CEVA-XM4™
CEVA-Xtend
Architecture
Specification

Rev 1.1.3.F

June 2016

Documentation Control

History Table

Version	Date	Description	Remarks
1.0.0.F	January 2015	Beta Release	
1.1.0.F	August 2015	Official Release	
1.1.1.0	August 2015	Beta Release	
1.1.1.1	August 2015	Beta Release	No changes
1.1.1.2	October 2015	Beta Release	No changes
1.1.1.3	November 2015	Beta Release	No changes
1.1.1.6	December 2015	Beta Release	No changes
1.1.1.A	January 2016	Official Release	
1.1.1.F	February 2016	Official Release	
1.1.2.F	March 2016	Official Release	
1.1.3.F	June 2016	Official Release	

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-XC5™, CEVA-XC321™, CEVA-XC323™, CEVA-XC8™, CEVA-Xtend™, CEVA-XC4000™, CEVA-XC4100™, CEVA-XC4200™, CEVA-XC4210™, CEVA-XC4400™, CEVA-XC4410™, CEVA-XC4500™, CEVA-XC4600™, 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.

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 might be beyond the scope of the documentation, an online Technical Support Service 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 website 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 the CEVA central support email address ceva-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 to help you resolve any problem.
- **License Keys**: Refer any license key requests or problems to sdtkeys@ceva-dsp.com. For SDT license keys installation information, see the *SDT Installation and Licensing Scheme Guide*.

Email: ceva-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	1174 Castro Street Suite 210 Mountain View, CA 94040 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**
Unit 1203, Building E Chamtime Plaza Office Lane 2889, Jinke Road Pudong New District Shanghai, 201203 China **Tel**: +86-21-20577000 **Fax**: +86-21-20577111	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	Rm 709, Tower A SCC Financial Centre No. 88 First Haide Avenue Nanshan District Shenzhen 518064 China **Tel**: +86-755-8435 6038 **Fax**: +86-755-8435 6077	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 South 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	1-6-5 Shibuya SK Aoyama Bldg. 3F Shibuya-ku, Tokyo 150-0002 Japan **Tel**: +81 3 5774 8250	RivieraWaves S.A.S 400, avenue Roumanille Les Bureaux Green Side 5, Bât 6 06410 Biot - Sophia Antipo-lis France **Tel**: +33 4 83 76 06 00 **Fax**: +33 4 83 76 06 01

Table of Contents

List of Figures

List of Tables

1 Introduction

The CEVA-XM4™ DSP cores family provides the end-user with the ability to customize the system according to a specific application(s). CEVA-Xtend™ units are additional user-defined instructions intended to expand the original core's instruction set for specific applications.

CEVA-Xtend units can be integrated along with a Scalar Processing Unit (SPU) and Vector Processing Unit (VPU). The end-user can create new instructions that activate the CEVA-Xtend hardware units. The syntax and the encoding of such instructions are customized and defined according to application needs and architecture guidelines.

1.1 Integration with the Scalar Processing Unit

The SPU can be integrated with the CEVA-Xtend unit. The CEVA-Xtend (XH) unit has two source registers and one destination register. The sources are read from the General Register File (GRF) and the results are written to the GRF, making them available to any other unit within the core.

The interface enables the user to activate the CEVA-Xtend Scalar unit simultaneously and in parallel with other units in the core. When a CEVA-Xtend instruction is issued, it is done on the expense of SPU0.

1.2 Integration with the Vector Processing Unit

The VPU can be integrated with the CEVA-Xtend unit. CEVA-Xtend (VXH) unit has two source input vectors and one destination vector. The destination is output directly to the Vector Register File (VRF).

The interface enables the user to activate the CEVA-Xtend simultaneously and in parallel with other core units. CEVA-Xtend instruction is issued on the expense of VPU0.

2 Operation Modes

The CEVA-Xtend can operate in two modes: Synchronous mode and Trigger (Asynchronous) mode.

2.1 Synchronous Mode

In Synchronous mode, the instruction executions are coupled with the core pipeline stages, and the results can be sampled at each of the pipeline stages. The XH is coupled to SPU0 and the results are sampled at the V1 pipeline stage. The VXH is coupled to VPU0 and the result is sampled at the V5 pipeline stage. The sampling stage is encoded within the instruction opcode. This means that the result must be ready and valid on the output bus for proper functionality. For more information regarding the output delay timing of the result, refer to the CEVA-Xtend Interface Timing section in the CEVAXM4 Integration Reference Guide.

The sampling stage is determined automatically by the assembler tool from the instruction opcode, which specifies the destination too. The destination serves as a general register when coupled to XH or a vector when coupled to VXH. At the specified pipeline stage, the core samples the input bus from the CEVA-Xtend and writes the contents to the specified register or vector.

In Synchronous mode, the tools can embed the instruction behavior and information within the cycleaccurate simulator and debugger. For Instruction template, refer to the CEVA-Xtend chapter in the CEVA-XM4 SmartNcode Assembler and Linker Users Guide.

Example 2-1. CEVA-Xtend Synchronous Mode

VXH.vflip vA.[u]c32, vB.[u]c32, vZ.c32

The dispatch mechanism identifies the instruction as a VXH instruction (coupled to the VPU0) and the VXH unit decodes the instruction as a *vflip* instruction. Two sources are read from the vectors and the result of the calculation is written to the destination vector at the pipeline stage, which is specified in the VXH opcode.

Note that *vflip* is just example of possible syntax, the instruction syntax and operation is defined by the user.

2.2 Trigger (Asynchronous) Mode

In Trigger (Asynchronous) mode, the instructions are coupled with the core pipeline clock. However, the execution results are not sampled back to the core, but rather saved internally within the CEVA-Xtend registers or vectors. Trigger mode is intended for executions that require more stages than Synchronous mode's defined pipe stages, in order to complete the operations. While in Trigger mode, the execution of such instructions takes more cycles than the pipe stages and they are not sampled back to the core. Therefore, such information cannot be represented within the software IP tools. In Trigger mode, the user can use the user-define bits presented in the opcode to encode the Xtend's internal register or vector destinations.

Eventually, in order to move the results of the trigger operation from the CEVA-Xtend back to the core's internal register or vector, an additional independent Synchronous move-type instruction must be issued. The move instruction can be issued once CEVA-Xtend operation is completed. An indication for completing the operation can be handled in the following ways:

- Insertion of an interrupt by the CEVA-Xtend unit once the operation is completed. The interrupt length can be configured between one to 16 cycles, based on the INT_COUNT field in the *XCI_COR* register. For more information regarding the *INT_COUNT* field, refer to the *CEVA-XM4 Architecture Specification Memory Subsystem*.
- Polling by the core on the user input pins. The indication is raised by the CEVA-Xtend and can be cleared by one of the output user-indication pins. It is recommended to latch the input pin by an external device.
- If the exact number of cycles required to complete the operation is known, no polling is needed. A Synchronous move-type instruction can be issued when the operation is assumed to be completed.

Example 2-2. CEVA-Xtend Trigger Mode

1. Using an Interrupt:

```
VXH.vmpg4 vA.c32, vB.c32, vxZ.c32 ; assuming veZ is an VXH internal vector (user-defined encoding)
 …                                  ; as soon as the operation is done, CXV_INT is asserted for one cycle.
 …
 Int_VXH_ service:
VXH.mov vxA.c32, vZ.c32            ; the result is moved from vx to VRF using a Synchronous movtype
                                    ; instruction.
reti
```

2. Using Polling:

```
VXH.vmpg4 vA.c32, vB.c32, vxZ.c32 ; assuming veZ is an VXH internal vector (user-defined encoding)
 …                                  ; as soon as the operation is done, CXV_UI is asserted for one cycle
  ; latched inside the GPIN MSS register.
_poll_loop:
  in {cmp} #gpin_reg, r0.ui         ; read the GPIN MSS register.
  nop #2
  tsts {bit} r0.ui, #bit, pr0       ; update pr0 if GPIN MSS[#bit] is set.
  br _poll_loop , ?pr0              ; branch back to polling loop.
VXH.mov vxA.c32, vZ.c32            ; the result is moved from vx to VRF using a Synchronous movtype
                                    ; instruction.
```

3. Exact Operation Cycle Count is Known:

```
VXH.vmpg4 vA.c32, vB.c32, vxZ.c32 ; assuming veZ is an VXH internal vector (user-defined encoding)
 inst1                              ; operation is assumed to be done in five cycles.
 inst2
 inst3
 inst4
VXH.mov vxA.c32, vZ.c32            ; the result is moved from vx to VRF using a Synchronous movtype
                                    ; instruction.
```

3 CEVA-Xtend Interface

3.1 CEVA-Xtend (XH) Connectivity Interface with the SPU

Table 3-1 describes the CEVA-Xtend connectivity towards the SPU interface.

Table 3-1. CEVA-Xtend Interface Signals for SPU

Signal Name	I/O	Size	Description
CXS DEST	Output	32	Output result for GRF (at V1 pipe stage).
CXS UI	Output	1	Indication bit that can be connected to the user input interface .
CXS INT	Output	1	CEVA-Xtend interrupt.
CXV VPRZ	Output	32	CEVA-Xtend predicate, indicates to the core which byte in CXV_DSET need to be written to the destination.
CXV VPRZ	Output	32	CEVA-Xtend predicate, indicates to the core which byte in CXV_DSET need to be written to the destination.
CXS SRC0	Input	32	Input source0 from core register.
CXS SRC1	Input	32	Input source1 from core register.
CXS PRX	Input	1	SPU predicates.
CXS OP VALID	Input	1	Operation Valid bit indication for the CXS OP bus.
CXS OP	Input	24	Input from core instruction register: Contains the instruction opcode encoding for the XH unit.
CXS 10EXT VALID	Input	1	Indication from the core dispatcher that 10 bits of the extension are valid. The 10-bit extension CW can be used as an immediate value or be user-defined.
CXS 26EXT VALID	Input	1	Indication from the core dispatcher that 26 bits of the extension are valid. The 26-bit extension can be used as an immediate value or be user-defined.
CXS EXT	Input	26	Input from the core instruction register: If necessary, used as an extension opcode.
CXS CLK	Input	1	Core clock.
CXS WAIT	Input	1	Halt indication from the core.
CXS NO_WAIT_VU	Input	1	Stall Indication from the core .
CXS RESET	Input	1	Reset indication.

3.2 CEVA-Xtend (VXH) Connectivity Interface with the VPU

Table 3-2 describes the CEVA-Xtend (VXH) connectivity towards the VPU interface.

Table 3-2. CEVA-Xtend Connectivity Interface Signals for VPU

Signal Name	I/O	Size	Description
CXV DEST	Output	256	Output result for VRF (at V5 pipe stage).
CXV UI	Output	1	Indication bit that can be connected to the user input interface.
CXV INT	Output	1	CEVA-Xtend interrupt .
CXV SRC0	Input	256	Input source0 from VRF.
CXV SRC1	Input	256	Input source1 from VRF.
CXV VPRX	Input	32	VPU predicates
CXV OP VALID	Input	1	Operation Valid bit indication for the CXV OP bus.
CXV OP	Input	24	Input from the core instruction register: Contains the instruction opcode encoding for the Xtend hardware unit.
CXV 10EXT VALID	Input	1	Indication from the core dispatcher that 10 bits for the extension are valid. The 10-bit extension CW can be used as an immediate value or be user-defined.
CXV 26EXT VALID	Input	1	Indication from the core dispatcher that 26 bits for the extension are valid. The 26-bit extension can be used as an immediate value or be user-defined.
CXV EXT	Input	26	Input from the core instruction register: If necessary, used as a 26-bit extension opcode.
CXV CLK	Input	1	Core clock.
CXV WAIT	Input	1	Halt indication from the core.
CXV NO_WAIT_VU	Input	1	Stall indication from the core.
CXV RESET	Input	1	Reset indication.

4 CEVA-Xtend Instruction Encoding

Figure 4-1 describes the instruction encoding scheme for the CEVA-Xtend hardware unit. The Xtend unit is controlled by bits [23−0] on the CX OP input bus.

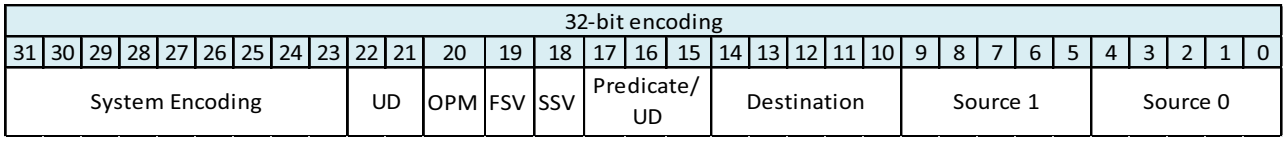

Figure 4-1. CEVA-Xtend Instruction Encoding

Table 4-1 describes the instruction (Opcode) fields and their description in order to help the user to build the decoder within the CEVA-Xtend. The description below defines the behavior for Xtend decoding.

Table 4-1. Xtend Instruction Fields in the Encoding

Field	Field Size	Location in Encoding	Description
System Encoding	8	[31:23]	Identifies the instruction as a CEVA Xtend instruction and identifies the cluster and the hosting unit.
User-define (UD)	3	[22:21]	User-defined.
Operation Mode (OPM)	1	[20]	[0]: Synchronous mode (default). [1]: Asynchronous mode.
First Source Valid (FSV)	1	[19]	[0]: First source is not valid. [1]: First source is valid (default).
Second Source Valid (SSV)	1	[18]	[0]: Second source is not valid. [1]: Second source is valid (default).
Predicate/User Define (UD)	3	[17:15]	In Synchronous mode: The field is used for predicate register encoding. The core uses the predicate for conditional execution to determine whether to write to a vector at the end of the execution. The MSB represents word or double-word vector predicates. In Asynchronous mode, it is user-defined.
Destination	5	[14:10]	In Synchronous mode: A destination vector. In Asynchronous mode: An internal CEVA-Xtend destination.
Source 1	5	[9:5]	Represents the second source vector.
Source 0	5	[4:0]	Represents the first source vector.

4.1 Remarks and Restrictions

- When using Synchronous mode and the source0 or source1 fields are not validated with FSV or SSV, respectively, the source0 or source1 fields can be used by the Xtend units for their own internal source registers/vectors set.
- When using Asynchronous mode, the destination field is used by the Xtend units for their own internal destination registers/vectors set. In contrast, when using Synchronous mode, the destination field values should be transferred back to the core via the CXS/V DEST output pins. Thus, the destination field cannot be used by the Xtend units internally.

4.2 Permitted Sequences

4.2.1 Instruction Types for CEVA-Xtend SPU (XH)

Table 4-2 describes the instruction type encoding that can be performed on the XH unit that is coupled to the SPU.

Table 4-2. Instruction Types for CEVA-Xtend SPU

Instruction Type	Operation Mode (OPM)	First Source Valid (FSV)	Second Source Valid (SSV)	Destination Valid
XH_0	0	1	1	1
XH_1	0	1	0	1
XH_2	0	0	0	1
XH_3	1	1	1	0
XH_4	1	1	0	0
XH_5	1	0	0	0

4.2.2 Instruction Types for CEVA-Xtend VPU (VXH)

Table 4-3 describes the instruction type encoding that can be performed on the VXH unit that is coupled to the VPU.

Table 4-3. Instruction Types for CEVA-Xtend for VPU

Instruction Type	Operation Mode (OPM)	First Source Valid (FSV)	Second Source Valid (SSV)	Destination Valid
VXH_0	0	1	1	1
VXH_1	0	1	0	1
VXH_2	0	0	0	1
VXH_3	1	1	1	0
VXH_4	1	1	0	0
VXH_5	1	0	0	0

5 CEVA-Xtend Coupled with Core Pipeline

The following figures show the core pipeline together with CEVA-Xtend interface behavior.

5.1 CEVA-Xtend with SPU Pipeline (XH)

Figure 5-1 describes a Synchronous mode XH operation. The instruction execution contains two stages. The result data should be sampled inside the XH unit at the end of the E2 stage. The XH issues the sampled result to the core at V1 and the core samples the result at the end of V2.

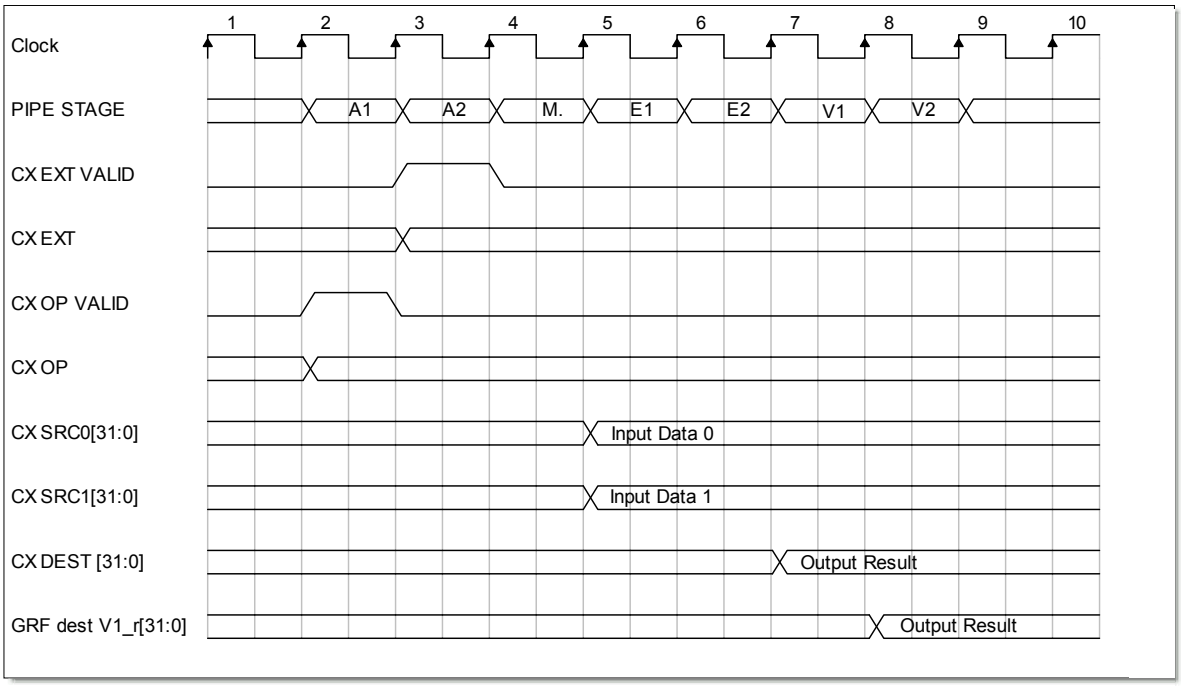

Figure 5-1. Pipeline Diagram of XH – Data Written to Core GRF at the End of V2

Legend:

- GRF dest V2_r: The internal CEVA-XM4 destination GRF that holds the CEVA-Xtend module output result.

5.2 CEVA-Xtend with VPU Pipeline (VXH)

Figure 5-2 describes a Synchronous mode VXH operation. The instruction execution contains four stages. The result data should be sampled inside the VXH unit at the end of V4 stage. The VXH issues the sampled result to the core at V5 and the core samples the result at the end of V5.

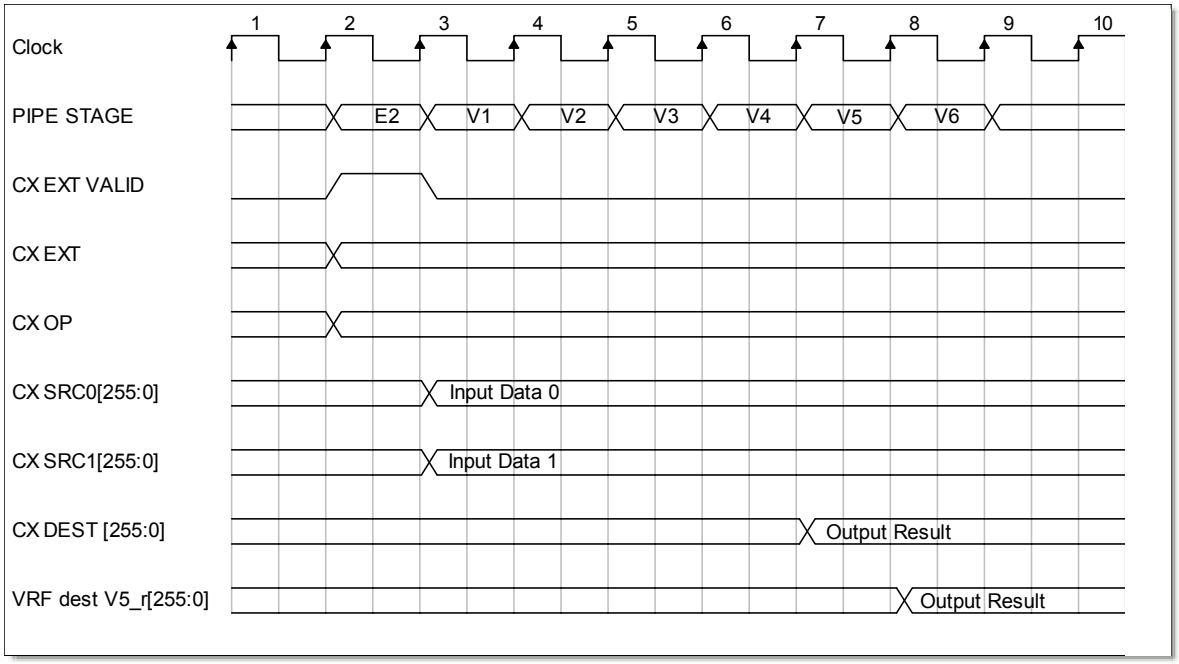

Figure 5-2. Pipeline Diagram of VXH − Data Written to VRF at the End of V5

Legend:

- VRF dest V5_r: The internal CEVA-XM4 destination VRF that holds the CEVAXtend module output result.

5.3 VXH with RAW Behavior

Figure 5-3 describes the case of a read-after-write (load-after-store) scenario in Synchronous mode using a VXH module that is connected to the VPU. In such a case, the core raises a wait signal
(CX wait_r) towards the VXH, but in order to complete the execution, the VXH receives a do not wait (CX no_wait_vu) signal for one cycle.

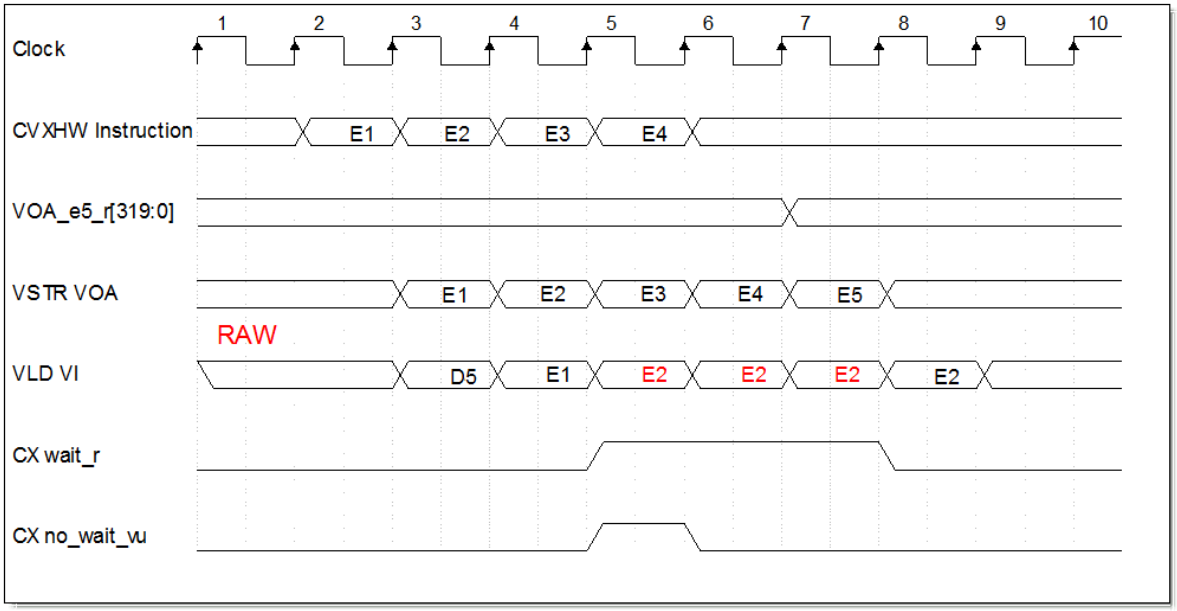

Figure 5-3. Pipeline Diagram of VXH − Data Written to VORF at the End of E5 with RAW

Legend:

- VOA_e5_r: The CEVA DSP VPU register that holds the CEVA-XC vector output to be written to the CEVA-Xtend module.
- VSTR VOA: The pipeline stage of the vstr instruction that causes the write operation.
- VLD VI: The pipeline stage of the vld instruction that causes the read operation.

6 Latency Tables

6.1 Write-After-Write (WAW) Restriction

A latency of one cycle is needed when a *movv* instruction is followed by an *xh* instruction.