Motorola DSP Developer's Kit

For Use with Simulink®

Modeling

Simulation

Implementation

The MATH WORKS Inc.

User's Guide

Version 1

How to Contact The MathWorks:

508-647-7000	Phone
508-647-7001	Fax
The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098	Mail
`http://www.mathworks.com`	Web
`ftp.mathworks.com`	Anonymous FTP server
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`subscribe@mathworks.com`	Subscribing user registration
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

Printing History:

March 2000	New for Version 1.0, Release 11.1 (Online only)
November 2000	Revised for Version 1.1, Release 12 (Online only)

Contents

2 Creating Motorola DSP MEX-Files

3 Motorola DSP MEX-File Programming Reference

Preface

Related Products and Documentation

Requirements

The Motorola DSP Developer's Kit is a multiplatform product, running on Microsoft Windows 95, Windows 98, Windows NT, and UNIX systems.

The Motorola DSP Developer's Kit requires:

- MATLAB® 6.0 (Release 12)
- Simulink® 4.0 (Release 12)

In addition, modified versions of the following Motorola Suite56™ DSP Simulators are provided:

- Motorola Suite56 DSP56300 Simulator 6.2.10
- Motorola Suite56 DSP56600 Simulator 6.2.9

To build the pre-existing functions available in the Motorola DSP Developer's Kit Toolboxes and Blocksets you need the following compilers:

Windows.

- Microsoft Visual C/C++ 6.0

UNIX.

- Sun's native CC 5.0

Note Other compilers may be used, however, they are not presently supported by the automated build process supplied. Using unsupported compilers requires that you customize the supplied build process, see "Custom Build Process" on page 1-8. However, there is no guarantee that the functions will integrate with the Motorola Suite56 DSP simulators.

Associated Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the Motorola DSP Developer's Kit. They are listed in the table below.

For more information about any of these products, see either:

- The online documentation for that product, if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at `http://www.mathworks.com`; see the "products" section

Note The products listed below complement the functionality of the Motorola DSP Developer's Kit.

Product	Description
DSP Blockset	Simulink block libraries for the design, simulation, and prototyping of digital signal processing systems
Signal Processing Toolbox	Tool for algorithm development, signal and linear system analysis, and time-series data modeling

Additional Reading

This guide does not attempt to repeat general information about Motorola DSP devices or software tools, nor the MATLAB and Simulink environments.

This guide should be read with reference to the following:

- *Motorola DSP Simulator Reference Manual*
- *Motorola DSP Assembler Reference Manual*
- *MATLAB Application Program Interface Guide*
- *Using Simulink*
- The MATLAB documentation
- *Writing S-Functions*

The *MATLAB Application Program Interface Reference* is also available online from the MATLAB Help Desk (enter `helpdesk` at the MATLAB prompt).

Also, Motorola DSP device and tools information can be found at:

```
http://www.motorola-dsp.com
```

Using This Guide

Expected Background

You are expected to be familiar with MATLAB and Simulink to use the basic features of the toolboxes and blocksets supplied with the Motorola DSP Developer's Kit. To take advantage of the advanced features of the Motorola DSP Developer's Kit as a powerful development tool, some knowledge of DSP devices and assembly and/or C coding is essential.

Organization of the Document

Use this guide in conjunction with the templates and software to learn about the features of the Motorola DSP Developer's Kit.

Chapter 1, "Introduction to the Motorola DSP Developer's Kit", provides a brief introduction to the Motorola DSP Developer's Kit describing the key features. It also describes the build process in detail, and shows you how to get started with the Motorola DSP Developer's Kit using a basic example.

Chapter 2, "Creating Motorola DSP MEX-Files", describes in detail how to write your own MATLAB MEX-file or Simulink S-function based on the templates provided. It also provides a step-by-step tutorial of advanced features, which shows you how to run MEX-files in *INTERACTIVE* mode in order to launch the Suite56 Simulators' graphical user interface (GUI).

Chapter 3, "Motorola DSP MEX-File Programming Reference", is a quick reference of available programming functions to assist you in writing your Motorola DSP Developer's Kit MEX-files or Simulink S-functions.

Chapter 4, "Motorola Toolbox Function Reference", contains information about all of the toolbox functions provided by the Motorola DSP Developer's Kit. This information allows you to incorporate these functions into one of your own existing system designs or to use them effectively, as supplied, to create new systems based on Motorola DSP devices.

Chapter 5, "Motorola Blockset Block Reference", describes the Motorola DSP Developer's Kit Blockset blocks. It also details the dialog box parameter options for each block.

The Appendix lists the directory structure and the files shipped with the Motorola DSP Developer's Kit.

Typographical Conventions

This manual uses some or all of these conventions.

To Indicate...	This Guide Uses...	Example
Example code	`Monospace` font	To assign the value 5 to `A`, enter `A = 5`
Function names/syntax	`Monospace` font	The `cos` function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Keys	**Boldface** with an initial capital letter	Press the **Return** key.
Mathematical expressions	*Italics* for variables Standard text font for functions, operators, and constants	This vector represents the polynomial $p = x^2 + 2x + 3$

MATLAB output	`Monospace` font	MATLAB responds with
		
		`A =`
		
		`   5`

Menu names, menu items, and controls	**Boldface** with an initial capital letter	Choose the **File** menu.
New terms	*Italics*	An *array* is an ordered collection of information.
String variables (from a finite list)	*`Monospace italics`*	`sysc = d2c(sysd, '`*`method`*`')`

Introduction to the Motorola DSP Developer’s Kit

Introduction

The Motorola DSP Developer’s Kit enables you to develop application software for Motorola DSPs in the MathWorks MATLAB and Simulink environments. It provides an object-oriented interface to program MEX-files or S-functions that call the appropriate Motorola Suite56 DSP Simulator.

With the Motorola DSP Developer’s Kit, you can develop implementation solutions based on the Motorola Suite56 DSP families. To achieve this, the tool allows you to implement algorithms in Motorola DSP assembly language or C code and run the generated object code directly within MATLAB or Simulink on the chosen Motorola Suite56 DSP simulator.

The Motorola DSP Developer’s Kit also provides toolboxes (MATLAB MEX -files) and blocksets (Simulink blocks based on S-functions) of commonly used DSP functions. Substituting existing MATLAB functions with the equivalent Motorola DSP functions at a behavioral level, you can evaluate the Motorola Suite56 family of DSPs.

In some situations, you use the toolbox and blockset functions of the Motorola DSP Developer’s Kit unmodified as supplied. These provide you with a set of basic functions for performing standard DSP operations. You can verify your system design by substituting specific Motorola DSP blocks or functions in place of the current MATLAB versions.

However, in the majority of cases, you design your own functions by modifying those supplied, or simply creating your own, based on the templates provided. This tailors your functions to the specific requirements of your application and allows efficient development of DSP algorithms running seamlessly on Motorola DSP simulators.

The Motorola DSP Developer's Kit

The Motorola DSP Developer's Kit is a powerful software development tool that features:

- Toolboxes and blocksets of example signal processing functions equivalent to existing MATLAB and Simulink functions
- Data import/export between the integrated Motorola Suite56 DSP simulator and MATLAB/Simulink environments
- Access to all features of the Motorola Suite56 DSP (DSP56300 and DSP56600) simulators
- A *NON_INTERACTIVE* operating mode that runs the integrated Motorola Suite56 DSP simulator with no visible user interface
- An *INTERACTIVE* operating mode that launches the GUI version of the Motorola Suite56 DSP simulator and allows assembly and C code debugging via direct access to DSP memory and registers
- Templates and build scripts to modify or create your own Motorola DSP MEX-files.

Getting Started

How to Get Help Online

There are a number of ways to get help on the Motorola DSP Developer's Kit from the available help directory.

- **Simulink Block Help**: Press the **Help** button in any block diagram parameter box to view the online reference documentation for that block.
- **Simulink Library Browser**: Right-click on a block name to access the help for that block.
- **Help Desk**: Select **Help** from the MATLAB **Help** menu, or type `helpdesk` or `doc` at the command line to access the Help Desk facility.
- **Release Information**: Type `whatsnew motdsp` or `info motdsp` at the MATLAB command line to view information about known software and documentation issues related to the version of the Motorola DSP Developer's Kit that you are using.

Note All data must be of normalized double-precision floating-point type (i.e., the data must lie between -1.0 and 1.0) before passing it to the DSP simulator via the input arguments of a MEX-file. Conversion to and from fixed-point is performed for you, to allow the Motorola simulator to manipulate your data. Be aware that this conversion may effect the accuracy of your output.

Demos

The Motorola DSP Developer’s Kit Demos can be accessed by typing

```
demos
```

at the MATLAB command line.

The demo models and slideshows demonstrate some of the Motorola DSP Developer’s Kit simple statistical and signal processing functions. For each Motorola DDK Blockset demo, select **Start** from the **Simulation** menu to run the simulation. For each Motorola DDK Toolbox slideshow demo, click the **Start** button to start the slideshow. Ensure that all related MEX-files and S-functions are built before attempting to run any of the demos.

Building Motorola DSP MEX-Files

This section provides detailed instructions for building Motorola DSP MEX-files. The two approaches available are:

1 Automated Build

A MATLAB M-script or pre-built options files are provided, which help you automatically build your MEX-files.

2 Custom Build

This approach gives more control over the build process to developers more knowledgeable about their compiler and system configuration. You must create your own project workspace via an Integrated Development Environment (IDE) or use your own makefiles.

For background information on building MEX-files, read "Getting Started" (Chapter 2) and "System Setup" (Chapter 8) of the *MATLAB Application Program Interface Guide*.

Automated Build Process

Configuring the MEX Build Setup (Windows Only)

This configuration needs to be performed for each C++ compiler you want to use on the Windows platform. Once completed, you are ready to build DSP MEX-files using the MATLAB `mex` build script.

Before you create a MEX-file, configure the default options file, `mexopts.bat`, for your compiler. This is done by entering

```
mex -setup
```

at the MATLAB prompt.

This `mexopts.bat` file needs further changes to build Motorola DSP MEX-files. These changes are made by running the `motdsp_build_mexopts.m` script, which generates new options files that provide the capability to build the DSP MEX-file correctly for a particular Motorola Suite56 DSP family. To run this script, type

```
motdsp_build_mexopts
```

at the MATLAB prompt.

The script creates two options files in `<matlab>/toolbox/motdsp/motdspmex`: one `OptionsFile` for the 56300 DSP family and the other for the 56600 DSP family. The `OptionsFiles` contain compiler directives and a list of arguments (e.g., include directories) used by your compiler to build the DSP MEX-files. The `OptionsFile` is passed to the `mex` command with the `-f` mex option.

Building the MEX-File

In the same directory as the `OptionsFile`, enter

```
mex -v -f <OptionsFile> <MEX-file>
```

at the MATLAB prompt, where `<MEX-file>` is the full or relative pathname of the MEX-file source you want to compile, and `<OptionsFile>` is replaced by the DSP family-specific file name. An example, while in the `<matlab>/toolbox/motdsp/motdspmex` directory, using relative pathnames might be

```
mex -v -f ./motdsp_mexopts3xx.sh
     ../motdsp/56300/mot563_mean.cpp
```

Note The full (or relative) pathnames are required as arguments to the `mex` command if any of the files do not exist in the current working directory, even if the MATLAB path is set correctly. On UNIX platforms, the full or relative path to the options file must always be specified, even if the options file is in the current directory. This latter case is shown in the preceding example.

Windows. Successful compilation results in the creation of a dynamically linked library (DLL) with the same name as the compiled MEX-file, but with a `.dll` extension. The created DLL resides in the current working directory unless you specify otherwise with the `-outdir` switch to the `mex` command. Type `help mex` at the MATLAB prompt for more information.

The error message

```
LINK fatal error: cannot open file <DLL>
```

is seen if you try to rebuild a MEX-file that has recently been run. By default, the DLL is still loaded into memory and must be explicitly released with the `clear` command by typing

```
clear mot563_mean
```

at the MATLAB prompt.

UNIX. Successful compilation results in the creation of a MEX shared library with the same name as the compiled MEX-file, but with a `.mexsol` extension for Solaris.

Custom Build Process

Differences between some compilers with respect to creating MEX-files are highlighted in Chapter 8, "System Setup", of the *MATLAB Application Program Interface Guide*.

To successfully build a DSP MEX-file, make certain that:

- `<matlab>/toolbox/motdsp/motdspmex/include/headers/56k` is added to the compilation include path
- The DSP563XX flag (or equivalent for another DSP family) is specified at the compile line
- Other compiler-specific flags are set correctly, for example, -Gx allows C++ style exceptions when set for the Microsoft Visual C++ compiler

Windows.

`<matlab>/bin/win32/libsimcore300.lib` (or equivalent for another DSP family) is *linked* with your MEX-file.

UNIX.

`<matlab>/bin/sol2/libsimcore300.so` for Solaris (or equivalent for another DSP family) is *linked* with your MEX-file.

Using Motorola DSP MEX-Files

The MATLAB MOT563_MAX Example

The `MOT563_MAX` function and the `MOT563_SMAX` S-function block provided in the Motorola DSP56300 Toolbox and Blockset are used as simple examples to illustrate some basic *NON_INTERACTIVE* mode features. These basic features include:

- Building and running a function
- Supplying optional arguments via the command line or as a parameter.

This tutorial assumes you have configured your environment as per “Building Motorola DSP MEX-Files” on page 1-6.

Building the MATLAB Function

To use the MATLAB `MOT563_MAX` function binary in the *NON_INTERACTIVE* mode, compile the source code file

`<matlab>/toolbox/motdsp/motdsp/56300/mot563_max.cpp.`

To do so, enter at the MATLAB command line

```
cd <matlab>/toolbox/motdsp/motdspmex
```

You can now compile the MATLAB MEX-file by typing the following command

```
mex -v -f ./motdsp_mexopts3xx.sh
     ../motdsp/56300/mot563_max.cpp
```

This creates a `mot563_max` library binary in the current directory.

Running the MATLAB MEX-File

To run the MATLAB MEX-file with a simple example input variable, enter

```
x = rand(10,1)
y = mot563_max(x)
```

at the MATLAB command line. This returns the contents of the output variable, *y*, as determined by the default DSP processor of the DSP56300 family, to the MATLAB workspace.

Optional Command Line Arguments

Note The default DSP processor for the 56300 DSP family is the 56301. Similarly, for the 56600 DSP family, the default processor is the 56602.

To change the DSP processor type within a selected DSP family, you must provide a string as an 'optional' input argument at the MATLAB command line. For example,

```
y = mot563_max(x,'56307')
```

If the chosen DSP device does not exist an error message such as

```
Unable to create DSP device. DevNo:0 DevPart: DSP56399
21


Error in ==> <matlab>/toolbox/motdsp/motdsp/mot563_max.dll
```

appears.

Similarly, to select a simulator command file, you must provide a string as an 'optional' input argument at the MATLAB command line. For example, type

```
y = mot563_max(x,'my_commands.cmd')
```

where `my_commands.cmd` is a valid simulator command file in the current directory or on the MATLAB path. For more information on command files, see "What You Provide" on page 2-3.

The Simulink MOT563_SMAX Example

Building the Simulink S-Function

To use the Simulink `MOT563_MAX` function binary in the *NON_INTERACTIVE* mode, compile the source code file

`<matlab>/toolbox/motdsp/motdspmex/56300/mot563_smax.cpp.`

To do so, enter at the MATLAB command line

```
cd <matlab>/toolbox/motdsp/motdspmex
```

You can now compile the Simulink MEX-file by typing the following command

```
mex -v -f ./motdsp_mexopts3xx.sh ./56300/mot563_smax.cpp
```

This creates a `mot563_smax` library binary in the current directory.

Running the Simulink S-Function

A Simulink library is provided containing the Motorola DSP Developer's Kit Blocksets with links to the supplied `motxxx_sxxxx.cpp` S-functions. To run the MOTDSP563 max block, which is based on the `mot563_smax.cpp` S-function, you need to create a simple Simulink model. For more detailed information, see "Building a Simple Model" in *Using Simulink*.

To start Simulink, click the Simulink icon in the MATLAB toolbar, or type

```
simulink
```

at the MATLAB command line.

On Windows platforms, right-click the **Motorola DSP Blockset** listing in the Simulink Library Browser to open the Motorola DSP Developer's Kit Blockset. Then double-click the **MOTDSP 56300 Blockset** icon to open the DSP56300 Blockset.

On UNIX platforms, the Simulink library window opens immediately when you launch Simulink. Double-click on the **Blocksets & Toolboxes** icon in the Simulink window. The blockset and toolbox libraries appear. Now, double-click on the **Motorola DSP Blockset** icon to open the Motorola DSP Developer's Kit Blockset, and then double-click on the **MOTDSP 56300 Blockset** icon to open the DSP56300 Blockset.

Alternatively, to open the Motorola DSP Developer's Kit Blockset directly, type

```
motdsplib
```

at the MATLAB command line.

To create a new model, select **New -> Model** from the Simulink **File** menu. Then simply drag the MOTDSP563 max block from the MOTDSP 56300 Blockset library into the new model window to begin building the system. Use blocks from the Sources and from the Sinks libraries or expanded browser lists to complete your simple model.

Setting Block Parameters

Double-click on the MOTDSP563 max block. A window appears that lets you set the block's parameters. For this simple example, use the default settings and select **Value** from the **Mode** pop-up menu. This is the simplest operating mode of the MOTDSP563 max block, and returns the maximum value of the input only.

Close the window by clicking the **OK** button or by pressing **Enter** on the keyboard.

Select **Start** from the **Simulation** menu to start the simulation.

Creating Motorola DSP MEX-Files

Overview of DSP MEX-File Development

Creation Steps

The steps for creating MEX-files for MATLAB and Simulink are the same:

- Create objects for all possible inputs and outputs to your function
- Instantiate an object representing the DSP simulator by using the macro `INSTANCE_SIMS(...)` or the C++ style `sim = new MOTDSP_...(...)`
- Import input data from the MATLAB/Simulink environment and write to DSP registers and/or memory within the simulator
- Run the DSP simulator by using the macro `SIM_RUN` or `sim->Run()`
- Write DSP data from the simulation to the function outputs and export to the MATLAB/Simulink environment
- Terminate the DSP simulator by using the macro `SIM_TERMINATE` or `sim->Terminate()`

The majority of the code added to the supplied templates (see *MEX-File Template Source*) to perform these functions is identical. There are differences between the two simulation environments which are covered in the rest of this chapter.

Note All data must be of normalized double-precision floating-point type (i.e., the data must lie between -1.0 and 1.0) before passing it to the DSP simulator via the input arguments of a MEX-file. Conversion to and from fixed-point is performed for you, to allow the Motorola simulator to manipulate your data. Be aware that this conversion may effect the accuracy of your output.

What the DSP Developer's Kit Provides

Suite56 DSP Simulator Libraries

These provide the simulator functionality and are linked into the MEX-file you create.

MEX-File Template Source

Separate C++ source templates are provided to create MATLAB MEX-files and Simulink S-functions. The files are:

- `<matlab>/toolbox/motdsp/motdspmex/templates/motdsp_template.cpp` (MATLAB)
- `<matlab>/toolbox/motdsp/motdspmex/templates/motdsp_stemplate.cpp` (Simulink)

The templates are actually working examples of DSP MEX functions based on the standard MATLAB *MAX* function.

Note The current templates use macros defined in `<matlab>/toolbox/motdsp/motdspmex/include/headers/56k/motdsp_api.h` in place of the C++ style function calls. For a list of these macros, see “Macros” on page 3-7.

Access Routines

A set of C++ methods and macros, built on top of the standard MATLAB API, is used to control and exchange data with the DSP Simulator from the MEX-file.

What You Provide

DSP56K Assembly or C-code

An associated assembly or C-code binary file you create is loaded into the simulator when the MEX-file is executed. It is possible to select from more than one binary at run-time, based on, for example, function input data types.

The restrictions on the DSP assembly are:

- The assembly binary (object file) must exist on the MATLAB path
- The assembly code must contain `BEGIN` and `END` labels
- All other labels in the assembly code must be 7 characters or less in length

The simulator starts execution of DSP instructions at `BEGIN` and terminates when it reaches the `END` address.

The name of any global variable in a C-code source file is translated to a symbol/label. This allows the MEX file to run object code independently, whether compiled from an assembly or a C-language source file.

For example,

```
        int IN;
```

may be equivalent to the assembly code

```
        org   x:$0
        IN
```

MEX-File

Use the supplied template as the basis for writing your own MEX-files. Simply add details about your particular function, as explained in the remainder of this chapter.

Use all of the standard features provided by the MATLAB API for C MEX-files, including calling any `mx` and `mex` routines where appropriate.

Locations in the supplied template files that must be modified for individual functions lie between

```
  *        *********** START USER CODE SECTION ***********
```

and

```
  *        ***********  END USER CODE SECTION ***********
```

comment blocks. Code outside these blocks should remain unchanged.

Simulator Command File

You can optionally provide a DSP Simulator command file to control the execution of the simulator itself, just as you would if running the simulator outside the MATLAB environment. Ensure that each simulator command line is followed by a carriage return.

If a command file contains any `load` and `device` commands, these take precedence over the assembly or C binary and device type supplied within the MEX-file when the simulator object is created (see “Instantiating the Simulink DSP Simulator” on page 2-20).

MATLAB MEX-Files

Before continuing, we recommend you read at least the “Creating C Language MEX Files” section under “External Interfaces” in the MATLAB online Help.

Required Definitions

Define the following macros in your MEX-file:

- `MEX_FUNC_NAME`

  The name *should* match the name of the MEX-file source (including case sensitivity), omitting the `.cpp` extension. The functions supplied in the Motorola DSP Toolbox (`<matlab>/toolbox/motdsp/motdsp/56300` and `<matlab>/toolbox/motdsp/56600`) are all prefixed with `mot563_` or `mot566_`.
- `MAX_INPUT_ARGS`
- `MIN_INPUT_ARGS`
- `MAX_OUTPUT_ARGS`
- `MIN_OUTPUT_ARGS`

  The four MAX*/MIN* macros are used to validate the call of the function from the MATLAB environment. The two optional DSP device type and simulator command file arguments *must* be included in the value of `MAX_INPUT_ARGS`.

An example using all of the macros mentioned might be

```
// Name of DSP MEX function
#define MEX_FUNC_NAME   "mot563_max"

/* Specify the expected number of function call arguments.
   Maximum number of possible input args includes fixed and
   optional args.
   Note: There are always 2 optional input arguments by default:
      1. Loading a particular dsp type (in the 56300/600 family)
         from the Matlab command line.
      2. Running command files ('xxx.cmd') from the Matlab
         command line. */
```

```
/* Maximum number of input args suggests there are 2 possible
fixed input arguments */
#define MAX_INPUT_ARGS          4
/* Minimum number of input args, i.e. the minimum number of
   fixed input arguments. */
#define MIN_INPUT_ARGS          1
/* Maximum number of output args, including fixed and optional */
#define MAX_OUTPUT_ARGS         2
/* Minimum number of output args, i.e. the number of
   fixed arguments only - generally always 1.*/
#define MIN_OUTPUT_ARGS         1
```

All other macros defined in the templates are implementation examples only and are not required to be present.

Declaring Input and Output Objects

Note You *must* enclose code that generates exceptions within a `try` block. Each `try` block is followed by one or more `catch` blocks, which specify the type of exception that can be caught and handled. This is the standard C++ exception handling mechanism.

Pointers to `MOTDSP_Input` or `MOTDSP_Output` objects are created outside the `try` block of the MEX-file for all possible input and output arguments.

For example,

```
MOTDSP_Input* a             = PTR_NIL;
MOTDSP_Input* b             = PTR_NIL;
MOTDSP_Output* y            = PTR_NIL;
MOTDSP_Output* index        = PTR_NIL;
```

Validation. Outside the `try` block, the `MOTDSP_Config` class parses the argument list and provides methods to gain access to the actual arguments. Use the supplied `MOTDSP_CONFIG` macro. This generates the following variables for use throughout the MEX-file.

```
char* MOTDSP_cmdfile  = config.GetCommandFile();
char* MOTDSP_partname = config.GetProcessorType();
int MOTDSP_data_inputs = config.GetFixedInputs();
int MOTDSP_dsptype = config.GetDspFamilyType();
```

Inside the try block, the MOTDSP_FuncVar class validates the function arguments. Use the supplied MOTDSP_FUNCVAR(nrhs,nlhs) macro.

Instantiating Input and Output Objects

Mapping Arguments to Objects.

To instantiate input and output objects you should use the macros defined as CREATE_INPUT_ARG and CREATE_OUTPUT_ARG in the <matlab>/toolbox/motdsp/motdspmex/include/headers/56k/motdsp_api.h file. Also, CREATE_NORM_INPUT_ARG is available if you are sure that the input will always be normalized. See Chapter 3, "Motorola DSP MEX-File Programming Reference."

For example,

```
CREATE_INPUT_ARG(a, 0);
CREATE_INPUT_ARG(b, 1);
num_data_inputs = config.GetFixedInputs();
    if (num_data_inputs == 2)       // i.e., two Matlab inputs
    {
        / REAL conditions...
        if ((a->GetType() == REAL_DOUBLE) &&
                    (b->GetType() == REAL_DOUBLE))
        {
            CREATE_OUTPUT_ARG(y, 0, REAL_DOUBLE);
        }
    }
```

Once the mapping of arguments to objects is complete, all access to the MEX-file arguments is through these newly created objects (i.e., the input objects a and b, and output object y in this example).

Mapping String Arguments.

For advanced MEX-files, use the CREATE_OPT_INPUT_ARG macro to create input objects from optional string input variables. For example, a filter function may require string inputs as optional arguments to indicate a high-pass or low-pass filter type. Implement this functionality by using code similar to

```
MOTDSP_Input* c = PTR_NIL;
```

```
...
try {
    // Validation Code goes here - see template
    ...
    char *opts[] = {'high', 'low'};
    int num_opts = sizeof(opts)/sizeof(opts[0]);
    // Instantiate the optional string arguments
    CREATE_OPT_INPUT_ARG(c,2,opts,num_opts);
    // The 2 indicates that the optional string input
    // is mapped from position 2 -- i.e. it is the third argument
    // expected on the command line.
    ...
    // To use the optional string inputs.
    // They are treated as fixed inputs...
    if (config.GetFixedInputs()==3) {
         // This suggest there are three input arguments -- NOT
         // including the possible OPTIONAL arguments; DSP type
         // or command file
         // strings that may be included on the command line.
         if (*(c->GetData()) == 0) {
         // This suggests an input of 'high'
         // Some useful code goes here ...
        }
        if (*(c->GetData()) == 1) {
         // This suggests an input of 'low'
         // Some more useful code goes here ...
        }
        else {
         // Error in string input arg
         // Throw an exception here ...
         sprintf(err_msg, "Invalid string input argument\n");
         THROW_MEX_ERROR(err_msg);
        }
    }
}
```

Instantiating the Motorola DSP Simulator

A simulator library and executable exist for each supported Motorola DSP family.

Note The MATLAB and Simulink MEX files supplied with the Motorola DSP Developer's Kit use a set of macros to implement the C++ style interface with the underlying Motorola DSP simulator. You are advised to use the macros provided (particularly if you intend to use INTERACTIVE mode). Alternatively, if you are familiar with C++, follow the C++ style instructions, also detailed in this chapter, on interfacing to the DSP simulator (i.e. the **`sim->`** object references).

The simulator is run in the background by instantiating a `MOTDSP_IssCustom` class object or run interactively with the full simulator graphical user interface (GUI) by using the `MOTDSP_IssStandAlone` class. First, you must use the `CREATE_SIMS` macro outside of the `try` block to create the required simulator class. To switch between the two modes rebuild your MEX-file by using compiler directives. For example, the templates and supplied functions use `STANDALONE`.

Note When running the simulator in INTERACTIVE mode, it is necessary to either run the loaded simulation or step through the code in order to obtain a non-zero output for that time step. Failure to perform this step will result in a discontinuity spike in the output data.

The constructors for each class are identical and take four string arguments:

- The name of the MEX function (not required when using the `INSTANCE_SIMS` macro)

  The `MEX_FUNC_NAME` or `S_MEX_FUNC_NAME` user-defined macros would normally be used here, since they must be defined in the MEX-file.
- The name of the assembly or C-code binary, for example, `"max-1r.cld"`

  This argument can simply be a string literal, but the MEX-file template examples provided also show the use of macros to help readability.

- The DSP device type, for example, "56309"

  This argument represents a particular Motorola DSP device. If not specified (via an empty string, ""), the simulator defaults to the DSP56301 device or the DSP56602 device depending on the DSP family chosen.
- A simulator command file, for example, "mot_mult.cmd"

  The name of the command file must be in lowercase and have a .cmd extension. If not specified for the MOTDSP_IssCustom object, the commands

  ```
  break #1 END
  go #1
  ```

  are inserted automatically by the Motorola DSP Developer's Kit and executed during simulation startup.

For example, use the INSTANCE_SIMS macro

```
INSTANCE_SIMS(      // name of assembly (or C) object file
                    "function.cld",
                    // DSP part name (eg, "56309")
                    "56309",
                    // simulator startup file (not compulsory)
                    "my_setup.cmd");
```

or the C++ style instantiation

```
sim = new MOTDSP_IssCustom( /* MEX function name (the only
                                   additional parameter)     */
                           "MEX_FUNC_NAME",
                           "function.cld",
                           "56309",
                           "my_setup.cmd");
```

When the simulator is instantiated, it loads the assembly or C-code binary and reads its symbol information.

Running the Simulation

Within the unchanged template sections, the `Run()` method of the simulator classes is called to start execution of the assembly or C-code application program. The macro alternative is `SIM_RUN`, which also contains the Simulator command '`change fast_mode 2`' for improved simulation speed. Another alternative is to use `SIM_RUN_CYCLE_ACC` to run the simulator with cycle accuracy. This is important for detailed information about the performance of the code on the simulator, however simulation speed is likely to increase.

Importing Data to DSP Simulator

Pass data into and out of the DSP simulator using appropriate methods of the simulator classes or the alternative macros provided.

Writing to DSP Registers

Available macros:

```
WRITE_REG( const char *regname, double regval )
WRITE_REG( const char *regname, ulong regval )
```

Alternative C++ calls:

```
WriteToDspReg( const char *regname, double regval )
WriteToDspReg( const char *regname, ulong regval )
```

If the argument is of type `unsigned long`, data is written as an integer. If the argument is of type `double`, data is written using fractional representation. An error will be asserted if the register name is not valid.

The behavior of the system is undefined if a fractional value is stored in a register intended to hold only integer values, for example, the Program Counter (PC).

```
// write an integer value into the Program Counter
WRITE_REG( "PC", (ulong)200 );
// store the value in Register R1 as a fraction
WRITE_REG("R1", 0.125 );
```

Writing to DSP Memory

Available macros:

```
WRITE_MEM_SYM( const char *label, double*/ulong* data, int
blocksize)
WRITE_MEM_MAP( const char *regname, double*/ulong* data, int
blocksize)
```

Alternative C++ calls:

```
WriteToDspMem( const char *label, double *mydata, int size )
WriteToDspMem( const char *label, ulong  *mydata, int size )
WriteToDspMem(enum mem_map, ulong addr, double *mydata, int size)
WriteToDspMem(enum mem_map, ulong addr, ulong  *mydata, int size)
```

The memory address for a write is specified using a symbolic label from the associated assembly or C-code file or by specifying the memory map (X-,Y- or P-memory) and address offset. The latter is used when the write memory locations vary with the size of the MEX-file's input arguments (i.e., dynamically allocated in the DSP memory).

Data is written to the DSP memory in either an integer or fractional format, depending on the type of the array passed to the method.

```
// Write input arg 'a' to address 'IN1'
WRITE_MEM_SYM( "IN1", a->GetData(), a->GetSize() );
// Write input arg 'a' to P Memory
WRITE_MEM_MAP( P_MEM, 0xF000, a->GetData(), a->GetSize());
```

Exporting Data to MATLAB

Reading from DSP Registers

Available macro:

```
READ_REG( const char *regname, enum datatype)
```

Alternative C++ call:

```
ReadFromDspReg( const char *regname, enum datatype )
```

The valid data types `M_INTTYPE` and `M_FRACTYPE`, which are defined in `motdsp_api.h`, are specified so that the value returned from the register is interpreted correctly.

```
int* addr = READ_REG( "PC", M_INTTYPE );
```

```
double* fracval = READ_REG( "R1", M_FRACTYPE );
```

Reading from DSP Memory

Available macros:

```
READ_MEM_SYM( const char *label, int size, enum datatype )
READ_MEM_MAP( enum mem_map, ulong addr, int size, enum datatype)
READ_MEM_MAP_INT( enum mem_map, ulong addr, int size )
READ_MEM_MAP_FRAC( enum mem_map, ulong addr, int size )
```

Alternative C++ calls:

```
double *ReadFromDspMem( const char *label, int size, enum datatype
)
double *ReadFromDspMem( enum mem_map, ulong addr,
                      int size, enum datatype )
```

These methods read a contiguous block of memory from the DSP. The start location is specified using a symbolic label from the associated assembly or C-code or by specifying the memory map and address offset. The correct data type is also indicated so that the values can be interpreted correctly, either by calling the correct macro (READ_MEM_MAP_INT or READ_MEM_MAP_FRAC) or by specifying the correct enumerated datatype (M_INTTYPE or M_FRACTYPE).

Terminating and Allocated Memory Cleanup

Within the unchanged template sections, the Terminate() method of the simulator classes is called to end the simulation. The macro alternative is SIM_TERMINATE.

Use the MEM_DELETE() macro as appropriate to free memory allocated during execution of the MEX-file. Similarly, use the DELETE_SIMS macro to free the simulator class object.

You should use these macros both at the very end of the try block and in all of the exception handling catch blocks of your functions.

For example,

```
...
        // Finally - Terminate the simulator and clean up.
        SIM_TERMINATE;
```

```
          DELETE_SIMS;
          MEM_DELETE(a);
          MEM_DELETE(b);

    } // End try block

    // Start catch block
    catch(MOTDSP_Exception &exc)
    {
         DELETE_SIMS;
         MEM_DELETE(a);
         MEM_DELETE(b);
    } /* End catch block */
```

Simulink S-Function MEX-Files

Before continuing, we recommend you read “Getting Started” (Chapter 1), “Creating a Model” (Chapter 3) and the “Writing S-Functions as C MEX-Files” section of “S-Functions” (Chapter 8), in *Using Simulink*. For more detailed information, refer to *Writing S-Functions*.

Note The supplied Simulink files use macros defined in `<matlab>/toolbox/motdsp/motdspmex/include/headers/56k/motdsp_api.h` in place of the C++ style function calls. The purpose of these macros is to use the correct simulator object for each iteration (sample) of the simulation when in INTERACTIVE mode. If a sample hit occurs, as specified from the dialog box parameter, the simulator object will launch the simulator GUI. At other times, it will run in NON_INTERACTIVE mode. For a list of these macros see “Macros” on page 3-7.

Required Definitions

Define *only* the following macros in your MEX-file:

- `S_MEX_FUNC_NAME`
- `S_FUNCTION_NAME`

  For the Simulink environment, these names *must* exactly match the name of the MEX-file source (including case sensitivity), omitting the `.cpp` extension. The functions supplied in the Motorola DSP Blockset (`<matlab>/toolbox/motdsp/motdspmex/56300` and `<matlab>/toolbox/motdsp/motdspmex/56600`) are all prefixed with `mot563_` or `mot566_`. Also, the `S_MEX_FUNC_NAME` macro is always defined as a string.
- `S_FUNCTION_LEVEL`

  These macros are required for any MEX S-function. The `S_FUNCTION_LEVEL` macro is always defined to equal 2.

For example,

```
//fn name as a string.
#define S_MEX_FUNC_NAME  "motdsp_stemplate"
//fn name for compile purposes.
#define S_FUNCTION_NAME  motdsp_stemplate
```

```
#define S_FUNCTION_LEVEL 2
```

The MAX*/MIN* macros described for MATLAB MEX-files are specifically not required for Simulink S-functions.

Also, the template provides an example of optional dialog box parameter checking within the `mdlCheckParameters` function.

```
/* Get the first dialog box parameter and
   check to see if it is a string.*/
if (!mxIsChar(ssGetSFcnParam(S,0)) )
    {
    ssSetErrorStatus(S,"1st parameter to S-function must be a "
                    "string which represents the filename "
                    "of the command file.");
    return;
}
```

Input Objects

Pointers to `MOTDSP_Input` objects *only* are declared outside the `try` block of the `mdlOutputs` function.

For example,

```
MOTDSP_Input* a = PTR_NIL;
MOTDSP_Input* b = PTR_NIL;
```

Before creating `MOTDSP_Input` objects declare `InputPtrsType` pointers to all possible input ports using the specific Simulink 'ss' macros.

For example, for input Port 1 use

```
InputPtrsType uPtrs1 = ssGetInputPortSignalPtrs(S,0);
```

and, for input Port 2

```
InputPtrsType uPtrs2 = ssGetInputPortSignalPtrs(S,1);
```

Note Reference the *i*th element of the input port signal array with, for example, `*uPtrs1[i]`.

Also, you *must* declare a pointer to an mxArray for each possible input port.

```
mxArray *array_ptr1;
mxArray *array_ptr2;
```

Then use the support functions MOTDSP_CreateComplexMatrix or MOTDSP_CreateRealMatrix to create a valid matrix after checking its type:

```
//Determine whether the port is complex.
const boolean_T c0 =
   (boolean_T)(ssGetInputPortComplexSignal(S,0) == COMPLEX_YES);
//Then create the correct matrix:
if (!c0) {
    /* If the matrix is Real...
       Use the MOTDSP_ function to create
       the appropriate matrix..*/
    MOTDSP_CreateRealMatrix(&array_ptr1,
                        (double *)*uPtrs1,
                        width);
    }
```

Remember to clear the memory created for each mxArray by calling the mxDestroyArray function.

Finally, create a pointer to right hand side (prhs) MEX equivalent variable before mapping the input ports to variables. For example,

```
    const mxArray *prhs [] = {array_ptr1, array_ptr2};
    CREATE_INPUT_ARG(a,0);
    CREATE_INPUT_ARG(b,1);
```

Information about the variables is accessed by the supplied methods as described previously in “MATLAB MEX-Files” on page 2-5.

Output Objects

MOTDSP_Output objects are *not* used in S-functions. Instead, you create pointers to the output ports using specific Simulink macros.

```
    // Create real versions of output port pointers
    // to output port 1
    // Real and Complex ...
   creal_T *y1 = (creal_T *)ssGetOutputPortSignal(S,0);
```

```
    // Real only ...
   real_T  *yr = ssGetOutputPortRealSignal(S,0);
```

You then “fill” these output signals with data returned from execution of the assembly or C program object file.

For a real array

```
    for (int k = 0; k < ssGetInputPortWidth(S,0); k++)
    {
       yr[k] = READ_MEM_SYM("OUT",a->GetSize(),M_FRACTYPE)[k];
    }
```

For real and imaginary parts of a single value complex output

```
//real part
y1[0].re = READ_REG("X1", M_FRACTYPE);
//imaginary part
y1[0].im = READ_REG("Y1", M_FRACTYPE);
```

For real and imaginary parts of a complex array output

```
for (int k = 0; k < ssGetInputPortWidth(S,0); k++)
{
    y1[k].re = READ_MEM_MAP_FRAC(X_MEM, StartAddress1,
                                a->GetSizeReal())[k];
    y1[k].im = READ_MEM_MAP_FRAC(Y_MEM, StartAddress1,
                                a->GetSizeImag())[k];
}
```

See “Importing Data to DSP Simulator” on page 2-11 for more information on how to access the data returned from the simulator.

Validation: Inside the `try` block, a `MOTDSP_FuncVar` object is instantiated with the number of input ports and output ports as arguments. Use the supplied `MOTDSP_FUNCVAR(NUM_INPORTS,NUM_OUTPORTS)` macro. This class is used to validate access to input/output and create default or initial conditions.

Compared with MATLAB MEX-files, no `MOTDSP_Config` class is required for S-functions, since an argument list is not parsed. Instead, arguments are obtained from a parameter dialog box.

S-Function Blocks

Perform these three steps to link your S-function MEX-file to a Simulink block:

1 Choose the S-Function User Definable block from the Simulink Library Browser.

2 Double-click on the S-Function block to open the dialog box.

- Enter the function name, for example `motdsp_stemplate`.
- Enter the names of the parameters to pass into the block. For example, `cmdFile, objfile, dsptype, NumInputs, NumOutputs, timestep`

The parameter names that you specify here are the variable names that are mapped to the dialog box prompts in the next step. These parameters are treated by Simulink as MATLAB `mxArrays`.

An example of accessing these parameters from your S-function MEX-file is provided in the template.

```
// Command File
#define CMD_FILE ssGetSFcnParam(S,0)
// Object File - not required in this file.
#define OBJ_FILE ssGetSFcnParam(S,1)
// DSP Type
#define DSP_TYPE ssGetSFcnParam(S,2)
```

The above three string parameters (most likely to be used in the Motorola S-function MEX-files) can be accessed by using the following support functions.

```
int dsptype     = MOTDSP_GetSFcnDspFamilyType(DSP_TYPE);
char * cmdfile  = MOTDSP_GetSFcnCmdFile(CMD_FILE);
char * partname = MOTDSP_GetSFcnDSPName(DSP_TYPE);
```

Alternatively, the integer parameters may be accessed by using `mx` functions.

```
// Number of input ports parameter specified
#define NUM_INPORTS  (int)mxGetPr(ssGetSFcnParam(S,3))[0]
// Number of output ports parameter specified
#define NUM_OUTPORTS (int)mxGetPr(ssGetSFcnParam(S,4))[0]
/* Cause a 'Sample Hit' to occur at this time during the
```

```
   simulation - to be used as shown in the template.*/
#define DEBUG_AT_TIME_STEP (int)mxGetPr(ssGetSFcnParam(S,5))[0]
```

3 Select **Edit->Mask S-Function**. This will open the Mask Editor. Masking your S-Function block is a useful way to provide dynamic parameters to your S-function MEX-file code via dialog box prompts. For more detailed information read Chapter 6, “Using Masks to Customize Blocks”, of *Using Simulink*.

Instantiating the Simulink DSP Simulator

See “Instantiating the Motorola DSP Simulator” on page 2-9 for more information on how to instantiate the DSP simulator.

Running Your Simulation

See “Running the Simulation” on page 2-11 for more information on how to start execution of the simulator program.

Importing Data to DSP Simulator

See “Importing Data to DSP Simulator” on page 2-11 for more information on writing data to the DSP simulator.

Exporting Data to Simulink

See “Exporting Data to MATLAB” on page 2-12 for more information on how to read data out of the DSP simulator.

Terminating and Allocating Memory Cleanup

See “Terminating and Allocated Memory Cleanup” on page 2-13 for more information on how to end the simulation and how to free memory allocated during the execution of the MEX-file.

Tutorial of Advanced Features

The MOT563_MEAN and MOT563_SMEAN functions in the Motorola DSP Developer's Kit Toolbox / Blockset respectively are used as examples to illustrate certain features of the Motorola DSP Developer's Kit in *INTERACTIVE* mode. This tutorial assumes you have configured your environment as per "Building Motorola DSP MEX-Files" on page 1-6.

Building and Running MOT563_MEAN

To use the MOT563_MEAN function binary in the *INTERACTIVE* mode with the GUI of the Suite56 DSP Simulator, recompile with the STANDALONE flag. To do so, enter at the MATLAB command prompt:

Recompile the MEX-function.

```
mex -v -DSTANDALONE -f motdsp_mexopts3xx.sh
    ../motdsp/56300/mot563_mean.cpp
```

This creates mot563_mean library binary in the current directory. To run the MEX-file with a simple input variable example, enter

```
x = rand(10,1)
y = mot563_mean(x)
```

at the MATLAB command prompt. The GUI splashscreen of the Suite56 DSP Simulator appears. At the same time, the existing MATLAB command window freezes and a new, temporary MATLAB workspace or engine session output window appears. Use this second MATLAB workspace and the Suite56 DSP Simulator to analyze DSP data.

The MEX-file execution is paused, ready to start execution of the loaded DSP assembly or C program. From this point, enter any valid Suite56 DSP Simulator command at the Simulator command line. For example, to simply run the loaded assembly or C-code, enter

```
go
```

or use the GUI buttons in the toolbar. For more information on using the Suite56 DSP simulator, read the *Motorola DSP Simulator Reference Manual*. Exiting the Simulator destroys the temporary MATLAB workspace or engine session and returns you to the main MATLAB workspace.

Building and Running MOT563_SMEAN

Use of the `MOT563_SMEAN` function binary in the *INTERACTIVE* mode with the GUI of the Suite56 DSP Simulator is essentially the same as for the `MOT563_MEAN function` as described in "Building and Running MOT563_MEAN" on page 2-21. Only the running of the S-Function MEX-file is performed differently to running the MATLAB `MOT563_MEAN` function.

Create a new Simulink model, incorporating the `MOT563_SMEAN` block. Edit the block's "*Simulation Time to enter Interactive Mode*" parameter to 5 and run the simulator. For more information on creating and running a new model, see "The Simulink MOT563_SMAX Example" on page 1-10

callMatlab

MATLAB engine library routines are incorporated into the supplied Suite56 DSP GUI Simulator to enable MATLAB functions and commands to be called from the simulator command window.

The syntax for the two ways of calling `callMatlab` is:

- `callMatlab put <mem_space>:<start_address>:<end_address>`

  When "put" is present, the DSP memory contents as specified by the last argument are copied into a variable in the current MATLAB engine session. The variable name is constructed from the colon-separated elements of the last argument.
- `callMatlab matlab_command`

  When called this way, the `matlab_command` expression is passed to the MATLAB engine and evaluated there, as if entered at the MATLAB prompt. The onus is on you to ensure that a valid MATLAB command is entered.

For example, issue the following two `callMatlab` commands in the GUI simulator command window.

```
callMatlab put x:0:10
callMatlab plot(memx_0_10)
```

There is now a variable named `memx_0_10` in the workspace of the MATLAB engine session created by the MEX function. The variable contains the DSP X memory block of 11 bytes from $0000 through to $000a inclusive.

A plot of the DSP memory block also appears after executing the MATLAB `plot` command above.

Note The `callMatlab` simulator command on UNIX is the only interface with the MATLAB engine session that runs in the background. A standard terminal appears that displays the redirected output of any MATLAB commands issued via the `callMatlab` simulator command. For example, you may need to use `callMatlab whos` to see a list of variables available in the MATLAB engine session workspace.

Data Snapshots

Snapshots provide access to DSP memory and/or DSP register data from within the MATLAB engine session. This allows debugging of the algorithm within the MEX function when running in the INTERACTIVE mode. *Access* means you are able to monitor and process a copy of any DSP memory data and DSP registers in the simulator without altering the contents.

Two types of snapshots are available: *Instant* and *Continuous* snapshots.

Instant Snapshot

The instant snapshot represents the data at the current point of simulator execution, i.e., at the time the snapshot command is issued. You must issue the instant snapshot command from the GUI simulator command window. Some examples include

```
snapshot reg r4
snapshot mem x:$0:$10
```

After the execution of these commands, observe the workspace of the MATLAB engine session that was created by the MEX function. There should be two structure variables, one is the snapshot of the DSP register R4, and the other is the snapshot of the DSP X memory block from $0000 to $0010. For accessing structure variables, refer to the “Programming with MATLAB” section in the MATLAB documentation. Each structure contains attribute information and the actual value as seen by the simulator.

Continuous Snapshot

Data from a continuous snapshot is updated after each DSP instruction is executed by the simulator. You must declare continuous snapshots in the MEX-file at compile time. The following source lines added to the MEX-file

<matlab>toolbox/motdsp/motdsp/56300/mot563_mean.cpp set up continuous snapshots.

```
#ifdef STANDALONE
  // Continuously monitor register 'X1'.
  ADD_SNAPSHOT_REG("Snapshot_X1", "X1");
  // Continuously monitor DSP X memory block at location $0000
  ADD_SNAPSHOT_MEM("Snapshot_MEM_X", memory_map_x,
                 0x00, (ulong)CUR_DIM_SIZE);
#endif
```

or using C++

```
#ifdef STANDALONE
  // Continuously monitor register 'X1'.
  sim->AddSnapShot("Snapshot_X1", "X1");
  // Continuously monitor DSP X memory block at location $0000
  sim->AddSnapShot("Snapshot_MEM_X", memory_map_x,
                   0x00, (ulong)CUR_DIM_SIZE);
#endif
```

Continuous snapshots for the DSP register X1, and DSP X memory block from address $0 to address CUR_DIM_SIZE (see the MEX-file for the definition of this block size macro) are created when you run MOT563_MEAN. Step through the DSP assembly or C-code and observe the contents of the snapshot variables updating in the MATLAB engine session.

System Analysis Return Data (SARD)

This feature returns information about the execution of the DSP assembly program within the simulator to the MATLAB environment. Both the *INTERACTIVE* and *NON_INTERACTIVE* modes support this feature. The data contains:

- The number of DSP instructions executed by the program
- The number of DSP clock cycles taken to execute the program
- The size of the program (P) memory space used by the program
- The size of the X memory space for the program

- The size of the Y memory space for the program

  (These P, X, and Y memory size values show only static information available at assembly time.)
- The contents of the Status Register when the DSP program exits

The creation and return of SARD data is optional. To enable this feature, set a MATLAB variable named <FUNCTION_NAME>_SARDflag. When you run the corresponding MEX function, a MATLAB struct array with the name <FUNCTION_NAME>_SARD appears in the workspace. Be aware that the variable <FUNCTION_NAME> is case-sensitive.

The following example shows the use of SARD with the MOT563_MAX function.

```
» x = rand(10,1)
» mot563_max_SARDflag = 1
» mot563_max(x)
ans =
        0.9318

» whos
  Name                      Size         Bytes  Class

mot563_max_SARD           1x1            792  struct array
mot563_max_SARDflag       1x1              8  double array
  ans                       1x1              8  double array
  x                        10x1             80  double array

Grand total is 13 elements using 888 bytes

» mot563_max_SARD

mot563_max_SARD =
      Instruction Count: 31
     ClockCycle Count: 49
         Program Size: 14
            XMem Size: 0
            YMem Size: 0
      Status Register: C00390
```

The SARD is updated each time a function of the same name executes.

3

Motorola DSP MEX-File Programming Reference

Public Methods

This is a list of publicly available C++ method prototypes that you may include in your DSP MEX-files.

class MOTDSP_Exception.

```
      MOTDSP_Exception(const char* error_message,
                     const int error_number = EEX_UNKNOWN);
```

class MOTDSP_FuncVar.

```
      MOTDSP_FuncVar(int nrhs, int nlhs);
      int GetSize(void);
      int GetSizeReal(void);
      int GetSizeImag(void);
      int GetNumRows(void);
      int GetNumCols(void);
      enum mat_type GetType(void);
      bool IsComplex(void);
      bool Isnormalized(double value);
      void SetSize(int rows, int columns);
```

class MOTDSP_Input : public MOTDSP_FuncVar.

```
      MOTDSP_Input(int arg_index, const mxArray* arg_pointer[],
                   enum inp_type);
      MOTDSP_Input(int arg_index, const mxArray* arg_pointer[],
                   const char* arg_options[], int num_arg_options,
                   enum inp_type);
      int GetIndex(void);
      double* GetData(void);
      double* GetDataReal(void);
      double* GetDataImag(void);
      double* GetDataComplex(void);
      double* GetData(int row, int col);
      double* GetDataReal(int row, int col);
      double* GetDataImag(int row, int col);
```

class MOTDSP_Output: public MOTDSP_FuncVar.

```
MOTDSP_Output(int arg_index,
            mxArray* arg_pointer[],
            enum mat_type m_type);
void SetSize(mxArray* arg_pointer[],
           int rows,
           int cols);
int GetIndex(void);
void PutData(double *data);
void PutDataReal(double *data);
void PutDataImag(double *data);
void PutDataComplex(double *data);
void PutData(double value);
void PutDataReal(double value);
void PutDataImag(double value);
void PutDataComplex(double value);
```

class MOTDSP_Simulator.

```
char* GetMexFuncName(void);
char* GetObjectFileName(void);
char* GetDeviceType(void);
char* GetCommandFile(void);
int GetDeviceNumber(void);
```

class MOTDSP_IssCustom : public MOTDSP_Simulator .

```
MOTDSP_IssCustom(const char* MEX_function_name,
                 const char* object_code_file,
                 const char* processor_type,
                 const char* sim_command_file);
ulong ConvertDoubleToFixed(double dval);
double ConvertFixedToDouble(ulong fixval);
bool DoCommand(const char* command);
ulong GetSymbolAddress(const char *symbol_name);
enum memory_map GetSymbolMemMap(const char *symbol_name);
double* ReadFromDspMem(const char* symbol_name,
                  int block_size,
                  enum sim_radix_type radix_type );
```

```
double* ReadFromDspMem(enum memory_map map,
                       ulong mem_address,
                       int block_size,
                       enum sim_radix_type radix_type );

ulong* ReadFromDspMemInt(enum memory_map map,
                         ulong mem_address,
                         int block_size );
double* ReadFromDspMemFrac(enum memory_map map,
                           ulong mem_address,
                         int block_size );
double* ReadFromDspReg(const char* reg_name,
                       enum sim_radix_type radix_type);
void Run(void);
void Terminate(void);
void WriteToDspMem(const char* symbol_name,
                   double* data,
                   int block_size);
void WriteToDspMem(enum memory_map map,
                   ulong mem_address,
                   double* data,
                   int block_size);
void WriteToDspMem(const char* symbol_name,
                   ulong* data,
                   int block_size);

void WriteToDspMem(enum memory_map map,
                   ulong mem_address,
                   ulong* data,
                   int block_size);
void WriteToDspReg(const char* reg_name, ulong value);
void WriteToDspReg(const char* reg_name, double value);
void WriteToDspReg(const char* reg_name, ulong* value);
void WriteToDspReg(const char* reg_name, double* value);
```

class MOTDSP_IssStandAlone : public MOTDSP_Simulator.

```
MOTDSP_IssStandAlone(const char* MEX_function_name,
                     const char* object_code_file,
                     const char* processor_type,
                     const char* sim_command_file);
void AddSnapShot(const char* var_name, char* reg_name);
void AddSnapShot(const char* var_name,
                 enum memory_map map,
                 ulong address,
                 int block_size);
ulong GetSymbolAddress(const char *symbol_name);
enum memory_map GetSymbolMemMap(const char *symbol_name);
double* ReadFromDspMem(const char* symbol_name,
                       int block_size,
                       enum sim_radix_type radix_type );
double* ReadFromDspMem(enum memory_map map,
                       ulong mem_address,
                       int block_size,
                       enum sim_radix_type radix_type );
ulong* ReadFromDspMemInt(enum memory_map map,
                         ulong mem_address,
                         int block_size );
double* ReadFromDspMemFrac(enum memory_map map,
                           ulong mem_address,
                           int block_size );
double* ReadFromDspReg(const char* reg_name,
                      enum sim_radix_type radix_type);
void Run(void);
void Terminate(void);
void WriteToDspMem(const char* symbol_name,
                   double* data,
                   int block_size);
void WriteToDspMem(enum memory_map map,
                   ulong mem_address,
                   double* data,
                   int block_size);
void WriteToDspMem(const char* symbol_name,
                   ulong* data,
                   int block_size);
```

```
void WriteToDspMem(enum memory_map map,
                   ulong mem_address,
                   ulong* data,
                   int block_size);
void WriteToDspReg(const char* reg_name, ulong value);
void WriteToDspReg(const char* reg_name, double value);
void WriteToDspReg(const char* reg_name, ulong* value);
void WriteToDspReg(const char* reg_name, double* value);
```

class MOTDSP_Config.

```
MOTDSP_Config(int argc_input,
              int argc_output,
              const mxArray* prhs[]);
char* GetCommandFile(void);
char* GetProcessorType(void);
int GetDspFamilyType(void);
int GetFixedInputs( void);
```

Macros

This is a list of available macros used in DSP MEX-file development. The macros supersede the C++ methods also listed in this chapter.

DSP Memory Space Definitions

```
#define P_MEM memory_map_p
#define X_MEM memory_map_x
#define Y_MEM memory_map_y
```

General Creation/Deletion Macros

```
#define CREATE_SIMS
#define CREATE_INPUT_ARG(identifier, arg_pos)
#define CREATE_NORM_INPUT_ARG(identifier, arg_pos)
(or, for known string input types)
#define CREATE_OPT_INPUT_ARG(identifier, arg_pos,
                             options, num_options)
#define CREATE_OUTPUT_ARG(identifier, arg_pos, group_type)
#define DELETE_SIMS
#define MEM_DELETE(identifier)
```

General DSP I/O Macros

```
#define WRITE_MEM_SYM(SYMBOL, DATA, BLK_SIZE)
#define WRITE_MEM_MAP(MEM_SPACE, START_ADDRESS, DATA, BLK_SIZE)
#define WRITE_REG(REG_NAME, VALUE)

#define READ_REG(REG_NAME, RADIX_TYPE)
#define READ_WIDEREG(REG_NAME, RADIX_TYPE)
#define READ_MEM_SYM(SYMBOL, BLK_SIZE, RADIX_TYPE)
#define READ_MEM_MAP(MEM_SPACE, START_ADDRESS,
                     BLK_SIZE, RADIX_TYPE)
#define READ_MEM_MAP_INT(MEM_SPACE, START_ADDRESS, BLK_SIZE)
#define READ_MEM_MAP_FRAC(MEM_SPACE, START_ADDRESS, BLK_SIZE)
```

Miscellaneous Macros

```
#define CHECK_DSP_PARAM(DSPTYPE) - S-functions only.
#define MOTDSP_CONFIG - MEX-files only.
#define MOTDSP_FUNCVAR(NUMBER_INPUTS, NUMBER_OUTPUTS)
#define INSTANCE_SIMS(OBJCODE_FILE, DSP_PART, MACRO_FILE)
#define SIM_DO_CMD(COMMAND)
#define SIM_RUN
#define SIM_RUN_CYCLE_ACC
#define ADD_SNAPSHOT_REG(VAR_NAME, REG_NAME)
#define ADD_SNAPSHOT_MEM(VAR_NAME, MEM_SPACE, START_ADDRESS,
                                                     BLK_SIZE)
#define SIM_CONV_FIXED_TO_DOUBLE(VALUE)
#define SIM_CONV_DOUBLE_TO_FIXED(VALUE)
#define THROW_MEX_ERROR(ERROR_MSG_STRING)
#define EEX_ABORT(MOTDSP_EXCEPTION_OBJECT)
#define SIM_TERMINATE
```

Miscellaneous Definitions

```
#define TRUE    1
#define FALSE   0

#define FAILURE 1
#define SUCCESS 0

#define PTR_NIL 0
```

Alphabetical List of Assembly Files

The following reference pages describe the supplied assembly files that contain the functional algorithms for both the toolbox and the blockset. These reference pages are listed alphabetically, *except* that the assembly file for the real case always appears before the one for the complex case. For example, `abs-r.asm` appears before `abs-c.asm`.

Note Under the "MATLAB Usage" heading on each reference page, the ### characters represent either the 563 or the 566 processor family.

MATLAB Usage `Y = mot###_abs(X)`

Description This function returns the absolute value of the input real vector X

Input/Output Input: Real vector X (elements of X are real data)

Output: Real vector Y

Algorithm

```
for ( i=0; i<size(X); i++)
{    Y[i] = abs(X[i]);    }
```

Note `abs` is an assembly instruction of the DSP56K instruction set

Memory& Register Memory allocation and register usage:

- Start address of real vector X is defined in symbol IN
- Size of vector X is stored in register R7
- Start address of vector Y is defined in symbol OUT

Status Register The assembly function `abs-r.asm` does not set any status registers/bits during the function execution.

Data Size Limit The length of vector X is limited by the size of available continuous data memory.

Data Range Limit The input vector X range is [–1.0, +1.0].

Precision In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit In the case of DSP563 and DSP566, there are 4 cycles for each element of input data.

abs-c.asm

MATLAB Usage `Y = mot###_abs( X )`

Description This function returns the magnitude of the input complex vector/sqrt(2). The input vector X is a complex vector

Input/Output Input: Complex vector X (includes real part `Xr`, and imaginary part `Xi`)

Output: Real vector Y

Algorithm

```
for ( i=0; i <size( X ); i++ )
{
Intervalue = Xr[i] * Xr[i];
Intervalue += Xi[i] * Xi[i];
y[i] = sqrt-sr(Intervalue >> 1);
}
```

Note If absolute values are desired, the scaling up factor is `sqrt(2)`. For a description of function `sqrt-sr`, refer to `sqrt-sr.asm` on page 3-96.

Memory & Register Memory allocation and register usage:

- Start address of the real part of vector X is defined in label IN_REAL
- Start address of the imaginary part of vector X is defined in label IN_IMAG
- Size of vector X is stored in register R7
- Start address of output vector Y is defined in label OUT

Status Register The assembly function `abs-c.asm` does not explicitly set any status register bits during the function execution.

Data Size Limit The length of vector X is limited by the size of available continuous data memory.

Data Range Limit The input vector X range is [–1.0, +1.0].

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

In the case of DSP563, there are 222 cycles for each element of input data.

In the case of DSP566, there are 150 cycles for each element of input data.

angle-c.asm

MATLAB Usage `Y = mot###_angle( X )`

Description

This function returns the inverse tangent (arctangent) of the input complex vector X

Input/Output

Input: Complex vector X (includes the real part Xr and the imaginary part Xi)

Output: Vector Y

Algorithm

Use the CORDIC algorithm. For each input vector, x represents the real part, and y represents the imaginary part of the input vector.

```
int z = 0, X, Y, Z =0, i;
        if (x == (unsigned frac)0)
           return (frac)0;
        X = x = x>>2 ;
        Y = y = y>>2 ;
    /* Circular Function */
        for (i = 0; i <= fracbits; ++i)
       {
            x = X >> i;
            y = Y >> i;
            z = atan[i];
            if (Y <= 0)
           {
                X -= y;
                Y += x;
                Z -= z;
           }
            else
           {
                X += y;
                Y -= x;
                Z += z;
           }
       }
        Z = Z << 2;
        return Z;
```

Memory & Register

Memory allocation:

- Label IN stores the start address of input vector X
- Label INREAL stores the start address of real data of input vector
- Label INIMAG stores the start address of complex data of input vector
- Label OUTREAL stores the start address of output vector
- Variable X, Y is saved starting from label XYZ

  X0 as x

  X1 as y

  Y1 as z

  B as Z

Register usage:

- Register R6 stores the number of items of the input array
- Register R2 stores the fraction bits +1

  In the case of DSP563, R2 stores 22

  In the case of DSP566, R2 stores 14

- Register R3 stores the value of the atan table
- Registers R0 and R5 store the label IN(OUT)
- Register R7 stores the start address of label XYZ
- Register R1 stores the loop counter
- Register Y0 is mainly used as shift number register (i.e., variable 'i' in the algorithm description)

StatusRegister

The assembly function `angle-c.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit

The length of vector X is limited by the size of available continuous data memory.

Data Range Limit

The input vector X range is [–1.0, +1.0].

Precision

In the case of DSP563, precision is 17 bits.

In the case of DSP566, precision is 10 bits.

Performance Limit

In the case of DSP563:

- When the input vector is a real vector, there are 25 cycles for each element of input data.
- When the input vector is a complex vector, there are 628 cycles for each element of input data.

In the case of DSP566:

- When the input vector is a real vector, there are 25 cycles for each element of input data.
- When the input vector is a complex vector, there are 609 cycles for each element of input data.

MATLAB Usage `C = mot###_conv( A,B )`

Description This function convolves vectors A and B, where both A and B are real vectors

Input/Output Input: Real vector A, and real vector B

Output: Real vector C

Algorithm

```
LengthC = LengthB + LengthA – 1

for ( i = 1; i  <= LengthC; i ++ ) {
if ( i <= LengthA ) {
   for ( j = 1; j <= i; j ++ ) {
       C[i] += A[j] * B[i – j + 1]; }}
else {
   for ( j = i – LengthA; j <= LengthA - 1; j ++ ) {
       C[i] += A[j + 1] * B[i – j];
     }}}
```

Memory & Register

Memory allocation:

In X memory:

- X:(#INA) stores the start address of vector A
- X:(#INA+LENGTH(A)) stores the start address of vector C

In Y memory:

- Y:(#INB) stores the start address of vector B

Register usage:

- R2 stores the length of vector A
- R7 stores the length of vector C
- R0 stores the index of vector A
- R4 and R5 store the index of vector B
- R3 stores the index of vector C
- R1, R2, and R6 are used as loop control
- R2-1 -> M5, assume M{0,1,2,3,4,6,7} = $ffffff

Status Register

The assembly function `conv-r.asm` does not set any status registers/bits during the function execution

Data Size Limit

The maximum length of vector A and B can't be larger than 1/3 of the continuous available data memory size.

Data Range Limit

The input data vector range is [–1.0, +1.0].

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

DSP566:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

MATLAB Usage `C = mot###_conv( A,B )`

Description This function convolves vectors A and B, where both A and B are complex vectors

Input/Output Input: Complex vector A (Ar is the real part, and Ai is the imaginary part), and complex vector B (Br is the real part, and Bi is the imaginary part)

Output: Complex vector C (Cr is the real part, and Ci is the imaginary part)

Algorithm

```
LengthC = LengthB + LengthA – 1
for ( i = 1; i  <= LengthC; i ++ ) {
if ( i <= LengthA ) {
   for ( j = 1; j <= i; j ++ ) {
       Cr[i] += Ar[j] * Br[i – j + 1] – Ai[j] * Bi[i – j  + 1];
       Ci[i] += Ai[j] * Br[i – j + 1] + Ar[j] * Bi[i – j  + 1];
   }
}
else {
   for ( j = i – LengthA; j <= LengthA - 1; j ++ ) {
       Cr[i] += Ar[j + 1] * Br[i – j] – Ai[j + 1] * Bi[i – j];
       Ci[i] += Ai[j + 1] * Br[i – j] + Ar[j + 1] * Bi[i – j];
    }
}
}
```

Memory & Register Memory allocation:

In X memory:

- X:(INA) stores the start address of the real part of vector A
- X:(INA + LENGTH(A)) stores the start address of the imaginary part of vector A
- X:(INA + LENGTH(A)*2) stores the start address of the real part of vector C

In Y memory:

- Y:(INB) stores the start address of the real part of vector B
- Y:(INB + LENGTH(B)) stores the start address of the imaginary part of vector B

- Y:INB + LENGTH(B)*2 stores the start address of the imaginary part of vector C

Register usage:

- R0 is used for the index of the real part of vector A
- R0+N0 is used for the index of the imaginary part of vector A
- R4 and R5 are used for the index of the real part of vector B
- R4 +N4 are used for the index of the imaginary part of vector B
- R3 is used for the index of the real part of vector C
- R7 is used for the index of the imaginary part of vector C
- R1, R2, and R6 are used as loop control
- R2-1 -> M5, assume `M{0,1,2,3,4,6,7} = $ffffff`

Status Register

The assembly function `conv-c.asm` does not set any status registers/bits during function execution.

Data Size Limit

The maximum length of vector A and B can't be larger than 1/4 of the continuous available data memory size.

Data Range Limit

The input data vector range is [–1.0, +1.0].

Precision

In the case of DSP563, precision is the full 23 bits.

In the case of DSP566, precision is the full 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 24 \times \Delta inputvectorlength = 24$$

DSP566:

$$\Delta\Delta cycle = 24 \times \Delta inputvectorlength = 24$$

MATLAB Usage `Y = mot###_decimate( X, r, nfilt, 'fir' )`

Description

This function `decimate` resamples data at a lower rate after lowpass FIR filtering. Input vector X is a real vector

Input/Output

Input Parameters: Real vector X, `int r`, `int nfilt`

Output Parameters: Real vector Y

Algorithm

The MEX function `decimate_fir.m` calculates vector b, vector list, `int nout`, and `int nbeg`. The CMEX function loads these parameters into dsp memory for asm function use.

```
nd = length(idata);
m = size(idata,1);
nout = ceil(nd/r);
b = fir1(nfilt,1/r);
gd = grpdelay(b,1,8);
list = round(gd(1)+1.25):r:nd;
lod = length(list);
nlen = nout-lod;
nbeg = r-(nd-list(length(list)));

Assembly function decimate-fir-r.asm then follows these steps to
calculate decimated vector Y:
nfilt = nfilt+1
itemp = 2*idata(1)-idata((nfilt+1):-1:2)

[odata,zf] = filter(b,1,idata,zi)
odata = odata(list)
```

Memory & Register

Memory allocation:

- Input vector X is located in X memory
- Input `r` is loaded in register N3
- Input `nfilt` is loaded in register N5
- Output vector Y is located in Y memory

In X memory:

- X:(#INA) stores the vector idata
- X:(#INA+(length of idata)) stores the vector b
- X:(#INA+(length of idata)+(length of b)) stores the vector *zi/zf*
- X:(#INA+(length of idata)+(length of b)+(length of zi)) stores nout
- X:(#INA+(length of idata)+(length of b)+(length of zi)+1) stores the length of list
- X:(#INA+(length of idata)+(length of b)+(length of zi)+2) stores the vector list

In Y memory:

- Y:(#INB) stores the vector odata
- Y:(#INB+(length of idata)) stores the vector itemp
  The length of itemp is 2*(nfilt+1)

Input data length:

Length of idata is n

Length of odata is n -> n/r

Length of tb is nfilt+1

Length of zi,zf is nfilt+1

Length of list is = n/r

Length of itemp is 2*(nfilt+1)

Register usage:

- N2 stores the length of idata (n)
- N3 stores r
- N5 stores nfilt
- R2 stores the length of vector b (nfilt+1)

Status Register

No status registers or bits are set explicitly during function execution.

Data Size Limit

Input vector X length must longer than r*(nfilt+1). The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit

The value range of input vector X is [–1.0, +1.0].

Precision

In the case of DSP563, precision is 21 bits.

In the case of DSP566, precision is 12 bits.

decimate-fir-c.asm

MATLAB Usage

```
Y = mot###_decimate( X, r, nfilt, 'fir' )
```

Description

Function decimate resamples data at a lower rate after lowpass FIR filtering. Input vector X is a complex vector

Input/Output

Input Parameters: complex Vector X, `int r`, `int nfilt`

Output Parameters: complex Vector Y

Algorithm

The MEX function `decimate_fir.m` calculates vector `b`, vector `list`, `int nout`, and `int nbeg`,. The CMEX function loads these parameters into dsp memory for asm function use.

```
nd = length(idata);
m = size(idata,1);
nout = ceil(nd/r);

b = fir1(nfilt,1/r);
gd = grpdelay(b,1,8);
list = round(gd(1)+1.25):r:nd;
lod = length(list);
nlen = nout-lod;
nbeg = r-(nd-list(length(list)));

Assembly function decimate-fir-c.asm, and then follow these steps
to calculate decimated vector Y:
nfilt = nfilt+1
itemp = 2*idata(1)-idata((nfilt+1):-1:2)

[odata,zf] = filter(b,1,idata,zi)
odata = odata(list)
```

Memory & Register

Memory allocation:

- Input vector X is located in X memory
- Input `r` is loaded in register N3
- Input `nfilt` is loaded in register N5
- Output vector Y is located in Y memory

In X memory:

- X:(#INA) stores the vector `idata`

  (In the `idata` area, the first half stores the real part, the second half stores the imaginary part)

- X:(#INA+(length of `idata`)) stores vector `b`

  (In the `b` area, the first half stores the real part, and the second half stores the imaginary part)

- X:(#INA+(length of `idata`)+(length of `b`)) stores the vector `zi/zf`

  (In the `zi/zf` area, the first half stores the real part, and the second half stores the imaginary part)

- X:(#INA+(length of `idata`)+(length of `b`)+(length of `zi`)) stores `nout`
- X:(#INA+(length of `idata`)+(length of `b`)+(length of `zi`)+1) stores the length of `list`
- X:(#INA+(length of `idata`)+(length of `b`)+(length of `zi`)+2) stores the vector `list`
- X:(#INA-1) stores the length of `idata`
- X:(#INA-2) stores `r`
- X:(#INA-3) stores `nfilt`
- X:(#INA-4) stores the length of vector `b`, `nfilt+1`
- X:(#INA-5) stores the length of decimated output vector, `nout`
- X:(#INA-6) stores the length of list, `lod`
- X:(#INA-7) stores `nbeg`
- X:(#INA-8) stores `nlen = nout-lod`

In Y memory:

- Y:(#INB) stores the vector `odata`

  (In the `odata` area, the first half stores the real part, and the second half stores the imaginary part)

- Y:(#INB+2*(length of `odata`)) stores the vector `itemp`

  length of itemp is 2*2*(nfilt+1)

Input data length:

Length of `idata` is `2*`

Length of `odata` is `2*`

Length of `b` is `nfilt+1`

Length of `zi, zf` is `nfilt+1`

Length of `list` is `lod`

Length of `itemp` is `2*2*(nfilt+1)`

Register usage:

- N2 stores the length of `idata (n)`
- N3 stores `r`
- N5 stores `nfilt`
- R2 stores the length of vector `b (nfilt+1)`

Status Register

No status registers or bits are set explicitly during the function execution.

Data Size Limit

The input vector X length must longer than `r*(nfilt+1)`. The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit

The input vector X range is [–1.0, +1.0].

Precision

In the case of DSP563, precision is 21 bits.

In the case of DSP566, precision is 12 bits.

MATLAB Usage `Y = mot###_decimate( X, r, nfilt )`

Description

Function `decimate` resamples data at a lower rate after lowpass IIR filtering. Input vector X is a real vector

Input/Output

Input parameters: Real vector X, `int r`, `int nfilt`

Output parameters: Real vector Y

Algorithm

The MEX function `decimate_iir.m` calculates vector `b`, vector `a`, vector `zi`, `int nout`, and `int nbeg`. The CMEX function loads these parameters into `dsp` memory for `asm` function use.

```
nd = length(idata);
m = size(idata,1);
nout = ceil(nd/r);
rip = 0.05;
[b,a] = cheby1(nfilt, rip, 0.8/r);
while (abs(filtmag_db(b, a, 0.8/r)+rip)>1e-6)
       nfilt = nfilt - 1;
       if nfilt == 0
           break
       end
       [b,a] = cheby1(nfilt, rip, 0.8/r);
end

if nfilt == 0
       error('Bad Chebyshev design, likely R is too big; try mult.
decimation (R=R1*R2).')
end

len = m;
b = b(:).';
a = a(:).';
nb = length(b);
na = length(a);
n_filt = max(nb,na);
nfact = 3*(n_filt-1);  % length of edge transient

if nb < n_filt, b(n_filt)=0; end   % zero-pad if necessary
if na < n_filt, a(n_filt)=0; end
```

```
rows = [1:n_filt-1  2:n_filt-1  1:n_filt-2];
cols = [ones(1,n_filt-1) 2:n_filt-1  2:n_filt-1];
data = [1+a(2) a(3:n_filt) ones(1,n_filt-2)  -ones(1,n_filt-2)];

sp = sparse(rows,cols,data);
zi = sp \ ( b(2:n_filt).' - a(2:n_filt).'*b(1) );

nbeg = r-(r*nout-nd);
```

Assembly function decimate-iir-r.asm then follow these steps to calculate decimated vector Y:

```
y=[2*x(1)-x((nfact+1):-1:2);x;2*x(len)-x((len-1):-1:len-nfact)]
y = filter(b,a,y,[zi*y(1)])
y = y(length(y):-1:1)
y = filter(b,a,y,[zi*y(1)])
y = y(length(y):-1:1)
y([1:nfact len+nfact+(1:nfact)]) = []
nbeg = r - (r*nout - nd)
odata = odata(nbeg:r:nd)
```

Memory & Register

Memory allocation:

- Input vector X is located in X memory
- Input `r` is loaded in register N3
- Input `nfilt` is loaded in register N5
- Output vector Y is located in Y memory

In X memory:

- X:(#INA) stores vector `a`
- X:(#INA+(length of a)) stores vector `b`
- X;(#INA+(length of a)+(length of b)) stores vector `zi/zf`
- X:(#INA+(length of a)+(length of b)+(length of zi)) stores vector `idata`
- X:(#INA+(length of idata)+(length of b)+(length of zi)+1) stores the length of `list`
- X:(#INA+(length of idata)+(length of b)+(length of zi)+2) stores the vector `list`
- X;(#INA-1) stores `nbeg`

In Y memory:

- Y:(#INB) stores vector `odata (C)`
- Y:(#INB+(length of idata)+2*nfactvector) stores `odata (D)`

Input data length:

Length of `idata` is `n`

Length of `odata` (C,D) is `+2*nfact`

Length of `tb` is `nfilt+1`

`Length of zi, zf` is `nfilt+1`

Register usage:

- N2 stores the length of `idata (n)`
- N3 stores `r`
- N5 stores `nfilt`
- R2 stores the length of vector `b` (`nfilt+1`)

Status Register

The assembly function `decimate-iir-r.asm` does not set explicitly any status registers/bits during the function execution.

Data Size Limit

The input vector X length must longer than `r*(nfilt+1)`. The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit

The value range of the input vector X is [–1.0, +1.0].

Precision

In the case of DSP563, precision is 21 bits.

In the case of DSP566, precision is 12 bits.

decimate-iir-c.asm

MATLAB Usage `Y = mot###_decimate( X, r, nfilt )`

Description Function `decimate` resamples data at a lower rate after lowpass IIR filtering. Input vector X is a complex vector

Input/Output Input parameters: Complex vector X, `int r`, `int nfilt`

Output parameters: Complex vector Y

Algorithm The MEX function `decimate_iir.m` calculates vector `b`, vector `a`, vector `zi`, `int nout`, and `int nbeg`. The CMEX function loads these parameters into `dsp` memory for `asm` function use.

```
nd = length(idata);
m = size(idata,1);
nout = ceil(nd/r);

rip = 0.05;
[b,a] = cheby1(nfilt, rip, 0.8/r);
while (abs(filtmag_db(b, a, 0.8/r)+rip)>1e-6)
        nfilt = nfilt - 1;
        if nfilt == 0
           break
        end
        [b,a] = cheby1(nfilt, rip, 0.8/r);
end

if nfilt == 0
        error('Bad Chebyshev design, likely R is too big; try
mult. decimation (R=R1*R2).')
end

len = m;
b = b(:).';
a = a(:).';
nb = length(b);
na = length(a);
n_filt = max(nb,na);
nfact = 3*(n_filt-1);  % length of edge transient

if nb < n_filt, b(n_filt)=0; end   % zero-pad if necessary
if na < n_filt, a(n_filt)=0; end

rows = [1:n_filt-1  2:n_filt-1  1:n_filt-2];
cols = [ones(1,n_filt-1) 2:n_filt-1  2:n_filt-1];
data = [1+a(2) a(3:n_filt) ones(1,n_filt-2)  -ones(1,n_filt-2)];

sp = sparse(rows,cols,data);
zi = sp \ ( b(2:n_filt).' - a(2:n_filt).'*b(1) );

nbeg = r-(r*nout-nd);

Assembly function decimate-iir-r.asm, and then follow these steps
to calculate decimated vector Y:

y=[2*x(1)-x((nfact+1):-1:2);x;2*x(len)-x((len-1):-1:len-nfact)]
y = filter(b,a,y,[zi*y(1)])
y = y(length(y):-1:1)
y = filter(b,a,y,[zi*y(1)])
y = y(length(y):-1:1)
y([1:nfact len+nfact+(1:nfact)]) = []
nbeg = r - (r*nout - nd)
odata = odata(nbeg:r:nd)
```

Memory & Register

Memory allocation:

- Input vector X is located in X memory
- Input `r` is loaded in register N3
- Input `nfilt` is loaded in register N5
- Output vector Y is located in Y memory

In X memory:

- X:(#INA) stores vector `a`
  (In the `a` area, the first half stores the real part, and the second half stores the imaginary part)
- X:(#INA+2*(length of a)) stores vector `b`
  (In the `b` area, the first half stores the real part, and the second half stores the imaginary part)

- X:(#INA+2*(length of a)+2*(length of b)) stores vector `zi/zf`

  (In the `zi/zf` area, the first half stores the real part, and the second half stores the imaginary part)

- X:(#INA+2*(length of a)+2*(length of b)+2*(length of zi)) stores vector `idata`
- X:(#INA-1) stores the length of vector `idata`
- X:(#INA-2) stores `r`
- X:(#INA-3) stores `nfilt`
- X:(#INA-4) stores the length of vectors `a, b, zi, zf`
- X:(#INA-5) stores `nfact`
- X:(#INA-6) stores the length of `itemp`, `n+2*nfact`
- X:(#INA-7) stores `nout`
- X:(#INA-8) stores `nbeg`

In Y memory:

- Y:(#INB) stores vector `odata (C)`
- Y:(#INB+2*((length of idata)+2*nfact)) stores vector `odata (D)`

Input data length:

Length of `idata` is `n`

Length of `odata` (C,D) = `(the length of idata)+2*nfact`

Length of `tb` is `nfilt+1`

Length of `zi, zf` is `nfilt+1`

Register usage:

- N2 stores the length of `idata (n)`
- N3 stores `r`
- N5 stores `nfilt`
- R2 stores the length of vector `b (nfilt+1)`

Status Register The assembly function `decimate-iir-c.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit

Input vector X length must be longer than `r*(nfilt+1)`. The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 21 bits.

In the case of DSP566, precision is 12 bits.

diff-r.asm

MATLAB Usage

`B = mot###_diff(A,N)`

Description

This function performs an N-th order difference between the elements of real vector A.

INput/Output

Input: Real vector A, int N (N-th order)

Output: Real vector B

Algorithm

```
for ( i = 1; i <= N; i ++ ) {
    for ( j = 1; j <= LengthA - 1; j ++ ) {
        A[j] = A[j + 1] – A[j];
   }
   -- LengthA;
}
B = A;
```

Memory & Register

Memory allocation and register usage:

- R2 stores the length of A
- R3 stores the input N
- R4 stores the start address of vector A
- R0 stores the index of vector A
- R1 stores the index of vector B
- R2 and R3 are used as loop control

  Assume `M{0...7} = $ffff`

Status Register

The assembly function `diff-r.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit

The length of vector A can’t be larger than the continuous available data memory size.

Data Range Limit

The value of input vector must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 6 \times \Delta inputvectorlength = 6$$

DSP566:

$$\Delta\Delta cycle = 6 \times \Delta inputvectorlength = 6$$

diff-c.asm

MATLAB Usage `B = mot###_diff(A,N)`

Description This function performs an N-th order difference between the elements of complex vector A.

Input/Output Input: Complex vector A, int N (N-th order)

Output: Complex vector B

Algorithm

```
for ( i = 1; i <= N; i ++ ) {
    for ( j = 1; j <= LengthA - 1; j ++ ) {
        Ar[j] = Ar[j + 1] – Ar[j];
        Ai[j] = Ai[j + 1] – Ai[j];
   }
   -- LengthA;
}
Br = Ar;
Bi = Ai;
```

Memory & Register Memory allocation:

- X memory: label IN1_RL points to the start address of the real part of vector A
- Y memory: label IN1_RL points to the start address of the imaginary part of vector A

Register usage:

- R4 stores the start address of vector A
- R0 stores the index of the real part of vector A
- R6 stores the index of the imaginary part of vector A
- R1 stores the index of the real part of vector B
- R5 stores the index of the imaginary part of vector B
- R2 and R3 are used as loop control

  Assume `M{0...7} = $ffff`

Status Register The assembly function `diff-c.asm` does not set explicitly any status registers/bits during the function execution.

Data Size Limit

The length of vector A can't be larger than the continuous available data memory size.

Data Range Limit

The value of input vector A must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 6 \times \Delta inputvectorlength = 6$$

DSP566:

$$\Delta\Delta cycle = 6 \times \Delta inputvectorlength = 6$$

fft-r.asm

MATLAB Usage `Y = mot###_fft( X )`

Description This function returns the discrete Fourier transform (DFT) of the input for real vector X

Input/Output Input: Vector `Xe` (even index input data, located in X memory), and vector `Xo` (odd index input data, located in Y memory)

Output: Complex Vector Y (includes real output data of vector `Yr`, and imaginary data output of vector `Yi`)

Algorithm First, use algorithm in `FFT-C.ASM` to calculate length/2 complex data FFT.

Then use split algorithm to calculate final result.

No scaling is required for the input data. The output data should be scaled up by 2^(r2+1).

For example, to get true FFT values, after the FFT is done and r2=7, every output item has to be shifted left 8 bits.

Note The variable “size” used in the code below is the half size of the input for real vector X.

The split algorithm is described below:

```
for (k =0; k < size/2 –1; k++)
{
    H1r = (Xr[k] + Xr[size-k]) / 2;
    H1i = (Xi[k] – Xi[size-k])  / 2;
    H2r = (Xi[k] + Xi[size-k]) / 2;
    H2i = (Xr[size-k] – Xr[k]) / 2;
    Yr[k] = H1r + (C2r[k]*H2r – C2i[k]*H2i);
    Yr[size-k] = H1r – (C2r[k]*H2r – C2r[k]*H2i);
    Yi[k] = H1i + (C2i[k]*H2r- C2r[k]*H2i);
    Yi[size-k] = -(H1i) + (C2i[k]*H2r – C2r[k]*H2i);
}
```

Memory & Register

Memory allocation:

- In X memory: label IN_REAL stores the location of the input data (real part)
- In Y memory: label IN_IMAG stores the location of the input data (imaginary part)

Note The above two symbols are also used as output symbols

- p:$F000 = half the length of the input real vector
- p:$F001 = address offset of coefficient lookup table C1 (used by Fft-c)
- p:$F002 = address offset of coefficient lookup table C2 (used by split)
- p:$F003 = address offset of output vector

Register usage:

- R7 is used for passnum
- R2 is used for scaling exponent

Status Register

The assembly function `fft-r.asm` does not set any status registers/bits during the function execution.

Data Size Limit

The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

MATLAB Usage `Y = mot###_fft X )`

Description This function performs the discrete Fourier transform (DFT) of input complex vector X

Input/Output Input: Complex vector X (includes the real input data of vector `Xr`, and the imaginary data input of vector `Xi`)

Output: Complex vector Y (includes the real output data of vector `Yr`, and the imaginary output data of vector `Yi`)

Algorithm The algorithm is radix-2 DIT FFT.

```
Passnum = (int) (Log2(size) + 0.5);  //calculated by mex function
GroupPerPass= 1;
ButterflyPerGroup = size / 2;
Set coefficient table addressing mode as bit-reversed;
Clear scaling bit;
Set scaling down mode;
Scalexp = 1;  //scaling exponent
for(i = 0; i < Passnum; i++)
{
    pA = 0;  /*address pointer of the first input of butterfly */
    pB = pA + ButterflyPerGroup;/*address pointer of the second
input of butterfly */
    pC = 0;  /*address pointer of coefficient lookup table C */
    for (j = 0; j < GroupPerPass; j++)
    {
        for (k = 0; k < ButterflyPerGroup; k++)
       {
            Xr[pA] = Xr[pA] + Xr[pB]*Cr[pC] + Xi[pB]*Ci[pC];
            Xi[pA] = Xi[pA]  + Xi[pB]*Cr[pC] – Xr[pB]*Ci[pC];
            Xr[pB] = 2*Xr[pA] – Xr[pA];
            Xi[pB] = 2*Xi[pA] – Xi[pA];
            pA ++;
            pB ++;

}
Clr scaling down mode;
if has overflow
{
Set scaling down mode;
Clr scaling bit;
Scalexp++;
}
pA += ButterflyPerGroup;
pB += ButterflyPerGroup;
pC += size / 4; /* bit-reverse */
}
ButterflyPerGroup >>= 1;
GroupPerPass<<= 1;
  }
  Clear scaling bit;
  Set no scaling mode;
  Convert bit reverse order to normal order in-place;
```

Note Actual output data should be scaled up by 2^(Scalexp-1). This will be done by the MEX function.

Memory & Register

Memory allocation:

- X memory: IN_REAL stores the start address of the real part of the input data
- Y memory: IN_IMAG stores the start address of the imaginary part of the input data

Note The above two symbols are also used as output symbol

Register usage:

- R3 stores the length of the input vector
- R7 stores the passnum
- R2 stores the scaling exponent

Status Register The assembly function `fft-c.asm` does not explicitly set any status registers/ bits during the function execution

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

MATLAB Usage `[D,ZF] = mot###_filter(B, A, C, ZI)`

Description This function filters the data in Real vector C with the filter described by real vectors A and B to create the filtered data D, using Zi as initial conditions

Input/Output Input: real vector C, real vector A and B

Output: real vector D, real vector Zf

Algorithm

```
// Zi should be increased by adding one zero value to it.
// The length of Zi will be equal to Max(LengthA, LengthB)
// LengthD = LengthC

for (j = 1; j <= LenghtD; j++ ) {
if ( Abs ( A[1] ) != 1 ) {
        D[j] = B[1]*C[j]/A[1] + Zi[1];
        for (i = 1; i <= (LengthZi – 1); i ++) {
            Zi[i] = B[i+1]*C[j]/A[1] + Zi[i+1] – A[i+1]*D[j]/
A[1];
       }
}
   else {
        D[j] = B[1]*C[j] + Zi[1];
        for (i = 1; i <= (LengthZi – 1); i ++) {
            Zi[i] = B[i+1]*C[j] + Zi[i+1] – A[i+1]*D[j];
        }
   }
}
```

Memory & Register Memory allocation:

In X memory:

- X:(#INA) stores the start address of vector A
- X:(#INA + LENGTH(A)) stores the start address of vector B
- X:(#INA + LENGTH(A) + LENGTH(B)) stores the start address of vector ZI

In Y memory:

- Y:(#INB) stores the start addresses of vector C and vector D

Register usage:

- R2 stores the length of vector MAX(A,B), ZI
- R6 stores the length of vector C
- R3 stores the start address of Zi
- R0 stores the index of vector A
- R0+N0 store the index of vector B
- R4 store the index of vectors C and D
- R1 stores the index of vector Zi
- R6 and R7 are used as loop control
- R5 is unused

Assumes `M{0...7} = $ffffff`

Status Register

The assembly function `filter-r.asm` does not set explicitly any status registers/bits during the function execution.

Data Size Limit

The length of vector C can't be larger than the continuous available data memory size.

Data Range Limit

The value of input vectors must be between –1.0 and +1.0.

Precision

In the case of DSP563, the precision is 21 bits.

In the case of DSP566, the precision is 12 bits

Performance Limit

DSP563:

$$\Delta cycle = 12 \times \Delta zi = 12$$

DSP566:

$$\Delta cycle = 12 \times \Delta zi = 12$$

MATLAB Usage `[Dr, Di, Zfr, Zfi] = mot###_filter(Br, Bi, Ar, Ai, Cr, Ci, Zir, Zii)`

Description This function filters the data in vector C with the filter described by vectors A and B to create the filtered data D, using Zi as initial conditions

Input/Output Input: Complex vector A, vector B, vector C, and vector Zi

Output: Complex vector D, and vector Zf

Algorithm

```
// Zi should be increased by adding one zero value to it.
// The length of Zi will be equal to Max(LengthA, LengthB)

LengthD = LengthC
for (j = 1; j <= LenghtD; j++ ) {
if ( Abs ( A[1] ) != 1 ) {
D[j] = (A[1]*B[1]*C[j] – A[1]*BI[1]*CI[j]  + AI[1]*B[1]*CI[j]
 + AI[1]*BI[1]*C[j]) / (A[1]*A[1] + AI[1]*AI[1]) + Zi[1];
   DI[j] = (A[1]*B[1]*CI[j] + A[1]*BI[1]*C[j] + AI[1]*BI[1]*CI[j]
– AI[1]*B[1]*C[j]) / (A[1]*A[1] + AI[1]*AI[1]) + ZiI[1];
for (i = 1; i <= (LengthZi – 1); i ++) {
Zi[i] = (A[1]*B[i+1]*C[j] – A[1]*BI[i+1]*CI[j]  +
AI[1]*B[i+1]*CI[j] + AI[1]*BI[i+1]*C[j]
- A[1]*A[i+1]*D[j] + A[1]*AI[i+1]*DI[j]
- AI[1]*A[i+1]*DI[j] - AI[1]*AI[i+1]*D[j]) /
(A[1]*A[1] + AI[1]*AI[1]) + Zi[i+1];
ZiI[i] = (A[1]*B[i+1]*CI[j] + A[1]*BI[i+1]*C[j]  +
AI[1]*BI[i+1]*CI[j] - AI[1]*B[i+1]*C[j]
- A[1]*A[i+1]*DI[j] - A[1]*AI[i+1]*D[j]
- AI[1]*AI[i+1]*DI[j] + AI[1]*A[i+1]*D[j]) /
(A[1]*A[1] + AI[1]*AI[1]) + ZiI[i+1];
}
}
else {
        D[j] = B[1]*C[j] – BI[1]*CI[j] + Zi[1];
        DI[j] = BI[1]*C[j] + B[1]*CI[j] + ZiI[1];
        for (i = 1; i <= (LengthZi – 1); i ++) {
            Zi[i] = B[i+1]*C[j] – BI[i+1]*CI[j] + Zi[i+1]
– A[i+1]*D[j] + AI[i+1]*DI[j];
                ZiI[i] = BI[i+1]*C[j] + B[i+1]*CI[j] + ZiI[i+1]
– AI[i+1]*D[j] - A[i+1]*DI[j];
```

}

}

}

Memory & Register

Memory allocation:

In X memory:

- X:(#INA) stores the start address of data of vector A
- X:(#INA+LENGTH(A)*2) stores the start address of vector B
- X:(#INA+LENGTH(A)*2 + LENGTH(B)*2) stores the start address of vector ZI

In Y memory:

- Y:(#INB) stores the start address of the vector C data
- Y:(#INB+LENGTH(C)*2) stores the start address of the vector D data

Register usage:

- R2 stores the length of vector MAX(A,B),ZI
- R6 stores the length of vector C
- N7 stores the start address of A
- N2 stores the start address of B
- N6 stores the start address of Zi
- R0 stores the index of A
- R1 stores the index of B
- R3 stores the index of Zi
- R4 stores the index of C
- R5 stories the index of D
- R2 -> N0, N1, N3
- R6 -> N4, N5
- R6 and R7 are used as loop control

Assumes `M{0...7} = $ffffff`

Status Register

The assembly function `filter-c.asm` does not set explicitly any status registers/bits during the function execution.

Data Size Limit

The length of vector C can't be larger than the continuous available data memory size.

Data Range Limit

The value of input vectors must be between –1.0 and +1.0.

Precision

In the case of DSP563, the precision is 21 bits.

In the case of DSP566, the precision is 12 bits

Performance Limit

DSP563:

$$\Delta cycle = 27 \times \Delta zi = 27$$

DSP566:

$$\Delta cycle = 27 \times \Delta zi = 27$$

ifft-r.asm

MATLAB Usage `Y = mot###_ifft( X )`

Description

This function returns the discrete Fourier transform (DFT) of the input for real vector X

Input/Output

Input: Vector Xe (even index input data, located in X memory), and vector Xo (odd index input data, located in Y memory)

Output: Vector Yr (real data of output), and vector Yi (imaginary data of output)

Algorithm

First, use the algorithm in FFT-C.ASM to calculate the length/2 complex data FFT.

Then, use the split algorithm to calculate the final results.

No scaling is required for the input data; the output data should be scaled up by `2^(r2+1)`.

For example, after the FFT is done and `r2=7`, to get true FFT values, every output item has to be shifted left 8 bits.

The scaling algorithm is described below:

Note The variable "size" used below is half the size of the input of the real vector X

```
for (k =0; k < size/2 –1; k++)
{
   H1r = (Xr[k] + Xr[size-k]) / 2;
   H1i = (Xi[k] – Xi[size-k])  / 2;
   H2r = (Xi[k] + Xi[size-k]) / 2;
   H2i = (Xr[size-k] – Xr[k]) / 2;
   Yr[k] = H1r + (C2r[k]*H2r – C2i[k]*H2i);
   Yr[size-k] = H1r – (C2r[k]*H2r – C2r[k]*H2i);
   Yi[k] = H1i + (C2i[k]*H2r- C2r[k]*H2i);
   Yi[size-k] = -(H1i) + (C2i[k]*H2r – C2r[k]*H2i);
}
```

Memory & Register

Memory allocations:

- X memory: IN_REAL stores the location of the real part of the input data
- Y memory: IN_IMAG stores the location of the imaginary part of the input

The above two symbols are also used as output symbols.

p:$F000 = half the length of the input real vector

p:$F001 = the address offset of the coefficient lookup table, C1 (used by Fft-c)

p:$F002 = the address offset of the coefficient lookup table, C2 (used by split)

p:$F003 = the address offset of the output vector

Register usage:

- R7 is used for passnum
- R2 is used for the scaling exponent

Status Register

The assembly function `ifft-r.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit

The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

ifft-c.asm

MATLAB Usage `Y = mot###_ifft( X )`

Description This function returns the discrete Fourier transform (DFT) of the input for complex vector X

Input/Output Input: Complex vector X, including vector Xr (real input) and vector Xi (imaginary input)

Output: Complex vector Yr (real output), and vector Yi (imaginary output)

Algorithm

Note The algorithm is radix-2 DIT FFT.

```
Passnum = (int) (Log2(size) + 0.5);//calculated by mex function
GroupPerPass= 1;
ButterflyPerGroup = size / 2;
Set coefficient table addressing mode as bit-reversed;
Clear scaling bit;
Set scaling down mode;
Scalexp = 1;//scaling exponent
for(i = 0; i < Passnum; i++)
{
    pA = 0;/*address pointer of the first input of butterfly */
    pB = pA + ButterflyPerGroup;/*address pointer of the second
input of butterfly */
    pC = 0;/*address pointer of coefficient lookup table C */
    for (j = 0; j < GroupPerPass; j++)
    {

for (k = 0; k < ButterflyPerGroup; k++)
       {
           Xr[pA] = Xr[pA] + Xr[pB]*Cr[pC] + Xi[pB]*Ci[pC];
           Xi[pA] = Xi[pA]  + Xi[pB]*Cr[pC] – Xr[pB]*Ci[pC];
           Xr[pB] = 2*Xr[pA] – Xr[pA];
           Xi[pB] = 2*Xi[pA] – Xi[pA];
           pA ++;
           pB ++;
       }
Clr scaling down mode;
if has overflow
```

```
            {
                 Set scaling down mode;
                 Clr scaling bit;
                 Scalexp++;
            }
            pA += ButterflyPerGroup;
            pB += ButterflyPerGroup;
            pC += size / 4;/* bit-reverse */
        }
        ButterflyPerGroup >>= 1;
        GroupPerPass<<= 1;
     }
     Clear scaling bit;

     Set no scaling mode;
     Convert bit reverse order to normal order in-place;
```

Note The actual output data should be scaled up by 2^(Scalexp-1). This will be done by the MEX function.

Memory & Register

Memory allocation:

- X memory: IN_REAL stores the start address of the real part of the input data
- Y memory: IN_IMAG stores the start address of the imaginary part of the input data

Above two symbols are also used as output symbols

Register usage:

- R3 stores the address of the input vector
- R7 is used for passnum
- R2 is used for the scaling exponent

Status Register

The assembly function `ifft-c.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

MATLAB Usage

`Y = mot###_interp(IDATA, R, LEN, B)`

Description

This function resamples data at a higher rate using lowpass interpolation. The input data is real.

Y = INTERP (IDATA, R, LEN, B) resamples the sequence in vector IDATA at R times the original sample rate. The resulting resampled vector Y is R times longer, LENGTH(Y) = R*LENGTH(IDATA). A symmetric filter, B, allows the original data to pass through unchanged and interpolates between so that the mean square error between them and their ideal values is minimized. B is the interpolation filter.

Input/Output

Input: Real vector IDATA, int R, int LEN, and real vector B

Output: Real vector Y

Algorithm

```
int I;
for ( I = 0; I < lengthIDATA*R; I ++)
{
    Y[I] = 1;
}
for ( I = 0; I < lengthIDATA ; I = I ++)
{
    Y[I*R] = IDATA[I];
}
for( I = 0; I < 2*LEN*R; I ++)
{
    OD[I] = 0;
}

for (I = 0; I < 2 * LEN; I++)
{
    OD[I*R] = 2*IDATA[LEN-1] –IDATA[2*LEN – I];
}
/* Call filter  */
[OD, ZI] = filter(B, LEN, OD);
[Y, ZF] = filter(B, LEN, Y, ZI];
for( I = 0, I < (lengthIDATA –LEN) * R; I++)
{
    Y[I] = Y[LEN*R+I];
}

for( I = 0; I < 2*LEN*R; I ++)
{
   OD[I] = 0;
}

for (I = 0; I < 2*LEN; I++)
{
   OD[I*R] = 2*IDATA[lengthIDATA] – IDATA[lengthIDATA –1 –I];
}
OD = filter (B, LEN, OD, ZF);
for(I=0;I < LEN*R; I++)
{
   Y[lengthIDATA*R – LEN*R + I] = OD[I];
}
```

Memory & Register

Memory allocation

In X memory:

- X:(#INA) stores the start address of vector `idata`
- X:(INA+n) stores the start address of vector `tb`

In Y memory:

- X:(#INB) stores the start address of vector `odata`
- X:(INB+n*r) stores the start address of vector `od`

Register usage:

- N2 stores the location of `n`
- N3 stores the location of `r`
- N5 stores the location of `l`
- R2 stores the length of vector `tb` (2*r*l+1)
- N7 stores the location of `n*r`
- N0, N1, N4, and N6 are used for offset addressing

Status Register

The assembly function `interp-r.asm` does not set explicitly any status registers/bits.

Data Size Limit

The length of vector IDATA can't be larger than the continuous available data memory size.

Data Range Limit

The input data vector range is from –1.0 to +1.0, inclusive.

Precision

In the case of DSP563, precision is 21 bits.

In the case of DSP566, precision is 12 bits.

interp-c.asm

MATLAB Usage `Y = mot###_interp(IDATA, R, LEN, B)`

Description This function resamples data at a higher rate using lowpass interpolation. The input data is complex data.

Y = INTERP (IDATA, R, LEN, B) resamples the sequence in vector IDATA at R times the original sample rate. The resulting resampled vector Y is R times longer, LENGTH(Y) = R*LENGTH(IDATA). A symmetric filter, B, allows the original data to pass through unchanged and interpolates between so that the mean square error between them and their ideal values is minimized.

Input/Output Input: Complex vector IDATA, int R, int LEN, and vector B

Output: Complex vector Y

Algorithm

```
int I;
for ( I = 0; I < lengthIDATA*R; I ++)
{
   Y_REAL [I] = 1;
   Y_IMAGE [I] = 1;
}
for ( I = 0; I < lengthIDATA ; I = I ++)
{
   Y_REAL[I*R] = IDATA_REAL[I];
   Y_IMAGE[I*R] = IDATA_IMAGE[I];
}
for( I = 0; I < 2*LEN*R; I ++)
{

OD_REAL[I] = 0;
   OD_IMAGE[I] = 0;
}

for (I = 0; I < 2 * LEN; I++)
{
   OD_REAL[I*R] = 2*IDATA_REAL [LEN-1] –IDATA_REAL [2*LEN – I];
   OD_IMAGE[I*R] = 2*IDATA_IMAGE [LEN-1] –IDATA_IMAGE [2*LEN –
I];
}
/* Call filter  */
```

```
[OD, ZI] = filter(B, LEN, OD);
[Y, ZF] = filter(B, LEN, Y, ZI];
for( I = 0, I < (lengthIDATA –LEN) * R; I++)
{
   Y_REAL[I] = Y_REAL[LEN*R+I];
Y_IMAGE[I] = Y_IMAGE[LEN*R+I];
}
for( I = 0; I < 2*LEN*R; I ++)
{
   OD_REAL[I] = 0;
   OD_IMAGE[I] = 0;
}

for (I = 0; I < 2*LEN; I++)
{
OD_REAL[I*R] = 2*IDATA_REAL[lengthIDATA] – IDATA_REAL[lengthIDATA
–1 –I];
OD_IMAGE[I*R] = 2*IDATA_IMAGE[lengthIDATA] –
IDATA_IMAGE[lengthIDATA –1 –I];
}
OD = filter (B, LEN, OD, ZF);
for(I=0;I < LEN*R; I++)
{
   Y_REAL[lengthIDATA*R – LEN*R + I] = OD_REAL[I];
   Y_IMAGE[lengthIDATA*R – LEN*R + I] = OD_IMAGE[I];
}
```

Memory & Register

Memory allocation:

X memory:

- #INA stores the start address of vector `idata`
- #INA+2*n stores the start address of vector `tb`

Y memory:

- #INB stores the start address of vector `odata`
- #INB+2*n*r stores the start address of vector `od`

Register usage:

- N2 stores the location of `n`
- N3 stores the location of `r`
- N5 stores the location of `l`
- R2 stores the length of vector `tb` (`2*r*l+1`)
- N7 stores the location of `n*r`
- N0, N1, N4, and N6 are used for offset addressing

Status Register

The assembly function `interp-c.asm` does not set explicitly any status registers/bits during the function execution.

Data Size Limit

The length of vector IDATA can't be larger than the continuous available data memory size.

Data Range Limit

The value of input vectors must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 21 bits.

In the case of DSP566, precision is 12 bits.

MATLAB Usage `Y = mot###_log(X)`

Description This function returns the natural logarithm elements of input vector X. The input vector X is a real vector

Input/Output Input: Real vector X

Output: Real vector Y

Algorithm Input:

```
Z in a1  (a2 = 0, a0 = 0)
```

Note The input should be a positive fraction.

Output:

```
log(Z) in a1

(a2 is sign extended and a0=don't care).
log(Z) = log2(Z)*log(2)
log2(Z)=  pi3*(Z**3) +pi2*(Z**2) + pi1*Z + pi0
     i = 1, 2, 3.
```

```
The range of the input number is divided into three different
subranges and corresponding pij's (i=1,2,3; j=0,1,2,3) are used for
each range.
Range 1:
1 > Z > 0.8
    p13 =  0.6651550174712363
    p12 = -2.691225081621167
    p11 =  4.830861130814611
    p10 = -2.8047790282999791

Range 2:
0.8 >= Z > 0.64
    p23 =  1.299130893496323
    p22 = -4.205039190029353
    p21 =  6.038576413515295
    p20 = -3.126707123185897

Range 3:
0.64 >= Z >= 0.5

    p33 =  2.62317472043452
    p32 = -6.720343733123614
    p31 =  7.635399480799159
    p30 = -3.465490657054861


In the following pseudocode cij = pij/8 (i=1,2,3)
 (j=0,1,2,3)
log2nrm(Z) = [log2(ZS)/8 + (-S)/8]/2

Find S such that ZS = (2**S)*Z lies in the range [0.5,1]
Find the range of ZS.
If 1 > ZS > 0.8
Range 1:
   Find Term1_1 = c13* ZS + c12
   Find Term1_2 = Term1_1*ZS + c11
   Find log2(ZS)/8 = Term1_2 * ZS + c10


If 0.8 >= ZS > 0.64
Range  2:
   Find Term2_1 = c23 * ZS + c22
   Find Term2_2 = Term2_1 * ZS +c21
   Find log2(ZS)/8 = Term2_3 * ZS + c20


If  0.64 >= ZS >= 0.5
Range 3:
   Find Term3_1 = c33 * ZS + c32
   Find Term3_2 = Term3_1 * ZS +c31
   Find log2(ZS)/8 = Term3_2 * ZS + c30
```

Memory & Register

Memory allocation

X memory:

- #IN1 & #OUT1 stores the location of vector A

Y memory:

- $0 stores the coefficient of the polynomial

Register usage:

- R6 stores the length of vector A
- R3 is used for the shift bit:
  6 for dsp56300
  5 for dsp56600
- R0 stores the index of vector A
- R1 stores the index of the coefficient
- R6 is used as loop control
- R2 stores the address for the vector `tmp`
- R3 is used for the input shift bit
- R4, R5, and R7 are unused

Assumes `M{0...7} = $ffff`

Status Register

The assembly function `log-r.asm` does not set explicitly any status registers/ bits during the function execution.

Data Size Limit

The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 16 bits.

In the case of DSP566, precision is 12 bits.

log-c.asm

MATLAB Usage

`Y = mot###_log( X )`

Description

This function returns the natural logarithm elements of complex input vector X.

Input/Output

Input: Complex vector X

Output: Complex vector Y

Algorithm

```
  log(a+bi) = log(sqrt(a^2 + b^2))+angle(a,b)*i
```

If an absolute result is needed, the real data of the result should be scaled up by 32 and imaginary data of the result should be scaled up by 4. Please refer to the algorithms used in `log-r.asm` on page 3-59 and `angle-c.asm` on page 3-14.

Memory & Register

Memory allocation:

In X memory:

- XYZ is used as a temporary variable area

```
  +0--X  +1--Y  +2--log(sqrt(Xr[i]*Xr[i]+Xi[i]*Xi[i])/32(or 16)
```

ATANTAB stores the atan table

Y memory:

- $10 is used for the coefficient of polynomial
- X:IN_REAL stores the start address of the real input data of vector A
- Y:IN_IMAG stores the start address of the imaginary input data of vector A
- X:IN_REAL stores the start address of the real output data of vector A
- Y:IN_IMAG stores the start address of the imaginary output data of vector A

Register usage:

- R0 points to the input vector area
- R1 is used for the index of coefficient
- R2 is used for loopnum
  In the case of dsp56300, loopnum = 22
  In the case of dsp56600, loopnum = 14

- R3 points to the start address of table ATANTAB
- R4 is used for temporary storage
- N4 is used for shift bits

  6 for dsp56300

  5 for dsp56600

- R5 points to the output vector area
- R6 stores the length of the input vector
- R7 points to the temporary variable area XYZ

  Assumes `M{0...7} = $ffff`

Status Register The assembly function `log-c.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 16 bits.

In the case of DSP566, precision is 12 bits.

log10-r.asm

MATLAB Usage `Y = mot###_log10(X)`

Description This function computes the base 10 logarithm of real vector X

Input/Output Input: Real vector X

Output: Real vector Y

Algorithm Input:

```
Z in a1(a2 = 0, a0 = 0)
```

Note The input should be a positive fraction.

Output:

```
log10(Z) in a1

(a2 is sign extended and a0=don't care)

log10(Z) = log2(Z)*log10(2)
log2(Z) =  pi3*(Z**3) +pi2*(Z**2) + pi1*Z + pi0
i = 1, 2, 3.

The range of the input number is divided into three different
subranges and corresponding pij's (i=1,2,3; j=0,1,2,3) are used
for each range.

Range 1:
   1 > Z > 0.8

         p13 =  0.6651550174712363
         p12 = -2.691225081621167
         p11 =  4.830861130814611
         p10 = -2.8047790282999791

Range 2:
   0.8 >= Z > 0.64
```

```
         p23 =  1.299130893496323
         p22 = -4.205039190029353
         p21 =  6.038576413515295
         p20 = -3.126707123185897

Range 3:
   0.64 >= Z >= 0.5

         p33 =  2.62317472043452
         p32 = -6.720343733123614
         p31 =  7.635399480799159
         p30 = -3.465490657054861

In the following pseudocode cij = pij/8 (i=1,2,3)
 (j=0,1,2,3)
log2nrm(Z) = [log2(ZS)/8 + (-S)/8]/2

Find S such that ZS = (2**S)*Zlies inthe range [0.5,1)
Find the range of ZS.

If 1 > ZS > 0.8

Range1:

Find Term1_1 = c13* ZS + c12
Find Term1_2 = Term1_1*ZS + c11
Find log2(ZS)/8 = Term1_2 * ZS + c10

If 0.8 >= ZS > 0.64

Range2:

Find Term2_1 = c23 * ZS + c22
Find Term2_2 = Term2_1 * ZS +c21
Find log2(ZS)/8 = Term2_3 * ZS + c20

If  0.64 >= ZS >= 0.5

Range 3:
```

```
Find Term3_1 = c33 * ZS + c32
Find Term3_2 = Term3_1 * ZS + c31
Find log2(ZS)/8 = Term3_2 * ZS + c30
```

Memory & Register

Memory allocation:

X memory:

- #IN1 & #OUT1 are used for vector A

Y memory:

- $0 is used for the coefficient of polynomial

Register usage:

- R6 stores the length of vector A
- R3 is used for the shift bit
  6 for dsp56300
  5 for dsp56600
- R0 stores the index of vector A
- R1 stores the index of coefficient
- R6 is used as loop control
- R2 stores the address for the `tmp` vector
- R3 is used for the input shift bit
- R4, R5, and R7 are unused

Assumes `M{0...7} = $ffff`

Status Register

The assembly function `log10-r.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit

The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 16 bits.

In the case of DSP566, precision is 12 bits.

log10-c.asm

MATLAB Usage

`Y = mot###_log10( X )`

Description

This function returns the base 10 logarithm of complex input vector X.

Input/Output

Input: Complex vector X

Output: Complex vector Y

Algorithm

```
  log10(a+bi) = log10(sqrt(a^2 + b^2))+angle(a,b)*log10(e)*i
```

If an absolute result is needed, the real data of the result should be scaled up by 32 and imaginary data of the result should be scaled up by 4. Please refer to the algorithms used in `log10-r.asm` on page 3-64 and `angle-c.asm` on page 3-14.

Memory & Register

Memory allocation:

In X memory:

- XYZ is used as a temporary variable area
  ```
  +0--X  +1--Y  +2--log(sqrt(Xr[i]*Xr[i]+Xi[i]*Xi[i])/32(or 16)
  ```

ATANTAB stores the atan table

Y memory:

- $10 is used for the coefficient of polynomial
- X:IN_REAL stores the start address of the real data input of vector A
- Y:IN_IMAG stores the start address of the imaginary data input of vector A
- X:IN_REAL stores the real data output for vector A
- Y:IN_IMAG stores the imaginary data output for vector A

Register usage:

- R0 points to the input vector area
- R1 is used for the index of coefficient
- R2 is used as loopnum
  In the case of dsp56300, loopnum is 22
  In the case of dsp56600, loopnum is 14

- R3 points to the start address of table ATANTAB
- R4 is used for temporary storage
- N4 is used for shift bits

  6 for dsp56300

  5 for dsp56600
- R5 points to the output vector area
- R6 stores the length of the input vector
- R7points to the temporary variable area XYZ

  Assumes `M{0...7} = $ffff`

Status Register The assembly function `log10-c.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 16 bits.

In the case of DSP566, precision is 12 bits.

max-1r.asm

MATLAB Usage `[y, index] = mot###_max(X)`

Description This function returns the largest element of real input vector X and its index

Input/Output Input: Real vector X

Output: Real value Y (largest element of X), index

Algorithm

```
#define    IN0
MaxValue = X[0];
Index = 0;
for(i=1; i<size; i++)
{
if(X[i] > MaxValue)
{
    MaxValue = X[i];
    index = i;
}
}
```

Memory & Register Memory allocation:

X memory: label IN points to the location of the real data input of vector X.

Register usage:

- Register A stores Max-1r Result y
- R1 stores the index of the largest element of X
- R4 stores the number of items in the array

Status Register The assembly function `max-1r-r.asm` does not explicitly set any status registers/bits.

Data Size Limit The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

DSP566:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

max-1c.asm

MATLAB Usage `[yr, yi, index] = mot###_max( X )`

Description This function returns the largest element of input complex vector X and its index

Input/Output Input: Complex vector X (includes real part `Xr` and imaginary part `Xi`)

Output: Complex vector Y (includes real data of the largest element `Yr`, and the imaginary data of the largest element Yi), and index

Algorithm

```
#define IN 0
MaxValue = Xr[0]* Xr[0] + Xi[0]* Xi[0];
Yr = Xr[0];
Yi = Xi[0];
Index = 0;
for(i=1; i<size; i++)
{
   Intervalue = Xr[i]* Xr[i] + Xi[i]* Xi[i];
   if(Intervalue > MaxValue)
   {
        MaxValue = Intervalue;
        Yr = Xr[i];
        Yi = Xi[i];
        index = i;
   }
}
```

Memory & Register Memory allocations:

X memory:

- Label IN_REAL stores the start address of the real data

Y memory:

- Label IN_IMAG stores the start address of the imaginary data
- Label IN stores the address offset of the input of the complex vector X

Register usage:

- X1stores the address of the real data of the result

- Y1 stores the address of the image data of the result
- R1 stores the address of the index of the result
- R7is used for the number of items in the array

Status Register

The assembly function `max-1c.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit

The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

max-2r.asm

MATLAB Usage `X = mot###_max(X, Y)`

Description This function returns a vector the same size as input vectors X and Y with the largest elements taken from vector X or Y,

Input/Output Input: Vector X (real data), vector Y (real data)

Output: Vector X (largest element taken from X or Y)

Algorithm

```
#define IN1 x:$0
#define IN2 y:$0
#define OUT x:$0

MaxValue = X[0];
for(i=1; i<size; i++)
{
   if(X[i] < Y[i])
   {
       X[i] = Y[i];
   }
}
```

Memory & Register X memory IN1 start address of location of input real vector X

Y memory IN2 start address of location of input real vector Y

X memory OUT start address of location of output real vector

Register R7 store number of items in the input array

Status Register The assembly function `max-2r.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit The length of input vectors can't be larger than the continuous available data memory size.

Data Range Limit Input data vector range [–1.0, +1.0]

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 4 \times \Delta inputvectorlength = 4$$

DSP566:

$$\Delta\Delta cycle = 4 \times \Delta inputvectorlength = 4$$

max-2c.asm

MATLAB Usage

`[ X ] = mot###_max( X, Y )`

Description

This function returns the largest elements taken from complex vector X or Y

Input/Output

Input: Complex Vector X (include real part Xr and imaginary part Xi), Complex Vector Y (include real part Yr and imaginary part Yi)

Output: Complex Vector X (include real part Xr and imaginary Xi)

Algorithm

```
for(i=0; i<size; i++)
{
   xValue = Xr[i]* Xr[i] + Xi[i]* Xi[i];
   yValue = Yr[i]* Yr[i] + Yi[i]* Yi[i];
   if(xValue < yValue)
       X[i] = Y[i];
}
```

Memory & Register

Start address of Vector X is in P:$F000

Start address of Vector Y is in P:$F001

Start address of result Vector is also in P:$F000

Register R7 stores the number of items in the array

Status Register

The assembly function max-2c.asm does not explicitly set any status registers/bits during the function execution.

Data Size Limit

The length of input vectors can't be larger than the continuous available data memory size.

Data Range Limit

Input data vector X range [–1.0, +1.0]

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

MATLAB Usage `y = mot###_mean(X)`

Description This function returns the mean value of the elements in real vector X

Input/Output Input: Real vector X

Output: Y (the mean value of the elements in real vector X)

Algorithm

```
y = 0;
for(i=0; i<size; i++)
{
   y += X[i];
}
y = div (y, size);
* The algorithm of div is :
fractional div ( fractional y, int size)
{
   Bit1 = count leading bits of variable size;
   Bit1 += fractionlenght ;
Normalize (size);
Bit2 = count leading bits of variable y;
Normalize (y);
y = y>>1;
Bit2 ++;
Bit1 -= Bit2;
fractional-div(y, size);//this is a standard algorithm
y >>= Bit1;
return y;  }
```

Memory & Register Input / Output:

X memory IN start address of location of input real vector X.

Register X0 store number of items in the array

Register R7 store bits/word, used by division (24 for 56300,16 for 56600)

Register A1 store Result

Status Register

The assembly function mean-r.asm does not explicitly set any status registers/ bits during the function execution.

Data Size Limit

The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit

Input data vector X range [–1.0, +1.0]

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 1 \times \Delta inputvectorlength = 1$$

DSP566:

$$\Delta\Delta cycle = 1 \times \Delta inputvectorlength = 1$$

MATLAB Usage `[y] = mot###_mean( X )`

Description This function returns the mean value of the elements in complex input vector X

Input/Output Input: Complex vector X, which includes vector Xr (real part of input X) and vector Xi (imaginary part of input X)

Output: scalar y, which includes yr (the mean value of the real elements in X), yi (the mean value of the imaginary elements in X)

Algorithm

```
yr = yi = 0;
for(i=0; i<size; i++)
{
   yr += Xr[i];
   yi += Xi[i];
}
yr = div (yr, size);
yi = div (yi, size);
```

Memory & Register IN start address of location of input complex vector X.

X memory IN_REAL start address of real data of X

Y memory IN_IMAG start address of imaginary data of X

Register X0 store number of items in the array

Register X1 store Result of mean value(real)

Register Y1 store Result of mean value(imaginary)

Register R7 store bits/word, used by division (24 for 56300,16 for 56600)

Status Register The assembly function mean-r.asm does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit

Input data vector X range [–1.0, +1.0]

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

DSP566:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

MATLAB Usage `[y, index] = mot###_min(X)`

Description This function returns the smallest element of input real vector X and its index

Input/Output Input: Real vector X

Output: Real value Y (smallest element of X), index

Algorithm

```
#define IN0
MinValue = X[0];
Index = 0;
for(i=1; i<size; i++)
{
   if(X[i] < MinValue)
{
   MinValue = X[i];
   index = i;
}
}
```

Memory & Register Label IN present location of input real vector X.

Register A store Min-1r Result y

R1 = Index of largest element of X

R4 = Number of items in the array

Status Register The assembly function max-1r-r.asm does not explicitly set any status registers/bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit Input data vector X range [–1.0, +1.0]

Precision In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

DSP566:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

MATLAB Usage `[y, index] = mot###_min( X )`

Description This function returns the smallest element of complex input vector X and its index

Input/Output Input: Complex vector X, include real part Xr and imaginary part Xi

Output: scalar y, include yr (real data of the smallest element of X), yi (imaginary data of the smallest element of X), index

Algorithm

```
#define IN0
MinValue = Xr[0]* Xr[0] + Xi[0]* Xi[0];
yr = Xr[0];
yi = Xi[0];
Index = 0;
for(i=1; i<size; i++)
{
   Intervalue = Xr[i]* Xr[i] + Xi[i]* Xi[i];
   if(Intervalue < MinValue)
{
MinValue = Intervalue;
yr = Xr[i];
yi = Xi[i];
index = i;
}
}
```

Memory & Register

X memory IN_REAL = start address of real data

Y memory IN_IMAG = start address of imaginary data

IN = address offset of input complex vector X

Register X1 = real data of the result

Y1 = image data of the result

R1 = index of the result

R7 = number of items in the array

Status Register
The assembly function min-1c.asm does not explicitly set any status registers/bits during the function execution.

Data Size Limit
The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit
Input data vector X range [–1.0, +1.0]

Precision
In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

MATLAB Usage `X = mot###_min(X, Y)`

Description This function returns a vector the same size as input vectors X and Y with the smallest elements taken from vector X or Y,

Input/Output Input: Vector X (real data), Vector Y (real data)

Output: Vector X (smallest element taken from X or Y)

Algorithm

```
#define IN1 x:$0
#define IN2 y:$0
#define OUT x:$0

MinValue = X[0];
for(i=1; i<size; i++)
{
if(X[i] > Y[i])
{
   X[i] = Y[i];
}
}
```

Memory & Register X memory IN1 start address of location of input real vector X

Y memory IN2 start address of location of input real vector Y

X memory OUT start address of location of output real vector

Register R7 stores the number of items in the input array

Status Register The assembly function min-2r.asm does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The total size of all vectors can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vectors must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 5 \times \Delta inputvectorlength = 5$$

DSP566:

$$\Delta\Delta cycle = 5 \times \Delta inputvectorlength = 5$$

MATLAB Usage `[ X ] = mot###_min( X, Y)`

Description This function returns the smallest elements taken from complex vector X or Y

Input/Output Input: Complex Vector X (include real part Xr and imaginary part Xi), Complex Vector Y (include real part Yr and imaginary part Yi)

Output: Complex Vector X (include real part Xr and imaginary part Xi)

Algorithm

```
for(i=0; i<size; i++)
{
   xValue = Xr[i]* Xr[i] + Xi[i]* Xi[i];
   yValue = Yr[i]* Yr[i] + Yi[i]* Yi[i];
   if(xValue > yValue)
       X[i] = Y[i];
}
```

Memory & Register Start address of Vector X is in P:$F000

Start address of Vector Y is in P:$F001

Start address of result Vector is also in P:$F000

Register R7 store Number of items in the array

Status Register The assembly function min-2c.asm does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The total size of all vectors can’t be larger than the continuous available data memory size.

Data Range Limit Input data vector range [–1.0, +1.0]

Precision In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

round-r.asm

MATLAB Usage `X = mot###_dspround(X)`

Description This function rounds the elements of input real vector X to the nearest integer

Input/Output Input: Vector X (real data)

Output: Vector X

Algorithm

```
for(i=0; i<size; i++)
{
   if (X[i] <= -0.5)
       X[i] = -1;
   else if(X[i] >= 0.5)
       X[i] = 1;
   else
       X[i] = 0;
}
```

Memory & Register X memory IN = start address of input data

X memory OUT = start address of result

Register R7 store length of input real vector X

Status Register The assembly function round-r.asm does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

MATLAB Usage

`[ X ] = mot###_dspround( X )`

Description

This function rounds the elements of complex input vector X to the nearest integer

Input/Output

Input: Complex vector X (includes the real part `Xr`, and the imaginary part `Xi`)

Output: `Xr`, `Xi`

Algorithm

```
for(i=0; i<size; i++)
{
    if (Xr[i] <= -0.5)
        Xr[i] = -1;
    else if(Xr[i] >= 0.5)
        Xr[i] = 1;
    else
        Xr[i] = 0;
    if (Xi[i] <= -0.5)
        Xi[i] = -1;
    else if(Xi[i] >= 0.5)
        Xi[i] = 1;
    else
        Xi[i] = 0;
}
```

Memory & Register:

Memory allocations:

X memory:

- Label IN_REAL stores the start address of the real input data
- Label OT_REAL stores the start address of the real result

Y memory

- Label IN_IMAG stores the start address of the imaginary input data
- Label OT_IMAG stores the start address of the imaginary result

Register usage:

- R7 stores the length of the input of the complex vector X

round-c.asm

Status Register The assembly function `round-c.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

MATLAB Usage `Y = mot###_sort(X)`

Description This function sorts real elements of input real vector X

Input/Output Input: Real vector X

Output: Real vector Y

Algorithm The heap sort algorithm will be used.

```
#define IN X:$0
#define OUT X:$0
heapsort(Length)
{
    for( i = Length/2; i>=1; --i)
        sift (i, Length);/* set up initial heap */
    for( k = Length; k >= 2; k--)
    {
        Intervalue = r[1];
        r[1] = r[k];
        r[k] = Intervalue;
        sift(1, k-1);
    }
}

sift (from, Length)
{
    Intervalue = r[from];
    k = from;
    j = 2 * from;
    while (j <= Length)
    {
    if ((j< Length) && (r[j] < r[j+1]))
        j++;
    if(Intervalue < r[j])
    {
        r[k] = r[j];
        k = j;
        j *= 2;
    }
```

```
        else
            j = Length + 1;/* return */
        r[k] = Intervalue;
    }
```

Memory & Register

Memory allocation:

X memory:

- Label IN stores the address of the real input of vector X
- Label OUT stores the address of the real output of vector Y

Register usage:

- R7stores the number of items in the array

Status Register

The assembly function `sort-1r.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit

The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

MATLAB Usage

`[Y, I] = mot###_sort(X)`

Description

This function sorts real elements of input real vector X

Input/Output

Input: Vector X

Output: Result vector Y, and index vector I

Algorithm

```
#define IN X:$0
#define OUT X:$0
#define INDEX Y:$0
```

The algorithm is the same as `sort-r1`, the only difference between these two functions is that `sort-r2` has one more output-index vector I.

Memory & Register

Memory allocation:

X memory:

- Label IN stores the address of the real input of vector X
- Label OUT stores the address of the real output of vector Y
  (Same as Label IN)
- Label INDEX stores the address of the index vector

Register usage:

- R7 store number of items in the array

Status Register

The assembly function `sort-2r.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit

The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit

The value of input vector X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

sort-c.asm

MATLAB Usage `[ Y, I] = mot###_sort( X )`

Description This function sorts the complex input vector X

Input/Output Input: Complex vector X (includes the real part `Xr`, and the imaginary part `Xi`)

Output: Complex vector Y, and vector I (index)

Algorithm

```
#define  IN       $0
#define  IN_REAL   x:$0
#define  IN_IMAG  y:$0
```

Algorithm

```
for(i = 0; i< size; i++)
{
    R[i] = (Xr[i]* Xr[i] + Xi[i]* Xi[i])>>1;
}
```

Sort vector R using heap sort algorithm, and then adjust vector X according to the index of vector R.

Generate vector R (and its index vector starting from 1), and then put the results in the address:

#IN+SIZE (X and Y memory, respectively)

Sort vector R and adjust its index vector accordingly.

Adjust the complex data according to the index.

Please refer to `sort-r1.asm` on page 3-91 for the heap sort algorithm.

Intermediate vector:

```
Vector R: R[i] = (Xr[i]*Xr[i] + Xi[i]*Xi[i])>>1
```

Start address of Vector R, and Vector I (index) will be calculated by the assembly code.

Vector I is located in Y memory.

Memory & Register Memory allocation:

- X memory: Label IN stores the address of the real input of vector X

- X memory label IN_REAL & Y memory IN_IMAG are used by the MEX function
- P:$F000 stores the start address of vector R
- P:$F001 stores the start address of vector I

Register usage:

- R7 stores the address of the number of items in the array

Status Register

The assembly function `sort-c.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit

The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit

Input data vector X range [–1.0, +1.0]

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

sqrt-sr.asm

MATLAB Usage `y = mot###_sqrt(x)`

Description This function calculates the square root of single input positive real data

Input/Output Input: x

Output: y

Algorithm Full 23(15) bit precision square root routine using a successive approximation technique.

```
y = 0;
guess = factor = 0.5;
for(i = 0; i < loopnum; i++)
{
  flag = x - guess*guess;
   if(flag >= 0)   y = guess;
   factor /= 2;
   guess = y + factor;
}
```

Memory & Register Register b = output root

a = temporary storage

x0 = guess

x1 = bit being tested

y0 = input number

r6 = loop number(23 for 56300, 15 for 56600)

Status Register The assembly function sqrt-sr.asm does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The input is a single value.

Data Range Limit The value of input X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

sqrt-pr.asm

MATLAB Usage `Y = mot###_sqrt(X)`

Description This function returns square root of the elements of input vector X

Input/Output Input: Vector X (elements of vector X are positive)

Output: Vector Y

Algorithm Full 23(15) bit precision square root routine using a successive approximation technique.

```
for (i=0; i<size; i++)
{
Y[i] = Sqrt_sr (X[i]);
}
```

Memory & Register Register usage:

- b is used for the output root
- a is used for temporary storage
- x0 is used for guess
- x1 is used for the bit being tested
- y0 is used for the input number
- r6 is used as the loop number (23 for 56300, 15 for 56600)

Status Register The assembly function `sqrt-pr.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

MATLAB Usage `[ Y ] = mot###_sqrt( X )`

Description This function returns the square root of the complex input vector

Input/Output Input: Vector X (includes the real part `Xr`, and the imaginary part `Xi`)

Output: Vector Y (includes the real part `Yr`, and the imaginary part `Yi`)

Algorithm

```
sqrt(a+bi) = [sqrt(sqrt(a^2+b^2))]*[cos(angle(a+bi)/
2)+i*sin(angle(a+bi)/2)]
```

To implement it in fix point dsp, we change this to

```
sqrt(a+bi) = sqrt(sqrt((a^2+b^2)/2))]*[cos(angle(a+bi)/
2)+i*sin(angle(a+bi)/2)] * [power(2,.25)]
```

For the ANGLE-C.ASM algorithm, please refer to `angle-c.asm` on page 3-14. The following describes how to calculate SIN and COS

Use the CORDIC algorithm

```
for (i=0; i < fracbits; i++)
    {
        atan_tab[i] = atan(pow(2,double(-i)))/4;
    }

         K = 0.30362641811371;
         X = x = K;
         Y = y = 0;
         Z = z = angle;

         Z = z = x>>2 ;

/* Circular Function */
         for (i = 0; i < fracbits; ++i)
        {
           x = X >> i;
           y = Y >> i;
           z = atan[i];

         if (Z >= 0)
        {
            X -= y;
            Y += x;
            Z -= z;
         }
          else
         {
            X += y;
            Y -= x;
            Z += z;
         }
       }
        X = X << 1;    //cos(angle)
        Y = Y << 1;    //sin(angle)

        return X and Y;
```

Memory & Register

Memory allocation:

X memory:

- Label IN stores the start address of the input of vector X
- Label IN_REAL stores the start address of the real data of the input vector
- Label IN_IMAG stores the start address of the complex (imaginary) data of the input vector
- Label OUTREAL stores the start address of the real data of the output vector

Y memory:

- Label OUTIMAG stores the start address of the imaginary data of the output vector
- Label XYZ stores the start address of variables `X,Y, i`

Register usage:

- X0 is x, as defined in CORDIC algorithm above
- X1 is y, as defined in CORDIC algorithm above
- Y1 is z, as defined in CORDIC algorithm above
- B is Z, as defined in CORDIC algorithm above
- R6 stores the address of the number of items in the input array

- R3 stores the address of the atan table,
- R0 and R5 are used for IN(OUT)
- R7is used for the start address of label XYZ
- R2 is used for number of fraction bits+1 (56300--22, 56600--14)
- R1 is used as the current loop number
- Y0 is mainly used as the shift number register (i.e., 'i' - see CORDIC algorithm above)
- N7 = 1. Use N7 rather than '1' to make a parallel move possible

Status Register The assembly function `sqrt-c.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can't be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 14 bits.

In the case of DSP566, precision is 15 bits.

MATLAB Usage `y = mot###_sum(X)`

Description This function returns the sum of the element of real input vector X

Input/Output Input: Vector X (real data)

Output: y (the sum of the element of X)

Algorithm

```
y = 0;
for(i=0; i<size; i++)
{
    y += X[i];
}
```

Memory & Register Memory allocation:

- X memory: IN stores the address of the input of real vector X.

Register usage:

- Register A is used for the result
- Register R7 is used for the number of items in the array

Status Register The assembly function `sum-r.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 1 \times \Delta inputvectorlength = 1$$

DSP566:

$$\Delta\Delta cycle = 1 \times \Delta inputvectorlength = 1$$

sum-c.asm

MATLAB Usage `[ y ] = mot###_sum( X )`

Description This function returns the sum of the complex input elements of X

Input/Output Input: Complex vector X (includes the real part `Xr`, and the imaginary part `Xi`)

Output: scalar y (includes the sum of the real data of the element `yr`, and the sum of the imaginary data of the element `yi`)

Algorithm

```
yr = yi = 0;
for(i=0; i<size; i++)
{
    yr += Xr[i];
    yi += Xi[i];
}
```

Memory & Resister Memory allocation:

- X memory label IN stores the start address of the input of real vector X.
- X memory label IN_REAL stores the start address of the real data of the input vector
- Y memory label IN_IMAG stores the start address of the complex (imaginary) data of the input vector

Register usage:

- Register A is used for the result of the real part
- Register B is used for the result of the imaginary part
- R7 is used for the number of items in the array

Status Register The assembly function `sum-c.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The length of vector X can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vector X must be between –1.0 and +1.0.

Precision

In the case of DSP563, precision is 23 bits.

In the case of DSP566, precision is 15 bits.

Performance Limit

DSP563:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

DSP566:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

xcorr-r.asm

MATLAB Usage `C=mot###_xcorr(A,B, MAXLAG, 'flag')`

Description This function returns the length 2*M-1 cross-correlation sequence in a column vector where A and B are length N real vectors

Input/Output Input: Vector A, vector B, int MAXLAG, and int 'flag'

Output: Vector C

Algorithm

```
N = LengthA;
//The following are algorithm of xcorr
    for (i = Max(1, (N – MAXLAG)); i <= N – 1; i ++) {
        for (j = 0; j < = i - 1; j ++) {
            C[i ] += B[j+1] * A[j + N – i + 1];
       }
        if (flag == “biased”) C[i] /= N;
        else if (flag == “unbiased”) C[i] /= i ;
    }
    for (i = N; i <= Min((2 * N – 1), (N + MAXLAG)); i ++) {
        for (j = 0; j <= 2 * N – i – 1; j ++) {
            C[i] += A[j+1] * B[j + i – N + 1];
       }
        if (flag == “biased”) C[i] /= N;
        else if (flag == “unbiased”) C[i] /= (2 * N – i);
    }
    if (flag == “coeff”) {
        for (i = Max(1, (N – MAXLAG)); i <= Min((2 * N – 1), (N +
MAXLAG)); i ++) {
            C[i] / = C[N];
       }
    }

    // Format vector C
    if (MAXLAG >= N) {
        LengthC = 2 * MAXLAG + 1;
        df = MAXLAG – N + 1;
        for (i = LengthC;  i >= 1 ; i --) {
            if ( i > df && i <= LengthC – df ) {
                C[i] = C[i - df];
           }
```

```
              else {
   C[i] = 0;
             }
         }
      }
      else {
          C = C(N – MAXLAG : LengthC);
      }
```

Memory & Register

Memory allocation:

In X memory:

- Label X:INA is used for input data of vector A
- Label X:INA+ LENGTH(A) is used for output data of vector C

In Y memory:

- Label Y:INB is used for the input data of vector B

Register usage:

- R2 stores the address of the length of vector A
- R3 = MAXLAG
- R7 = `'flag'`
- R7= 0 -- `'none'`
- R7 = 1 -- `'biased'`
- R7 = 4 -- `'coeff'`
- N7 = 24 for dsp56300, or 16 for dsp56600
- R1 stores the address of the index of vector C
- R0 stores the address of the index of vector A
- R4 stores the address of the index of vector B
- N7 is used for temporary storage of R7
- R7 is used for temporary storage of the address of vector A or B
- R5 and R6 are used as loop control

Assumes `M{0...7} = $ffffff`

Status Register The assembly function `xcorr-r.asm` does not explicitly set any status registers/ bits during the function execution.

Data Size Limit The total size of all vectors can’t be larger than the continuous available data memory size.

Data Range Limit The value of input vectors must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 18 bits.

In the case of DSP566, precision is 10 bits.

Perfumes Limit DSP563:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

DSP566:

$$\Delta\Delta cycle = 2 \times \Delta inputvectorlength = 2$$

MATLAB Usage `[C ]=mot###_xcorr( A, B, MAXLAG, 'flag')`

Description This function returns the length 2*M-1 cross-correlation sequence in a column vector where A and B are length N vectors.

Input/Output Input: Vector A (includes the real part `Ar`, and the imaginary part `Ai`), vector B (includes the real part `Br`, and the imaginary part `Bi`), `int MAXLAG`, and `int 'flag'`

Output: Vector C (includes the real part `Cr`, and the imaginary part `Ci`)

Algorithm

```
N = LengthA;
for (i = Max(1, (N – MAXLAG)); i <= N – 1; i ++) {
    for (j = 0; j < = i - 1; j ++) {
        C[i]+= B[j+1] * A[j+ N – i + 1]+ BI[j+1] * AI[j+ N – i +1];
        CI[i]+= BI[j+1] * A[j+ N –i +1]-B[j+1] * AI[j + N –i +1];
        }
        CI[i] = -CI[i];// complex conjugate
        if (flag == “biased”) C[i] /= N;
        else if (flag == “unbiased”) C[i] /= i ;
    }
    for (i = N; i <= Min((2 * N – 1), (N + MAXLAG)); i ++) {
        for (j = 0; j <= 2 * N – i – 1; j ++) {
            C[i] += A[j+1] * B[j+i– N+1]+ AI[j+1]* BI[j+i–N+1];
            CI[i]+= AI[j+1] * B[j+i– N+1]-A[j+1] * BI[j+i–N+1];
        }
        if (flag == “biased”) C[i] /= N;
        else if (flag == “unbiased”) C[i] /= (2 * N – i);
    }
    if (flag == “coeff”) {
       for(i=Max(1,(N–MAXLAG));i<= Min((2*N–1),(N+MAXLAG));i++){
            C[i]=C[i]*C[N]+CI[i]*CI[N])/C[N]*C[N]+CI[N]*CI[N];
C[i] = (CI[i]*C[N] – C[i]*CI[N]) / (C[N]* C[N] + CI[N] *CI[N];
       }
    }

// Format vector C
    if (MAXLAG >= N) {
        LengthC = 2 * MAXLAG + 1;
        df = MAXLAG – N + 1;
```

```
                for (i = LengthC;  i >= 1 ; i --) {
                    if ( i > df && i <= LengthC – df ) {
                        C[i] = C[i - df];
                        CI[i] = CI[i - df];
                   }
                    else {
    C[i] = 0;
    CI[i] = 0;
                   }
                }
            }
            else {
                C = C(N – MAXLAG : LengthC);
                CI = CI(N – MAXLAG : LengthC);
                }
```

Memory & Register

Memory allocation:

- X memory label X:INA stores the address of the input data of vector A
- Y memory label Y:INB stores the address of the input data of vector B
- X memory label X:INA+ 2*LENGTH(A) stores the output data of vector C

Register usage

- R2 stores the address of the length of vector A
- R3 = `MAXLAG`
- R7 = `'flag'`
- R7 = 0 -- `'none'`
- R7 = 1 -- `'biased'`
- R7 = 2 -- `'unbiased'`
- R7 = 4 -- `'coeff'`
- N7 = 24 for dsp56300 or 16 for dsp56600
- R1 stores the address of the index of vector C
- R0 stores the address of the index of vector A
- R4 stores the address of the index of vector B

- R7 stores the temporary address of vector A or B
- R5 and R6 are used as loop control

Assumes `M{0...7} = $ffffff`

Status Register The assembly function `xcorr-c.asm` does not explicitly set any status registers/bits during the function execution.

Data Size Limit The total size of all vectors can't be larger than the continuous available data memory size.

Data Range Limit The value of input vectors must be between –1.0 and +1.0.

Precision In the case of DSP563, precision is 18 bits.

In the case of DSP566, precision is 10 bits.

Performance Limit DSP563:

$$\Delta\Delta cycle = 16 \times \Delta inputvectorlength = 16$$

DSP566:

$$\Delta\Delta cycle = 16 \times \Delta inputvectorlength = 16$$

Motorola Toolbox Function Reference

Using This Reference Chapter

This chapter contains information on all Motorola DSP Developer's Kit Toolbox functions. You should turn to this chapter when you need to find information on a particular function.

The function reference entries appear in alphabetical order and each contains the following information:

- The function *name*, at the top of the page.
- The *purpose* of the function.
- A *description* of the function's use.
- The *characteristic* of the function which demo a example on how to use it.
- The *arguments* which indicates the tunable parameters.
- A *See Also* list of related blocks and functions.

Note These reference pages are also accessible online via the **help** command in the MATLAB workspace.

Multiple assembly functions exist for each provided Toolbox function to perform the necessary DSP algorithms. You can view the assembly source code in the `<matlab>/toolbox/motdspasm/src` directory.

Each toolbox function has an equivalent Simulink block accessible via the Simulink library interface (see "Motorola Blockset Block Reference" on page 5-1). The Simulink blocks use the same assembly source code to implement the functional algorithm.

Motorola 56300 Family ToolBox Functions

The function library within Motorola 56300 DSP Developer's Kit ToolBox contains all 21 Motorola 56300 DSP functions which vary from elementary math computations (mot563_abs, mot563_log), to frequency domain transforms (mot563_fft). Filtering design functions (mot563_filter, mot563_interp) are also supplied.

Table of Functions

Motorola 56300 DDK Toolbox	
mot563_abs	mot563_angle
mot563_conv	mot563_decimate
mot563_diff	mot563_dspround
mot563_fft	mot563_filter
mot563_ifft	mot563_interp
mot563_length	mot563_log
mot563_log10	mot563_max
mot563_mean	mot563_min
mot563_round	mot563_sort
mot563_sqrt	mot563_sum
mot563_xcorr	

Purpose

Output the absolute value of the input.

MATLAB Syntax

Y=mot563_abs(X)

Description

The `mot563_abs` function computes the absolute value of the input. When `X` is complex, `mot563_abs(X)` is the complex modulus (magnitude) of the elements of `X`.

Characteristics

In the workspace, if the input is:

```
X=[0.9501+0.2311j    0.6068-0.4860j]

Y=mot563_abs(X,'56301')
```

the answer returned by the function is:

```
Y=[0.9778    0.7774]
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Output vector in range [0, 1.414].

See Also

`mot563_log`   `mot563_log10`   `mot563_sqrt`

mot563_angle

Purpose

Compute the phase angle of a real- or complex-valued signal.

MATLAB Syntax

Y=mot563_angle(x)

Description

`mot563_angle` returns the phase angles, in radians, of every elements within a matrix. Input elements can be complex.

Characteristics

In the workspace, if the input is:

```
x=[-0.6813 - 0.6822i   0.5028 - 0.1509i   0.3046 - 0.8600i;
   -0.3795 + 0.3028i   0.7095 - 0.6979i   0.1897 + 0.8537i;
    0.8318 - 0.5417i  -0.4289 - 0.3784i   0.1934 - 0.5936i]

mot563_angle(x,'56301')
```

then the returned value is:

```
ans=-2.3555   -0.2916   -1.2304
     2.4681   -0.7772    1.3521
    -0.5772   -2.4187   -1.2558
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Phase angle in radians, ranges from $-\pi$ to π.

See Also

`mot563_log`   `mot563_log10`   `mot563_sqrt`

Purpose

Compute the convolution of two vectors.

MATLAB Syntax

Y=mot563_conv(A,B)

Description

`mot563_conv(A,B)` convolves vectors `A` and `B`.The length of the resulting vector is the length of `A` + the length of `B` -1. If `A` and `B` are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

Characteristics

In the workspace, if the input is:

```
A=[0.0695 + 0.0957i   0.0621 + 0.0523i   0.0795 + 0.0880i]
B=[0.0173 - 0.0252i   0.0980 - 0.0876i   0.0271 - 0.0737i]
mot563_conv(a,b,'56301')
```

then the returned value is:

```
ans=0.0036 - 0.0001i   0.0176 + 0.0026i   0.0232 -
0.0033i   0.0210 - 0.0015i   0.0086 - 0.0035i
```

Arguments

`A,B`

Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`

Resulting vector with length of `length(A)+length(B)`-1

Algorithm

For a length-M input vector u (indexed from 1 to M) and a length-N input vector v (indexed from 1 to N), the convolution output is a vector of length M+N-1 with elements

$$y(n) = \sum_{k=1}^{max(M,N)} u(k)v^*(n-k+1) \qquad 1 \le n \le M+N-1$$

where * denotes conjugation, and u and v are zero when indexed outside of their valid ranges.

When both inputs are real, the output is real as well. When one or both inputs are complex, the output is complex.

See Also

`mot563_xcorr`

mot563_decimate

Purpose Filter and down sample an input signal.

MATLAB Syntax

```
Y = mot563_decimate(X,R,N)
Y = mot563_decimate(X,R,'FIR')
Y = mot563_decimate(X,R,N,'FIR')
```

Description The `mot563_decimate` function resamples the input `X` at an integer rate R times slower than the input sample rate, where R is defined as Decimation factor parameter. This process consists of two steps:

- The function filters the input data with an FIR filter.
- The function downstages the filtered data to a lower rate.

The `mot563_decimate` function implements the FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than straightforward filter-then-decimate algorithms. The output of the decimator is the first filter phase.

`mot563_decimate` filters the data with an eighth order Chebyshev type I lowpass filter with cutoff frequency, `8*(Fs/2)/R`, before resampling.

`mot563_decimate(X,R,N)` uses an Nth order Chebyshev filter.

`mot563_decimate(X,R,'FIR')` uses the 30 point FIR filter generated by

`FIR1(30,1/R)` to filter the data.

`mot563_decimate(X,R,N,'FIR')` uses the N-point FIR filter.

NOTE: For large R, the Chebyshev filter design might be incorrect due to numeric precision limitations. In this case `mot563_decimate` will use a lower filter order. For better anti-aliasing performance, try breaking R up into its factors and calling `mot563_decimate` several times.

Characteristics In the workspace, if the input is:

```
X=[0.0682   0.0302   0.0541   0.0150   0.0697   0.0378   0.0860
   0.0853   0.0593   0.0496   0.0899   0.0821   0.0644   0.0818
   0.0660   0.0342   0.0289   0.0341   0.0534   0.0727]
R=3
N=8
mot563_decimate(X,R,N,’fir’)
```

then the returned value is:

```
ans=0.0682  0.0406  0.0705  0.0674  0.0756  0.0441  0.0541
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit). Length of `X` should keep larger than two times of `N+1`.

`R`

Decimation factor (integer) by which to decrease the sample rate of the input sequence.

`N`
Chebyshev filter order.

`FIR`
Option for “fir” or “iir” type.

`Y`
Resulting vector after FIR filtered and decimated.

See Also

`mot563_interp`

mot563_diff

Purpose Difference and approximate derivative.

MATLAB Syntax

```
Y=mot563_diff(X,N)
Y=mot563_diff(X,N,DIM)
```

Description For a vector `X`, `mot563_diff(X)`, is `[X(2)-X(1) X(3)-X(2)... X(n)-X(n-1)]`.

For a matrix `X`, `mot563_diff(X)`, is the matrix of column differences,

`[X(2:n,:) - X(1:n-1,:)]`.

`mot563_diff(X,N)` is the N-th order difference along the first non-singleton dimension (denote it by `DIM`). If `N >= size(X,DIM)`, `mot563_diff` takes successive differences along the next non-singleton dimension.

`mot563_diff(X,N,DIM)` is the Nth difference function along dimension `DIM`. If `N >= size(X,DIM)`, `DIFF` returns an empty array.

Characteristics If

```
X = [0.3 0.7 0.5
     0   0.9 0.2]
```

then `mot563_diff(X,1,1)` is `[-0.3 0.2 -0.3]`,

`mot563_diff(X,1,2)` is

```
[0.4 -0.2
 0.9 -0.7],
```

`mot563_diff(X,2,2)` is the 2nd order difference along the dimension 2, and `mot563_diff(X,3,2)` is the empty matrix.

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Resulting derivatives.

See Also `mot563_conv` `mot563_xcorr`

mot563_dspround

Purpose

Perform convergent rounding.

MATLAB Syntax

`Y=mot563_dspround10(X)`

Description

`mot563_dspround(X)` rounds the elements of `X` to the nearest integers using "convergent rounding" algorithm

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
mot563_dspround(X)
```

the returned value is:

```
ans= 0.4807        0.8319 - 0.2921i
     0 + 0.6717i   0.5273 + 0.7144i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Result after dsprounding.

See Also

`mot563_round`

mot563_fft

Purpose Compute the FFT of the input.

MATLAB Syntax

```
Y=mot563_fft(X)
Y=mot563_fft(X,N)
Y=mot563_fft(X,N,DIM)
```

Description `mot563_fft(X)` is the discrete Fourier transform (DFT) of vector `X`. If the length of `X` is not a power of two, it will be padded with zeros to make the length a power of two.

For matrices, the FFT operation is applied to each column.

`mot563_fft(X,N)` is the N-point FFT.

`mot563_fft(X,N,DIM)` applies the FFT operation across the dimension `DIM`.

Characteristics In the workspace, if the input is:

```
X = [-0.0023 - 0.0101i  -0.0446 - 0.0229i  -0.0634 - 0.0223i
     -0.0701 - 0.0234i  -0.1196 - 0.0103i  -0.0244 - 0.0014i]
```

then execute the function:

```
mot563_fft(X,2,2)
ans=-0.0469 - 0.0330i   0.0423 + 0.0128i
    -0.1897 - 0.0337i   0.0495 - 0.0131i
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Number of transform point.

`DIM`
Dimension of a matrix.

See Also `mot563_ifft`

Purpose

Apply one-dimensional digital filter to an input signal.

MATLAB Syntax

```
Y = mot563_filter(B,A,X)
[Y,Zf] = mot563_filter(B,A,X,Zi)
[Y,Zf] = mot563_filter(B,A,X,Zi,DIM)
```

Description

`mot563_filter(B,A,X)` Filters the data in vector `X` with the filter described by vectors `A` and `B` to create the filtered data `Y`. The Filter is a "Direct Form II Transposed" implementation of the standard difference equation:

```
    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
                  - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
```

If a(1) is not equal to 1, `mot563_filter` normalizes the Filter coefficients by a(1).

When `X` is a matrix, `mot563_filter` operates on the columns of `X`.

`[Y,Zf] = mot563_filter(B,A,X,Zi)` gives access to initial and final conditions, `Zi` and `Zf`, of the delays. Zi is a vector of length = `MAX (LENGTH(A), LENGTH(B))-1`.

`mot563_filter(B,A,X,[],DIM)` or `mot563_filter(B,A,X,Zi,DIM)` operates along the dimension `DIM`.

Characteristics

In the workspace, if the input is:

```
X=[0.6822  0.3028  0.5417  0.1509  0.6979  0.3784  0.8600]
A=[1 0.2500    0.1538   -0.2206   -0.0780   -0.0949   -0.1901
0.0743 ];
B=[1 0.2187    0.0031   -0.0705   -0.2378   -0.2488]
mot563_filter(B,A,X)
```

the returned value is:

```
ans=0.6822    0.2814    0.4348    0.2207    0.5423    0.2647
0.8196
```

Arguments

`X`

Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`B,A`

Vectors which consist of coefficients of Direct II filter respectively.

`Zi`
Vector representing initial condition, with a length of `mot563_max (LENGTH(A),LENGTH(B))-1`.

`Zf`
Vector representing final condition, similar to `Zi`.

`DIM`
Matrix dimension number.

See Also `mot563_sort`

Purpose

Compute the IFFT of the input.

MATLAB Syntax

```
Y=mot563_ifft(X)
Y=mot563_ifft(X,N)
Y=mot563_ifft(X,N,DIM)
```

Description

`mot563_ifft(X)` is the inverse discrete Fourier transform of `X`.

`mot563_ifft(X,N)` is the `N`-point inverse transform.

`mot563_ifft(X,N,DIM)` is the inverse discrete Fourier transform of `X` across the dimension `DIM`.

Characteristics

In the workspace, if the input is:

```
X = [-0.0023 - 0.0101i  -0.0446 - 0.0229i  -0.0634 - 0.0223i
     -0.0701 - 0.0234i  -0.1196 - 0.0103i  -0.0244 - 0.0014i]
```

then execute the function:

```
mot563_ifft(X,2,2)
ans=-0.0234 - 0.0165i   0.0211 + 0.0064i
    -0.0948 - 0.0168i   0.0248 - 0.0066i
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Scalar defining the number of inverse transform point.

`DIM`
Scalar defining the dimension when the inverse transform is performed across a matrix.

See Also

`mot563_fft`

mot563_interp

Purpose

Upsample and filter an input signal.

MATLAB Syntax

```
Y = mot563_interp(X,R,N)
Y = mot563_interp(X,R,L,ALPHA)
[Y,B] = mot563_interp(X,R,L,ALPHA)
```

Description

`mot563_interp(X,R)` resamples the sequence in vector `X` at `R` times the original sample rate. The resulting resampled vector `X` is `R` times longer, `mot563_length(Y) = R*mot563_length(X)`.

A symmetric filter, `B`, allows the original data to pass through unchanged and interpolates between so that the mean square error between them and their ideal values is minimized.

`mot563_interp(X,R,L,ALPHA)` allows specification of arguments L and `ALPHA` which otherwise default to 0.4 and 0.5 respectively. `2*L` is the number of original sample values used to perform the interpolation. Ideally L should be less than or equal to 10. The length of `B` is `2*L*R+1`. The signal is assumed to be band limited with cutoff frequency 0 < `ALPHA`<= 1.0.

`[Y,B] = mot563_interp(X,R,L,ALPHA)` returns the coefficients of the interpolation filter `B`.

Characteristics

In the workspace, if the input is:

```
X=[0.5641    0.4093    0.7053    0.6748    0.7565    0.4412
0.5435]
Alpha=0.34
R=2
L=3
[Y,B]=mot563_interp(X,R,L,Alpha)
```

the returned value is:

```
Y=[0.5641   0.4514   0.4093   0.5476   0.7053   0.7076   0.6748
   0.7392   0.7565   0.5978   0.4412   0.4552   0.5435   0.5966
B= -0.0000
    0.0162
         0
   -0.1095
    0.0000
    0.5934
   1.0000
   0.5934
   0.0000
  -0.1095
        0
   0.0162
  -0.0000
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit). Length of `X` should remain larger than two times `L+1`

`R`

Interpolation factor (integer) by which to increase the sample rate of the input sequence.

`ALPHA`
Cutoff frequency, normalized in [0,1].

`L`
Half the number of elements involved in interpolating.

`Y`
Interpolation result.

`B`
Coefficients of the interpolation filter.

See Also

`mot563_decimate`

mot563_length

Purpose

Return the number of elements in a vector.

MATLAB Syntax

```
Y=mot563_length(X)
```

Description

`mot563_length(X)` returns the length of vector `X`. It is equivalent to `MAX(SIZE(X))` for non-empty arrays and 0 for empty ones.

Characteristics

In the workspace, if the input is:

```
X=[0.3 0.7 0.5]
```

the returned value is:

```
motdsp_length(X)
ans=3
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Length of a vector or the number of column arrays for matrix input.

See Also

`mot563_conv` `mot563_xcorr`

Purpose

Perform a natural logarithm.

MATLAB Syntax

`Y=mot563_log(X)`

Description

`mot563_log(X)` returns the natural logarithm value of vector or matrix `X`. Complex results are produced if `X` is either negative or complex.

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
```

the returned value is:

```
mot563_log(X)
ans=-0.7325            -0.1259 - 0.3377i
    -0.3979 + 1.5708i  -0.1189 + 0.9350i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Natural logarithm of input.

See Also

`mot563_log10` `mot563_sqrt`

mot563_log10

Purpose Perform a common (Base 10) logarithm.

MATLAB Syntax

```
Y=mot563_log10(X)
```

Description `mot563_log10(X)` returns the common logarithm value of vector or matrix `X`. Complex results are produced if `X` is either negative or complex.

Characteristics In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
```

the returned value is:

```
mot563_log10(X)
ans= -0.3181             -0.0547 - 0.1467i
     -0.1728 + 0.6822i  -0.0516 + 0.4060i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Common (base 10) logarithm of input.

See Also `mot563_log` `mot563_sqrt`

Purpose

Find the maximum value and index of one or two input vector(s).

MATLAB Syntax

```
N=mot563_max(X,Y)
N=mot563_max(X,DIM)
[N,I]=mot563_max(X,DIM)
```

Description

For vectors, `mot563_max(X)` is the largest element in `X`. For matrices, `mot563_max(X)` is a row vector containing the maximum element from each column.

`[N,I] = mot563_max(X)` returns the indices of the maximum values in vector `I`.

If the values along the first non-singleton dimension contain more than one maximal element, the index of the first one is returned.

`mot563_max(X,Y)` returns an array the same size as `X` and `Y` with the largest elements taken from `X` or `Y`. Either one can be a scalar.

`[N,I] = mot563_max(X,DIM)` operates along the dimension `DIM`. When complex, the magnitude `mot563_max(mot563_abs(X))` is used.

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.0319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
Y=[-.1245 + 0.2242i    0.2623 - 0.7241i
   0.7800 + 0.1121i           -0.2324 ]
```

the returned value is:

```
N= mot563_max(X,Y)
N= 0.4807             0.2623 - 0.7241i
   0.7800 + 0.1121i   0.5273 + 0.7144i
```

Arguments

`X,Y`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Vector or matrix with the same size as input, which consists of maximum value in every position.

See Also

`mot563_min`

mot563_mean

Purpose Determine the mean value of the elements of a matrix along rows or columns.

MATLAB Syntax

```
Y=mot563_mean(X)
Y=mot563_mean(X,DIM)
```

Description For vectors, `mot563_mean(X)` is the mean of the elements of `X`. For matrices, `mot563_mean(X)` is a row vector with the mean over each column.

`mot563_mean(X,DIM)` means along the dimension `DIM`.

Characteristics In the workspace, if

```
X = [0   0.1  0.2
     0.3 0.4  0.1]
```

then

```
mot563_mean(X,1) = [0.1500   0.2500   0.1500]
mot563_mean(X,2) = [ 0.1000
                     0.2667]
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
The mean result of input.

See Also `mot563_sum`

Purpose

Find the minimum value and index of one or two input vector(s).

MATLAB Syntax

```
N=mot563_min(X,Y)
N=mot563_min(X,DIM)
[N,I]=mot563_min(X,DIM)
```

Description

For vectors, `mot563_min(X)` is the smallest element in `X`. For matrices, `mot563_min(X)` is a row vector containing the minimum element from each column.

`[N,I] = mot563_min(X)` returns the indices of the minimum values in vector `I`.

If the values along the first non-singleton dimension contain more than one minimal element, the index of the first one is returned.

`mot563_min(X,Y)` returns an array the same size as `X` and `Y` with the smallest elements taken from `X` or `Y`. Either one can be a scalar.

`[N,I] = mot563_min(X,DIM)` operates along the dimension `DIM`. When complex, the magnitude `mot563_min(mot563_abs(X))` is used.

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.0319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
Y=[-.1245 + 0.2242i   0.2623 - 0.7241i
   0.7800 + 0.1121i   -0.2324 ]
```

then the returned value is:

```
N= mot563_min(X,Y)
N=-0.1245 + 0.2242i   0.0319 - 0.2921i
   0 + 0.6717i        -0.2324
```

Arguments

`X,Y`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Vector or matrix with the same size as input, which consists of minimum value in every position.

See Also

`mot563_max`

mot563_round

Purpose Perform common mathematical rounding.

MATLAB Syntax

```
Y=mot563_round10(X)
```

Description `mot563_round(X)` rounds the elements of `X` to the nearest integers.

Characteristics In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
```

then the returned value is:

```
mot563_round(X)
ans= 0             1.0000
     0 + 1.0000i   1.0000 + 1.0000i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Result after rounding.

See Also `mot563_dspround`

Purpose

Sort the elements in a vector by value.

MATLAB Syntax

```
Y=mot563_sort(X)
Y=mot563_sort(X,DIM)
[Y,I]=mot563_sort(X)
```

Description

`mot563_sort` sorts in ascending order.

For vectors, `mot563_sort(X)` sorts the elements of `X` in ascending order.

For matrices, `mot563_sort(X)` sorts each column of `X` in ascending order.

`mot563_sort(X,DIM)` sorts along the dimension `DIM`.

`[Y,I]=mot563_sort(X)` also returns an index matrix I. If `X` is a vector, then `Y = X(I)`. If `X` is an m-by-n matrix, then

```
    for j = 1:n, Y(:,j) = X(I(:,j),j); end
```

When `X` is complex, the elements are sorted by `mot563_abs(X)`.

Characteristics

In the workspace, if the input is:

```
  x=[0.3 0.7 0.5
     0   0.4 0.2]
```

then the returned value is:

```
  motdsp_sort(X,1)
  ans=[0   0.4 0.2
       0.3 0.5 0.7]
  motdsp_sort(X,2)
  ans=[0.3 0.5 0.7
       0   0.2 0.4];
```

Arguments

`X`

Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`I`

Matrix index.

`Y`

Result vector or matrix sorted in ascending order.

See Also `mot563_max` `mot563_mean` `mot563_min`

Purpose

Perform a square root.

MATLAB Syntax

`Y=mot563_sqrt(X)`

Description

`mot563_sqrt(X)` is the square root of the elements of `X`. Complex results are produced if `X` is either positive or complex.

Characteristics

In the workspace, if the input is:

```
X=[0.2311    0.6068-0.4860i
   -0.4512   -.2324+0.7534i]
Y=mot563_sqrt(X)
```

the answer returned by the function is:

```
Y= 0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Square root result of input.

See Also

`mot563_log` `mot563_log10` `mot563_abs`

mot563_sum

Purpose Sum the elements of a matrix along rows or columns.

MATLAB Syntax

```
Y=mot563_sum(X)
Y=mot563_sum(X,DIM)
```

Description For vectors, `mot563_sum(X)` is the sum of the elements of `X`. For matrices, `mot563_sum(X)` is a row vector with the sum over each column.

`mot563_sum(X,DIM)` sums along the dimension `DIM`.

Characteristics In the workspace, if

```
X = [0   0.1 0.2
     0.3 0.4 0.1]
```

then

```
mot563_sum(X,1) = [0.3 0.5 0.3]
mot563_sum(X,2) = [ 0.3
                    0.8]
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
The sum result of input.

See Also `mot563_diff`

Purpose

Compute the correlation of two vectors.

MATLAB Syntax

```
Y=mot563_xcorr(A,B,MAXLAG,’flag’)
Y=mot563_xcorr(A,MAXLAG,’flag’)
```

Description

`mot563_xcorr` Cross-correlation function estimates `C=mot563_xcorr (A,B)`, where `A,B` are length `M` vectors (`M>1`), returns the length `2*M-1` cross-correlation sequence `C`. If `A,B` are vectors of different length, the shortest one is zero-padded. `C` will be a row vector if `A` is a row vector, and a column vector if `A` is a column vector.

`mot563_xcorr(A)`, where `A` is a vector, is the auto-correlation sequence. The zeroth lag of the output correlation is in the middle of the sequence, at element `M`.

`mot563_xcorr(A,B,MAXLAG)` computes the cross-correlation over the range of lags: `-MAXLAG` to `MAXLAG`, i.e., `2*MAXLAG+1` lags. `mot563_xcorr(A,MAXLAG)` computes the auto-correlation over the range of lags. If missing, the default is `MAXLAG` = `M-1`.

`[C,LAGS] = mot563_xcorr` returns a vector of lag indices (LAGS).

`mot563_xcorr(A,'flag'), mot563_xcorr(A,B,'flag')` or

`mot563_xcorr(A,B,MAXLAG,'flag')` normalizes the correlation according to `'flag'`:

- biased - scales the raw cross-correlation by `1/M`.
- unbiased - scales the raw correlation by `1/(M-motdsp_abs(lags))`
- coeff - normalizes the sequence so that the auto-correlations at zero lag are identically 1.0.
- none - no scaling (this is the default).

Characteristics

In the workspace, if the input is:

```
A=[0.0695 + 0.0957i   0.0621 + 0.0523i   0.0795 + 0.0880i]
B=[0.0173 - 0.0252i   0.0980 - 0.0876i   0.0271 - 0.0737i]
MAXLAG=5
flag=’coeff’
mot563_xcorr(A,B,MAXLAG,flag,'56301')
```

the returned value is:

```
ans=   0     0     0     -0.0290 - 0.1213i     -0.0056 - 0.6214i
-0.1389 - 0.7645i   -0.1288 - 0.7385i   -0.1779 - 0.2655i
   0     0     0
```

Arguments

`A,B`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`MAXLAG`
Scalar to define the maximum range of lags. If missing, the default value is `M`-1.

`flag`
A character which determine the correlation type. See “Description” part.

`Y`

Resulting vector.

See Also

`mot563_conv`

Motorola 56600 Family ToolBox Functions

The function library within Motorola 56600 DSP Developer's Kit ToolBox contains all 21 Motorola 56600 DSP functions which vary from elementary math computations (mot566_abs, mot566_log), to frequency domain transforms (mot566_fft). Filtering design functions (mot566_filter, mot566_interp) are also supplied.

Table of Functions

Motorola 56600 DDK Toolbox	
`mot566_abs`	`mot566_angle`
`mot566_conv`	`mot566_decimate`
`mot566_diff`	`mot566_dspround`
`mot566_fft`	`mot566_filter`
`mot566_ifft`	`mot566_interp`
`mot566_length`	`mot566_log`
`mot566_log10`	`mot566_max`
`mot566_mean`	`mot566_min`
`mot566_round`	`mot566_sort`
`mot566_sqrt`	`mot566_sum`
`mot566_xcorr`	

Purpose Output the absolute value of the input.

MATLAB Syntax Y=mot566_abs(X)

Description The `mot566_abs` function computes the absolute value of the input. When `X` is complex, `mot566_abs(X)` is the complex modulus (magnitude) of the elements of `X`.

Characteristics In the workspace, if the input is:

```
X=[0.9501+0.2311j    0.6068-0.4860j]

Y=mot566_abs(X,'56601')
```

the answer returned by the function is:

```
Y=[0.9778    0.7774]
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Output vector in range [0, 1.414].

See Also `mot566_log` `mot566_log10` `mot566_sqrt`

mot566_angle

Purpose

Compute the phase angle of a real- or complex-valued signal.

MATLAB Syntax

Y=mot566_angle(x)

Description

`mot566_angle` returns the phase angles, in radians, of every elements within a matrix. Input elements can be complex.

Characteristics

In the workspace, if the input is:

```
x=[-0.6813 - 0.6822i   0.5028 - 0.1509i   0.3046 - 0.8600i;
   -0.3795 + 0.3028i   0.7095 - 0.6979i   0.1897 + 0.8537i;
    0.8318 - 0.5417i  -0.4289 - 0.3784i   0.1934 - 0.5936i]

mot566_angle(x,'56601')
```

then the returned value is:

```
ans=-2.3555   -0.2916   -1.2304
     2.4681   -0.7772    1.3521
    -0.5772   -2.4187   -1.2558
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Phase angle in radians, ranges from $-\pi$ to π.

See Also

`mot566_log`   `mot566_log10`   `mot566_sqrt`

Purpose Compute the convolution of two vectors.

MATLAB Syntax Y=mot566_conv(A,B)

Description `mot566_conv(A,B)` convolves vectors `A` and `B`.The length of the resulting vector is the length of `A` + the length of `B` -1. If `A` and `B` are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

Characteristics In the workspace, if the input is:

```
A=[0.0695 + 0.0957i   0.0621 + 0.0523i   0.0795 + 0.0880i]
B=[0.0173 - 0.0252i   0.0980 - 0.0876i   0.0271 - 0.0737i]
mot566_conv(a,b,'56601')
```

then the returned value is:

```
ans=0.0036 - 0.0001i   0.0176 + 0.0026i   0.0232 -
0.0033i   0.0210 - 0.0015i   0.0086 - 0.0035i
```

Arguments `A,B`

Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`

Resulting vector with length of `length(A)+length(B)`-1

Algorithm For a length-M input vector u (indexed from 1 to M) and a length-N input vector v (indexed from 1 to N), the convolution output is a vector of length M+N-1 with elements

$$y(n) = \sum_{k=1}^{max(M,N)} u(k)v^*(n-k+1) \qquad 1 \le n \le M+N-1$$

where * denotes conjugation, and u and v are zero when indexed outside of their valid ranges.

When both inputs are real, the output is real as well. When one or both inputs are complex, the output is complex.

See Also `mot566_xcorr`

mot566_decimate

Purpose

Filter and down sample an input signal.

MATLAB Syntax

```
Y = mot566_decimate(X,R,N)
Y = mot566_decimate(X,R,'FIR')
Y = mot566_decimate(X,R,N,'FIR')
```

Description

The `mot566_decimate` function resamples the input `X` at an integer rate R times slower than the input sample rate, where R is defined as Decimation factor parameter. This process consists of two steps:

- The function filters the input data with an FIR filter.
- The function downstages the filtered data to a lower rate.

The `mot566_decimate` function implements the FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than straightforward filter-then-decimate algorithms. The output of the decimator is the first filter phase.

`mot566_decimate` filters the data with an eighth order Chebyshev type I lowpass filter with cutoff frequency, `8*(Fs/2)/R`, before resampling.

`mot566_decimate(X,R,N)` uses an Nth order Chebyshev filter.

`mot566_decimate(X,R,'FIR')` uses the 30 point FIR filter generated by

`FIR1(30,1/R)` to filter the data.

`mot566_decimate(X,R,N,'FIR')` uses the N-point FIR filter.

NOTE: For large R, the Chebyshev filter design might be incorrect due to numeric precision limitations. In this case `mot566_decimate` will use a lower filter order. For better anti-aliasing performance, try breaking R up into its factors and calling `mot566_decimate` several times.

Characteristics

In the workspace, if the input is:

```
X=[0.0682   0.0302   0.0541   0.0150   0.0697   0.0378   0.0860
   0.0853   0.0593   0.0496   0.0899   0.0821   0.0644   0.0818
   0.0660   0.0342   0.0289   0.0341   0.0534   0.0727]
R=3
N=8
mot566_decimate(X,R,N,’fir’)
```

then the returned value is:

```
ans=0.0682  0.0406  0.0705  0.0674  0.0756  0.0441  0.0541
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit). Length of `X` should keep larger than two times of `N+1`.

`R`

Decimation factor (integer) by which to decrease the sample rate of the input sequence.

`N`
Chebyshev filter order.

`FIR`
Option for “fir” or “iir” type.

`Y`
Resulting vector after FIR filtered and decimated.

See Also

`mot566_interp`

mot566_diff

Purpose Difference and approximate derivative.

MATLAB Syntax

```
Y=mot566_diff(X,N)
Y=mot566_diff(X,N,DIM)
```

Description For a vector `X`, `mot566_diff(X)`, is `[X(2)-X(1) X(3)-X(2)... X(n)-X(n-1)]`.

For a matrix `X`, `mot566_diff(X)`, is the matrix of column differences,

```
  [X(2:n,:) - X(1:n-1,:)].
```

`mot566_diff(X,N)` is the N-th order difference along the first non-singleton dimension (denote it by `DIM`). If `N >= size(X,DIM)`, `mot566_diff` takes successive differences along the next non-singleton dimension.

`mot566_diff(X,N,DIM)` is the Nth difference function along dimension `DIM`. If `N >= size(X,DIM)`, `DIFF` returns an empty array.

Characteristics If

```
X = [0.3 0.7 0.5
     0   0.9 0.2]
```

then `mot566_diff(X,1,1)` is `[-0.3 0.2 -0.3]`,

`mot566_diff(X,1,2)` is

```
[0.4 -0.2
 0.9 -0.7],
```

`mot566_diff(X,2,2)` is the 2nd order difference along the dimension 2, and `mot566_diff(X,3,2)` is the empty matrix.

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Resulting derivatives.

See Also `mot566_conv` `mot566_xcorr`

Purpose

Perform convergent rounding.

MATLAB Syntax

`Y=mot566_dspround10(X)`

Description

`mot566_dspround(X)` rounds the elements of `X` to the nearest integers using "convergent rounding" algorithm

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
mot566_dspround(X)
```

the returned value is:

```
ans= 0.4807        0.8319 - 0.2921i
     0 + 0.6717i   0.5273 + 0.7144i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Result after dsprounding.

See Also

`mot566_round`

mot566_fft

Purpose Compute the FFT of the input.

MATLAB Syntax

```
Y=mot566_fft(X)
Y=mot566_fft(X,N)
Y=mot566_fft(X,N,DIM)
```

Description `mot566_fft(X)` is the discrete Fourier transform (DFT) of vector `X`. If the length of `X` is not a power of two, it will be padded with zeros to make the length a power of two.

For matrices, the FFT operation is applied to each column.

`mot566_fft(X,N)` is the N-point FFT.

`mot566_fft(X,N,DIM)` applies the FFT operation across the dimension `DIM`.

Characteristics In the workspace, if the input is:

```
X = [-0.0023 - 0.0101i  -0.0446 - 0.0229i  -0.0634 - 0.0223i
     -0.0701 - 0.0234i  -0.1196 - 0.0103i  -0.0244 - 0.0014i]
```

then execute the function:

```
mot566_fft(X,2,2)
ans=-0.0469 - 0.0330i   0.0423 + 0.0128i
    -0.1897 - 0.0337i   0.0495 - 0.0131i
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Number of transform point.

`DIM`
Dimension of a matrix.

See Also `mot566_ifft`

Purpose

Apply one-dimensional digital filter to an input signal.

MATLAB Syntax

```
Y = mot566_filter(B,A,X)
[Y,Zf] = mot566_filter(B,A,X,Zi)
[Y,Zf] = mot566_filter(B,A,X,Zi,DIM)
```

Description

`mot566_filter(B,A,X)` Filters the data in vector `X` with the filter described by vectors `A` and `B` to create the filtered data `Y`. The Filter is a "Direct Form II Transposed" implementation of the standard difference equation:

```
    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
                - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
```

If a(1) is not equal to 1, `mot566_filter` normalizes the Filter coefficients by a(1).

When `X` is a matrix, `mot566_filter` operates on the columns of `X`.

`[Y,Zf] = mot566_filter(B,A,X,Zi)` gives access to initial and final conditions, `Zi` and `Zf`, of the delays. Zi is a vector of length = `MAX (LENGTH(A), LENGTH(B))-1`.

`mot566_filter(B,A,X,[],DIM)` or `mot566_filter(B,A,X,Zi,DIM)` operates along the dimension `DIM`.

Characteristics

In the workspace, if the input is:

```
X=[0.6822  0.3028  0.5417  0.1509  0.6979  0.3784  0.8600]
A=[1 0.2500    0.1538   -0.2206   -0.0780   -0.0949   -0.1901
0.0743 ];
B=[1 0.2187    0.0031   -0.0705   -0.2378   -0.2488]
mot566_filter(B,A,X)
```

the returned value is:

```
ans=0.6822    0.2814    0.4348    0.2207    0.5423    0.2647
0.8196
```

Arguments

`X`

Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`B,A`

Vectors which consist of coefficients of Direct II filter respectively.

`Zi`
Vector representing initial condition, with a length of `mot566_max (LENGTH(A),LENGTH(B))-1`.

`Zf`
Vector representing final condition, similar to `Zi`.

`DIM`
Matrix dimension number.

See Also `mot566_sort`

mot566_ifft

Purpose

Compute the IFFT of the input.

MATLAB Syntax

```
Y=mot566_ifft(X)
Y=mot566_ifft(X,N)
Y=mot566_ifft(X,N,DIM)
```

Description

`mot566_ifft(X)` is the inverse discrete Fourier transform of `X`.

`mot566_ifft(X,N)` is the `N`-point inverse transform.

`mot566_ifft(X,N,DIM)` is the inverse discrete Fourier transform of `X` across the dimension `DIM`.

Characteristics

In the workspace, if the input is:

```
X = [-0.0023 - 0.0101i  -0.0446 - 0.0229i  -0.0634 - 0.0223i
     -0.0701 - 0.0234i  -0.1196 - 0.0103i  -0.0244 - 0.0014i]
```

then execute the function:

```
mot566_ifft(X,2,2)
ans=-0.0234 - 0.0165i   0.0211 + 0.0064i
    -0.0948 - 0.0168i   0.0248 - 0.0066i
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Scalar defining the number of inverse transform point.

`DIM`
Scalar defining the dimension when the inverse transform is performed across a matrix.

See Also

`mot566_fft`

mot566_interp

Purpose Upsample and filter an input signal.

MATLAB Syntax

```
Y = mot566_interp(X,R,N)
Y = mot566_interp(X,R,L,ALPHA)
[Y,B] = mot566_interp(X,R,L,ALPHA)
```

Description `mot566_interp(X,R)` resamples the sequence in vector `X` at `R` times the original sample rate. The resulting resampled vector `X` is `R` times longer, `mot566_length(Y) = R*mot566_length(X)`.

A symmetric filter, `B`, allows the original data to pass through unchanged and interpolates between so that the mean square error between them and their ideal values is minimized.

`mot566_interp(X,R,L,ALPHA)` allows specification of arguments L and `ALPHA` which otherwise default to 0.4 and 0.5 respectively. `2*L` is the number of original sample values used to perform the interpolation. Ideally L should be less than or equal to 10. The length of `B` is `2*L*R+1`. The signal is assumed to be band limited with cutoff frequency 0 < `ALPHA`<= 1.0.

`[Y,B] = mot566_interp(X,R,L,ALPHA)` returns the coefficients of the interpolation filter `B`.

Characteristics In the workspace, if the input is:

```
X=[0.5641    0.4093    0.7053    0.6748    0.7565    0.4412
0.5435]
Alpha=0.34
R=2
L=3
[Y,B]=mot566_interp(X,R,L,Alpha)
```

the returned value is:

```
Y=[0.5641   0.4514   0.4093   0.5476   0.7053   0.7076   0.6748
   0.7392   0.7565   0.5978   0.4412   0.4552   0.5435   0.5966
B= -0.0000
    0.0162
         0
   -0.1095
    0.0000
    0.5934
```

```
             1.0000
             0.5934
             0.0000
            -0.1095
                  0
             0.0162
            -0.0000
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit). Length of `X` should remain larger than two times `L+1`

`R`

Interpolation factor (integer) by which to increase the sample rate of the input sequence.

`ALPHA`
Cutoff frequency, normalized in [0,1].

`L`
Half the number of elements involved in interpolating.

`Y`
Interpolation result.

`B`
Coefficients of the interpolation filter.

See Also

`mot566_decimate`

mot566_length

Purpose

Return the number of elements in a vector.

MATLAB Syntax

```
Y=mot566_length(X)
```

Description

`mot566_length(X)` returns the length of vector `X`. It is equivalent to `MAX(SIZE(X))` for non-empty arrays and 0 for empty ones.

Characteristics

In the workspace, if the input is:

```
   X=[0.3 0.7 0.5]
```

the returned value is:

```
   motdsp_length(X)
   ans=3
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Length of a vector or the number of column arrays for matrix input.

See Also

`mot566_conv` `mot566_xcorr`

Purpose

Perform a natural logarithm.

MATLAB Syntax

```
Y=mot566_log(X)
```

Description

`mot566_log(X)` returns the natural logarithm value of vector or matrix `X`. Complex results are produced if `X` is either negative or complex.

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
```

the returned value is:

```
mot566_log(X)
ans=-0.7325            -0.1259 - 0.3377i
    -0.3979 + 1.5708i  -0.1189 + 0.9350i
```

Arguments

`X`

Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`

Natural logarithm of input.

See Also

`mot566_log10`   `mot566_sqrt`

mot566_log10

Purpose Perform a common (Base 10) logarithm.

MATLAB Syntax

```
Y=mot566_log10(X)
```

Description `mot566_log10(X)` returns the common logarithm value of vector or matrix `X`. Complex results are produced if `X` is either negative or complex.

Characteristics In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
```

the returned value is:

```
mot566_log10(X)
ans= -0.3181            -0.0547 - 0.1467i
     -0.1728 + 0.6822i  -0.0516 + 0.4060i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Common (base 10) logarithm of input.

See Also `mot566_log` `mot566_sqrt`

Purpose

Find the maximum value and index of one or two input vector(s).

MATLAB Syntax

```
N=mot566_max(X,Y)
N=mot566_max(X,DIM)
[N,I]=mot566_max(X,DIM)
```

Description

For vectors, `mot566_max(X)` is the largest element in `X`. For matrices, `mot566_max(X)` is a row vector containing the maximum element from each column.

`[N,I] = mot566_max(X)` returns the indices of the maximum values in vector `I`.

If the values along the first non-singleton dimension contain more than one maximal element, the index of the first one is returned.

`mot566_max(X,Y)` returns an array the same size as `X` and `Y` with the largest elements taken from `X` or `Y`. Either one can be a scalar.

`[N,I] = mot566_max(X,DIM)` operates along the dimension `DIM`. When complex, the magnitude `mot566_max(mot566_abs(X))` is used.

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.0319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
Y=[-.1245 + 0.2242i    0.2623 - 0.7241i
   0.7800 + 0.1121i           -0.2324 ]
```

the returned value is:

```
N= mot566_max(X,Y)
N= 0.4807             0.2623 - 0.7241i
   0.7800 + 0.1121i   0.5273 + 0.7144i
```

Arguments

`X,Y`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Vector or matrix with the same size as input, which consists of maximum value in every position.

See Also

`mot566_min`

mot566_mean

Purpose

Determine the mean value of the elements of a matrix along rows or columns.

MATLAB Syntax

```
Y=mot566_mean(X)
Y=mot566_mean(X,DIM)
```

Description

For vectors, `mot566_mean(X)` is the mean of the elements of `X`. For matrices, `mot566_mean(X)` is a row vector with the mean over each column.

`mot566_mean(X,DIM)` means along the dimension `DIM`.

Characteristics

In the workspace, if

```
X = [0   0.1  0.2
     0.3 0.4  0.1]
```

then

```
mot566_mean(X,1) = [0.1500   0.2500   0.1500]
mot566_mean(X,2) = [ 0.1000
                     0.2667]
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
The mean result of input.

See Also

`mot566_sum`

Purpose

Find the minimum value and index of one or two input vector(s).

MATLAB Syntax

```
N=mot566_min(X,Y)
N=mot566_min(X,DIM)
[N,I]=mot566_min(X,DIM)
```

Description

For vectors, `mot566_min(X)` is the smallest element in `X`. For matrices, `mot566_min(X)` is a row vector containing the minimum element from each column.

`[N,I] = mot566_min(X)` returns the indices of the minimum values in vector `I`.

If the values along the first non-singleton dimension contain more than one minimal element, the index of the first one is returned.

`mot566_min(X,Y)` returns an array the same size as `X` and `Y` with the smallest elements taken from `X` or `Y`. Either one can be a scalar.

`[N,I] = mot566_min(X,DIM)` operates along the dimension `DIM`. When complex, the magnitude `mot566_min(mot566_abs(X))` is used.

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.0319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
Y=[-.1245 + 0.2242i   0.2623 - 0.7241i
   0.7800 + 0.1121i   -0.2324 ]
```

then the returned value is:

```
N= mot566_min(X,Y)
N=-0.1245 + 0.2242i   0.0319 - 0.2921i
   0 + 0.6717i        -0.2324
```

Arguments

`X,Y`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`N`
Vector or matrix with the same size as input, which consists of minimum value in every position.

See Also

`mot566_max`

mot566_round

Purpose

Perform common mathematical rounding.

MATLAB Syntax

```
Y=mot566_round10(X)
```

Description

`mot566_round(X)` rounds the elements of `X` to the nearest integers.

Characteristics

In the workspace, if the input is:

```
X=[0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i]
```

then the returned value is:

```
mot566_round(X)
ans= 0             1.0000
     0 + 1.0000i   1.0000 + 1.0000i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Result after rounding.

See Also

`mot566_dspround`

Purpose

Sort the elements in a vector by value.

MATLAB Syntax

```
Y=mot566_sort(X)
Y=mot566_sort(X,DIM)
[Y,I]=mot566_sort(X)
```

Description

`mot566_sort` sorts in ascending order.

For vectors, `mot566_sort(X)` sorts the elements of `X` in ascending order.

For matrices, `mot566_sort(X)` sorts each column of `X` in ascending order.

`mot566_sort(X,DIM)` sorts along the dimension `DIM`.

`[Y,I]=mot566_sort(X)` also returns an index matrix I. If `X` is a vector, then `Y = X(I)`. If `X` is an m-by-n matrix, then

```
    for j = 1:n, Y(:,j) = X(I(:,j),j); end
```

When `X` is complex, the elements are sorted by `mot566_abs(X)`.

Characteristics

In the workspace, if the input is:

```
x=[0.3 0.7 0.5
   0   0.4 0.2]
```

then the returned value is:

```
motdsp_sort(X,1)
ans=[0   0.4 0.2
     0.3 0.5 0.7]
motdsp_sort(X,2)
ans=[0.3 0.5 0.7
     0   0.2 0.4];
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`I`
Matrix index.

`Y`
Result vector or matrix sorted in ascending order.

See Also mot566_max mot566_mean mot566_min

Purpose

Perform a square root.

MATLAB Syntax

```
Y=mot566_sqrt(X)
```

Description

`mot566_sqrt(X)` is the square root of the elements of `X`. Complex results are produced if `X` is either positive or complex.

Characteristics

In the workspace, if the input is:

```
X=[0.2311    0.6068-0.4860i
   -0.4512   -.2324+0.7534i]
Y=mot566_sqrt(X)
```

the answer returned by the function is:

```
Y= 0.4807 + 0.0000i   0.8319 - 0.2921i
   0.0000 + 0.6717i   0.5273 + 0.7144i
```

Arguments

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
Square root result of input.

See Also

`mot566_log`   `mot566_log10`   `mot566_abs`

mot566_sum

Purpose Sum the elements of a matrix along rows or columns.

MATLAB Syntax

```
Y=mot566_sum(X)
Y=mot566_sum(X,DIM)
```

Description For vectors, `mot566_sum(X)` is the sum of the elements of `X`. For matrices, `mot566_sum(X)` is a row vector with the sum over each column.

`mot566_sum(X,DIM)` sums along the dimension `DIM`.

Characteristics In the workspace, if

```
X = [0   0.1 0.2
     0.3 0.4 0.1]
```

then

```
mot566_sum(X,1) = [0.3 0.5 0.3]
mot566_sum(X,2) = [ 0.3
                    0.8]
```

Argument

`X`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`Y`
The sum result of input.

See Also `mot566_diff`

Purpose

Compute the correlation of two vectors.

MATLAB Syntax

```
Y=mot566_xcorr(A,B,MAXLAG,'flag')
Y=mot566_xcorr(A,MAXLAG,'flag')
```

Description

`mot566_xcorr` Cross-correlation function estimates `C=mot566_xcorr (A,B)`, where `A,B` are length `M` vectors (`M>1`), returns the length `2*M-1` cross-correlation sequence `C`. If `A,B` are vectors of different length, the shortest one is zero-padded. `C` will be a row vector if `A` is a row vector, and a column vector if `A` is a column vector.

`mot566_xcorr(A)`, where `A` is a vector, is the auto-correlation sequence. The zeroth lag of the output correlation is in the middle of the sequence, at element `M`.

`mot566_xcorr(A,B,MAXLAG)` computes the cross-correlation over the range of lags: `-MAXLAG` to `MAXLAG`, i.e., `2*MAXLAG+1` lags. `mot566_xcorr(A,MAXLAG)` computes the auto-correlation over the range of lags. If missing, the default is `MAXLAG` = `M`-1.

`[C,LAGS] = mot566_xcorr` returns a vector of lag indices (LAGS).

`mot566_xcorr(A,'flag'), mot566_xcorr(A,B,'flag')` or

`mot566_xcorr(A,B,MAXLAG,'flag')` normalizes the correlation according to `'flag'`:

- biased - scales the raw cross-correlation by `1/M`.
- unbiased - scales the raw correlation by `1/(M-motdsp_abs(lags))`
- coeff - normalizes the sequence so that the auto-correlations at zero lag are identically 1.0.
- none - no scaling (this is the default).

Characteristics

In the workspace, if the input is:

```
A=[0.0695 + 0.0957i   0.0621 + 0.0523i   0.0795 + 0.0880i]
B=[0.0173 - 0.0252i   0.0980 - 0.0876i   0.0271 - 0.0737i]
MAXLAG=5
flag='coeff'
mot566_xcorr(A,B,MAXLAG,flag,'56601')
```

the returned value is:

```
ans=   0     0     0     -0.0290 - 0.1213i     -0.0056 - 0.6214i
-0.1389 - 0.7645i   -0.1288 - 0.7385i   -0.1779 - 0.2655i
   0     0     0
```

Arguments

`A,B`
Vector (or matrix) with elements normalized in the range [-1,1] (for complex elements, both real and imaginary parts should comply with this limit).

`MAXLAG`
Scalar to define the maximum range of lags. If missing, the default value is `M-1`.

`flag`
A character which determine the correlation type. See “Description” part.

`Y`

Resulting vector.

See Also

`mot566_conv`

5

Motorola Blockset Block Reference

Using This Reference Chapter

This chapter contains information on every block implemented in the Motorola DSP Developer's Kit Blocksets. You should turn to this chapter when you need to find information on a particular block. Each Simulink blockset function has an equivalent MEX-file toolbox function accessible via the MATLAB command line (see "Motorola Toolbox Function Reference" on page 4-1).

The block reference entries appear in alphabetical order and each contains the following information:

- The block *name*, at the top of the page.
- The *purpose* of the block.
- The *library* or libraries where the block can be found.
- A *description* of the block's use.
- The block's *parameters and dialog box*.
- A *See Also* list of related blocks and functions.

Note These reference pages are also available online via the **Help** button in each block's dialog box. For more information on how to access the online help pages see "How to Get Help Online" on page 1-4.

The sections in this reference are:

- "Motorola 56300 Family Blockset" on page 5-3
- "Motorola 56600 Family Blockset" on page -57

Motorola 56300 Family Blockset

The block library within the Motorola 56300 DSP Developer's Kit Blockset contains 21 Motorola 56300 DSP blocks which vary from elementary math computations (MOTDSP563 Abs, MOTDSP563 Log), to frequency domain transforms (MOTDSP563 FFT). Filtering design blocks are also supplied.

Motorola 56300 DDK Blocks Listed by Category

Motorola 56300 DDK Blockset	
MOTDSP563 Abs	MOTDSP563 Angle
MOTDSP563 Convolution	MOTDSP563 FIR Decimation
MOTDSP563 Difference	MOTDSP563 DSP Rounding
MOTDSP563 FFT	MOTDSP563 Direct-Form II Transpose Filter
MOTDSP563 IFFT	MOTDSP563 FIR Interpolation
MOTDSP563 Length	MOTDSP563 Log
MOTDSP563 Log10	MOTDSP563 Maximum
MOTDSP563 Matrix Mean	MOTDSP563 Minimum
MOTDSP563 Rounding	MOTDSP563 Sort
MOTDSP563 Sqrt	MOTDSP563 Matrix Sum
MOTDSP563 Correlation	

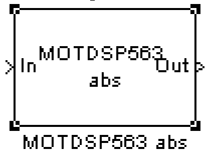
Purpose

Output the absolute value of the input.

Library

`Motdsp563lib.`

Description

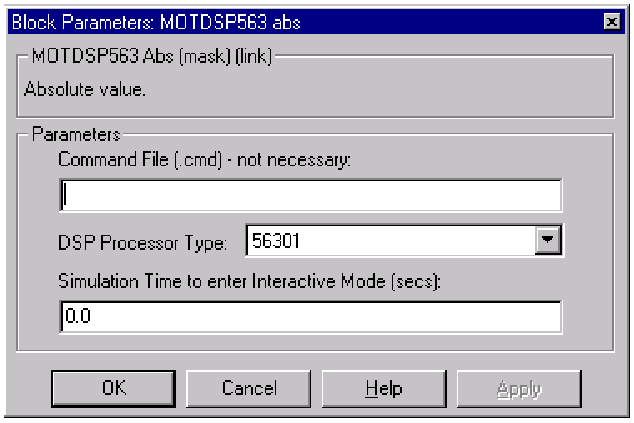


The MOTDSP563 Abs block generates as output the absolute value of the input.

Data Type Support

The MOTDSP563 Abs block accepts a real- or complex-valued input of type double and generates a real output of type double.

Parameters and Dialog Box



Block Parameters: MOTDSP563 abs

MOTDSP563 Abs (mask) (link)

Absolute value.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

MOTPurpose

See Also `MOTDSP563 Log`, `MOTDSP563 Log10`, `MOTDSP563 Sqrt`

Purpose

Compute the phase angle of a real- or complex-valued signal.

Library

Motdsp563lib.

Description

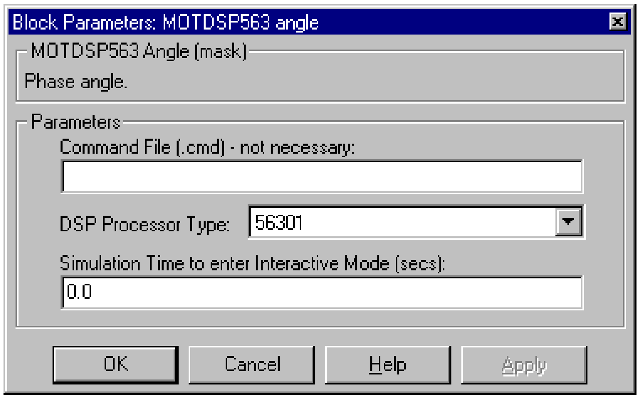
In
MOTDSP563
angle
Out
MOTDSP563 angle

The MOTDSP563 Angle block accepts a real- or complex-valued signal of type double. It outputs the phase angle of the input signal. The outputs are real values of type double. The input may be a vector of complex signals, in which case the output signals are also vectors. The angle output similarly contains the angles of the input elements.

Data Type Support

See the description above.

Parameters and Dialog Box



Block Parameters: MOTDSP563 angle

MOTDSP563 Angle (mask)

Phase angle.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

`MOTDSP563 Log` `MOTDSP563 Log10` `MOTDSP563 Sqrt`

Purpose Compute the convolution of two vectors.

Library Motdsp563lib.

Description The MOTDSP563 Convolution block convolves corresponding columns (channels)

In1
MOTDSP563
conv
Out
In2

MOTDSP563 conv

of Mu-by-N input matrix u and Mv-by-N input matrix v.

Frame-Based Inputs

Matrix inputs must be frame-based. The output, y, is a frame-based

(Mu+Mv-1)-by-N matrix.

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_{k,j} v^*_{(i-k+1),j} \qquad 1 \le i \le (M_u + M_v - 1$$

where * denotes the complex conjugate. Inputs u and v are zero when indexed outside of their valid ranges. When both inputs are real, the output is real; when one or both inputs are complex, the output is complex.

When one input is a column vector (single channel) and the other is a matrix (multiple channels), the single-channel input is independently convolved with each channel of the multichannel input. For example, if u is a Mu-by-1 column vector and v is an Mv-by-N matrix, the output is an (Mu+Mv-1)-by-N matrix whose column has elements.

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_k v^*_{(i-k+1),j} \qquad 1 \le i \le (M_u + M_v - 1$$

Sample-Based Inputs

If u and v are sample-based vectors with lengths Mu and Mv, the convolution block performs the vector convolution.

$$y_i = \sum_{k=1}^{\max(M_u, M_v)} u_k v^*_{(i-k+1)} \qquad 1 \le i \le (M_u + M_v - 1)$$

The dimensions of the sample-based output vector are determined by the dimensions of the input vectors:

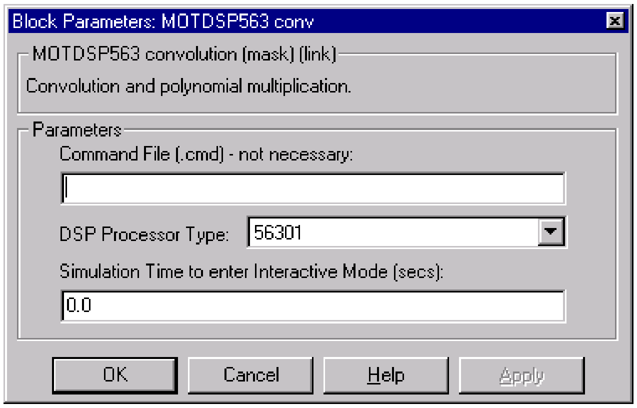
When both inputs are row vectors, or when one input is a row vector and the other is a 1-D vector, the output is a 1-by-(Mu+Mv-1) row vector.

When both inputs are column vectors, or when one input is a column vector and the other is a 1-D vector, the output is a (Mu+Mv-1)-by-1 column vector.

When both inputs are 1-D vectors, the output is a 1-D vector of length Mu+Mv-1.

The Convolution block does not accept sample-based full-dimension matrix inputs, or mixed sample-based row vector and column vector inputs.

Parameters and Dialog Box


Block Parameters: MOTDSP563 conv

MOTDSP563 convolution (mask) (link)

Convolution and polynomial multiplication.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 Correlation

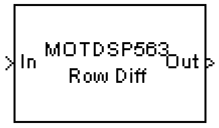
Purpose

Compute the element-to-element difference along a vector.

Library

Motdsp563lib.

Description



MOTDSP563 diff

The MOTDSP563 Difference block computes the difference between successive vector elements. That is, for an input vector u of length N,

```
  y = [u(2)-u(1) u(3)-u(2) ... u(N)-u(N-1)]
```

or

```
  y = mot563_diff(u)          % equivalent MATLAB code
```

The output is a vector of length N-1. A matrix input, u, is treated as a vector, u(:).

Columnwise Differencing

When Columns is selected from the Difference along parameter, the block computes differences between adjacent column elements.

For sample-based inputs, the output is a sample-based (M-1)-by-N matrix whose column has elements:

$$y_{i,j} = u_{i+1,j} - u_{i,j} \qquad 1 \le i \le (M-1)$$

For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors for columnwise differencing, and the output is 1-D.

$$y_{i,j} = u_{i+1,j} - u_{i,j} \qquad 2 \le i \le M$$

The first row of the first output contains zeros, and the first row of each subsequent output contains the difference between the first row of the current input (time t) and the last row of the previous input (time t-Tf).

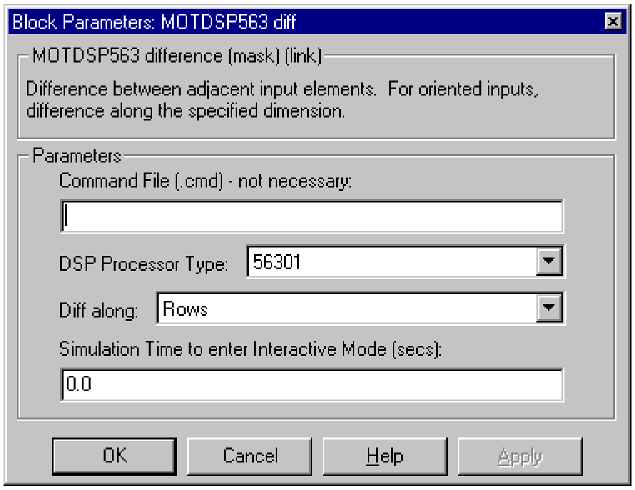
$$y_{1,j}(t) = u_{M,j}(t - T_f) - u_{1,j}(t$$

Rowwise Differencing

When Rows is selected from the Difference along parameter, the block computes differences between adjacent row elements. The output is an M-by-(N-1) matrix.

$$y_{i,j} = u_{i+1,j} - u_{i,j} \qquad 1 \le i \le (M-1)$$

Parameters and Dialog Box


Block Parameters: MOTDSP563 diff

MOTDSP563 difference (mask) (link)

Difference between adjacent input elements. For oriented inputs, difference along the specified dimension.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Diff along: Rows

Simulation Time to enter Interactive Mode (secs):

0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

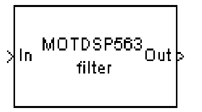
See Also

`MOTDSP563 Convolution`, MOTDSP563 Correlation

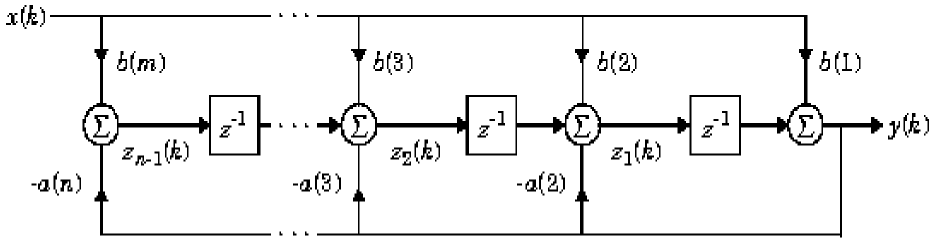
Purpose Apply an IIR filter to the input.

Library Motdsp563lib.

Description The MOTDSP563 Direct-Form II Transpose Filter block applies a transposed direct-form II IIR filter to the input.



MOTDSP563 Direct-Form II Transpose Filter



This is a canonical form that has the minimum number of delay elements. The filter order is max(m, n)-1.

The filter is specified in the parameter dialog box by its transfer function,

- $$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + \ldots + b_{m+1} z^{-(m-1)}}{a_1 + a_2 z^{-1} + \ldots + a_{n+1} z^{-(n-1)}}$$

where the **Numerator** parameter specifies the vector of numerator coefficients,

```
[b(1) b(2) ... b(m)]
```

and the **Denominator** parameter specifies the vector of denominator coefficients,

```
[a(1) a(2) ... a(n)]
```

Note that the filter coefficients are normalized by a1.

Initial Conditions

In its default form, the filter initializes the internal filter states to zero, which is equivalent to assuming past inputs and outputs are zero. The block also accepts optional nonzero initial conditions for the filter delays. Note that the number of filter states (delay elements) per input channel is

```
max(m,n)-1.
```

The **Initial conditions** parameter may take one of four forms:

- Empty matrix

  The empty matrix, [], causes a zero (0) initial condition to be applied to all delay elements in each filter channel.

- Scalar

  The scalar value is copied to all delay elements in each filter channel. Note that a value of zero is equivalent to setting the **Initial conditions** parameter to the empty matrix.

- Vector

  The vector has a length equal to the number of delay elements in each filter channel, max(m,n)-1, and specifies a unique initial condition for each delay element in the filter channel. This vector of initial conditions is applied to each filter channel.

- Matrix

  The matrix specifies a unique initial condition for each delay element, and can specify different initial conditions for each filter channel. The matrix must have the same number of rows as the number of delay elements in the filter, max(m,n)-1, and must have one column per filter channel.

  The **Frame-based inputs** parameter allows you to choose between sample-based and frame-based operation.

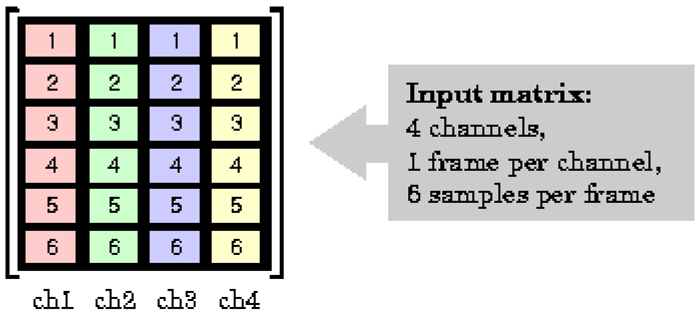
Sample-Based Operation

When the check box is not selected (default), the block assumes that the input is a 1-by-N sample vector or M-by-N sample matrix. Each of the N vector elements (or M*N matrix elements) is treated as an independent channel, and the block filters each channel over time.

Frame-Based Operation

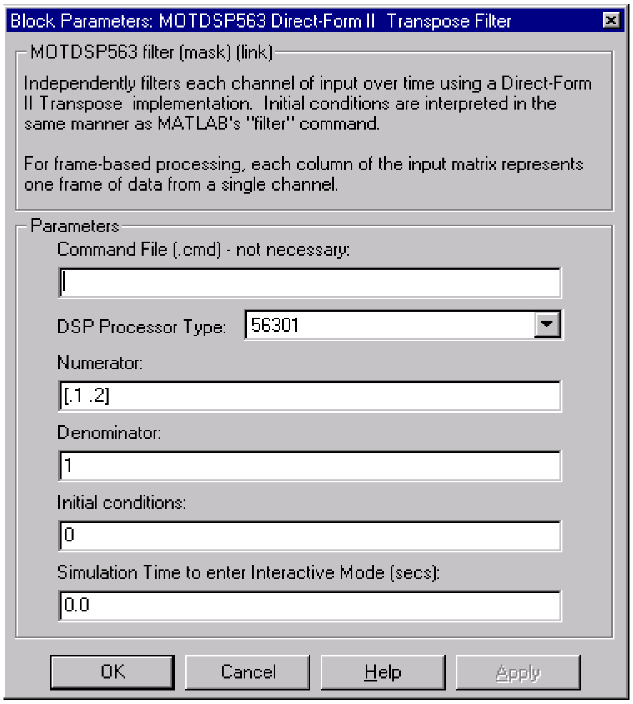
When the Frame-based inputs check box is selected, the block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal. The illustration below shows a 6-by-4 matrix input:

.



The **Number of channels** parameter specifies the number of independent channels (columns), N, in the matrix, and the block filters each channel independently over time. Frame-based operation provides substantial increases in throughput rates, at the expense of greater model latency.

Parameters and Dialog Box



Block Parameters: MOTDSP563 Direct-Form II Transpose Filter

MOTDSP563 filter (mask) (link)

Independently filters each channel of input over time using a Direct-Form II Transpose implementation. Initial conditions are interpreted in the same manner as MATLAB's "filter" command.

For frame-based processing, each column of the input matrix represents one frame of data from a single channel.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Numerator: [.1 .2]

Denominator: 1

Initial conditions: 0

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Numerator

The filter numerator.

Denominator

The filter denominator.

Initial conditions

The filter's initial conditions, a scalar, vector, or matrix.

Frame-based inputs

Selects frame-based operation.

Number of channels

For frame-based operation, the number of columns (channels) in the input matrix.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 Sort

Purpose

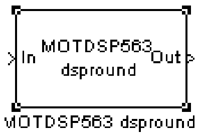
Perform convergent rounding.

Library

Motdsp563lib.

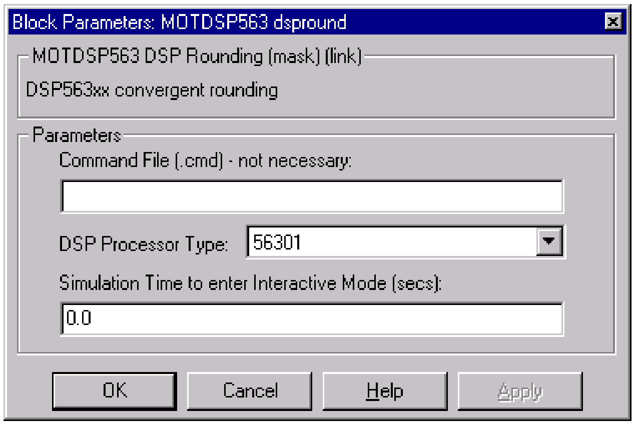
Description

The MOTDSP563 DSP Rounding Function block performs convergent rounding function.



The block accepts and outputs real- or complex-valued signals of type double.

Parameters and Dialog Box


Block Parameters: MOTDSP563 dspround

MOTDSP563 DSP Rounding (mask) (link)

DSP563xx convergent rounding

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs):

0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

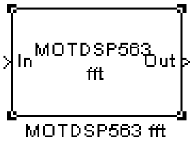
MOTDSP563 DSP Rounding MOTDSP563 Length MOTDSP563 Abs

MOTDSP563 FFT

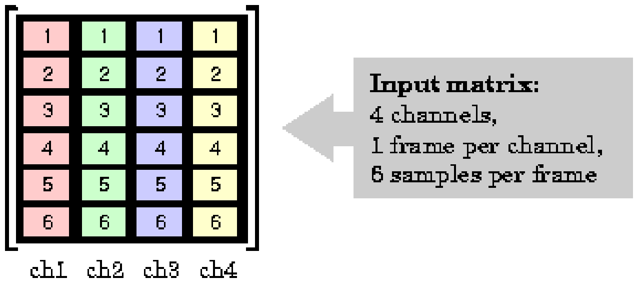
Purpose Compute the FFT of the input.

Library Motdsp563lib.

Description The MOTDSP563 FFT block computes the fast Fourier transform (FFT) of each input channel independently at each sample time. The block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal.

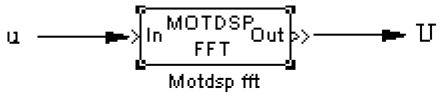


The illustration below shows a 6-by-4 matrix input:



The Number of channels parameter specifies the number of independent channels (columns), N, in the matrix. The output is complex and has the same dimension and sample rate as the input (i.e., the FFT is computed at M frequency points for each channel).

The FFT operation for a single-channel input (**Number of channels** = 1) is shown below.

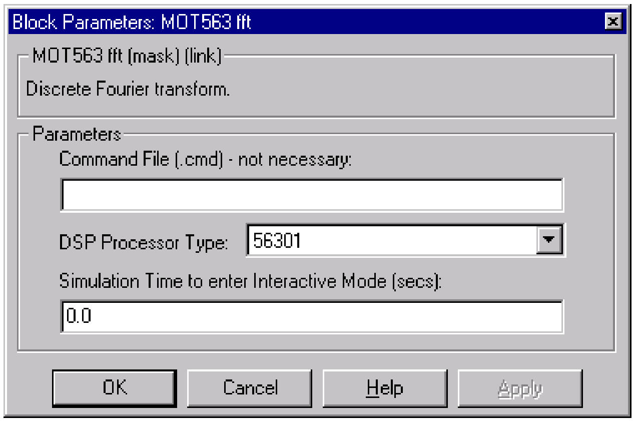


•

$$U(k) = \sum_{m=0}^{M-1} u(m)e^{-j2\pi(mk/M)} \qquad k = 0, \ldots, M-1$$

The input frame size, M, must be a power of two. To work with other frame sizes, use the Zero Pad block to pad or truncate the input frame to a power-of-two length.

Parameters and Dialog Box


Block Parameters: MOT563 fft

MOT563 fft (mask) (link)

Discrete Fourier transform.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of channels

The number of columns (frames) in the input matrix.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 IFFT

Purpose Filter and downsample an input signal.

Library Motdsp563lib.

Description The MOTDSP563 FIR Decimation block resamples the input at an integer rate K times slower than the input sample rate, where K is specified by the **Decimation factor** parameter. This process consists of two steps:

MOTDSP563 FIR Decimation

- The block filters the input data with an FIR filter.
- The block downsamples the filtered data to a lower rate.

The MOTDSP FIR Decimation block implements the FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than straightforward filter-then-decimate algorithms. The output of the decimator is the first filter phase.

In practice, the filter specified by the **FIR filter coefficients** vector should be a lowpass FIR with normalized cutoff frequency no greater than 1/K. The coefficients in the vector are ordered in descending powers of z.

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

The length-m coefficient vector, [b(1) b(2)... b(m)], can be generated by one of the filter design functions in the Signal Processing Toolbox. The filter should be lowpass with normalized cutoff frequency no greater than 1/K. All filter states are internally initialized to zero.

Frame-based inputs

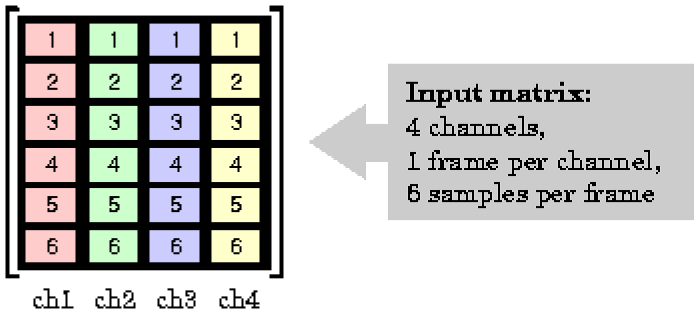
The parameter allows you to choose between sample-based and frame-based operation.

Sample-Based Operation

When the check box is not selected (default), the block assumes that the input is a 1-by-N sample vector or M-by-N sample matrix. Each of the N vector elements (or M*N matrix elements) is treated as an independent channel, and the block decimates each channel over time.

Frame-Based Operation

When the **Frame-based inputs** check box is selected, the block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal. The illustration below shows a 6-by-4 matrix input:



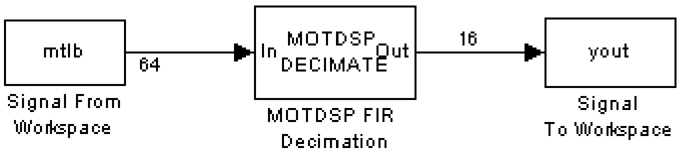
The **Number of channels** parameter specifies the number of independent channels (columns), N, in the matrix, and the block decimates each channel independently over time. Frame-based operation provides substantial increases in throughput rates, at the expense of greater model latency.

In frame-based operation, the **Framing** parameter determines how the block adjusts the rate at the output. There are two available options:

- **Maintain input frame rate**

  The block generates the output at the slower (decimated) rate by using a proportionally smaller frame size than the input. For decimation by a factor of K, the output frame size is K times smaller than the input frame size, but the input and output frame rates are equal. The input frame size must be a multiple of the decimation factor.

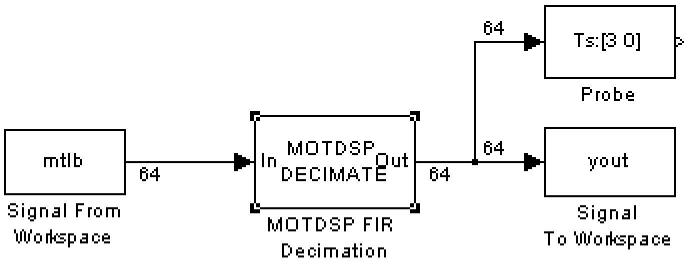
  The example below shows a single-channel input of frame size 64 being decimated by a factor of 4 to a frame size of 16. The block's input and output frame rates are identical.



- **Maintain input frame size**

  The block generates the output at the slower (decimated) rate by using a proportionally longer frame period at the output port than at the input port. For decimation by a factor of K, the output frame period is K times longer than the input frame period, but the input and output frame sizes are equal.

  The example below shows a single-channel input (frame size = 64) with a sample period of 1 second being decimated by a factor of 3 to a sample period of 3 seconds. The input and output frame sizes are identical



Latency

Zero Latency. The FIR Decimation block has zero tasking latency for all single-rate operations. The block is single-rate for the particular combinations of sampling mode and parameter settings shown in the table.

Sampling Mode	Parameter Settings
Sample-based	**Decimation factor** parameter, K, is 1.
Frame-based	**Decimation factor** parameter, K, is 1, or **Framing parameter** is **Maintain input frame rate**.

Note that in sample-based mode, single-rate operation occurs only in the trivial case of factor-of-1 decimation.

The block also has zero latency for sample-based multirate operations in Simulink's single-tasking mode. Zero tasking latency means that the block propagates the first filtered input sample (received at t=0) as the first output sample, followed by filtered input samples K+1, 2K+1, and so on.

Nonzero Latency.

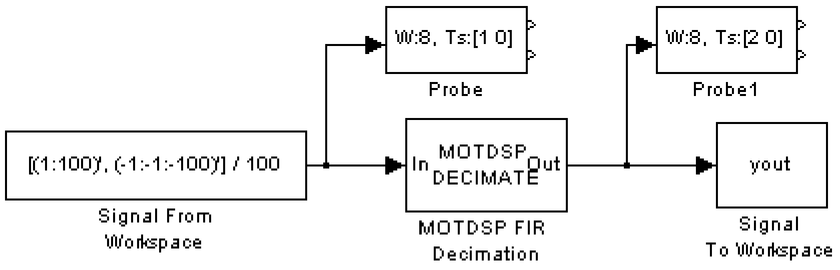
The FIR Decimation block is multirate for all settings other than those in the above table. The amount of latency for multirate operation depends on Simulink's tasking mode and the block's sampling mode, as shown in the table below.

Multirate...	Sample-Based Latency	Frame-Based Latency
Single-tasking	None	One frame (Mi samples)
Multitasking	One sample	One frame (Mi samples)

In cases of one-sample latency, a zero initial condition appears as the first output sample in each channel. The first filtered input sample appears as the second output sample, followed by filtered input samples K+1, 2K+1, and so on.

In cases of one-frame latency, the first Mi output rows contain zeros, where Mi is the input frame size. The first filtered input sample (first filtered row of the input matrix) appears in the output as sample Mi+1, followed by filtered input samples K+1, 2K+1, and so on. See the example below for an illustration of this case.

Example Construct the frame-based model shown below.



Adjust the block parameters as follows.

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2,..., 100, and the second channel should contain the negative ramp signal -1, -2,..., -100.

```
Signal = [(1:100)' (-1:-1:-100)']/100
Sample time = 0.25
Samples per frame = 4
```

- Configure the FIR Decimation block to decimate the two-channel input by decreasing the output frame rate by a factor of 2 relative to the input frame rate. Use a third-order filter with normalized cutoff frequency, fn0, of 0.25. (Note that fn0 satisfies fn0 1/K)

```
FIR filter coefficients = fir1(3,0.25)
Downsample factor = 2
Frame-based inputs
Number of channels = 2
Framing = Maintain input frame size
```

- Configure the Signal To Workspace block for the two-channel input.

```
Frame-based inputs
Number of channels = 2
```

- Configure the Probe blocks by deselecting the **Probe width** and **Probe** complex signal check boxes (if desired).

This model is multirate because there are at least two distinct sample rates, as shown by the two Probe blocks. To run this model in Simulink's multitasking mode, select **Fixed-step** and **discrete** from the **Type** controls in the **Solver** panel of the **Simulation Parameters** dialog box, and select **MultiTasking** from the **Mode** parameter. Also set the **Stop time** to 30.

Run the model and look at the output, yout. The first few samples of each channel are shown below.

```
yout =
                 0                  0
                 0                  0
                 0                  0
                 0                  0
   0.00038576126099  -0.00038576126099
   0.01500010490417  -0.01500010490417
   0.03499984741211  -0.03499984741211
   0.05500006675720  -0.05500006675720
   0.07500004768372  -0.07500004768372
   0.09500002861023  -0.09500002861023
   0.11500000953674  -0.11500000953674
```

Since we ran this frame-based multirate model in multitasking mode, the first four (Mi) output rows are zero. The first filtered input matrix row appears in the output as sample 5 (i.e., sample Mi+1).

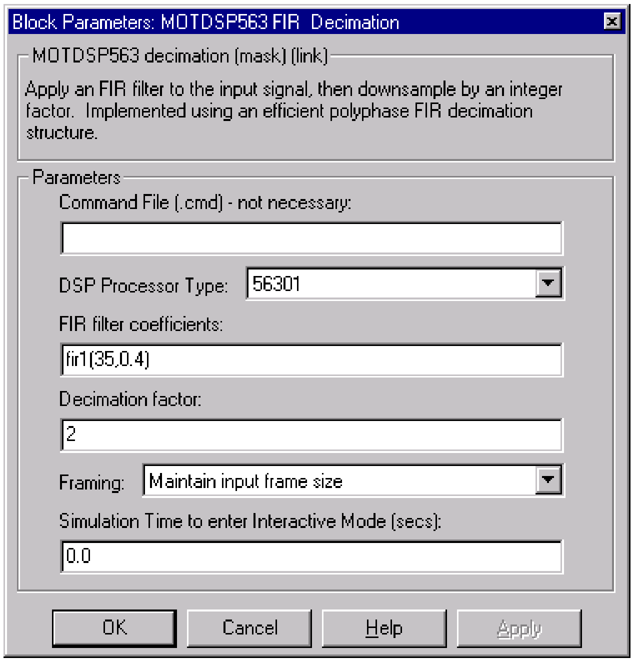
The filter coefficient vector generated by fir1(3,0.25) is

[0.0386 0.4614 0.4614 0.0386]

or, equivalently,

- $H(z) = B(z) = 0.0386 + 0.04614z^{-1} + 0.04614z^{-2} + 0.0386z^{-3}$

Parameters and Dialog Box


Block Parameters: MOTDSP563 FIR Decimation

MOTDSP563 decimation (mask) (link)

Apply an FIR filter to the input signal, then downsample by an integer factor. Implemented using an efficient polyphase FIR decimation structure.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

FIR filter coefficients: fir1(35,0.4)

Decimation factor: 2

Framing: Maintain input frame size

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Specify the DSP processor to be used.

FIR filter coefficients

The FIR filter coefficients, in descending powers of z.

Decimation factor

The integer factor, K, by which to decrease the sample rate of the input sequence.

Frame-based inputs

Selects frame-based operation.

Number of channels

For frame-based operation, the number of columns (channels) in the input matrix, N.

Framing

For frame-based operation, the method by which to implement the decimation; reduce the output frame rate, or reduce the output frame size.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 FIR Interpolation

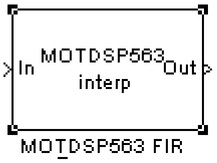
Purpose

Upsample and filter an input signal.

Library

Motdsp563lib.

Description



The MOTDSP563 FIR Interpolation block resamples the input at an integer rate L times faster than the input sample rate, where L is specified by the **Interpolation factor** parameter. This process consists of two steps:

- The block upsamples the input to a higher rate by inserting L-1 zeros between samples.
- The block filters the upsampled data with an FIR filter.

The MOTDSP563 FIR Interpolation block implements the upsampling and FIR filtering steps together using a polyphase filter structure, which is more efficient than straightforward upsample-then-filter algorithms.

The MOTDSP563 FIR filter coefficients parameter specifies the number of numerator coefficients of the FIR filter transfer function H(z).

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

The coefficient vector, [b(1) b(2)... b(m)], can be generated by one of the filter design functions in the Signal Processing Toolbox, and should have a length greater than the interpolation factor (m>L). The filter should be lowpass with normalized cutoff frequency no greater than 1/L. All filter states are internally initialized to zero.

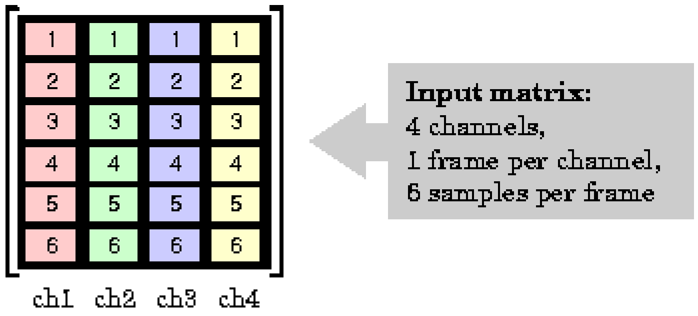
The **Frame-based inputs** parameter allows you to choose between sample-based and frame-based operation.

Sample-Based Operation

When the check box is not selected (default), the block assumes that the input is a 1-by-N sample vector or M-by-N sample matrix. Each of the N vector elements (or M*N matrix elements) is treated as an independent channel, and the block interpolates each channel over time.

Frame-Based Operation

When the **Frame-based inputs** check box is selected, the block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal. The illustration below shows a 6-by-4 matrix input:



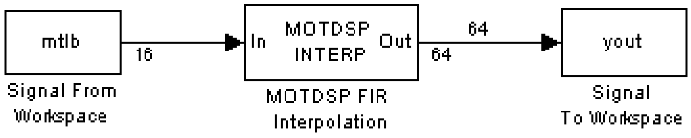
The **Number of channels** parameter specifies the number of independent channels (columns, N) in the matrix, and the block interpolates each channel independently over time. Frame-based operation provides substantial increases in throughput rates, at the expense of greater model latency.

In frame-based operation, the **Framing** parameter determines how the block adjusts the rate at the output. There are two available options:

- **Maintain input frame rate**

  The block generates the output at the faster (interpolated) rate by using a proportionally larger frame size than the input. For interpolation by a factor of L, the output frame size is L times larger than the input frame size, but the input and output frame rates are equal.

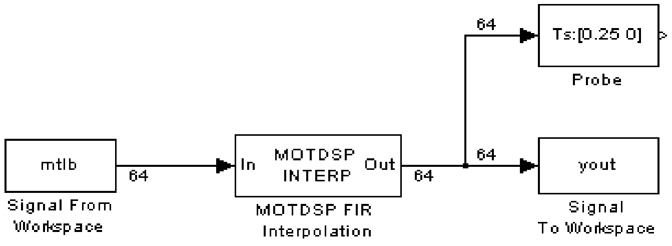
  The example below shows a single-channel input of frame size 16 being upsampled by a factor of 4 to a frame size of 64. The block's input and output frame rates are identical.



- **Maintain input frame size**

  The block generates the output at the faster (interpolated) rate by using a proportionally shorter frame period at the output port than at the input port. For interpolation by a factor of L, the output frame period is L times shorter than the input frame period, but the input and output frame sizes are equal.

  The example below shows a single-channel input (frame size = 64) with a frame period of 1 second being upsampled by a factor of 4 to a frame period of 0.25 seconds. The input and output frame sizes are identical.



Latency

Zero Latency. The FIR Interpolation block has zero tasking latency for all single-rate operations. The block is single-rate for the particular combinations of sampling mode and parameter settings shown in the table below.

Sampling Mode	Parameter Settings
Sample-based	**Interpolation factor** parameter, K, is 1.
Frame-based	**Interpolation factor** parameter, K, is 1, or **Framing parameter** is **Maintain input frame rate**.

Note that in sample-based mode, single-rate operation occurs only in the trivial case of factor-of-1 interpolation.

The block also has zero latency for sample-based multirate operations in Simulink's single-tasking mode. Zero tasking latency means that the block propagates the first filtered input (received at t=0) as the first input sample, followed by L-1 interpolated values, the second filtered input sample, and so on.

Nonzero Latency.

The FIR Interpolation block is multirate for all settings other than those in the above table. The amount of latency for multirate operation depends on Simulink's tasking mode and the block's sampling mode, as shown in the table below.

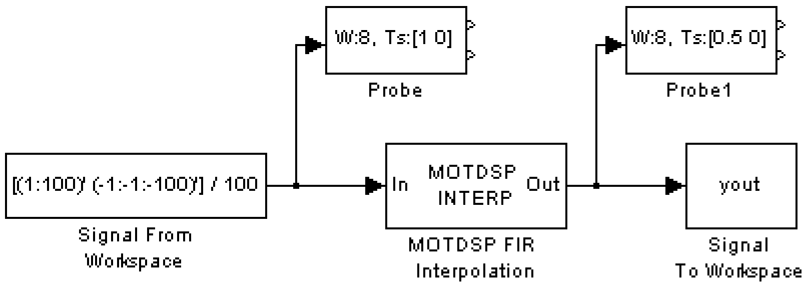
Multirate...	Sample-Based Latency	Frame-Based Latency
Single-tasking	None	One frame (Mi samples)
Multitasking	One sample	One frame (Mi samples)

In cases of one-sample latency, a zero initial condition appears as the first output sample in each channel, followed immediately by the first filtered input sample, L-1 interpolated values, and so on.

In cases of one-frame latency, the first Mi output rows contain zeros, where Mi is the input frame size. The first filtered input sample (first filtered row of the input matrix) appears in the output as sample Mi+1, followed by L-1 interpolated values, the second filtered input sample, and so on. See the example below for an illustration of this case.

Example

Construct the frame-based model shown below.



Adjust the block parameters as follows.

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2,..., 100, and the second channel should contain the negative ramp signal -1, -2,..., -100.

```
Signal = [(1:100)' (-1:-1:-100)']/100
Sample time = 0.25
Samples per frame = 4
```

- Configure the FIR Decimation block to decimate the two-channel input by decreasing the output frame rate by a factor of 2 relative to the input frame rate. Use a third-order filter with normalized cutoff frequency, fn0, of 0.25. (Note that fn0 satisfies fn0 1/K)

```
FIR filter coefficients = fir1(3,0.25)
Downsample factor = 2
Frame-based inputs
Number of channels = 2
```

```
Framing = Maintain input frame size
```

- Configure the Signal To Workspace block for the two-channel input.

```
Frame-based inputs
Number of channels = 2
```

- Configure the Probe blocks by deselecting the **Probe width** and **Probe** complex signal check boxes (if desired).

This model is multirate because there are at least two distinct sample rates, as shown by the two Probe blocks. To run this model in Simulink's multitasking mode, select **Fixed-step** and **discrete** from the **Type** controls in the **Solver** panel of the **Simulation Parameters** dialog box, and select **MultiTasking** from the **Mode** parameter. Also set the **Stop time** to 30.

Run the model and look at the output, yout. The first few samples of each channel are shown below.

```
yout =
                   0                  0
                   0                  0
                   0                  0
                   0                  0
   0.00038576126099  -0.00038576126099
   0.00461423397064  -0.00461423397064
   0.00538575649261  -0.00538575649261
   0.00961422920227  -0.00961422920227
   0.01038587093353  -0.01038587093353
```

Since we ran this frame-based multirate model in multitasking mode, the first four (Mi) output rows are zero. The first filtered input matrix row appears in the output as sample 5 (i.e., sample Mi+1). Every second row is an interpolated value.

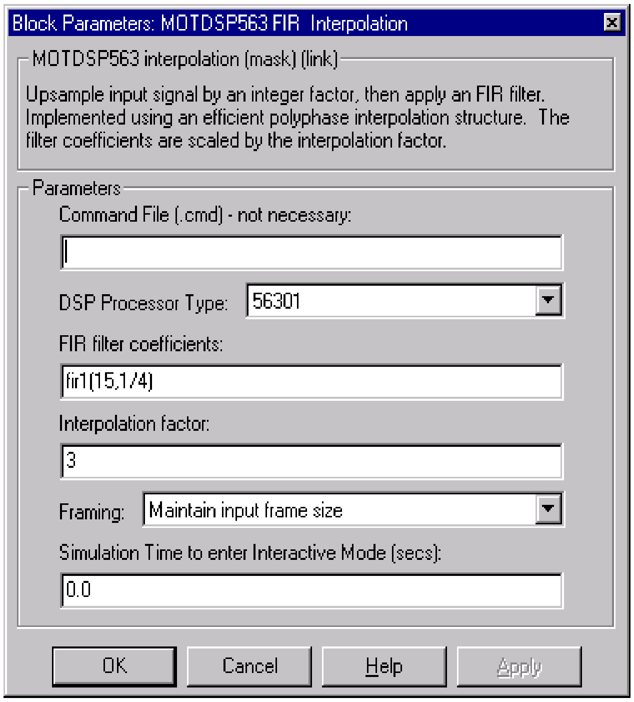
The filter coefficient vector generated by fir1(3,0.25) is

[0.0386 0.4614 0.4614 0.0386]

or, equivalently.

- $H(z) = B(z) = 0.0386 + 0.04614z^{-1} + 0.04614z^{-2} + 0.0386z^{-3}$

Parameters and Dialog Box


Block Parameters: MOTDSP563 FIR Interpolation

MOTDSP563 interpolation (mask) (link)

Upsample input signal by an integer factor, then apply an FIR filter. Implemented using an efficient polyphase interpolation structure. The filter coefficients are scaled by the interpolation factor.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

FIR filter coefficients: fir1(15,1/4)

Interpolation factor: 3

Framing: Maintain input frame size

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Specify the DSP processor to be used.

FIR filter coefficients

The FIR filter coefficients, in descending powers of z.

Interpolation factor

The integer factor, L, by which to increase the sample rate of the input sequence.

Frame-based inputs

Selects frame-based operation.

Number of channels

For frame-based operation, the number of columns (channels) in the input matrix, N.

Framing

For frame-based operation, the method by which to implement the interpolation: increase the output frame rate, or increase the output frame size.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 FIR Decimation

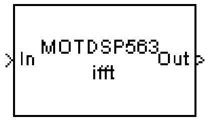
Purpose

Compute the IFFT of the input.

Library

Motdsp563lib.

Description



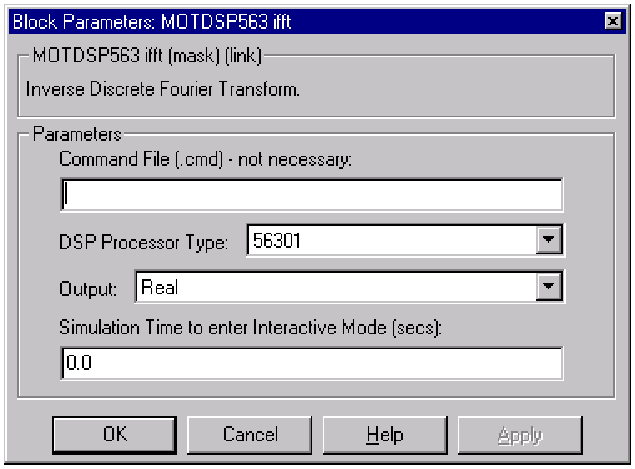
MOTDSP563 ifft

The MOTDSP563 IFFT block computes the inverse fast Fourier transform (IFFT) of each real or complex input channel independently at each sample time. The block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal.

For both sample-based and frame-based inputs, the block assumes that each input column is a frame containing M consecutive frequency-samples from an independent channel. The input must be complex, and the frame size, M, must be a power-of-two.

If the input is frame-based, the output is frame-based; otherwise, the output is sample-based. In either case, the output port rate is the same as the input port rate. For convenience, length-M 1-D vector inputs and *sample-based* length-M row vector inputs are processed as single channels (i.e., as M-by-1 column vectors), and the output has the same dimension as the input.

Parameters and Dialog Box


Block Parameters: MOTDSP563 ifft

MOTDSP563 ifft (mask) (link)

Inverse Discrete Fourier Transform.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Output: Real

Simulation Time to enter Interactive Mode (secs):

0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of channels

The number of channels (columns) in the input.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

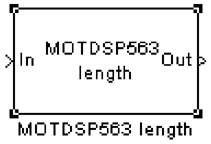
MOTDSP563 FFT

MOTDSP563 Length

Purpose Get number of elements in a vector.

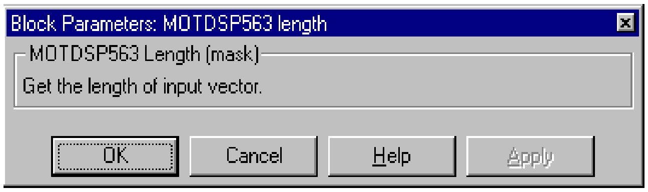
Library Motdsp563lib.

Description The MOTDSP563 Length block returns elements in a vector or the number of row in a matrix.



For length-M 1-D vector inputs or a *sample-based* length-M row vector inputs, the output is the number of element M; For the M-by-N full matrix inputs, the output is the row number M.

Parameters and Dialog Box



Block Parameters: MOTDSP563 length

MOTDSP563 Length (mask)

Get the length of input vector.

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

see also MOTDSP563 Convolution MOTDSP563 Correlation

Purpose

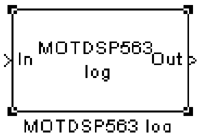
Perform a natural logarithm.

Library

Motdsp563lib.

Description

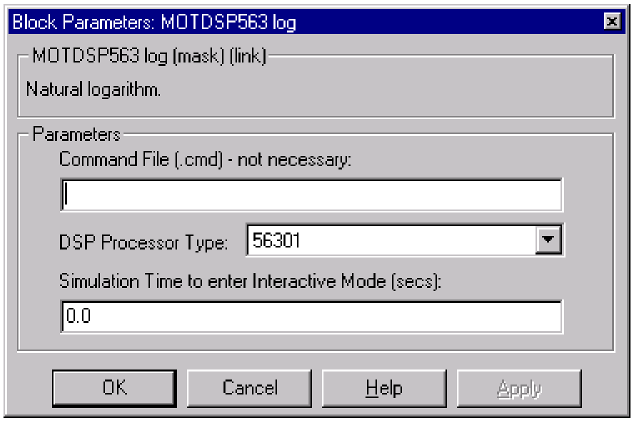
Perform a natural logarithm



Data Type Support

The MOTDSP563 Log block accepts complex or real-valued signals or signal vectors of type double. The output signal type depends on input signal type.

Parameters and Dialog Box


Block Parameters: MOTDSP563 log

MOTDSP563 log (mask) (link)

Natural logarithm.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 Log10 MOTDSP563 Sqrt

MOTDSP563 Log10

Purpose Perform a common (Base 10) logarithm.

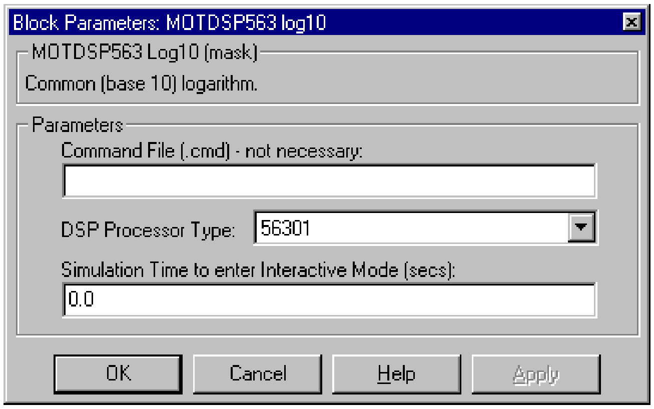
Library Motdsp563lib.

Description Perform a common (Base 10) logarithm.

In
MOTDSP563
log10
Out
MOTDSP563 log10

The MOTDSP563 Log10 block accepts complex or real-valued vectors of type double.The output signal type depends on input signal type.

Parameters and Dialog Box


Block Parameters: MOTDSP563 log10

MOTDSP563 Log10 (mask)

Common (base 10) logarithm.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP563 Log MOTDSP563 Sqrt

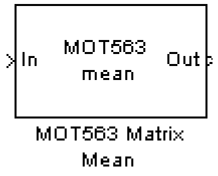
Purpose

Mean the elements of a matrix along rows or columns.

Library

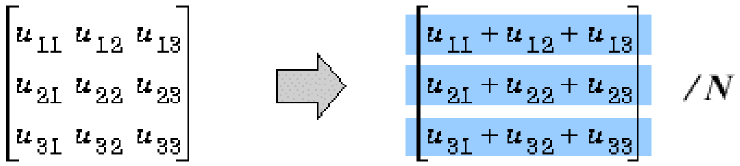
Motdsp563lib.

Description



The MOTDSP563 Matrix Mean block means the elements of an M-by-N input matrix u along either the rows or columns.

When the **Mean along** parameter is set to **Rows**, the block means across the elements of each row and outputs the resulting M-by-1 vector.



$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} + u_{12} + u_{13} \\ u_{21} + u_{22} + u_{23} \\ u_{31} + u_{32} + u_{33} \end{bmatrix} / N$$

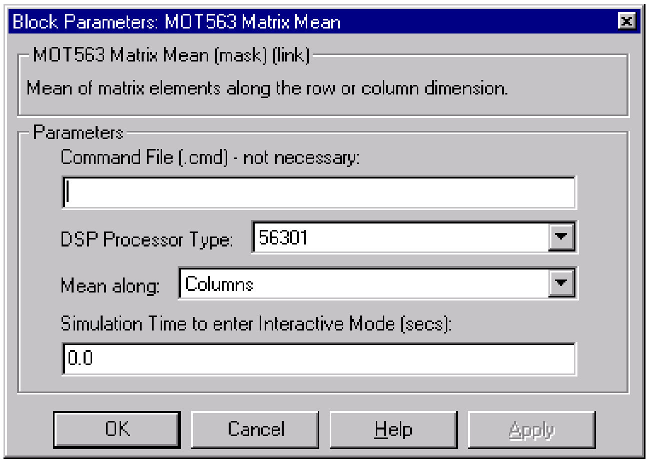
This is equivalent to

```
   y = mot563_mean(u,2)           % equivalent MATLAB code
```

When the **Mean along** parameter is set to **Columns**, the block means down the elements of each column and outputs the resulting 1-by-N vector. This is equivalent to

y = mot563_mean(u)   %equivalent MATLAB code

Parameters and Dialog Box



Block Parameters: MOT563 Matrix Mean

MOT563 Matrix Mean (mask) (link)

Mean of matrix elements along the row or column dimension.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Mean along: Columns

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of input ports

Number of input ports.

Number of columns in input

The number of columns in the input matrix.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 Matrix Sum

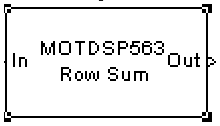
Purpose

Sum the elements of a matrix along rows or columns.

Library

Motdsp563lib.

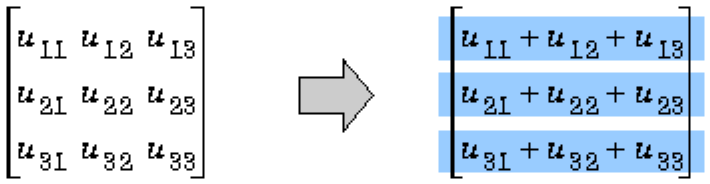
Description



MOTDSP563 Matrix Sum

The MOTDSP563 Matrix Sum block sums the elements of an M-by-N input matrix u along either the rows or columns.

When the **Sum along** parameter is set to **Rows**, the block sums across the elements of each row and outputs the resulting M-by-1 matrix. A length-N 1-D vector input is treated as a 1-by-N matrix..

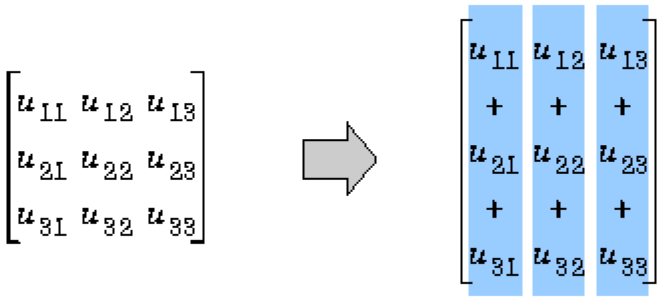


$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11}+u_{12}+u_{13} \\ u_{21}+u_{22}+u_{23} \\ u_{31}+u_{32}+u_{33} \end{bmatrix}$$

This is equivalent to

```
  y = mot563_sum(u,2)          % equivalent MATLAB code
```

When the **Sum along** parameter is set to **Columns**, the block sums down the elements of each column and outputs the resulting 1-by-N matrix. A length-M 1-D vector input is treated as a M-by-1 matrix..



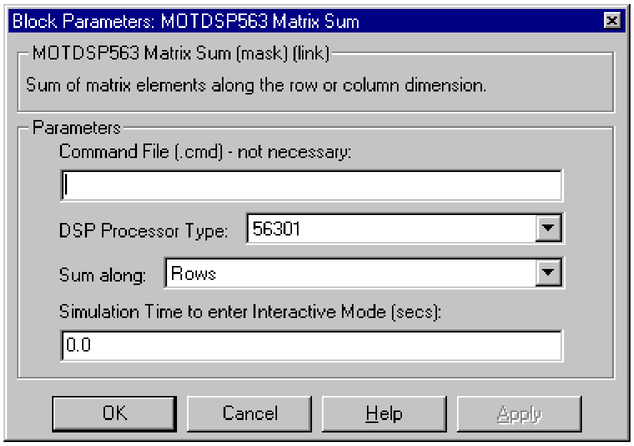
$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ + & + & + \\ u_{21} & u_{22} & u_{23} \\ + & + & + \\ u_{31} & u_{32} & u_{33} \end{bmatrix}$$

This is equivalent to

```
  y = mot563_sum(u)            % equivalent MATLAB code
```

If the input is sample-based, the output is sample-based; if the input is frame-based, the output is frame-based.

Parameters and Dialog Box



Block Parameters: MOTDSP563 Matrix Sum

MOTDSP563 Matrix Sum (mask) (link)

Sum of matrix elements along the row or column dimension.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Sum along: Rows

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

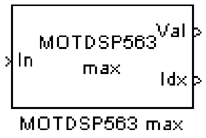
See Also

MOTDSP563 Matrix Mean

Purpose Find the maximum value of one or two input vector(s).

Library Motdsp563lib.

Description



The MOTDSP563 Maximum block identifies the value and position of the largest element in the input.

If the block has two input vectors, the block performs an element-by-element comparison of the input vectors. Each element of the block output vector is the result of the comparison of the elements of the input vectors.

If the block has only one input vector, the **Mode** parameter specifies the block's mode of operation and can be set to **Value**, **Index**, or **Value and Index**. These settings are described below.

Value

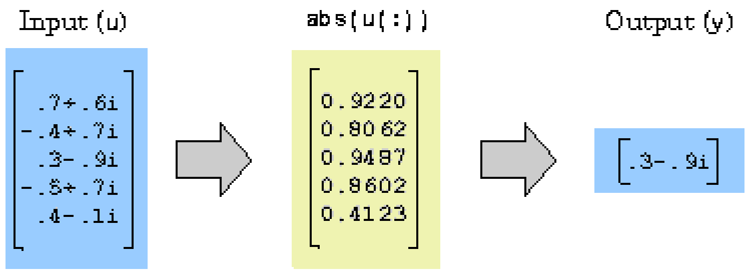
When **Mode** is set to **Value**, the block computes the maximum value in each column of the M-by-N input matrix u independently at each sample time.

[y,i] = mot563_max(u(:))   % equivalent MATLAB code

The block output, y, is the maximum value of the input vector.

For convenience, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors. The output at each sample time, val, is a sample-based 1-by-N vector containing the maximum value of each column in u.

For complex inputs the block uses the magnitude of the input, abs(u(:)), to identify the maximum. The output is the corresponding complex value from the input. as shown below.



Index

When **Mode** is set to **Index**, the block performs the computation shown above, and outputs the index, i, corresponding to the position of the maximum value in the input vector. The index is an integer in the range [1 length(u(:))].

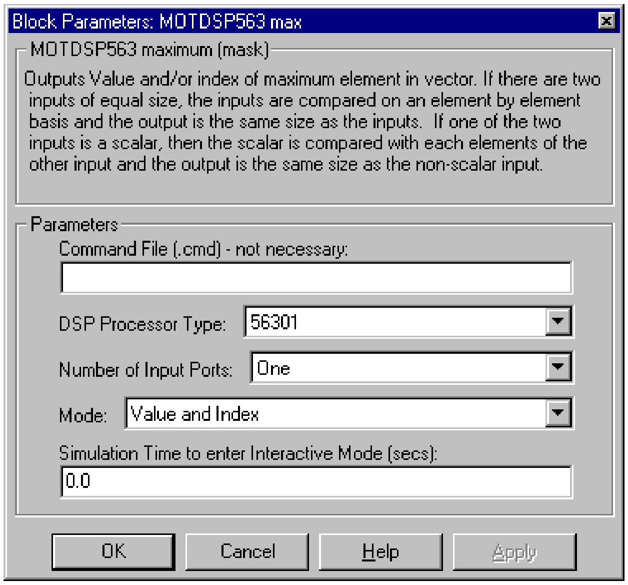
If there are duplicates of the maximum value in the input, the index corresponds to the first occurrence. For example, if the vector input is [.3.2 .1 .2.3], the index of the maximum value is 1, not 5.

Value and Index

When **Mode** is set to **Value and Index**, the block outputs both the value, y, and the index, i.

In all three of the above modes, a matrix input, u, is treated as a vector, u(:)

Parameters and Dialog Box


Block Parameters: MOTDSP563 max

MOTDSP563 maximum (mask)

Outputs Value and/or index of maximum element in vector. If there are two inputs of equal size, the inputs are compared on an element by element basis and the output is the same size as the inputs. If one of the two inputs is a scalar, then the scalar is compared with each elements of the other input and the output is the same size as the non-scalar input.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Number of Input Ports: One

Mode: Value and Index

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of input ports

Number of input ports.

Mode

The block's mode of operation: Output the maximum value of each input, the index of the maximum value, both the value and the index.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 Minimum

MOTDSP563 Minimum

Purpose Find the minimum value of one or two input vectors.

Library Motdsp563lib.

Description The MOTDSP563 Minimum block identifies the value and position of the smallest element in the input.

MOTDSP563
min
In
Val
Idx
MOTDSP563 min

If the block has two input vectors, the block performs an element-by-element comparison of the input vectors. Each element of the block output vector is the result of the comparison of the elements of the input vectors.

If the block has only one input vector, the **Mode** parameter specifies the block's mode of operation and can be set to **Value**, **Index**, or **Value and Index**. These settings are described below.

Value Mode

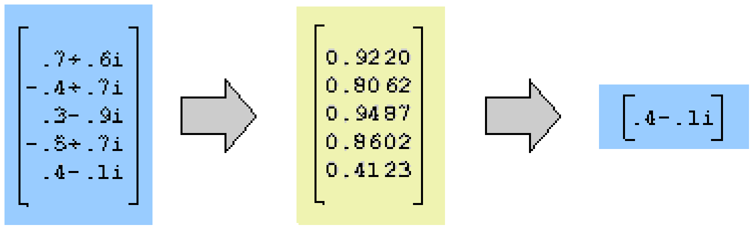
When **Mode** is set to **Value**, the block computes the minimum value of the M-by-N input matrix u independently at each sample time.

```
   [y,i] = mot563_MIN(u(:))        % equivalent MATLAB code
```

The block output, y, is the minimum value of the input vector.

For convenience, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors.

For complex inputs the block uses the magnitude of the input, abs(u(:)), to identify the minimum. The output is the corresponding complex value from the input. as shown below..



Index Mode

When Mode is set to Index, the block performs the computation shown above, and outputs the index, i, corresponding to the position of the minimum value in the input vector. The index is an integer in the range [1 length(u(:))].

If there are duplicates of the minimum value in the input, the index corresponds to the first occurrence.

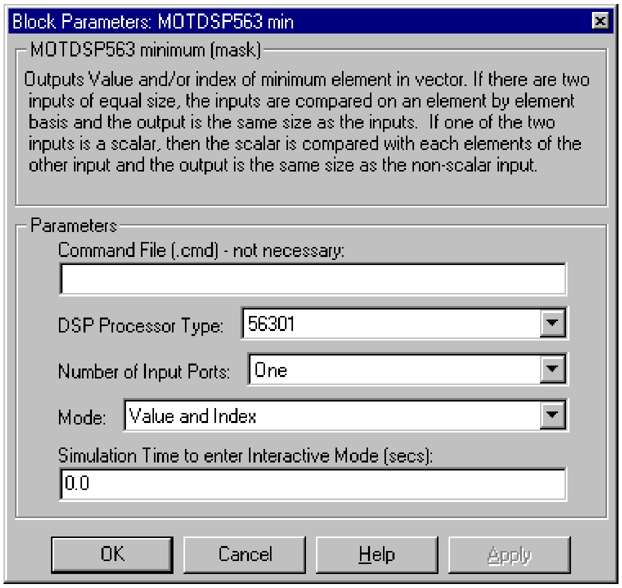
For example, if the vector input is [.1.2.3.2.1], the index of the minimum value is 1, not 5.

Value and Index Mode

When **Mode** is set to **Value and Index**, the block outputs both the vector of minima, val, and the vector of indices, idx.

In all three of the above modes, a matrix input, u, is treated as a vector, u(:).

Parameters and Dialog Box


Block Parameters: MOTDSP563 min

MOTDSP563 minimum (mask)

Outputs Value and/or index of minimum element in vector. If there are two inputs of equal size, the inputs are compared on an element by element basis and the output is the same size as the inputs. If one of the two inputs is a scalar, then the scalar is compared with each elements of the other input and the output is the same size as the non-scalar input.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Number of Input Ports: One

Mode: Value and Index

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of input ports

Number of input ports.

Mode

The block's mode of operation: Output the maximum value of each input, the index of the maximum value, both the value and the index.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP563 Maximum

MOTDSP563 Rounding

Purpose Perform common mathematical rounding.

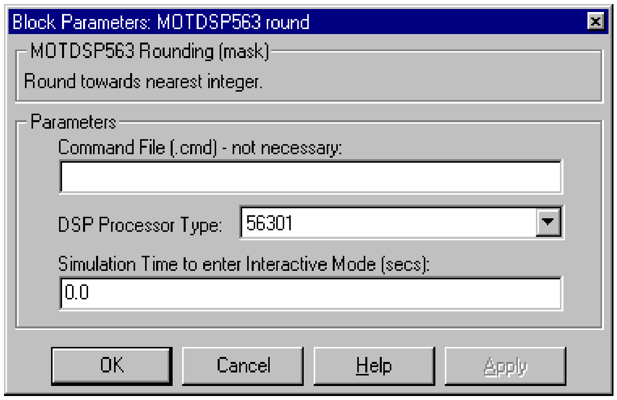
Library Motdsp563lib.

Description The MOTDSP563 Rounding Function block performs common mathematical rounding function.

In
MOTDSP563
round
Out
MOTDSP563 round

The block accepts and output real- or complex-valued signals of type double.

Parameters and Dialog Box



MOTDSP563 Rounding (mask)

Round towards nearest integer.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP563 DSP Rounding MOTDSP563 Length MOTDSP563 Abs

MOTDSP563 Sort

Purpose

Sort the elements in a vector by value.

Library

Motdsp563lib.

Description

In
MOTDSP563
sort
Val
MOTDSP563 sort

The MOTDSP563 Sort block sorts the elements in a real or complex input vector by value using a Quick sort algorithm. The output vector, y, contains the input values arranged in order of ascending.

```
[y,i] = sort(u(:))          % equivalent MATLAB code (ascending)
```

The **Mode** parameter specifies the block's output, and can be set to **Value**, **Index**, or **Value and Index**:

Value Mode

When Mode is set to Value, the block sorts the elements in each column of the M-by-N input matrix u in order of ascending, as specified by the Sort order parameter.

For convenience, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors.

The output at each sample time is a sample-based M-by-N matrix containing the sorted columns of u. Complex inputs are sorted by magnitude.

Index Mode

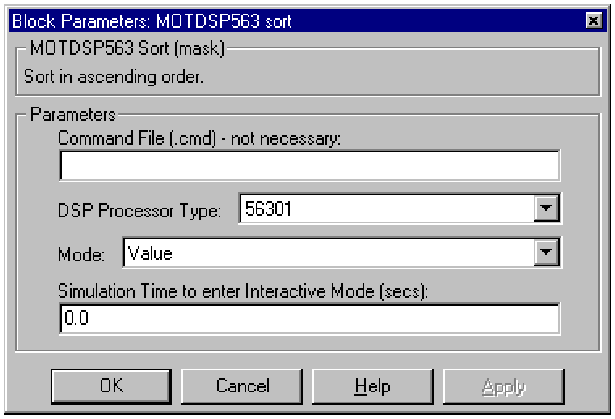
When Mode is set to Index, the block sorts the elements in each column of the M-by-N input matrix u. and outputs the sample-based M-by-N index matrix.

As in Value mode, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors.

Value and Index Mode

When Mode is set to Value and Index, the block outputs both the sorted matrix, and the index matrix. Note that a matrix input is sorted as a single vector, u(:), rather than column by column.

Parameters and Dialog Box



Block Parameters: MOTDSP563 sort

MOTDSP563 Sort (mask)

Sort in ascending order.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Mode: Value

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Mode

The block's mode of operation: Output the sorted vector, the index vector, or both.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 Convolution MOTDSO563 Correlation

MOTDSP563 Sqrt

Purpose Perform a Square Root.

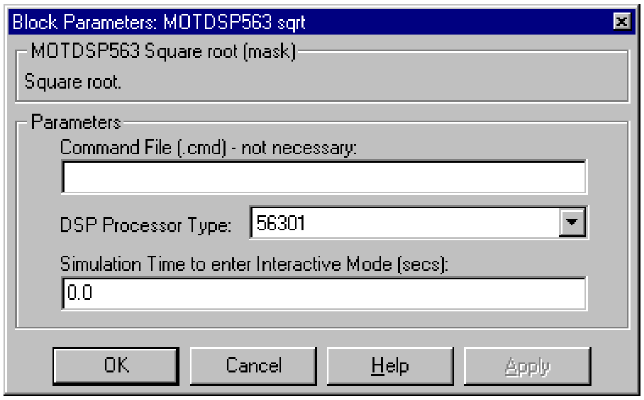
Library Motdsp563lib.

Description Perform a Square Root.

In
MOTDSP563
sqrt
Out
MOTDSP563 sqrt

The MOTDSP563 sqrt block accepts complex or real-valued signals or signal vectors of type double.The output signal type depends on input signal.

Parameters and Dialog Box


Block Parameters: MOTDSP563 sqrt

MOTDSP563 Square root (mask)

Square root.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP563 Abs MOTDSO563 Log MOTDSO563 Log10

Purpose Compute the correlation of two vectors.

Library Motdsp563lib.

Description When both inputs are real, the output is real as well. When one or both inputs are complex, the output is complex.

In1
MOTDSP563
corr
Out
In2
MOTDSP563 correlation

Frame-Based Inputs

Matrix inputs must be frame-based. The output, y, is a frame-based (Mu+Mv-1)-by-N matrix whose column has elements:

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_{k,j} v^*_{(k+i-M_u),j} \qquad 1 \le i \le (M_u + M_v - 1$$

where * denotes the complex conjugate. Inputs u and v are zero when indexed outside of their valid ranges. When both inputs are real, the output is real; when one or both inputs are complex, the output is complex.

When one input is a column vector (single channel) and the other is a matrix (multiple channels), the single-channel input is independently cross-correlated with each channel of the multichannel input. For example, if u is a Mu-by-1 column vector and v is an Mv-by-N matrix, the output is an (Mu+Mv-1)-by-N matrix whose column has elements:

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_k v^*_{(k+i-M_u),j} \qquad 1 \le i \le (M_u + M_v - 1$$

Sample-Based Inputs

If u and v are sample-based vectors with lengths Mu and Mv, the Correlation block performs the vector cross-correlation.

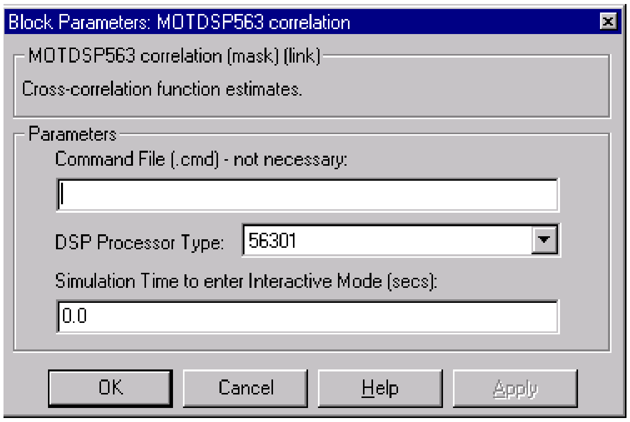
The dimensions of the sample-based output vector are determined by the dimensions of the input vectors:

MOTDSP563 Correlation

$$y_i = \sum_{k=1}^{\max(M_u,\, M_v)} u_k v^*_{(k+i-M_u)} \qquad 1 \le i \le (M_u + M_v - 1$$

The Correlation block does not accept sample-based row vector inputs, or mixed sample-based row vector and column vector inputs.

Parameters and Dialog Box


Block Parameters: MOTDSP563 correlation

MOTDSP563 correlation (mask) (link)

Cross-correlation function estimates.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56301

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP563xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP563 Convolution

Motorola 56600 Family Blockset

The block library within the Motorola 56600 DSP Developer's Kit Blockset contains all 21 Motorola 56600 DSP blocks which vary from elementary math computation (MOTDSP566 Abs, MOTDSP566 Log), to frequency domain transform(MOTDSP566 FFT), even Filtering design is involved.

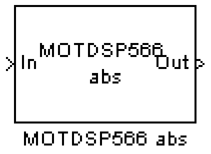
Motorola 56600 DDK Blocks Listed by Category

Motorola 56600 DDK Blockset	
MOTDSP566 Abs	MOTDSP566 Angle
MOTDSP566 Convolution	MOTDSP566 FIR Decimation
MOTDSP566 Difference	MOTDSP566 DSP Rounding
MOTDSP566 FFT	MOTDSP566 Direct-Form II Transpose Filter
MOTDSP566 IFFT	MOTDSP566 FIR Interpolation
MOTDSP566 Length	MOTDSP566 Log
MOTDSP566 Log10	MOTDSP566 Maximum
MOTDSP566 Matrix Mean	MOTDSP566 Minimum
MOTDSP566 Rounding	MOTDSP566 Sort
MOTDSP566 Sqrt	MOTDSP566 Matrix Sum
MOTDSP566 Correlation	

Purpose Output the absolute value of the input.

Library Motdsp566lib.

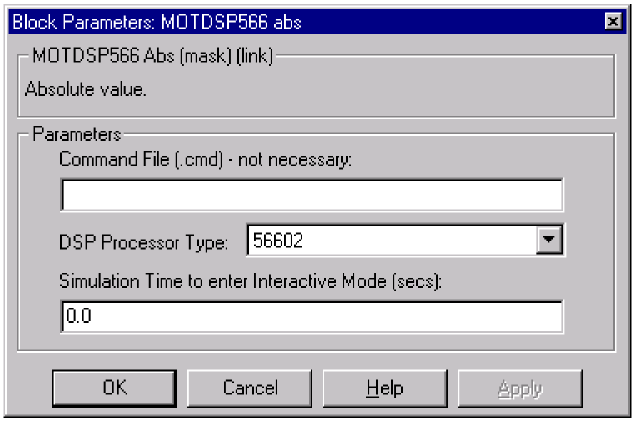
Description The MOTDSP566 Abs block generates as output the absolute value of the input.



Data Type Support

The MOTDSP566 Abs block accepts a real- or complex-valued input of type double and generates a real output of type double.

Parameters and Dialog Box


Block Parameters: MOTDSP566 abs

MOTDSP566 Abs (mask) (link)

Absolute value.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also `MOTDSP566 Log` `MOTDSP566 Log10` `MOTDSP566 Sqrt`

Purpose

Compute the phase angle of a real- or complex-valued signal.

Library

Motdsp566lib.

Description

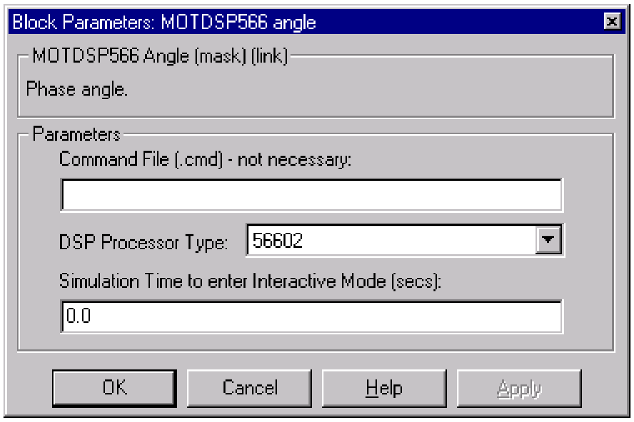
In
MOTDSP566 angle
Out
MOTDSP566 angle

The MOTDSP566 Angle block accepts a real- or complex-valued signal of type double. It outputs the phase angle of the input signal. The outputs are real values of type double. The input may be a vector of complex signals, in which case the output signals are also vectors. The angle output similarly contains the angles of the input elements.

Data Type Support

See the description above.

Parameters and Dialog Box


Block Parameters: MOTDSP566 angle

MOTDSP566 Angle (mask) (link)

Phase angle.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

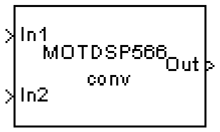
See Also

`MOTDSP566 Log` `MOTDSP566 Log10` `MOTDSP566 Sqrt`

Purpose Compute the convolution of two vectors.

Library Motdsp566lib.

Description The MOTDSP566 Convolution block convolves corresponding columns (channels)



MOTDSP566 conv

of Mu-by-N input matrix u and Mv-by-N input matrix v.

Frame-Based Inputs

Matrix inputs must be frame-based. The output, y, is a frame-based

(Mu+Mv-1)-by-N matrix.

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_{k,j} v^*_{(i-k+1),j} \qquad 1 \le i \le (M_u + M_v - 1$$

where * denotes the complex conjugate. Inputs u and v are zero when indexed outside of their valid ranges. When both inputs are real, the output is real; when one or both inputs are complex, the output is complex.

When one input is a column vector (single channel) and the other is a matrix (multiple channels), the single-channel input is independently convolved with each channel of the multichannel input. For example, if u is a Mu-by-1 column vector and v is an Mv-by-N matrix, the output is an (Mu+Mv-1)-by-N matrix whose column has elements.

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_k v^*_{(i-k+1),j} \qquad 1 \le i \le (M_u + M_v - 1$$

Sample-Based Inputs

If u and v are sample-based vectors with lengths Mu and Mv, the convolution block performs the vector convolution.

$$y_i = \sum_{k=1}^{\max(M_u, M_v)} u_k v^*_{(i-k+1)} \qquad 1 \le i \le (M_u + M_v - 1)$$

The dimensions of the sample-based output vector are determined by the dimensions of the input vectors:

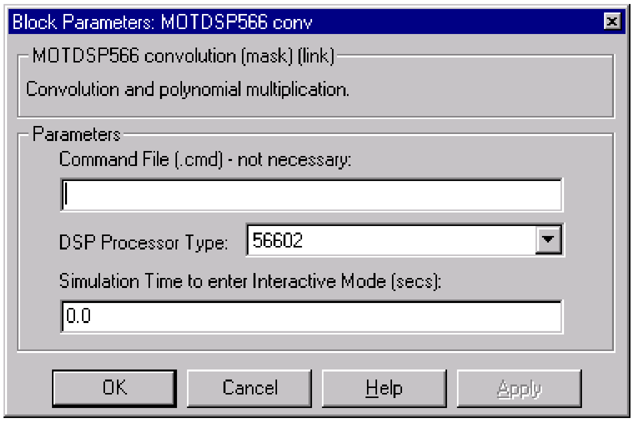
When both inputs are row vectors, or when one input is a row vector and the other is a 1-D vector, the output is a 1-by-(Mu+Mv-1) row vector.

When both inputs are column vectors, or when one input is a column vector and the other is a 1-D vector, the output is a (Mu+Mv-1)-by-1 column vector.

When both inputs are 1-D vectors, the output is a 1-D vector of length Mu+Mv-1.

The Convolution block does not accept sample-based full-dimension matrix inputs, or mixed sample-based row vector and column vector inputs.

Parameters and Dialog Box


Block Parameters: MOTDSP566 conv

MOTDSP566 convolution (mask) (link)

Convolution and polynomial multiplication.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

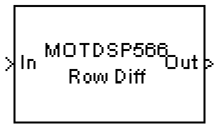
See Also

MOTDSP566 Correlation

Purpose Compute the element-to-element difference along a vector.

Library Motdsp566lib.

Description The MOTDSP566 Difference block computes the difference between successive vector elements. That is, for an input vector u of length N,



MOTDSP566 diff

```
  y = [u(2)-u(1) u(3)-u(2) ... u(N)-u(N-1)]
```
or
```
  y = mot566_diff(u)          % equivalent MATLAB code
```
The output is a vector of length N-1. A matrix input, u, is treated as a vector, u(:).

Columnwise Differencing

When Columns is selected from the Difference along parameter, the block computes differences between adjacent column elements.

For sample-based inputs, the output is a sample-based (M-1)-by-N matrix whose column has elements:

$$y_{i,j} = u_{i+1,j} - u_{i,j} \qquad 1 \le i \le (M-1)$$

For convenience, length-M 1-D vector inputs are treated as M-by-1 column vectors for columnwise differencing, and the output is 1-D.

$$y_{i,j} = u_{i+1,j} - u_{i,j} \qquad 2 \le i \le M$$

The first row of the first output contains zeros, and the first row of each subsequent output contains the difference between the first row of the current input (time t) and the last row of the previous input (time t-Tf).

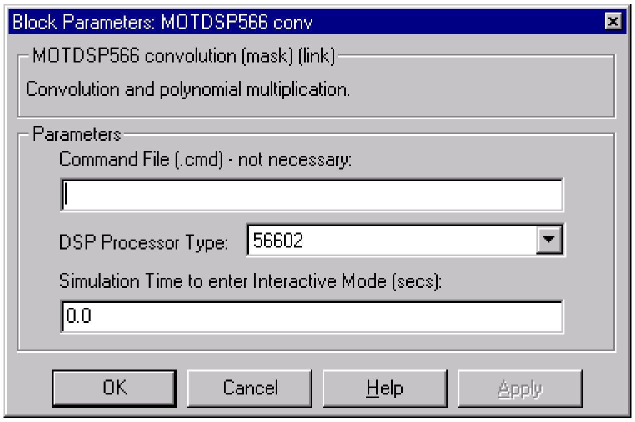
$$y_{1,j}(t) = u_{M,j}(t - T_f) - u_{1,j}(t$$

Rowwise Differencing

When Rows is selected from the Difference along parameter, the block computes differences between adjacent row elements. The output is an M-by-(N-1) matrix.

$$y_{i,j} = u_{i+1,j} - u_{i,j} \qquad 1 \le i \le (M-1)$$

Parameters and Dialog Box


Block Parameters: MOTDSP566 conv

MOTDSP566 convolution (mask) (link)

Convolution and polynomial multiplication.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

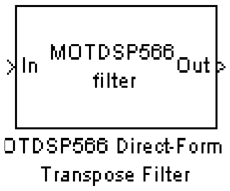
MOTDSP566 Convolution MOTDSP566 Correlation

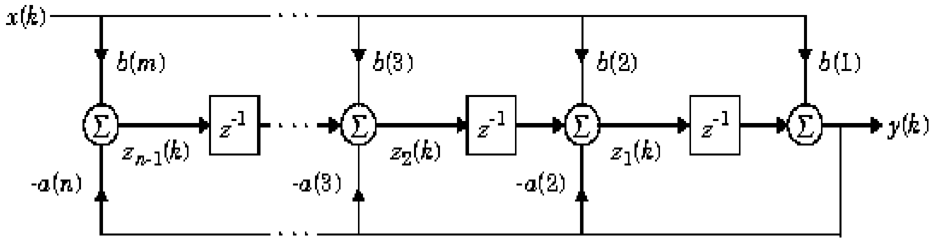
MOTDSP566 Direct-Form II Transpose Filter

Purpose Apply an IIR filter to the input.

Library Motdsp566lib.

Description The MOTDSP566 Direct-Form II Transpose Filter block applies a transposed direct-form II IIR filter to the input.





This is a canonical form that has the minimum number of delay elements. The filter order is max(m, n)-1.

The filter is specified in the parameter dialog box by its transfer function,

- $$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} + \ldots + b_{m+1} z^{-(m-1)}}{a_1 + a_2 z^{-1} + \ldots + a_{n+1} z^{-(n-1)}}$$

where the **Numerator** parameter specifies the vector of numerator coefficients,

```
[b(1) b(2) ... b(m)]
```

and the **Denominator** parameter specifies the vector of denominator coefficients,

```
[a(1) a(2) ... a(n)]
```

Note that the filter coefficients are normalized by a1.

Initial Conditions

In its default form, the filter initializes the internal filter states to zero, which is equivalent to assuming past inputs and outputs are zero. The block also accepts optional nonzero initial conditions for the filter delays. Note that the number of filter states (delay elements) per input channel is

```
max(m,n)-1.
```

The **Initial conditions** parameter may take one of four forms:

- Empty matrix

  The empty matrix, [], causes a zero (0) initial condition to be applied to all delay elements in each filter channel.

- Scalar

  The scalar value is copied to all delay elements in each filter channel. Note that a value of zero is equivalent to setting the **Initial conditions** parameter to the empty matrix.

- Vector

  The vector has a length equal to the number of delay elements in each filter channel, max(m,n)-1, and specifies a unique initial condition for each delay element in the filter channel. This vector of initial conditions is applied to each filter channel.

- Matrix

  The matrix specifies a unique initial condition for each delay element, and can specify different initial conditions for each filter channel. The matrix must have the same number of rows as the number of delay elements in the filter, max(m,n)-1, and must have one column per filter channel.

  The **Frame-based inputs** parameter allows you to choose between sample-based and frame-based operation.

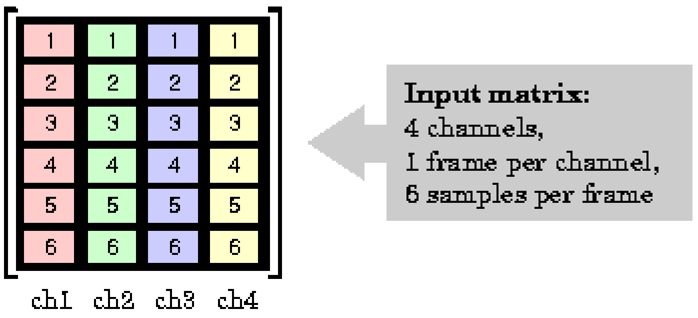
Sample-Based Operation

When the check box is not selected (default), the block assumes that the input is a 1-by-N sample vector or M-by-N sample matrix. Each of the N vector elements (or M*N matrix elements) is treated as an independent channel, and the block filters each channel over time.

Frame-Based Operation

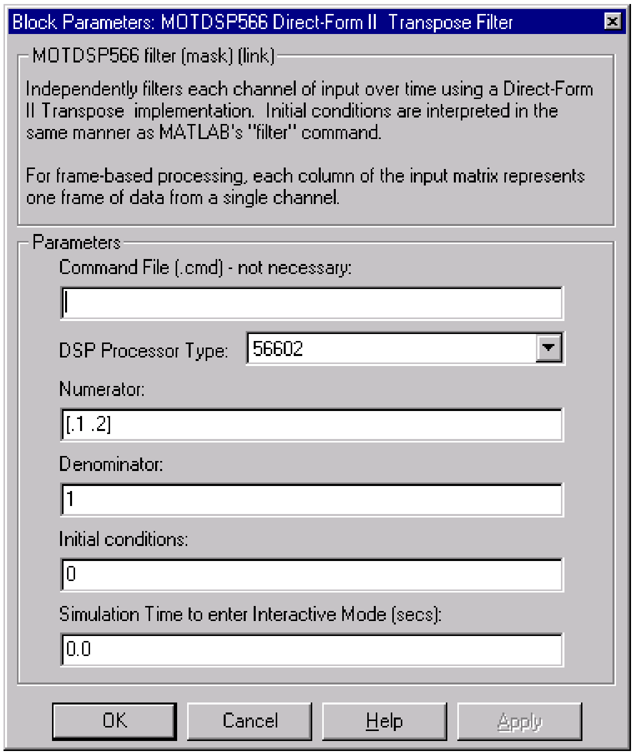
When the Frame-based inputs check box is selected, the block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal. The illustration below shows a 6-by-4 matrix input:

.



The **Number of channels** parameter specifies the number of independent channels (columns), N, in the matrix, and the block filters each channel independently over time. Frame-based operation provides substantial increases in throughput rates, at the expense of greater model latency.

Parameters and Dialog Box



Block Parameters: MOTDSP566 Direct-Form II Transpose Filter

MOTDSP566 filter (mask) (link)

Independently filters each channel of input over time using a Direct-Form II Transpose implementation. Initial conditions are interpreted in the same manner as MATLAB's "filter" command.

For frame-based processing, each column of the input matrix represents one frame of data from a single channel.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Numerator: [.1 .2]

Denominator: 1

Initial conditions: 0

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Numerator

The filter numerator.

Denominator

The filter denominator.

Initial conditions

The filter's initial conditions, a scalar, vector, or matrix.

Frame-based inputs

Selects frame-based operation.

Number of channels

For frame-based operation, the number of columns (channels) in the input matrix.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP566 Sort

MOTDSP566 Rounding

Purpose Perform convergent rounding.

Library Motdsp566lib.

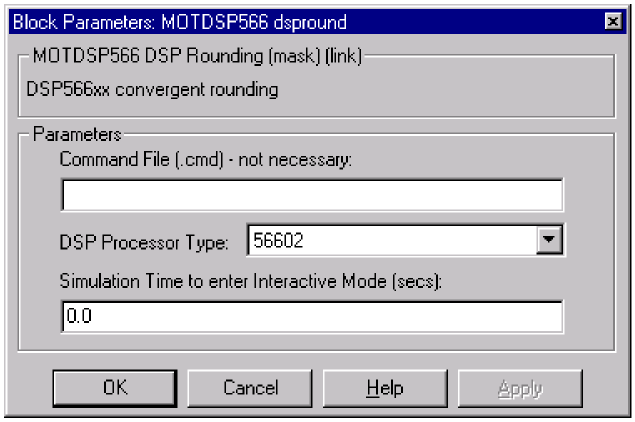
Description The MOTDSP566 DSP Rounding Function block performs convergent rounding function.

In
MOTDSP566
dspround
Out

MOTDSP566 dspround

The block accepts and outputs real- or complex-valued signals of type double.

Parameters and Dialog Box



MOTDSP566 DSP Rounding (mask) (link)

DSP566xx convergent rounding

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP566 DSP Rounding MOTDSP566 Length MOTDSP566 Abs

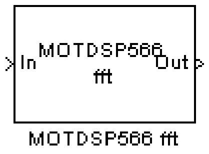
Purpose

Compute the FFT of the input.

Library

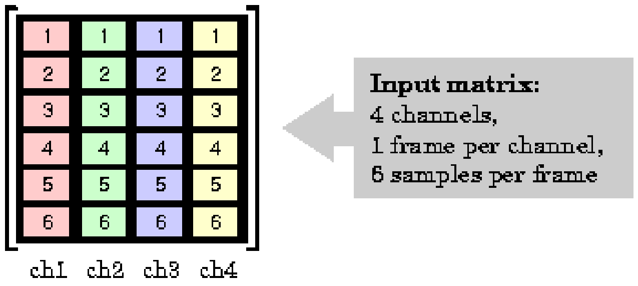
Motdsp566lib.

Description



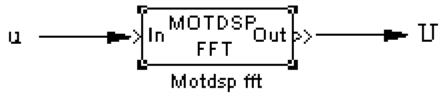
The MOTDSP566 FFT block computes the fast Fourier transform (FFT) of each input channel independently at each sample time. The block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal.

The illustration below shows a 6-by-4 matrix input:



The Number of channels parameter specifies the number of independent channels (columns), N, in the matrix. The output is complex and has the same dimension and sample rate as the input (i.e., the FFT is computed at M frequency points for each channel).

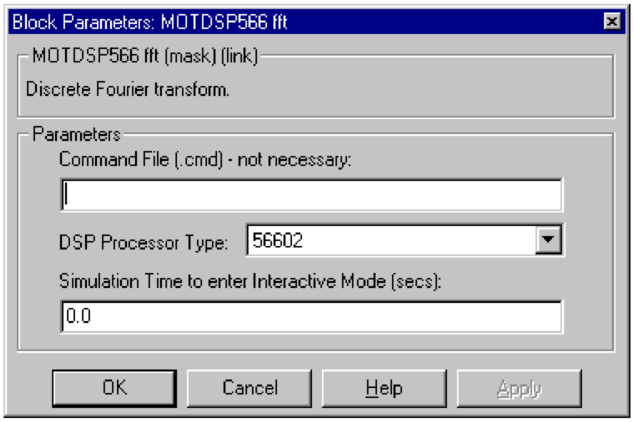
The FFT operation for a single-channel input (**Number of channels** = 1) is shown below.



- $$U(k) = \sum_{m=0}^{M-1} u(m)e^{-j2\pi(mk/M)} \qquad k = 0, \ldots, M-1$$

The input frame size, M, must be a power of two. To work with other frame sizes, use the Zero Pad block to pad or truncate the input frame to a power-of-two length.

Parameters and Dialog Box


Block Parameters: MOTDSP566 fft

MOTDSP566 fft (mask) (link)

Discrete Fourier transform.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of channels

The number of columns (frames) in the input matrix.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP566 IFFT

Purpose Filter and downsample an input signal.

Library Motdsp566lib.

Description The MOTDSP566 FIR Decimation block resamples the input at an integer rate K times slower than the input sample rate, where K is specified by the **Decimation factor** parameter. This process consists of two steps:

In
MOTDSP566
decimate
Out

MOTDSP566 FIR Decimation

- The block filters the input data with an FIR filter.
- The block downsamples the filtered data to a lower rate.

The MOTDSP566 FIR Decimation block implements the FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than straightforward filter-then-decimate algorithms. The output of the decimator is the first filter phase.

In practice, the filter specified by the **FIR filter coefficients** vector should be a lowpass FIR with normalized cutoff frequency no greater than 1/K. The coefficients in the vector are ordered in descending powers of z.

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

The length-m coefficient vector, [b(1) b(2)... b(m)], can be generated by one of the filter design functions in the Signal Processing Toolbox. The filter should be lowpass with normalized cutoff frequency no greater than 1/K. All filter states are internally initialized to zero.

Frame-based inputs

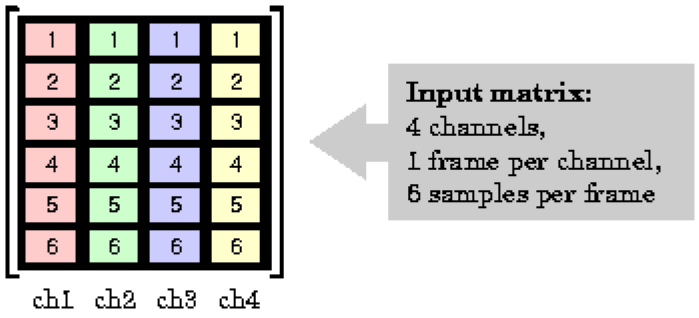
The parameter allows you to choose between sample-based and frame-based operation.

Sample-Based Operation

When the check box is not selected (default), the block assumes that the input is a 1-by-N sample vector or M-by-N sample matrix. Each of the N vector elements (or M*N matrix elements) is treated as an independent channel, and the block decimates each channel over time.

Frame-Based Operation

When the **Frame-based inputs** check box is selected, the block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal. The illustration below shows a 6-by-4 matrix input:



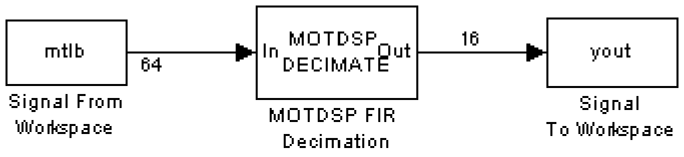
The **Number of channels** parameter specifies the number of independent channels (columns), N, in the matrix, and the block decimates each channel independently over time. Frame-based operation provides substantial increases in throughput rates, at the expense of greater model latency.

In frame-based operation, the **Framing** parameter determines how the block adjusts the rate at the output. There are two available options:

- **Maintain input frame rate**

  The block generates the output at the slower (decimated) rate by using a proportionally smaller frame size than the input. For decimation by a factor of K, the output frame size is K times smaller than the input frame size, but the input and output frame rates are equal. The input frame size must be a multiple of the decimation factor.

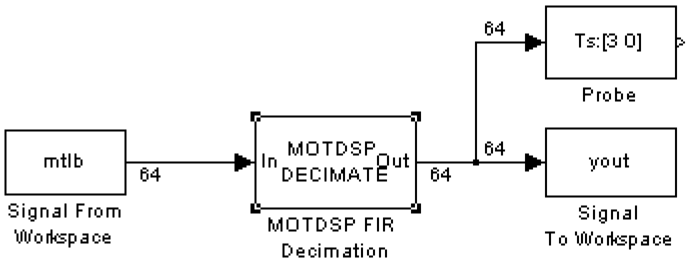
  The example below shows a single-channel input of frame size 64 being decimated by a factor of 4 to a frame size of 16. The block's input and output frame rates are identical.



- **Maintain input frame size**

  The block generates the output at the slower (decimated) rate by using a proportionally longer frame period at the output port than at the input port. For decimation by a factor of K, the output frame period is K times longer than the input frame period, but the input and output frame sizes are equal.

  The example below shows a single-channel input (frame size = 64) with a sample period of 1 second being decimated by a factor of 3 to a sample period of 3 seconds. The input and output frame sizes are identical



Latency

Zero Latency. The FIR Decimation block has zero tasking latency for all single-rate operations. The block is single-rate for the particular combinations of sampling mode and parameter settings shown in the table.

Sampling Mode	Parameter Settings
Sample-based	**Decimation factor** parameter, K, is 1.
Frame-based	**Decimation factor** parameter, K, is 1, or **Framing parameter** is **Maintain input frame rate**.

Note that in sample-based mode, single-rate operation occurs only in the trivial case of factor-of-1 decimation.

The block also has zero latency for sample-based multirate operations in Simulink's single-tasking mode. Zero tasking latency means that the block propagates the first filtered input sample (received at t=0) as the first output sample, followed by filtered input samples K+1, 2K+1, and so on.

Nonzero Latency.

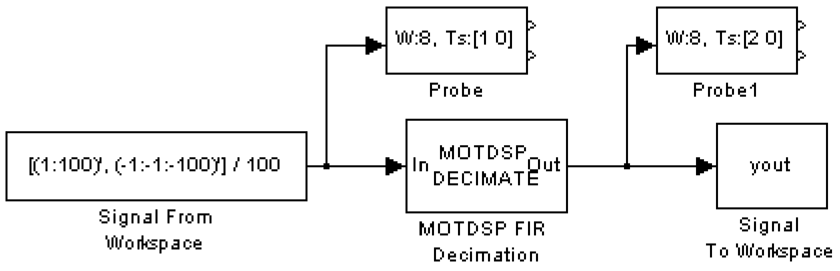
The FIR Decimation block is multirate for all settings other than those in the above table. The amount of latency for multirate operation depends on Simulink's tasking mode and the block's sampling mode, as shown in the table below.

Multirate...	Sample-Based Latency	Frame-Based Latency
Single-tasking	None	One frame (Mi samples)
Multitasking	One sample	One frame (Mi samples)

In cases of one-sample latency, a zero initial condition appears as the first output sample in each channel. The first filtered input sample appears as the second output sample, followed by filtered input samples K+1, 2K+1, and so on.

In cases of one-frame latency, the first Mi output rows contain zeros, where Mi is the input frame size. The first filtered input sample (first filtered row of the input matrix) appears in the output as sample Mi+1, followed by filtered input samples K+1, 2K+1, and so on. See the example below for an illustration of this case.

Example Construct the frame-based model shown below.



Adjust the block parameters as follows.

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2,..., 100, and the second channel should contain the negative ramp signal -1, -2,..., -100.

```
Signal = [(1:100)' (-1:-1:-100)']/100
Sample time = 0.25
Samples per frame = 4
```

- Configure the FIR Decimation block to decimate the two-channel input by decreasing the output frame rate by a factor of 2 relative to the input frame rate. Use a third-order filter with normalized cutoff frequency, fn0, of 0.25. (Note that fn0 satisfies fn0 1/K)

```
FIR filter coefficients = fir1(3,0.25)
Downsample factor = 2
Frame-based inputs
Number of channels = 2
Framing = Maintain input frame size
```

- Configure the Signal To Workspace block for the two-channel input.

```
Frame-based inputs
Number of channels = 2
```

- Configure the Probe blocks by deselecting the **Probe width** and **Probe** complex signal check boxes (if desired).

This model is multirate because there are at least two distinct sample rates, as shown by the two Probe blocks. To run this model in Simulink's multitasking mode, select **Fixed-step** and **discrete** from the **Type** controls in the **Solver** panel of the **Simulation Parameters** dialog box, and select **MultiTasking** from the **Mode** parameter. Also set the **Stop time** to 30.

Run the model and look at the output, yout. The first few samples of each channel are shown below.

```
yout =
                  0                  0
                  0                  0
                  0                  0
                  0                  0
   0.00038576126099  -0.00038576126099
   0.01500010490417  -0.01500010490417
   0.03499984741211  -0.03499984741211
   0.05500006675720  -0.05500006675720
   0.07500004768372  -0.07500004768372
   0.09500002861023  -0.09500002861023
   0.11500000953674  -0.11500000953674
```

Since we ran this frame-based multirate model in multitasking mode, the first four (Mi) output rows are zero. The first filtered input matrix row appears in the output as sample 5 (i.e., sample Mi+1).

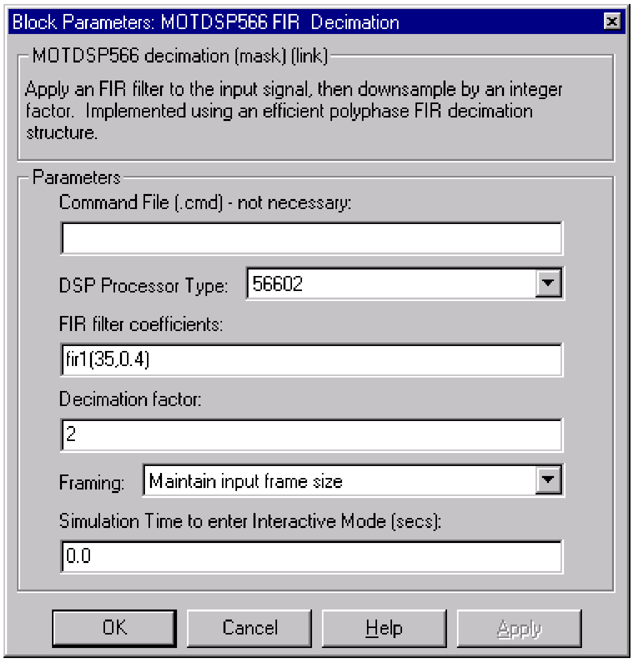
The filter coefficient vector generated by fir1(3,0.25) is

[0.0386 0.4614 0.4614 0.0386]

or, equivalently,

- $H(z) = B(z) = 0.0386 + 0.04614z^{-1} + 0.04614z^{-2} + 0.0386z^{-3}$

Parameters and Dialog Box



Block Parameters: MOTDSP566 FIR Decimation

MOTDSP566 decimation (mask) (link)

Apply an FIR filter to the input signal, then downsample by an integer factor. Implemented using an efficient polyphase FIR decimation structure.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

FIR filter coefficients: fir1(35,0.4)

Decimation factor: 2

Framing: Maintain input frame size

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Specify the DSP processor to be used.

FIR filter coefficients

The FIR filter coefficients, in descending powers of z.

Decimation factor

The integer factor, K, by which to decrease the sample rate of the input sequence.

Frame-based inputs

Selects frame-based operation.

Number of channels

For frame-based operation, the number of columns (channels) in the input matrix, N.

Framing

For frame-based operation, the method by which to implement the decimation; reduce the output frame rate, or reduce the output frame size.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

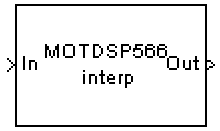
See Also

MOTDSP566 FIR Interpolation

Purpose Upsample and filter an input signal.

Library Motdsp566lib.

Description



MOTDSP566 FIR Interpolation

The MOTDSP566 FIR Interpolation block resamples the input at an integer rate L times faster than the input sample rate, where L is specified by the **Interpolation factor** parameter. This process consists of two steps:

- The block upsamples the input to a higher rate by inserting L-1 zeros between samples.
- The block filters the upsampled data with an FIR filter.

The MOTDSP566 FIR Interpolation block implements the upsampling and FIR filtering steps together using a polyphase filter structure, which is more efficient than straightforward upsample-then-filter algorithms.

The MOTDSP566 FIR filter coefficients parameter specifies the number numerator coefficients of the FIR filter transfer function H(z).

$$H(z) = B(z) = b_1 + b_2 z^{-1} + \ldots + b_m z^{-(m-1)}$$

The coefficient vector, [b(1) b(2)... b(m)], can be generated by one of the filter design functions in the Signal Processing Toolbox, and should have a length greater than the interpolation factor (m>L). The filter should be lowpass with normalized cutoff frequency no greater than 1/L. All filter states are internally initialized to zero.

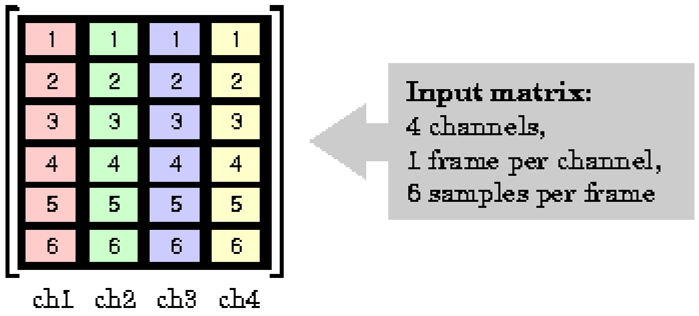
The **Frame-based inputs** parameter allows you to choose between sample-based and frame-based operation.

Sample-Based Operation

When the check box is not selected (default), the block assumes that the input is a 1-by-N sample vector or M-by-N sample matrix. Each of the N vector elements (or M*N matrix elements) is treated as an independent channel, and the block interpolates each channel over time.

Frame-Based Operation

When the **Frame-based inputs** check box is selected, the block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal. The illustration below shows a 6-by-4 matrix input:



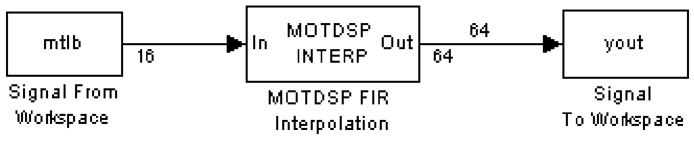
The **Number of channels** parameter specifies the number of independent channels (columns, N) in the matrix, and the block interpolates each channel independently over time. Frame-based operation provides substantial increases in throughput rates, at the expense of greater model latency.

In frame-based operation, the **Framing** parameter determines how the block adjusts the rate at the output. There are two available options:

- **Maintain input frame rate**

  The block generates the output at the faster (interpolated) rate by using a proportionally larger frame size than the input. For interpolation by a factor of L, the output frame size is L times larger than the input frame size, but the input and output frame rates are equal.

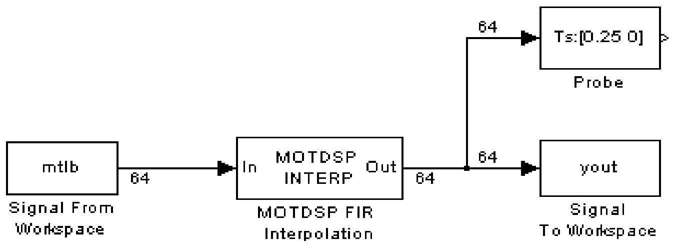
  The example below shows a single-channel input of frame size 16 being upsampled by a factor of 4 to a frame size of 64. The block's input and output frame rates are identical.



- **Maintain input frame size**

  The block generates the output at the faster (interpolated) rate by using a proportionally shorter frame period at the output port than at the input port. For interpolation by a factor of L, the output frame period is L times shorter than the input frame period, but the input and output frame sizes are equal.

  The example below shows a single-channel input (frame size = 64) with a frame period of 1 second being upsampled by a factor of 4 to a frame period of 0.25 seconds. The input and output frame sizes are identical.



Latency

Zero Latency. The FIR Interpolation block has zero tasking latency for all single-rate operations. The block is single-rate for the particular combinations of sampling mode and parameter settings shown in the table below.

Sampling Mode	Parameter Settings
Sample-based	**Interpolation factor** parameter, K, is 1.
Frame-based	**Interpolation factor** parameter, K, is 1, or **Framing parameter** is **Maintain input frame rate**.

Note that in sample-based mode, single-rate operation occurs only in the trivial case of factor-of-1 interpolation.

The block also has zero latency for sample-based multirate operations in Simulink's single-tasking mode. Zero tasking latency means that the block propagates the first filtered input (received at t=0) as the first input sample, followed by L-1 interpolated values, the second filtered input sample, and so on.

Nonzero Latency.

The FIR Interpolation block is multirate for all settings other than those in the above table. The amount of latency for multirate operation depends on Simulink's tasking mode and the block's sampling mode, as shown in the table below.

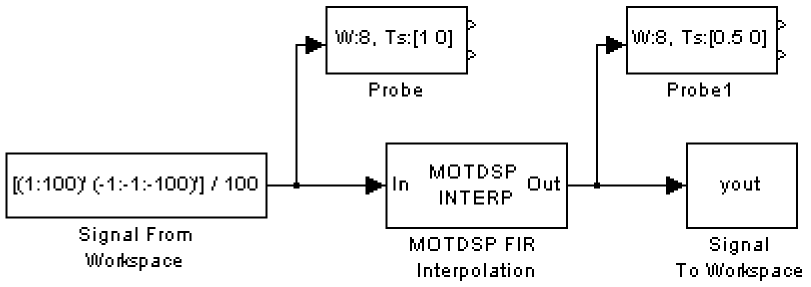
Multirate...	Sample-Based Latency	Frame-Based Latency
Single-tasking	None	One frame (Mi samples)
Multitasking	One sample	One frame (Mi samples)

In cases of one-sample latency, a zero initial condition appears as the first output sample in each channel, followed immediately by the first filtered input sample, L-1 interpolated values, and so on.

In cases of one-frame latency, the first Mi output rows contain zeros, where Mi is the input frame size. The first filtered input sample (first filtered row of the input matrix) appears in the output as sample Mi+1, followed by L-1 interpolated values, the second filtered input sample, and so on. See the example below for an illustration of this case.

Example

Construct the frame-based model shown below.



Adjust the block parameters as follows.

- Configure the Signal From Workspace block to generate a two-channel signal with frame size of 4 and sample period of 0.25. This represents an output frame period of 1 (0.25*4). The first channel should contain the positive ramp signal 1, 2,..., 100, and the second channel should contain the negative ramp signal -1, -2,..., -100.

```
Signal = [(1:100)' (-1:-1:-100)']/100
Sample time = 0.25
Samples per frame = 4
```

- Configure the FIR Decimation block to decimate the two-channel input by decreasing the output frame rate by a factor of 2 relative to the input frame rate. Use a third-order filter with normalized cutoff frequency, fn0, of 0.25. (Note that fn0 satisfies fn0 1/K)

```
FIR filter coefficients = fir1(3,0.25)
Downsample factor = 2
Frame-based inputs
Number of channels = 2
```

```
Framing = Maintain input frame size
```

- Configure the Signal To Workspace block for the two-channel input.

```
Frame-based inputs
Number of channels = 2
```

- Configure the Probe blocks by deselecting the **Probe width** and **Probe** complex signal check boxes (if desired).

This model is multirate because there are at least two distinct sample rates, as shown by the two Probe blocks. To run this model in Simulink's multitasking mode, select **Fixed-step** and **discrete** from the **Type** controls in the **Solver** panel of the **Simulation Parameters** dialog box, and select **MultiTasking** from the **Mode** parameter. Also set the **Stop time** to 30.

Run the model and look at the output, yout. The first few samples of each channel are shown below.

```
yout =
                   0                    0
                   0                    0
                   0                    0
                   0                    0
    0.00038576126099  -0.00038576126099
    0.00461423397064  -0.00461423397064
    0.00538575649261  -0.00538575649261
    0.00961422920227  -0.00961422920227
    0.01038587093353  -0.01038587093353
```

Since we ran this frame-based multirate model in multitasking mode, the first four (Mi) output rows are zero. The first filtered input matrix row appears in the output as sample 5 (i.e., sample Mi+1). Every second row is an interpolated value.

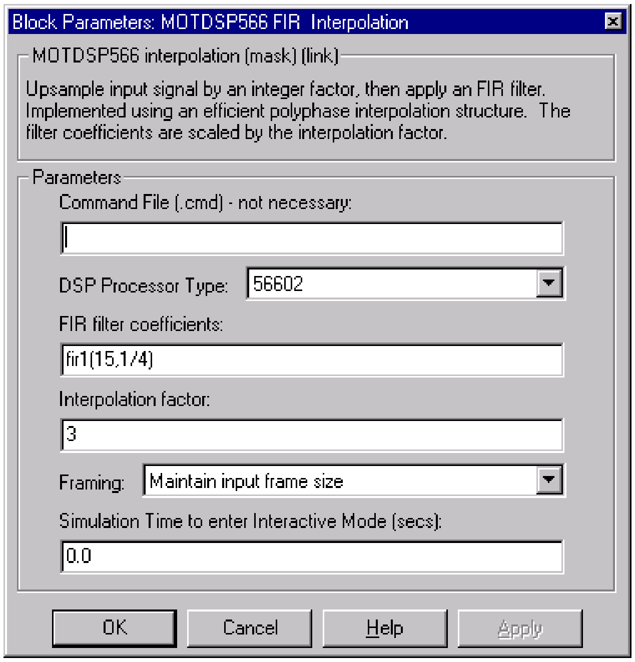
The filter coefficient vector generated by fir1(3,0.25) is

[0.0386 0.4614 0.4614 0.0386]

or, equivalently.

- $H(z) = B(z) = 0.0386 + 0.04614z^{-1} + 0.04614z^{-2} + 0.0386z^{-3}$

Parameters and Dialog Box


Block Parameters: MOTDSP566 FIR Interpolation

MOTDSP566 interpolation (mask) (link)

Upsample input signal by an integer factor, then apply an FIR filter. Implemented using an efficient polyphase interpolation structure. The filter coefficients are scaled by the interpolation factor.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

FIR filter coefficients: fir1(15,1/4)

Interpolation factor: 3

Framing: Maintain input frame size

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx.

DSP Processor Type

Specify the DSP processor to be used.

FIR filter coefficients

The FIR filter coefficients, in descending powers of z.

Interpolation factor

The integer factor, L, by which to increase the sample rate of the input sequence.

Frame-based inputs

Selects frame-based operation.

Number of channels

For frame-based operation, the number of columns (channels) in the input matrix, N.

Framing

For frame-based operation, the method by which to implement the interpolation: increase the output frame rate, or increase the output frame size.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP566 FIR Decimation

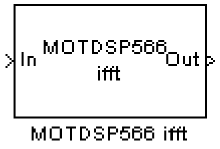
Purpose

Compute the IFFT of the input.

Library

Motdsp566lib.

Description

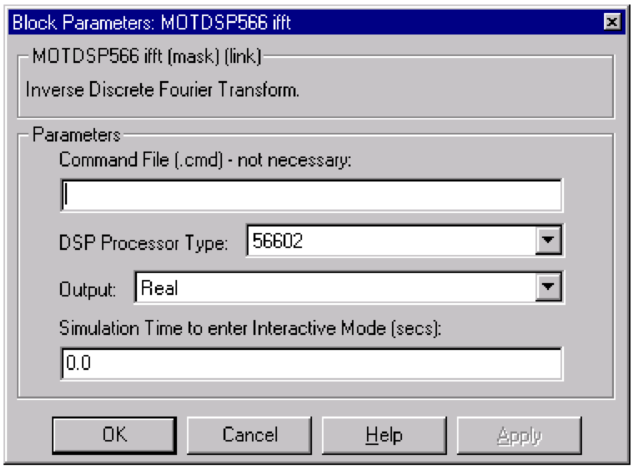


The MOTDSP566 IFFT block computes the inverse fast Fourier transform (IFFT) of each real or complex input channel independently at each sample time. The block assumes that the input is an M-by-N frame matrix. Each of the N frames in the matrix contains M sequential time samples from an independent signal.

For both sample-based and frame-based inputs, the block assumes that each input column is a frame containing M consecutive frequency-samples from an independent channel. The input must be complex, and the frame size, M, must be a power-of-two.

If the input is frame-based, the output is frame-based; otherwise, the output is sample-based. In either case, the output port rate is the same as the input port rate. For convenience, length-M 1-D vector inputs and *sample-based* length-M row vector inputs are processed as single channels (i.e., as M-by-1 column vectors), and the output has the same dimension as the input.

Parameters and Dialog Box


Block Parameters: MOTDSP566 ifft

MOTDSP566 ifft (mask) (link)

Inverse Discrete Fourier Transform.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Output: Real

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of channels

The number of channels (columns) in the input.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

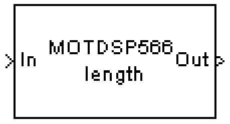
See Also

MOTDSP566 FFT

Purpose Get number of elements in a vector.

Library Motdsp566lib.

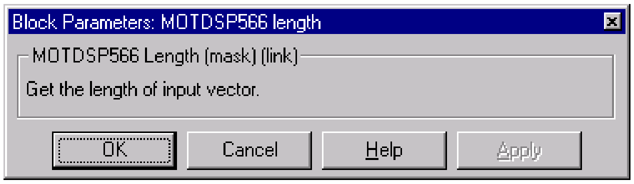
Description The MOTDSP566 Length block returns elements in a vector or the number of row in a matrix.



MOTDSP566 length

For length-M 1-D vector inputs or a *sample-based* length-M row vector inputs, the output is the number of element M; For the M-by-N full matrix inputs, the output is the row number M.

Parameters and Dialog Box



MOTDSP566 Length (mask) (link)

Get the length of input vector.

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

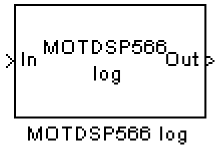
see also MOTDSP566 Convolution MOTDSP566 Correlation

MOTDSP566 Log

Purpose Perform a natural logarithm.

Library Motdsp566lib.

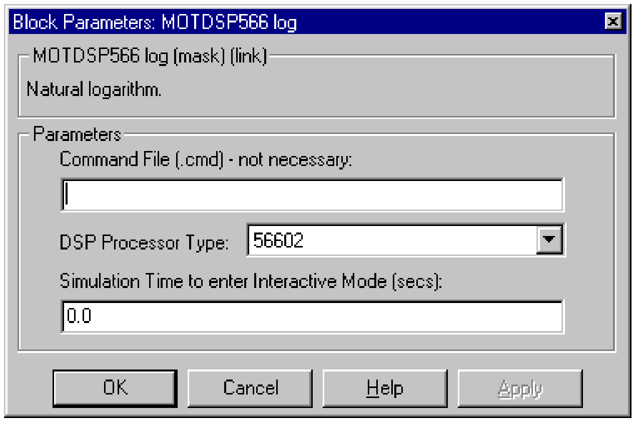
Description Perform a natural logarithm



Data Type Support

The MOTDSP566 Log block accepts complex or real-valued signals or signal vectors of type double. The output signal type depends on input signal type.

Parameters and Dialog Box



Block Parameters: MOTDSP566 log

MOTDSP566 log (mask) (link)

Natural logarithm.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP566 Log10 MOTDSP566 Sqrt

MOTDSP566 Log10

Purpose Perform a common (Base 10) logarithm.

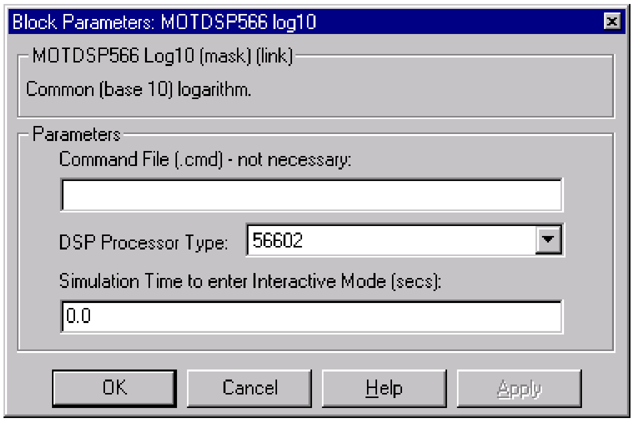
Library Motdsp566lib.

Description Perform a common (Base 10) logarithm.

In
MOTDSP566
log10
Out
MOTDSP566 log10

The MOTDSP566 Log10 block accepts complex or real-valued vectors of type double.The output signal type depends on input signal type.

Parameters and Dialog Box



MOTDSP566 Log10 (mask) (link)

Common (base 10) logarithm.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

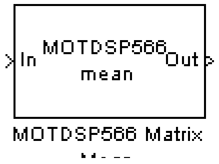
See Also MOTDSP566 Log MOTDSP566 Sqrt

MOTDSP566 Matrix Mean

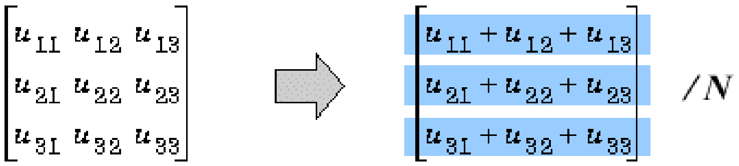
Purpose Mean the elements of a matrix along rows or columns.

Library Motdsp566lib.

Description The MOTDSP566 Matrix Mean block means the elements of an M-by-N input matrix u along either the rows or columns.



When the **Mean along** parameter is set to **Rows**, the block means across the elements of each row and outputs the resulting M-by-1 vector.



$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11}+u_{12}+u_{13} \\ u_{21}+u_{22}+u_{23} \\ u_{31}+u_{32}+u_{33} \end{bmatrix} / N$$

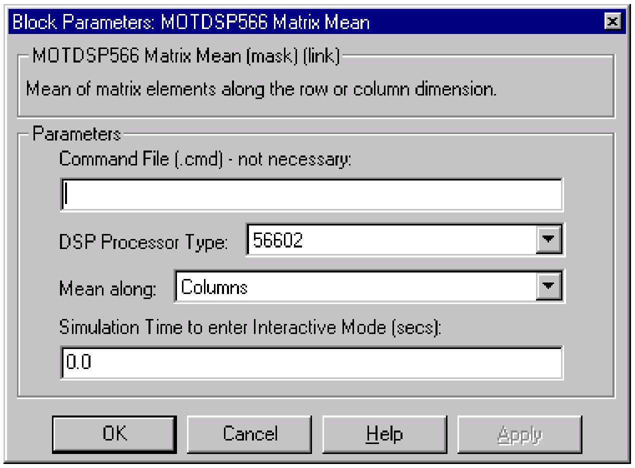
This is equivalent to

```
   y = mot566_mean(u,2)          % equivalent MATLAB code
```

When the **Mean along** parameter is set to **Columns**, the block means down the elements of each column and outputs the resulting 1-by-N vector. This is equivalent to

y = mot566_mean(u)   %equivalent MATLAB code

Parameters and Dialog Box



Block Parameters: MOTDSP566 Matrix Mean

MOTDSP566 Matrix Mean (mask) (link)

Mean of matrix elements along the row or column dimension.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Mean along: Columns

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of input ports

Number of input ports.

Number of columns in input

The number of columns in the input matrix.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP566 Matrix Sum

MOTDSP566 Matrix Sum

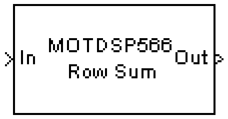
Purpose

Sum the elements of a matrix along rows or columns.

Library

Motdsp566lib.

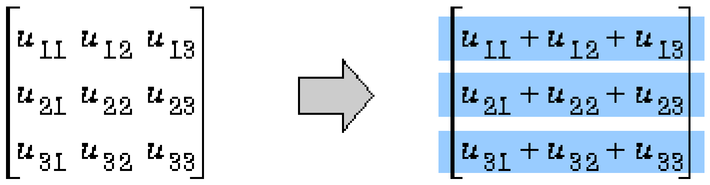
Description



MOTDSP566 Matrix Sum

The MOTDSP566 Matrix Sum block sums the elements of an M-by-N input matrix u along either the rows or columns.

When the **Sum along** parameter is set to **Rows**, the block sums across the elements of each row and outputs the resulting M-by-1 matrix. A length-N 1-D vector input is treated as a 1-by-N matrix..

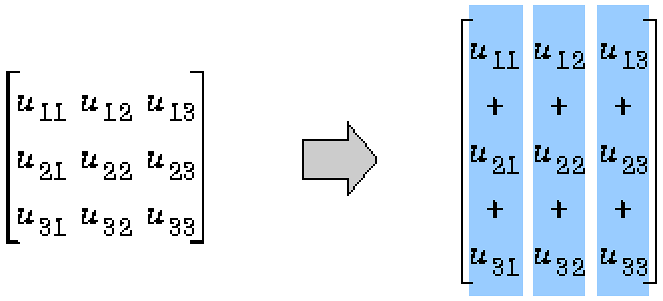


$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11}+u_{12}+u_{13} \\ u_{21}+u_{22}+u_{23} \\ u_{31}+u_{32}+u_{33} \end{bmatrix}$$

This is equivalent to

```
y = mot566_sum(u,2)          % equivalent MATLAB code
```

When the **Sum along** parameter is set to **Columns**, the block sums down the elements of each column and outputs the resulting 1-by-N matrix. A length-M 1-D vector input is treated as a M-by-1 matrix..



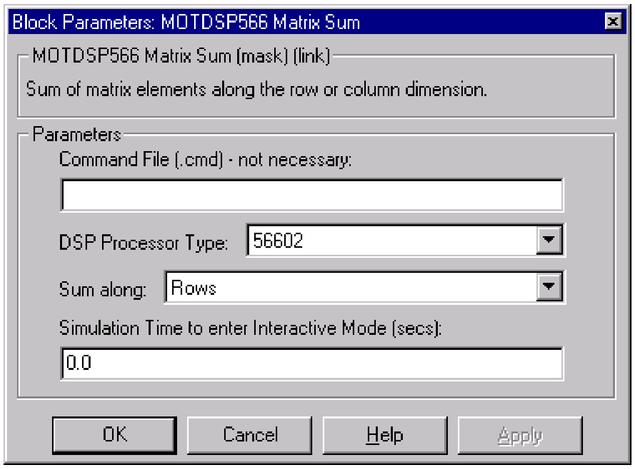
$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ + & + & + \\ u_{21} & u_{22} & u_{23} \\ + & + & + \\ u_{31} & u_{32} & u_{33} \end{bmatrix}$$

This is equivalent to

```
y = mot566_sum(u)            % equivalent MATLAB code
```

If the input is sample-based, the output is sample-based; if the input is frame-based, the output is frame-based.

Parameters and Dialog Box


Block Parameters: MOTDSP566 Matrix Sum

MOTDSP566 Matrix Sum (mask) (link)

Sum of matrix elements along the row or column dimension.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Sum along: Rows

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

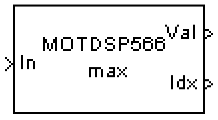
See Also

MOTDSP566 Matrix Mean

Purpose Find the maximum value of one or two input vector(s).

Library Motdsp566lib.

Description The MOTDSP566 Maximum block identifies the value and position of the largest element in the input.



MOTDSP566 max

If the block has two input vectors, the block performs an element-by-element comparison of the input vectors. Each element of the block output vector is the result of the comparison of the elements of the input vectors.

If the block has only one input vector, the **Mode** parameter specifies the block's mode of operation and can be set to **Value**, **Index**, or **Value and Index**. These settings are described below.

Value

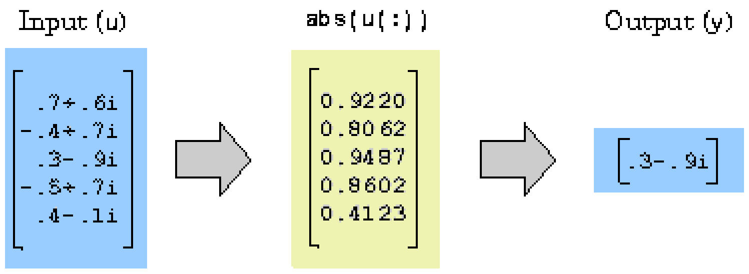
When **Mode** is set to **Value**, the block computes the maximum value in each column of the M-by-N input matrix u independently at each sample time.

[y,i] = mot566_max(u(:))   % equivalent MATLAB code

The block output, y, is the maximum value of the input vector.

For convenience, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors. The output at each sample time, val, is a sample-based 1-by-N length vector containing the maximum value of each column in u.

For complex inputs the block uses the magnitude of the input, abs(u(:)), to identify the maximum. The output is the corresponding complex value from the input. as shown below.



Index

When **Mode** is set to **Index**, the block performs the computation shown above, and outputs the index, i, corresponding to the position of the maximum value in the input vector. The index is an integer in the range [1 length(u(:))].

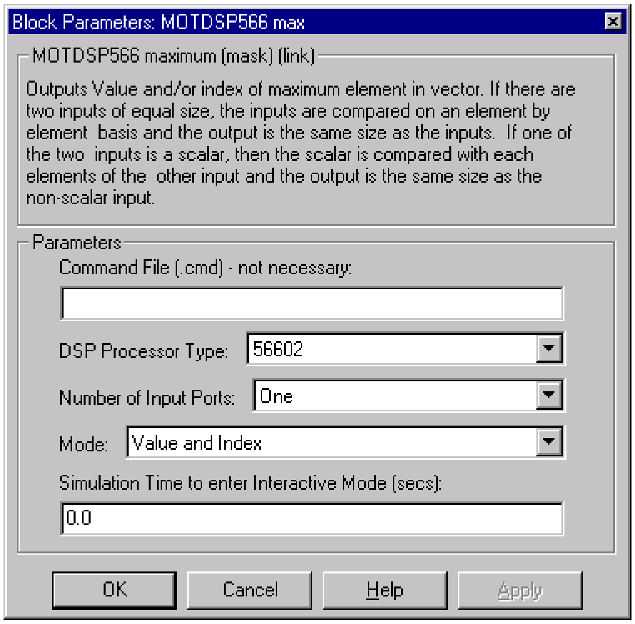
If there are duplicates of the maximum value in the input, the index corresponds to the first occurrence. For example, if the vector input is [.3.2 .1 .2.3], the index of the maximum value is 1, not 5.

Value and Index

When **Mode** is set to **Value and Index**, the block outputs both the value, y, and the index, i.

In all three of the above modes, a matrix input, u, is treated as a vector, u(:)

Parameters and Dialog Box


Block Parameters: MOTDSP566 max

MOTDSP566 maximum (mask) (link)

Outputs Value and/or index of maximum element in vector. If there are two inputs of equal size, the inputs are compared on an element by element basis and the output is the same size as the inputs. If one of the two inputs is a scalar, then the scalar is compared with each elements of the other input and the output is the same size as the non-scalar input.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Number of Input Ports: One

Mode: Value and Index

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of input ports

Number of input ports.

Mode

The block's mode of operation: Output the maximum value of each input, the index of the maximum value, both the value and the index.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

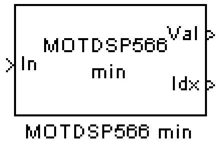
See Also

MOTDSP566 Minimum

Purpose Find the minimum value of one or two input vectors.

Library Motdsp566lib.

Description The MOTDSP566 Minimum block identifies the value and position of the smallest element in the input.



If the block has two input vectors, the block performs an element-by-element comparison of the input vectors. Each element of the block output vector is the result of the comparison of the elements of the input vectors.

If the block has only one input vector, the **Mode** parameter specifies the block's mode of operation and can be set to **Value**, **Index**, or **Value and Index**. These settings are described below.

Value Mode

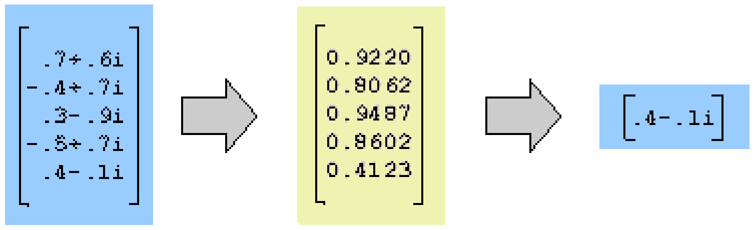
When **Mode** is set to **Value**, the block computes the minimum value of the M-by-N input matrix u independently at each sample time.

```
   [y,i] = mot566_MIN(u(:))        % equivalent MATLAB code
```

The block output, y, is the minimum value of the input vector.

For convenience, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors.

For complex inputs the block uses the magnitude of the input, abs(u(:)), to identify the minimum. The output is the corresponding complex value from the input. as shown below..



Index Mode

When Mode is set to Index, the block performs the computation shown above, and outputs the index, i, corresponding to the position of the minimum value in the input vector. The index is an integer in the range [1 length(u(:))].

If there are duplicates of the minimum value in the input, the index corresponds to the first occurrence.

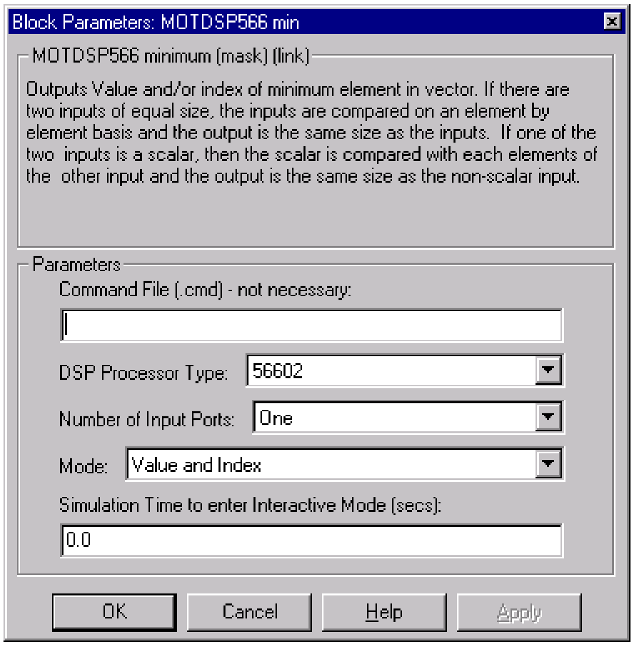
For example, if the vector input is [.1.2.3.2.1], the index of the minimum value is 1, not 5.

Value and Index Mode

When **Mode** is set to **Value and Index**, the block outputs both the vector of minima, val, and the vector of indices, idx.

In all three of the above modes, a matrix input, u, is treated as a vector, u(:).

Parameters and Dialog Box



Block Parameters: MOTDSP566 min

MOTDSP566 minimum (mask) (link)

Outputs Value and/or index of minimum element in vector. If there are two inputs of equal size, the inputs are compared on an element by element basis and the output is the same size as the inputs. If one of the two inputs is a scalar, then the scalar is compared with each elements of the other input and the output is the same size as the non-scalar input.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Number of Input Ports: One

Mode: Value and Index

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Number of input ports

Number of input ports.

Mode

The block's mode of operation: Output the maximum value of each input, the index of the maximum value, both the value and the index.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

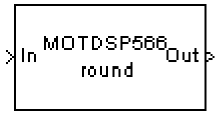
MOTDSP566 Maximum

MOTDSP566 Rounding

Purpose Perform common mathematical rounding.

Library Motdsp566lib.

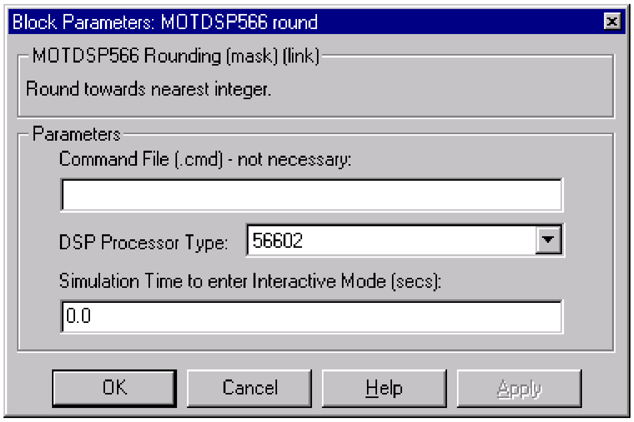
Description The MOTDSP566 Rounding Function block performs common mathematical rounding function.



MOTDSP566 round

The block accepts and output real- or complex-valued signals of type double.

Parameters and Dialog Box



Block Parameters: MOTDSP566 round

MOTDSP566 Rounding (mask) (link)

Round towards nearest integer.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File
Command file used by Motorola DSP566xx simulator core.

DSP Processor Type
Select the DSP Processor to be used.

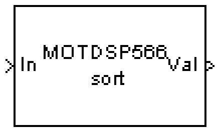
Simulation Time to enter Interactive Mode (secs)
Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP566 DSP Rounding MOTDSP566 Length MOTDSP566 Abs

Purpose Sort the elements in a vector by value.

Library Motdsp566lib.

Description



MOTDSP566 sort

The MOTDSP566 Sort block sorts the elements in a real or complex input vector by value using a Quick sort algorithm. The output vector, y, contains the input values arranged in order of ascending.

```
  [y,i] = sort(u(:))          % equivalent MATLAB code (ascending)
```

The **Mode** parameter specifies the block's output, and can be set to **Value**, **Index**, or **Value and Index**:

Value Mode

When Mode is set to Value, the block sorts the elements in each column of the M-by-N input matrix u in order of ascending, as specified by the Sort order parameter.

For convenience, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors.

The output at each sample time is a sample-based M-by-N matrix containing the sorted columns of u. Complex inputs are sorted by magnitude.

Index Mode

When Mode is set to Index, the block sorts the elements in each column of the M-by-N input matrix u. and outputs the sample-based M-by-N index matrix.

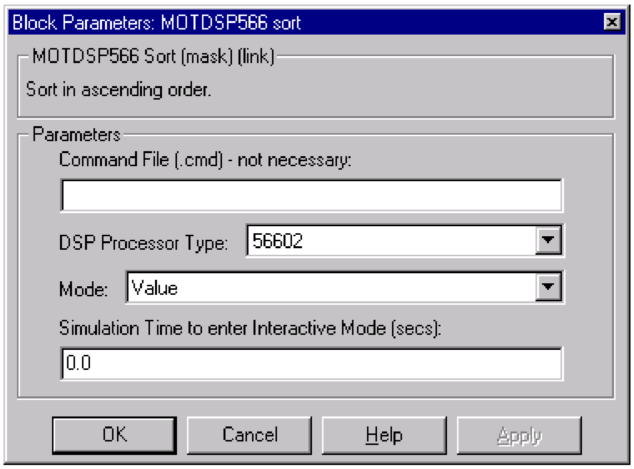
As in Value mode, length-M 1-D vector inputs and sample-based length-M row vector inputs are both treated as M-by-1 column vectors.

Value and Index Mode

When Mode is set to Value and Index, the block outputs both the sorted matrix, and the index matrix. Note that a matrix input is sorted as a single vector, u(:), rather than column by column.

MOTDSP566 Sort

Parameters and Dialog Box



Block Parameters: MOTDSP566 sort

MOTDSP566 Sort (mask) (link)

Sort in ascending order.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Mode: Value

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Mode

The block's mode of operation: Output the sorted vector, the index vector, or both.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP566 Convolution MOTDSO566 Correlation

Purpose Perform a Square Root.

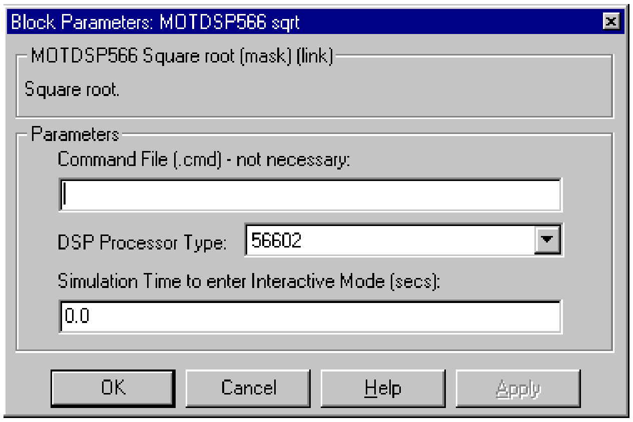
Library Motdsp566lib.

Description Perform a Square Root.

In
MOTDSP566
sqrt
Out
MOTDSP566 sqrt

The MOTDSP566 sqrt block accepts complex or real-valued signals or signal vectors of type double.The output signal type depends on input signal.

Parameters and Dialog Box



Block Parameters: MOTDSP566 sqrt

MOTDSP566 Square root (mask) (link)

Square root.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command File used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs):

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also MOTDSP566 Abs MOTDSO566 Log MOTDSO566 Log10

MOTDSP566 Correlation

Purpose Compute the correlation of two vectors.

Library Motdsp566lib.

Description

In1
MOTDSP566
corr
Out
In2

MOTDSP566 correlation

When both inputs are real, the output is real as well. When one or both inputs are complex, the output is complex.

Frame-Based Inputs

Matrix inputs must be frame-based. The output, y, is a frame-based (Mu+Mv-1)-by-N matrix whose column has elements:

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_{k,j} v^*_{(k+i-M_u),j} \qquad 1 \le i \le (M_u + M_v - 1$$

where * denotes the complex conjugate. Inputs u and v are zero when indexed outside of their valid ranges. When both inputs are real, the output is real; when one or both inputs are complex, the output is complex.

When one input is a column vector (single channel) and the other is a matrix (multiple channels), the single-channel input is independently cross-correlated with each channel of the multichannel input. For example, if u is a Mu-by-1 column vector and v is an Mv-by-N matrix, the output is an (Mu+Mv-1)-by-N matrix whose column has elements:

$$y_{i,j} = \sum_{k=1}^{\max(M_u, M_v)} u_k v^*_{(k+i-M_u),j} \qquad 1 \le i \le (M_u + M_v - 1$$

Sample-Based Inputs

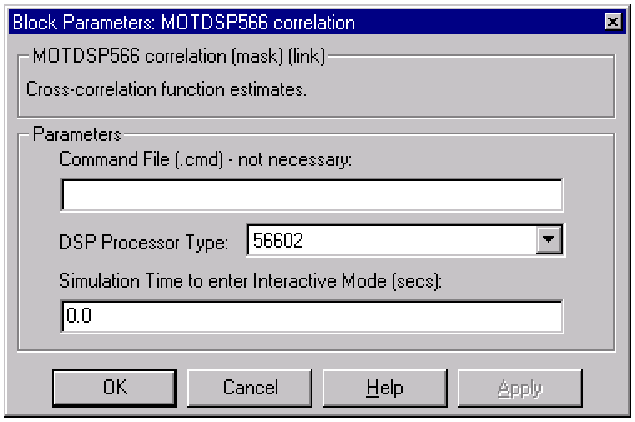
If u and v are sample-based vectors with lengths Mu and Mv, the Correlation block performs the vector cross-correlation.

$$y_i = \sum_{k=1}^{\max(M_u, M_v)} u_k v^*_{(k+i-M_u)} \qquad 1 \le i \le (M_u + M_v - 1$$

The dimensions of the sample-based output vector are determined by the dimensions of the input vectors:

The Correlation block does not accept sample-based row vector inputs, or mixed sample-based row vector and column vector inputs.

Parameters and Dialog Box


Block Parameters: MOTDSP566 correlation

MOTDSP566 correlation (mask) (link)

Cross-correlation function estimates.

Parameters

Command File (.cmd) - not necessary:

DSP Processor Type: 56602

Simulation Time to enter Interactive Mode (secs): 0.0

Command File

Command file used by Motorola DSP566xx simulator core.

DSP Processor Type

Select the DSP Processor to be used.

Simulation Time to enter Interactive Mode (secs)

Simulation time to step into the assembly code by launching the assembly language debugger. This parameter must be a scalar greater than or equal to zero. This parameter will be ignored if not using Interactive Mode.

See Also

MOTDSP566 Convolution

Directory Organization

Directory Organization

This is a description of the directory organization for the Motorola DSP Developer's Kit on Windows and UNIX, where `<matlab>` symbolizes the top-level directory in which MATLAB is installed on your system.

- **`<matlab>/toolbox/motdsp/motdspasm`**
  - `<matlab>/toolbox/motdsp/motdspasm/src/563_600`

    Motorola DSP 56300 and 56600 family assembly source for toolbox and blockset functions.
  - `<matlab>/toolbox/motdsp/motdspasm/bin`

    Assembly binaries for all DSP families.
- **`<matlab>/toolbox/motdsp/motdspblks`**

  Motorola DSP Blockset libraries.
- **`<matlab>/toolbox/motdsp/motdspdemos`**

  Motorola DSP Blockset demonstrations.
  - `<matlab>/toolbox/motdsp/motdspdemos/toolbox`

    Motorola DSP Toolbox demonstrations.
- **`<matlab>/toolbox/motdsp/motdspmasks`**

  Masks for the Motorola DSP Blocksets.
- **`<matlab>/toolbox/motdsp/motdsp`**

  Motorola DSP Toolbox help and binaries.
  - `<matlab>/toolbox/motdsp/motdsp/56300`
  - `<matlab>/toolbox/motdsp/motdsp/56600`

    Motorola DSP Toolbox MEX source files for each DSP family.
- **`<matlab>/toolbox/motdsp/motdspmex`**

  Motorola DSP Blockset S-function binaries.
  Build scripts, Options files and Suite56 simulator binaries.
  - `<matlab>/toolbox/motdsp/motdspmex/56300`
  - `<matlab>/toolbox/motdsp/motdspmex/56600`
    Motorola DSP Blockset S-function source files for each DSP family.
  - `<matlab>/toolbox/motdsp/motdspmex/include/headers/56k`
    Required header source files to build Motorola DSP MEX-files.

- `<matlab>/toolbox/motdsp/motdspmex/include/lib`
  Suite56 simulator export libraries.
- `<matlab>/toolbox/motdsp/motdspmex/templates`
  MATLAB MEX and Simulink S-MEX templates.

Index