CEVA
The DSP Powerhouse

CEVA-Toolbox™

MATLAB® – Debugger Communications Module Reference Guide

Rev 15.1.0

August 2015

DSP Division

Documentation Control

History Table:

Version	Date	Description	Remarks
10.3	18/09/2014	Document creation and update to V10.3	
15.1.0.	12/08/2015	Updates for the V15 version, snapshots and invocation changed	

Disclaimer and Proprietary Information Notice

The information contained in this document is subject to change without notice and does not represent a commitment on any part of CEVA®, Inc. CEVA®, Inc. and its subsidiaries make no warranty of any kind with regard to this material, including, but not limited to implied warranties of merchantability and fitness for a particular purpose whether arising out of law, custom, conduct or otherwise.

While the information contained herein is assumed to be accurate, CEVA®, Inc. assumes no responsibility for any errors or omissions contained herein, and assumes no liability for special, direct, indirect or consequential damage, losses, costs, charges, claims, demands, fees or expenses, of any nature or kind, which are incurred in connection with the furnishing, performance or use of this material.

CEVA®, CEVA-XC™, CEVA-XC321™, CEVA-XC323™, CEVA-Xtend™, CEVA-XC4000™, CEVA-XC4100™, CEVA-XC4200™, CEVA-XC4210™, CEVA-XC4400™, CEVA-XC4410™, CEVA-XC4500™, CEVA-TeakLite™, CEVA-TeakLite-II™, CEVA-TeakLite-III™, CEVA-TL3210™, CEVA-TL3211™, CEVA-TeakLite-4™, CEVA-TL410™, CEVA-TL411™, CEVA-TL420™, CEVA-TL421™, CEVA-Quark™, CEVA-Teak™, CEVA-X™, CEVA-X1620™, CEVA-X1622™, CEVA-X1641™, CEVA-X1643™, Xpert-TeakLite-II™, Xpert-Teak™, CEVA-XS1100A™, CEVA-XS1200™, CEVA-XS1200A™, CEVA-TLS100™, Mobile-Media™, CEVA-MM1000™, CEVA-MM2000™, CEVA-SP™, CEVA-VP™, CEVA-MM3000™, CEVA-MM3100™, CEVA-MM3101™, CEVA-XM™, CEVA-XM4™, CEVA-X2™ CEVA-Audio™, CEVA-HD-Audio™, CEVA-VoP™, CEVA-Bluetooth™, CEVA-SATA™, CEVA-SAS™, CEVA-Toolbox™, SmartNcode™ are trademarks of CEVA, Inc.

All other product names are trademarks or registered trademarks of their respective owners

Support

CEVA® makes great efforts to provide a user-friendly software and hardware development environment. Along with this, CEVA® provides comprehensive documentation, enabling users to learn and develop applications on their own. Due to the complexities involved in the development of DSP applications that may be beyond the scope of the documentation, an on-line Technical Support Service (support@ceva-dsp.com) has been established. This service includes useful tips and provides fast and efficient help, assisting users to quickly resolve development problems.

How to Get Technical Support:

FAQs: Visit our web site http://www.ceva-dsp.com or your company's protected page on the CEVA® website for the latest answers to frequently asked questions.

Application Notes: Visit our website http://www.ceva-dsp.com or your company's protected page on the CEVA® website for the latest application notes.

Email: Use CEVA's central support email address support@ceva-dsp.com. Your email will be forwarded automatically to the relevant support engineers and tools developers who will provide you with the most professional support in order to help you resolve any problem.

License Keys: Please refer any license key requests or problems to sdtkeys@ceva-dsp.com. Refer to the *SDT Installation & Licensing Scheme Guide* for SDT license keys installation information.

Email: support@ceva-dsp.com
Visit us at: www.ceva-dsp.com

List of Sales and Support Centers

Israel	USA	Ireland	Sweden
2 Maskit Street P.O.Box 2068 Herzelia 46120 Israel **Tel:** +972 9 961 3700 **Fax:** +972 9 961 3800	1943 Landings Drive Mountain View, CA 94043 USA **Tel:** +1-650-417-7923 **Fax:** +1-650-417-7924	Segrave House 19/20 Earlsfort Terrace 3rd Floor Dublin 2 Ireland **Tel**: +353 1 237 3900 **Fax**: +353 1 237 3923	Klarabergsviadukten 70 Box 70396 107 24 Stockholm, Sweden **Tel**: +46(0)8 506 362 24 **Fax**: +46(0)8 506 362 20
China (Shanghai)	**China (Beijing)**	**China Shenzhen**	**Hong Kong**
Room 517, No. 1440 Yan An Road (C) Shanghai 200040 China **Tel:** +86-21-22236789 **Fax:** +86 21 22236800	Rm 503, Tower C, Raycom InfoTech Park No.2, Kexueyuan South Road, Haidian District Beijing 100190, China **Tel:** +86-10 5982 2285 **Fax:** +86-10 5982 2284	2702-09 Block C Tiley Central Plaza II Wenxin 4th Road, Nanshan District Shenzhen 518054 **Tel:** +86-755-86595012	Level 43, AIA Tower, 183 Electric Road, North Point Hong Kong **Tel:** +852-39751264 **:**
South Korea	**Taiwan**	**Japan**	**France**
#478, Hyundai Arion, 147, Gumgok-Dong, Bundang-Gu, Sungnam-Si, Kyunggi-Do, 463-853, Korea **Tel**: +82-31-704-4471 **Fax**:+82-31-704-4479	Room 621 No.1, Industry E, 2nd Rd Hsinchu, Science Park Hsinchu 300 Taiwan R.O.C **Tel**: +886 3 5798750 **Fax**: +886 3 5798750	3014 Shinoharacho Kasho Bldg. 4/F Kohoku-ku Yokohama, Kanagawa 222-0026 Japan **Tel:** +81 045-430-3901 **Fax:** +81 045-430-3904	RivieraWaves S.A.S 400, avenue Roumanille Les Bureaux Green Side 5, Bât 6 06410 Biot - Sophia Antipolis, France **Tel:** +33 4 83 76 06 00 **Fax:** +33 4 83 76 06 01

Table of Contents

1. Introduction

MATLAB® is an integrated technical computing environment that combines numeric computation, graphics and visualization, and a high-level programming language (defined also as a Fourth Generation Language, 4GL).

MATLAB cannot be used for algorithm implementation on a chip. When performing algorithm execution on a chip, it is necessary to translate the algorithm to a high-level language (defined also as a Third Generation Language, 3GL) such as C/C++ or even Assembly language followed by code linking, and then to execute it on the CEVA ToolBox Debugger specific Simulator or Emulator. In theory this is a simple task, but in reality most users translate the algorithm part by part, making it necessary to use both MATLAB and the ToolBox Debugger in parallel to verify the translation and validate the algorithm's accuracy.

2. MATLAB-Debugger Communication Module

The MATLAB-Debugger Communication module (**mtldbg**) handles conversations between MATLAB and the ToolBox Debugger. A conversation is defined as a communication session between two active application processes using a specific protocol.

The MATLAB-Debugger Communication module (mtldbg from here) can be found in the tools directory under "mtldbg" directory, for example:

%CEVAXCTOOLS%\mtldbg\mtldbg.mexw32 (for MATLAB 32bit)

%CEVAXCTOOLS%\mtldbg\mtldbg.mexw32 (for MATLAB 64bit)

The communication between MATLAB and the ToolBox Debugger is based on a TCP/IP connection between the "mtldbg" and a listening remote debugger server which controls the debugger using the CEVA-Debugger API.

In order to initiate a connection, from the MATLAB application call the function 'opendbg()' (defined in the mtldbg module).

Example:

opendbg('CEVA-XC4210', 'example');

Will open a remote debugger server with a CEVA-XC4210 core loaded and the session name will be 'example'.

2.1 Supported MATLAB versions

The CEVA-Debugger MATLAB connection is available on the following MATLAB versions:

- Windows MATLAB R2013a – 32bit
- Windows MATLAB R2013a – 64bit

2.2 Setting Up the MATLAB Environment

The following steps should be taken in order to include the MATLAB-Debugger communication options:

1. Activate the MATLAB application.

2. Select the **Set Path** option from the **File** menu, or type '**pathtool'** in the command window prompt. The **Set Path** dialog box opens:

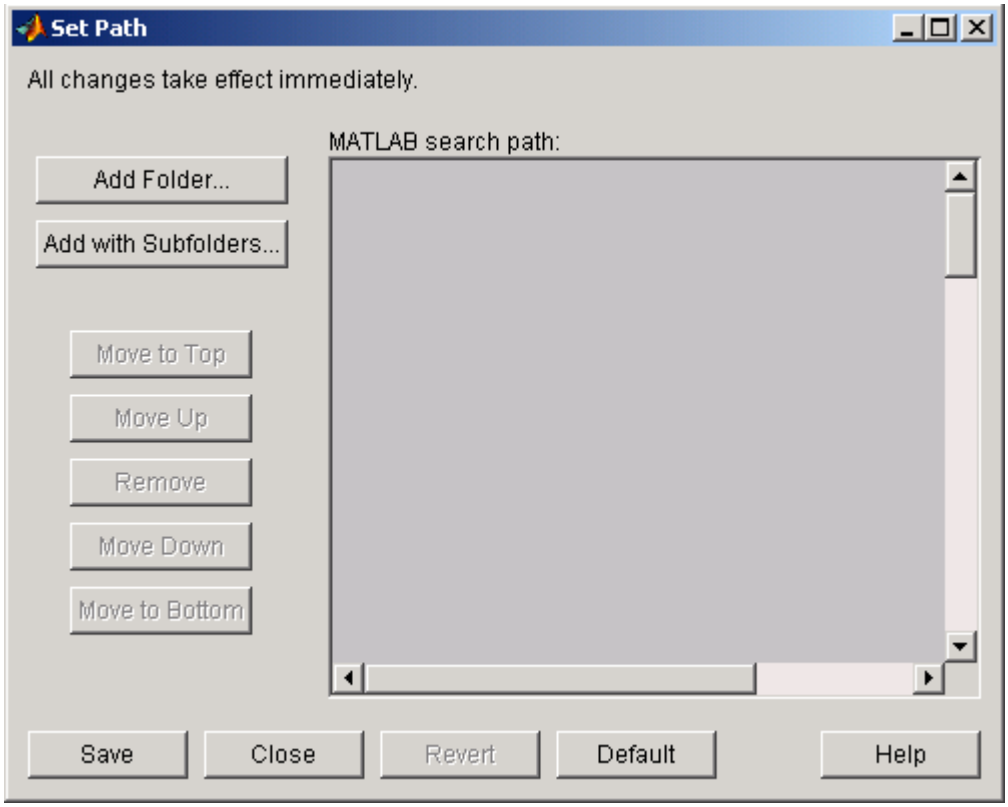

Figure 1: Set Path Dialog Box

3. Click the **Add With Subfolder** button. The **Browse for Folder** dialog box opens:

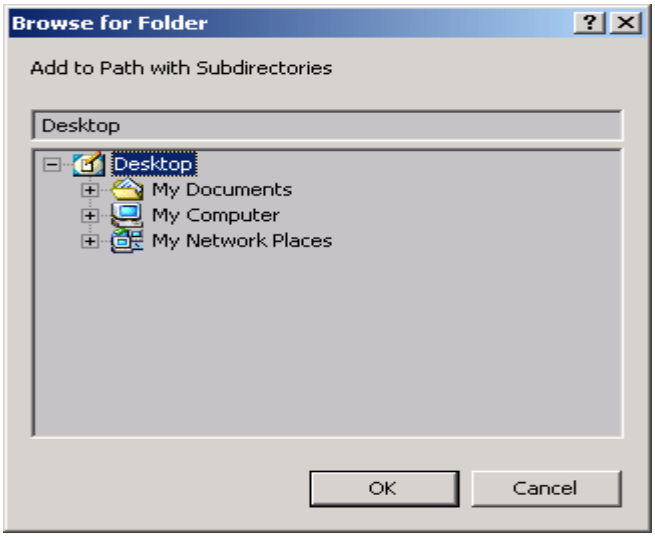

Figure 2: Browse for Folder Dialog Box

4. In the **Browse for Folder** dialog box, use the view of your file system to select the MATLAB add-in directory (mtldbg), and then click **OK**.
   For example select: c:\CEVA-ToolBox\V10\ceva-xc\mtldbg

5. Move the 'mxfiles' directory to above the 'helpfiles' directory, by choosing the mxfile directory and moving it by clicking on the **Move Up** button.

6. Click the **Save** button in the **Browse for Folder** dialog box.

Note: *A license manager protects the MATLAB-Debugger module. The first time the user tries to use the Debugger function through the communication module, the license dialog-box appear if the user has no active license. For more information on how to set a license please refer to the CEVA-Toolbox SDT Installation User's Guide*

3. MATLAB-Debugger Communication Usage

The MATLAB-Debugger Communication module provides a collection of functions, which characterize the communication.

The functions are divided into four groups:

1. Communication
2. Retrieval Information Data
3. Setting Information Data
4. CLI Execution

3.1 Communication Functions

This group of functions is responsible for establishing conversations between MATLAB and the Debugger. The following functions are included:

1. **opendbg** – Starts a remote debugger server communication with a core loaded and a given session name.

   Syntax:

   opendbg (coreName, sessionName)

   Return Value:

   NULL

   Parameters:

   coreName – name of the core requested to load
   sessionName – name of the debugger API session name

   Example:
   opendbg('CEVA-XC4500', 'example');

2. **openDbg3** – deprecated as of CEVA-ToolBox CEVA-XC4xxx V10.3

3. **connectDbg** – deprecated as of CEVA-ToolBox CEVA-XC4xxx V10.3

3.2 Retrieval Information Data Functions

This group of functions is responsible for retrieving data from the Debugger and includes the following functions:
Note: **get/set functions are supported only with MATLABs' default "double" type.**

1. **getdata** – Returns the values of a data memory address array from the Debugger.

   Optional syntax:

   ```
   data = getdata    (startAddress, numAddresses);
   data = getdata    ('startAddressLabel', numAddresses);
   data = getdata    ('startAddressLabel', 'endAddressLabel');
   ```

   Return Value:

   data – vector of values as returned from the Debugger (double)

   Parameters:

   startAddress– memory start address

   numAddresses – number of requested addresses (length)

   startAddressLabel – start address label

   endAddressLabel – end address label

   Example:

   ```
   getdata ('label1', 10);
   getdata ('label1', 'label2');
   getdata(0, 10);
   ```

2. **getcode** – This function returns the values of a code memory address range from the Debugger.

   Optional syntax:

   ```
   data = getcode  (startAddress, numAddresses);
   data = getcode  ('startAddressLabel', numAddresses);
   data = getcode  ('startAddressLabel', 'endAddressLabel');
   ```

   Return Value:

   Data – vector of values as returned from the Debugger (double)

Parameters:

startAddress – memory start address

numAddresses – number of requested addresses (length)

startAddressLabel – start address label

endAddressLabel – end address label

Example:

getcode ('label1', 10);

getcode ('label1', 'label2');

getcode (0, 10);

3. **getreg –** This function returns a register value from the Debugger.

Optional syntax:

data = getreg(‘regName’)

Return Value:

data – The requested register value as returned from the Debugger (double)

Parameters:

regName – a string holding the requested register’s name.

Example:

getreg ('pc');

4. **getdataarray** – Returns the values of a data memory address array (2D) from the Debugger.

Optional syntax:

data = getDataArray (‘startAddressLabel’, rows, cols, byte_num);
data = getDataArray (startAddress, rows, cols, byte_num);

Return Value:

data – vector of values as returned from the Debugger (double)

Parameters:

startAddress– memory start address

startAddressLabel – start address label

rows – number of rows

cols – number of cols

byte_num – 1,2,4 - in bytes

Example:

getDataArray ('label1', 100, 100, 2);

getDataArray (hex2dec('89AC'),100,100,4);

3.3 Setting Information Data Functions

This group of functions is responsible for setting data into the Debugger and includes the following functions:
Note: **get/set functions are supported only with MATLABs' default "double" type.**

1. **setdata** – This function sets the values of a data memory address range in the Debugger.

   Optional syntax:

   setdata (startAddress, addressesValue);

   setdata ('startAddressLabel', addressesValue);

   Return Value:

   NULL

   Parameters:

   startAddress – memory start address

   addressesValue – a matrix of values, the Debugger reads the matrix row by row and sets the requested addresses.

   Example:

   setdata ('label', [1 2 3]);

   setdata (0, [1 2 3]);

2. **setcode** – This function sets the values of a code memory address range in the Debugger.

   Optional syntax:

   setcode (startAddress, addressesValue);

   setCode ('startAddressLabel', addressesValue);

   Return Value:

   NULL

Parameters:

startAddress - memory start address

addressesValue - a matrix of values, the Debugger reads the matrix row by row and sets the requested addresses.

Example:

setcode ('label1', [1 2 3]);

setCode(0 , [1 2 3]);

3. **setreg** – This function sets the value of a register into the Debugger.

Optional syntax:

setreg (‘regName’, value);

Return Value:

NULL

Parameters:

regName – a string holding the requested register name

value – value to set into the requested register

Example:

setreg ('pc', 0);

4. **setdataarray** – This function sets the values of a data memory address array range in the Debugger.

Optional syntax:

setdataarray (‘startAddressLabel’, addresses_value, byte_num);
setdatadrray (startAddress, addresses_value, byte_num);

Return Value:

NULL

Parameters:

startAddress– memory start address

startAddressLabel – start address label (or legal Debugger address syntax)

addresses_value – Matlab array variable name

byte_num – 1,2,4 - in bytes

Example:

```
setdataarray ('label', array2, 2);
setdataarray (hex2dec('89AC'), array1,2);
```

3.4 CLI Execution Functions

This group of functions is responsible for executing all CLI command in the Debugger and includes the following functions:

1. **runclicmd** – This function sends a request to the Debugger to execute a CLI command.

   Optional syntax:

   runclicmd (‘clicmd’);

   Return Value:

   None

   Parameters:

   clicmd – CLI command string to be sent to the Debugger

   Example:

   runclicmd ('go');

2. **runCliCmdNotBlocked** – deprecated as of CEVA-ToolBox CEVA-XC4xxx V10.3

4. A Simple Example of usage

This example demonstrates the communication process between MATLAB and the Debugger.

The example computes a multiplication table by sending each iteration two numbers from the MATLAB application to the Debugger; the Debugger multiplies the numbers and returns the result to the MATLAB application.
In the end, the MATLAB application displays the multiplication table.

The functions that are used for this communication are:

1. opendbg
2. runclicmd
3. setdata
4. getdata

This example is made up of two sources:
C++ source file for the DSP core
M file for the MATLAB application

4.1 DSP Side

```
// Declare three global variables, two for the computation (x, y) and ‘res’ for the
result
int x   __attribute__((section(".DSECT x_var")));
int y   __attribute__((section(".DSECT y_var")));
int res __attribute__((section(".DSECT res_var")));

// The main body
void main()
{
        res = x*y;
}
```

4.2 MATLAB

1. % perform reset:

```
clear
```

2. % open CEVA-XC4500 Debugger with the session name of 'example':

```
opendbg('CEVA-XC4500', ‘example’);
```

3. % load the executable file to the Debugger:

```
runclicmd('disable message all');
runclicmd('load coff matdbg.a');

x = 0;
y = 0;
res = 0;
```

4. % compute 3*3 multiplication table:

```
for x=1:3,
   for y=1:3,
```

5. % set the multiplication parameters:

```
setdata('x_var', x);
setdata('y_var', y);
```

7.%compute the result:

```
runclicmd('run');
```

8. %get the multiplication result:

```
z = getdata('res_var', 1);
```

9. % store the result in the matrix

```
res(x,y) = z;
```

10. % reset the Debugger

```
runclicmd('reset');
end   %for
end   %for

disp(res);
```

5. Remote Debugger connection

As mentioned above the MATLAB debugger connection initiates a remote debugger server loaded with the requested core.
This server is also accessible via the Remote debugger connection (Eclipse debug configuration).
It is possible to connect the CEVA-ToolBox debugger to the running session to source debug the application loaded into the core DSP.

5.1 Connection from an external CEVA-ToolBox debugger

The following is an example of connecting the external Debugger to an existing remote session.

1. Open the CEVA-ToolBox IDE.
2. In the CEVA-Toolbox IDE create a “CEVA Remote Debugger” debug configuration

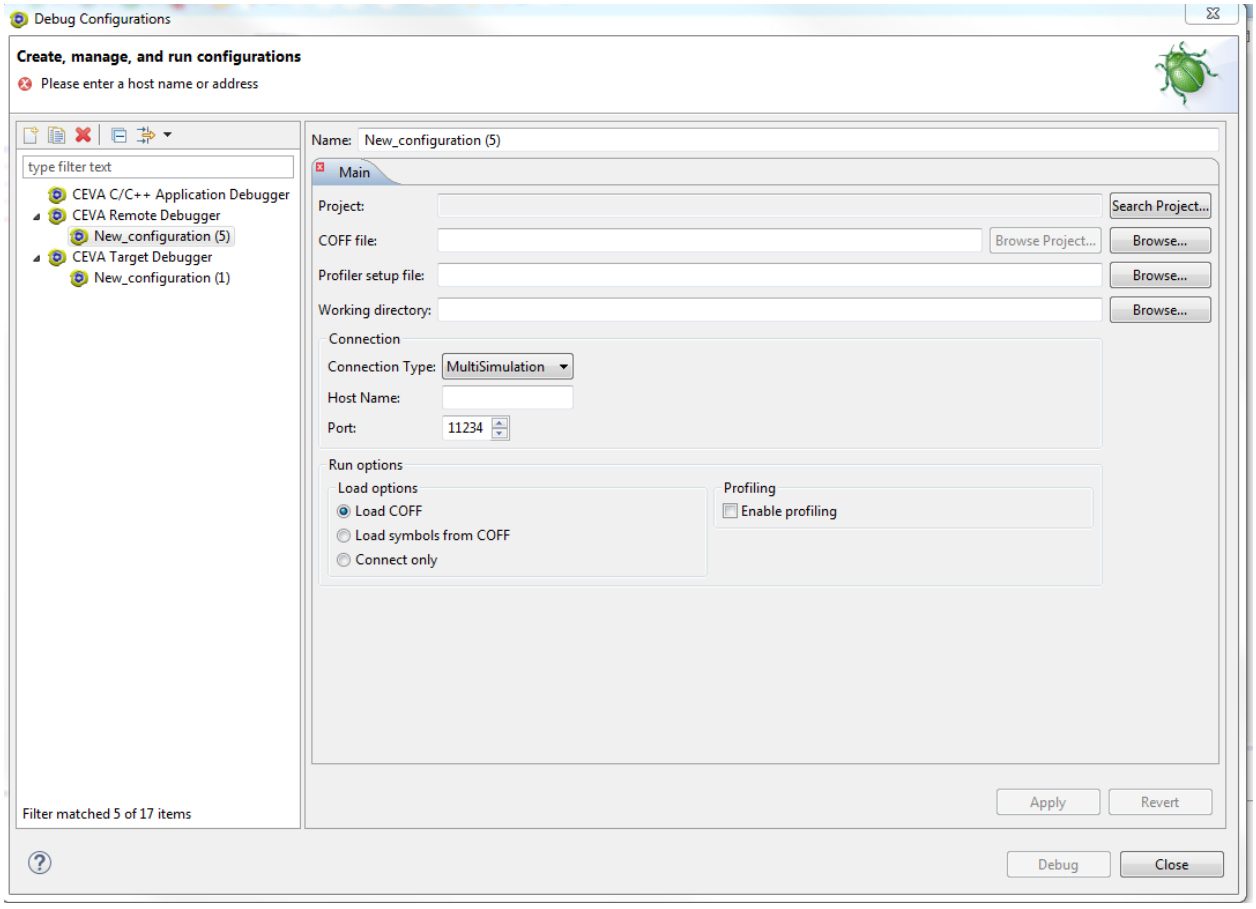

Figure 3: CEVA Remote debugger in the ToolBox IDE

3. Set connection type to “MultiSimulation” and port number to “21234”

4. Choose the correct working directory and correct coff file to load symbols.
5. Choose “Load symbols from COFF” as the coff is already loaded by the MATLAB debug module.
6. Enter debug