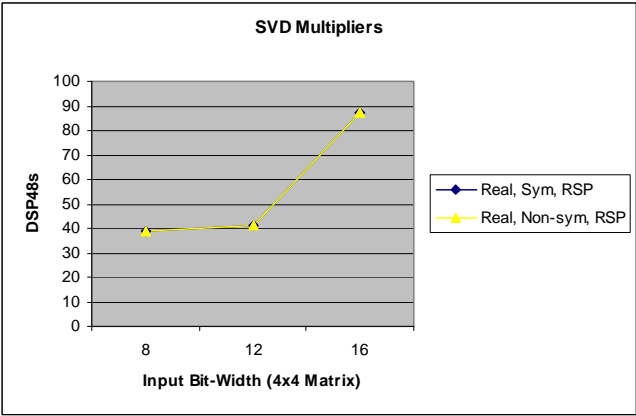
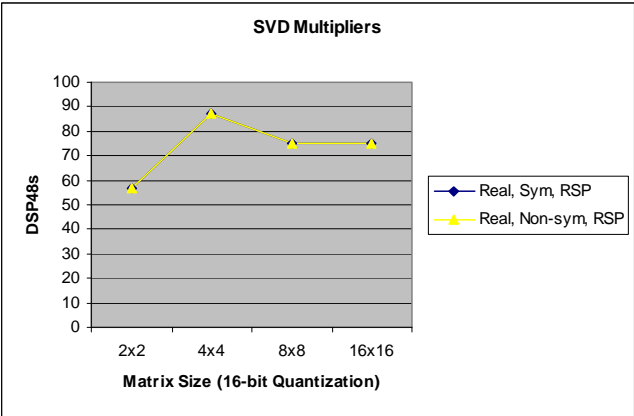
Number of Slices



Number of Memories



Number of Multipliers



switch

Supported for synthesis.

Example

```
switch (x)
    case (0)
        y = y + 1;
    otherwise (1)
        y = y + 2;
end
```

tan

Description

Tangent Function

Related MATLAB Functions

[tan](#)

Differences with the MATLAB Function

1. The inputs must be pre-quantized to the accuracy specified by the Input Bit Width.
2. The shape of the input is limited to a scalar, vector or 2-D matrix.

Extended Features of accel_sin()

The accel_tan() function provides the same functionality as the tan() function plus the following extended features:

1. The accel_tan() function allows you to manually set the Input Bit Width.

Syntax

AccelDSP tan() Function Call	Supported MATLAB Syntax
y = tan(x);	y = tan(x);
AccelDSP accel_tan() Function Call	Notes
y = accel_tan(x);	y = tan(x)

Parameters

Functional Parameters			
Name	Description	Type	Range
Input Bit Width	The number of bits for the input word.	String	Range [4 to 24] Default = 10

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create. Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	{ Bipartite Tables Linearly Interpolated LUT} Default = Bipartite Tables
Bipartite Sections	Bipartite Tables: Specify the bit width parameters to use for the Bipartite Tables [A B C] or just [A]. When [A B C] is specified A+B+C must equal IBITS. If just [A] is specified B and C will be selected to minimize the table sizes.	String	Range = 1 to 5 Default = auto
Address bits	Linearly Interpolated LUT: max(16, N), where N is the value required to limit the number of address bits to 14.	String	Range 1 to 10 Default = auto
Output bitwidth	The number of bits in the output word(s).	String	Range 4 to 32 Default = auto
Implementation	Select the hardware architecture to create. Bipartite Tables (more) Linearly Interpolated Lookup Tables (more)	String	{ Bipartite Tables Linearly Interpolated LUT} Default = Bipartite Tables

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
x	May be a scalar, vector or 2-D matrix.	Real	

Output(s)			
Name	Description	Type	Range
y	May be a scalar, vector, or 2-D matrix representing the sine of the corresponding input element. OBITS = Output bitwidth.	Real	Range = $[-2^{OBITS-1}$ to $2^{OBITS-1}-1]/2^{OFBITS}$

times/accel_complex_times

Description

Return the result of element-by-element multiplication.

Related MATLAB Functions

[times](#)

Syntax

AccelDSP mtimes() Function	Supported MATLAB Syntax
<code>C = times(A,B);</code>	<code>C = times(A,B);</code>

AccelDSP accel_complex_mtimes() Function Call	Notes
<code>[C_real,C_imag] = accel_complex_times(A_real,A_imag,B_real,B_imag);</code>	

Parameters

Implementation Parameters			
Name	Description	Type	Range
Implementation	Select the hardware architecture to create.	String	{ vector-vector vector-scalar scalar-vectorT} Default = vector-vector

Inputs / Outputs

times() Inputs			
Name	Description	Type	Range
A	May be a scalar, vector or 2-D matrix.	Real	
B	May be a scalar, vector or 2-D matrix.	Real	

accel_complex_times() Inputs			
Name	Description	Type	Range
A_real	May be a scalar, vector or 2-D matrix.	Real	
A_imag	May be a scalar, vector or 2-D matrix.	Real	
B_real	May be a scalar, vector or 2-D matrix.	Real	
B_imag	May be a scalar, vector or 2-D matrix.	Real	

times() Output			
Name	Description	Type	Range
C	Result of matrix multiplication.	Real	

accel_complex_times() Outputs			
Name	Description	Type	Range
C_real	Result of matrix multiplication.	Real	
C_imag	Result of matrix multiplication.	Real	

true

Supported for synthesis.

Example 1

```
A = true;
```

Example 2

```
A = true(1,5);
```

var

Description

Variance

Related MATLAB Functions

[var](#)

Differences with the MATLAB Function

In MATLAB, the length of <x> may vary from call to call. This var() function requires the length of <x> to be constant from call to call. The function can be created with a known mean value on the input sequence, which significantly reduces the complexity of the hardware.

Syntax

AccelDSP Function Call	Supported MATLAB Syntax
<code>y = var(x,flag);</code> <code>y = var(x,w,dim);</code>	<code>y = var(x);</code> <code>y = var(x,w,dim);</code>

Parameters

Input Quantization			
Name	Description	Type	Range
Sign Mode	fixed : signed fixed-point mode ufixed : unsigned fixed-point mode.	String	Mode = fixed ufixed Default = fixed
Round Mode	floor : round both positive and negative number toward positive infinity round : round to the nearest allowable quantized value. Numbers that are halfway between the two nearest allowable quantized values are rounded up.	String	RoundMode = [floor round] Default = floor
Overflow Mode	wrap : wrap on overflow. saturate : saturate a maximum value on overflow.	String	OverflowMode = [wrap saturate] Default = wrap

Input Quantization			
Name	Description	Type	Range
Word Length	The number of bits for the input word(s).	String	Default = 10
Fractional Length	The number of fractional bits for the input word(s).	String	Default = 5

Implementation Parameters			
Name	Description	Type	Range
N (2:65000)	Length of the input vector.	Positive Integer	Range = [2 to 65000] Default = 16
Flag (0:1)	Sets the normalization factor to use. '0' causes the value to be normalized by N-1, while '1' causes the value to be normalized by N.	Binary	Range = 0 1 Default = 0
Output bitwidth	Number of bits used to represent the output y.	Positive Integer	Range = [4 to 32] Default = auto
Mean	If a number is supplied for the mean(x), no hardware will be implemented to compute the mean value. Instead, the value supplied will be directly substituted into the var equation. Hardware will be used to calculate the mean if this parameter is set to "Computed".	Real	Default = Computed
Data I/O Format	Scalar, one sample at a time; Array, N array of samples at a time. 'Array' is typically used when var is embedded in a design.	String	Range = Scalar Array Default = Scalar

Inputs / Outputs

Input(s)			
Name	Description	Type	Range
x	An input stream of vectors to compute the variance. Each element in the vector is IBITS-bits wide with IFBITS-bits representing the fractional part.	Real	Range = $[0 \text{ to } 2^{\text{IBITS}} - 1]$ $/2^{\text{IFBITS}}$

Output(s)			
Name	Description	Type	Range
y	An output stream of numbers each OBITS-bits wide with OFBITS-bits representing the fractional part. OFBITS is automatically selected.	Real	Range = $[0 \text{ to } 2^{\text{OBITS}} - 1]$ $/2^{\text{OFBITS}}$
z	An optional output stream of numbers each OBITS-bits wide with OFBITS-bits representing the fractional part. OFBITS is automatically selected.	Real	Range = $[0 \text{ to } 2^{\text{OBITS}} - 1]$ $/2^{\text{OFBITS}}$

while

Supported for synthesis.

Example

```
while (i < 5)
    a = a + 10;
    i = i + 1;
end
```

zeros

Supported for synthesis in this release for up to two dimensions.

Example 1

```
a = zeros;
```

Example 2

```
a = zeros(1,5);
```


Table of Synthesizable MATLAB Constructs

Programming with MATLAB

Data Types and Quantize Functions

Name	Description
Scalar arrays	An array with dimensions 1 x 1
Vector arrays	An array with dimensions 1 x n
Matrix arrays	An array with dimensions m x n
structure	Create a structure inside the design function
char	Limited to String Constants
gf	Create a Galois field array
quantize	Defines the fixed-point parameters of a variable
quantizer	Applies the quantize object to a variable

Flow Control

Name	Description
if-elseif-else	Evaluates an expression and executes a set of commands
for / end	Repeats a set of commands a specified number of times
switch-case	Executes a set of commands based on an expression
while / end	Repeats a set of commands until a logical condition is false
end	Defines the end of an IF, FOR and WHILE statement
otherwise	Supported as part of the switch statement.

Scripts and Functions

Name	Description
function	Defines a function containing executable MATLAB
persistent	Define persistent variable
%	Comment

Basic Information

Name	Description
eps	Floating-point relative accuracy
isempty	True for an empty matrix. Supported for the initialization of persistent variables.
length	Length of a vector
ndims	Number of array dimensions
size	Size of a matrix

Array Operations and Manipulations

Name	Description
:	Index into array
dot	Scalar product of two vectors
end	Last index
max	Max elements of array
min	Min elements of array
reshape	Use to modify the shape of a matrix
inv	Matrix Inverse.
norm	Vector and matrix norm
mean	Return the mean of a matrix or array elements
all	Test to determine if all elements are non-zero
any	Test for non-zeros
sum	Sum of an array of elements
cumsum	Cumulative sum along different dims
prod	Product of the elements of an array
cumprod	Cumulative product of the elements of an array
fliplr	Flip matrices left-right
flipud	Flip matrices up-down
rot90	Rotate matrix 90 degrees
diff	Differences and approximate derivatives
cat	Concatenation. Limited to two variables

Elementary Matrices and Arrays

Name	Description
:	Regularly spaced vector
eye	Identity matrix
ones	Create array on all ones
zeros	Create array of all zeros
bi2de	Convert input matrices to decimal numbers. Input is restricted to base 2 numbers.
de2bi	Convert decimal numbers to binary vectors

Opening, Loading, Saving Files

Name	Description
load	load workspace from disk

Mathematics

Mathematical Operators

Name	Description
+, plus, accel_complex_plus	Addition
-, minus, accel_complex_minus	Subtraction
.*, times, accel_complex_times	Array multiplication
*, mtimes, accel_complex_mtimes	Matrix multiplication
.^, power	Array power
^, mpower	Matrix power where the exponent must be a scalar integer
pow2	Base 2 power and scale floating-point number
nextpow2	Next power of two
./	Right array divide
/, rdivide	Right matrix divide
.\	Left array divide
\, ldivide	Left matrix divide
'	Transpose
.'	Noconjugated transpose

Relational Operators

Name	Description
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

Name	Description
<code>==, eq</code>	Test for equality
<code>~=, ne</code>	Not equal

Logical Operators

Name	Description
<code>&&</code>	Logical AND
<code> </code>	Logical OR
<code>&</code>	Logical AND for arrays
<code> </code>	Logical OR for arrays
<code>~</code>	Logical NOT
<code>false</code>	False array
<code>true</code>	True array

MATLAB Bit-wise Operators

Name	Description
<code>bitand</code>	Bitwise and
<code>bitcmp</code>	Bitwise compare. Overloaded with <code>accel_bitcmp</code>
<code>bitget</code>	Bitwise get
<code>bitor</code>	Bitwise or
<code>bitset</code>	Bitwise set
<code>bitshift</code>	Bitwise shift. Overloaded with <code>accel_bitshl</code> and <code>accel_bitshr</code> .
<code>bitxor</code>	Bitwise xor

AccelDSP Bit-wise Operators

Base Function	Description
accel_bitand	Returns the unsigned bit-wise AND of two integers
accel_bitcmp	Returns the unsigned bit-wise complement of an integers
accel_bitmerge	Cincatenates two components, MSB (nost-significant bits) and LSB (least-significant bits) into one output word
accel_bitnand	Returns the unsigned bit-wise NAND of two integers
accel_bitnor	Returns the unsigned bit-wise NOR of two integers
accel_bitor	Returns the unsigned bit-wise OR of two integers
accel_bitpack	Returns a single value-representation of the bit-vector argument x quantized to quantizer q
accel_bitrev2	Return the unsigned digit reversal of the argument IN
accel_bitrev	Return the unsigned bit-wise reversal of the argument IN
accel_bitshl	Returns the unsigned bit-wise shift-left of the argument IN shifted left by N bits
accel_bitshr	Returns the unsigned bit-wise shift-right of the argument IN shifted right by N bits
accel_bitsplit	Splits the input value into MSB (most-significant bits) and LSB (least-significant bits) components according to the bitwidths specified in the specified quantizers
accel_bitunpack	Returns a row-vector bit-representation of the scalar argument x quantized to the quantizer qout
accel_bitunpackselect	Returns the i'th element from the output of accel_bitunpack(x,q)
accel_bitxor	Returns the unsigned bit-wise XOR of two integers

Linear Algebra

Name	Description
chol	Matrix factorization using Cholesky method.
inv	Matrix Inverse (QR,Cholesky, Upper Triangular).
qr	Matrix factorization using QR Decomposition method.
qrdrls	QRD-RLS Spatial Filter.
svd	Singular value decomposition.
vector rotation	Performs a givens rotation on a vector pair.
Triangular System of Equations Solver	Computes the solution to an upper or lower triangular system of equations using backward or forward substitution, respectively.

Statistics

Name	Description
mean	Mean value.
norm	norm.
std	Standard Deviation.
var	Variance.

Trigonometric Functions

Name	Description
sin, accel_sin	Sine
cos, accel_cos	Cosine
tan, accel_tan	Tangent
asin, accel_asin	Inverse sine
acos, accel_acos	Inverse cosine
atan, accel_atan	Inverse tangent
atan2, accel_atan2	Four-quadrant inverse tan

Polynomials

Name	Description
polyval	Returns the value of a polynomial.

Exponential Functions

Name	Description
exp	Exponential
log	Natural logarithm
log10	Base 2 logarithm
log2	Base 10 logarithm
pow2	Base 2 power
power	Array power. Same as $X.^Y$
mpower	Matrix power where the exponent must be a scalar integer. Same as X^Y where Y is a scalar integer.
reallog	Base e logarithm
sqrt	Square root
realsqrt	Square root for non-negative real arrays
Inverse Square Root	Returns $1/\text{sqrt}(x)$.
realpow	Array power for real output

Complex Numbers

Name	Description
abs, accel_abs	Absolute value
angle, accel_angle	Returns phase angle and magnitude
cart2pol	Cartesian coordinates to polar or cylindrical
complex norm	Complex Normalization.
pol2cart	Polar or cylindrical coordinates to Cartesian
conj	Complex conjugate of the elements of the array
complex	Construct complex data from real and imaginary components
imag	Imaginary part of a complex number
real	Real part of a complex number

Rounding and Remainder

Name	Description
ceil	Round towards positive infinity
convergent	Round to nearest integer
fix	Round towards zero
floor	Round towards negative infinity
mod	Modulus after division
rem	Remainder after division
nearest	Round towards integer
sign	Signum

Discrete Math

Name	Description
factorial	Is the product of all the integers from 1 to n

Math Constants

Name	Description
pi	Ratio of a circle's circumference / diameter

Signal Processing Library

Filters - FIR - General Purpose

Name	Description
dfilt	Discrete-Time Filters.
filter	Fixed coefficient FIR filter.
filter - loadcoef	Loadable coefficients FIR filter.
filter - multchan	Multi-channel FIR filter.

Filters - Multirate

Name	Description
cicdecim	Cascaded Integrator-Comb decimation filter.
cicinterp	Cascaded Integrator-Comb interpolation filter.
mfilt.firdecim	Construct direct-form FIR polyphase decimator filter.
mfilt.firtdecim	Construct direct-form transposed FIR filter.
firhalfband	Half-band FIR filter.

Filters - Other

Name	Description
a_dsinccompensation	A/D Sinc Compensation filter.
cicdecimate	Cascaded Integrator-Comb (CIC) decimation filter.
cicinterpolate	Cascaded Integrator-Comb (CIC) interpolation filter.
rcosfir	Root-Raised Cosine (RRC) filter.

Transformations

Name	Description
fft - radix 2	Fast-Fourier transform.
ifft - radix 2	Inverse Fast-Fourier transform.
fft - radix 4	Fast-Fourier transform.
ifft - radix 4	Inverse Fast-Fourier transform.

Communications Library

Direct Digital Synthesizers

Name	Description
dds	Direct Digital Synthesizer.

Encoders/Decoders

Name	Description
convenc	Convolutional encoder.
convintlv	Convolutional interleaver.
convdeintlv	Convolutional deinterleaver.
rsenc	Reed Solomon/BCH encoder.
rsdec	Reed Solomon/BCH decoder.
vitdec	Viterbi decoder.
bchenc	BCH encoder.
bchdec	BCH decoder.

Scramblers / Descramblers

Name	Description
scrambler	Custom 16-bit wide scrambler.
descrambler	Custom 16-bit wide scrambler.

Index

- (subtraction) 40

Symbols

41
% (comment) 42
& (logical AND for arrays) 42
&& (logical AND) 42
* (multiplication) 40
+ (addition) 40
./ (array right divide) 41
.\ (array left divide) 41
/ (right division) 40
==, eq (equal) 42
> (greater than) 41
>= (greater than or equal) 41
\ (left division) 41
^ (array power) 41
^ (matrix power) 41
| (logical OR for arrays) 42
|| (logical OR) 42
~ (logical NOT) 42
~=, ne (not equal) 42

A

a_dsincocomensation 42
abs 43
accel_abs 43
accel_angle 110
accel_asin 114
accel_atan 117
accel_atan2 120
accel_chol_factor 62
accel_cmplxrot 67
accel_complex_minus 160
accel_complex_mtimes 166
accel_complex_plus 169
accel_complex_times 209
accel_cos 107, 136
accel_exp 141
accel_givens_rotation 68
accel_power 175
accel_qr_factor/accel_complex_qr_factor 73
accel_qr_inverse/accel_complex_qr_inve
rse 81, 89, 97
accel_sin 192

AccelDSP Base Library Functions

accel_bitand 47
accel_bitcmp 48
accel_bitmerge 49
accel_bitnand 50
accel_bitnor 51
accel_bitor 52
accel_bitpack 53
accel_bitrev 55
accel_bitrev2 54
accel_bitshl 56
accel_bitshr 57
accel_bitsplit 58
accel_bitunpack 59
accel_bitunpackselect 60
accel_bitxor 61
acos 107
all 110
angle 110
any 113
asin 114
atan 117
atan2 120

B

bchdec 123
bchenc 123
bi2de 123
bitand 124
bitcmp 124
bitget 124
bitor 124
bitset 124
bitshift 124

C

cart2pol 124
case 124
cat 125
ceil 125
Char
 data type 28
char 125
 data type 26, 28, 29
chol 125

chol_inverse/accel_complex_chol_invers
e
 accel_qr_inverse/accel_complex_qr_in
verse 126
cicdecimate 131
cicinter 131
colon 135
complex 132
Complex Normalization 132
Complex Rotation 67
conj 134
Constant Data 25
Constant Variables
 initializing 26
convdenintlv 134
convenc 134
convergent 134
convinctlv 134
cos 136
cumprod 139
cumsum 139

D

Data Type
 char 26, 28, 29
 double 26
 gf 29
 structure 26
Data Types
 for I/O ports 21
 not supported 32
 supported 26
de2bi 140
Design Body Function Call
 inputs and outputs 22
Design Function
 mapping to hardware elements 23
dfilt 140
diff 140
dot 140
Double
 data type 26

E
else 140
elseif 140
end 141

eps 141
eq 141
exp 141
eye 144

F

factorial 145
false 42, 145
fft 145
filter 145
firhalfband 145
fix 145
fliplr 145
fliprl 145
flipud 145
floor 146
for 146
for loop 21
function 147
Function call
 design body inputs and outputs 22
 top-level design inputs and outputs 21
Function M-file 20, 21

G

Galois Field
 data type 29
 primitive polynomials 30
 rules for using 31
 type propagation 30
gf 147
Givens Array Rotation 147
Global variables
 not currently supported 26, 147

H

hypot 147

I

I/O
 data types 21
if 147
ifft 148
imag 149
Input Ports 23
InputType

native form 36
native from vs scaled form 37
scaled form 36
inv 149
Inverse Square Root 149
isempty 149

L

ldivide 149
length 149
load 149
log 149
log10 153
log2 156

M

MATLAB construct
 41, 41
 - (subtraction) 40
 % (comment) 42
 & (logical AND for arrays) 42
 && (logical AND) 42
 * (multiplication) 40
 + (addition) 40
 ./ (array right divide) 41
 .> (greater than) 41
 .>= (greater than or equal) 41
 .\ (array left divide) 41
 .^ (array power) 41
 / (right division) 40
 ==, eq (equal) 42
 \ (left division) 41
 ^ (matrix power) 41
 | (logical OR for arrays) 42
 || (logical OR) 42
 ~ (logical NOT) 42
 ~=, ne (not equal) 42
abs 43
accel_chol_factor 62
accel_givens_rotation 68
accel_qr_factor/accel_complex_qr_factor 73
accel_qr_inverse/accel_complex_qr_inverse 81
accel_triang_inverse/accel_complex_triang_inverse 89
accel_triang_solver/accel_complex_triang_solver 97
acos 107
all 110

angle 110
any 113
asin 114
atan 117
atan2 120
bchdec 123
bchenc 123
bi2de 123
bitand 124
bitcmp 124
bitget 124
bitor 124
bitshift 124
bitxor 124
cart2pol 124
case 124
cat 125
ceil 125
char 125
chol 125
chol_inverse/accel_complex_chol_inverse 126
colon 135
convergent 134
cos 136
cumprod 139
cumsum 139
de2bi 140
diff 140
dot 140
else 140
elseif 140
end 141
eps 141
eq 141
exp 141
eye 144
factorial 145
false 42, 145
fft 145
filter 145
fix 145
fliplr 145
fliprl 145
flipud 145
floor 146
for 146
function 147
gf 147
hypot 147
if 147
ifft 148

inv 149
 isempty 149
 ldivide 149
 length 149
 load 149
 log 149
 log10 153
 log2 156
 max 159
 mean 159
 min 159
 minus 160
 mod 162
 mpower 165
 mtimes 166
 ndims 167
 ne 167
 nexpow2 167
 norm 167
 ones 168
 otherwise 168
 pause 168
 persistent 168
 pi 168
 plus 169
 pol2cart 170
 pow2 175
 power 175
 prod 177
 qr 177
 qrdr1s 178
 quantize 182
 quantizer 182
 rdivide 182
 reallog 186
 realpow 187
 realsqrt 187
 rem 187
 reshape 191
 rot90 191
 round 191
 sign 192
 sin 192
 size 195
 sqrt 197
 sum 200
 switch 207
 tan 207
 times 209
 true 42, 211
 var 212
 while 214

zeros 214
 max 159
 mean 159
 mfilt.firdecim 159
 mfilt.firtdecim 159
 min 159
 minus 160
 mod 162
 mpower 165
 mtimes 166

N

Native Form Input Type 36
 ndims 167
 ne 167
 nexpow2 167
 norm 167

O

ones 168
 otherwise 168
 Output Ports 23

P

pause 168
 persistent 168
 Persistent variables
 are supported by AccelDSP 26
 pi 168
 plus 169
 pol2cart 170
 poly2trellis 170
 polyval 171
 pow2 175
 power 175
 Pre-analyse phase 21
 prod 177

Q

qr 177
 qrdr1s 178
 quantize 182
 quantizer 182

R

ralsqrt 187

rcosflt 182
 rdivide 182
 real 187
 reallog 186
 realpow 187
 rem 187
 reshape 191
 rot90 191
 round 191
 rsdec 191
 rsenc 191

S

Scaled Form Input Type 36
 Script M-file 20
 sign 192
 sin 192
 size 195
 Slice 21
 sqrt 197
 std 199
 Streaming Loop 21
 structure 199
 data type 26
 sum 200
 svd 201
 switch 207

T

tan 207
 times 209
 Top-Level Design Function Call
 inputs and outputs 21
 true 42, 211

V

var 212
 Variable Data 25
 Variables
 global 26, 147
 persistent 26

W

while 214

Z

zeros 214