



CEVA-Toolbox[™]



CEVA-Xtend[™] Reference Guide

Rev 15.1.0

August 2015

Documentation Control

History Table:

Version	Date	Description	Remarks
10.0	01/12/2012	First tracked version	
10.0.2	02/09/2014	Updated for TKL4 Xtend	
15.1.0	12/08/2015	Updates for the V15	

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.

This document contains proprietary information, which is protected by U.S. and international copyright laws. All rights reserved. No part of this document may be reproduced, photocopied, or translated into another language without the prior written consent of CEVA®, Inc.

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	1-1
2	INSTALLATION	2-1
3	CREATING CEVA-XTEND MODULE	3-1
3.1	Define New Instructions in CEVA-Xtend Designer	3-2
3.2	Create an Application that Uses the New Instructions	3-2
4	CEVA-XTEND DESIGNER OPERATION	4-1
4.1	Overview	4-1
4.2	Registers Tab	4-2
4.2.1	REGISTER FIELDS	4-2
4.2.2	REGISTER LIST	4-3
4.2.3	REGISTER VISUALIZATION DISPLAY	4-3
4.2.4	ADVANCE MODE	4-4
4.3	Registers Groups Tab	4-5
4.3.1	REGISTERS GROUP	4-5
4.3.2	ENCODING GROUP	4-7
4.4	Switches Tab	4-7
4.4.1	SWITCHES GROUP	4-8
4.4.2	SWITCHES LIST	4-8
4.4.3	OPTION TABLE	4-8
4.5	Instructions Tab	4-9
4.5.1	INSTRUCTION GROUP	4-9
4.5.2	INSTRUCTION TEMPLATE	4-10
4.5.3	OPCODE DISPLAY	4-12
4.6	Functions Tab	4-13
4.6.1	FUNCTION CONTROL	4-14
4.6.2	STAGE SELECTION	4-14
4.6.3	FUNCTION IMPLEMENTATION	4-14
4.6.4	SIMULATION FILE CREATION	4-15
4.7	Instruction DB Tab	4-16
4.7.1	SORT	4-16
4.7.2	INSTRUCTIONS TABLE	4-16
4.8	XML Doc Tab	4-17
4.8.1	REGISTER TAB	4-18
4.8.2	VARIABLES TAB (REGISTERS GROUPS)	4-18
4.8.3	SWITCHES TAB	4-18
4.8.4	INSTRUCTIONS TAB	4-18
4.8.5	EDITOR MACRO SUPPORT	4-18
5	CEVA-XTEND MANUAL CREATION	5-1
5.1	XML File Description	5-1
5.2	CEVA-Xtend XML File Definition	5-2
5.3	CEVA-Xtend Types Example	5-3
5.4	Instruction Definition Example	5-4
5.5	CEVA-Xtend Simulation File	5-4
5.5.1	SIMULATION FILE STRUCTURE	5-4
5.5.2	SIMULATION FILE MACROS	5-5
5.5.3	SIMULATION FILE TYPE	5-11
5.5.5	SIMULATION FILE - EXAMPLE 1	5-12
5.5.6	SIMULATION FILE – EXAMPLE 2	5-13
6	DEBUG CEVA-XTEND PROJECT	6-1
7	CEVA-XTEND CORE SPECIFIC	7-1
7.1	Limitations CEVA-XC	7-1
7.2	CEVA-TL4xx Templates naming conventions	7-3

7.2.1	ARCHITECTURE SPECIFICATION AND CEVA-XTEND DESIGNER.....	7-3
7.2.2	CORE REGISTER GROUPS	7-3
7.3	Xtend Templates	7-4
7.3.1	CEVA-XC321	7-4
7.3.2	CEVA-XC323	7-4
7.3.3	CEVA-XC4210	7-5
7.3.4	CEVA-TL4xx XTEND TEMPLATES.....	7-6
7.3.5	CEVA-XM4.....	7-13

List of Figures

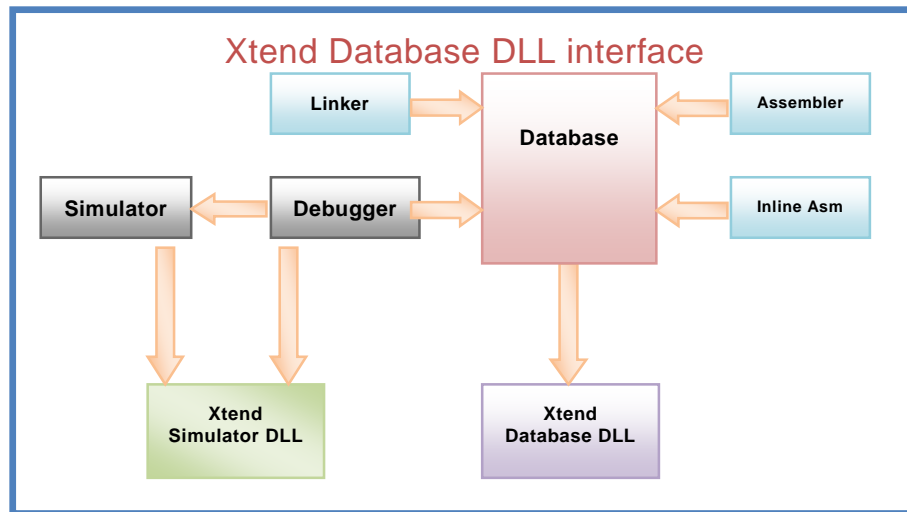
Figure 3-1: The CEVA-Xtend Editor.....	3-1
Figure 4-1: Register Tab - Simple Mode.....	4-2
Figure 4-2: Register Tab - Advanced Mode.....	4-4
Figure 4-3: Registers Groups Tab	4-5
Figure 4-4: Switches Tab	4-7
Figure 4-5: Instructions Tab	4-9
Figure 4-6 Encoding	4-10
Figure 4-7 Instruction Template	4-11
Figure 4-8: Functions Tab	4-13
Figure 4-9: Instruction DB Tab.....	4-16
Figure 4-10: XML Doc Tab.....	4-17

1 Introduction

CEVA-Xtend™ SDT enables the user to define new instructions which he can implement in the new CEVA-Xtend units. The user can define new switches and variables in addition to the basic encoding area that is different between instructions.

Using an automatic process, the user can define new instructions for the CEVA-Xtend units, create a Database and Simulator CEVA-Xtend DLLs, and by using them is able to:

- Use these instruction in assembly code
- View them in the Debugger disassembly window
- View and manually set the value of registers using CLI commands
- Simulate the behavior of the new instructions in both instruction set and cycle accurate modes



In order to enable automatic capability of generating the required DLLs, the following tools are provided:

- CEVA-Xtend Designer, which allows fast and easy creation of the descriptor XML file that defines the Xtend module.
- XML parser utility which generates C++ source files which have all the information regarding the CEVA-Xtend instructions
- CEVA-Xtend.bat batch file that runs XML parser utility and creates Database DLL and CEVA-Xtend Simulator DLL.

2 Installation

In order to allow proper execution of the CEVA-Xtend, the following are required:

1. Perl 5.10
2. Adding Perl\bin directory to the PATH
3. Microsoft Visual 2013 C++
4. Adding devenv (Microsoft Visual 2013 main interface) to the PATH

3 Creating CEVA-Xtend Module

To Create a new CEVA-Xtend module:

1. Open the CEVA-Toolbox IDE
2. From the top bar menu choose “CEVA Tools”, and “Xtend Editor”

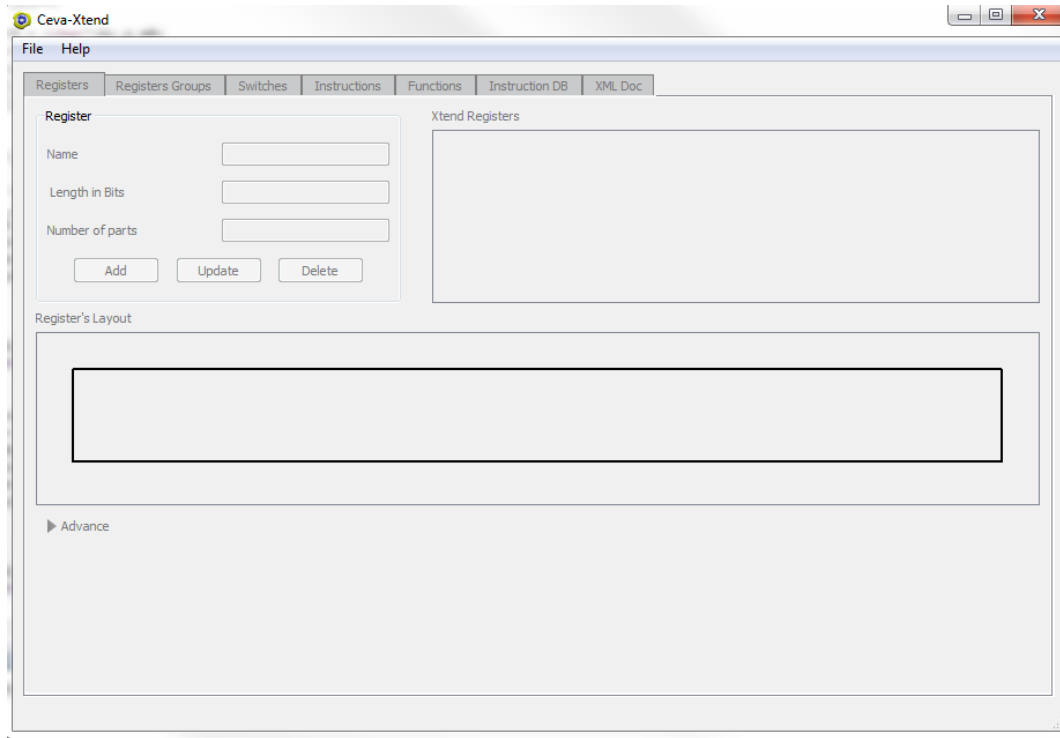


Figure 3-1: The CEVA-Xtend Editor

3.1 Define New Instructions in CEVA-Xtend Designer

Defining new instruction and their behavior is very intuitive and trivial and can be done through the CEVA-Xtend Designer project. The process should go through all the tabs of the designer as seen in Figure 3-1 and can be finalized by building the project.

For more details please refer to the description in the following chapter.

3.2 Create an Application that Uses the New Instructions

After we finished defining the Xtend module with the CEVA-Xtend Designer, we need to build it.

1. Through the CEVA-Xtend designer save the Xtend XML file into the requested directory.
2. Create a sim file for each of the functions, according to the format described in chapter 5.
3. The directory hierarchy of the files should look like this:

```
- sim_files
  |- <file1>.sim
  |- <file2>.sim
  |- <.....>.sim
- <xml_file>.xtnd
```

All the sim files should be under the sim_files directory, while the xml file should be saved at the root directory with an “.xtnd” suffix.

4. Copy the “CEVA-Xtend” directory, located at the tools “cd %<Core>TOOLS% “ to your xml file directory.
5. In the “CEVA-Xtend” run “Generate_Xtend.exe <xmlFilePath> -<Core Type>”. This creates CEVA-Xtend Simulator DLL and CEVA-Xtend Database DLL. These libraries automatically are placed into CEVA-Xtend tools directory.
6. XML file and Simulation files location is relative to the execution folder.
7. Following generation the new DLL’s process is completed, the tools now support the new instructions.

After the build of the Xtend module is successful, we can start using the new instructions we defined in the CEVA-Xtend Designer. Create a new project in the workspace which can use the new Xtend module.

*** Note: Each time you change something in the CEVA-Xtend Designer you need to first build the Xtend module and then build the application.**

4 CEVA-Xtend Designer Operation

4.1 Overview

CEVA-Xtend™ designer objective is to create an easy and intuitive GUI to enable users to design their own external hardware. CEVA-Xtend architecture design enables users to add new instructions and registers to the CEVA-XC™, CEVA-TL4xx™ and CEVA-XM4™ core families through the CEVA-Xtend mechanism using this new graphical user interface (GUI).

This reference guide describes the features of the CEVA-Xtend Designer, implementation guidelines and also describes the various views, menus and toolbars of the GUI.

The CEVA-Xtend designer GUI is divided to tabs, each tab addresses a different design functionality of the CEVA-Xtend.

- **Registers** – The user can add, edit or delete registers.
- **Registers Groups** – The user can group registers in order to use such group as source/destination of an assembly instruction.
- **Switches** – Enables adding options for assembly instruction (for example a signed/unsigned switch).
- **Instructions** – Can define new instructions that will use the various registers groups and switches.
- **Functions** – Write/Edit the behavior of the assembly Instructions.
- **Instruction DB** – A tables that summarizes all the instructions and their Huffman coding.
- **XML Doc** – A two way editor that represents the tabs

Warning - Wrong XML editing can cause an unrecoverable error, all changes should be made in CEVA-Xtend Designer tabs.

4.2 Registers Tab

The Registers tab allows the user to define new registers in the CEVA-Xtend mechanism.

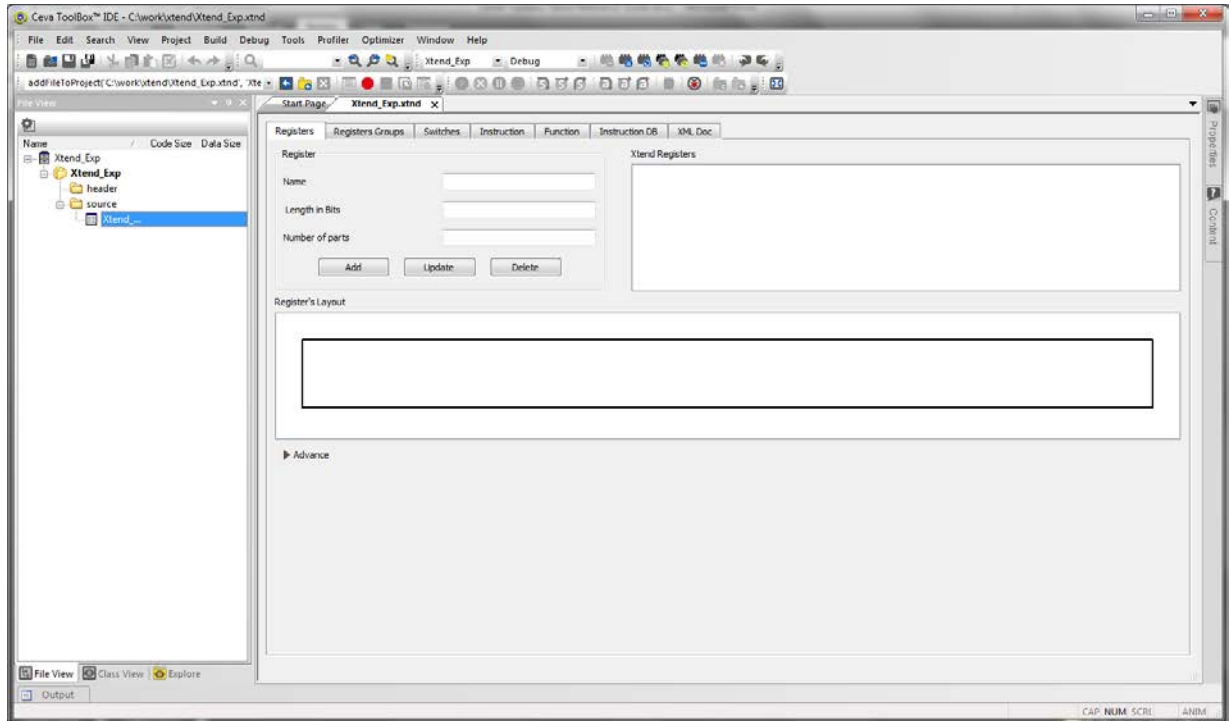


Figure 4-1: Register Tab - Simple Mode

4.2.1 Register fields

Register group contain several fields:

- **Name** – The name of the register.
- **Length** – Length in bits of the register.
- **Number of parts** – The number of parts that the register has.

It also contains 3 buttons:

- **Add** – create a new register and add it to the registers list.
adding new register must meet the following conditions:
 - Register name must be unique.
 - Length is at least 1 bit.
 - Number of part between 1 and length size.
- **Update** – updates the selected register from the register list. Once a register is selected all the fields in the register group are filled with the register information and are available for changes. When using multi selection the name field is grayed out and the length and number of parts fields are available for changes. In case the selected register does not have the same values in length or number of parts, the

fields that differ are also grayed out and are not updated. In any case, the update transaction fails if the fields do not comply with the rules of the Add button.

- **Delete** – deletes the selected register from the register list. When using multi selection it is possible to delete several selected registers at once.

4.2.2 Register List

This list holds all the register created. When a register is selected from the list it updates the fields in the register group and in the register visualization display.

It is possible to have multi selection in the register list using the standard multi selection method (Ctrl', Shift).

4.2.3 Register Visualization Display

This box holds the graphical display of the register from the register group. The register image is generated automatically when all the fields are filled. The size of the whole register is the length and the register is divided automatically by the number of parts parameter. Each part is assigned by a default name which consists of the name of the register and a sequential number starting from 0 up until the number of parts N.

Each separate part of register can be double clicked and enables to manually change the default name of the register part.

Each separator line has an indication above it that enables the user to manually change the size of the register part by moving the separator left or right. When moving the indication and changing the register part size a number below the separator line also changes and indicates the current ending point of the register part before the separator and the beginning point of the register part after the separator.


When moving the indication that is at 0, a new register part is automatically generated. This also affects the number of register parts in the register group increasing it by one. If the left most indication that is not on 0 is shifted left until zero, the number of register parts decreases by one. When adding or removing register parts this way, the register parts names are not changed.

When not all of the fields are populated, the display remains empty.

When in multi-selection mode, the register parts names are not visible.

4.2.4 Advance Mode

When pressing the advance button, a table that holds all the register parts appears.



Register's Layout

▼ Advance

Register Part Table

Name	Start Bit	Length
x3_0	0	8
x3_1	8	8
x3_2	16	8
x3_3	24	8

Figure 4-2: Register Tab - Advanced Mode

The Register part Table is a direct translation of the register graphical display.

The table contains the following columns:

- **Name** – The name of the register part.
- **Starts Bit** – In which bit does the register part start.
- **Length** – The length of the register part.

When the number of register parts is 0 the table is empty.

Each value in the table can be changed by double clicking on it.

The table's rows can be changed through the menu, which is popped up when right clicking above any row on the table. The right click menu contains 'add row' and 'delete row' and also marks the relevant target row below as an indication for the user. The 'delete row' deletes the marked row and the 'add row' adds a new row to the table following the marked row.

The minimal length allowed for each register part is one.

4.3 Registers Groups Tab

The Registers Groups tab enables the user to define a new group of registers and define the encoding for each register in the group.

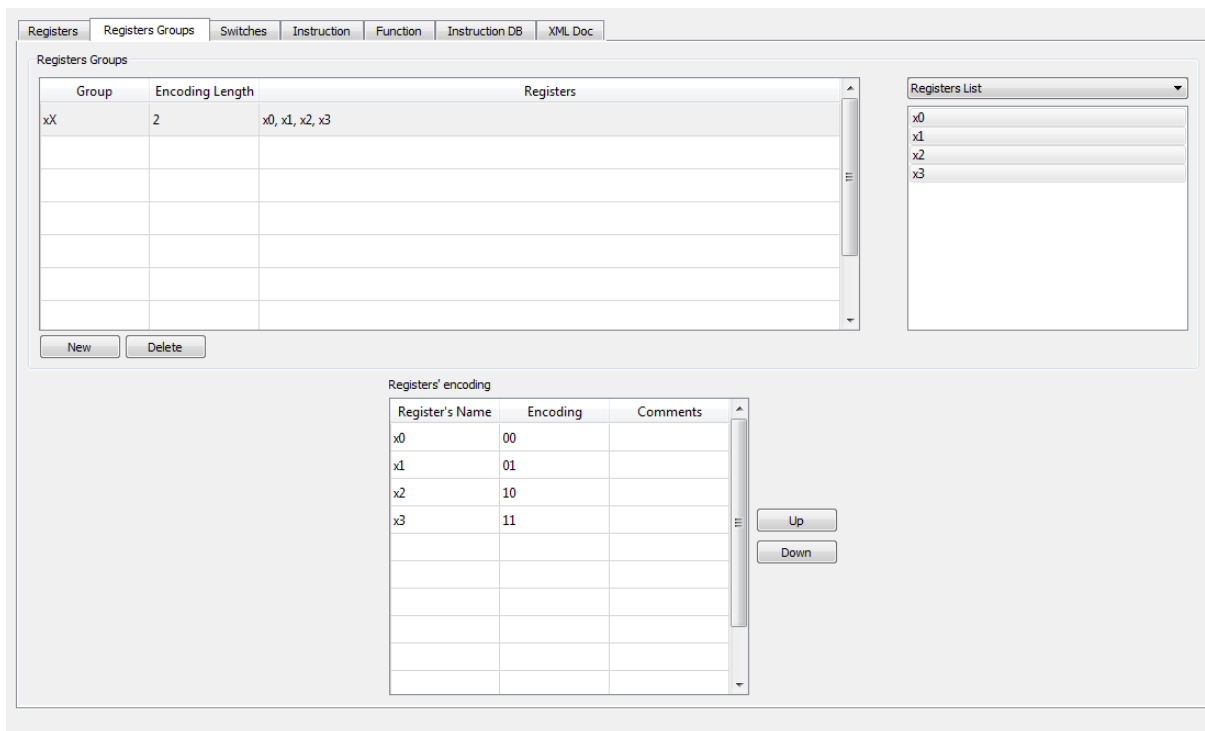


Figure 4-3: Registers Groups Tab

4.3.1 Registers Group

The Registers Group holds two objects:

- **Groups Table** – Holds all the groups that the user already defined.
- **Register List** – Holds all the registers that are defined by the user in the Registers Tab.

The Groups Table consists of three columns:

- **Group** – The name of the group.
- **Encoding Length** – The length of the encoding required for the amount of registers.
- **Registers** – A sorted list of all the registers that are selected in the group.

There are two modes to add new groups:

- **Manually** – Double click on any field in the table enters edit mode. In this mode the line that the field was selected is highlighted and the values can be changed.
- **Automatically** – Select registers from the register list (Support multi select). After the desired registers are selected, right click and press add new registers group. Now the user is in edit mode and a new row is added to the table, and be filled automatically by the following rules:
 - The Type field is the longest common prefix of all the selected registers with the postfix of X. If X is already taken then it is filled with the Y, Z, W. If there is no common prefix then the prefix *tmp* is selected. If a *tmp* prefix already exists the next one is *tmp1...n*.
- Encoding Length filled: Ceiling ($\log_2(\text{Number of registers selected})$)

Exiting edit mode is done by pressing Return or clicking with the left mouse button anywhere outside the line in the Groups Table or register list. The edit mode cannot end if one of the fields in the Groups Table is empty. When the escape button is pressed during editing, the program exits the edit mode without saving.

When pressing once on a single line in the Groups Table it is marked and the registers list shows only the registers that are relevant to that line.

When double clicking on the Registers column in the Groups Table or on the Registers List when a line is selected in the Groups Table all the registers are shown in the Registers List. The already selected registers are marked with blue background and the rest has a white background. The selection of registers in the Registers List is done by single clicking on the register, thus toggling the selection of the register.

At the top of the Registers List there is a combo box that has two options. The first is the list of all the registers and the last is a list of all the registers parts.

4.3.2 Encoding Group

The Encoding Table holds a list of all the registers and their encoding for a specific selected line in the Groups Table.

The Encoding Table includes three columns:

- **Register Name** – A list of all the registers that are selected in the selected line in the Groups Table.
- **Encoding** – The proposed encoding for each register.
- **Comments** – Enables the user to add comment to each register.

This table is generated automatically when a line is selected in the Groups Table. The encoding is generated starting from 0 for the first line and incremented until the encoding is filled according to the Encoding Length. All the registers in this table are alphabetically sorted.

The only elements that can be changed in this table are the registers and comments. When selecting a line in the table a register is selected, by using the arrows the register can go up or down in the table and be aligned with desired encoding. It is possible that some encoding stays empty but it is not possible that a register does not have an encoding.

4.4 Switches Tab

The Switches tab enables the user to define new switches in the CEVA-Xtend mechanism.

Switches

Name: op

Encoding Length: 3

Add Update Delete

Xtend Switches

op
3w

Switch' options encoding

Option Name	Encoding	Comments
neg	000	Less than zero
	001	
eq (default)	010	equal to
	011	
gt	100	greater than
lt	101	less than
	110	
	111	

Default

Up

Down

Figure 4-4: Switches Tab

4.4.1 Switches Group

Switches group contains the following fields:

- **Name** – The name of the switch.
- **Encoding Length** – The length of the encoding.

It also contains the following buttons:

- **Add** – create a new switch and add it to the switches list. The add transaction fails in the following scenarios and a warning message pops up:
 - Switch with the same name already exist.
 - Encoding length is lower than 1.
- **Update** – update the selected switch from the switches list. Once a switch is selected all the fields in the switches group are filled with the switch information and is available to changes. In any case, the update transaction fails if the fields do not comply with the rules of the Add button.
- **Delete** – delete the selected switch from the switches list. When using multi selection it is possible to delete several selected switches at once.

4.4.2 Switches List

This list holds all the switches created. When a switch is selected from the list it updates the fields in the switches group and in the Option Table.

It is possible to have multi-selection in the switches list using the standard multi selection method (Ctrl, Shift). The multi selection is used here for deleting switches.

4.4.3 Option Table

The Option Table holds a list of all the switch options and their encoding for a specific selected line in the Switches List.

The Option Table has three columns:

- **Option Name** – A list of all the options that are relevant to the switch.
- **Encoding** – The proposed encoding for each switch option.
- **Comments** – the user can add a small explanation for each switch.

This table is generated automatically when a line is selected in the Switches List. The encoding is generated starting from 0 for the first line and incrementing until the encoding is filled according to the Encoding Length. The name column is empty at this stage.

By double clicking on the name column, a switch option can be added. A switch option with the same name cannot be added if it already exists.

When right clicking on any switch option a menu appears with the “set as default” option. In this case the default switch option is marked with a “(default)” notation.

The switch option can move up or down using the arrows. The default property stays with the switch option as it moves in the table.

4.5 Instructions Tab

Instruction Tab allows the user to define new instructions by selecting the templates and units of the new instruction, assigning opcodes, writing the syntax of the instruction and connect them to simulation functions.

The Instruction Tab consists of three major objects:

- **Instruction group** – The user can set all the basic definitions of the instruction.
- **Instruction template** – Write the instruction according to the template selected.
- **Opcode display** – Display the opcode of the instruction.

4.5.1 Instruction Group

In this section the user can create a new instruction or update an already existing one.

The screenshot shows the 'Instructions Tab' in the CEVA-Xtend Designer. It features several input fields and a large opcode display grid.

Instruction Fields:

- Instruction Name:** add
- Function Name:** (empty)
- Unit:** TL4tx
- Template:** TL4tx_3
- Automatic Encoding:** ☒
- Encoding:** 0
- Buttons:** Add, Update, Delete

Instruction Template:

To add switches and/or registers to the instructions' syntax, click on 'Switch' or 'Reg' items below

add {3w} Switch Switch xX Reg GRF Reg ar Reg

Opcode Display:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
0	0	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	1	0	0	0	ar_accr				0	ar_	grf_grf					
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	3w	xX	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	

Figure 4-5: Instructions Tab

In this group the user has a few fields that define the instruction.

Instruction name dropdown box contains all the instructions already created, but also holds at the top the “new...” instruction. If the “new...” instruction is selected, the user can write over it and thus create a new instruction. When a new instruction is created all the fields in the Instruction tab are cleared. If an already existing instruction is selected, then the last values of the instruction return to all the fields in the instruction tab.

Function name dropdown box connects the instruction with its simulation implementation function, created in the Function Tab. If no function is available, this field can remain empty.

Unit dropdown box selects which unit the instruction is relevant to. The units are defined according to selected core when creating the CEVA-Xtend project.

Template dropdown box select which template the user would like to use for the instruction. The option changes according to the selected unit.

When the dropdown box is open and the mouse cursor is over a template, then the display changes in the top part of the Instruction template object in this tab. The check box for automatic encoding is set by default. If it is not set then the encoding input appears beneath it. The user is able to insert the encoding of the instruction manually. The contention checking is done “on-line” when the user inputs the encoding. If an invalid encoding is entered, then the input box background changes to red and the update button is grayed out.

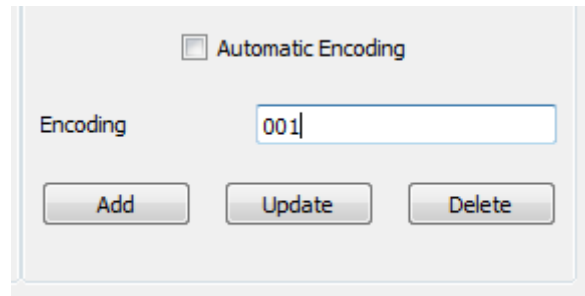


Figure 4-6 Encoding

When the user is in automatic mode, the Huffman coding (opcode) is generated automatically for each instruction, according to the unused opcode. The length of the opcode is the minimal size possible according to the number of instructions under each template starting from size of one. As more instructions are added to the template, the size of the opcode increases so the opcode can facilitate more instructions.

The calculation is: Ceiling (Log2(Number of instructions per template))

For example:

With three instructions, then the length of the opcode is two. When the number of instructions exceeds four then the length changes to three and all the instruction under that template are set with new opcodes with the new length.

At the bottom of this group there are three buttons:

- **Update** – If an existing instruction has been changed, the user can press this button to update it and save the changes.
- **Delete** – The user can delete an existing instruction. A warning message pops warning the user about the deletion.
- **Add** – The user can add a new instruction.

4.5.2 Instruction Template

This group enables the user to view the template selected and edit the instruction. This object is divided into two parts.

The top most part shows the user the selected template. All the fields that are editable are grayed out and the constant fields are in black. The lower part displays the selections done in the top part.

As the mouse pointer passes above each field in the top part, the field changes to a dropdown box that displays the available possibilities for each field. Once the field is selected, the dropdown list opens and the field remains selected. The list is a multi-select list. If nothing is selected in the list,

the display in the bottom part is not changed. If the user presses the return button or left click anywhere else, the selection is saved. If the user presses the Escape button, nothing is changed and the shape of the dropdown box returns to its text form. Once an editable field is changed, it is highlighted in the background to indicate the user it was changed.

The Instruction Name is updated by default from the Instruction Name field in the Instruction box. If the user clicks the template on the Instruction Name, an input text appears and the name can be edited. The behavior for saving and escaping is the same as the dropdown box.

Following are examples of the coloring with the instruction itself:

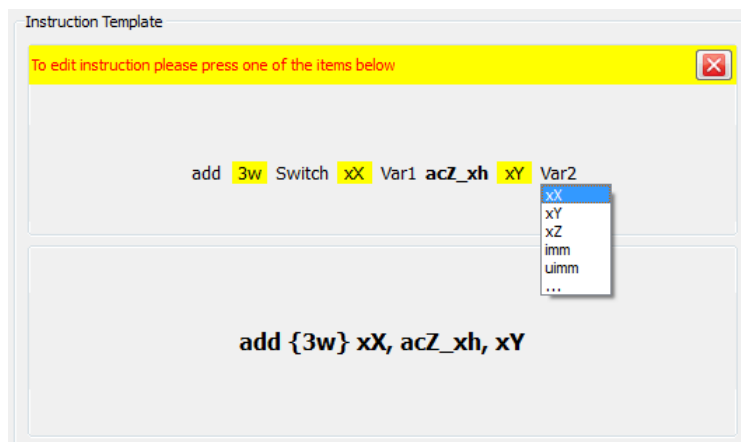


Figure 4-7 Instruction Template

4.5.3 Opcode Display

The Opcode display enables the user to observe the encoding of the instruction created.

When no instruction has been edited yet the display is of the basic selected template, all the system bits are unified and the background marked with the writing “system”.

All the constant pre assigned bits of the template are in single bit mode, they have gray background and the assigned encoding writing in them.

All the template accumulators and predicate, if exist, are bit unified for each one with a different background color. On each unified field there is the name of it owner.

All the remaining possible bits that can be changed are in single bit mode with white background and an “x” on them. The application treats them as a concatenated encoding string even though they are displayed as separate fields.

In order to fill this concatenated string the user starts with the instruction encoding entered in the Instruction group. The digits are changed from “x” to the encoding on the display, the background color changes from white but the bit remains in single mode.

Following the instruction encoding, switches and variables are added. The encoding is taken from the pool of switches or variables that were in the configuration file or from the newly defined switches or variables in this project. In this case the bits are unified according to the encoding size, and the background color changes and the name of the switch or variable is displayed.

In the event that more bits are required than are available in the basic template, it is possible to have an extension up to 32 bits. There are two extensions, 16 bits and 32 bits.

The 16-bit extension has a header of 6 bits and enables the user an extra of 10 bits in addition to the bits in the original template.

The 32-bit extension has a header of 10-bit and enables the user an extra 26 bits in addition to the bits in the original template.

In any case the header field is unified, the background color different than white and the unified field states “Header”. The rest of the bits behave as defined in the description above.

If the user exceeds the 32-bit extension, a warning message pops indicating that the instruction encoding is not valid since it has exceeded the maximum allowed.

Opcode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
System								0	0	1	3w	0	1	0	0	0	3w	xX	ac2_xh				xX	xY	x	x	x	x	x	x	x

4.6 Functions Tab

The Function Tab enables the user to define new function behavior that the instruction can be connected to.

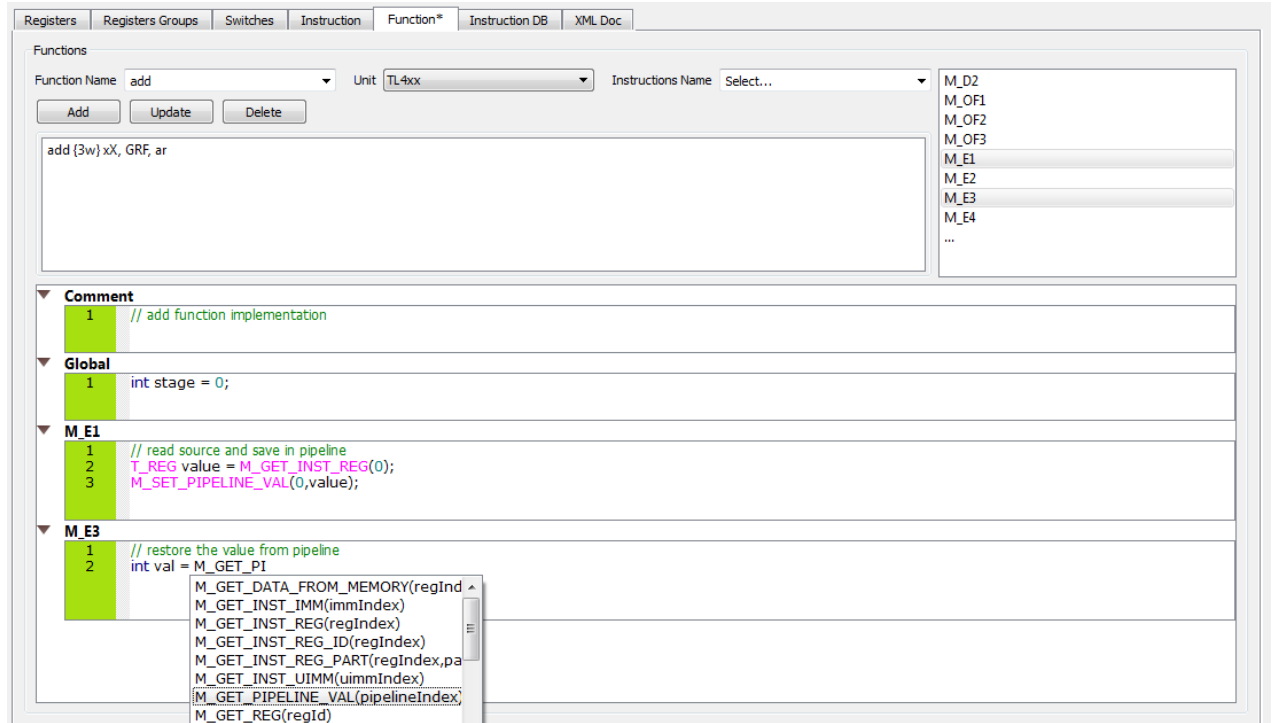


Figure 4-8: Functions Tab

The Function Tab consists of three major sections:

- **Function control** – Here the user can add a new function or select an already existing function and edit it.
- **Stage selection** – Selecting the stage relevant for each function.
- **Function Implementation** – The user actually writes the code for each stage of the function.

4.6.1 Function Control

This section contains a dropdown box that holds a list of all the already defined functions in this project. The top most option in the drop box is the new option.

When the new option is selected, a name of the new function can be inserted in the drop box. All other fields in this tab are reset. A new file is created with the function name with “.sim” extension. If not all the fields are implemented, then the user cannot leave this tab or create a new function. A warning pops indicating the missing parts.

If an already existing function is selected, then all the fields in this tab load the previous set values of that function. When an existing function is selected, the two buttons come in use. The delete button deletes the function from the project. This button also deletes the file of the function. A Warning message pops warning the deletion of the function. The update button saves the changes made to the function to the function file.

This section contains also instructions dropdown box that holds a list of all the instructions that are already implemented in the project. This list is a multi-select list. Once an instruction is selected in this list, the field of the instruction is edited and the name of the current instruction is updated there. All the names of the instruction in the list have a prefix that includes the name of the template. If instructions with different templates are selected, then a warning message pops.

4.6.2 Stage Selection

This list holds all the basic stages that a core has. This is loaded with the project from a configuration file. For each instruction (and its template) selected the relevant stages that holds synchronous read/write registers are selected and added.

Each item in the list can be selected by clicking on it. This is a multi-selection list. When a stage is selected from the list a new stage is opened for editing in the function implementation table. When a stage is deselected, then this stage is deleted from the function implementation table. If the stage is empty at the table, then the stage is just deselected. If the stage in the table has some implementation in it, then a warning message pops indicating that the stage is not empty in the table.



There are three points button at the bottom of the list, if pressed the user can add manually more stages that are not originally in the core, like E100.

4.6.3 Function Implementation

This table holds the actual implementation of the functions.

This table consists of two constant rows that hold comments for the function, describing what it does and definition of global variables used throughout the implemented function’s stages.

The other rows in the table are generated depending on the Stage selection. When a stage is selected in the stage selection list, then it is added to the table and the user can insert the implementation for that specific stage.

All the rows in this table can be collapsed just to the name of the row by pressing the  indication and fully opened by pressing the  indication.

The stages of synchronous read/write registers are available by default and actual implementations of read/write of the relevant registers are included in it automatically from the configuration file.

The user has a pool of predefined macros that helps to interact with the simulator in an easy and intuitive way. A list of all the macros and a detailed explanation of the simulation file structure can be found in section 5.5 [CEVA-Xtend Simulation File](#)

4.6.4 Simulation File Creation

The basic file should look like this:

```
{
    M_PIPE_LINE_START
    {
        M_PIPE_LINE_DEFAULT
        {
            M_PIPE_LINE_STAGE_END
        }
    }
}
```

This file has no comments, no global variables and no stages were selected.

The file created from the Function Tab that does have implementations looks like this:

```
{
    // mov src -> dst:
    M_PIPE_LINE_START
    {
        M_PIPE_LINE_STAGE(M_E1)
        {
            // read source and save in pipeline
            T_REG val = M_GET_INST_REG(0); // 0 is the index of the source register in the
                                           instruction.
            M_SET_PIPELINE_VAL(0, val); // 0 is the index of the pipeline values array (saved values
                                           to restore in a different pipeline)
            M_PIPE_LINE_STAGE_END
        }
        M_PIPE_LINE_STAGE(M_E3)
        {
            // restore value from pipeline
            int val = M_GET_PIPELINE_VAL(0);
            // write destination
            M_SET_INST_REG(1, val); // 1 is the index of the destination register in the
                                   instruction.
            M_INST_FINISHED
            M_PIPE_LINE_STAGE_END
        }
        M_PIPE_LINE_DEFAULT
        {
            M_PIPE_LINE_STAGE_END
        }
    }
}
```

In this example the comment row was filled with “mov src -> dst:”

No global variables were selected.

Only two stages were selected and filled: E1 and E3.

Each stage must end with M_PIPE_LINE_STAGE_END.

The last stage must contain the macro M_PIPE_LINE_STAGE_END at its end.

4.7 Instruction DB Tab

The Instruction DB tab enables the user to view all the instructions defined in the current CEVA-Xtend project and see clearly all the instructions Huffman codings.

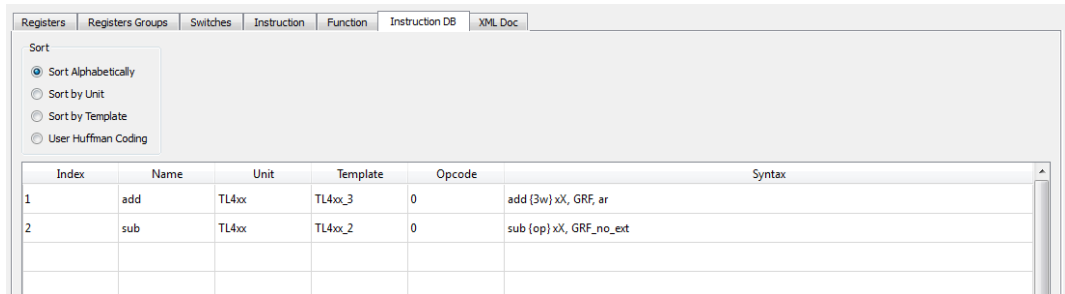


Figure 4-9: Instruction DB Tab

The Instruction DB Tab consists of three major sections:

- **Sort** – The user can sort the contention table.
- **Instructions table** – Display all the instructions and their opcodes defined in the current project.

4.7.1 Sort

The user can sort the instructions table by selecting the desired option from the radio box. The available options are:

- **Sort alphabetically** – sorts the table by the name on the instructions.
- **Huffman coding** – sorts the table according to the Huffman encoding of the instructions.

4.7.2 Instructions Table

The Instruction Table displays all the instructions and there opcode defined in the current project sorted by the selection in the Sort area.

If the sort is alphabetically, then all the instructions are sorted by their name. If the sort is by Huffman coding, then all the instructions are sorted by their encoding.

The display contains for each instruction, an incremented numerical index starting from 1, the name of the instruction and her encoding. The encoding includes the system encoding of each instruction with the background color is red. The coding of the instruction is with background color of green. The entire template constant encoding, other than the system, has a gray background. The rest are white.

In this table the user can export the content to an excel table by pressing the export button. The excel sheet holds the name on the instruction end the encoding.

4.8 XML Doc Tab

Warning - Wrong XML editing can cause an unrecoverable error, all changes should be made in CEVA-Xtend Designer tabs.

The XML Doc is a two way editor between the Registers, Variables, Switches and the Instruction tabs. Whatever is written in these tabs is stored in the XML document according to the defined structure.

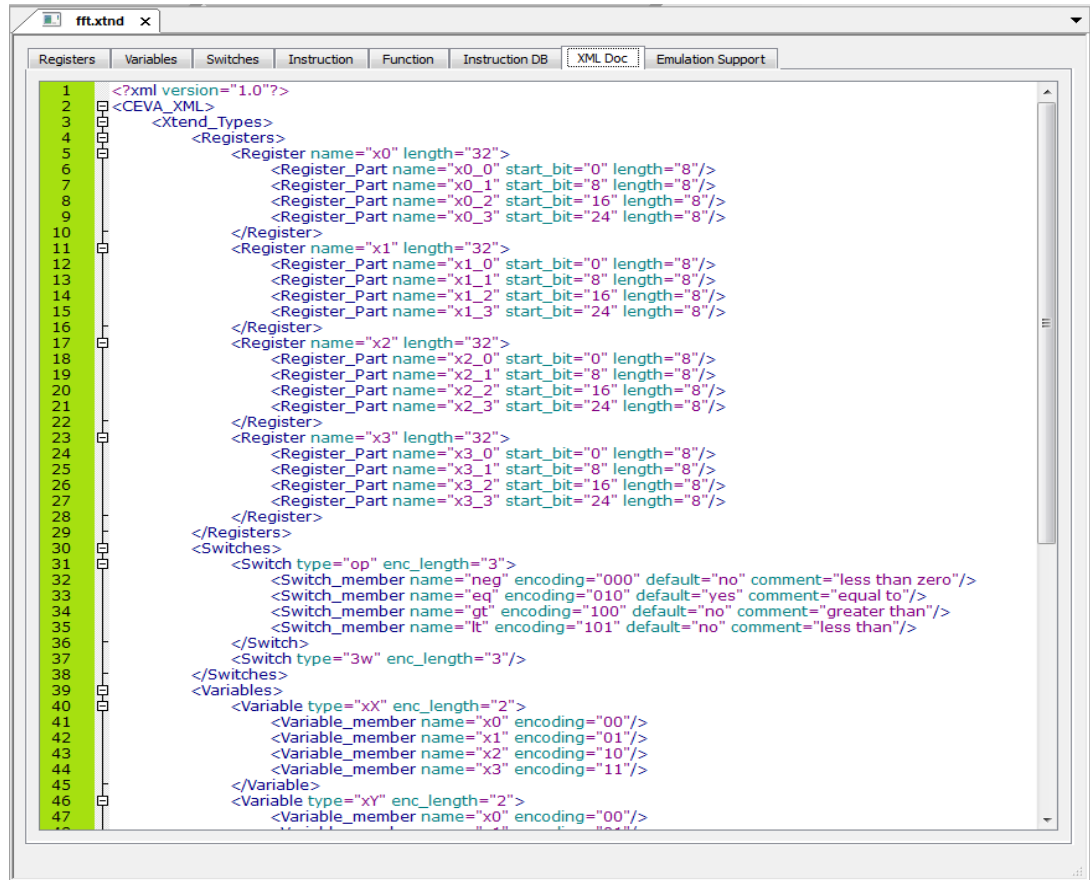


Figure 4-10: XML Doc Tab

The basic structure of the XML document is is:

```

<?xml version="1.0"?>
<CEVA_XML>
  <Xtend_Types>
    <Registers>
    </Registers>
    <Switches>
    </Switches>
    <Variables>
    </Variables>
  </Xtend_Types>
  <Units>
  </Units>
</CEVA_XML>

```

Each tab has its own structure.

4.8.1 Register Tab

The register tab structure is:

```
<Register name="f0"          length="32">
  <Register_Part name="f0_frac"      start_bit="0"   length="23"/>
  <Register_Part name="f0_exp"      start_bit="23"  length="8" />
  <Register_Part name="f0_sign"     start_bit="31"  length="1" />
</Register>
```

The structure can be without the “Register_Part”, just the “Register”.

4.8.2 Variables Tab (Registers Groups)

The variables tab structure is:

```
<Variable type="fX" enc_length="2">
  <Variable_member name="f0" encoding="00"/>
  <Variable_member name="f1" encoding="01"/>
  <Variable_member name="f2" encoding="10"/>
  <Variable_member name="f3" encoding="11"/>
</Variable>
```

4.8.3 Switches Tab

The switches tab structure is:

```
<Switch type="op" enc_length="2">
  <Switch_member name="eq" encoding="00" default="yes"/>
  <Switch_member name="ne" encoding="01" default="no"/>
  <Switch_member name="gt" encoding="10" default="no"/>
  <Switch_member name="lt" encoding="11" default="no"/>
</Switch>
```

4.8.4 Instructions Tab

The instructions tab structure is:

```
<Units>
  <Unit Name="XH0">
    <Instruction Name="cmp" Template="XH_4" Inst_Enc="">
      <Syntax description="cmp {op} fX, fX, acZ_xh"/>
      <Simulation file="..\..\sim_files\cmp_fX_fX_acZ.sim"/>
    </Instruction>
  </Unit>
</Units>
```

4.8.5 Editor Macro Support

By default, the editor is support all available macros that are used by the simulator. Please look into section 5.5.2.

5 CEVA-Xtend Manual Creation

5.1 XML File Description

The XML file needs to satisfy Database, Simulator and Debugger required information.

The XML file has two main parts:

1. CEVA-Xtend Types Definition
2. CEVA-Xtend Instruction Definition

Basic XML file structure

```
<CEVA-Xtend_Types>
    <Registers>
</Registers>
    <Switches>
</Switches>
    <Variables>
</Variables>
</Types>
<Units>
    <Unit Name= "XH0">
        <Instruction>
            <Syntax />
            <Simulation file />
        </Instruction>
        ...
    </Unit>
    <Unit Name= "XH1">
        ...
    </Unit>
    <Unit Name= "VXH">
        ...
    </Unit>
</Units>
```

5.2 CEVA-Xtend XML File Definition

1. **CEVA_XML**: A node marking the root of the xml tree
2. **Xtend_Types**: Holds a list of CEVA-Xtend Types
 - **Registers**: Holds a list of CEVA-Xtend registers which can be accessed by the CEVA-Xtend units
 - a. **Register**: Holds a list of the register parts, and contains the name of the CEVA-Xtend register, and its total length in bits
 - b. **Register_Part**: Contains the part's name, its start bit in the register and its length in bits
 - **Switches**: Holds a list of CEVA-Xtend Switch types
 - a. **Switch**: Contains a type name, encoding length in bits, and holds a list of the member switches
 - i. **Switch_member**: Contains the name of the switch, its encoding, and a flag which determines whether this member is the default member of this switch which is used if a switch member is not written in the instruction syntax
 - **Variables**: Holds a list of CEVA-Xtend Variable types and CEVA-Xtend Immediate types
 - a. **Variable**: Contains a type name, variable size in bits, encoding length in bits, and holds a list of the member variables
 - i. **Variable_member**: Contains the name of the variables, and its encoding.
The variable may be a register or a register part
3. **Units**: Holds a list of units
 - **Unit**: Contains the unit name and holds a list of instructions included in this unit. The unit name can be only one of the three CEVA-Xtend units available – and it must appear not more than once.
 - i. **Instruction**: Contains the instruction name, the Template it is based on, the instruction base Instruction encoding, the DPS value, and holds a list of the instruction attributes.
The DPS value can be one of the following: E2, E3, E4, E5 – but they have to fit their template.
The instruction name is unique in the unit: it must not appear more than once in the same unit.
 - **Syntax**: Contains a single string describing the order of the switches and variables.
Immediate variables are described only here without being mentioned in the variable definition.
 - **Simulation**: Contains the file name of the C++ source file which holds the implementation of this instruction.

5.3 CEVA-Xtend Types Example

```

<Xtend_Types>

  < Registers>

    < Register    name="uv0"  length="20">
      < Register_Part  name="uv0l" start_bit="0"  length="8" />
      < Register_Part  name="uv0h" start_bit="8"  length="8" />
      < Register_Part  name="uv0e" start_bit="16" length="4" />
    </Register >
    < Register    name="uv1"  length="20">
      < Register_Part  name="uv1l" start_bit="0"  length="8" />
      < Register_Part  name="uv1h" start_bit="8"  length="8" />
      < Register_Part  name="uv1e" start_bit="16" length="4" />
    </Register >
    < Register    name="uac0"      length="128" />
    < Register    name="uac1"      length="128" />
    < Register    name="uac2"      length="128" />

  </Registers>

  <Switches>
    <Switch type="usX" enc_length="2">
      <Switch_member name="us0" encoding="00" default="yes"/>
      <Switch_member name="us1" encoding="01" default="no" />
      <Switch_member name="us2" encoding="10" default="no" />
    </Switch>
  </Switches>

  <Variables>
    <Variable type="uvXh" enc_length="1">
      <Variable_member name="uv0h" encoding="0" />
      <Variable_member name="uv1h" encoding="1" />
    </Variable>

    <Variable type="uacX" enc_length="2">
      <Variable_member name="uac0" encoding="00" />
      <Variable_member name="uac1" encoding="01" />
      <Variable_member name="uac2" encoding="10" />
    </Variable>
  </Variables>

</Types>

```

5.4 Instruction Definition Example

```

<Units>
  <Unit Name= "XH0">
    <Instruction Name="userInst01" Template="XH_0" Inst_Enc="1101101">
      <syntax="userInst01 {usX} acX, acY, uaXh, uimm16, uacX, acZ">
        <Simulation file="userInst01.sim"/>
      </Instruction>
    <Instruction Name="userInst02" Template="VXH_0" Inst_Enc="0011010">
      ...
    </Instruction>
    ...
  </Unit>
  <Unit Name= "XH1">
    ...
  </Unit>
  <Unit Name= "VXH">
    ...
  </Unit>
  ...
</Units>

```

5.5 CEVA-Xtend Simulation File

5.5.1 Simulation File Structure

The user defines CEVA-Xtend instructions that can be supported in the core simulator. In order to do so, the user should supply a unique simulation file that describes each instruction's functionality.

The simulation file should be written in C++ language, using specific statements (macros) described below. These special macros enable the user to set or get CEVA-Xtend registers values, access the XC registers, use the instruction's switches, etc.

The User's simulation file is translated into two execution functions – one for the simulator's instruction-set mode and the other for the cycle-accurate mode. In order to support both modes, the simulation file should detail the pipeline stage of each instruction (both synchronous and asynchronous). In the instruction-set mode, all stages are performed at once, while in the cycle-accurate mode each stage is performed in the proper cycle.

When receiving a value of particular source and using it on another pipeline stage, it should be saved in special structure by `M_SET_PIPELINE_VAL` macro. This value should be received by `M_GET_PIPELINE_VAL` macro when is used on another pipeline stage. The definition of these macros can be found on the next paragraph. The examples of using of this macro's can be found in examples.

The following example describes the basic structure of the simulation file:

```

M_PIPE_LINE_START
{
    M_PIPE_LINE_STAGE(M_D3)
        [User C++ Code Block]

    M_PIPE_LINE_STAGE(M_D4)
        [User C++ Code Block]

    M_PIPE_LINE_STAGE(M_E1)
        [User C++ Code Block]

    M_PIPE_LINE_STAGE(M_E2)
        [User C++ Code Block]

    ...
}

```

5.5.2 Simulation File Macros

5.5.2.1 Pipeline Macros

Purpose: Sets the register value in the pipeline.
Input: index - Index of the register in the instruction
value - register value
Output: None.

M_SET_PIPELINE_VAL(index, value)

Purpose: Get the register value from the pipeline.
Input: index - Index of the register in the instruction
Output: Value of the register.

M_GET_PIPELINE_VAL(index)

Purpose: Marks the last stage of the instruction.
Input: None.
Output: None.

M_INST_FINISHED

5.5.2.2 REGISTERS MACROS

Purpose: Returns register value. Can return int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.
Input: Index of the register in the instruction
Output: Value

M_GET_INST_REG (regIndex)

Purpose: Sets the register value. Can get int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: index of the register in the instruction.
Value.

Output: None.

M_SET_INST_REG(regIndex, val)

Purpose: Returns registers part value. Can return int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: Index of the register in the instruction
Part index within the register

Output: Value

M_GET_INST_REG_PART (regIndex, partInd)

Purpose: Sets the registers part value. Can get int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: index of the register in the instruction.
Part index within the register
Value.

Output: None.

M_SET_INST_REG_PART(regIndex, partInd, val)

Purpose: returns ID of the register according to its index in the instruction Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: Index of the register in the instruction

Output: ID of the register.

M_GET_INST_REG_ID(regIndex)

Purpose: Returns register value. Can return int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: ID of the register.

Output: Value.

M_GET_REG(regId)

Purpose: Sets the register value. Can get int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: ID of the register, register value

Output: None.

M_SET_REG(regId, val)

Purpose: Returns registers part value. Can return int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: ID of the register.
Part index within the register

Output: Value.

M_GET_REG_PART(regId, partInd)

Purpose: Sets the registers part value. Can get int, int64 or T_REG. Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: ID of the register, register value
Part index within the register

Output: None.

M_SET_REG_PART(regId, partInd, val)

Purpose: Returns the length of the register in bytes Notice that the register index refers to the registers only, not to the total variable index which includes immediate values as well.

Input: ID of the register.

Output: Length of the register in bytes

M_GET_REG_LEN_IN_BYTES(regId)

Purpose: Returns a register ID when getting the register name as input

Input: Register name given as a string

Output: Register ID

M_GET_REG_ID_BY_NAME(char*)

Purpose: Get the value of the predicate of the current instruction

Input: None.

Output: Value

M_GET_PREDICATE_VAL

Purpose: Get the value of the vector predicate of the current instruction

Input: None.

Output: Value

M_GET_V_PRDICATE

Purpose: Get the specific bit value in th vector predicate of the current instruction.

Input: None.

Output: Value.

M_GET_V_PRDICATE_VAL(bitNumber)

 Purpose: Returns the instruction's predicate value.
 Input: None.
 Output: bool – predicate's value. If the instruction has no predicates
 it'll return "true".

M_GET_PREDICATE_VAL

Supported only in TL4xx cores

 Purpose: Returns data from memory according to the provided operand's index and perform the operand post
 modification (If there is any)
 Input: index of the register in the instruction
 Output: INT64 – Returns value

M_GET_DATA_FROM_MEMORY(regIndex)

Supported only in TL4xx cores

 Purpose: Sets to the memory data according to value and provided operand's index and perform the operand
 post modification (If there is any)
 Input: INT64 value
 index of the register in the instruction
 Output: none

M_SET_DATA_TO_MEMORY(val, regIndex)

Supported only in TL4xx cores

 Purpose: Returns data from stack according to the stack pointer 1dw and performs the sp post modification
 Input: LONG * - pointer to LONG value to be filled (1dw)
 Output: none

M_POP_DW_FROM_STACK(valueArr)

Supported only in TL4xx cores

 Purpose: Returns data from stack according to the stack pointer 2dw and performs the sp post modification
 Input: LONG * - pointer to LONG values to be filled (2dw), the 'arrSize' must be 2
 Output: none

M_POP_2DW_FROM_STACK(valuesArr, ArrSize)

Supported only in TL4xx cores

 Purpose: Returns data from stack according to the stack pointer 4dw and performs the sp post modification
 Input: LONG * - pointer to LONG values to be filled (4dw), the 'arrSize' must be 4
 Output: none

M_POP_4DW_FROM_STACK(valuesArr, ArrSize)

Supported only in TL4xx cores

 Purpose: Sets data to stack according to the stack pointer 1dw and performs the sp post modification
 Input: LONG value - value to be pushed (1dw)
 Output: none

M_PUSH_DW_TO_STACK(valueArr)

Supported only in TL4xx cores

Purpose: Sets data to stack according to the stack pointer 2dw and performs the sp post modification
 Input: LONG * - pointer to LONG values to be pushed (2dw), the 'arrSize' must be 2
 Output: none

M_PUSH_2DW_TO_STACK(valuesArr, ArrSize)

Supported only in TL4xx cores

Purpose: Sets data to stack according to the stack pointer 4dw and performs the sp post modification
 Input: LONG * - pointer to LONG values to be pushed (4dw), the 'arrSize' must be 4
 Output: none

M_PUSH_4DW_TO_STACK(valuesArr, ArrSize)

5.5.2.3 INSTRUCTION INFO MACROS

Purpose: Get immediate value in the current instruction. Notice that the immediate index refers to the immediates only, not to the variable index.
 Input: Index of the immediate in the instruction.
 Output: Value.

M_GET_INST_IMM(index)

Purpose: Get unsigned immediate value in the current instruction. Notice that the immediate index refers to the immediates only, not to the variable index.
 Input: Index of the immediate in the instruction.
 Output: Value.

M_GET_INST_UIMM(index)

5.5.2.4 PIPE LINE STAGES MACROS

Purpose: Marks the start of the instruction stages description
 Input: None.
 Output: None.

M_PIPE_LINE_START

Purpose: Marks the start stage description
 Input: Stage name.
 Output: None.

M_PIPE_LINE_STAGE(stage)

Purpose: Marks the start of the default stage description
 This stage is executed when there is no other stage to execute (until M_INST_FINISHED is called).

Input: None.
Output: None.

M_PIPE_LINE_DEFAULT

Purpose: Marks the end of the stage description
Input: None
Output: None.

M_PIPE_LINE_STAGE_END

5.5.2.5 SWITCHES MACROS

Purpose: Decides if a specific switch was used in the current
 instruction
Input: switchStr – a string of the desired switch.
Output: bool – “true” if the switch is used in the current
 instruction, “false” otherwise.

M_IS_SWITCH_USED (switchStr)

5.5.3 Simulation File Type

T_REG – Register value type. Its length in bits is not limited, however arithmetic manipulations over it need to be done using access to its parts. The access can be done using int or int64.

Example:

```
T_REG myReg;
```

```
...
```

```
int var32 = myReg[i];
```

```
Int64 var64 = myReg[i];
```

5.5.5 Simulation File - Example 1

XH0.mov fX, acZ_xh

```

M_PIPE_LINE_START
{
    M_PIPE_LINE_STAGE(M_D3)
    {
        int      srcRegId    = M_GET_INST_REG_ID(0);
        T_REG val            = M_GET_REG(srcRegId);

        M_SET_PIPELINE_VAL(0, val);
        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_STAGE(M_D4)
    {
        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_STAGE(M_E1)
    {
        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_STAGE(M_E2)
    {
        if (M_GET_PREDICATE_VAL == false)
        {
            M_INST_FINISHED
            M_PIPE_LINE_STAGE_END
        }

        int      dstRegId    = M_GET_INST_REG_ID(1);
        T_REG val            = M_GET_PIPELINE_VAL(0);

        M_SET_REG(dstRegId, val);

        M_INST_FINISHED
        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_DEFAULT
    {
        M_PIPE_LINE_STAGE_END
    }
}

```

Description:

- Stage D3: receives the source register value and setting it in the pipeline
- Stage D4: empty stage
- Stage E1: empty stage
- Stage E2: Checking the predicate value. If it is 0 – the instruction is finished. Otherwise the destination value is set. Instruction is finished.

5.5.6 Simulation File – Example 2

XH0.add fX, fX, fX

```

M_PIPE_LINE_START
{
    M_PIPE_LINE_STAGE(M_D3)
    {
        int    srcRegId    = M_GET_INST_REG_ID(0);
        T_REG  val         = M_GET_REG(srcRegId);
                                M_SET_PIPELINE_VAL(0, val);

        srcRegId    = M_GET_INST_REG_ID(1);
        val         = M_GET_REG(srcRegId);
        M_SET_PIPELINE_VAL(1, val);

        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_STAGE(M_D4)
    {
        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_STAGE(M_E1)
    {
        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_STAGE(M_E2)
    {
        M_PIPE_LINE_STAGE_END
    }
    M_PIPE_LINE_DEFAULT
    {
        int    dstRegId    = M_GET_INST_REG_ID(2);
        T_Reg  currentValue = M_GET_REG(dstRegId);

        If (currentValue == 0x50)
        {
            T_REG  val =    M_GET_PIPELINE_VAL(0) +
                            M_GET_PIPELINE_VAL(1);
            M_SET_REG(dstRegId, val);
            M_INST_FINISHED
        }
        M_PIPE_LINE_STAGE_END
    }
}

```

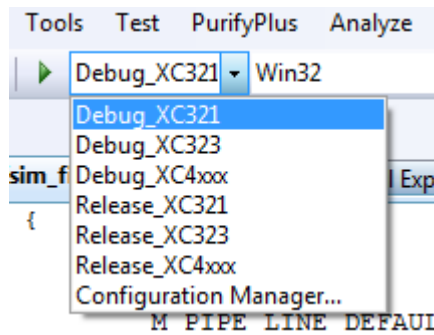
Description:

- Stage D3: receiving the source register values and setting them in the pipeline.
- Stage D4: empty stage.
- Stage E1: empty stage.
- Stage E2: empty stage.
- Default stage: Checking the destination value. If it is not 0x50 – do nothing. Otherwise, receive the source values from the pipeline, perform addition and set the result in the destination.
- This instruction is executed until the destination value is set to 0x50. When the destination is 0x50, the sum of the sources is written to the destination.

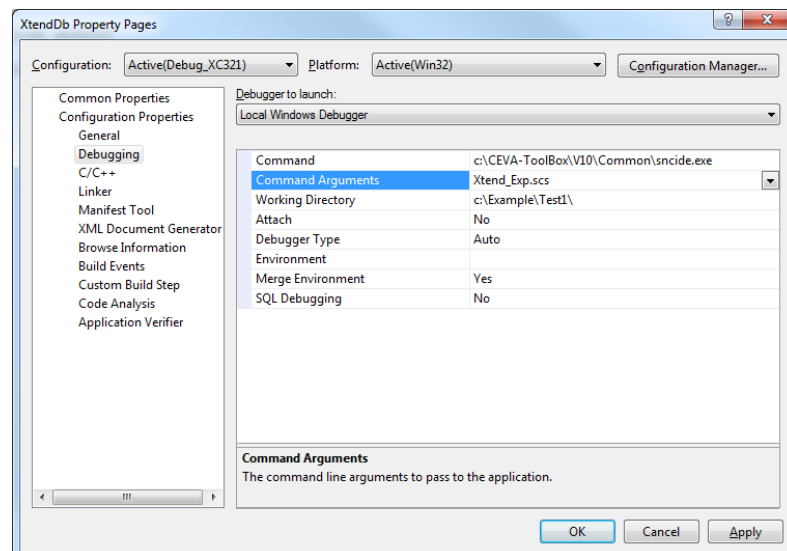
6 Debug CEVA-Xtend Project

In order to debug the new instructions that we created, we will need to load the Visual Studio project that was created and compile it in debug. This can be done easily by following the next steps:

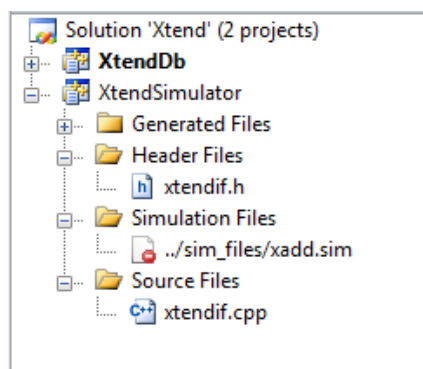
1. Open the solution of the project that is located in you project directory under “CEVA-Xtend\Project”.
2. Select the configuration that you wish to run. For debug, you should select “Debug_<CORE>”.



3. Build the solution.
4. Copy the files created in the build to the tools directory.
 - a. XtendSimulator.dll
 - b. cevaxdb_<core>_xtend.dll
5. In the Visual Studio, open the properties of the XtendSimulator project. Under “Configuration Properties->Debugging” set the project to open the CEVA-ToolBox through the Visual Studio as seen in the following image.



6. Select the sim file you wish to debug, by opening the XtendSimulator project and double clicking the required sim file under Simulation Files.



7. Place a break point in the sim file and start debug.

7 CEVA-Xtend Core Specific

7.1 Limitations CEVA-XC

- CEVA-Xtend unit comes instead of a regular unit in the instruction packet:
 - $XH0 \leftrightarrow MS0$
 - $XH1 \leftrightarrow MS1$
 - $VXH \leftrightarrow VB$
- For each VU configuration 16/32/64 there is a VU CEVA-Xtend unit is duplicated appropriately
- All CEVA-Xtend units can be used together in a single instruction packet, together with other CEVA-XC instructions
- Pipe stages and predicates are given only for sync instructions
- An immediate extension control word can be added to an CEVA-Xtend instruction
- Naming Conventions:
 - Variable and Switches names must comply with the following regular expression definition
 $[_a-zA-Z] [_a-zA-Z0-9]^+$
 - CEVA-Xtend instructions can use core resources by the following rule:
 - Instructions implemented in XH0 and XH1 units
 - Source 0, source 1, and destination
 - acX
 - $a0-a7$
 - Predicate
 - prX
 - Instructions implemented in VXH unit
 - Source 0
 - viX_{vxi}
 - $via, vib, vic, vid, vie, vif, vig, vih$
 - Source 1
 - viY_{vx_up}
 - vie, vif, vig, vih
 - Destination
 - $voZ0_{vx}$
 - $voa0, vob0$
 - $viZ0_{vx}$
 - $via0, vib0, vic0, vid0, vie0, vif0, vig0, vih0$

- Vector Predicate – 16-bit predication over writing to destination – per word or double word
 - *vprX_vx*
 - *vpr0_w* , *vpr1_w* , *vpr2_w*
 - *vpr0_dw* , *vpr1_dw* , *vpr2_dw*
 - *True*

CEVA-Xtend Units:

Units must be provided by the user, in the following form:

- XH0
- XH1
- VXH

Syntax Limitations

- Switches and variables must comply with the legal variable name syntax
- A signed or unsigned immediate can be used as a variable

Encoding

- User instruction encoding is encoded first in the user-defined area
- Each variable and switch needs a specific definition of the beginning of their encoding bit, relatively to the last Hoffman bit, when counting only user-defined area bits

Immediate extension usage

- The user can use 10- or 26-bit immediate extension as part of his user-defined encoding area

7.2 CEVA-TL4xx Templates naming conventions

7.2.1 Architecture specification and CEVA-Xtend Designer

The naming of the Architecture specification Xtend Templates differ from the CEVA-Xtend Designer templates. See below [CEVA-TL4xx Xtend templates](#) for detailed template naming

7.2.2 Core register groups

The naming of the core register groups in the Architecture reference guide differs from the CEVA-Xtend Designer core registers naming, the following table lists the CEVA-Xtend Designer core register groups and the registers which are part of the group, divided to two options, for CEVA-TL4x0 and CEVA-TL4x1 :

CEVA-TL4x0

Core register group	Register list
GRF_no_sp_y_lc_sv	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h
GRF	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;sp;y_r;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h;lc;y0;sv
GRF_no_ext	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;sp;y_r;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h;lc;y0;sv
GRF_no_sp	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;y_r;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h;lc;y0;sv
rN	r0;r1;r2;r3;r4;r5;r6;r7
rB	r6;r7
accr	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;a0l;a1l;b0l;b1l;a0h;a1h;b0h;b1h
ar	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7

CEVA-TL4x1

Core register group	Register list
GRF_no_sp_y_lc_sv	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h;c0;c1;c0l;c0h;c1l;c1h
GRF	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;sp;y_r;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h;lc;c0;c1;sv;c0l;c0h;c1l;c1h
GRF_no_ext	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;sp;y_r;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h;lc;c0;c1;sv;c0l;c0h;c1l;c1h
GRF_no_sp	a0;a1;b0;b1;r0;r1;r2;r3;r4;r5;r6;r7;y_r;p0;p1;a0l;a0h;b0l;b0h;a1l;a1h;b1l;b1h;lc;sv;c0;c1;c0l;c0h;c1l;c1h
rN	r0;r1;r2;r3;r4;r5;r6;r7
rB	r6;r7
accr	a0;a1;b0;b1;c0;c1;r0;r1;r2;r3;r4;r5;r6;r7;a0l;a1l;b0l;b1l;c0l;c1l;c0h;c1h;a0h;a1h;b0h;b1h
ar	a0;a1;b0;b1;c0;c1;r0;r1;r2;r3;r4;r5;r6;r7

7.3 Xtend Templates

7.3.1 CEVA-XC321

XH (0, 1)

instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	system								user_define				opm		DPS		fsv		ssv		prX		acZ_xh		acW_xh		acX_xh					
xh_1 acX_xh, acW_xh, acZ_xh [.?prX]	system								x				0		0		0		1		1		prX		acZ_xh		acW_xh		acX_xh			
xh_2 acX_xh, acW_xh, acZ_xh [.?prX]	system								x				0		0		1		1		1		prX		acZ_xh		acW_xh		acX_xh			
xh_3 acX_xh, acZ_xh [.?prX]	system								x				0		0		0		1		0		prX		acZ_xh		x		acX_xh			
xh_4 acX_xh, acZ_xh [.?prX]	system								x				0		0		1		1		0		prX		acZ_xh		x		acX_xh			
xh_5 acZ_xh [.?prX]	system								x				0		0		0		0		0		prX		acZ_xh		x		x			
xh_6 acZ_xh [.?prX]	system								x				0		0		1		0		0		prX		acZ_xh		x		x			
xh_7 acX_xh, acW_xh	system								x				1		x		1		1		x		x		acW_xh		acX_xh					
xh_8 acX_xh	system								x				1		x		1		0		x		x		x		acX_xh					
xh_9	system								x				1		x		0		0		x		x		x		x					

VXH

instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	system								define				opm				fsv	ssv	vprX_vx		voz0_vx		viy_up_vx		vix_vx							
vxx_1 vix_vx, viy_up_vx, viz0_vx [.?vprX_vx]	system								x				0		1	0	1	1	vprX_vx		viz0_vx		viy_up_vx		vix_vx							
vxx_2 vix_vx, viy_up_vx, voz0_vx [.?vprX_vx]	system								x				0		1	1	1	1	vprX_vx		voz0_vx		viy_up_vx		vix_vx							
vxx_3 vix_vx, viz0_vx [.?vprX_vx]	system								x				0		1	0	1	0	vprX_vx		viz0_vx		x		vix_vx							
vxx_4 vix_vx, voz0_vx [.?vprX_vx]	system								x				0		1	1	1	0	vprX_vx		voz0_vx		x		vix_vx							
vxx_5 viz0_vx [.?vprX_vx]	system								x				0		1	0	0	0	vprX_vx		viz0_vx		x		x							
vxx_6 voz0_vx [.?vprX_vx]	system								x				0		1	1	0	0	vprX_vx		voz0_vx		x		x							
vxx_7 vix_vx, viy_up_vx	system								x				1		x	1	1		x		x		viy_up_vx		vix_vx							
vxx_8 vix_vx	system								x				1		x	1	0		x		x		x		vix_vx							
vxx_9	system								x				1		x	0	0		x		x		x		x							

7.3.2 CEVA-XC323

XH (0, 1, 2, 3)

instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	system								user_define				opm	DPS		fsv	ssv	prX		acZ_xh	acW_xh	acX_xh										
xh_1 acX_xh, acW_xh, acZ_xh [.?prX]	system								x				0	0	0	1	1	prX		acZ_xh	acW_xh	acX_xh										
xh_2 acX_xh, acW_xh, acZ_xh [.?prX]	system								x				0	0	1	1	1	prX		acZ_xh	acW_xh	acX_xh										
xh_3 acX_xh, acZ_xh [.?prX]	system								x				0	0	0	1	0	prX		acZ_xh	x	acX_xh										
xh_4 acX_xh, acZ_xh [.?prX]	system								x				0	0	1	1	0	prX		acZ_xh	x	acX_xh										
xh_5 acZ_xh [.?prX]	system								x				0	0	0	0	0	prX		acZ_xh	x	x										
xh_6 acZ_xh [.?prX]	system								x				0	0	1	0	0	prX		acZ_xh	x	x										
xh_7 acX_xh, acW_xh	system								x				1	x	1	1	1	x		x	acW_xh	acX_xh										
xh_8 acX_xh	system								x				1	x	1	0	0	x		x	x	acX_xh										
xh_9	system								x				1	x	0	0	0	x		x	x	x										
xh_10 acX_xh, acW_xh, acZ_xh [.?prX]	system								x				0	1	0	1	1	prX		acZ_xh	acW_xh	acX_xh										
xh_11 acX_xh, acZ_xh [.?prX]	system								x				0	1	0	1	0	prX		acZ_xh	x	acX_xh										
xh_12 acZ_xh [.?prX]	system								x				0	1	0	0	0	prX		acZ_xh	x	x										

VXH

instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	system								user_define				opm	DPS		fsv	ssv	vprX_vx		voz0_vx	viy_up_vx		vix_vx									
vxx_1 vix_vx, viy_up_vx, viz0_vx [.?vprX_vx]	system								0	x				0	1	0	1	1	vprX_vx		viz0_vx	viy_up_vx		vix_vx								
vxx_2 vix_vx, viy_up_vx, voz0_vx [.?vprX_vx]	system								0	x				0	1	1	1	1	vprX_vx		voz0_vx	viy_up_vx		vix_vx								
vxx_3 vix_vx, viz0_vx [.?vprX_vx]	system								0	x				0	1	0	1	0	vprX_vx		viz0_vx	x		vix_vx								
vxx_4 vix_vx, voz0_vx [.?vprX_vx]	system								0	x				0	1	1	1	0	vprX_vx		voz0_vx	x		vix_vx								
vxx_5 viz0_vx [.?vprX_vx]	system								0	x				0	1	0	0	0	vprX_vx		viz0_vx	x		x								
vxx_6 voz0_vx [.?vprX_vx]	system								0	x				0	1	1	0	0	vprX_vx		voz0_vx	x		x								
vxx_7 vix_vx, viy_up_vx	system								0	x				1	x		1	1	1	x		x		viy_up_vx		vix_vx						
vxx_8 vix_vx	system								0	x				1	x		1	0	0	x		x		x		vix_vx						
vxx_9	system								0	x				1	x		0	0	0	x		x		x		x						

7.3.3 CEVA-XC4210

VXH

instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	system								user_define				opm	DPS		fsv	ssv	vprX_vx		voz0_vx		viy_up_vx		vix_vx								
vXH_1 vix_vx, viy_vx, viz0_vx [,?vprX_vx]									x	x	x	x	0	1	0	1	1	vprX_vx		viz0_vx		viy_vx		vix_vx								
vXH_2 vix_vx, viy_vx, voz0_vx [,?vprX_vx]									x	x	x	x	0	1	1	1	1	vprX_vx		voz0_vx		viy_vx		vix_vx								
vXH_3 vix_vx, viz0_vx [,?vprX_vx]									x	x	x	x	0	1	0	1	0	vprX_vx		viz0_vx		x		x	x	x	x	vix_vx				
vXH_4 vix_vx, voz0_vx [,?vprX_vx]									x	x	x	x	0	1	1	1	0	vprX_vx		voz0_vx		x		x	x	x	x	vix_vx				
vXH_5 viz0_vx [,?vprX_vx]									x	x	x	x	0	1	0	0	0	vprX_vx		viz0_vx		x		x	x	x	x	x	x	x	x	
vXH_6 voz0_vx [,?vprX_vx]									x	x	x	x	0	1	1	0	0	vprX_vx		voz0_vx		x		x	x	x	x	x	x	x	x	x
vXH_7 vix_vx, viy_vx									x	x	x	x	1	x	x	1	1	x	x	x	x	x	x	x	viy_vx		vix_vx					
vXH_8 vix_vx									x	x	x	x	1	x	x	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
vXH_9									x	x	x	x	1	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

7.3.4 CEVA-TL4xx Xtend templates

7.3.4.1 CEVA-TL4x0 – Audio/Voice

Template Name	Template Description
TL4xx_0	xinst
TL4xx_1	xinst ar
TL4xx_2	xinst GRF_no_ext
TL4xx_3	xinst GRF, ar
TL4xx_4	xinst GRF_no_ext, GRF_no_ext
TL4xx_5	xinst GRF_no_ext, GRF_no_ext, ar
TL4xx_6	xinst GRF_no_sp, #imm16
TL4xx_7	xinst GRF_no_sp, #imm16, ar
TL4xx_8	xinst #imm16
TL4xx_9	xinst #imm16, GRF_no_sp
TL4xx_10	xinst #imm16, ar
TL4xx_11	xinst #imm16, GRF_no_sp, ar
TL4xx_12	xinst GRF_no_ext, (rN)[.dw qw][+pm]
TL4xx_13	xinst GRF_no_ext, (rN)[.dw qw][+pm], ar
TL4xx_14	xinst GRF_no_ext, (rB+#imm16)[.dw qw]
TL4xx_15	xinst GRF_no_ext, (rB+#imm16)[.dw qw], ar
TL4xx_16	xinst (rB+#imm16)[.dw qw]
TL4xx_17	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv
TL4xx_18	xinst (rB+#imm16)[.dw qw], ar
TL4xx_19	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv, ar
TL4xx_20	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm]
TL4xx_21	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm], ar
TL4xx_22	xinst (rN)[.dw qw][+pm]
TL4xx_23	xinst (rN)[.dw qw][+pm], ar
TL4xx_24	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv
TL4xx_25	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv, ar
TL4xx_26	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm]
TL4xx_27	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm], ar
TL4xx_28	xinst (rN)[.dw qw][+pm], (rB+#imm16)[.dw qw]
TL4xx_45	xst (rB+#imm16)[.dw qw]
TL4xx_46	xst (rN)[.dw qw][+pm]
TL4xx_49	xinst GRF_no_ext, GRF_no_ext xst (rN)[.dw][+pm]
TL4xx_50	xinst GRF_no_ext, GRF_no_ext, ar xst (rN)[.dw][+pm]
TL4xx_51	xinst GRF_no_sp, #imm16 xst (rN)[.dw][+pm]
TL4xx_52	xinst GRF_no_sp, #imm16, ar xst (rN)[.dw][+pm]
TL4xx_53	xinst #imm16, GRF_no_sp xst (rN)[.dw][+pm]
TL4xx_54	xinst #imm16, GRF_no_sp, ar xst (rN)[.dw][+pm]

TL4xx_55	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rN)[.dw][+pm]
TL4xx_56	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rN)[.dw][+pm]
TL4xx_57	xinst GRF_no_ext, (rB+#imm16)[.dw] xst (rN)[.dw][+pm]
TL4xx_58	xinst GRF_no_ext, (rB+#imm16)[.dw], ar xst (rN)[.dw][+pm]
TL4xx_59	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_60	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_61	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_62	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_63	xinst GRF_no_ext, GRF_no_ext xst (rB+#imm16)[.dw]
TL4xx_64	xinst GRF_no_ext, GRF_no_ext, ar xst (rB+#imm16)[.dw]
TL4xx_65	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rB+#imm16)[.dw]
TL4xx_66	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rB+#imm16)[.dw]
TL4xx_67	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv xst (rB+#imm16)[.dw]
TL4xx_68	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rB+#imm16)[.dw]
TL4xx_74	xpop2dw
TL4xx_76	xpushdw
TL4xx_77	xpopdw

7.3.4.2 CEVA-TL4x0 – Compatibility

Template Name	Template Description
TL4xx_0	xinst
TL4xx_1	xinst ar
TL4xx_2	xinst GRF_no_ext
TL4xx_3	xinst GRF, ar
TL4xx_4	xinst GRF_no_ext, GRF_no_ext
TL4xx_5	xinst GRF_no_ext, GRF_no_ext, ar
TL4xx_6	xinst GRF_no_sp, #imm16
TL4xx_7	xinst GRF_no_sp, #imm16, ar
TL4xx_8	xinst #imm16
TL4xx_9	xinst #imm16, GRF_no_sp
TL4xx_10	xinst #imm16, ar
TL4xx_11	xinst #imm16, GRF_no_sp, ar
TL4xx_12	xinst GRF_no_ext, (rN)[.dw qw][+pm]
TL4xx_13	xinst GRF_no_ext, (rN)[.dw qw][+pm], ar
TL4xx_14	xinst GRF_no_ext, (rB+#imm16)[.dw qw]
TL4xx_15	xinst GRF_no_ext, (rB+#imm16)[.dw qw], ar
TL4xx_16	xinst (rB+#imm16)[.dw qw]
TL4xx_17	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv
TL4xx_18	xinst (rB+#imm16)[.dw qw], ar
TL4xx_19	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv, ar
TL4xx_20	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm]

Template Name	Template Description
TL4xx_21	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm], ar
TL4xx_22	xinst (rN)[.dw qw][+pm]
TL4xx_23	xinst (rN)[.dw qw][+pm], ar
TL4xx_24	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv
TL4xx_25	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv, ar
TL4xx_26	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm]
TL4xx_27	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm], ar
TL4xx_28	xinst (rN)[.dw qw][+pm], (rB+#imm16)[.dw qw]
TL4xx_45	xst (rB+#imm16)[.dw qw]
TL4xx_46	xst (rN)[.dw qw][+pm]
TL4xx_47	xst (rB+#imm16)[.dw] xst (rN)[.dw][+pm]
TL4xx_48	xst (rN)[.dw][+pm] xst (rN)[.dw][+pm]
TL4xx_49	xinst GRF_no_ext, GRF_no_ext xst (rN)[.dw][+pm]
TL4xx_50	xinst GRF_no_ext, GRF_no_ext, ar xst (rN)[.dw][+pm]
TL4xx_51	xinst GRF_no_sp, #imm16 xst (rN)[.dw][+pm]
TL4xx_52	xinst GRF_no_sp, #imm16, ar xst (rN)[.dw][+pm]
TL4xx_53	xinst #imm16, GRF_no_sp xst (rN)[.dw][+pm]
TL4xx_54	xinst #imm16, GRF_no_sp, ar xst (rN)[.dw][+pm]
TL4xx_55	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rN)[.dw][+pm]
TL4xx_56	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rN)[.dw][+pm]
TL4xx_57	xinst GRF_no_ext, (rB+#imm16)[.dw] xst (rN)[.dw][+pm]
TL4xx_58	xinst GRF_no_ext, (rB+#imm16)[.dw], ar xst (rN)[.dw][+pm]
TL4xx_59	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_60	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_61	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_62	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_63	xinst GRF_no_ext, GRF_no_ext xst (rB+#imm16)[.dw]
TL4xx_64	xinst GRF_no_ext, GRF_no_ext, ar xst (rB+#imm16)[.dw]
TL4xx_65	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rB+#imm16)[.dw]
TL4xx_66	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rB+#imm16)[.dw]
TL4xx_67	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv xst (rB+#imm16)[.dw]
TL4xx_68	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rB+#imm16)[.dw]
TL4xx_73	xpush2dw
TL4xx_74	xpop2dw
TL4xx_76	xpushdw
TL4xx_77	xpopdw

7.3.4.3 CEVA-TL4x1 – Audio / Voice

Template Name	Template Description
TL4xx_0	xinst
TL4xx_1	xinst ar
TL4xx_2	xinst GRF_no_ext
TL4xx_3	xinst GRF, ar
TL4xx_4	xinst GRF_no_ext, GRF_no_ext
TL4xx_5	xinst GRF_no_ext, GRF_no_ext, ar
TL4xx_6	xinst GRF_no_sp, #imm16
TL4xx_7	xinst GRF_no_sp, #imm16, ar
TL4xx_8	xinst #imm16
TL4xx_9	xinst #imm16, GRF_no_sp
TL4xx_10	xinst #imm16, ar
TL4xx_11	xinst #imm16, GRF_no_sp, ar
TL4xx_12	xinst GRF_no_ext, (rN)[.dw qw][+pm]
TL4xx_13	xinst GRF_no_ext, (rN)[.dw qw][+pm], ar
TL4xx_14	xinst GRF_no_ext, (rB+#imm16)[.dw qw]
TL4xx_15	xinst GRF_no_ext, (rB+#imm16)[.dw qw], ar
TL4xx_16	xinst (rB+#imm16)[.dw qw]
TL4xx_17	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv
TL4xx_18	xinst (rB+#imm16)[.dw qw], ar
TL4xx_19	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv, ar
TL4xx_20	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm]
TL4xx_21	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm], ar
TL4xx_22	xinst (rN)[.dw qw][+pm]
TL4xx_23	xinst (rN)[.dw qw][+pm], ar
TL4xx_24	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv
TL4xx_25	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv, ar
TL4xx_26	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm]
TL4xx_27	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm], ar
TL4xx_28	xinst (rN)[.dw qw][+pm], (rB+#imm16)[.dw qw]
TL4xx_29	xinst (rN)[.dw qw][+pm], (rB+#imm16)[.dw qw], ar
TL4xx_30	xinst (rN)[.dw][+pm], GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar, ar xinst (rN)[.dw][+pm], (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, GRF_no_sp_y_lc_sv, ar, ar
TL4xx_31	ar
TL4xx_32	xinst GRF_no_ext, GRF_no_ext, accr, accr, ar, ar
TL4xx_33	xinst (rN)[.dw][+pm], GRF_no_ext, GRF_no_sp_y_lc_sv, accr, ar, ar
TL4xx_34	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr, ar, ar
TL4xx_35	xinst (rN)[.dw][+pm], GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar
TL4xx_36	xinst (rN)[.dw][+pm], (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, GRF_no_sp_y_lc_sv, ar
TL4xx_37	xinst GRF_no_ext, GRF_no_ext, accr, accr, ar
TL4xx_38	xinst (rN)[.dw][+pm], GRF_no_ext, GRF_no_sp_y_lc_sv, accr, ar
TL4xx_39	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr, ar

Template Name	Template Description
TL4xx_40	xinst (rN)[.dw][+pm], GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv
TL4xx_41	xinst (rN)[.dw][+pm], (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, GRF_no_sp_y_lc_sv
TL4xx_42	xinst GRF_no_ext, GRF_no_ext, accr, accr
TL4xx_43	xinst (rN)[.dw][+pm], GRF_no_ext, GRF_no_sp_y_lc_sv, accr
TL4xx_44	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr
TL4xx_45	xst (rB+#imm16)[.dw qw]
TL4xx_46	xst (rN)[.dw qw][+pm]
TL4xx_49	xinst GRF_no_ext, GRF_no_ext xst (rN)[.dw][+pm]
TL4xx_50	xinst GRF_no_ext, GRF_no_ext, ar xst (rN)[.dw][+pm]
TL4xx_51	xinst GRF_no_sp, #imm16 xst (rN)[.dw][+pm]
TL4xx_52	xinst GRF_no_sp, #imm16, ar xst (rN)[.dw][+pm]
TL4xx_53	xinst #imm16, GRF_no_sp xst (rN)[.dw][+pm]
TL4xx_54	xinst #imm16, GRF_no_sp, ar xst (rN)[.dw][+pm]
TL4xx_55	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rN)[.dw][+pm]
TL4xx_56	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rN)[.dw][+pm]
TL4xx_57	xinst GRF_no_ext, (rB+#imm16)[.dw] xst (rN)[.dw][+pm]
TL4xx_58	xinst GRF_no_ext, (rB+#imm16)[.dw], ar xst (rN)[.dw][+pm]
TL4xx_59	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_60	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_61	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_62	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_63	xinst GRF_no_ext, GRF_no_ext xst (rB+#imm16)[.dw]
TL4xx_64	xinst GRF_no_ext, GRF_no_ext, ar xst (rB+#imm16)[.dw]
TL4xx_65	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rB+#imm16)[.dw]
TL4xx_66	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rB+#imm16)[.dw]
TL4xx_67	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv xst (rB+#imm16)[.dw]
TL4xx_68	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rB+#imm16)[.dw]
TL4xx_69	xinst GRF_no_ext, GRF_no_ext, accr, accr xst (rN)[.dw qw][+pm]
TL4xx_70	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr xst (rN)[.dw qw][+pm]
TL4xx_71	xinst GRF_no_ext, GRF_no_ext, accr, accr, ar xst (rN)[.dw qw][+pm]
TL4xx_75	xpop2dw
TL4xx_76	xpushdw
TL4xx_77	xpopdw
TL4xx_78	xpop4dw

7.3.4.4 CEVA-TL4x1 – Compatibility

Template Name	Template Description
TL4xx_0	xinst
TL4xx_1	xinst ar
TL4xx_2	xinst GRF_no_ext
TL4xx_3	xinst GRF, ar
TL4xx_4	xinst GRF_no_ext, GRF_no_ext
TL4xx_5	xinst GRF_no_ext, GRF_no_ext, ar
TL4xx_6	xinst GRF_no_sp, #imm16
TL4xx_7	xinst GRF_no_sp, #imm16, ar
TL4xx_8	xinst #imm16
TL4xx_9	xinst #imm16, GRF_no_sp
TL4xx_10	xinst #imm16, ar
TL4xx_11	xinst #imm16, GRF_no_sp, ar
TL4xx_12	xinst GRF_no_ext, (rN)[.dw qw][+pm]
TL4xx_13	xinst GRF_no_ext, (rN)[.dw qw][+pm], ar
TL4xx_14	xinst GRF_no_ext, (rB+#imm16)[.dw qw]
TL4xx_15	xinst GRF_no_ext, (rB+#imm16)[.dw qw], ar
TL4xx_16	xinst (rB+#imm16)[.dw qw]
TL4xx_17	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv
TL4xx_18	xinst (rB+#imm16)[.dw qw], ar
TL4xx_19	xinst (rB+#imm16)[.dw qw], GRF_no_sp_y_lc_sv, ar
TL4xx_20	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm]
TL4xx_21	xinst (rB+#imm16)[.dw qw], (rN)[.dw qw][+pm], ar
TL4xx_22	xinst (rN)[.dw qw][+pm]
TL4xx_23	xinst (rN)[.dw qw][+pm], ar
TL4xx_24	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv
TL4xx_25	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv, ar
TL4xx_26	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm]
TL4xx_27	xinst (rN)[.dw qw][+pm], (rN)[.dw qw][+pm], ar
TL4xx_28	xinst (rN)[.dw qw][+pm], (rB+#imm16)[.dw qw]
TL4xx_29	xinst (rN)[.dw qw][+pm], (rB+#imm16)[.dw qw], ar
TL4xx_30	xinst (rN)[.dw][+pm], GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar, ar xinst (rN)[.dw][+pm], (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, GRF_no_sp_y_lc_sv, ar, ar
TL4xx_31	ar
TL4xx_32	xinst GRF_no_ext, GRF_no_ext, accr, accr, ar, ar
TL4xx_33	xinst (rN)[.dw][+pm], GRF_no_ext, GRF_no_sp_y_lc_sv, accr, ar, ar
TL4xx_34	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr, ar, ar
TL4xx_35	xinst (rN)[.dw][+pm], GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar
TL4xx_36	xinst (rN)[.dw][+pm], (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, GRF_no_sp_y_lc_sv, ar
TL4xx_37	xinst GRF_no_ext, GRF_no_ext, accr, accr, ar
TL4xx_38	xinst (rN)[.dw][+pm], GRF_no_ext, GRF_no_sp_y_lc_sv, accr, ar
TL4xx_39	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr, ar

Template Name	Template Description
TL4xx_40	xinst (rN)[.dw][+pm], GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv
TL4xx_41	xinst (rN)[.dw][+pm], (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, GRF_no_sp_y_lc_sv
TL4xx_42	xinst GRF_no_ext, GRF_no_ext, accr, accr
TL4xx_43	xinst (rN)[.dw][+pm], GRF_no_ext, GRF_no_sp_y_lc_sv, accr
TL4xx_44	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr
TL4xx_45	xst (rB+#imm16)[.dw qw]
TL4xx_46	xst (rN)[.dw qw][+pm]
TL4xx_47	xst (rB+#imm16)[.dw] xst (rN)[.dw][+pm]
TL4xx_48	xst (rN)[.dw][+pm] xst (rN)[.dw][+pm]
TL4xx_49	xinst GRF_no_ext, GRF_no_ext xst (rN)[.dw][+pm]
TL4xx_50	xinst GRF_no_ext, GRF_no_ext, ar xst (rN)[.dw][+pm]
TL4xx_51	xinst GRF_no_sp, #imm16 xst (rN)[.dw][+pm]
TL4xx_52	xinst GRF_no_sp, #imm16, ar xst (rN)[.dw][+pm]
TL4xx_53	xinst #imm16, GRF_no_sp xst (rN)[.dw][+pm]
TL4xx_54	xinst #imm16, GRF_no_sp, ar xst (rN)[.dw][+pm]
TL4xx_55	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rN)[.dw][+pm]
TL4xx_56	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rN)[.dw][+pm]
TL4xx_57	xinst GRF_no_ext, (rB+#imm16)[.dw] xst (rN)[.dw][+pm]
TL4xx_58	xinst GRF_no_ext, (rB+#imm16)[.dw], ar xst (rN)[.dw][+pm]
TL4xx_59	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_60	xinst (rB+#imm16)[.dw], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_61	xinst (rN)[.dw qw][+pm], GRF_no_sp_y_lc_sv xst (rN)[.dw][+pm]
TL4xx_62	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rN)[.dw][+pm]
TL4xx_63	xinst GRF_no_ext, GRF_no_ext xst (rB+#imm16)[.dw]
TL4xx_64	xinst GRF_no_ext, GRF_no_ext, ar xst (rB+#imm16)[.dw]
TL4xx_65	xinst GRF_no_ext, (rN)[.dw][+pm] xst (rB+#imm16)[.dw]
TL4xx_66	xinst GRF_no_ext, (rN)[.dw][+pm], ar xst (rB+#imm16)[.dw]
TL4xx_67	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv xst (rB+#imm16)[.dw]
TL4xx_68	xinst (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, ar xst (rB+#imm16)[.dw]
TL4xx_69	xinst GRF_no_ext, GRF_no_ext, accr, accr xst (rN)[.dw qw][+pm]
TL4xx_70	xinst GRF_no_ext, (rN)[.dw][+pm], GRF_no_sp_y_lc_sv, accr xst (rN)[.dw qw][+pm]
TL4xx_71	xinst GRF_no_ext, GRF_no_ext, accr, accr, ar xst (rN)[.dw qw][+pm]
TL4xx_72	xpush2dw
TL4xx_75	xpop2dw
TL4xx_76	xpushdw
TL4xx_77	xpopdw
TL4xx_78	xpop4dw

7.3.5 CEVA-XM4

XH Unit

instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
xh_0 rA_xtd.dd, rB_xtd.dd, rZ_xtd.dd [, &prX_xtd]	system									x	x	0	1	1	prX_xtd			rZ_xtd.dd			rB_xtd.dd			rA_xtd.dd									
xh_1 rA_xtd.dd, rZ_xtd.dd [, &prX_xtd]	system									x	x	0	1	0	prX_xtd			rZ_xtd.dd			x	x	x	x	x	rA_xtd.dd							
xh_2 rZ_xtd.dd [, &prX_xtd]	system									x	x	0	0	0	prX_xtd			rZ_xtd.dd			x	x	x	x	x	x	x	x	x	x	x		
xh_3 rA_xtd.dd, rB_xtd.dd	system									x	x	1	1	1	x	x	x	x	x	x	x	x	rB_xtd.dd			rA_xtd.dd							
xh_4 rA_xtd.dd	system									x	x	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	rA_xtd.dd					
xh_5	system									x	x	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

VXH Unit

instruction	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vxh_0 vA_vxtd.dd, vB_vxtd.dd, vZ_vxtd.dd [, ?vprX_vxtd]	system									x	x	0	1	1	vprX_vxtd			vZ_vxtd.dd			vB_vxtd.dd			vA_vxtd.dd								
vxh_1 vA_vxtd.dd, vZ_vxtd.dd [, ?vprX_vxtd]	system									x	x	0	1	0	vprX_vxtd			vZ_vxtd.dd			x	x	x	x	vA_vxtd.dd							
vxh_2 vZ_vxtd.dd [, ?vprX_vxtd]	system									x	x	0	0	0	vprX_vxtd			vZ_vxtd.dd			x	x	x	x	x	x	x	x	x	x	x	
vxh_3 vA_vxtd.dd, vB_vxtd.dd	system									x	x	1	1	1	x	x	x	x	x	x	x	x	vB_vxtd.dd			vA_vxtd.dd						
vxh_4 vA_vxtd.dd	system									x	x	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	vA_vxtd.dd					
vxh_5	system									x	x	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x