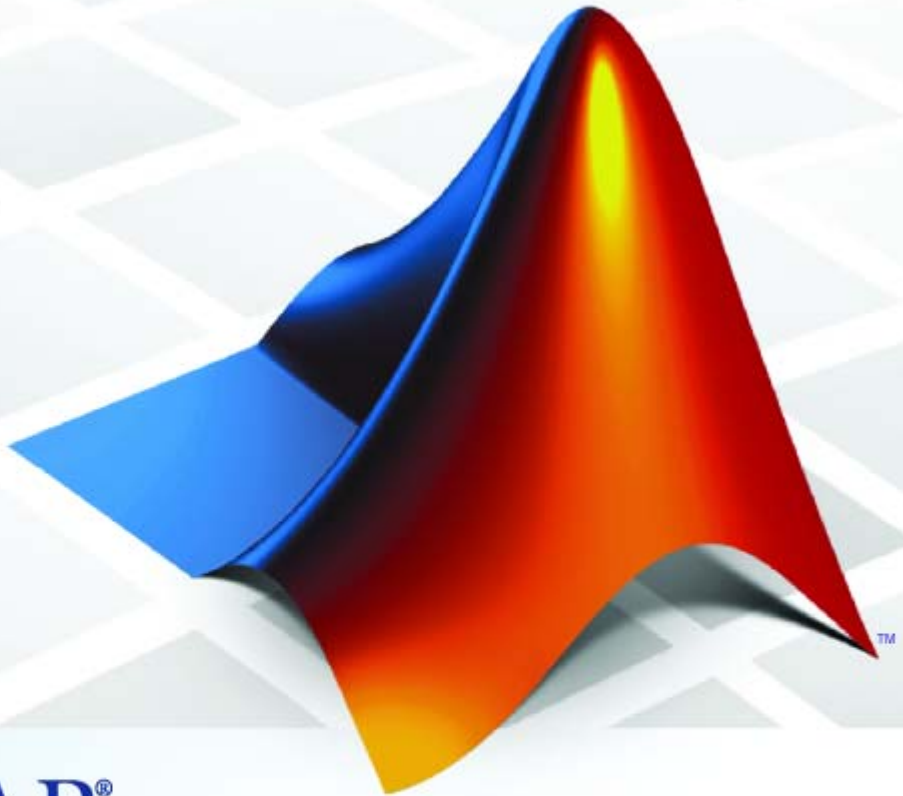


# Embedded IDE Link™ 4 User's Guide

*For Use with Analog Devices™ VisualDSP++®*



**MATLAB®**

## How to Contact The MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com) Product enhancement suggestions  
[bugs@mathworks.com](mailto:bugs@mathworks.com) Bug reports  
[doc@mathworks.com](mailto:doc@mathworks.com) Documentation error reports  
[service@mathworks.com](mailto:service@mathworks.com) Order status, license renewals, passcodes  
[info@mathworks.com](mailto:info@mathworks.com) Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.  
3 Apple Hill Drive  
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

*Using Embedded IDE Link™ with Analog Devices™ VisualDSP++®*

© COPYRIGHT 2007-2010 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

The MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

May 2007	Online only	New for Version 1.0 (Release 2007a+)
September 2007	Online only	Revised for Version 1.1 (Release 2007b)
March 2008	Online only	Revised for Version 2.0 (Release 2008a)
October 2008	Online only	Revised for Version 2.1 (Release 2008b)
March 2009	Online only	Revised for Version 2.2 (Release 2009a)
September 2009	Online only	Revised for Version 4.0 (Release 2009b)
March 2010	Online only	Revised for Version 4.1 (Release 2010a)



## Getting Started

### 1

<b>Product Overview</b> .....	1-2
<b>Software Structure and Components</b> .....	1-4
Automation Interface .....	1-4
Project Generator .....	1-5
Verification .....	1-5
<b>Software Requirements</b> .....	1-6
<b>Installation and Configuration</b> .....	1-7

## Automation Interface

### 2

<b>Getting Started with Automation Interface</b> .....	2-2
Introducing the Automation Interface Tutorial .....	2-2
Running the Interactive Tutorial .....	2-5
Selecting Your Session and Processor .....	2-6
Querying Objects for VisualDSP++ IDE .....	2-7
Loading Files into VisualDSP++ IDE .....	2-9
Running the Project .....	2-11
Working with Global Variables and Memory .....	2-12
Working with Local Variables and Memory .....	2-13
Closing Files and Projects .....	2-16
Closing the Connections or Cleaning Up VisualDSP++ Software .....	2-16
Tutorial Summary .....	2-17
<b>Constructing Objects</b> .....	2-18
Example — Constructor for adivdsp Objects .....	2-18

<b>Properties and Property Values</b> .....	<b>2-20</b>
Setting and Retrieving Property Values .....	<b>2-20</b>
Setting Property Values Directly at Construction .....	<b>2-21</b>
Setting Property Values with set .....	<b>2-21</b>
Retrieving Properties with get .....	<b>2-22</b>
Direct Property Referencing to Set and Get Values .....	<b>2-22</b>
Overloaded Functions for adivdsp Objects .....	<b>2-23</b>
<b>adivdsp Object Properties</b> .....	<b>2-24</b>
Quick Reference to adivdsp Properties .....	<b>2-24</b>
Details About adivdsp Object Properties .....	<b>2-25</b>

## Project Generator

### 3

<b>Introducing Project Generator</b> .....	<b>3-2</b>
<b>Project Generator Tutorial</b> .....	<b>3-3</b>
Building the Model .....	<b>3-3</b>
Adding the Target Preferences Block to Your Model .....	<b>3-4</b>
Specifying Simulink Configuration Parameters for Your Model .....	<b>3-8</b>
<b>Model Reference</b> .....	<b>3-11</b>
How Model Reference Works .....	<b>3-11</b>
Using Model Reference .....	<b>3-12</b>
Configuring Targets to Use Model Reference .....	<b>3-14</b>

## Block Reference

### 4

<b>Block Library: idelinklib_adivdsp</b> .....	<b>4-2</b>
--	------------

## Blocks — Alphabetical List

---

**5**

### Reported Limitations and Tips

---

**A**

<b>Reported Issues</b> .....	<b>A-2</b>
Using 64-bit Symbols in a 64-bit Memory Section on SHARC Processors .....	<b>A-2</b>

### Supported Processors

---

**B**

<b>Supported Platforms</b> .....	<b>B-2</b>
Product Features Supported by Each Processor or Family .....	<b>B-2</b>
Supported Processors and Simulators .....	<b>B-2</b>
Custom Board Support .....	<b>B-3</b>

### Index

---





# Getting Started

---

- “Product Overview” on page 1-2
- “Software Structure and Components” on page 1-4
- “Software Requirements” on page 1-6
- “Installation and Configuration” on page 1-7

## Product Overview

Embedded IDE Link™ software provides a connection between MATLAB® and the VisualDSP++® IDE to enable you to access the processor from MATLAB. You can, manipulate data on the processor, and manage projects within the IDE, while simultaneously utilizing the MATLAB tools of numerical analysis and simulation. Using Embedded IDE Link software, you can perform the following tasks, and others related to Model-Based Design:

- Function calls — Write scripts in MATLAB software to execute any function in the VisualDSP++ IDE
- Automation — Write automated tests in MATLAB software to be executed on your processor, including control and verification operations
- Host-Processor Communication — Communicate with the processor directly from MATLAB software, without going to the IDE
- Verification and Validation
  - Load and execute projects into the VisualDSP++ IDE from the MATLAB command line
  - Build and compile code, and then use vectors of test data and parameters to test the code
  - Build and compile your code, and then download the code to the processor and execute it
- Design models — Design models and algorithms in MATLAB and Simulink® software and run them on the processor
- Generate code— Generate executable code for your processor directly from the models designed in Simulink software, and execute it

Embedded IDE Link software connects MATLAB software and Simulink software with Analog Devices™ VisualDSP++® integrated development and debugging environment from Analog Devices™. Embedded IDE Link software enables you to use MATLAB and Simulink software to debug and verify embedded code running on all Analog Devices DSPs that VisualDSP++ software supports, such as the Analog Devices™ Blackfin®, Analog Devices™ SHARC® and Analog Devices™ TigerSHARC® processor families.

Embedded IDE Link software includes a project generator component. With the project generator component, you can generate a complete project for the VisualDSP++ IDE from your Simulink software models, including ANSI<sup>®</sup> C code generated with Real-Time Workshop<sup>®</sup> software. Thus, you use the Real-Time Workshop and Real-Time Workshop<sup>®</sup> Embedded Coder<sup>™</sup> software to generate generic ANSI C code projects for VisualDSP++ software from models. You can then build and run these projects on Blackfin<sup>®</sup>, SHARC<sup>®</sup>, and TigerSHARC<sup>®</sup> processors.

The following list suggests some of the uses for the capabilities of the software:

- Create test benches in MATLAB and Simulink software for testing your manually written or automatically generated code running on ADI DSPs
- Generate code and project files for VisualDSP++ software from Simulink models for rapid prototyping or deployment of a system or application
- Build, debug, and verify embedded code on ADI DSPs
- Perform processor-in-the-loop (PIL) testing of embedded code

## Software Structure and Components

In this section...
“Automation Interface” on page 1-4
“Project Generator” on page 1-5
“Verification” on page 1-5

Embedded IDE Link software comprises components—the Automation Interface component, the Project Generation component, and the Verification component. The Automation Interface component enables communication between MATLAB software and Embedded IDE Link software. The Project Generation component leverages Simulink software and lets you build models, simulate them, and generate code from the models directly to the processor.

The Verification component offers capabilities that help you use Model-Based Design to validate and verify your projects. With the Verification component, you can simulate algorithms and processes in Simulink models and concurrently on your processor. Comparing the results helps verify the fidelity of your model or algorithm code.

### Automation Interface

The Automation Interface component allows you to use Embedded IDE Link functions and methods to communicate with the VisualDSP++ IDE to perform the following tasks:

- Automate project management
- Debug programs
- Manipulate the data in the processor internal and external memory, and in the registers
- Communicate between the host and processor applications

The Debug Component of automation interface includes methods and functions for project automation, debugging, and data manipulation.

## Project Generator

The Project Generator component comprises methods that utilize the VisualDSP++ API to create projects in VisualDSP++ software and generate code with Real-Time Workshop and Real-Time Workshop Embedded Coder software. With the interface, you can do the following:

- Automatic project-based build process — Automatically create and build projects for code generated by Real-Time Workshop or Real-Time Workshop Embedded Coder software.
- Custom code generation — Use Embedded IDE Link software with any Real-Time Workshop System Target File (STF) to generate processor-specific and optimized code.
- Automatic downloading and debugging — Debug generated code in the VisualDSP++ debugger, using either the instruction set simulator or real hardware.
- Create and build projects for VisualDSP++ software from Simulink models — Project Generator uses Real-Time Workshop or Real-Time Workshop Embedded Coder software to build projects that work with Analog Devices processors.
- Generate custom code using the Configuration Parameters in your model with the system target files `vdsplink_ert.tlc` and `vdsplink_grt.tlc`.

## Verification

Verifying your processes and algorithms is an essential part of developing applications. The components of Embedded IDE Link software combine to provide the following verification tools for you to apply as you develop your code:

### Processor-in-the-Loop Cosimulation

Use cosimulation techniques to verify generated code running in an instruction set simulator or real hardware environment.

### Task Execution and Stack Usage Profiling

Gather execution profiling measurements with VisualDSP++ instruction set simulator to establish the timing requirements of your algorithm. Also, verify the stack usage is appropriate and as expected.

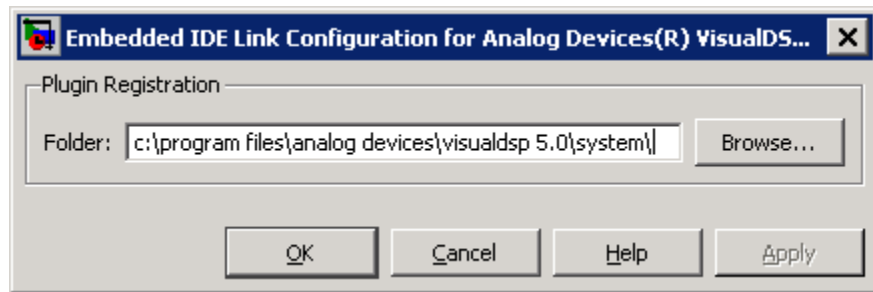
## Software Requirements

For detailed information about the software and hardware required to use Embedded IDE Link software, refer to the Embedded IDE Link system requirements areas on the MathWorks Web site:

- Requirements for Embedded IDE Link:  
[www.mathworks.com/products/ide-link/requirements.html](http://www.mathworks.com/products/ide-link/requirements.html)
- Requirements for use with VisualDSP++:  
[www.mathworks.com/products/ide-link/adi-adaptor.html](http://www.mathworks.com/products/ide-link/adi-adaptor.html)

## Installation and Configuration

- 1 Install VisualDSP++ according to the instructions provided with that software.
- 2 Enter `adivdspsetup` on the MATLAB command line.
- 3 Use **Browse** to locate the system folder for Analog Devices VisualDSP++. This action registers the Embedded IDE Link with that IDE.



- 4 Confirm that the installation works by entering `IDE_Obj = adivdsp` on the MATLAB command line. This action creates an IDE handle object for VisualDSP++ in MATLAB, and starts VisualDSP++.





# Automation Interface

---

- “Getting Started with Automation Interface” on page 2-2
- “Constructing Objects” on page 2-18
- “Properties and Property Values” on page 2-20
- “adivdsp Object Properties” on page 2-24

## Getting Started with Automation Interface

In this section...
“Introducing the Automation Interface Tutorial” on page 2-2
“Running the Interactive Tutorial” on page 2-5
“Selecting Your Session and Processor” on page 2-6
“Querying Objects for VisualDSP++ IDE” on page 2-7
“Loading Files into VisualDSP++ IDE” on page 2-9
“Running the Project” on page 2-11
“Working with Global Variables and Memory ” on page 2-12
“Working with Local Variables and Memory” on page 2-13
“Closing Files and Projects” on page 2-16
“Closing the Connections or Cleaning Up VisualDSP++ Software” on page 2-16
“Tutorial Summary” on page 2-17

### Introducing the Automation Interface Tutorial

Embedded IDE Link software provides a connection between MATLAB software and a processor in VisualDSP++ software. You can use objects as a mechanism to control and manipulate a signal processing application using the computational power of MATLAB software. This approach can help you while you debug and develop your application. Another possible use for automation is creating MATLAB scripts that verify and test algorithms that run in their final implementation on your production processor.

---

**Note** Before using the functions available with the objects, you must select a session in the VisualDSP++ IDE. The object you create is specific to a designated session in VisualDSP++ IDE.

---

To get you started using objects for VisualDSP++ software, Embedded IDE Link software includes an example script `vdspautointtutorial.m`. As you

work through this tutorial, you perform the following tasks that step you through creating and using objects for VisualDSP++ IDE.

- 1 Select your session.
- 2 Create and query objects to VisualDSP++ IDE.
- 3 Use MATLAB software to load files into VisualDSP++ software IDE.
- 4 Work with your VisualDSP++ IDE project from MATLAB software.
- 5 Close the connections you opened to VisualDSP++ IDE.

You use these tasks in any development work you do with signal processing applications. Thus, the tutorial provided here gives you a working process and best practice for using Embedded IDE Link software and your signal processing programs to develop programs for a range of Analog Devices processors.

The tutorial covers some methods and functions for Embedded IDE Link software. The functions listed first do not require an `adivdsp` object. The functions listed after that require an existing `adivdsp` object before you can use the function syntax.

### Functions for Working with VisualDSP++ Software

The following table shows functions that do not require an object.

Function	Description
<code>listsessions</code>	Return information about the boards that VisualDSP++ IDE recognizes as installed on your PC.
<code>adivdsp</code>	Construct an object that refers to a VisualDSP++ IDE session. When you construct the object you specify the session by processor.

## Methods for Working with adivdsp Objects in VisualDSP++ Software

The following table presents some of the methods that require an adivdsp object.

<b>Methods</b>	<b>Description</b>
add	Add a file to a project
address	Return the address and page for an entry in the symbol table in VisualDSP++ IDE
build	Build the project in VisualDSP++ software
cd	Change the working directory
display	Display the properties of an object that references a VisualDSP++ software session
halt	Terminate execution of a process running on the processor
info	Return information about the object or session
isrunning	Test whether the processor is executing a process
load	Load a built project to the processor
open	Open a file in the project
read	Retrieve data from memory on the processor
reset	Restore the program counter (PC) to the entry point for the current program
run	Execute the program loaded on the processor
save	Save files or projects
visible	Set whether VisualDSP++ IDE window is visible on the desktop while VisualDSP++ IDE is running
write	Write data to memory on the processor

## Running VisualDSP++ Software on Your Desktop – Visibility

When you create an `adivdsp` object in the tutorial in the next section, Embedded IDE Link starts VisualDSP++ software in the background.

If VisualDSP++ software is running in the background, it does not appear on your desktop, in your task bar, or on the **Applications** page in the Task Manager. It does appear as a process, `idde.exe`, on the **Processes** tab in Task Manager.

You can make the VisualDSP++ IDE visible with the function `visible`. The function `isvisible` returns the status of the IDE—is it visible on your desktop. To close the IDE when it is not visible and MATLAB is not running, use the **Processes** tab in Windows® Task Manager and look for `idde.exe`.

If an object that refers to VisualDSP++ software exists when you close VisualDSP++ software, the application does not close. Windows software moves it to the background (it becomes invisible). Only after you clear all objects that access VisualDSP++ IDE, or close MATLAB, does closing VisualDSP++ unload the application. You can see if VisualDSP++ IDE is running in the background by checking in the Windows Task Manager. When VisualDSP++ IDE is running, the entry `idde.exe` appears in the **Image Name** list on the **Processes** tab.

## Running the Interactive Tutorial

You have the option of running this tutorial from the MATLAB command line or entering the functions as described in the following tutorial sections.

To run the tutorial in MATLAB, click `run vdspautointtutorial`. This command launches the tutorial in an interactive mode where the tutorial program provides prompts and text descriptions to which you respond to move to the next section. The interactive tutorial covers the same information provided by the following tutorial sections. You can view the tutorial MATLAB file used here by clicking `vdspautointtutorial.m`.

---

**Note** To run the interactive tutorial, you must have at least one session configured in VisualDSP++ software. If you do not yet have a session, use the Analog Devices VisualDSP++ Configurator to create a session to use for this tutorial.

---

## Selecting Your Session and Processor

Embedded IDE Link IDE requires that you have at least one session available for VisualDSP++ software. To help you select the session to use for this tutorial, and for any development work, Embedded IDE Link software provides a command line tool, called `listsessions`, which prints a list of the available sessions. So that you can use this function in a script, `listsessions` can return a MATLAB structure that you use when you want your script to select a session in the IDE without your help.

---

**Note** The session you select is used throughout the tutorial.

---

- 1 To see a list of the sessions that you can use, enter the following command at the MATLAB prompt:

```
session_list = listsessions
```

MATLAB returns a list that shows all the sessions that Embedded IDE Link IDE recognizes as available in your installation.

```
session_list =  
  
    'ADSP-21060 ADSP-2106x Simulator'  
    'ADSP-21362 ADSP-2136x Simulator'
```

- 2 `listsessions` has a verbose mode that provides further details about the sessions in a cell array. The array contains structures that describe each session—the target type, the platform, and the processor.

```
sessionsinfo = listsessions('verbose');  
  
echo off
```

```
sessionname: 'ADSP-21362 ADSP-2136x Simulator'  
targettype: 'ADSP-2136x Family Simulator'  
platformname: 'ADSP-2136x Simulator'  
processors: 'ADSP-21362'
```

- 3 Use `adivdsp` to create an object that accesses a session in VisualDSP++ IDE.

```
IDE_Obj = adivdsp('sessionname','ADSP-21362 ADSP-2136x Simulator','procnum',0)
```

`Sessionname` and `procnum` are property names that specify the property to set. `ADSP-21362 ADSP-2136x Simulator` is the session to access, and `0` is the number of the processor to refer to in the session.

When you use `adivdsp`, you create an object, in this case `IDE_Obj`, that refers to the session you specify in `sessionname`.

## Querying Objects for VisualDSP++ IDE

In this tutorial section you create the connection between MATLAB and VisualDSP++ IDE. This connection, or object, is a MATLAB object, which for this session you save as variable `IDE_Obj`. You use function `adivdsp` to create objects. When you create objects, `adivdsp` input arguments let you define other object properties, such as the global time-out. Refer to the `adivdsp` reference information for more about the input arguments.

Use the generated object `IDE_Obj` to direct actions to your session processor. In the following tasks, `IDE_Obj` appears in all function syntax that interact with IDE session and the processor: The object `IDE_Obj` identifies and refers to a specific session. You need to include the object in any method syntax you use to access and manipulate a project or files in a session in VisualDSP++ IDE.

- 1 Create an object that refers to your selected session and processor. Enter the following command at the prompt.

```
IDE_Obj = adivdsp('sessionname','ADSP-21362 ADSP-2136x Simulator','procnum',0)
```

If you watch closely, and your machine is not too fast, you see VisualDSP++ software appear briefly when you call `adivdsp`. If VisualDSP++ was not

running before you created the new object, VisualDSP++ software starts and runs in the background.

Usually, you need to interact with VisualDSP++ while you develop your application. The function `visible`, controls the state of VisualDSP++ software on your desktop. `visible` accepts Boolean inputs that make VisualDSP++ software either visible on your desktop (input to `visible`  $\geq 1$ ) or invisible on your desktop (input to `visible` = 0). For this tutorial, you need to interact with the development environment, so use `visible` to set the IDE visibility to 1.

- 2** To make VisualDSP++ IDE show on your desktop, enter the following command at the prompt:

```
visible(IDE_Obj,1)
```

- 3** Next, enter `display(IDE_Obj)` at the prompt to see the status information.

```
ADIVDSP Object:
  Session name      : ADSP-21362 ADSP-2136x Simulator
  Processor name    : ADSP-21362
  Processor type    : ADSP-21362
  Processor number  : 0
  Default timeout   : 10.00 secs
```

Embedded IDE Link software provides three methods to read the status of a processor:

- `info` — Return a structure of testable session conditions.
- `display` — Print information about the session and processor.
- `isrunning` — Return the state (running or halted) of the processor.

- 4** Type `procinfo = info(IDE_Obj)`.

The `IDE_Obj` link status information provides data about the hardware, as follows:

```
procinfo =

  procname: 'ADSP-21362'
  proctype: 'ADSP-21362'
```



```
revision: ''
```

- 5 Verify that the processor is running by entering

```
runstatus = isrunning(IDE_Obj)
```

MATLAB responds, indicating that the processor is stopped, as follows:

```
runstatus =
```

```
0
```

## Loading Files into VisualDSP++ IDE

In this part of the tutorial, you load the executable code for the CPU in the IDE. Embedded IDE Link software includes a tutorial project file for VisualDSP++ IDE. Through the next commands in the tutorial, you locate the tutorial project file and load it into VisualDSP++ IDE. The `open` method directs VisualDSP++ software to load a project file or workspace file.

---

**Note** To continue the tutorial, you must identify or create a directory to which you have write access. Embedded IDE Link software cannot create a directory for you. If you do not have a writable directory, create one in Windows software before you proceed with the rest of this tutorial.

---

VisualDSP++ software has its own workspace and workspace files that are quite different from MATLAB workspace files and the MATLAB workspace. Remember to monitor both workspaces. The next steps change the working directory to your new writable directory.

- 1 Use `cd` to switch to the writable directory

```
prj_dir = cd('C:\vdsp_demo')
```

where the name and path to the writable directory is a string, such as `C:\vdsp_demo` as used in the example. Replace `C:\vdsp_demo` with the full path to your directory.

- 2 Change your working directory to the new directory by entering the following command:

```
cd(IDE_Obj,prj_dir)
```

- 3 Next, use the following command to create a new VisualDSP++ software project named `dot_product.c.dpj` in the new directory:

```
new(IDE_Obj, 'debug_demo.dpj')
```

Look in the IDE to verify that your new project exists. Next you need to add source files to your project.

- 4 Add the provided source file—`scalarprod.c` to the project `debug_demo.dpj` using the following command:

```
add(IDE_Obj, [matlabroot '\toolbox\vdsp\link\vdspdemos\src\scalarprod.c'])
```

The variable *matlabroot* indicates the root directory of your MATLAB installation. Replace *matlabroot* with the path to MATLAB on your machine. For more information about the MATLAB root directory, refer to *matlabroot* in the MATLAB documentation.

- 5 Open the file in the IDE from MATLAB by issuing the following command to open the file:

```
open(IDE_Obj,[matlabroot '\toolbox\vdsp\link\vdspdemos\src\scalarprod.c'])
```

Switch to the IDE to verify that the files are in your project and open.

- 6 Save your project.

```
save(IDE_Obj, 'debug_demo.dpj', 'project')
```

Your IDE project is saved with the name `debug_demo.dpj` in your writable directory. The input string 'project' specifies that you are saving a project file.

## Running the Project

After you create `dot_project_c.dpj` in the IDE, you can use Embedded IDE Link functions to create executable code from the project and load the code to the processor.

The next steps in this tutorial build the executable and download and run it on your processor.

- 1 Use the following build command to build an executable module from the project `dot_product_c.dpj`.

```
build(IDE_Obj,30) % The optional input argument 30 sets the time out period to 30 seconds.
```

At the end of the build process, Embedded IDE Link software returns a value of 1 to indicate that the build succeeded. If the build process returns a 0, the build failed.

```
ans =
     1
```

- 2 To load the new executable to the processor, use `load` with the project file name and the object name. The name of the executable is `debug_demo.dxe`, and it is stored with the project in your writable directory, in a subdirectory named `debug`.

```
load(IDE_Obj, 'c:\vdsp_demo\debug\debug_demo.dxe', 30);
```

Embedded IDE Link software provides methods to control processor execution—`run`, `halt`, and `reset`. To demonstrate these methods, use `run` to start the program you loaded on the processor, and then use `halt` to stop the processor.

Try the following methods at the command prompt.

```
run(IDE_Obj)      % Start the program running on the processor.
halt(IDE_Obj)     % Halt the processor.
reset(IDE_Obj)    % Reset the program counter to start of program.
```

## Working with Global Variables and Memory

After you load your program on the processor, you can access memory locations and variables. You can then read variables either from the program symbol table or directly from addresses in memory. Three methods—`address`, `read`, and `write`, let you get, read, and write to and from your project and processor.

Start by getting the address of the global variable `v1` from the `debug_demo` project symbol table.

- 1 Enter the following command to retrieve the address for `v1`.

```
address_v1 = address(IDE_Obj, 'v1')

address_v1 =

    753666      1
```

- 2 Convert the address from decimal format to hexadecimal.

```
dec2hex(address_v1(1))

ans =

    B8002
```

The address of global data array `v1` is `0xB8002`, which is stored in type `1` memory on the processor

- 3 With the address of `v1` saved as `address_v1`, use `read` to return the data from that location. To specify the data type and the number of values to read, add the `datatype` (`'int32'`) and count (`32`) input arguments.

```
value_v1 = read(IDE_Obj, address_v1, 'int32', 32) % Interpret the data as 32-bit integers.

value_v1 =

    Columns 1 through 10

    -37    -133     31    -104     32     66    -123     19     140    -28

    Columns 11 through 20
```

```

    16    80    -2    83   -243   148    56   163    46    45

Columns 21 through 30

   -217    -11   -164    49    -3    99    21   -61   -26   101

Columns 31 through 32

   -101    -151

```

- 4** Repeat the read process for another global variable in the project—`v2`. Nest the `address` method inside the `read` method to reduce typing.

```

value_v2 = read(IDE_Obj,address(IDE_Obj,'v2'),'int32',32) % Read and address methods in one call.

value_v2 =

Columns 1 through 10

   -50     5   -17    28     5    31   -23   -156    68    -5

Columns 11 through 20

  -220     5   -14    57   214   183   213    40   175   144

Columns 21 through 30

   -12   -77   -18    77   130   -39   132   107    52   -59

Columns 31 through 32

   127    -117

```

## Working with Local Variables and Memory

If you examine the source files for `debug_demo` in the IDE, you can verify the values for `v1` and `v2` in the source file `scalarprod.c`. You can also use the `address` method to get the addresses of local variables on the stack, after the variable is in scope.

To get the variables in scope (on the stack), you run the program. Adding a breakpoint to the program allows you to read the stack contents when the program stops at the breakpoint. Without the breakpoint, the program runs to completion, and you cannot read the contents of the stack because it no longer exists.

Begin the process by adding a breakpoint to the project file `scalarprod.c`:

- 1** Insert a breakpoint on line 100 of program `scalarprod.c` with the following command:

```
insert(IDE_Obj, 'scalarprod.c', 100)
```

- 2** Run the program to add the variable to the stack, and move the program counter to the breakpoint. Add the optional input argument `timeout` sets the time out value to 30s instead of the default 20s value:

```
run(IDE_Obj, 'runtohalt', 30)
```

The program stops at the breakpoint on line 100.

- 3** Read the address of the local variable `result`, and convert it to its hexadecimal equivalent value.

```
address_result = address(IDE_Obj, 'result', 'local') % address_result is a 'local' variable.

address_result =

    933884         1

dec2hex(address_result(1))

ans =

    E3FFC
```

`address` returns 933884 as the location of `result` in memory, in type 1 memory on the processor, stored in the MATLAB variable `address_result`.

- 4** Use the variable `address_result` to get the value stored at that address by issuing the following `read` command:

```
actual_value_result = read(IDE_Obj, address_result, 'int32')  
  
actual_value_result =  
  
18875
```

Verify in the IDE Output Window that 18875 is the correct value for the dot product.

- 5 Use the following command to remove the breakpoint set on line 100.

```
remove(IDE_Obj, 'scalarprod.c', 100)
```

MATLAB includes a dot product function to use to verify the value in `actual_value_result`. Called `dot`, the function calculates the dot product of two input vectors. In this case, the inputs are vectors `value_v1` and `value_v2`.

Comparing the two results—`expected_value_result` in MATLAB with `actual_value_result` from the processor implementation validates your simulation and implementation. With Automation Interface methods, you can create MATLAB file scripts to test and verify algorithms in their implementation on a processor.

- 1 Calculate the expected result by performing the dot function with two input vectors.

```
expected_value_result = dot(value_v1, value_v2)  
  
expected_value_result =  
  
18875
```

- 2 Test to see if the actual and expected results match.

```
isequal(expected_value_result, actual_value_result)  
  
ans =  
  
1
```

- 3 After verifying the result and removing the breakpoint, run the program to completion, and then halt and reset the processor.

```
run(IDE_Obj)
halt(IDE_Obj)
reset(IDE_Obj)
```

### Closing Files and Projects

You can close files in your projects from the MATLAB command line. The method `close` works at the command line to close programs or projects in the IDE through the `adivdsp` object and input keywords that describe the kind of file to close.

To finish this tutorial, close the open documents or files in the IDE, and then close the project `debug_demo.dpj`.

- 1 Close all of the open files and documents in the IDE. All of the open files are text files, so use the `text` input argument.

```
close(IDE_Obj, 'all', 'text')
```

- 2 Now, close the project.

```
close(IDE_Obj, 'debug_demo.dpj', 'project')
```

---

**Note** If you close the VisualDSP++ IDE manually outside of MATLAB, clear the IDE handle object in MATLAB. For example, at the MATLAB command line enter:

```
clear IDE_Obj
```

---

### Closing the Connections or Cleaning Up VisualDSP++ Software

Objects that you create in Embedded IDE Link software have connections to VisualDSP++ software. Until you delete these handles, the VisualDSP++ process (`idde.exe` in the Windows Task Manager) remains in memory.



Closing MATLAB removes these objects automatically, but there may be times when it helps to delete the handles manually, without quitting MATLAB.

---

**Note** When you clear the last `adivdsp` IDE handle object, Embedded IDE Link software closes VisualDSP++ software. When it closes the IDE, the link software does not save current projects or files in the IDE, and it does not prompt you to save them. A best practice is to save all of your projects and files before you clear `adivdsp` objects from your MATLAB workspace.

---

- 1 Use the following command to make the IDE invisible if it is visible on your desktop.

```
visible(IDE_Obj.0)
```

- 2 To delete your connection to VisualDSP++ IDE, use `clear IDE_Obj`.

## Tutorial Summary

During the tutorial you performed the following tasks:

- 1 Selected your session.
- 2 Created and queried objects that refer to a session in Embedded IDE Link to get information about the session and processor.
- 3 Used MATLAB to load files into VisualDSP++ IDE, and used methods in MATLAB to run that file.
- 4 Accessed variables in the program symbol table and on the processor.
- 5 Used the Automation Interface methods to compare the results of a simulation in MATLAB with the same algorithm running on a processor.
- 6 Closed the files, projects, and connections you opened to VisualDSP++ IDE.

## Constructing Objects

When you create a connection to a session in VisualDSP++ software using the `adivdsp` function, you create an object. The object implementation relies on MATLAB object-oriented programming capabilities similar to the objects you find in MATLAB or Filter Design Toolbox.

The discussions in this section apply to the objects in Embedded IDE Link software. Because `adivdsp` objects use the MATLAB programming techniques, the information about working with the objects, such as how you get or set properties, or use methods, apply to the objects you create in Embedded IDE Link software.

Like other MATLAB structures, objects in Embedded IDE Link software have predefined fields referred to as *object properties*.

You specify object property values by one of the following methods:

- Specifying the property values when you create the object
- Creating an object with default property values, and changing some or all of these property values later

For examples of setting link properties, refer to “Setting Property Values with `set`.”

### Example – Constructor for `adivdsp` Objects

The easiest way to create an object is to use the function `adivdsp` to create an object with the default properties. Create an object named `IDE_Obj` referring to a session in VisualDSP++ software by entering the following syntax:

```
IDE_Obj = adivdsp
```

MATLAB responds with a list of the properties of the object `IDE_Obj` you created along with the associated default property values.

```
ADIVDSP Object:  
  Session name      : ADSP-21362 ADSP-2136x Simulator  
  Processor name    : ADSP-21362  
  Processor type    : ADSP-21362
```

```
Processor number : 0  
Default timeout  : 10.00 secs
```

The object properties are described in the `adivdsp` documentation.

---

**Note** These properties are set to default values when you construct links.

---

## Properties and Property Values

In this section...
“Setting and Retrieving Property Values” on page 2-20
“Setting Property Values Directly at Construction” on page 2-21
“Setting Property Values with set” on page 2-21
“Retrieving Properties with get” on page 2-22
“Direct Property Referencing to Set and Get Values” on page 2-22
“Overloaded Functions for adivdsp Objects” on page 2-23

Objects in this software have properties associated with them. Each property is assigned a value. You can set the values of most properties, either when you create the link or by changing the property value later. However, some properties have read-only values. Also, a few property values, such as the board number and the processor to which the link attaches, become read-only after you create the object. You cannot change those after you create your link.

### Setting and Retrieving Property Values

You can set adivdsp object property values by either of the following methods:

- Directly when you create the link — see “Setting Property Values Directly at Construction”
- By using the `set` function with an existing link — see “Setting Property Values with set”

Retrieve Embedded IDE Link software object property values with the `get` function.

Direct property referencing lets you either set or retrieve property values for adivdsp objects.

## Setting Property Values Directly at Construction

To set property values directly when you construct an object, include the following entries in the input argument list for the constructor method `adivdsp`:

- A string for the property name to set followed by a comma. Enclose the string in single quotation marks as you do any string in MATLAB.
- The associated property value. Sometimes this value is also a string.

Include as many property names in the argument list for the object construction command as there are properties to set directly.

### Example — Setting Object Property Values at Construction

Suppose that you want to create a link to a session in VisualDSP++ software and set the following object properties:

- Refer to the specified session.
- Connect to the first processor.
- Set the global time-out to 5 s. The default is 10 s.

Set these properties by entering

```
IDE_Obj = adivdsp('sessionname','ADSP-21060 ADSP-2106x Simulator','procnum',0,'timeout',5);
```

The `sessionname`, `procnum`, and `timeout` properties are described in Link Properties, as are the other properties for links.

## Setting Property Values with `set`

After you construct an object, the `set` function lets you modify its property values.

Using the `set` function, you can change the value of any writable property of an object.

### **Example — Setting Object Property Values Using set**

To set the time-out specification for the link IDE\_Obj from the previous section, enter the following syntax:

```
set(IDE_Obj, 'timeout', 8);  
  
get(IDE_Obj, 'timeout');  
ans =  
  
      8
```

The display reflects the changes in the property values.

### **Retrieving Properties with get**

You can use the get command to retrieve the value of an object property.

### **Example — Retrieving Object Property Values Using get**

To retrieve the value of the sessionname property for vd2, and assign it to a variable, enter the following syntax:

```
session = get(vd2, 'sessionname')  
  
session =  
  
ADSP-21060 ADSP-2106x Simulator
```

### **Direct Property Referencing to Set and Get Values**

You can directly set or get property values using MATLAB structure-like referencing. Do this by using a period to access an object property by name, as shown in the following example.

### **Example — Direct Property Referencing in Links**

To reference an object property value directly, perform the following steps:

- 1** Create a link with default values.
- 2** Change its time-out and number of open channels.

```
IDE_Obj = adivdsp;  
IDE_Obj.time = 6;
```

## Overloaded Functions for adivdsp Objects

Several methods and functions in Embedded IDE Link software have the same name as functions in other MathWorks™ products. These functions behave similarly to their original counterparts, but you apply them to an object. This concept of having functions with the same name operate on different types of objects (or on data) is called *overloading* of functions.

For example, the `set` command is overloaded for objects. After you specify your object by assigning values to its properties, you can apply the methods in this toolbox (such as `address` for reading an address in memory) directly to the variable name you assign to your object. You do not have to specify your object parameters again.

For a complete list of the methods that act on `adivdsp` objects, refer to the in the function reference pages.

## **adivdsp Object Properties**

<b>In this section...</b>
“Quick Reference to adivdsp Properties” on page 2-24
“Details About adivdsp Object Properties” on page 2-25

Embedded IDE Link software provides links to your processor hardware so you can communicate with processors for which you are developing systems and algorithms. Because Embedded IDE Link software uses objects to create the links, the parameters you set are called properties and you treat them as properties when you set them, retrieve them, or modify them.

This section details the properties for the objects for VisualDSP++ software. First the section provides tables of the properties, for quick reference. Following the tables, the section offers in-depth descriptions of each property, its name and use, and whether you can set and get the property value associated with the property. Descriptions include a few examples of the property in use.

MATLAB users may find much of this handling of objects familiar. Objects in Embedded IDE Link software behave like objects in MATLAB and the other object-oriented toolbox products. C++ programmers may already understand the concepts described in this section.

### **Quick Reference to adivdsp Properties**

The following table lists the properties for the links in Embedded IDE Link software. The second column indicates the object to which the property belongs. Knowing which property belongs to each object tells you how to access the property.



Property Name	User Settable?	Description
sessionname	At construction only	Reports the name of the session in VisualDSP++ IDE that the object references.
procnum	At construction only	Stores the number of the processor in the session. If you have more than one processor, this number identifies the specific processor.
timeout	Yes/default	Contains the global time-out setting for the link.

Some properties are read only. Thus, you cannot set the property value. Other properties you can change at any time. If the entry in the User Settable column is “At construction only”, you can set the property value only when you create the object. Thereafter it is read only.

## Details About adivdsp Object Properties

To use the objects for VisualDSP++ interface, set values for the following:

- `sessionname` — Specify the session with which the object interacts.
- `procnum` — Specify the processor in the session. If the board has multiple processors, `procnum` identifies the processor to use.
- `timeout` — Specify the global time-out value. (Optional. Default is 10 s.)

Details of the properties associated with adivdsp objects appear in the following sections, listed in alphabetical order by property name.

### **procnum**

Property `procnum` identifies the processor referenced by an object for Embedded IDE Link IDE. Use `procnum` to specify the processor you are working with in the session specified by `sessionname`. The VisualDSP++ Configurator assigns a number to each processor installed in each session. To determine the value of `procnum` for a processor, use `listsessions` or the Configurator.

To identify a processor, you need the `sessionname` and `procnum` values. For sessions with one processor, `procnum` equals 0. VisualDSP++ IDE numbers the processors on multiprocessor boards sequentially from 0 to the total number of processors. For example, on a board with four processors, the processors are numbered 0, 1, 2, and 3.

### **sessionname**

Property `sessionname` identifies the session referenced by a Embedded IDE Link software. When you create an object, you use `sessionname` to specify the session you are intending to interact with. To get the value for `sessionname`, use `listsessions` or the Analog Devices VisualDSP++ Configurator. The Configurator utility assigns the name for each session available on your system.

### **timeout**

Property `timeout` specifies how long VisualDSP++ software waits for any process to finish. You set the global time-out when you create an object for a session in VisualDSP++ IDE. The default global time-out value 10 s. The following example shows the `timeout` value for object `vd2`.

```
display(vd2)
```

```
ADIVDSP Object:
```

```
Session name      : ADSP-21060 ADSP-2106x Simulator  
Processor name    : ADSP-21060  
Processor type    : ADSP-21060  
Processor number  : 0  
Default timeout   : 10.00 secs
```

# Project Generator

---

- “Introducing Project Generator” on page 3-2
- “Project Generator Tutorial” on page 3-3
- “Model Reference” on page 3-11

## Introducing Project Generator

Project generator provides the following features for developing projects and generating code:

- Automated project building for VisualDSP++ software that lets you create VisualDSP++ software projects from code generated by Real-Time Workshop and Real-Time Workshop Embedded Coder software. Project generator populates projects in the VisualDSP++ software development environment.
- Blocks in the library `idelinklib_adivdsp` for controlling the scheduling and timing in generated code.
- Highly configurable code generation using model configuration parameters and target preferences block options.
- Capability to use Embedded IDE Link software with one of two system target files to generate code specific to your processor.
- Highly configurable project build process.
- Automatic downloading and running of your generated projects on your processor.

To configure your Simulink software models to use the Project Generator component, do one or both of the following tasks:

- Add a Target Preferences block from the `idelinklib_adivdsp` library to the model.
- To use the asynchronous scheduler capability in Embedded IDE Link software, add one or more hardware interrupt blocks or idle task block from the `idelinklib_adivdsp` library.

The following sections describe the blockset and the blocks in it, the scheduler, and the Project Generator component.

# Project Generator Tutorial

**In this section...**

“Building the Model” on page 3-3

“Adding the Target Preferences Block to Your Model” on page 3-4

“Specifying Simulink Configuration Parameters for Your Model” on page 3-8

In this tutorial you build a model and generate a project from the model into VisualDSP++ IDE.

---

**Note** The model demonstrates project generation only. You cannot build and run the model on your processor without additional blocks.

---

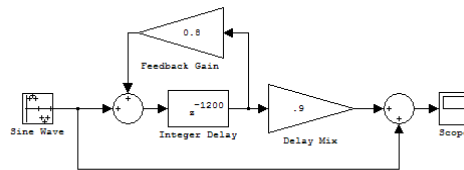
To generate a project from a model, complete the following tasks:

- 1** Use Simulink blocks, Signal Processing Blockset™ blocks, and blocks from other blocksets to create the model application.
- 2** Add the target preferences block from the Embedded IDE Link Target Preferences library to your model. Verify and set the block parameters for your hardware. In most cases, the default settings work fine.
- 3** Set the configuration parameters for your model, including the following parameters:
  - Solver parameters such as simulation start and solver options
  - Real-Time Workshop software options such as processor configuration and processor compiler selection
- 4** Generate your project.
- 5** Review your project in VisualDSP++ software.

## Building the Model

To build the model for audio reverberation, follow these steps:

- 1 Start Simulink software.
- 2 Create a new model by selecting **File > New > Model** from the **Simulink** menu bar.
- 3 Use Simulink blocks and Signal Processing Blockset blocks to create the following model.



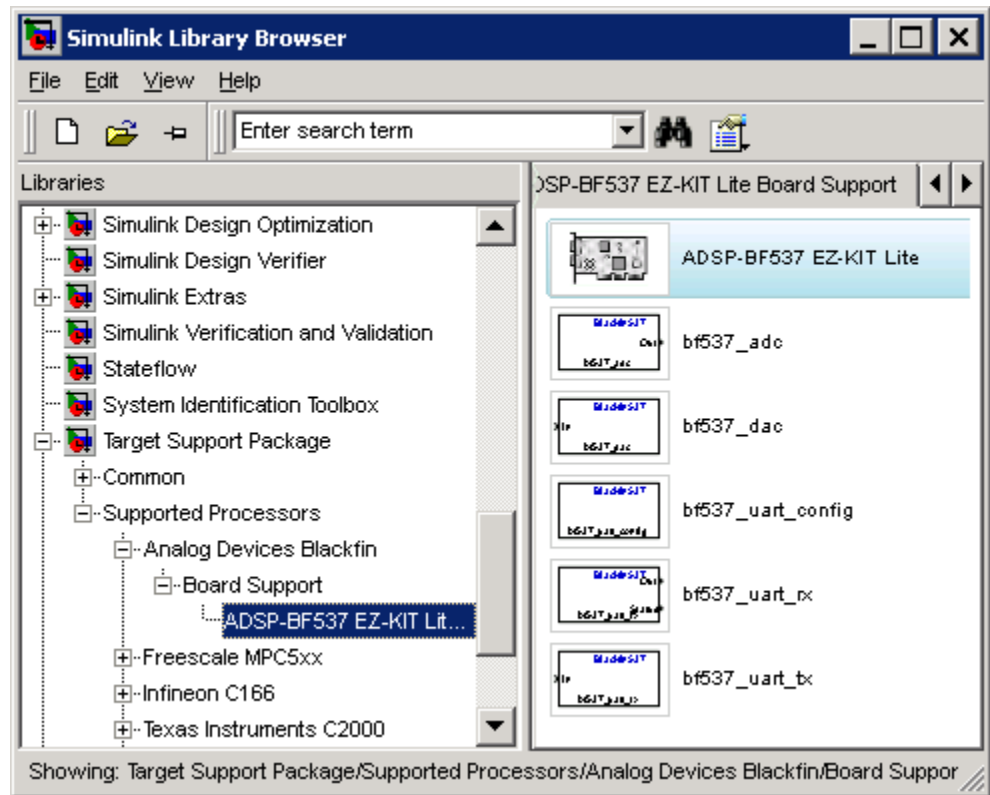
Look for the Integer Delay block in the Discrete library of Simulink and the Gain block in the Commonly Used Blocks library. Do not add the Custom Board block at this time.

- 4 Save your model with a suitable name before continuing.

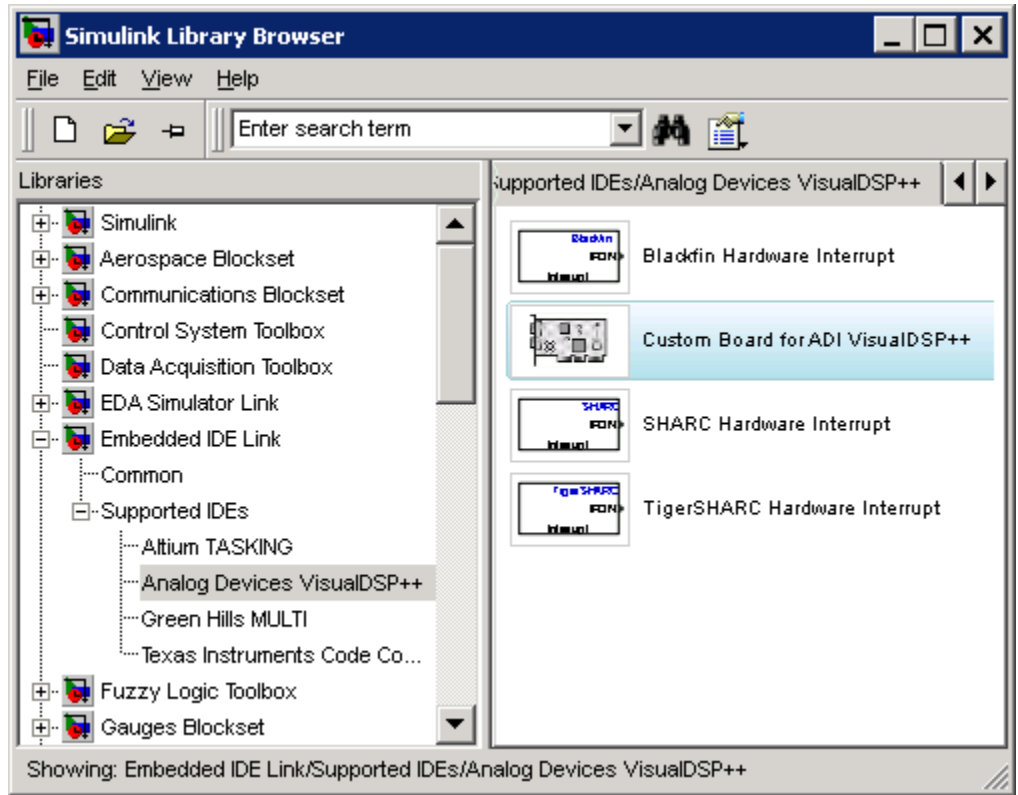
## **Adding the Target Preferences Block to Your Model**

To configure your model to work with Analog Devices processors, add a target preferences block to your model.

If you have Target Support Package™ software, check the list of supported processors in the Simulink library browser for a pre-configured Target Preferences block for your processor. For example:



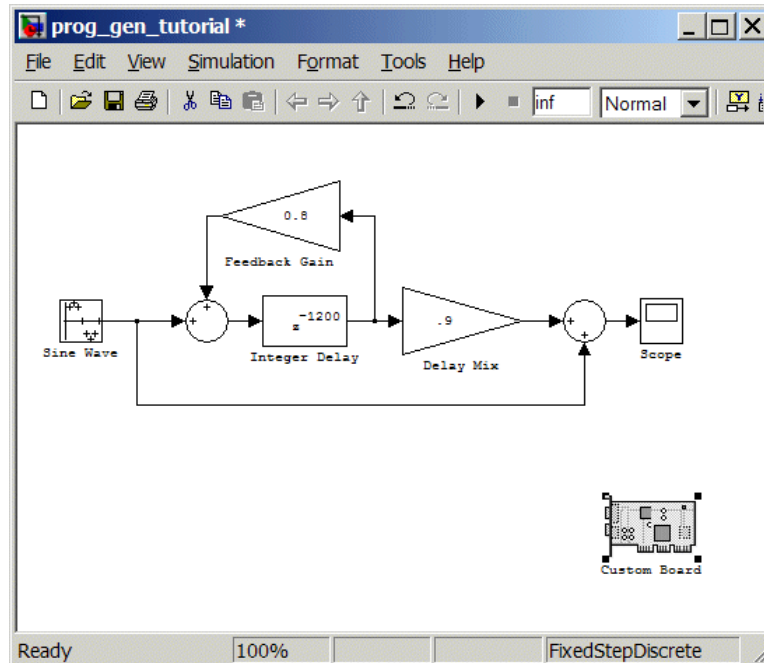
Otherwise, use the Target Preferences/Custom Board for ADI VisualDSP++ block, located in the `idlinklib_adivdsp` block library.



To configure the Target Preferences/Custom Board for ADI VisualDSP++ (the “Custom Board”) block in your model:

- 1 Drag and drop the Custom Board block to your model as shown in the following figure.





- 2 Double-click the Custom Board block to open the block dialog box.
- 3 In the block dialog box, select your processor from the **Processor** list.
- 4 Verify the **CPU clock** value.
- 5 Select the session name from the **Session name** list. Verify that the session processor matches the one you selected from the **Processor** list.
- 6 Review the settings on the **Memory** and **Sections** tabs to verify that they are correct for the processor you selected.
- 7 Click **OK** to close the Target Preferences dialog box.

You have completed the model. Next, configure the model configuration parameters to generate a project in VisualDSP++ IDE from your model.

## Specifying Simulink Configuration Parameters for Your Model

The following sections describe how to configure the build and run parameters for your model. Generating a project, or building and running a model on the processor, starts with configuring model options in the Configuration Parameters dialog box in Simulink software.

### Setting Solver Options

After you have designed and implemented your digital signal processing model in Simulink software, complete the following steps to set the configuration parameters for the model:

- 1 Open the Configuration Parameters dialog box and set the appropriate options on the **Solver** category for your model and for Embedded IDE Link software.
  - Set **Start time** to 0.0 and **Stop time** to `inf` (model runs without stopping). If you set a stop time, your generated code does not honor the setting. Set this to `inf` for completeness.
  - Under **Solver options**, select the fixed-step and discrete settings from the lists when you generate executable projects. When you use PIL, use any setting on the **Type** and **Solver** lists.
  - Set the **Fixed step size** to Auto and the **Tasking Mode** to Single Tasking.

---

**Note** Generated code does not honor Simulink stop time from the simulation. Stop time is interpreted as `inf`. To implement a stop in generated code, you must put a Stop Simulation block in your model.

---

Ignore the Data Import/Export, Diagnostics, and Optimization categories in the Configuration Parameters dialog box. The default settings are correct for your new model.

## Setting Real-Time Workshop Software Options

To configure Real-Time Workshop software to use the correct processor files and to compile and run your model executable file, you set the options in the Real-Time Workshop category of the **Select** tree in the Configuration Parameters dialog box. Follow these steps to set the Real-Time Workshop software options to generate code tailored for your DSP:

- 1 Select Real-Time Workshop on the **Select** tree.
- 2 In Target selection, click **Browse** to select the system target file for Analog Devices processors—`vdsplink_grt.tlc`. It may already be the selected target file.

Clicking **Browse** opens the **System Target File Browser** to allow you to change the system target file.

- 3 On the **System Target File Browser**, select the system target file `vdsplink_grt.tlc`, and click **OK** to close the browser.

## Setting Embedded IDE Link Options

After you set the Real-Time Workshop options for code generation, set the options that apply to your Analog Devices processor.

- 1 Change the category on the **Select** tree to Hardware Implementation.
- 2 Verify that the Device type is the correct value for your processor—ADI Blackfin, ADI SHARC, or ADI TigerSHARC.
- 3 From the **Select** tree, choose Embedded IDE Link to specify code generation options that apply to the processor.
- 4 Under **Code Generation**, clear all of the options.
- 5 (optional) Under **Link Automation**, provide a name for the handle in **IDE handle name**.
- 6 Set the following options in the dialog box under **Project options**:
  - Set **Project options** to Custom.
  - Set **Compiler options string** and **Linker options string** to blank.

**7** Set the following **Runtime** options:


- **Build action:** Create\_project.
- **Interrupt overrun notification method:** Print\_message.

You have configured the Real-Time Workshop options that let you generate a project for your processor. A few Real-Time Workshop categories on the **Select** tree, such as **Comments**, **Symbols**, and **Optimization** do not require configuration for use with Embedded IDE Link software. In some cases, you may decide to set options in the other categories.

For your new model, the default values for the options in these categories are correct. For other models you develop, you may want to set the options in these categories to provide information during the build and to run TLC debugging when you generate code. Refer to your Simulink and Real-Time Workshop documentation for more information about setting the configuration parameters.

### Creating Your Project

After you set the configuration parameters and configure Real-Time Workshop software to create the files you need, you direct the software to create your project:

- 1** Click **OK** to close the Configuration Parameters dialog box.
- 2** Click Incremental Build () on the model toolbar to generate your project into VisualDSP++ IDE.

When you click  with Create\_project selected for **Build action**, the automatic build process starts VisualDSP++ software and populates a new project in the development environment.

## Model Reference

In this section...
“How Model Reference Works” on page 3-11
“Using Model Reference” on page 3-12
“Configuring Targets to Use Model Reference” on page 3-14

Model reference lets your model include other models as modular components. This technique is useful because it provides the following capabilities:

- Simplifies working with large models by letting you build large models from smaller ones, or even large ones.
- Lets you generate code once for all the modules in the entire model and then only regenerate code for modules that change.
- Lets you develop the modules independently.
- Lets you reuse modules and models by reference, rather than including the model or module multiple times in your model. Also, multiple models can refer to the same model or module.

Your Real-Time Workshop documentation provides much more information about model reference.

### How Model Reference Works

Model reference behaves differently in simulation and in code generation. For this discussion, you need to know the following terms:

- The *Top model* is the root model block or model. It refers to other blocks or models. In the model hierarchy, this is the topmost model.
- *Referenced models* are blocks or models that other models reference, such as models the top model refers to. All models or blocks below the top model in the hierarchy are reference models.

The following sections describe briefly how model reference works. More details are available in your Real-Time Workshop documentation in the online Help system.

## **Model Reference in Simulation**

When you simulate the top model, Real-Time Workshop software detects that your model contains referenced models. Simulink software generates code for the referenced models and uses the generated code to build shared library files for updating the model diagram and simulation. It also creates an executable (.mex file) for each reference model that is used to simulate the top model.

When you rebuild reference models for simulations or when you run or update a simulation, Simulink software rebuilds the model reference files. Whether reference files or models are rebuilt depends on whether and how you change the models and on the **Rebuild options** settings. You can access these setting through the **Model Reference** pane of the Configuration Parameters dialog box.

## **Model Reference in Code Generation**

Real-Time Workshop software requires executables to generate code from models. If you have not simulated your model at least once, Real-Time Workshop software creates a .mex file for simulation.

Next, for each referenced model, the code generation process calls `make_rtw` and builds each referenced model. This build process creates a library file for each of the referenced models in your model.

After building all the referenced models, the software calls `make_rtw` on the top model, linking to all the library files it created for the associated referenced models.

## **Using Model Reference**

With few limitations or restrictions, Embedded IDE Link software provides full support for generating code from models that use model reference.

### **Build Action Setting**

The most important requirement for using model reference with the Analog Devices targets is that you must set the **Build action** (select **Configuration Parameters > Embedded IDE Link**) for all models referred to in the simulation to `Archive_library`.

To set the build action, perform the following steps:

- 1** Open your model.
- 2** Select **Simulation > Configuration Parameters** from the model menus.  
The Configuration Parameters dialog box opens.
- 3** From the **Select** tree, choose **Embedded IDE Link**.
- 4** In the right pane, under **Runtime**, select set `Archive_library` from the **Build action** list.

If your top model uses a reference model that does not have the build action set to `Archive_library`, the build process automatically changes the build action to `Archive_library` and issues a warning about the change.

Selecting the `Archive_library` setting removes the following options from the dialog box:

- **Interrupt overrun notification method**
- **Compiler options string**
- **Linker options string**
- **System stack size (MAUs)**
- **Profile real-time execution**

### **Target Preferences Blocks in Reference Models**

Each referenced model and the top model must include a Target Preferences block for the correct processor. You must configure all the Target Preferences blocks for the same processor.

The referenced models need target preferences blocks to provide information about which compiler and which archiver to use. Without these blocks, the compile and archive processes do not work.

By design, model reference does not allow information to pass from the top model to the referenced models. Referenced models must contain all the

necessary information, which the Target Preferences block in the model provides.

### **Other Block Limitations**

Model reference with Embedded IDE Link software does not allow you to use the following blocks or S-functions in reference models:

- No noninlined S-functions
- None of the following blocks:
  - Custom Board (Target Preferences)
  - Memory Allocate
  - Memory Copy
  - Idle Task
  - Hardware Interrupt for SHARC, TigerSHARC, or Blackfin DSPs

### **Configuring Targets to Use Model Reference**

When you create models to use in Model Referencing, keep in mind the following considerations:

- Your model must use a system target file derived from the ERT or GRT targets files.
- When you generate code from a model that references other models, you must configure the top-level model and the referenced models for the same system target file.
- Real-Time Workshop software builds and Embedded IDE Link software do not support external mode in model reference. If you select the external mode option, it is ignored during code generation.
- Your TMF must support use of the shared utilities directory, as described in Supporting Shared Utility Directories in the Build Process in the Real-Time Workshop documentation.

To use an existing processor, or a new processor, with Model Reference, set the `ModelReferenceCompliant` flag for the processor. For information about



setting this option, refer to `ModelReferenceCompliant` in the online Help system.

If you start with a model that was created prior to MATLAB release R14SP3, use the following command to set the `ModelReferenceCompliant` flag to `On` to make your model compatible with model reference:

```
set_param(bdroot, 'ModelReferenceCompliant', 'on')
```

Code that you generate from Simulink software models by using Embedded IDE Link software automatically include the model reference capability. You do not need to set the flag.



# Block Reference

---

Block Library: idelinklib\_aidvdsp  
(p. 4-2)

Blocks for Analog Devices  
VisualDSP++

## **Block Library: idelinklib\_aidvsp**

Blackfin Hardware Interrupt

SHARC Hardware Interrupt

TigerSHARC Hardware Interrupt

Generate Interrupt Service Routine

Generate Interrupt Service Routine

Generate Interrupt Service Routine

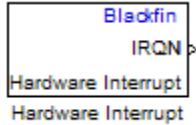
# Blocks — Alphabetical List

---

# Blackfin Hardware Interrupt

**Purpose** Generate Interrupt Service Routine

**Library** Block Library: idelinklib\_avidvsp



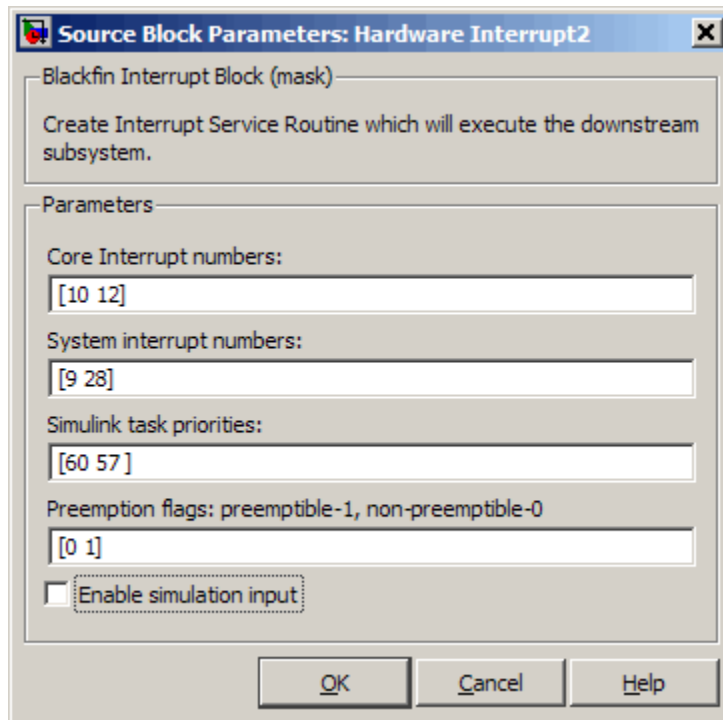
**Description** Create interrupt service routines (ISR) in the software generated by the build process. When you incorporate this block in your model, code generation results in ISRs on the processor that run the processes that are downstream from the this block or an Idle Task block connected to this block. Core interrupts trigger the ISRs. System interrupts trigger the core interrupts. In the following figure, you see the mapping possibilities between system interrupts and core interrupts.

## Interrupts

Blackfin processors support the interrupt numbers shown in the following table. Some Blackfin processors do not support all of the system interrupts.

Interrupt Description	Valid Range in Embedded IDE Link Software
Core interrupt numbers	7 to 14
System interrupt numbers	0 to 31 (The upper end value depends on the processor. May be less than 31.)

## Dialog Box



### Core interrupt numbers

Specify a vector of one or more interrupt numbers for the interrupt service routines (ISR) to install. The valid range is 7 to 14, where 7 through 13 are hardware driven, and 14 is software driven. Core interrupts numbered 0 to 6 are reserved and cannot be entered in this field. Each interrupt value must be unique.

The width of the block output signal corresponds to the number of interrupt values you specify in this field. Triggering of each ISR depends on the core interrupt value, the system interrupt value, and the preemption flag you enter for each interrupt. These three values define how the code and processor respond to interrupts during asynchronous scheduler operations.

# Blackfin Hardware Interrupt

---

## System interrupt numbers

System interrupt numbers identify system interrupts to map to core interrupts. Enter one or more values as a vector. Each interrupt value must be unique. The valid range is generally 0 through 31, although the range depends on your processor. Some processors do not support the full range of 32 system interrupts. Embedded IDE Link software does not test for valid system interrupt values. You must verify that your values are valid for your processor. To use asynchronous scheduling, you must specify a value for at least one system interrupt number.

The block maps the first interrupt value in this field to the first core interrupt value in **Core interrupt numbers**, it maps the second system interrupt value to the second core interrupt value, and so on until it has mapped all of the system interrupt values to core interrupt values. You cannot map more than one system interrupt to the same core interrupt. You must enter the same number of system interrupts as core interrupts.

When you trigger one of the system interrupts in this field, the block triggers the ISR associated with the core interrupt that is mapped to the system interrupt.

## Simulink task priorities

Each output of the Hardware Interrupt block drives a downstream block (for example, a function call subsystem). Simulink model task priority specifies the priority of the downstream blocks. Specify an array of priorities corresponding to the interrupt numbers entered in **Interrupt numbers**.

Proper code generation requires rate transition code (see Rate Transitions and Asynchronous Blocks). The task priority values ensure absolute time integrity when the asynchronous task must obtain real time from its base rate or its caller. Typically, assign priorities for these asynchronous tasks that are higher than the priorities assigned to periodic tasks.



## **Preemption flags preemptible – 1, non-preemptible – 0**

Higher priority interrupts can preempt interrupts that have lower priority. To control this preemption, use the preemption flags to specify whether an interrupt can be preempted.

- Entering 1 indicates the corresponding core interrupt can be preempted.
- Entering 0 indicates the corresponding interrupt cannot be preempted.

When **Core interrupt numbers** contains more than one interrupt priority, you can assign different preemption flags to each interrupt by entering a vector of preemption flag values that correspond to the order of the interrupts in **Core interrupt numbers**. If **Core interrupt numbers** contains more than one interrupt, and you enter only one flag value in this field, that status applies to all interrupts.

For example, the default settings [0 1] indicate that the interrupt with value 10 in **Core interrupt numbers** is not preemptible and the value 12 interrupt can be preempted.

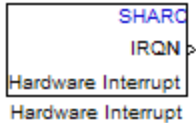
## **Enable simulation input**

When you select this option, Simulink software adds an input port to the Hardware Interrupt block. This port receives input only during simulation. Connect one or more simulated interrupt sources to the simulation input.

# SHARC Hardware Interrupt

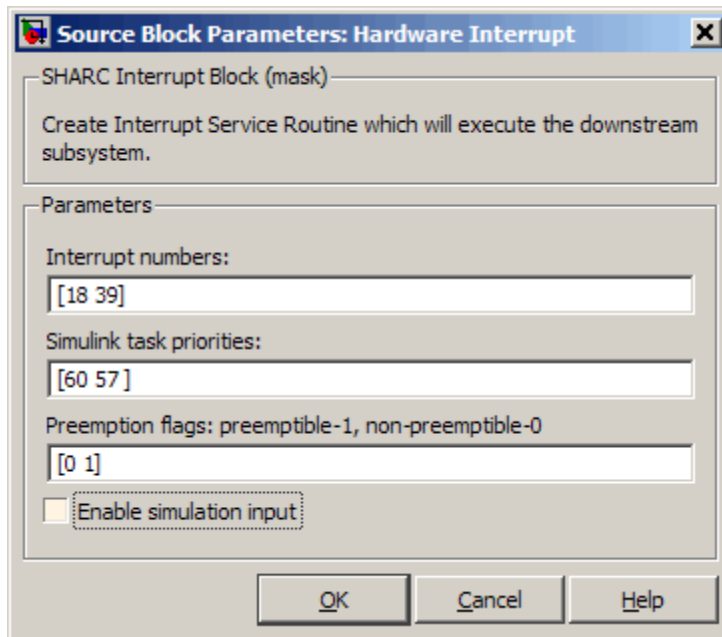
**Purpose** Generate Interrupt Service Routine

**Library** Block Library: idelinklib\_avidvsp



**Description** Create interrupt service routines (ISR) in the software generated by the build process. When you incorporate this block in your model, code generation results in ISRs on the processor that either run the processes that are downstream from this block or trigger an Idle Task block connected to this block.

## Dialog Box



## Interrupt numbers

Specify an array of interrupt numbers for the interrupts to install. The valid ranges are 8-36 and 38-40.

The width of the block output signal corresponds to the number of interrupt numbers specified in this field. The values in this field and the preemption flag entries in **Preemption flags: preemptible-1, non-preemptible-0** define how the code and processor handle interrupts during asynchronous scheduler operations.

## Simulink task priorities

Each output of the Hardware Interrupt block drives a downstream block (for example, a function call subsystem). Simulink model task priority specifies the priority of the downstream blocks. Specify an array of priorities corresponding to the interrupt numbers entered in **Interrupt numbers**.

Proper code generation requires rate transition code (refer to Rate Transitions and Asynchronous Blocks in the Real-Time Workshop documentation). The task priority values ensure absolute time integrity when the asynchronous task must obtain real time from its base rate or its caller. Typically, assign priorities for these asynchronous tasks that are higher than the priorities assigned to periodic tasks.

## Preemption flags preemptible – 1, non-preemptible – 0

Higher-priority interrupts can preempt interrupts that have lower priority. To allow you to control preemption, use the preemption flags to specify whether an interrupt can be preempted.

- Entering 1 indicates that the interrupt can be preempted.
- Entering 0 indicates the interrupt cannot be preempted.

When **Interrupt numbers** contains more than one interrupt value, you can assign different preemption flags to each interrupt by entering a vector of flag values to correspond to the order of the interrupts in **Interrupt numbers**. If **Interrupt numbers**

# SHARC Hardware Interrupt

---

contains more than one interrupt, and you enter only one flag value in this field, that status applies to all interrupts.

In the default settings [0 1], the interrupt with priority 18 in **Interrupt numbers** is not preemptible and the priority 39 interrupt can be preempted.

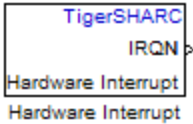
## **Enable simulation input**

When you select this option, Simulink software adds an input port to the Hardware Interrupt block. This port is used in simulation only. Connect one or more simulated interrupt sources to the simulation input.

# TigerSHARC Hardware Interrupt

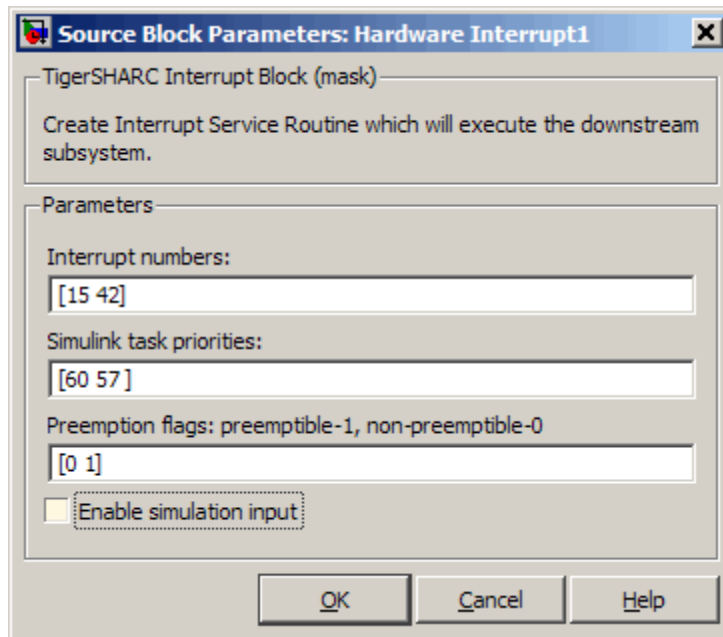
**Purpose** Generate Interrupt Service Routine

**Library** Block Library: idelinklib\_adivdsp



**Description** Create interrupt service routines (ISR) in the software generated by the build process. When you incorporate this block in your model, code generation results in ISRs on the processor that run the processes that are downstream from the this block or an Idle Task block connected to this block.

## Dialog Box



# TigerSHARC Hardware Interrupt

---

## **Interrupt numbers**

Specify an array of interrupt numbers for the interrupts to install. The valid interrupts are 2, 3, 6-9, 14-17, 22-25, 29-32, 37, 38, 41-44, 52.

The width of the block output signal corresponds to the number of interrupt numbers specified in this field. Combined with the **Simulink task priorities** that you enter and the preemption flag you enter for each interrupt, these three values define how the code and processor handle interrupts during asynchronous scheduler operations.

## **Simulink task priorities**

Each output of the Hardware Interrupt block drives a downstream block (for example, a function call subsystem). Simulink model task priority specifies the priority of the downstream blocks. Specify an array of priorities corresponding to the interrupt numbers entered in **Interrupt numbers**.

Simulink model task priority values are required to generate the proper rate transition code (refer to Rate Transitions and Asynchronous Blocks in the Real-Time Workshop documentation). The task priority values are also required to ensure absolute time integrity when the asynchronous task needs to obtain real time from its base rate or its caller. Typically, you assign priorities for these asynchronous tasks that are higher than the priorities assigned to periodic tasks.

## **Preemption flags preemptible – 1, non-preemptible – 0**

Higher priority interrupts can preempt interrupts that have lower priority. To allow you to control preemption, use the preemption flags to specify whether an interrupt can be preempted.

Entering 1 indicates that the interrupt can be preempted. Entering 0 indicates the interrupt cannot be preempted. When **Interrupt numbers** contains more than one interrupt priority, you can assign different preemption flags to each interrupt by entering a vector of flag values, corresponding to the order of

the interrupts in **Interrupt numbers**. If **Interrupt numbers** contains more than one interrupt, and you enter only one flag value in this field, that status applies to all interrupts.

In the default settings [0 1], the interrupt with priority 15 in **Interrupt numbers** is not preemptible and the priority 42 interrupt can be preempted.

### **Enable simulation input**

When you select this option, Simulink software adds an input port to the Hardware Interrupt block. This port is used in simulation only. Connect one or more simulated interrupt sources to the simulation input.

# TigerSHARC Hardware Interrupt

---



# Reported Limitations and Tips

---

## Reported Issues

Some long-standing issues affect the Embedded IDE Link software. When you are using `adivdsp` objects and methods to work with VisualDSP++ software and supported hardware or simulators, recall the information in this section.

The latest issues in the list appear at the bottom. PIL means “processor-in-the-loop” and is similar to hardware-in-the-loop operations.

### **Using 64-bit Symbols in a 64-bit Memory Section on SHARC Processors**

VisualDSP++ compiler design prevents Embedded IDE Link from generating code that accesses 64-bit memory locations correctly. To avoid unexpected results, do not allocate 64-bit data or symbols to 64-bit memory locations on SHARC processors.

When 64-bit data is in 64-bit memory, the compiler generates code that accesses the 64-bit locations as two 32-bit values. Thus, the code does not read and write the 64-bit data correctly. It reads or writes every other 32-bit location, returning or writing incorrect values and possibly exceeding the allocated memory.

Refer to pp. 5-33 in the *ADSP-2136x SHARC Processor Programming Reference, revision 1.0* for a description of how the compiler treats 64-bit (long word) data values.

# Supported Processors

---

This appendix provides the details about the processors, simulators, and software that work with Embedded IDE Link.

## Supported Platforms

In this section...
“Product Features Supported by Each Processor or Family” on page B-2
“Supported Processors and Simulators” on page B-2
“Custom Board Support” on page B-3

This appendix lists the processors and simulators that work with the latest released version of Embedded IDE Link for use with Analog Devices VisualDSP++.

### Product Features Supported by Each Processor or Family

The following table indicates which Embedded IDE Link features are available by processor family.

#### Features by Processor Family

Automation Interface Component		Project Generator Component	Verification	
Processor Family	Debug Mode	Code Generation	PIL	Real-Time Execution Profiling
BF52x	Yes	Yes	Yes	Yes
BF531-BF534, BF536-BF539	Yes	Yes	Yes	Yes
SHARC 2136x	Yes	Yes	Yes	Yes
TS20x	Yes	Yes	Yes	Yes

### Supported Processors and Simulators

Embedded IDE Link has been tested on the following processors and boards produced by ADI:

- BF52x
  - ADI Simulators (BF52x)
  - ADSP-BF527 EZ-KIT LITE
- BF531-BF534, BF536-BF539
  - ADI Simulators (BF53x)
  - ADDS-BF537-EZLITE
- SHARC 2136x
  - ADI Simulators (ADSP-2136x Simulator)
  - ADSP-21364 EZ-KIT LITE
  - ADSP-21369 EZ-KIT LITE
- TS 20x
  - ADI Simulators (ADSP-TS201 rev. 1.x/2.x Single Processor Simulator)
  - ADSP-TS201S EZ-KIT LITE

## **Custom Board Support**

You can use Embedded IDE Link with your custom board if:

- The board uses one or more of the supported processors in the preceding list.
- Your processor appears in the Processor list of the Target Preferences or Custom Board block.
- You are able to use Analog Devices VisualDSP++ to interact with your board/processor combination.



## A

- access properties 2-20
- adivdsp 2-18
- adivdsp object properties 2-26
  - procnum 2-25
  - sessionname 2-26
- Analog Devices model reference 3-11
- Archive\_library 3-12

## B

- block limitations using model reference 3-14

## F

- functions
  - overloading 2-23

## G

- getting properties 2-22

## L

- link filters properties
  - getting 2-22
- link properties
  - about 2-25
  - setting 2-22
- link properties, details about 2-25
- linking objects
  - quick reference 2-24
- links
  - closing VisualDSP++® 2-16
  - details 2-25
  - loading files into VisualDSP++® IDE 2-9
  - working with your processor 2-11

## M

- model reference 3-11

- about 3-11
- Archive\_library 3-12
- block limitations 3-14
- modelreferencecompliant flag 3-14
- setting build action 3-12
- target preferences blocks 3-13
- using 3-12
- modelreferencecompliant flag 3-14

## O

- object
  - adivdsp 2-18
- object properties
  - about 2-24
  - quick reference table 2-24
- objects
  - creating objects for VisualDSP++® IDE 2-7
  - introducing the objects for VisualDSP++® IDE tutorial 2-2
  - selecting processors for VisualDSP++® IDE 2-6
  - tutorial about using Automation Interface for VisualDSP++® IDE 2-2
- overloading 2-23

## P

- procnum 2-25
- properties
  - object properties 2-24
  - referencing directly 2-22
  - retrieving 2-20
    - function for 2-22
  - retrieving by direct property referencing 2-22
  - setting 2-20

## S

- sessionname 2-26
- set properties 2-20

structure-like referencing 2-22

**T**

target preferences blocks in referenced  
models 3-13  
timeout  
timeout 2-26

tutorials

objects for VisualDSP++® 2-2

**V**

VisualDSP++® IDE objects  
tutorial about using 2-2