HDVICP2.0 / IVA-HD Coprocessor API Library

Programmer's Reference

THIS PAGE IS INTENTIONALLY LEFT BLANK

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive & Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications & Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers & Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energyapps
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video & Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas
75265 Copyright© 2013, Texas Instruments Incorporated

THIS PAGE IS INTENTIONALLY LEFT BLANK

Contents

1. GETTING STARTED 11

1.1 INTRODUCTION 11
1.2 HDVICP2.0/IVA-HD API LIBRARY 11
1.3 GUIDE TO THIS MANUAL 12
1.4 TERMS AND ABBREVIATIONS 12
1.5 INSTALLATION 13
1.5.1 SYSTEM REQUIREMENTS 13
1.5.2 INSTALLING THE API LIBRARY 14
1.5.3 INSTALLING HDVCIP2.0/IVA-HD CSL/CSP 15
1.5.4 SETUP ENVIRONMENT VARIABLES 15
1.5.5 INSTALLING IVA-HD SIMULATOR 15
1.6 BUILD AND SAMPLE USAGE 16
1.6.1 BUILDING IVA-HD API LIBRARY 16
1.7 IVA-HD API BASIC DATA TYPES 16

2. HARDWARE RESOURCE FILES 19

2.1 COPROCESSOR BASE ADDRESSES (CSL) 19
2.2 IVA-HD REGISTER OVERLAYS AND MASKS 19
2.3 COMMON HARDWARE STRUCTURE DATA TYPES 19
2.3.1 CSL_CALC3_IPGW_REGS 19
2.3.2 CSL_CALC3_BFSW_REGS 20
2.3.3 CSL_CALC3_MMR_REGS 21
2.3.4 CSL_CALC3_MEM_REGS 21
2.3.5 CSL_ECD3_IPGW_REGS 22
2.3.6 CSL_ECD3_BFSW_REGS 23
2.3.7 CSL_ECD3_MEM_REGS 23
2.3.8 CSL_ECD3_JPG_REGS 24
2.3.9 CSL_ECD3_MP2_REGS 24
2.3.10 CSL_ECD3_MP4_REGS 25
2.3.11 CSL_ECD3_VC1_REGS 25
2.3.12 CSL_ECD3_H264_REGS 26
2.3.13 CSL_ECD3_AVS_REGS 26
2.3.14 CSL_ECD3_CDC_REGS 26
2.3.15 CSL_ECD3_MMR_REGS 27
2.3.16 CSL_ICECRUSHER1_CFG_IC968_REGS 28
2.3.17 CSL_ICONT_IMEM_REGS 29
2.3.18 CSL_ICONT1_MMR_DM_REGS 30
2.3.19 CSL_ICONT_DMEM_REGS 31
2.3.20 CSL_ICONT1_MMR_CFG_REGS 31
2.3.21 CSL_ICONT1_MMR_SBH_REGS 31
2.3.22 CSL_ICONT_IRQREG_REGS 33
2.3.23 CSL_ILF3_REGS 35
2.3.24 CSL_IME3_REGS 36
2.3.25 CSL_IPE3_IPGW_REGS 39
2.3.26 CSL_IPE3_BFSW_REGS 40
2.3.27 CSL_IPE3_MMR_REGS 41
2.3.28 CSL_IPE3_MEM_REGS 42
2.3.29 CSL_MC3_IPGW_REGS 42
2.3.30 CSL_MC3_BFSW_REGS 43
2.3.31 CSL_MC3_MMR_REGS 44
2.3.32 CSL_MC3_MEM_REGS 44
2.3.33 CSL_LSE_REGS 45
2.3.34 CSL_MAILBOX_REGS 45

2.3.35 CSL_SL2IF_CFG_REGS 46
2.3.36 CSL_CONF_REGS 47
2.3.37 CSL_ICONT1_SB_SYNCBOX_REGS 47
2.3.38 CSL_SYSCTRL_REGS 51
2.3.39 CSL_VDMA_REGS 52

3. IVA-HD API DATA TYPES 55

3.1 COMMON DATA TYPES 55
3.1.1 EHWANODE 55
3.1.2 ECPU_ID 55
3.1.3 THDVICP20MPEG4DFRAMEINFO 55
3.1.4 THDVICP20VC1DDECSTATE 57
3.1.5 THDVICP20MPEG2DFRAMEINFO 59
3.1.6 THDVICP20MC3MPEG2DDYNAMICSTRUCT 59
3.2 DATA TYPES FOR ECD3 60
3.2.1 THDVICP20ECD3H264DFRAMEINIT 60
3.2.2 THDVICP20ECD3H264EFRAMEINIT 62
3.2.3 THDVICP20ECD3MPEG4ENCINITSTRUCT 64
3.3 DATA TYPES FOR CALC3 66
3.3.1 THDVICP20CALC3H264DFRAMEINIT 66
3.3.2 THDVICP20CALC3H264EFRAMEINIT 68
3.3.3 THDVICP20CALC3MPEG4ENCINITSTRUCT 70
3.4 DATA TYPES FOR MC3 71
3.4.1 THDVICP20MC3H264DFRAMEINIT 71
3.4.2 MCREFDATA 71
3.4.3 THDVICP20MC3H264EFRAMEINIT 72
3.5 DATA TYPES FOR IME3 73
3.5.1 THDVICP20IME3MPEG4ENCINITSTRUCT 73
3.6 DATA TYPES FOR IPE3 74
3.7 DATA TYPES FOR ILF3 74
3.8 DATA TYPES FOR LSE (LOAD-STORE ENGINE) 74
3.8.1 ELSETASK 74
3.8.2 THDVICP20LSECOPY1DPARAMSTRUCT 74
3.8.3 THDVICP20LSECOPY2DPARAMSTRUCT 74
3.8.4 THDVICP20LSEUPDATECOPYADDRPARAMSTRUCT 75
3.9 DATA TYPES FOR SB (SYNCBOX) 76
3.9.1 THDVICP20SBINITHWATASKPARAMSTRUCT 76
3.9.2 THDVICP20SBINITHWAPARAMSTRUCT 77
3.9.3 THDVICP20SBINITCPUTASKPARAMSTRUCT 78
3.9.4 THDVICP20SBINITCPUPARAMSTRUCT 78
3.10 DATA TYPES FOR SBH (SYNCBOX HANDLER) 79
3.10.1 ESBHTASKTYPE 79
3.10.2 ESBHCOUNTERID 79
3.10.3 THDVICP20SBHTASKCTRLPARAMSTRUCT 79
3.11 DATA TYPES FOR VDMA 80
3.11.1 THDVICP20VDMAPARAMETERSSTRUCT 80
3.11.2 THDVICP20VDMAGROUPDEFINITIONSTRUCT 80
3.11.3 THDVICP20VDMAPADDINGPARAMSTRUCT 81
3.11.4 THDVICP20VDMA1DOBJAUTOINCPARAMSTRUCT 82
3.11.5 THDVICP20VDMAUVINTERLEAVINGPARAMSTRUCT 82
3.11.6 THDVICP20VDMADECIMATIONPARAMSTRUCT 83
3.11.7 THDVICP20VDMACOMPRESSIONPARAMSTRUCT 83
3.11.8 THDVICP20VDMASTATESTRUCT 83
3.12 DATA TYPES FOR DM (DATA MOVER) 84
3.13 DATA TYPES FOR MB (MAIL BOX) 84
3.13.1 EMAILBOXUSERS 84

3.13.2 eMailBoxNums 84
3.14 Data types for SMSET (Software Messages & System Event Trace) 84

4. CALC3 FUNCTIONS 85

4.1 List of functions 85
4.2 HDVICP20_TI_CALC3_H264D_FrameInit 86
4.3 HDVICP20_TI_CALC3_H264E_FrameInit 86
4.4 HDVICP20_TI_CALC3_MPEG4D_FrameInit 87
4.5 HDVICP20_TI_CALC3_MPEG4E_FrameInit 87
4.6 HDVICP20_TI_CALC3_VC1D_FrameInit 88
4.7 HDVICP20_TI_CALC3_MPEG2D_FrameInit 88
4.8 CALC3 API Usage 89

5. ECD3 FUNCTIONS 91

5.1 List of functions 91
5.2 HDVICP20_TI_ECD3_H264D_FrameInit 92
5.3 HDVICP20_TI_ECD3_H264E_FrameInit 92
5.4 HDVICP20_TI_ECD3_H264E_SliceLevelConfig 93
5.5 HDVICP20_TI_ECD3_MPEG4D_FrameInit 93
5.6 HDVICP20_TI_ECD3_MPEG4E_FrameInit 94
5.7 HDVICP20_TI_ECD3_VC1D_FrameInit 94
5.8 HDVICP20_TI_ECD3_MPEG2D_FrameInit 95
5.9 ECD3 API Usage 96

6. ILF3 FUNCTIONS 97

6.1 List of functions 97
6.2 HDVICP20_TI_ILF3_H264D_FrameInit 98
6.3 HDVICP20_TI_ILF3_H264E_FrameInit 98
6.4 HDVICP20_TI_ILF3_VC1D_FrameInit 98
6.5 iLF3 API Usage 99

7. IME3 FUNCTIONS 101

7.1 iME3 Basic Programming Model 101
7.2 List of functions 101
7.3 HDVICP20_TI_IME3_IvahdPicInit 102
7.4 HDVICP20_TI_IME3_Config 102

8. IPE3 FUNCTIONS 103

8.1 List of functions 103
8.2 HDVICP20_TI_IPE3_H264E_FrameInit 104

9. MC3 FUNCTIONS 105

9.1 List of functions 105
9.2 HDVICP20_TI_MC3_H264D_FrameInit 106
9.3 HDVICP20_TI_MC3_H264E_FrameInit 106
9.4 HDVICP20_TI_MC3_MPEG4D_FrameInit 107
9.5 HDVICP20_TI_MC3_MPEG4E_FrameInit 107
9.6 HDVICP20_TI_MC3_VC1D_FrameInit 108
9.7 HDVICP20_TI_MC3_MPEG2D_FrameInit 108

10. DATA MOVER FUNCTIONS 111

10.1 List of Macros 111
10.2 HDVICP20_TI_DM_SetContextParams 112
10.3 HDVICP20_TI_DM_SetSourceAddress 113
10.4 HDVICP20_TI_DM_SetDestinationAddress 113
10.5 HDVICP20_TI_DM_GetCurrentStatus 114

10.6 HDVICP20_TI_DM_WAIT 114
10.7 DM API USAGE EXAMPLE 115

11. LOAD-STORE ENGINE FUNCTIONS 117

11.1 LIST OF FUNCTIONS 117
11.2 HDVICP20_TI_LSE_INIT 118
11.3 HDVICP20_TI_LSE_SETPARAMADDR 118
11.4 HDVICP20_TI_LSE_SETLOADADDRESS 119
11.5 HDVICP20_TI_LSE_SETSTOREADDRESS 119
11.6 HDVICP20_TI_LSE_TRIGGER 120
11.7 HDVICP20_TI_LSE_PREPARECOMMANDCOPY1D 121
11.8 HDVICP20_TI_LSE_PREPARECOMMANDCOPY2D 121
11.9 HDVICP20_TI_LSE_PREPARECOMMANDUPDATECOPYADDRESS 122
11.10 HDVICP20_TI_LSE_PREPARECOMMANDREGWRITE 123
11.11 HDVICP20_TI_LSE_PREPARECOMMANDNOP 123
11.12 HDVICP20_TI_LSE_END 124
11.13 HDVICP20_TI_LSE_PREPAREMULTICOPY1D 124
11.14 HDVICP20_TI_LSE_PREPARECOPY2DWITHUPDATECOPYADDRESS 125
11.15 HDVICP20_TI_LSE_PREPARECOMMANDEND 126
11.16 HDVICP20_TI_LSE_PREPARECOMMANDUPDATEADPTVFLAGADDR 126
11.17 HDVICP20_TI_LSE_PREPARECOMMANDADPTVADD 127
11.18 HDVICP20_TI_LSE_ADPTVFLAGADDRCNTRUPDATE 127
11.19 HDVICP20_TI_LSE_COPYADDRCNTRUPDATE 128
11.20 LSE API EXAMPLE USAGE 129

12. SYNCBOX FUNCTIONS 131

12.1 LIST OF FUNCTIONS 132
12.2 HDVICP20_TI_SB_FILLCOMMONHWACFGPARAMS 132
12.3 HDVICP20_TI_SB_FILLASYNCHWACFGPARAMS 133
12.4 HDVICP20_TI_SB_FILLTASKHWACFGPARAM 134
12.5 HDVICP20_TI_SB_CONFIGUREHWAREGISTERS 135
12.6 HDVICP20_TI_SB_FILLCOMMONCPUCFGPARAMS 135
12.7 HDVICP20_TI_SB_FILLTASKCPUCFGPARAM 136
12.8 HDVICP20_TI_SB_CONFIGURECPUREGISTERS 137
12.9 HDVICP20_TI_SB_READSYNCBOXREGISTER 137
12.10 HDVICP20_TI_SB_WRITESYNCBOXREGISTER 138
12.11 HDVICP20_TI_SB_SYNCBOXSWRESET 138
12.12 HDVICP20_TI_SB_SYNCBOXCLEARSTATUS 139
12.13 SYNCBOX API EXAMPLE USAGE 139

13. SYNCBOX HANDLER FUNCTIONS 143

13.1 LIST OF FUNCTIONS & MACROS 144
13.2 HDVICP20_TI_SBH_CONFIGURECTRLPARAMS 145
13.3 HDVICP20_TI_SBH_READACTVTASKLISTREGISTER 145
13.4 HDVICP20_TI_SBH_CONFIGURETASKACK 146
13.5 HDVICP20_TI_SBH_CONFIGURETONOTIFYENDOFTASK 146
13.6 HDVICP20_TI_SBH_CONFIGURETASKCONTROLPARAMS 147
13.7 HDVICP20_TI_SBH_UPDATEDMLGLCHANNELENABLE 147
13.8 HDVICP20_TI_SBH_FILLTASKCONTROLPARAMS 148
13.9 HDVICP20_TI_SBH_MAPVDMAGROUPTODMLGLCHANNEL 149
13.10 HDVICP20_TI_SBH_VDMAGROUPSTARTINBYPASSMODE 150
13.11 HDVICP20_TI_SBH_READVDMAGROUPENDINSBBYPASS 150
13.12 HDVICP20_TI_SBH_ISTASKACTIVATED 151
13.13 HDVICP20_TI_SBH_SETIRQMASK 151
13.14 HDVICP20_TI_SBH_RESETIRQMASK 152
13.15 HDVICP20_TI_SBH_SAVEIRQMASK 152

13.16 HDVICP20_TI_SBH_READIRQSTATUS 153
13.17 HDVICP20_TI_SBH_SETENDOFTASK 153
13.18 HDVICP20_TI_SBH_SENDACKOFTASK 154
13.19 HDVICP20_TI_SBH_SETCOUNTER 154
13.20 SBH API USAGE EXAMPLE 155

14. VDMA FUNCTIONS 157

14.1 LIST OF FUNCTIONS 158
14.2 LIST OF MACROS 158
14.3 HDVICP20_TI_VDMA_OPEN 159
14.4 HDVICP20_TI_VDMA_PUSHTRFDESCNONDET 160
14.5 HDVICP20_TI_VDMA_SWRESET 160
14.6 HDVICP20_TI_VDMA_TRIGGERGROUPDET 161
14.7 HDVICP20_TI_VDMA_WAIT 161
14.8 HDVICP20_TI_VDMA_CLEAR 162
14.9 HDVICP20_TI_VDMA_CONFIGURE1DCOPYDET 162
14.10 HDVICP20_TI_VDMA_CONFIGURE1DCOPYWITHAUTOINCDET 163
14.11 HDVICP20_TI_VDMA_CONFIGURE2DCOPYDET 164
14.12 HDVICP20_TI_VDMA_CONFIGURE2DCOPYWITHAUTOINCDET 164
14.13 HDVICP20_TI_VDMA_CONFIGURE2DCOPYWITHAUTOINCPADDINGDET 165
14.14 HDVICP20_TI_VDMA_CONFIGURE2DCOPYWITHAUTOINCDATAPROCDET 166
14.15 HDVICP20_TI_VDMA_UPDATE1DCOPYWITHAUTOINCDET 168
14.16 HDVICP20_TI_VDMA_PREPTRFDESC1DCOPYNONDETSHORT 168
14.17 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYSL2TOSL2NONDETSHORT 169
14.18 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYNONDETSHORT 170
14.19 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYDDRTODDRNONDET 171
14.20 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYNONDET 172
14.21 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYWITHPADDINGNONDET 173
14.22 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYWITHDECIMATIONNONDET 174
14.23 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYWITHUVINTERLEAVINGNONDET 175
14.24 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYWITHCOMPRESSIONNONDET 176
14.25 HDVICP20_TI_VDMA_PREPTRFDESC2DCOPYWITHDATAPROCESSINGNONDET 177
14.26 HDVICP20_TI_VDMA_UPDATEDSTMODULO 178
14.27 HDVICP20_TI_VDMA_UPDATESRCMODULO 179
14.28 HDVICP20_TI_VDMA_UPDATESRCBYTEINC 179
14.29 HDVICP20_TI_VDMA_UPDATENUMBYTES 180
14.30 HDVICP20_TI_VDMA_UPDATEDST 180
14.31 HDVICP20_TI_VDMA_UPDATESRC 181
14.32 HDVICP20_TI_VDMA_UPDATEYOFFSET 181
14.33 HDVICP20_TI_VDMA_UPDATEXOFFSET 182
14.34 HDVICP20_TI_VDMA_UPDATEDSTPITCH 182
14.35 HDVICP20_TI_VDMA_UPDATEDSTPITCHNONDET 183
14.36 HDVICP20_TI_VDMA_UPDATEDSTNONDET 183
14.37 HDVICP20_TI_VDMA_UPDATESRCNONDET 184
14.38 HDVICP20_TI_VDMA_UPDATEYOFFSETNONDET 184
14.39 HDVICP20_TI_VDMA_UPDATEXOFFSETNONDET 185
14.40 HDVICP20_TI_VDMA_SETFIRSTFLAGNONDET 185
14.41 HDVICP20_TI_VDMA_SETLASTFLAGNONDET 186
14.42 HDVICP20_TI_VDMA_RESETFIRSTFLAGNONDET 186
14.43 HDVICP20_TI_VDMA_RESETLASTFLAGNONDET 187
14.44 HDVICP20_TI_VDMA_TURNOFFPADDINGDET 187
14.45 HDVICP20_TI_VDMA_TURNOFFPADDINGNONDET 188
14.46 VDMA API USAGE EXAMPLE 188

15. MAILBOX FUNCTIONS 191

15.1 LIST OF FUNCTIONS 191

15.2 HDVICP20_TI_MBX_SOFTRESET 191
15.3 HDVICP20_TI_MBX_IRQINTERRUPTENABLE 192
15.4 HDVICP20_TI_MBX_IRQINTERRUPTDISABLE 192
15.5 HDVICP20_TI_MBX_GETIRQSTATUS 193
15.6 HDVICP20_TI_MBX_CLEARIRQSTATUS 194
15.7 HDVICP20_TI_MBX_GETMESSAGESTATUS 194
15.8 HDVICP20_TI_MBX_GETFIFOSTATUS 195
15.9 HDVICP20_TI_MBX_WRITEMESSAGE 195
15.10 HDVICP20_TI_MBX_READMESSAGE 196
15.11 MAILBOX API USAGE EXAMPLE 197

16. SMSET FUNCTIONS 201

16.1 LIST OF FUNCTIONS 201
16.2 HDVICP20_TI_SMSET_GETOWNERSHIP 201
16.3 HDVICP20_TI_SMSET_ENABLE 201
16.4 HDVICP20_TI_SMSET_RESETSTARTADDRESS 202
16.5 HDVICP20_TI_SMSET_ENABLEALLEVENTS 202
16.6 HDVICP20_TI_SMSET_STARTTRACE 203

17. HOST (M3) SIDE APIS 205

17.1 LIST OF FUNCTIONS 205
17.2 HDVICP20_TI_VDMA_OPEN_EXTMEM_SL2 206
17.3 HDVICP20_TI_VDMA_PREPARE_EXTMEM_SL2 206
17.4 HDVICP20_TI_VDMA_TRIGGER_EXTMEM_SL2 207
17.5 HDVICP20_TI_VDMA_WAITFORDATA_EXTMEM_SL2 208
17.6 HDVICP20_TI_DM_PREPARE_SL2_TCM 208
17.7 HDVICP20_TI_DM_TRIGGER_SL2_TCM 209
17.8 HDVICP20_TI_DM_WAITFORDATA_SL2_TCM 209

18. APPENDIX A 211

18.1 USAGE OF COMPUTE IP APIS 211

1. Getting Started

1.1 Introduction

HDVICP2.0/IVA-HD API library is an enabler for external development of video codecs by 3P's & customers during Pre (limited) and Post silicon phase. It encapsulates the hardware low level programming APIs to facilitate the ease of development on IVA-HD.

HDVICP2.0/IVA-HD API library package contains

- HDVICP2.0/IVA-HD co-processor API library with source code
- API User guide (this document)

Following sub-sections provide details of each of the above mentioned components of IVA-HD API library package.

1.2 HDVICP2.0/IVA-HD API Library

IVA-HD encapsulates six algorithm accelerators, multiple peripherals, and two ARM968 CPUs to support high-definition real-time video coding. To facilitate software development, a set of API functions are defined to support different coding standards.

IVA-HD API library provides interface APIs for the following modules/blocks of IVA-HD:

- Calculation Engine (CALC3)
- Entropy Coding/Decoding Engine (ECD3)
- Intra-prediction Estimation Engine (IPE3)
- Loop Filter Engine (iLF3)
- Motion Compensation Engine (MC3)
- Motion Estimation Engine (iME3)
- VDMA (Video DMA Engine)
- Load-Store Engine (LSE)
- SyncBox (SB)
- SyncBox Handler (SBH)
- Data Mover (DM)
- Mail Box (MB)

Of the mentioned hardware blocks CALC3, ECD3, IPE3, iLF3, MC3 and iME3 are known as algorithm specific accelerators as the programming of these modules is different for each video codec algorithm/standard. Remaining modules are non-algorithm specific and their programming is generic for all video codec implementations.

IVA-HD API library provides APIs for algorithm specific accelerators for following video codecs only.

- H.264 encoder and decoder (Baseline profile)
- MPEG-4 encoder and decoder (Simple profile)
- VC-1 decoder (Advanced profile)
- MPEG-2 decoder (Main profile)

APIs for non-algorithm accelerators provided in the IVA-HD API library can be used for any of video codecs and other video processing algorithm implementations on IVA_HD.

For more details on APIs, please refer to chapters 4 to 15 of this document.

1.3 Guide to this Manual

To use this manual, the following chapters and sections are recommended to be read:

Section 1.3: Terms and abbreviations used specifically in this document

Section 1.4: This section describes the steps to install the required software and the API library.

Section 1.5: This section describes the steps to build the API library

Section 1.8: Basic data types used in the IVA-HD API implementation

Chapter 2: Hardware resource definitions and register masks

Chapter 3: Codec-specific data structures used by the IVA-HD APIs for the relevant IPs

Chapter 4 to 9: API functions defined for the Compute IPs for the following codecs: H.264 Decoder, H.264 Encoder, MPEG-4 Decoder, MPEG-4 Encoder, VC1 Decoder, and MPEG-2 Decoder.

Chapter 10 to 17: API functions defined for the Non-Compute IPs. These APIs are codec-independent and can be used in all codec implementations.

1.4 Terms and abbreviations

IVA-HD	Imaging and Video Accelerators – High Definition
Accelerators	Algorithm accelerators embedded in IVA-HD
HWA	Hardware Accelerator. Same as Accelerator above.
CPU	Refers to one of the two ARM968E processors present in IVA-HD.
iCONT	iCONT1 and iCONT2 are two identical controller units present in IVA-HD. Each iCONT comprises of an ARM968E CPU, a SyncBox, a SyncBox Handler, a Data Mover, and a IRQ controller. Each also has two Tightly Coupled Memories (TCMs) – Instruction TCM and Data TCM.
CSL	Chip Support Library

IP	IP refers to any hardware algorithm accelerator or peripheral (eg., VDMA, SyncBox) in IVA-HD
Compute IPs	The accelerators CALC3, ECD3, iLF3, iME3, IPE3 and MC3 are termed as Compute IPs in this document.
Non-Compute IPs	The IPs VDMA, SyncBox, SyncBox Handler, LSE, Data Mover, and Mailbox are termed as Non-Compute IPs in this document.
SL2 Memory	Shared L2 Memory that is commonly used by all IPs and iCONTs.

1.5 Installation

1.5.1 System requirements

Hardware

None

Software

The following are the software requirements for the normal functioning of the codec:

- **Development Environment:** This project is developed using Code Composer Studio version (CCSv4 Platinum) 4.0.2.01003. It has been tested IVA-HD simulator.
  - CCSv4 installer can be downloaded from http://software-dl-1.ti.com/dsps/forms/self_cert_export.html?prod_no=setup_CCS_4_Platinum.zip&ref_url=http://software-dl.ti.com/dsps/dsps_public_sw/sdo_ccstudio/CCSv4/CCS_4_0_2/
  - IVA-HD simulator can be installed thorugh CCSv4 updates.
- **Code Generation Tools:** This project is compiled, assembled, archived and linked using the TMS470Rx code generation tools version 4.5.1 (for HDVICP2 processor related APIs) and code generation tools version 5.0.3(for Media Controller processor related APIs). These tools come as a part of CCSv4 Installation and also can be downlaoded form https://www-a.ti.com/downloads/sds_support/CodeGenerationTools.htm

1.5.2 Installing the API library

The HDVICP2.0/IVA-HD API library is released as a compressed archive. To install the API library, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a top-level directory called 500_V_HDVICP20_X_01_00. The following directories exist in the root directory of the IVA-HD Library package:

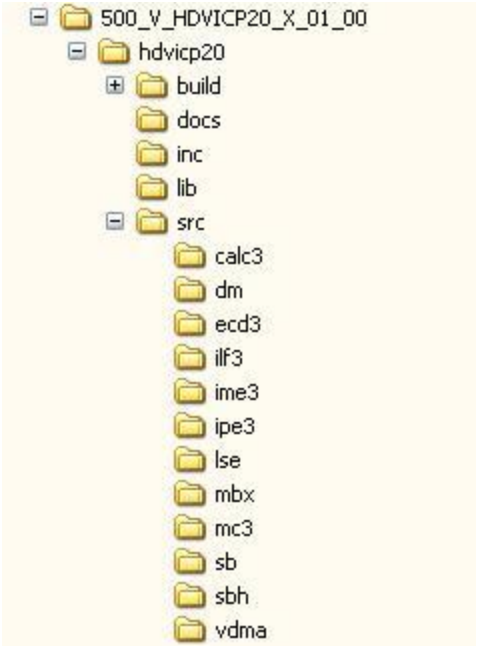

Table 1. HDVICP2.0/IVA-HD API Library Directories

Sub-Directory	Description
\hdvicp20 \build\	Contains the CCSv4 project files for the IVA-HD API library
\hdvicp20 \build\host_m3	Contains the CCSv4 project files for the IVA-HD Host API library
\ hdvicp20 \docs\	Contains IVA-HD API document – User Guide (this document).
\ hdvicp20 \inc\	Contains IVA-HD API header files required for using the IVA-HD API library and Host API library.
\ hdvicp20\src\{ip}	Each folder contains IVA-HD API source files pertaining to the respective IPs. There is a single source file for each of the non-compute IPs. For compute IPs, there is a separate source file for each codec component. Note that there are 12 IPs {ip} – calc3, dm, ecd3, ilf3, ime3, ipe3, lse, mbx, mc3, sb, sbh and vdma.

Sub-Directory	Description
\ hdvicp20 \lib\	Contains the IVA-HD API library for ARM968 and Host API library for Cortex-M3 **ivahd_ti_api_arm968.lib** : API library built in ARM mode with ELF format **ivahd_ti_api_arm968_thumb.lib** : IVA-HD API library built in THUMB mode with ELF format **ivahd_ti_api_vM3.lib :** API library for HOST/M3 side usage built for Cortex-M3 with ELF format. Please refer to Chapter 17 for details of HOST APIs

1.5.3 Installing HDVCIP2.0/IVA-HD CSL/CSP

The HDVICP2.0/IVA-HD API library needs pre-installed IVA-HD CSL/CSP (Chip Support Library/Package). As the CSL is not part of this package, please contact Texas Instruments to get the same. The version of the CSL used current HDVICP2.0 API library is mentioned in the release notes. After CSL/CSP installation, set the system environment variable **CSP_INSTALL_DIR** to point to the location where the CSL is present. Following is directory structure for IVAHD CSP/CSL.

Ex: If IVA-HD CSP/CSL is installed in D:\IVAHD_CSL\, set environment variable CSP_**INSTALL_DIR to D:\ IVAHD_CSL\csp**

1.5.4 Setup Environment Variables

- Set a system environment variable named **CSP_INSTALL_DIR** pointing to <csp_directory>\csp. Please refer section 1.5.3

- Set a system environment variable named **CG_TOOL_DIR** pointing to <cg_tools v4.5.1>. Ex. Set CG_TOOL_DIR environment variable to C:\CCStudioV4.0\ccsv4\tms470\cgtools\4_5_1\ (This path contains bin, docs, include folders and etc,)

- Set a system environment variable named **CG_TOOL_DIR_M3** pointing to <cg_tools v5.0.3>. Ex. Set CG_TOOL_DIR_M3 environment variable to C:\CCStudioV4.0\ccsv4\tms470\cgtools\5_0_3\ (This path contains bin, docs, include folders and etc,)

1.5.5 Installing IVA-HD Simulator

Make sure that CCSv4 is installed already in your system with code generation tools v4.5.1 and v5.0.3 for TMS470Rxx.

After installing CCS, install IVA-HD simulator from CCSv4 updates. Detailed instructions for installation can be found in ivahd_sim_user_guide.pdf present in <CCSv4 Installation Dir>\simulation_csp_omap4\docs\pdf\ directory.

1.6 Build and Sample usage

To build and use the HDVICP2.0/IVA-HD API library in Code Composer Studio (CCSv4), follow these steps described in below paragraphs.

1.6.1 Building IVA-HD API Library

1) Verify that you have installed TI's Code Composer Studio (CCSv4) with code generation tools version 4.5.1, code generation tools version 5.0.3 and IVA-HD simulator. Start the Code Composer Studio and setup the CCS for Netra/OMAP4 configuration.

2) After Installing Code Generation Tools, set the system environment variable **CG_TOOL_DIR** pointing to cg tools v4.5.1 and **CG_TOOL_DIR_M3** pointing to CG tools v5.0.3. Please refer to Sec 1.5.4 for details.

3) Verify that you have installed IVA-HD CSP (containing CSL) and set the system environment variable **CSP_INSTALL_DIR** to point to the location where the CSP/CSL is present. Please refer to Sec 1.5.3 above for details.

4) Add the IVA-HD API library project named "ivahd_ti_api_arm968" through "Import Existing CCS/CCE Eclipse Project" option to the workspace. All files required for this project are available in the \500_V_HDVICP20_X_01_00\hdvicp20\build\ sub-directory.

5) Import macros definition file: Select File->Import->CCS->Managed Build Macros and browse through projector to select "macros.ini" file available in \500_V_HDVICP20_X_01_00\hdvicp20\build\ sub-directory.

6) Select Project > Rebuild All. This will build and create the final API library file, ivahd_ti_api_arm968.lib at \500_V_HDVICP20_X_01_00\hdvicp20\lib\ sub-directory.

7) The generated library can be included in codec specific project files to use IVA-HD APIs. The header files needed for using API library are available at \500_V_HDVICP20_X_01_00\hdvicp20\inc\ sub-directory

8) HDVICP2 Media Controller (M3/Host) API Library ivahd_ti_api_vM3.lib is generated by compiling the vM3-API related sources mentioned in \500_V_HDVICP20_X_01_00\hdvicp20\build\host_m3\makefile. To compile using this makefile please follow the below steps.

   - Open the windows command prompt and visit the directory of makefile \500_V_HDVICP20_X_01_00\hdvicp20\build\host_m3 and run the following commands
   - Project Clean: gmake –k –s clean
   - Project Build: gmake –k –s deps

     gmake –k –s all

Note: Default CCS project build configurations "Debug" & "Release" will build "ivahd_ti_api_arm968.lib" in ARM mode. To build the THUMB mode API library "ivahd_ti_api_arm968_thumb.lib", please use CCS project build configuration named "Thumb_Mode".

1.7 IVA-HD API Basic Data types

Below basic data types are followed in the IVA-HD API library source code. These data types are defined in hdvicp20_ti_datatypedefs.h.

Name	Data Type
U8	unsigned char
U16	unsigned short
U32	unsigned integer
U64	unsigned long
S8	Char
S16	Short
S32	Int
S64	long int
pU8	pointer to unsigned char
pU16	pointer to unsigned short
pU32	pointer to unsigned integer
pU64	pointer to unsigned long
pS8	pointer to char
pS16	pointer to short
pS32	pointer to integer
pS64	pointer to signed long

THIS PAGE IS INTENTIONALLY LEFT BLANK

2. Hardware Resource Files

API functions are based on a set of hardware resources. These resources include coprocessor base addresses, coprocessor buffers and registers.

Header files that contain definitions of these resources are listed in the following sections.

2.1 Coprocessor base addresses (CSL)

Registers and buffers are addressed by offset to the base address of the coprocessors observed by the host. These addresses are defined in a single file and included as part of the chip support library (CSL).

File $ROOT/csp/csl_soc/cslr_ivahd.h

Macros Defined

See file for details.

2.2 IVA-HD Register Overlays and Masks

Register definition includes register masks. These overlays and masks are defined in the header files in the following directory. Accelerator internal buffers are also defined as overlay structures containing arrays of respective sizes.

Directory $ROOT/csp/csl_ivahd/*.h

Macros Defined

See files for details.

2.3 Common Hardware Structure Data Types

Structure data types are defined for both the accelerators and the codecs. These type definitions are categorized as hardware data types and codec data types. Hardware data types describe register within each hardware blocks. Codec data types describe codec status and parameters. The following is the list of IVA-HD hardware structure data types.

2.3.1 CSL_Calc3_ipgw_Regs

CALC3 IPGW Registers: These registers are used to control the interrupts associated with CALC3.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	CALC3_SYSCONFIG	Configuration Register
U32	CALC3_IRQ_EOI	End of Interrupt number specification
U8	RSVD0[4]	Reserved
U32	CALC3_IPQSTATUS_RAW_0	Per-event raw interrupt status vector, int_end load (LSE). Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	CALC3_IPQSTATUS_RAW_1	Per-event raw interrupt status vector, int_end (CALC3

		Core).
U32	CALC3_IPQSTATUS_RAW_2	Per-event raw interrupt status vector, int_end store (LSE).
U32	CALC3_IPQSTATUS_RAW_3	Per-event raw interrupt status vector, int_undef (LSE).
U8	RSVD1[12]	Reserved
U32	CALC3_IRQSTATUS_0	Per-event "enabled" interrupt status vector, int_end load (LSE). Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	CALC3_IRQSTATUS_1	Per-event "enabled" interrupt status vector, int_end (CALC3 Core).
U32	CALC3_IRQSTATUS_2	Per-event "enabled" interrupt status vector, int_end store (LSE).
U32	CALC3_IRQSTATUS_3	Per-event "enabled" interrupt status vector, int_undef (LSE).
U8	RSVD1[12]	Reserved
U32	CALC3_IRQENABLE_SET_0	Per-event interrupt enable bit vector, int_end load (LSE). Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	CALC3_IRQENABLE_SET_1	Per-event interrupt enable bit vector, int_end (CALC3 Core).
U32	CALC3_IRQENABLE_SET_2	Per-event interrupt enable bit vector, int_end store (LSE).
U32	CALC3_IRQENABLE_SET_3	Per-event interrupt enable bit vector, int_undef (LSE).
U8	RSVD3[12]	Reserved
U32	CALC3_IRQENABLE_CLR_0	Per-event interrupt enable bit vector, int_end load (LSE). Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	CALC3_IRQENABLE_CLR_1	Per-event interrupt enable bit vector, int_end (CALC3 Core).
U32	CALC3_IRQENABLE_CLR_2	Per-event interrupt enable bit vector, int_end store (LSE).
U32	CALC3_IRQENABLE_CLR_3	Per-event interrupt enable bit vector, int_undef (LSE).
U8	RSVD4[72]	Reserved
U32	CALC3_IRQSTATUS_ACLREN	Auto Clear enable

2.3.2 CSL_Calc3_bfsw_Regs

CALC3 Buffer Switch (BFSW) Module Registers: These registers are used to control the buffer switching in ping-pong mode and to switch masters (HWA / DMA) for the buffers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	VIEWMODE	View Mode Register. It selects full-view mode or ping-pong view mode.
U32	MSTID1	Master ID 1 Register

		Select master between HWA and DMA bus. This register is used in both full view and ping-pong view mode. This register is for buffers which has two physical memories.
U32	MSTID2	Master ID 2 Register Select master between HWA and DMA bus. This register is for buffers which has only one physical memory.

2.3.3 CSL_Calc3_mmr_Regs

CALC3 Memory Mapped Registers: These registers are used to control the CALC3 core.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	CALC_PID	Peripheral Identification Register
U32	CALC_COUNT	CALC3 cycle counter register
U32	CALC_CTRL	CALC3 control register
U32	CALC_TEST	CALC3 test register It is only for debug purpose.
U32	CALC_MODE	CALC3 mode select register
U32	CALC_FWDQ_RND_INTRA	CALC3 forward quantiation's rounding coefficients and shift offsets for intra MB
U32	CALC_FWDQ_RND_INTER	CALC3 forward quantiation's rounding coefficients and shift offsets for inter MB
U32	CALC_FWDQ_RND_INTRA_DC	Round offset value setting for fwd Q, intra and DC coefficient.
U32	CALC_FWDQ_RND_INTER_DC	Round offset value setting for fwd Q, inter and DC coefficient.

2.3.4 CSL_Calc3_mem_Regs

CALC3 Internal Memory: This structure defines the buffers used by the CALC3 core within the CALC3 internal memory.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	CALC3RPBUF_A_START	CALC3RPBUF A Buffer Start
U8	RSVD0[3064]	Buffer Space
U32	CALC3RPBUF_A_END	CALC3RPBUF A Buffer End
U8	RSVD1[1024]	Buffer Space
U32	CALC3RPBUF_B_START	CALC3RPBUF B Buffer Start
U8	RSVD2[3064]	Buffer Space
U32	CALC3RPBUF_B_END	CALC3RPBUF B Buffer End
U8	RSVD3[1024]	Buffer Space
U32	CALC3ROBUF_A_START	CALC3ROBUF A Buffer Start

U8	RSVD4[2040]	Buffer Space
U32	CALC3ROBUF_A_END	CALC3ROBUF A Buffer End
U32	CALC3ROBUF_B_START	CALC3ROBUF B Buffer Start
U8	RSVD5[2040]	Buffer Space
U32	CALC3ROBUF_B_END	CALC3ROBUF B Buffer End
U32	CALC3WBUF_START	CALC3WBUF A Buffer Start
U8	RSVD6[2040]	Buffer Space
U32	CALC3WBUF_END	CALC3WBUF A Buffer End
U32	CALC3QBUF_START	CALC3QBUF A Buffer Start
U8	RSVD7[2040]	Buffer Space
U32	CALC3QBUF_END	CALC3QBUF A Buffer End

2.3.5 CSL_Ecd3_ipgw_Regs

ECD3 IPGW Registers: These registers are used to control the interrupts associated with ECD3.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ECD3_SYSCONFIG	Configuration Register
U32	ECD3_IRQ_EOI	
U32	ECD3_IRQ_MERGED_STATUS	
U32	ECD3_IPQSTATUS_RAW_0	
U32	ECD3_IPQSTATUS_RAW_1	
U32	ECD3_IPQSTATUS_RAW_2	
U32	ECD3_IPQSTATUS_RAW_3	
U32	ECD3_IPQSTATUS_RAW_4	
U32	ECD3_IPQSTATUS_RAW_5	
U32	ECD3_IPQSTATUS_RAW_6	
U32	ECD3_IRQSTATUS_0	
U32	ECD3_IRQSTATUS_1	
U32	ECD3_IRQSTATUS_2	
U32	ECD3_IRQSTATUS_3	
U32	ECD3_IRQSTATUS_4	
U32	ECD3_IRQSTATUS_5	
U32	ECD3_IRQSTATUS_6	
U32	ECD3_IRQENABLE_SET_0	
U32	ECD3_IRQENABLE_SET_1	
U32	ECD3_IRQENABLE_SET_2	
U32	ECD3_IRQENABLE_SET_3	

U32	ECD3_IRQENABLE_SET_4	
U32	ECD3_IRQENABLE_SET_5	
U32	ECD3_IRQENABLE_SET_6	
U32	ECD3_IRQENABLE_CLR_0	
U32	ECD3_IRQENABLE_CLR_1	
U32	ECD3_IRQENABLE_CLR_2	
U32	ECD3_IRQENABLE_CLR_3	
U32	ECD3_IRQENABLE_CLR_4	
U32	ECD3_IRQENABLE_CLR_5	
U32	ECD3_IRQENABLE_CLR_6	
U8	RSVD0[60]	Reserved
U32	ECD3_IRQSTATUS_ACLREN	

2.3.6 CSL_Ecd3_bfsw_Regs

ECD3 Buffer Switch (BFSW) Module Registers: These registers are used to control the buffer switching in ping-pong mode and to switch masters (HWA / DMA) for the buffers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	VIEWMODE	View Mode Register. It selects full-view mode or ping-pong view mode.
U32	MSTID1	Master ID 1 Register Select master between HWA and DMA bus. This register is used in both full view and ping-pong view mode. This register is for buffers which has two physical memories.
U32	MSTID2	Master ID 2 Register Select master between HWA and DMA bus. This register is for buffers which has only one physical memory.

2.3.7 CSL_Ecd3_mem_Regs

ECD3 Internal Memory: This structure defines the buffers used by the ECD3 core within the ECD3 internal memory.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ECD3ABUF_A_START	ECD3ABUF A Buffer Start
U8	RSVD0[2040]	Buffer Space
U32	ECD3ABUF_A_END	ECD3ABUF A Buffer End
U32	ECD3ABUF_B_START	ECD3ABUF B Buffer Start
U8	RSVD1[2040]	Buffer Space
U32	ECD3ABUF_B_END	ECD3ABUF B Buffer End
U8	RSVD2[4096]	Reserved

U32	ECD3WBUF_START	ECD3WBUF Buffer Start
U8	RSVD3[6136]	Buffer Space
U32	ECD3WBUF_END	ECD3WBUF Buffer End
U8	RSVD4[10240]	Buffer Space
U32	ERSDBUF_A_START	ERSDBUF A Buffer Start
U8	RSVD5[2040]	Buffer Space
U32	ERSDBUF_A_END	ERSDBUF A Buffer End
U32	ERSDBUF_B_START	ERSDBUF B Buffer Start
U8	RSVD6[2040]	Buffer Space
U32	ERSDBUF_B_END	ERSDBUF B Buffer End

2.3.8 CSL_Ecd3_jpg_Regs

This structure defines the ECD3 JPEG-specific registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	INT_STATUS	
U32	INT_MASK	
U32	JPEG_CTRL	
U32	VLC_HUFFPTR_DC	
U32	VLC_HUFFPTR_AC	
U32	UVLD_CTRL_TBPTR_DC	
U32	UVLD_CTRL_TBPTR_AC	
U32	UVLD_CODE_TBPTR_DC	
U32	UVLD_CODE_TBPTR_AC	
U32	UVLD_TBL_TYPE	
U32	DC_PRED_CHROMA	

2.3.9 CSL_Ecd3_mp2_Regs

This structure defines the ECD3 MPEG2-specific registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	MP2_STAT	Status data
U32	MP2_MASK	MPEG2 mask: sets codec parameters
U32	MP2_WORK0	Work information
U32	MP2_WORK1	Work information
U32	MP2_WORK2	Work information
U32	MP2_WORK3	Work information

U32	MP2_WORK4	Work information
U32	MP2_WORK5	Work information
U32	MP2_WORK6	Work information
U8	RSVD0[8]	Reserved

2.3.10 CSL_Ecd3_mp4_Regs

This structure defines the ECD3 MPEG4-specific registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	MP4_STAT	Status data
U32	MP4_MASK	MPEG4 mask: sets codec parameters
U32	MP4_WORK0	Work information
U32	MP4_WORK1	Work information
U32	MP4_WORK2	Work information
U32	MP4_WORK3	Work information
U32	MP4_WORK4	Work information
U32	MP4_WORK5	Work information
U32	MP4_WORK6	Work information
U32	MP4_WORK7	Work information
U8	RSVD0[4]	Reserved

2.3.11 CSL_Ecd3_vc1_Regs

This structure defines the ECD3 VC1-specific registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	VC1_STAT	Status data
U32	VC1_MASK	VC1 mask: sets codec parameters
U32	VC1_WORK	Work information
U32	VC1_VP_4	
U32	VC1_VP_5	
U32	VC1_VP_6	
U32	VC1_VP_7	
U32	VC1_VP_8	
U32	VC1_VP_9	
U32	VC1_VP_A	
U8	RSVD0[8]	Reserved

2.3.12 CSL_Ecd3_h264_Regs

This structure defines the ECD3 H.264-specific registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ERR_MSK	Error Mask
U32	ERR_STAT	Error Status
U32	CFG_QP	Configuration and QP
U32	SKIP_RUN	Number of skipped macroblocks left in CAVLC
U32	CABAC_REG	CABAC Registers
U32	SYM_CNT	Total number of CABAC bin symbols encoded since the last CABAC initialization. This register is reset to zero when H.264 core is invoked for a first macroblock in a slice. This register is not used in decoding or in CAVLC mode.
U32	BITS_OSTD	The total number of outstanding bits in CABAC encoder. This register is not used in decoding or in CAVLC mode.
U32	MVD_CUR_PTR	MVD Current Pointer
U32	MVD_LFT_PTR	MVD Left Pointer
U8	RSVD0[4]	Reserved

2.3.13 CSL_Ecd3_avs_Regs

This structure defines the ECD3 AVS-specific registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	AVS_STAT	Status data
U32	AVS_MASK	AVS mask: sets codec parameters
U32	AVS_WORK0	Work information
U32	AVS_WORK1	Work information
U8	RSVD0[28]	Reserved

2.3.14 CSL_Ecd3_cdc_Regs

This structure combines all the ECD3 codec-specific registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
CSL_Ecd3_jpg_Regs	JPEG	JPEG-specific registers
CSL_Ecd3_mp2_Regs	MPEG2	MPEG2-specific registers
CSL_Ecd3_mp4_Regs	MPEG4	MPEG4-specific registers
CSL_Ecd3_vc1_Regs	VC1	VC1-specific registers
CSL_Ecd3_h264_Regs	H264	H264-specific registers

CSL_Ecd3_avs_Regs	AVS	AVS -specific registers

2.3.15 CSL_Ecd3_mmr_Regs

ECD3 Memory Mapped Registers: These registers are used to control the ECD3 core.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ECD_PID	Peripheral Identification Register
U32	ECD_COUNT	Cycle counter
U32	ECD_CTRL	ECD3 Control
U32	ECD_STAT	ECD3 Status
U32	SBC_CTRL	Stream Buffer Controller (SBC) Control
U32	SBC_STAT	Stream Buffer Controller (SBC) Status
U32	SBC_BUFCFG	Stream Buffer Controller (SBC) Buffer Configuration
U32	SBC_A_BITPTR	SBC Bitstream Pointer for A buffer
U32	SBC_A_DMAPG	Indicates if the page (A buffer) is being accessed from DMA for bit-stream data transferring.
U32	SBC_B_BITPTR	SBC Bitstream Pointer for B buffer
U32	SBC_B_DMAPG	Indicates if the page (B buffer) is being accessed from DMA for bit-stream data transferring.
U32	SBC_TTLCNT	Total bit count.
U32	SBC_RSDCNT	Bit count for residual layer
U8	RSVD0[4]	Reserved
U32	SBC_SRCH_PG_CNT	ECD3 searches start code untill the page counter reach this number.
U32	SBC_FMO_DMA_STAT	Indicates FMO_DMA_ID for stream interrupt at buffer page boundary.
U32	MBPC_PIC_DIM	Macroblock Position Controller (MBPC) Picture Dimensions
U8	RSVD1[12]	Reserved
U32	MBPC_STAT	MBPC Status
U32	MBPC_POS	MBPC MB position
U32	MBPC_PMC	Macroblock count in the picture
U32	MBPC_SMC	Macroblock count in the slice
U8	RSVD2[4]	Reserved
U32	DTBC_BP_MB	Data Buffer Controller (DTBC) base pointers to the upper macroblock buffer and current macroblock buffer
U32	DTBC_BP_COL	(DTBC) base pointers to the co-located macroblock buffers A & B

U32	DTBC_BP_RSD	Base pointer to the residual data buffer
U32	DTBC_DP_UL	Current pointer to the upper left macroblock data in work buffer
U32	DTBC_DP_UU_UR	Current pointer to the upper & upper right macroblock data in work buffer
U32	DTBC_DP_LL_CUR	Current pointer to the left & current macroblock data in work buffer
U8	RSVD3[8]	Reserved
U32	DTBC_DP_ULUR2	current pointer to the macroblock left to the upper left macroblock data in work buffer & current pointer to the macroblock right to the upper right macroblock data in work buffer
U32	DTBC_DP_SLICE	pointer to slice or picture information data in work buffer
U32	DTBC_DP_LL2	current pointer to the macroblock left to the left macroblock data in work buffer
U32	DTBC_CUR_MB_SIZE	data element size for Current MB, Upper right MB in ECDABUF and Upper MB, Current/Left MB in ECDWBUF
U8	RSVD4[12]	Reserved
U32	CDC_MODE	Codec Mode
CSL_Ecd3_cdc_Regs	CDC_SPECIFIC	Codec-specific registers
U8	RSVD5[32]	Reserved
U32	CMDP_GPR0	Command Processor General purpose registers (GPR0, GPR1, and GPR2) are 32-bit registers which can be used freely.
U32	CMDP_GPR1	Command Processor General purpose registers (GPR0, GPR1, and GPR2) are 32-bit registers which can be used freely.
U32	CMDP_GPR2	Command Processor General purpose registers (GPR0, GPR1, and GPR2) are 32-bit registers which can be used freely.

2.3.16 CSL_Icecrusher1_cfg_ic968_Regs

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	VERSION	
U32	DCCR	
U32	DCSR	
U8	RSVD0[4]	Reserved
U32	TCR	
U32	RCSR	

U32	THREADID_CLAIM	
U32	THREADID	
U32	ICSR	
U32	ETM_CTRL	
U32	ETM_PROCID	
U32	TESTREG	
U8	RSVD1[16]	Reserved
U32	BMCTL0	
U32	BMCTR0	
U8	RSVD2[8]	Reserved
U32	BMCTL1	
U32	BMCTR1	
U8	RSVD3[40]	Reserved
U32	HWBP_CTRL0	
U32	HWBP_ADDR0	
U32	HWBP_MASK0	
U8	RSVD4[4]	Reserved
U32	HWBP_CTRL1	
U32	HWBP_ADDR1	
U32	HWBP_MASK1	
U8	RSVD5[4]	Reserved
U32	HWBP_CTRL2	
U32	HWBP_ADDR2	
U32	HWBP_MASK2	
U8	RSVD6[4]	Reserved
U32	HWBP_CTRL3	
U32	HWBP_ADDR3	
U32	HWBP_MASK3	
U8	RSVD7[60]	Reserved
U32	OS_LOCK	
U32	DCON	

2.3.17 CSL_Icont_imem_Regs

iCONTx ITCM: This structure defines the internal code memory of iCONTx.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ICONT_IMEM_START	ICONT Instruction Memory Start

U8	RSVD0[32760]	ITCM Memory
U32	ICONT_IMEM_END	ICONT Instruction Memory End

2.3.18 CSL_Icont1_mmr_dm_Regs

Data Mover (DM) Memory Mapped Registers: These registers are used to control the DM functionality.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ICONT_DM_IRQ_EOI	End of Interrupt number specification
U32	ICONT_DM_IRQSTATUS_RAW	Per-event raw interrupt status vector (DM interrupt). Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	ICONT_DM_IRQSTATUS	Per-event "enabled" interrupt status vector (DM interrupt). Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	ICONT_DM_IRQENABLE_SET	Per-event interrupt enable bit vector (DM interrupt). Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	ICONT_DM_IRQENABLE_CLR	Per-event interrupt enable bit vector (DM interrupt). Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U8	RSVD0[12]	Reserved
U32	ICONT_DMSOURCEADD_0	Data mover source address for context 0 (Byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMDESTADD_0	Data mover destination address for context 0 (byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMCONTEXT_0	Data Mover Context 0. A new DM transfer can be scheduled only if this register has been writen by SW.
U32	ICONT_DMSTATUS_0	Data Mover status register for context 0.
U8	RSVD1[16]	Reserved
U32	ICONT_DMSOURCEADD_1	Data mover source address for context 1 (Byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMDESTADD_1	Data mover destination address for context 1 (byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMCONTEXT_1	Data Mover Context 1. A new DM transfer can be scheduled only if this register has been writen by SW.
U32	ICONT_DMSTATUS_1	Data Mover status register for context 1.
U8	RSVD2[16]	Reserved

U32	ICONT_DMSOURCEADD_2	Data mover source address for context 2 (Byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMDESTADD_2	Data mover destination address for context 2 (byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMCONTEXT_2	Data Mover Context 2. A new DM transfer can be scheduled only if this register has been writen by SW.
U32	ICONT_DMSTATUS_2	Data Mover status register for context 2.
U8	RSVD3[16]	Reserved
U32	ICONT_DMSOURCEADD_3	Data mover source address for context 3 (Byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMDESTADD_3	Data mover destination address for context 3 (byte address, must be aligned on 128-bit boundary)
U32	ICONT_DMCONTEXT_3	Data Mover Context 3. A new DM transfer can be scheduled only if this register has been writen by SW.
U32	ICONT_DMSTATUS_3	Data Mover status register for context 3.

2.3.19 CSL_Icont_dmem_Regs

iCONTx DTCM: This structure defines the internal data memory of iCONTx.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ICONT_DMEM_START	ICONT Data Memory Start
U8	RSVD0[16376]	DTCM Memory
U32	ICONT_DMEM_END	ICONT Data Memory End

2.3.20 CSL_Icont1_mmr_cfg_Regs

iCONTx Memory Mapped Registers: iCONTx configuration registers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ICONT_REVISION	ICONT Revision Identifier (X.Y.R) Used by software to track features, bugs, and compatibility
U32	ICONT_HWINFO	Information about the IP module's hardware configuration.
U8	RSVD0[8]	Reserved
U32	ICONT_SYSCONFIG	Clock management configuration

2.3.21 CSL_Icont1_mmr_sbh_Regs

SyncBox Handler (SBH) Memory Mapped Registers: These registers are used to control the SBH functionality.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ICONT_SBH_CTRL	SyncBox Handler control register
U32	ICONT_SBH_ATLR	Activation Task List register. It logs reference of the task interrupting CPU. When bitfield x of TX_0 field is 1, Task x has generated an interrupt to the CPU. Bitfield n of this register is cleared when writing 1 in bitfiled x of ICONT_SBH_ACK register.
U32	ICONT_SBH_ACK	Acknowledge Task register. Task x is automatically acknowledged on writing 1 into bitfield x of this register. Register is automatically cleared.
U32	ICONT_SBH_EOT	End Of Task register. SBH notify to the SB that Task x is ended on writing 1 to this register. Register is automatically cleared.
U8	RSVD0[16]	Reserved
U32	ICONT_SBH_IRQ_EOI	End of Interrupt number specification
U32	ICONT_SBH_IRQSTATUS_RAW_0	Per-event raw interrupt status vector, line #0 (end of transfer). Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	ICONT_SBH_IRQSTATUS_0	Per-event "enabled" interrupt status vector, line #0 (end of transfer). Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	ICONT_SBH_IRQENABLE_SET_0	Per-event interrupt enable bit vector, line #0 (end of transfer). Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	ICONT_SBH_IRQENABLE_CLR_0	Per-event interrupt enable bit vector, line #0 (end of transfer). Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	ICONT_SBH_IRQSTATUS_RAW_1	Per-event raw interrupt status vector, line #1 (end of transfer). Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	ICONT_SBH_IRQSTATUS_1	Per-event "enabled" interrupt status vector, line #1 (end of transfer). Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	ICONT_SBH_IRQENABLE_SET_1	Per-event interrupt enable bit vector, line #1 (end of

		transfer). Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	ICONT_SBH_IRQENABLE_CLR_1	Per-event interrupt enable bit vector, line #1 (end of transfer). Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U8	RSVD1[12]	Reserved
U32	ICONT_SBH_TCCR[6]	Task n Configuration Control Registers
U8	RSVD2[40]	Reserved
U32	ICONT_SBH_COUNTER[4]	Initialization value of SyncBox Handler Counter n
U8	RSVD3[16]	Reserved
U32	ICONT_SBH_DMLCH_CFG[4]	This register defines which VDMA group is associated to the Data Mover Logical Channel n.
U8	RSVD4[16]	Reserved
U32	ICONT_SBH_BYP_VDMAG_START	Syncbox Bypass mode VDMA group Start register. Register is cleared when ICONT_SBHCTRL.BYPASS bit is written to 1. Writing to this register is allowed only when ICONT_SBHCTRL.BYPASS bit is 1. Writing a '1' into bitfield n will START VDMA group n. This register is cleared when ICONT_SBHCTRL.BYPASS bit is written to 1.
U32	ICONT_SBH_BYP_VDMAG_END	Syncbox Bypass mode VDMA group End register status. This register is cleared when ICONT_SBHCTRL.BYPASS bit is written to 1. It is updated every time a VDMA group is completed, i.e. on VDMA End of Group detection, only if ICONT_SBHCTRL.BYPASS bit is 1.

2.3.22 CSL_Icont_irqreg_Regs

iCONTx IRQ Controller Registers: These registers are used to control the iCONTx IRQ Controller functionality.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ICONT_IRQ_EOI	End Of Interrupt register - Interrupt handler
U32	ICONT_IRQSTATUS_RAW	Per-event raw interrupt status vector (interrupt handler). Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	ICONT_IRQSTATUS	Per-event "enabled" interrupt status vector (interrupt handler). Enabled status isn't set unless event is enabled.

		Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	ICONT_IRQENABLE_SET	Per-event interrupt enable bit vector (interrupt handler). Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	ICONT_IRQENABLE_CLR	Per-event interrupt enable bit vector (interrupt handler). Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U8	RSVD0[12]	Reserved
U32	ICONT_SWI_EOI	End Of Interrupt register - Software interrupt
U32	ICONT_SWISTATUS_RAW	Per-event raw interrupt status vector (software interrupt). Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	ICONT_SWISTATUS	Per-event "enabled" interrupt status vector (software interrupt). Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	ICONT_SWIENABLE_SET	Per-event interrupt enable bit vector (software interrupt). Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	ICONT_SWIENABLE_CLR	Per-event interrupt enable bit vector (software interrupt). Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U8	RSVD1[12]	Reserved
U32	ICONT_SYNCMASK	This register allow to mask each input of Sync logic. When ICONT_SYNCMASK.SYN_IRQn is set, input event n is disabled When ICONT_SYNCMASK.SYNC_IRQn is cleared, input event n is enabled
U32	ICONT_SYNCCLR	Clear Interrupt line This register is used to clear status bit of interrupt n. write 0: no effect write 1: clears corresponding bit in IRQ_SYNCSTATUS and IRQ_SYNCRAWSTATUS registers
U32	ICONT_SYNCSTATUS	Status of masked Sync interrupt input n 0: Interrupt n has not occurred or masked 1: Interrupt n is enabled and has occurred
U8	RSVD2[4]	Reserved
U32	ICONT_SYNCRAWSTATUS	Status of Sync interrupt n input before mask 0: Interrupt n has not occurred 1: Interrupt n has occurred
U8	RSVD3[12]	Reserved

U32	ICONT_SYNC_IRQ_EOI	End of Interrupt - SYNC interrupt
U32	ICONT_SYNC_IRQSTATUS_RAW	Per-event raw interrupt status vector. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	ICONT_SYNC_IRQSTATUS	Per-event "enabled" interrupt status vector, line #0. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	ICONT_SYNC_IRQENABLE_SET	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	ICONT_SYNC_IRQENABLE_CLR	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.

2.3.23 CSL_Ilf3_Regs

iLF3 Registers: These registers are used to program the iLF3 IP.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	ILF_REVISION	IP Revision Identifier (X.Y.R) Used by software to track features, bugs, and compatibility
U8	RSVD0[12]	Reserved
U32	ILF_SYSCONFIG	
U8	RSVD1[4]	Reserved
U32	ILF_IRQ_EOI	
U32	ILF_IRQSTATUS_RAW_0	
U32	ILF_IRQSTATUS_0	
U32	ILF_IRQENABLE_SET_0	
U32	ILF_IRQENABLE_CLR_0	
U8	RSVD2[4]	Reserved
U32	ILF_CONFIG	
U32	ILF_STATUS	
U32	ILF_MBCONFIG_SLICEINFO01	
U32	ILF_MBCONFIG_SLICEINFO2	
U32	ILF_MBCONFIG_MBINFO_0	
U32	ILF_MBCONFIG_MBINFO_1	
U32	ILF_MBCONFIG_MBINFO_2	
U32	ILF_MBCONFIG_MB_0... ILF_MBCONFIG_MB_15	

U32	ILF_MBCONFIG_COEFFICIENTS0123	
U32	ILF_MBCONFIG_COEFFICIENTS4567	
U32	ILF_MBCONFIG_GDPCONFIG_0... ILF_MBCONFIG_GDPCONFIG_3	
U8	RSVD3[12]	Reserved
U32	ILF_MBCONFIG_AUTOINC	
U32	ILF_MBCONFIG_NEXTMBCONFIG	
U32	ILF_MBSTATUS	
U32	ILF_SLICESTATUS_0	
U32	ILF_SLICESTATUS_1	
U32	ILF_SLICESTATUS_2	
U8	RSVD4[24]	Reserved
U32	ILF_QP_0...ILF_QP_17	
U32	ILF_QP_IDX_0... ILF_QP_IDX_39	
U32	ILF_BS_0...ILF_BS_148	
U32	ILF_IPB_0... ILF_IPB_703	
U8	RSVD5[224]	Reserved
U32	ILF_COMMAND	

2.3.24 CSL_Ime3_Regs

iME3 Registers: These registers are used to program the iME3 IP.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	IME3_REVISION	IP Revision Identifier (X.Y.R) Used by software to track features, bugs, and compatibility
U8	RSVD0[12]	Reserved
U32	IME3_SYSCONFIG	Clock management configuration
U8	RSVD1[12]	Reserved
U32	IME3_IRQ_EOI	End Of Interrupt number specification
U32	IME3_IRQSTATUS_RAW	Per-event raw interrupt status vector, line #0. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.

U32	IME3_IRQSTATUS	Per-event "enabled" interrupt status vector, line #0. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	IME3_IRQENABLE_SET	Per-event interrupt enable bit vector, line #0. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	IME3_IRQENABLE_CLR	Per-event interrupt enable bit vector, line #0. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U8	RSVD2[12]	Reserved
U32	IME3_INTERPOL_PARAMETER_STACK_0	Interpolation Filter Pass Configuration
U32	IME3_INTERPOL_PARAMETER_STACK_1	Interpolation Filter Pass Configuration
U32	IME3_INTERPOL_PARAMETER_STACK_2	Interpolation Filter Pass Configuration
U32	IME3_INTERPOL_PARAMETER_STACK_3	Interpolation Filter Pass Configuration
U8	RSVD3[48]	Reserved
U32	IME3_PARAMETERSTACK_0... IME3_PARAMETERSTACK_31	Parameter Stack register 0 to 31 (32-bit wide). Contains parameters used by program to control the iME units.
U32	IME3_CURRENTBLOCK_0... IME3_CURRENTBLOCK_64	Current Block : 16 lines of 16 bytes containing the Current MacroBlock for SAD computation.
U32, U32	ERRTABLELSB_0, ERRTABLEMSB_0... ERRTABLELSB_31, ERRTABLEMSB_31	Error Table : Register File for SAD computation final errors. Each entry comprises a 16-bit error field, and 2 14-bit coordinate fields (dx and dy).
U32, U32	BMTABLELSB0_0, BMTABLEMSB0_0... BMTABLELSB0_15, BMTABLEMSB0_15	Best Match Table : register file contain result of Error Table comparisson. BestMatchTable0 is for List0 computation. BestMatchTable1 is for List1 computation. Each Best Match Table position

		correspond to one sub-partition of the 16x16 MB. For 1-MV type computation, best match is stored in entry 0 For 16-MV stored computation, each entry correspond to one of the 16 4x4 partitions. Otherwise, entries 0 to 8 are used.
U32, U32	BMTABLELSB1_0, BMTABLEMSB1_0... BMTABLELSB1_15, BMTABLEMSB1_15	Best Match Table : register file contain result of Error Table comparisson. BestMatchTable0 is for List0 computation. BestMatchTable1 is for List1 computation. Each Best Match Table position correspond to one sub-partition of the 16x16 MB. For 1-MV type computation, best match is stored in entry 0 For 16-MV stored computation, each entry correspond to one of the 16 4x4 partitions. Otherwise, entries 0 to 8 are used.
U32	MVCT0_3	MV Cost Table
U32	MVCT4_7	MV Cost Table
U32	MVCT8_11	MV Cost Table
U32	MVCT12_14	MV Cost Table
U32	VEC_VAR_HOR_LO	
U32	VEC_VAR_HOR_HI	
U32	VEC_VAR_VER_LO	
U32	VEC_VAR_VER_HI	
U32	IME3_VECABSMEANHOR	
U32	IME3_VECABSMEANVER	
U32	IME3_CIRCULAR_BUFFER_DESC0	
U32	IME3_CIRCULAR_BUFFER_DESC1	
U32	ILF_IPB_0... ILF_IPB_703	
U32	IME3_CPUSTATUSREG	CPU Status Register provides information on the progress of the CPU execution
U32	IME3_CYCLECOUNT	Cycle Counter
U8	RSVD4[8]	Reserved
U32	IME3_CONDITIONREGISTER	Absolute Minimum Reached bit register, used in Mcomp()

		operator.
U8	RSVD5[8]	
U32	IME3_MINERRORTHRESHOLD	Minimum Error Threshold register, used in Mcomp() operator.
U32	IME3_CIRCULAR_BUFFER_CURRENT_POSITION0	
U32	IME3_CIRCULAR_BUFFER_CURRENT_POSITION1	
U32	IME3_VALID_AREA0_TOP_LEFT_COORDINATES	
U32	IME3_VALID_AREA0_BOTTOM_RIGHT_COORDINATES	
U32	IME3_VALID_AREA1_TOP_LEFT_COORDINATES	
U32	IME3_VALID_AREA1_BOTTOM_RIGHT_COORDINATES	
U32	IME3_VECMEANHOR	
U32	IME3_VECMEANVER	
U32	IME3_INTERPOLATION_REFERENCE	The Interpol Reference is the MV based on which the last interpolation has been performed. This register is updated by the Interpol APIs, and is later used for computing the MVCost and addresses when manipulating pixels from the interpolated planes.
U8	RSVD6[7048]	Reserved
U32	IME3_COMMANDREG	iME3 command register: a write to this register decodes a command, a read returns 0. 0x1 -> Step() 0x2 -> StopSeq() 0x3 -> DbgEnable() 0x4 -> DbgDisable()
U32	IME3_PROGRAMBUFFER_0... IME3_PROGRAMBUFFER_1023	Program Memory 32-bit word

2.3.25 CSL_Ipe3_ipgw_Regs

IPE3 IPGW Registers: These registers are used to control the interrupts associated with IPE3.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	IPE3_SYSCONFIG	Clock management configuration
U32	IPE3_IRQ_EOI	End Of Interrupt number specification
U8	RSVD0[4]	Reserved
U32	IPE3_IPQSTATUS_RAW_0	Per-event raw interrupt status vector. Raw status is set even if event is not enabled.

		Write 1 to set the (raw) status, mostly for debug.
U32	IPE3_IPQSTATUS_RAW_1	Per-event raw interrupt status vector.
U32	IPE3_IPQSTATUS_RAW_2	Per-event raw interrupt status vector.
U32	IPE3_IPQSTATUS_RAW_3	Per-event raw interrupt status vector.
U8	RSVD1[12]	Reserved
U32	IPE3_IRQSTATUS_0	Per-event "enabled" interrupt status vector. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	IPE3_IRQSTATUS_1	Per-event "enabled" interrupt status vector.
U32	IPE3_IRQSTATUS_2	Per-event "enabled" interrupt status vector.
U32	IPE3_IRQSTATUS_3	Per-event "enabled" interrupt status vector.
U8	RSVD2[12]	Reserved
U32	IPE3_IRQENABLE_SET_0	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	IPE3_IRQENABLE_SET_1	Per-event interrupt enable bit vector.
U32	IPE3_IRQENABLE_SET_2	Per-event interrupt enable bit vector.
U32	IPE3_IRQENABLE_SET_3	Per-event interrupt enable bit vector.
U8	RSVD3[12]	Reserved
U32	IPE3_IRQENABLE_CLR_0	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	IPE3_IRQENABLE_CLR_1	Per-event interrupt enable bit vector.
U32	IPE3_IRQENABLE_CLR_2	Per-event interrupt enable bit vector.
U32	IPE3_IRQENABLE_CLR_3	Per-event interrupt enable bit vector.
U8	RSVD4[72]	Reserved
U32	IPE3_IRQSTATUS_ACLREN	Auto Clear Enable

2.3.26 CSL_Ipe3_bfsw_Regs

IPE3 Buffer Switch (BFSW) Module Registers: These registers are used to control the buffer switching in ping-pong mode and to switch masters (HWA / DMA) for the buffers.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	VIEWMODE	View Mode Register. It selects full-view mode or ping-pong view mode.
U32	MSTID	Master ID Register Select master between HWA and DMA bus. This register is used in both full view and ping-pong view mode. This register is for buffers which has two physical memories.

2.3.27 CSL_Ipe3_mmr_Regs

IPE3 Memory Mapped Registers: These registers are used to control the IPE3 core.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	IPE_PID	Peripheral ID Register
U32	IPE_COUNT	IPE cycle counter register
U32	IPE_CTRL	IPE control register
U32	IPE_NS	Horizontal noise suppression register. Do not write any value during IPE is running (during IPE_EN is '1').
U32	IPE_NA	nA mode register Do not write any value during IPE is running (during IPE_EN is '1')
U32	IPE_L_LF_T0	Luma left neighboring data IPE3 read this register value as luma left samples at the beginning of estimation. Also right-most current pixels are written at the ending of estimation to use data as left sample of next macroblock. Do not write any value during IPE is running (during IPE_EN is '1')
U32	IPE_L_LF_T1	Luma left neighboring data
U32	IPE_L_LF_T2	Luma left neighboring data
U32	IPE_L_LF_T3	Luma left neighboring data
U32	IPE_L_LF_B0	Luma left neighboring data IPE3 read this register value as luma left samples at the beginning of estimation. Also right-most current pixels are written at the ending of estimation to use data as left sample of next macroblock. Do not write any value during IPE is running (during IPE_EN is '1')
U32	IPE_L_LF_B1	Luma left neighboring data
U32	IPE_L_LF_B2	Luma left neighboring data
U32	IPE_L_LF_B3	Luma left neighboring data
U32	IPE_C_LF_T0	Chroma left neighboring data IPE3 read this register value as chroma left samples at the beginning of estimation. Also right-most current pixels are written at the ending of estimation to use data as left sample of next macroblock. Do not write any value during IPE is running (during IPE_EN is '1')
U32	IPE_C_LF_T1	Chroma left neighboring data
U32	IPE_C_LF_T2	Chroma left neighboring data
U32	IPE_C_LF_T3	Chroma left neighboring data
U32	IPE_C_LF_B0	Chroma left neighboring data IPE3 read this register value as chroma left samples at the beginning of estimation. Also right-most current pixels are written at the ending of estimation to use data as left sample of next macroblock.

		Do not write any value during IPE is running (during IPE_EN is '1')
U32	IPE_C_LF_B1	Chroma left neighboring data
U32	IPE_C_LF_B2	Chroma left neighboring data
U32	IPE_C_LF_B3	Chroma left neighboring data

2.3.28 CSL_Ipe3_mem_Regs

IPE3 Internal Memory: This structure defines the buffers used by the IPE3 core within the IPE3 internal memory.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	IPORGBUF_A_START	IP Original Buffer A Buffer Start
U8	RSVD0[2040]	Buffer Space
U32	IPORGBUF_A_END	IP Original Buffer A Buffer End
U32	IPORGBUF_B_START	IP Original Buffer B Buffer Start
U8	RSVD1[2040]	Buffer Space
U32	IPORGBUF_B_END	IP Original Buffer B Buffer End

2.3.29 CSL_Mc3_ipgw_Regs

MC3 IPGW Registers: These registers are used to control the interrupts associated with MC3.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	MC3_SYSCONFIG	Clock management configuration
U32	MC3_IRQ_EOI	End Of Interrupt number specification
U32	MC3_IRQ_MERGED_STATUS	
U32	MC3_IPQSTATUS_RAW_0	Per-event raw interrupt status vector. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	MC3_IPQSTATUS_RAW_1	Per-event raw interrupt status vector. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	MC3_IPQSTATUS_RAW_2	Per-event raw interrupt status vector. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	MC3_IPQSTATUS_RAW_3	Per-event raw interrupt status vector. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U8	RSVD0[12]	Reserved
U32	MC3_IRQSTATUS_0	Per-event "enabled" interrupt status vector. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).

U32	MC3_IRQSTATUS_1	Per-event "enabled" interrupt status vector. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	MC3_IRQSTATUS_2	Per-event "enabled" interrupt status vector. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	MC3_IRQSTATUS_3	Per-event "enabled" interrupt status vector. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U8	RSVD1[12]	Reserved
U32	MC3_IRQENABLE_SET_0	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	MC3_IRQENABLE_SET_1	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	MC3_IRQENABLE_SET_2	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	MC3_IRQENABLE_SET_3	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U8	RSVD2[12]	Reserved
U32	MC3_IRQENABLE_CLR_0	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	MC3_IRQENABLE_CLR_1	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	MC3_IRQENABLE_CLR_2	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	MC3_IRQENABLE_CLR_3	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U8	RSVD3[72]	Reserved
U32	MC3_IRQSTATUS_ACLREN	Auto Clear Enable

2.3.30 CSL_Mc3_bfsw_Regs

MC3 Buffer Switch (BFSW) Module Registers: These registers are used to control the buffer switching in ping-pong mode and to switch masters (HWA / DMA) for the buffers.

DATA TYPE	REGISTER FIELD	DESCRIPTION

U32	VIEWMODE	View Mode Register. It selects full-view mode or ping-pong view mode.
U32	MSTID1	Master ID 1 Register Select master between HWA and DMA bus. This register is used in both full view and ping-pong view mode. This register is for buffers which has two physical memories.
U32	MSTID2	Master ID 2 Register Select master between HWA and DMA bus. This register is for buffers which has only one physical memory.

2.3.31 CSL_Mc3_mmr_Regs

MC3 Memory Mapped Registers: These registers are used to control the MC3 core.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	MC_PID	Peripheral ID register
U32	MC_CNT	Cycle Counter Register
U32	MC_CTRL	MC3 Control Register
U32	MC_PARAM0	MC Parameters Register
U32	MC_PARAM1	MC3 Parameters Register
U32	MC_PARAM2	MC3 Parameters Register
U32	MC_ADDR_0	Base Address of reference data Y L0 top and bottom
U32	MC_ADDR_1	Base Address of reference data Y L1 top and bottom
U32	MC_ADDR_2	Base Address of reference data C L0 top and bottom
U32	MC_ADDR_3	Base Address of reference data C L1 top and bottom

2.3.32 CSL_Mc3_mem_Regs

MC3 Internal Memory: This structure defines the buffers used by the MC3 core within the MC3 internal memory.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	MCBUF_A_START	MC Buffer A Buffer Start
U8	RSVD0[10232]	Buffer Space
U32	MCBUF_A_END	MC Buffer A Buffer End
U32	MCBUF_B_START	MC Buffer B Buffer Start
U8	RSVD1[10232]	Buffer Space
U32	MCBUF_B_END	MC Buffer B Buffer End
U32	IPRDBUF_A_START	Inter Prediction Buffer A Buffer Start
U8	RSVD2[1016]	Buffer Space
U32	IPRDBUF_A_END	Inter Prediction Buffer A Buffer End

U32	IPRDBUF_B_START	Inter Prediction Buffer B Buffer Start
U8	RSVD3[1016]	Buffer Space
U32	IPRDBUF_B_END	Inter Prediction Buffer B Buffer End

2.3.33 CSL_Lse_Regs

LSE Registers: These registers are used to program the LSE. The LSE module is instantiated in ECD3, CALC3, MC3 and IPE3.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	LSE_CTRL	LSE Control Register
U32	LSE_PARAM	Parameter address for SB Bypass mode

2.3.34 CSL_Mailbox_Regs

Mailbox Registers: These registers are associated with the Mailbox module. The mailbox is used for inter-processor communication (IPC) (eg., between Host and IVA-HD).

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	MAILBOX_REVISION	
U8	RSVD0[12]	Reserved
U32	MC3_SYSCONFIG	
U32	MAILBOX_SYSSTATUS	
U8	RSVD1[40]	Reserved
U32	MAILBOX_MESSAGE[6]	
U8	RSVD2[40]	Reserved
U32	MAILBOX_FIFOSTATUS[6]	
U32	MAILBOX_MSGSTATUS[6]	
U8	RSVD3[40]	Reserved
U32	MAILBOX_IRQSTATUS_RAW_0	
U32	MAILBOX_IRQSTATUS_CLR_0	
U32	MAILBOX_IRQENABLE_SET_0	
U32	MAILBOX_IRQENABLE_CLR_0	
U32	MAILBOX_IRQSTATUS_RAW_1	
U32	MAILBOX_IRQSTATUS_CLR_1	
U32	MAILBOX_IRQENABLE_SET_1	
U32	MAILBOX_IRQENABLE_CLR_1	
U32	MAILBOX_IRQSTATUS_RAW_2	
U32	MAILBOX_IRQSTATUS_CLR_2	
U32	MAILBOX_IRQENABLE_SET_2	

U32	MAILBOX_IRQENABLE_CLR_2	
U32	MAILBOX_IRQSTATUS_RAW_3	
U32	MAILBOX_IRQSTATUS_CLR_3	
U32	MAILBOX_IRQENABLE_SET_3	
U32	MAILBOX_IRQENABLE_CLR_3	
U32	MAILBOX_IRQ_EOI	

2.3.35 CSL_Sl2if_cfg_Regs

SL2 Interface Registers: These registers are used to configure the SL2 interface.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	SL2IF_REVISION	
U32	SL2IF_HWINFO	
U8	RSVD0[8]	Reserved
U32	SL2IF_SYSCONFIG	
U8	RSVD1[236]	Reserved
U32	SL2IF_CR_0	
U32	SL2IF_CR_1	
U32	SL2IF_CR_2	
U32	SL2IF_CR_3	
U32	SL2IF_CR_4	
U32	SL2IF_CR_5	
U32	SL2IF_CR_6	
U32	SL2IF_CR_7	
U32	SL2IF_CR_8	
U32	SL2IF_CR_9	
U32	SL2IF_CR_10	
U32	SL2IF_CR_11	
U32	SL2IF_CR_12	
U32	SL2IF_CR_13	
U32	SL2IF_CR_14	
U32	SL2IF_CR_15	
U32	SL2IF_CR_16	

2.3.36 CSL_Conf_Regs

SMSET Registers: The Software Message and System Event Trace module, referenced as SMSET, is a trace module used for monitoring key system events and transferring software messages.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	SMSET_ID	
U8	RSVD0[12]	Reserved
U32	SMSET_SCFG	
U32	SMSET_SR	
U8	RSVD1[12]	Reserved
U32	SMSET_CFG	
U32	SMSET_SESW	
U32	RSVD2[4]	Reserved
U32	SMSET_SEDEN1	
U32	SMSET_SEDEN2	
U32	SMSET_SEDEN3	
U32	SMSET_SEDEN4	
U32	SMSET_SEDEN5	
U32	SMSET_SEDEN6	
U32	SMSET_SEDEN7	
U32	SMSET_SEDEN8	

2.3.37 CSL_Icont1_sb_syncbox_Regs

SyncBox Registers: These registers are used to program the SyncBox. The SyncBox module is instantiated in all accelerators and iCONTs.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	SYNCBOX_REVISION	Contains the revision number on 32 bit compliant Higlander
U8	RSVD0[12]	Reserved
U32	SYSCONFIG	This register allows controlling various parameters of the OCP interface
U8	RSVD1[44]	Reserved
U32	RXMESSAGE	Register containing the received message, from the CTL_IN interface or MSG_IN interface
U32	NODEIDENTIFIER	Contains the node identifier. Initialized by default to the tie-off value
U32	ERRORMSGDEST	register containing the node and task identifier to send an

		activation message when an error is detected
U32	ERRORLOG	contains the different errors latched during message decoding
U8	RSVD2[8]	Reserved
U32	TASKSYNCTYPE	This register is used to code the task synchronous type: bit = 0: synchronous task bit = 1: asynchronous task
U8	RSVD3[164]	Reserved
U32	REMOTESYNCACTMSGREG1_0	contains the task and node identifier to sent the activation message to, upon detection of an asynchronous event
U32	REMOTESYNCACTMSGREG2_0	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG3_0	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG4_0	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	ACTIVATIONMASK_0	For synchronous task description of bitfields given below applies. For asynchronous task, only the 16 lsbs are used. Each of the 16 lsb is corresponding to a 1hot encoding value for the node identifier. 16 MSBs write access is protected by the corresponding bit in the TasksyncType register.
U32	ACTIVATIONCONTROL_0	contains the 4 activation counters
U32	NEWTASKCOUNTER_0	
U8	RSVD4[100]	Reserved
U32	REMOTESYNCACTMSGREG1_1	contains the task and node identifier to sent the activation message to, upon detection of an asynchronous event
U32	REMOTESYNCACTMSGREG2_1	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG3_1	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG4_1	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	ACTIVATIONMASK_1	For synchronous task description of bitfields given below applies.

		For asynchronous task, only the 16 lsbs are used. Each of the 16 lsb is corresponding to a 1hot encoding value for the node identifier. 16 MSBs write access is protected by the corresponding bit in the TasksyncType register.
U32	ACTIVATIONCONTROL_1	contains the 4 activation counters
U32	NEWTASKCOUNTER_1	
U8	RSVD5[100]	Reserved
U32	REMOTESYNCACTMSGREG1_2	contains the task and node identifier to sent the activation message to, upon detection of an asynchronous event
U32	REMOTESYNCACTMSGREG2_2	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG3_2	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG4_2	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	ACTIVATIONMASK_2	For synchronous task description of bitfields given below applies. For asynchronous task, only the 16 lsbs are used. Each of the 16 lsb is corresponding to a 1hot encoding value for the node identifier. 16 MSBs write access is protected by the corresponding bit in the TasksyncType register.
U32	ACTIVATIONCONTROL_2	contains the 4 activation counters
U32	NEWTASKCOUNTER_2	
U8	RSVD6[100]	Reserved
U32	REMOTESYNCACTMSGREG1_3	contains the task and node identifier to sent the activation message to, upon detection of an asynchronous event
U32	REMOTESYNCACTMSGREG2_3	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG3_3	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG4_3	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	ACTIVATIONMASK_3	For synchronous task description of bitfields given below applies. For asynchronous task, only the 16 lsbs are used.

		Each of the 16 lsb is corresponding to a 1hot encoding value for the node identifier. 16 MSBs write access is protected by the corresponding bit in the TasksyncType register.
U32	ACTIVATIONCONTROL_3	contains the 4 activation counters
U32	NEWTASKCOUNTER_3	
U8	RSVD7[100]	Reserved
U32	REMOTESYNCACTMSGREG1_4	contains the task and node identifier to sent the activation message to, upon detection of an asynchronous event
U32	REMOTESYNCACTMSGREG2_4	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG3_4	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG4_4	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	ACTIVATIONMASK_4	For synchronous task description of bitfields given below applies. For asynchronous task, only the 16 lsbs are used. Each of the 16 lsb is corresponding to a 1hot encoding value for the node identifier. 16 MSBs write access is protected by the corresponding bit in the TasksyncType register.
U32	ACTIVATIONCONTROL_4	contains the 4 activation counters
U32	NEWTASKCOUNTER_4	
U8	RSVD8[100]	Reserved
U32	REMOTESYNCACTMSGREG1_5	contains the task and node identifier to sent the activation message to, upon detection of an asynchronous event
U32	REMOTESYNCACTMSGREG2_5	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG3_5	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	REMOTESYNCACTMSGREG4_5	Contains destination task identifier and destination node identifier for activation messages to be sent upon task completion. Valid bit is used to loop on all registers of the bundle
U32	ACTIVATIONMASK_5	For synchronous task description of bitfields given below applies. For asynchronous task, only the 16 lsbs are used. Each of the 16 lsb is corresponding to a 1hot encoding value for

		the node identifier. 16 MSBs write access is protected by the corresponding bit in the TasksyncType register.
U32	ACTIVATIONCONTROL_5	contains the 4 activation counters
U32	NEWTASKCOUNTER_5	

2.3.38 CSL_Sysctrl_Regs

IVA-HD System Control Registers: These registers are used to contro the system clock and IRQ.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	IVAHD_REVISION	IVA-HD Revision Identifier (X.Y.R) Used by software to track features, bugs, and compatibility
U32	IVAHD_HWINFO	Information about the IP module's hardware configuration.
U8	RSVD0[8]	Reserved
U32	IVAHD_SYSCONFIG	Clock management configuration
U8	RSVD1[12]	Reserved
U32	IVAHD_IRQ_EOI	End Of Interrupt number specification
U32	IVAHD_IRQSTATUS_RAW	Per-event raw interrupt status vector. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	IVAHD_IRQSTATUS	Per-event "enabled" interrupt status vector, line #0. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	IVAHD_IRQENABLE_SET	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	IVAHD_IRQENABLE_CLR	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	IVAHD_SYNC_IRQSTATUS_RAW	Per-event raw interrupt status vector. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	IVAHD_SYNC_IRQSTATUS	Per-event "enabled" interrupt status vector, line #0. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	IVAHD_SYNC_IRQENABLE_SET	Per-event interrupt enable bit vector. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.

U32	IVAHD_SYNC_IRQENABLE_CLR	Per-event interrupt enable bit vector. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U8	RSVD2[12]	Reserved
U32	IVAHD_CLKCTRL	IVAHD clock control register
U32	IVAHD_CLKST	IVA-HD clock status register
U32	IVAHD_STDBYST	IVA-HD STANDBY status

2.3.39 CSL_Vdma_Regs

VDMA Registers: These registers are used to program the VDMA module. The VDMA is used for data transfer from/to external memory to/from IVA-HD SL2 memory.

DATA TYPE	REGISTER FIELD	DESCRIPTION
U32	VDMA_REVISION	IP Revision Identifier This allows a PID showing X.Y.R in silicon to relate the RTL release with a (close-to-correct) spec version X.Y.S. A peripheral ID register must be included at address offset 0 of a peripherals control register MAP. The purpose is to let software read the peripheral to understand what type of peripheral is there and what features are enabled as well as what bugs or issues may exist in a particular version.
U8	RSVD0[12]	Reserved
U32	VDMA_SYSCONFIG	Clock management configuration
U8	RSVD1[12]	Reserved
U32	VDMA_IRQ_EOI	End Of Interrupt number specification
U32	VDMA_IRQSTATUS_RAW_0	Per-event raw interrupt status vector, line #0. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	VDMA_IRQSTATUS_0	Per-event "enabled" interrupt status vector, line #0. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	VDMA_IRQENABLE_SET_0	Per-event interrupt enable bit vector, line #0. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	VDMA_IRQENABLE_CLR_0	Per-event interrupt enable bit vector, line #0. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	VDMA_IRQSTATUS_RAW_1	Per-event raw interrupt status vector, line #1. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.

U32	VDMA_IRQSTATUS_1	Per-event "enabled" interrupt status vector, line #1. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	VDMA_IRQENABLE_SET_1	Per-event interrupt enable bit vector, line #1. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	VDMA_IRQENABLE_CLR_1	Per-event interrupt enable bit vector, line #1. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	VDMA_IRQSTATUS_RAW_2	Per-event raw interrupt status vector, line #2. Raw status is set even if event is not enabled. Write 1 to set the (raw) status, mostly for debug.
U32	VDMA_IRQSTATUS_2	Per-event "enabled" interrupt status vector, line #2. Enabled status isn't set unless event is enabled. Write 1 to clear the status after interrupt has been serviced (raw status gets cleared, i.e. even if not enabled).
U32	VDMA_IRQENABLE_SET_2	Per-event interrupt enable bit vector, line #2. Write 1 to set (enable interrupt). Readout equal to corresponding _CLR register.
U32	VDMA_IRQENABLE_CLR_2	Per-event interrupt enable bit vector, line #2. Write 1 to clear (disable interrupt). Readout equal to corresponding _SET register.
U32	VDMA_SYNCHR_LIST_LEVEL	
U32	VDMA_ASYNCHR_LIST_LEVEL	
U32	VDMA_NON_DETERM_FIFO_LEVEL	
U32	VDMA_TBA	Tiler address mapping.
U32	VDMA_CONTEXT_STATUS	When individual bit is reset, corresponding context is available When individual bit is set, corresponding context is allocated.
U32	VDMA_GROUP_TRIGGER	Register entry for SW user to trigger groups through CPU writes. write "1" to desired bit triggers corresponding group. write "0" has no effect.
U32	VDMA_MAX_CONTEXT_SYNCHR	SW user configurable maximum number of context synchronous list can get benefit of.
U32	VDMA_MAX_CONTEXT_ASYNCHR	SW user configurable maximum number of context asynchronous list can get benefit of.
U32	VDMA_IRQ_NEOG	
U32	VDMA_GROUP_STATUS[32]	
U32	VDMA_GROUP_DEFINITION[32]	Group_definition register set is the SW user entry to define groups mapping and routing into and through event engine: -throw group into synchronous or asynchronous list

		-pick descriptors from non deterministic or deterministic memory -start address of descriptors when belonging to a deterministic group.
U32	VDMA_NON_DETERM[4]	Each descriptor entry is made of four 32b words. One or two non_determ_descr_entries (each made of of four 32b accesses) are required to complete one non deterministic object description.
U8	RSVD2[1656]	Reserved
U32	VDMA_SRC_DEST_COUNTER[128]	
U8	RSVD3[1536]	Reserved
U32	VDMA_DETERM[128][8]	Each deterministic descriptor entry is made of eight 32b words.

3. IVA-HD API Data types

This chapter describes data types and date structures used by the IVA-HD APIs for relevant hardware accelerators.

3.1 Common Data types

This section describes data types commonly used in APIs for more than one accelerator programming in a specific video codec implementation.

3.1.1 eHWANode

Description

This enum enumerates the node IDs for HWAs. HWA Node numbers are same as the ones mentioned in Sync Box Specification/TRM

Members

Member	Value	Description
NODE_ILF3	2	Node ID for iLF3 HWA
NODE_IME3	3	Node ID for iME3 HWA
NODE_CALC3	4	Node ID for CALC3 HWA
NODE_IPE3	5	Node ID for IPE3 HWA
NODE_MC3	6	Node ID for MC3 HWA
NODE_ECD3	7	Node ID for ECD3 HWA

3.1.2 eCPU_ID

Description

This enum enumerates the iCONTs.

Members

Member	Value	Description
iCONT1	0	iCONT1 CPU
iCONT2	1	iCONT2 CPU

3.1.3 tHdvicp20MPEG4DFrameInfo

Description
This structure defines variables to store the frame level parameters parsed out from MPEG-4 bit stream. This structure is passed as argument to all the MPEG-4 decoder specific APIs to configure the IVA-HD hardware accelerators (CALC3 & ECD3) for decoding process.

Fields

Field	Data type	Input / output	Range	Description
Short_video_header	S32	Input	0,1	Internal flag set to 1 for

				H.263.
quant_type	S32	Input	0,1	Internal flag set to 1 for MPEG4.
h263annexes	U16	Input	0,1	Internal flag set to 1 for H.263. When this bit is 1 H263 Annex I configuration is enabled.
sorenson_spark_stream	U8	Input	0,1	When the codec type is H.263 and this flag is 1, it becomes sorenson spark.
heightInMBs	S32	Input	[0, 16383]	Current picture height in macroblocks
widthInMBs	S32	Input	[0, 16383]	Current picture width in macroblocks
MBAlength	U16	Input	[0, 2^16-1]	Parameter used to get the information of MBA
quant_scale	U8	Input	[0, 32]	Sets the quant type. It is effective only for MPEG4.
intra_dc_vlc_thr	S32	Input	[0, 4]	Specifies a threshold value of quantizer scale used to switch between two VLC's for coding of Intra DC coefficients as per the standard.
vop_fcode_forward	S32	Input	[0, 4]	Indicates the range of forward motion vectors used for the frame.
vop_fcode_backward	S32	Input	0, 1	Indicates the range of backward motion vectors used for the frame.
vop_rounding_type	S32	Input	0,1	Signals the value of the parameter rounding_control used for pixel value interpolation in motion compensation for P- and S(GMC)- VOPs .
vop_time_increment	S32	Input	[0, 2^31-1]	Represents the absolute vop_time_increment from the synchronization point marked by the modulo_time_base measured in the number of clock ticks.
bitlen_of_vop_time_increment	U16	Input	[0, 2^16-1]	Bit length of vop_time_increment.
curr_modulo_time_base	U32	Input	[0 – 2^32-1]	Represents the local time base in one second resolution units.

MBA	S32	Input	[0, 2^31-1]	Gives the macroblock count in the picture.
MBx	S16	Input	[0, 2^15-1]	Macroblock number in x direction.
MBy	S16	Input	[0, 2^15-1]	Macroblock number in y direction.
Bitcnt	U64	Input	[0 – 2^64-1]	The next bit position in the byte at BYTEPTR_A

3.1.4 tHdvicp20VC1DDecState

Description

This structure defines the parameters that are common to all the MBs of a frame/slice in a VC-1 bit stream. This structure is passed as argument to all VC-1 decoder specific APIs to configure IVA-HD accelerators for VC-1 decoding process.

Fields

Field	Data Type	Input / output	Range	Description
ucMc3RefDtRingBufSize	pU8	Input	Valid Address Range	MC3 Reference Data Ring buffer size.
pMc3RefDataOffset	pU8	Input	Valid Address Range	MC3 Reference data offset.
pStreamBuffer	U32	Input	Valid Address Range	Pointer to the stream buffer in SL2.
NotFirstMode3InFrame	U32	Input	0,1	This flag is 1 if not first mode 3 escape in frame.
LevelCodeSize	U32	Input	[0 – 2^32-1]	Level code size for mode 3 escape, per frame.
RunCodeSize	U32	Input	[0 – 2^32-1]	Run code size for mode 3 escape, per frame.
RndCtrl	U32	Input	0,1	Rounding control for the frame.
BitsUsed	U32	Input	[0 – 2^32-1]	Number of used bits in the buffer.

WidthMB	U16	Input	0-16383	Width in macroblocks of coded picture.
HeightMB	U16	Input	0-16383	Height in macroblocks of coded picture.
ZeroRun	U32	Input	[0 – 2^32-1]	Number of consecutive zeroes read from the bitstream.
uiBytesConsumed	U32	Input	[0 – 2^32-1]	Number of bytes consumed till now from starting of the PAGE.
frame_type	vc1_eFrameTypes	Input	[0, 4]	vc1_eFrameTypes is a enum whose elements are IFRAME, PFRAME, BFRAME, BIFRAME, P_SKIPPEDFRAME.
ePictureType	vc1_ePictureType	Input	[0, 4]	vc1_ePictureType is a enum whose elements are vc1_PictureTypeI, vc1_PictureTypeP, vc1_PictureTypeB, vc1_PictureTypeBI, vc1_PictureTypeSkipped.
eProfile	vc1_eProfile	Input	[0, 3]	vc1_eProfile is a enum whose elements are vc1_ProfileSimple, vc1_ProfileMain, vc1_ProfileReserved, vc1_ProfileAdvanced
ePictureFormat	vc1_ePictureFormat	Input	[0, 3]	vc1_ePictureFormat is a enum whose elements are vc1_ProgressiveFrame, vc1_InterlacedFrame, vc1_InterlacedField, vc1_PictureFormatNone
eQuantizer	vc1_eQuantizer	Input	[0, 3]	vc1_eQuantizer is a enum whose elements are vc1_QuantizerImplicit, vc1_QuantizerExplicit, vc1_QuantizerNonUniform, vc1_QuantizerUniform
eConditionalOverlap	vc1_eCondOver	Input	[0, 2]	vc1_eCondOver is a enum whose elements are vc1_CondOverNone, vc1_CondOverAll, vc1_CondOverSome
eMVMode	vc1_eMVMode	Input	[0, 4]	vc1_eMVMode is a enum whose elements are

				vc1_MVMode1MVHalfPelBilinear, vc1_MVMode1MVHalfPel, vc1_MVMode1MV, vc1_MVModeMixedMV, vc1_MVModeIntensityCompensation

3.1.5 tHdvicp20MPEG2DFrameInfo

Description
This structure defines the parameters that are common to all the MBs of a frame in a MPEG-2 bit stream. This structure is passed as argument to all MPEG-2 decoder specific APIs to configure IVA-HD accelerators for MPEG-2 decoding process.

Fields

Field	Data Type	Input / output	Range	Description
pictureStructure	U8	Input	0,1	Specifies the picture structure. 0-Frame 1-Field
MPEG1_Flag	U8	Input	0,1	Has a value of 1 if the codec type is MPEG1.
widthInMBs	U32	Input	[0, 16383]	Picture width in macroblocks.
heightInMBs	U32	Input	[0, 16383]	Picture height in macroblocks.
q_scale_type	U32	Input	0,1	Specifies quant scale type. Used only in MPEG2.
intra_dc_precision	U32	Input	0,1	Parameter used by the command wrapper.
Bitcnt	U64	Input	[0 – 2^64-1]	The next bit position in the byte at BYTEPTR_A.
Width	S32	Input	[0 – 2^31-1]	Width of the picture in pixels.

3.1.6 tHdvicp20mc3MPEG2dDynamicStruct

Description
This structure defines the parameters that are common to all the MBs of a slice.

Fields

Field	Data Type	Input / output	Range	Description
pubSrc[8]	pU8	Input	Valid Address Range	Variable to hold the source address.

pubDst[8]	pU8	Input	Valid Address Range	Variable to hold the destination address.
uiXOffset[8]	U32	Input	Valid Address Range	DDR Offset in the X-direction where the 2D object is placed.
uiYOffset[8]	U32	Input	Valid Address Range	DDR Offset in the Y-direction where the 2D object is placed.
uiWidth[8]	U32	Input	[0 – 2^32-1]	Variable to hold the picture width in Mbs.
uiHeight[8]	U32	Input	[0 – 2^32-1]	Variable to hold the picture height in Mbs.
uiSrcPitch[8]	U32	Input	[0 – 2^32-1]	Variable to hold the source pitch.
uiDstPitch[8]	U32	Input	[0 – 2^32-1]	Variable to hold the destination pitch.

3.2 Data types for ECD3

This section describes data types used in APIs for ECD3 programming in different video codec implementation.

3.2.1 tHdvicp20ecd3H264dFrameinit

Description
This Structure defines the parameters that are required for initializing the ECD3 module for H.264 Decoder

Fields

Field	Data type	Input / output	Range	Description
num_slice_groups_minus1	U16	Input	NA	Total number of Slice groups per picture decremented by one
ASO_in_use	U8	Input	0,1	Flag which states if ASO is in use, Range(1/0)
mb_width_slicePars	U16	Input	1 - 120	Frame width in terms of number of macroblocks
mb_height_slicePars	U16	Input	1-68	Frame height in terms of number of macroblocks
mb_aff_frame_flag	U8	Input	0,1	Flag which states if MBAFF mode is in use, Range(1/0) 0 – Non-Mbaff Picture 1 - Mabaff Picture
buf_ptr	S16	Input	Valid Address Range	Pointer to Stream Buffer which is stored in SL2 memory

first_mb_in_slice	U16	Input	0-MAX_PIC_SIZE_IN_MBS-1	Macro block number of the First macro block in the slice. Range: 0-MAX_PIC_SIZE_IN_MBS-1
mb_width_codecParams	U16	Input	1 - 120	Frame width in terms of number macro blocks
bitsLeft	U32	Input	NA	Bits left in the current word
mb_height_codecParams	U16	Input	1-68	Frame height in terms of number number of macro blocks
Pecdabuf	U32	Input	Valid Address Range	Base address of ECDABUF
ptopMb	U32	Input	Valid Address Range	Address of Upper macro block in ECDABUF
ptopLeftMb	U32	Input	Valid Address Range	Address of Upper Left macro block in ECDABUF
pcurrentMb_abuf	U32	Input	Valid Address Range	Address of Current macro block in ECDABUF
pupperRightMb_abuf	U32	Input	Valid Address Range	Address of Upper Right macro block in ECDABUF
pcoLocatedMbA	U32	Input	Valid Address Range	Address of Co located macro block A in ECDABUF
pcoLocatedMbB	U32	Input	Valid Address Range	Address of Co located macro block B in ECDABUF
Pcommands	U32	Input	Valid Address Range	Pointer to ECD commands
Pecdwbuf	U32	Input	Valid Address Range	Base address of ECDWBUF
pupperMb	U32	Input	Valid Address Range	Address of Upper macro block in ECDWBUF

pleftMb	U32	Input	Valid Address Range	Address of Left macro block in ECDWBUF
pcurrentMb_wbuf	U32	Input	Valid Address Range	Address of Current macro block in ECDWBUF
pupperLeftMb	U32	Input	Valid Address Range	Address of Upper Left macro block in ECDWBUF
pupperRightMb_wbuf	U32	Input	Valid Address Range	Address of Upper Right macro block in ECDWBUF
psliceInfo	U32	Input	Valid Address Range	Address of Slice Info in ECDWBUF
pcurrentMvd	U32	Input	Valid Address Range	Address of MVD for current macro block in ECDWBUF
pbottomCurrentMbaffMvd	U32	Input	Valid Address Range	Address of bottom MB of current macro block pair in MBAFF mode
pleftMvd	U32	Input	Valid Address Range	Address of MVD for Left macro block in ECDWBUF
pbottomLeftMbaffMvd	U32	Input	Valid Address Range	Address of bottom macro block of Left macro block pair in MBAFF mode
Prsdbuf	U32	Input	Valid Address Range	Base address of Residue Buffer
Presidual	U32	Input	Valid Address Range	Address of residual data in Residue Buffer
size_MbInfoH264	U32	Input	NA	Data element size for Current macro block Info
sizeof_ResidualH264	U32	Input	NA	Residual data size

3.2.2 tHdvicp20ecd3H264eFrameinit

Description
This Structure defines the parameters that are required for initializing the ECD3 module for H.264 Encoder.

Fields

Field	Data type	Input / output	Range	Description
pecdabuf	U32	Input	Valid Address Range	Base address of ECDABUF
pcurrentMb_abuf	U32	Input	Valid Address Range	Address of Current MB in ECDABUF
pcurrentMb_wbuf	U32	Input	Valid Address Range	Address of Current MB in ECDWBUF
pupperRightMb_abuf	U32	Input	Valid Address Range	Address of Upper Right MB in ECDABUF
pupperRightMb_wbuf	U32	Input	Valid Address Range	Address of Upper Right MB in ECDWBUF
pcoLocatedMbA	U32	Input	Valid Address Range	Address of Colocated MB in ECDABUF
commands	U32 *	Input	Valid Address Range	Pointer to ECD commands
prsdbuf	U32	Input	Valid Address Range	Base address of Residue Buffer
presidual	U32	Input	Valid Address Range	Address of residual data in Residue Buffer
pecdwbuf	U32	Input	Valid Address Range	Base address of ECDWBUF
pupperLeftMb	U32	Input	Valid Address Range	Address of Upper Left MB in ECDWBUF
pupperMb	U32	Input	Valid Address Range	Address of Upper MB in ECDWBUF
pleftMb	U32	Input	Valid Address Range	Address of Left MB in ECDWBUF
psliceInfo	U32	Input	Valid Address Range	Address of Slice Info in ECDWBUF

pcurrentMvd	U32	Input	Valid Address Range	Address of MVD for current MB in ECDWBUF
pbottomCurrentMbaffMvd	U32	Input	Valid Address Range	Address of bottom macro block of current macro block pair in MBAFF mode
pleftMvd	U32	Input	Valid Address Range	Address of MVD for Left MB in ECDWBUF
pbottomLeftMbaffMvd	U32	Input	Valid Address Range	Address of bottom macro block of Left macro block pair in MBAFF mode
streamBufferOffset	U32	Input	NA	Address Offset of stream Buffer in SL2
lseLdPrmsECD	U32	Input	Valid Address Range	Address of LSE load commands in SL2
lseStPrmsECD	U32	Input	Valid Address Range	Address of LSE store commands in SL2
size_MbInfoH264	U32	Input	NA	Data element size for Current MB Info
mbWidth	U16	Input	1 to 128	Width of picture in unit of macro blocks
mbHeight	U16	Input	1 to 128	Height of picture in unit of macro blocks

3.2.3 tHdvicp20Ecd3MPEG4ENCInitStruct

Description

This structure defines the parameters that are required for initializing the ECD3 module for MPEG-4 Encoder.

Fields

Field	Data Type	Input / output	Range	Description
short_video_header	U32	Input	0,1	Internal flag set to 1 for H.263
data_partitioned	U32	Input	0,1	'1' indicates data partitioning is enabled
reversible_vlc	U32	Input	0,1	'1' indicates RVLC is enabled
interlaced	U32	Input	0,1	Set to 0 => progessive video only
modulo_time_base	U16	Input	[0, 2^16 – 1]	modulo_time_base is number of seconds with respect to the last non-zero

				modulo_time_base or GOV time
vop_time_increment	U16	Input	[0, vop_time_increment_resolution)	Value in range of [0, vop_time_increment_resolution)
intra_dc_vlc_thr	U32	Input	[0, 7]	Set to 0 => Use Intra DC VLC for entire VOP
numBitsVopTimeIncr	U16	Input	[0, 2^16 – 1]	Number of bits used to code vop_time_increment
mbWidth	S16	Input	[0, 2^15 – 1]	Width of picture in unit of macroblocks
mbHeight	S16	Input	[0, 2^15 – 1]	Height of picture in unit of macroblocks
vop_coding_type	U32	Input	0,1	0 => "I", 1 => "P"
vop_fcode_forward	U32	Input	[1, 7]	Value in range of [1, 7] - determines MV range
vop_quant	U32	Input	[1, 31]	Value in range of [1, 31]
mbNumLength	U32	Input	[0, 2^32 – 1]	Number of bits used to represent macroblock_number
sl2_memory_start	void*	Input	Valid Address Range	Start address of SL2 memory
bitstream_buffer_start	void*	Input	Valid Address Range	Start address of bitstream buffer
sizeofMBInfo	U32	Input	Valid Address Range	Size of MbInfoMpeg4 struct that contains MB information
pABuf	void*	Input	Valid Address Range	Start address of ecdabuf (ECD3 Auxiliary buffer)
pABufcurrentMb	void*	Input	Valid Address Range	&ecdabuf->currentMb (MB info of current MB)
pABufupperRightMb	void*	Input	Valid Address Range	& ecdabuf->upperRightMb (MB info of upper right MB)
pABufcoLocatedMbA	void*	Input	Valid Address Range	& ecdabuf->coLocatedMbA (MB info of colocated MB)

pABufcommands	void*	Input	Valid Address Range	& ecdabuf->commands (ECD3 Command memory)
pABufRegInfo_SBC_TTLCNT	void*	Input	Valid Address Range	&ecdabuf->RegInfo_SBC_TTLCNT
pABufRegInfo_SBC_A_BITPTR	void*	Input	Valid Address Range	&ecdabuf->RegInfo_SBC_A_BITPTR (Stream Buffer Controller Bitstream pointer)
pRsdBuf	void*	Input	Valid Address Range	&rsdbuf (ECD3 Residual buffer)
pRsdBufresidual	void*	Input	Valid Address Range	&rsdbuf->residual (Residual data)
pWBuf	void*	Input	Valid Address Range	&ecdwbuf (ECD3 Work Buffer)
pWBufVOPinfo	void*	Input	Valid Address Range	& ecdwbuf->VOPinfo (Frame/Slice info)
pWBufupperLeftMb	void*	Input	Valid Address Range	& ecdwbuf->upperLeftMb (MB info of upper left MB)
pWBufupperMb	void*	Input	Valid Address Range	& ecdwbuf->upperMb (MB info of upper MB)
pWBufupperRightMb	void*	Input	Valid Address Range	& ecdwbuf->upperRightMb (MB info of upper right MB)
pWBufleftMb	void*	Input	Valid Address Range	& ecdwbuf->leftMb (MB info of left MB)
pWBufcurrentMb	void*	Input	Valid Address Range	& ecdwbuf->currentMb (MB info of current MB)
pWBufmvdBuffer	void*	Input	Valid Address Range	& ecdwbuf->mvdBuffer (MV difference buffer)

3.3 Data types for CALC3

This section describes data types used in APIs for CALC3 programming in different video codec implementations.

3.3.1 tHdvicp20calc3H264dFrameinit

Description
This Structure defines the parameters that are required for initializing the CALC3 module for H.264 Decoder.

Fields

Field	Data	Input /	Range	Description

	type	output		
picParamsWgt8x8	U16 *	Input	Valid Address Range	pointer to Weight matrix 8x8 stored in SL2 memory
picParamsWgt4x4	U16 *	Input	Valid Address Range	pointer to Weight matrix 4x4 stored in SL2 memory
seq_scaling_matrix_present_flag	U8	Input	0,1	Flag wich states if seq scaling matrix used is default/non-default, Range(0/1)
pic_scaling_matrix_present_flag	U8	Input	0,1	Flag which states if pic scaling matrix used is default/non-default, Range(0/1)
mb_aff_frame_flag	U8	Input	0,1	Flag which states if MBAFF mode is used, Range(0/1) 0 – Non-Mbaff Picture 1 - Mabaff Picture
field_pic_flag	U8	Input	0,1	Flag which states if picture structure is Frame/Field. Range(Frame-0, Field-1)
constrained_intra_pred_flag	U8	Input	0,1	Constrained Intra prediction flag. 0: Intra Prediction from Inter neighbors allowed 1: Intra Prediction from Inter neighbors not allowed
pCALC_TEST	volatile U32 *	Output	Valid Address Range	pointer to CALC_TEST register
pCALC_FWDQ_RND_INTRA	volatile U32 *	Output	Valid Address Range	Pointer to CALC_FWDQ_RND_INTRA register
pCALC_FWDQ_RND_INTER	volatile U32 *	Output	Valid Address Range	Pointer to CALC_FWDQ_RND_INTE register
pCALC_MODE	volatile U32 *	Output	Valid Address Range	Pointer to CALC_MODE register
pCALC3_IRQENABLE_SET_0	volatile U32 *	Output	Valid Address Range	Pointer to CALC3_IRQENABLE_SET_0
pCALC3_IRQENABLE_SET_1	volatile U32 *	Output	Valid Address Range	Pointer to CALC3_IRQENABLE_SET_1

pCALC3_IRQENABLE_SET_2	volatile U32 *	Output	Valid Address Range	Pointer to CALC3_IRQENABLE_SET_2
pCALC3_IRQENABLE_SET_3	volatile U32 *	Output	Valid Address Range	Pointer to CALC3_IRQENABLE_SET_3

3.3.2 tHdvicp20calc3H264eFrameinit

Description
This Structure defines the parameters that are required for initializing the CALC3 module for H.264 Encoder

Fields

Field	Data type	Input / output	Range	Description
pwgt4x4	U16 *	Input	Valid Address Range	Pointer to weight matrix 4x4 in CALCWBUF
pwgt8x8	U16 *	Input	Valid Address Range	Pointer to weight matrix 8x8 in CALCWBUF
pwgt8x8_01	U16 *	Input	Valid Address Range	Pointer to weight matrix 8x8 &(wgt8x8[0][1]) stored in CALCWBUF
pwgt8x8_00	U16 *	Input	Valid Address Range	Pointer to weight matrix 8x8 &(wgt8x8[0][0]) stored in CALCWBUF
pwgt8x8_11	U16 *	Input	Valid Address Range	Pointer to weight matrix 8x8 &(wgt8x8[1][1]) stored in CALCWBUF
pwgt8x8_10	U16 *	Input	Valid Address Range	Pointer to weight matrix 8x8 &(wgt8x8[1][0]) stored in CALCWBUF
pDefault_IpredModes	U64 *	Input	Valid Address Range	Pointer to default Intra prediction modes
pDefault_ID_IDX	U64 *	Input	Valid Address Range	Pointer to default IDX

pDefault_MV	U64 *	Input	Valid Address Range	Pointer to default motion vector
psH264ConstTables	U16 *	Input	Valid Address Range	Pointer to constant tables structure in state structure
psH264ConstTables_Src	U16 *	Input	Valid Address Range	Pointer to constant tables
Sizeof_sH264ConstTables_t	U32	Input	NA	Size of Constant tables structure
pdequantCoef1	U16 *	Input	Valid Address Range	Dequantization tables-stored in state structure
pdequantCoef8x8	U16 *	Input	Valid Address Range	Dequantization tables 8x8-stored in state structure
pwgt4x4_ConstTables	U16 *	Input	Valid Address Range	Pointer to weight matrix 4x4-in state structure
pwgt8x8_ConstTables	U16 *	Input	Valid Address Range	Pointer to weight matrix 8x8-in state structure
Sizeof_wgt4x4	U32	Input	NA	Size of weight matrix, stored in state structure
pu32interOrIntra	U32 *	Input	0,1	Inter/Intra flag in state structure, Range(0/1)
lseLdPrmsCALC	U32	Input	Valid Address Range	Pointer to LSE load commands for CALC in SL2
lseStPrmsCALC	U32	Input	Valid Address Range	Pointer to LSE store commands for CALC in SL2
constrained_intra_pred_flag	S16	Input	0,1	Constrained_intra_prediction_flag, Range(0/1)

CBPCnt0_Inter_Threshold	U32	Input	0,1,2	Enable/Disable CBP control#0 flag for Inter block & set the threshold cost for the block to be not coded if the block has very few small amplitude coefficients * '0' - Disable * '1' - Threshold cost 1 * '2' - Threshold cost 2
CBPCnt0_LumaDCTrans_Threshold	U32	Input	0,1,2	Enable/Disable CBP control#0 flag for Luma DC transform mode & set the threshold cost for the block to be not coded if the block has very few small amplitude coefficients * '0' - Disable * '1' - Threshold cost 1 * '2' - Threshold cost 2
CBPCnt0_ChroDCTrans_Threshold	U32	Input	0,1,2	Enable/Disable CBP control#0 flag for chroma DC transform mode & set the threshold cost for the block to be not coded if the block has very few small amplitude coefficients * '0' - Disable * '1' - Threshold cost 1 * '2' - Threshold cost 2

3.3.3 tHdvicp20Calc3MPEG4ENCInitStruct

Description

This structure defines the parameters that are required for initializing the CALC3 module for MPEG-4 Encoder.

Fields

Field	Data type	Input / output	Range	Description
uishort_video_header	U32	Input	0,1	Internal flag set to 1 for H.263
uiacpredEnable	U32	Input	0,1	1 => Enable AC prediction
uiIsEncoder	U32	Input	0,1	0 -> decoder 1-> encoder
uithresholdingCost	U32	Input	0, 1, 2	Thresholding cost used to set a block to be not_coded if the block has very few small amplitude coeffs. * '0' - Disable * '1' - Threshold cost 1 * '2' - Threshold cost 2

3.4 Data types for MC3

This section describes data types used in APIs for MC3 programming in different video codec implementations.

3.4.1 tHdvicp20mc3H264dFrameinit

Description
This Structure defines the parameters that are required for initializing the MC3 module for H.264 Decoder

Fields

Field	Data type	Input / output	Range	Description
weighted_bipred_idc	U8	Input	0,1,2	weighted bi prediction idc
weighted_pred_flag	U8	Input	0,1	weighted bi prediction flag, Range(0/1)
mb_aff_frame_flag	U8	Input	0,1	Flag which states if MBAFF mode is used, Range(0/1)
pMC_CNT	volatile U32 *	Output	Valid Address Range	pointer to MC counter register
pMC_PARAM0	volatile U32 *	Output	Valid Address Range	pointer to MC_PARAM0 register
pMC_ADDR_0	volatile U32 *	Output	Valid Address Range	pointer to MC_ADDR_0 register
pMC_ADDR_2	volatile U32 *	Output	Valid Address Range	pointer to MC_ADDR_1 register
pMC_ADDR_1	volatile U32 *	Output	Valid Address Range	pointer to MC_ADDR_2 register
pMC_ADDR_3	volatile U32 *	Output	Valid Address Range	pointer to MC_ADDR_3 register
mcRefData	U32 *	Output	Valid Address Range	pointer to mcRefData structure

3.4.2 McRefData

Description
This Structure defines the buffers required for storing the reference data required for MC3.

Fields

Field	Data type	Input / output	Range	Description
luma[LUMA_MCREF_HEIGHT][LUMA_MCREF_PITCH]	U8	Input	NA	Luma L0 buffer
lumaL1[LUMA_MCREF_HEIGHT][LUMA_MCREF_PITCH]	U8	Input	NA	Luma L1 buffer
lumaMb1L0[LUMA_MCREF_HEIGHT][LUMA_MCREF_PITCH]	U8	Input	NA	Luma L0 buffer (MBAFF)
lumaMb1L1[LUMA_MCREF_HEIGHT][LUMA_MCREF_PITCH]	U8	Input	NA	Luma L1 buffer (MBAFF)
chroma[CHROMA_MCREF_HEIGHT][CHROMA_MCREF_PITCH]	U8	Input	NA	Chroma L0 buffer
chromaL1[CHROMA_MCREF_HEIGHT][CHROMA_MCREF_PITCH]	U8	Input	NA	Chroma L1 buffer
chromaMb1L0[CHROMA_MCREF_HEIGHT][CHROMA_MCREF_PITCH]	U8	Input	NA	Chroma L0 buffer (MBAFF)
chromaMb1L1[CHROMA_MCREF_HEIGHT][CHROMA_MCREF_PITCH]	U8	Input	NA	Chroma L1 buffer (MBAFF)

3.4.3 tHdvicp20mc3H264eFrameinit

Description
This Structure defines the parameters that are required for initializing the MC3 module for H.264 Encoder

Fields

Field	Data type	Input / output	Range	Description
mcRefData_luma	U32	Input	Valid Address Range	pointer to Reference data Luma L0 in MCABUF
mcRefData_chroma	U32	Input	Valid Address Range	pointer to Referenced data Chroma L0 in MCABUF
mcRefData_lumaL1	U32	Input	Valid Address Range	pointer to Reference data Luma L1 in MCABUF

mcRefData_chromaL1	U32	Input	Valid Address Range	pointer to Reference data Chroma L1 in MCABUF

3.5 Data types for iME3

This section describes data types used in APIs for iME3 programming in different video codec implementations.

3.5.1 tHdvicp20Ime3MPEG4ENCInitStruct

Description
This structure defines the parameters needed in the MPEG4Enc iME3 Init function.

Fields

Field	Data type	Input / output	Range	Description
uivop_quant	U32	Input	[1, 31]	Quantization Parameter
uivop_coding_type	U32	Input	0,1	0 => "I", 1 => "P"
ubfirstFrame	U8	Input	0,1	1 => First frame of sequence
ubvop_fcode_forward	U8	Input	[1,7]	Determines motion vector search range
sl2startaddress	void *	Input	Valid Address Range	SL2 Start Address
sl2IMEMem	tHdvicp20Ime3 MPEG4ENCSL2_IME _Mem_Arch *	Input	Valid Address Range	Pointer to structure that contains the iME3 buffers in SL2 memory
mbOrigPixels	void *	Input	Valid Address Range	Address(mbOrigPixels): mbOrigPixels is a struct of type MbFormat that contains actual current frame pixel values
uisizeofmbOrigPixels	U32	Input	[0, 2^32-1]	Size of MbFormat struct that contains actual pixel values
currentInfo	void *	Input	Valid Address Range	Pointer to struct MbInfoMpeg4 that contains MB information regarding current MB
uisizeofMbInfo	U32	Input	[0, 2^32-1]	Size of MbInfoMpeg4 struct that contains MB information

3.6 Data types for IPE3

None

3.7 Data types for iLF3

None

3.8 Data types for LSE (Load-Store Engine)

This section describes data types used in APIs for LSE programming.

3.8.1 eLSETask

Description

This enum enumerates the tasks with LSEs.

Members

LOAD, *COMPUTE* and *STORE*

3.8.2 tHdvicp20LSECopy1DParamStruct

Description
This structure defines the parameters that are required for LSE 1D Copy command preparation.

Fields

Field	Data type	Input / output	Range	Description
uiDstBaseAdd	U32	Input	Valid Address Range	Destination base address
uiSrcBaseAdd	U32	Input	Valid Address Range	Source base address
uiSwap	U8	Input	0,1	Enable/disable byte swap
uiCondBit	U8	Input	0,1	Conditional copy bit for conditional copy
uiCondEnable	U8	Input	0,1	Conditional Copy Enable: If this is set to 1, 1D_copy is only applied when : (target bit pointed by Adptv_add command) = cond_bit If this is set to 0, 1D_copy is applied
uiSize	U16	Input	(0,256]	Size (Byte unit) to be copied; MAX 256

3.8.3 tHdvicp20LSECopy2DParamStruct

Description
This structure defines the parameters that are required for LSE 2D Copy command preparation.

Fields

Field	Data type	Input / output	Range	Description
uiDstBaseAdd	U32	Input	Valid Address Range	Destination base address
uiSrcBaseAdd	U32	Input	Valid Address Range	Source base address
uiDstOfs	U16	Input	Valid Address Range	Destination memory offset address (Byte unit)
uiSrcOfs	U16	Input	Valid Address Range	Source memory offset address (Byte unit)
uiSwap	U8	Input	0,1	Enable/disable byte swap
uiCondBit	U8	Input	0,1	Conditional copy bit for conditional copy
uiCondEnable	U8	Input	0,1	Conditional Copy Enable: If this is set to 1, 2D_copy is only applied when : (target bit pointed by Adptv_add command) = cond_bit If this is set to 0, 2D_copy is applied
uiHorzSize	U16	Input	Any positive value	Horizontal size. The actual horizontal size is uiHorzSize+1.
uiVertSize	U16	Input	Any positive value	Vertical size. The actual vertical size is uiVertSize+1.

3.8.4 tHdvicp20LSEUpdateCopyAddrParamStruct

Description
This structure defines the parameters that are required for preparation of update copy address command.

Fields

Field	Data type	Input / output	Range	Description
uiSrcCnt	U8	Input	[0, uiSrcMax-1]	Counter value for SRC_BASE_ADD
uiDstCnt	U8	Input	[0, uiDstMax-1]	Counter value for DST_BASE_ADD

uiSrcMax	U8	Input	Any positive value	Maximum number of buffers for source
uiDstMax	U8	Input	Any positive value	Maximum number of buffers for destination
uiSrcJmp	U16	Input	Any positive value	Address gap between two adjacent source buffers. SRC_JMP shall be a multiple of 16 for 128-bit word alignment [byte units]
uiDstJmp	U16	Input	Any positive value	Address gap between two adjacent destination buffers. DST_JMP shall be a multiple of 16 for 128-bit word alignment [byte units]

3.9 Data types for SB (SyncBox)

This section describes data types used in APIs for Sync Box programming.

3.9.1 tHdvicp20SBInitHWATaskParamStruct

Description
This structure contains the HWA Sync Box task-specific configuration parameters.

Fields

Field	Data type	Input / output	Range	Description
ubSyncActivationDest[ ]	U8 Array of size SYNCBOX_ MAX_SUCCESSSORS _ PER_TASK	Input	N/A	Destination task and node IDs for the successor activations
uiActivationMask	U32	Input	N/A	Activation Mask for the task
usParamAddressBase	U16	Input	Valid Address Range	Base offset into SL2 of the configuration parameters structure for a given task. Pointer value is expressed as a 128 bit aligned address.
usParamAddressInc	U16	Input	[0, 2^16-1]	ParamAddressInc contains the increment to

				be added to the current pointer
ubNumSuccessors	U8	Input	[0,4]	Number of successors to the task

3.9.2 tHdvicp20SBInitHWAParamStruct

Description
This structure contains the HWA SyncBox configuration parameters.

Fields

Field	Data type	Input / output	Range	Description
uiNodeAddress	eSBNode	Input	ECD3, CALC3,MC3, IPE3, iME3 and iLF3	Enum denoting the HWA associated with the SB
ubNumTasks	U8	Input	[1,2] for HWA SB	Number of tasks associated with the node
usErrorMsgDest	U8	Input	N/A	The 9-bit ERR_MSG_DEST value contains the node and task identifier to send an activation message when an error is detected
ubParamAddressModulo	U8	Input	[1,6]	Number of MB configuration arrays in pipeline
ubNumAsycLines	U8	Input	[0,6]	Number of asynchronous lines
ubAsyncEventAckReq	U8	Input	N/A	Defines if asynchronous line event needs a acknowledge upon task completion
ubAsyncActivationDest[]	U8 Array of size SYNCBOX_	Input	N/A	Values containing Async

	MAX_ASYNC_ EVENTS			destination activations
stTaskParam[]	tHdvicp20 SBInitHWATask ParamStruct array of size 2	Input	N/A	Task-specific SB configuration parameters for maximum of 2 tasks for HWA SyncBox

3.9.3 tHdvicp20SBInitCPUTaskParamStruct

Description
This structure contains the CPU SyncBox task-specific configuration parameters.

Fields

Field	Data type	Input / output	Range	Description
ubSyncActivationDest[]	U8 Array of size SYNCBOX_ MAX_SUCCESSSORS_ PER_TASK	Input	N/A	Destination task and node IDs for the successor activations
uiActivationMask	U32	Input	N/A	Activation Mask for the task
ubNumSuccessors	U8	Input	[0,4]	Number of successors to the task

3.9.4 tHdvicp20SBInitCPUParamStruct

Description
This structure contains the CPU SyncBox configuration parameters.

Fields

Field	Data type	Input / output	Range	Description
uiNodeAddress	eCPU_ID	Input	iCONT0, iCONT1	Enum denoting the iCONT associated with the SB
ubNumTasks	U8	Input	[1,6]	Number of tasks associated with the CPU
usErrorMsgDest	U8	Input		The 9-bit ERR_MSG_DEST value contains the node and task identifier to send an

			N/A	activation message when an error is detected
ubTaskSyncMode	U8	Input	0,1	Task Synchronization Mode (Synchronous / Asychronous task)
stTaskParam[]	tHdvicp20 SBInitCPUTask ParamStruct array of size SYNCBOX_MAX_ CPU_TASKS	Input	N/A	Task-specific SB configuration parameters for each task

3.10 Data types for SBH (SyncBox Handler)

This section describes data types used in APIs for SBH programming.

3.10.1 eSBHTaskType

Description

Enumeration for SBH task types

Members

Member	Value	Description
IMMEDIATE_TASK_END	0	Immediate task end
CPU_ONLY	1	CPU only activated
VDMA_ONLY	2	VDMA only activated
DM_CPU	3	DM then CPU activated
DM_VDMA	4	DM then VDMA only activated
CPU_VDMA	5	CPU then VDMA only activated
CPU_DM	6	CPU then DM activated
CPU_DM_VDMA	7	CPU then DM then VDMA activated

3.10.2 eSBHCounterId

Description

Enumeration for SBH counter IDs

Members

Member	Value	Description
SBH_COUNTER0	0	SBH Counter 1
SBH_COUNTER1	1	SBH Counter 2
SBH_COUNTER2	2	SBH Counter 3
SBH_COUNTER3	3	SBH Counter 4

3.10.3 tHdvicp20SBHTaskCtrlParamStruct

Description
This structure contains the SBH task specific control parameters.

Fields

Field	Data type	Input / output	Range	Description
ubTaskType	U8	Input	[0,7]	To configure the task type in Task Config Control Register (TCCR)
ubDMLglChannelEnable	U8	Input	N/A	DM logical channel enable mask
uiVDMAGroupEnable	U32	Input	N/A	vDMA Group mask
ubCounterEnable	U8	Input	N/A	Counter Enable Mask
ubCounterId	U8	Input	[0,3]	Select Counter
ubVDMASuperGroupSelect	U8	Input	N/A	VDMA Super Group Selector
ubVDMAGroupSelector	U8	Input	N/A	VDMA Group Selector
ubCount	U8	Input	[0,255]	Initial Counter Value

3.11 Data types for VDMA

This section describes data types used in APIs for VDMA programming.

3.11.1 tHdvicp20VDMAParametersStruct

Description
This structure contains the parameters that are required for initializing certain control registers of VDMA eg., No of contexts for sync/async transfers etc.

Fields

Field	Data type	Input / output	Range	Description
uiEOGSignalling	U32	Input	0,1	Enable/disable interrupt to signal End of Group transfer
ubMaxContextSyncTrf	U8	Input	[0,16]	Maximum number of contexts allocated to synchronous transfers
ubMaxContextAsyncTrf	U8	Input	[0,16]	Maximum number of contexts allocated to asynchronous transfers

3.11.2 tHdvicp20VDMAGroupDefinitionStruct

Description
This structure contains the parameters that define a group transfer.

Fields

Field	Data type	Input / output	Range	Description
ubGroupTransferType	U8	Input	[0,2]	GroupTransferType: Deterministic / Non-deterministic / unused
ubGroupMaster	U8	Input	0,1	GroupMaster: iCont1 / iCont2
ubGroupNumTransfers	U8	Input	[1,128]	Number of transfers in group
ubGroupSynchronousFlag	U8	Input	0,1	GroupSynchronousFlag: Synchronous / Asynchronous group transfer
puiGroupTrfDescAddress	pU32	Input	Valid Address Range	Base address of group transfer descriptor

3.11.3 tHdvicp20VDMAPaddingParamStruct

Description
This structure contains the parameters control padding (done by VDMA).

Fields

Field	Data type	Input / output	Range	Description
uiPaddingFlag	U8	Input	0,1	PaddingFlag: Enable / disable padding
uiPadWordLength	U8	Input	0,1	Pad Word Length: 0->To extend 1 byte(luma) 1->To extend 2 byte(chroma)
uiXDir	U8	Input	[0,3]	Padding Direction along horizontal direction: 0 - No Padding 1 - Undefined value 2 - Padding to the Left 3 - Padding to the Right
uiYDir	U8	Input	[0,3]	Padding Direction along vertical direction: 0 - No Padding 1 - Undefined value 2 - Padding to the Top 3 - Padding to the Bottom
uiXLength	U8	Input	[0,63]	Number of bytes to be padded: Here the actual number of pixels padded is (uiXLength+1) if uiXDir set to some value

uiYLength	U8	Input	[0,63]	Number of bytes to be padded: Here the actual number of pixels padded is (uiYLength+1).

3.11.4 tHdvicp20VDMA1DObjAutoIncParamStruct

Description
This structure contains the parameters control auto increment.

Fields

Field	Data type	Input / output	Range	Description
ubAutoIncFlag	U8	Input	0,1	AutoIncFlag: Enable / disable Auto Increment
siSrcByteInc	S16	Input	[0,4095]	SL2 increment it has to do for each trigger of vDMA.
uiSrcByteIncPostModulo	U16	Input	[0,4095]	SL2 increment it has to do when ever the number of times vDMA triggered reaches the iSrcModulo
siDestByteInc	S16	Input	[0,4095]	DDR increment it has to do for each trigger of vDMA.
uiDestByteIncPostModulo	U16	Input	[0,4095]	DDR increment it has to do when ever the number of times vDMA triggered reaches the uiDestModulo
uiSrcModulo	U16	Input	[0,255]	Source Modulo
uiDestModulo	U16	Input	[0,255]	Destination Modulo
uiSrcInitCntr	U16	Input	[0,255]	Source Counter Initialization
uiDestInitCntr	U16	Input	[0,255]	Destination Counter Initialization

3.11.5 tHdvicp20VDMAUVInterleavingParamStruct

Description
This structure contains the parameters control UV interleaving.

Fields

Field	Data type	Input / output	Range	Description
ubUVInterleave	U8	Input	0,1	UV Interleave Enable/Disable

3.11.6 tHdvicp20VDMADecimationParamStruct

Description
This structure contains the parameters control on-the-fly decimation.

Fields

Field	Data type	Input / output	Range	Description
ubDecimationEnableFlag	U8	Input	0,1	Enable / disable decimation
ubDecimationType	U8	Input	{0xx,100, 101,110, 111}	Type of decimation. 110: Skip 1 byte out of 2 in both direction 111: Skip 3 bytes out of 4 in both direction 100: Average by 2 in both direction 101: Average by 4 in both direction 0xx: No decimation at all

3.11.7 tHdvicp20VDMACompressionParamStruct

Description
This structure contains the parameters control on-the-fly de/compression.

Fields

Field	Data type	Input / output	Range	Description
ubCompression	U8	Input	0,1	On-the-fly de/compression Enable/Disable

3.11.8 tHdvicp20VDMAStateStruct

Description
This structure which has all the information regarding group definition, group transfer descriptor address etc.

Fields

Field	Data type	Input / output	Range	Description
tGroupDefinition[]	tHdvicp20VDMAGroupDefinition	Input	N/A	Group definition parameters for each group

3.12 Data types for DM (Data Mover)

None

3.13 Data types for MB (Mail Box)

This section describes data types used in APIs for Mailbox programming.

3.13.1 eMailBoxUsers

Description

Enumeration for the ID of Mail box users

Members

Member	Value	Description
MAILBOX_USER_0	0	Indicates the Cortex A9's User ID
MAILBOX_USER_1	1	Indicates the DSP sub-system's User ID
MAILBOX_USER_2	2	Indicates the Cortex-M3's User ID
MAILBOX_USER_3	3	Indicates the IVAHD's User ID

3.13.2 eMailBoxNums

Description

Enumeration for Mail box numbers

Members

Member	Value	Description
MAILBOX_0	0	Mali box 0
MAILBOX_1	1	Mali box 1
MAILBOX_2	2	Mali box 2
MAILBOX_3	3	Mali box 3
MAILBOX_4	4	Mali box 4
MAILBOX_5	5	Mali box 5

3.14 Data types for SMSET (Software Messages & System Event Trace)

None

4. CALC3 Functions

The CALC3 accelerator is used to perform core computation. For video encoders, it performs Intra prediction (if applicable), DC/AC prediction (if applicable), Transform, Quantization, Inverse Transform, Inverse Quantization and reconstruction of a macro block. For video decoders, it performs the same tasks except transform and quantization.

CALC3 embeds its own SyncBox module for synchronizing tasks with other HWAs. CALC3 is not capable of working with shared memory hence it has a Load/Store unit to transfer data from/to internal memories to/from Shared L2 memory. CALC3's SyncBox is a two-task syncbox and the tasks of it are: "Load data from SL2 to CALC3 internal memory" and "Store data from CALC3 internal memory to SL2".

The following are the APIs related to CALC3 available in the IVA-HD API Library. The CALC3-related APIs are codec-specific and are typically used for frame-level initialization of CALC3 before the coding/decoding pipeline is set up and triggered.

4.1 List of functions

1. HDVICP20_TI_CALC3_H264D_FrameInit
2. HDVICP20_TI_CALC3_H264E_FrameInit
3. HDVICP20_TI_CALC3_MPEG4D_FrameInit
4. HDVICP20_TI_CALC3_MPEG4E_FrameInit
5. HDVICP20_TI_CALC3_VC1D_FrameInit
6. HDVICP20_TI_CALC3_MPEG2D_FrameInit

4.2 HDVICP20_TI_CALC3_H264D_FrameInit

Name

```
HDVICP20_TI_CALC3_H264D_FrameInit
```

Synopsis

```
void HDVICP20_TI_CALC3_H264D_FrameInit
(
     tHdvicp20calc3H264dFrameinit *pFrmPrm
)
```

Arguments

Inputs

pFrmPrm	Pointer to CALC3 specific context structure, initialized at Frame level

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the CALC3 IP module. The function does the following:

1. Initializes the CALC_MODE register.
2. Initializes the weight matrix.

4.3 HDVICP20_TI_CALC3_H264E_FrameInit

Name

```
HDVICP20_TI_CALC3_H264E_FrameInit()
```

Synopsis

```
void HDVICP20_TI_CALC3_H264E_FrameInit
(
     tHdvicp20calc3H264eFrameinit *pFrmPrm
)
```

Arguments

Inputs

pFrmPrm	Pointer to CALC3 specific context structure, initialized at Frame level for H.264 encoder

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the CALC3 IP module for H.264 encoder (High Profile). The function does the following:

1. Initializes the CALC_MODE register.

2. Initializes the weight matrix.

4.4 HDVICP20_TI_CALC3_MPEG4D_FrameInit

Name

```
HDVICP20_TI_CALC3_MPEG4D_FrameInit()
```

Synopsis

```
void HDVICP20_TI_CALC3_MPEG4D_FrameInit
(
    tHdvicp20MPEG4DFrameInfo *pFrameInfo
)
```

Arguments

Inputs

*pFrameInfo	Pointer to the mpeg4 decoder state structure containing frame level information required for CALC3 initialization

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the CALC3 IP module. The function does the following:

1. itializes the CALC_MODE register.
2. Copies the weight matrix into work buffer.

4.5 HDVICP20_TI_CALC3_MPEG4E_FrameInit

Name

```
HDVICP20_TI_CALC3_MPEG4ENC_FrameInit()
```

Synopsis

```
void HDVICP20_TI_CALC3_MPEG4E_FrameInit
(
     tHdvicp20Calc3MPEG4ENCInitStruct *Calc3MPEG4ENCInitParams
)
```

Arguments

Inputs

Calc3MPEG4ENCInitParams	Pointer to the tHdvicp20Calc3MPEG4ENCInitStruct structure which contains the parameters needed in the MPEG4 Encoder CALC3 initilization function.

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the CALC3 IP module. The function does the following:

1. Programs the CALC3 Buffer Switch module to set up the ping-pong buffers.
2. Initializes the CALC_MODE register.

4.6 HDVICP20_TI_CALC3_VC1D_FrameInit

Name

```
HDVICP20_TI_CALC3_VC1D_FrameInit()
```

Synopsis

```
void HDVICP20_TI_CALC3_VC1D_FrameInit(tHdvicp20VC1DDecState *decstate)
```

Arguments

Inputs

decstate	Pointer to the vc1 decoder state structure **Error! Reference source not found.**

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the CALC3 IP module. The function does the following:

1. Initializes the CALC_MODE register.
2. Sets the direction of the code(encoder/decoder).

4.7 HDVICP20_TI_CALC3_MPEG2D_FrameInit

Name

```
HDVICP20_TI_CALC3_MPEG2D_FrameInit()
```

Synopsis

```
void HDVICP20_TI_CALC3_MPEG2D_FrameInit
(
    tHdvicp20MPEG2DFrameInfo *pFrameInfo
)
```

Arguments

Inputs

pFrameInfo	Pointer to the mpeg2 decoder state structure containing frame level information required for initializing CALC3

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the CALC3 IP module. The function does the following:

1. Initializes all the CALC3 registers.
2. Sets the direction of the code(encoder/decoder).

4.8 CALC3 API Usage

The CALC3-related APIs are codec-specific and are used for frame-level initialization of CALC3 before the coding/decoding pipeline is set up and triggered.
Once CALC3 is initialized before the pipeline is set up and triggered, the actual CALC3 processing and load/store operations happen while the pipeline is executing, and are triggered by the respective activators as programmed in the task-specific activation mask register in the SyncBox associated with CALC3.

See the appendix "Usage of Compute IP APIs" for an illustration of the usage of CALC3 APIs in a codec implementation.

THIS PAGE IS INTENTIONALLY LEFT BLANK

5. ECD3 Functions

The ECD3 accelerator is designed to entropy encode/decode the stream. For video encoder, it encodes the macroblock information and residual data into bitstream. For video decoder, it decodes the bitstream and recovers the macroblock information and residual data.

ECD3 embeds its own SyncBox module for synchronizing tasks with other HWAs. ECD3 is not capable of working with shared memory hence it has a Load/Store unit to transfer data from/to internal memories to/from Shared L2 memory. ECD3's SyncBox is a two-task syncbox and its tasks are: "Load data from SL2 to ECD3 internal memory" and "Store data from ECD3 internal memory to SL2". However, for bit-stream read and write operations, it directly works with SL2.

The following are the APIs related to ECD3 available in the IVA-HD API Library. The ECD3-related APIs are codec-specific and are typically used for frame-level initialization of ECD3 before the coding/decoding pipeline is set up and triggered.

5.1 List of functions

1. HDVICP20_TI_ECD3_H264D_FrameInit
2. HDVICP20_TI_ECD3_H264E_FrameInit
3. HDVICP20_TI_ECD3_H264E_SliceLevelConfig
4. HDVICP20_TI_ECD3_MPEG4D_FrameInit
5. HDVICP20_TI_ECD3_MPEG4E_FrameInit
6. HDVICP20_TI_ECD3_VC1D_FrameInit
7. HDVICP20_TI_ECD3_MPEG2D_FrameInit

5.2 HDVICP20_TI_ECD3_H264D_FrameInit

Name

```
HDVICP20_TI_ECD3_H264D_FrameInit ()
```

Synopsis

```
void HDVICP20_TI_ECD3_H264D_FrameInit
     (tHdvicp20ecd3H264dFrameinit *pFrmPrm)
```

Arguments

Inputs

pFrmPrm	Pointer to ECD3 specific context structure, initialized at Frame level

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the ECD3 IP module. The function does the following:

1. Programs the ECD3 Buffer Switch module to set up the ping-pong buffers.
2. Sets up the stream buffer controller.
3. Sets up the data buffer controller.
4. Initializes the MB position controller.
5. Prepares ECD3 commands and stores them in both ping and pong buffers of ECD3 Auxiliary Buffer.

5.3 HDVICP20_TI_ECD3_H264E_FrameInit

Name

```
HDVICP20_TI_ECD3_H264E_FrameInit()
```

Synopsis

```
void HDVICP20_TI_ECD3_H264E_FrameInit
     (tHdvicp20ecd3H264eFrameinit *pFrmPrm)
```

Arguments

Inputs

pFrmPrm	Pointer to ECD3 specific context structure, initialized at Frame level

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the ECD3 IP module for H.264 Encoder (High Profile). The function does the following:

1. Sets up the stream buffer controller.
2. Sets up the data buffer controller.
3. Initializes the MB position controller.

4. Prepares ECD3 commands and stores them in both ping and pong buffers of ECD3 Auxiliary Buffer.

5.4 HDVICP20_TI_ECD3_H264E_SliceLevelConfig

Name

```
HDVICP20_TI_ECD3_H264E_SliceLevelConfig()
```

Synopsis

```
void HDVICP20_TI_ECD3_H264E_SliceLevelConfig
     (U16 first_mb_in_slice,
      S16 *pbuf_ptr,
      S16 *pbit_ptr,
      U16 mbWidth)
```

Arguments

Inputs

pFrmPrm	Pointer to ECD3 specific context structure, initialized at Frame level
pbuf_ptr	Pointer to stream buffer in SL2 – byte pointer
pbit_ptr	Bit stream pointer – bit pointer
mbWidth	Frame width in terms of number of macro blocks

Outputs

None

Return Value

None

Description

Configures the ECD3 registers at Slice level for H.264 encoder.

5.5 HDVICP20_TI_ECD3_MPEG4D_FrameInit

Name

```
HDVICP20_TI_ECD3_MPEG4D_FrameInit()
```

Synopsis

```
void HDVICP20_TI_ECD3_MPEG4D_FrameInit
(
      tHdvicp20MPEG4DFrameInfo *pFrameInfo,
)
```

Arguments

Inputs

*pFrameInfo	Pointer to the mpeg4 decoder state structure

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the ECD3 IP module. The function does the following:

- Initializes all the interrupt enable registers.
- Sets up the stream buffer controller.
- Sets up the data buffer controller.
- Initializes the MB position controller.
- Initializes the mode registers.
- Initializes the MPEG4 specific registers.

5.6 HDVICP20_TI_ECD3_MPEG4E_FrameInit

Name

```
HDVICP20_TI_ECD3_MPEG4E_FrameInit()
```

Synopsis

```
void HDVICP20_TI_ECD3_MPEG4E_FrameInit
(
     tHdvicp20Ecd3MPEG4ENCInitStruct *Ecd3MPEG4ENCInitParams
)
```

Arguments

Inputs

Ecd3MPEG4ENCInitParams	Pointer to the tHdvicp20Ecd3MPEG4ENCInitStruct structure which contains the parameters needed for ECD3 Initialization.

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the ECD3 IP module. The function does the following:

1. Programs the ECD3 Buffer Switch module to set up the ping-pong buffers.
2. Sets up the stream buffer controller.
3. Sets up the data buffer controller.
4. Initializes the MB position controller.
5. Prepares ECD3 commands and stores them in both ping and pong buffers of ECD3 Auxiliary Buffer.
6. Initializes the mode registers.
7. Initializes the MPEG4 specific registers.
8. Initializes the frame/slice information buffer in the ECD3 Work Buffer.

5.7 HDVICP20_TI_ECD3_VC1D_FrameInit

Name

```
HDVICP20_TI_ECD3_VC1D_FrameInit ()
```

Synopsis

```
void HDVICP20_TI_ECD3_VC1D_FrameInit
(
      tHdvicp20VC1DDecState *decstate
```

```
)
```

Arguments

Inputs

*decstate	Pointer to the vc1 decoder state structure

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the ECD3 IP module. The function does the following:

1. Initializes all the interrupt enable registers.
2. Sets up the stream buffer controller.
3. Sets up the data buffer controller.
4. Sets the offset of residual data in residual buffer and that of all required Mbs in work buffer.
5. Sets the picture dimensions and the codec type.
6. Initializes the bit pointer and slice data pointer registers.
7. Initializes the codec specific registers.
8. Initializes the buffer switch registers.
9. Prepares the ECD3 commands and stores it in ECDABUF.

5.8 HDVICP20_TI_ECD3_MPEG2D_FrameInit

Name

```
HDVICP20_TI_ECD3_MPEG2D_FrameInit ()
```

Synopsis

```
void HDVICP20_TI_ECD3_MPEG2D_FrameInit
(
      tHdvicp20MPEG2DFrameInfo *pFrameInfo,
)
```

Arguments

Inputs

*pFrameInfo	Pointer to the mpeg2 decoder state structure

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the ECD3 IP module. The function does the following:

1. Initializes all the interrupt enable registers.
2. Sets up the stream buffer controller.
3. Writes the picture level information into ECDWBUF.

4. Initializes the MB controller registers.
5. Initializes the buffer pointer and mode registers.
6. Initializes the MPEG2 specific registers.
7. Prepares ECD3 commands and stores it in ECD3 command buffer.

5.9 ECD3 API Usage

The ECD3-related APIs are codec-specific and are used for frame-level initialization of ECD3 before the coding/decoding pipeline is set up and triggered.

Once ECD3 is initialized before the pipeline is set up and triggered, the actual ECD3 processing and load/store operations happen while the pipeline is executing, and are triggered by the respective activators as programmed in the task-specific activation mask register in the SyncBox associated with ECD3.

See the appendix "Usage of Compute IP APIs" for an illustration of the usage of ECD3 APIs in a codec implementation.

6. iLF3 Functions

The iLF3 accelerator is used to perform in-loop filtering (for de-blocking). It can also be used for out-loop filtering. iLF3 embeds its own SyncBox module for synchronizing tasks with other HWAs. iLF3 is capable of working with shared memory and its SyncBox is a single-task syncbox. The task associated with the iLF3 syncbox is "Do Loop Filtering".

The following are the APIs related to iLF3 available in the IVA-HD API Library. The iLF3-related APIs are codec-specific and are typically used for frame-level initialization of iLF3 before the coding/decoding pipeline is set up and triggered.

6.1 List of functions

1. HDVICP20_TI_ILF3_H264D_FrameInit
2. HDVICP20_TI_ILF3_H264E_FrameInit
3. HDVICP20_TI_ILF3_VC1D_FrameInit

6.2 HDVICP20_TI_ILF3_H264D_FrameInit

Name

HDVICP20_TI_ILF3_H264D_FrameInit()

Synopsis

```
void HDVICP20_TI_ILF3_H264D_FrameInit(U32 *puiLF3CfgAddrBase)
```

Arguments

Inputs

puiLF3CfgAddrBase	ILF3 Config base address (see CSL)

Outputs

None

Return Value

None

Description

Configures the ILF_CONFIG register at frame level for H.264 decoder.

6.3 HDVICP20_TI_ILF3_H264E_FrameInit

Name

HDVICP20_TI_ILF3_H264E_FrameInit()

Synopsis

```
void HDVICP20_TI_ILF3_H264E_FrameInit(U32 *puiLF3CfgAddrBase)
```

Arguments

Inputs

puiLF3CfgAddrBase	ILF3 Config base address (see CSL)

Outputs

None

Return Value

None

Description

Configures the iLF3 ILF_CONFIG register at frame level for H.264 Encoder.

6.4 HDVICP20_TI_ILF3_VC1D_FrameInit

Name

HDVICP20_TI_ILF3_VC1D_FrameInit()

Synopsis

```
void HDVICP20_TI_ILF3_VC1D_FrameInit(U32 *puiLF3CfgAddrBase)
```

Arguments

Inputs

puiLF3CfgAddrBase	ILF3 Config base address (see CSL)

Outputs

None

Return Value

None

Description

This function configures iLF3 ILF_CONFIG register at frane level for VC-1 decoder.

6.5 iLF3 API Usage

The iLF3-related APIs are codec-specific and are used for frame-level or slice-level initialization of iLF3 before the coding/decoding pipeline is set up and triggered.
Once iLF3 is initialized before the pipeline is set up and triggered, the actual iLF3 processing happens while the pipeline is executing, and is triggered by the respective activators as programmed in the task-specific activation mask register in the SyncBox associated with iLF3.

Following is an example of H.264 decoder sequence for iLF3 programming and execution.

1. Set the ILF_CONFIG register using API "HDVICP20_TI_ILF3_H264D_FrameInit" provided here.
2. Prepare MBCT (MB info and MB data) in SL2
3. Wait for Sync box start message (Sync Box APIs can be used for this)
4. Sync box send a start task message to load MBCT to local registers
5. iLF3 loads MBCT, processes MB , store filtered pixels
6. iLF3 sends end-of-task signal to Sync box and waits for new start

THIS PAGE IS INTENTIONALLY LEFT BLANK

7. iME3 Functions

The iME3 accelerator is used to perform motion estimation in video encode processing. It compares a block of current frame to various reference search blocks, and provides means to find out which one is the least different to the reference block.

It is also capable of interpolating a block with half or quarter pixel precision. Additionally, it supports searching for the best matching block within the interpolated data.

It is programmable accelerator and its performance depends upon the motion estimation algorithm. However, it is capable of executing a reasonable algorithm within a 900 cycles per macroblock budget.

iME3 embeds its own Sync Box module for synchronizing tasks with other HWAs. iME3 is capable of working with shared memory and its SyncBox is a single-task SyncBox. The task associated with the iME3 SyncBox is "Do Motion Estimation".

The following are the APIs related to iME3 available in the IVA-HD API Library. The iME3-related APIs are codec-specific and are typically used for frame-level initialization of iME3 before the encoding pipeline is set up and triggered.

Note: IVA-HD API library provides only initialization API which loads ME program to iME3 hardware accelerator. Other iME3 specific APIs and related documentation required for Motion Estimation (ME) algorithm implementation in encoder side are available separately. For details on these iME3 APIs for ME algorithm implementation and tools required for iME3 program generation, please contact Texas Instruments.

7.1 iME3 Basic Programming Model

IME supports a wide range of user defined ME algorithms, based on the SAD (Sum of Absolute Differences, for each pixel of a macro-block or partition of a macro-block) metric. It supports Frame and Field processing, as well as P-Frame, B-frame (including Bi-prediction) or multiple Frame type of algorithms. IME embeds a generic processor that supports ANSI C processing. However its ISA has been extended with specific instructions that make use of dedicated operators. Please refer to iME3 TRM for details on APIs provided for ME algorithm implementation using dedicated instructions. Following are the steps for basic programming of iME3:

1. Write specific ME algorithm implementation code using APIs provided in iME3 TRM
2. Compile and generate iME3 program using the tool provide by TI
3. Use IVA-HD APIs to boot iME3 and load the program to hardware accelerator
4. Once the program is loaded to iME3, it is ready for executing with new motion estimation algorithm which can be triggered by SyncBox in encoder pipeline implementation.

7.2 List of functions

1. HDVICP20_TI_IME3_IvahdPicInit
2. HDVICP20_TI_IME3_Config

7.3 HDVICP20_TI_IME3_IvahdPicInit

Name

HDVICP20_TI_IME3_IvahdPicInit()

Synopsis

```
void HDVICP20_TI_IME3_IvahdPicInit(tHdvicp20Ime3InitStruct *pFrmPrm)
```

Arguments

Inputs

pFrmprm	Pointer to the tHdvicp20Ime3InitStruct structure which contains the parameters needed for the initialization of iME3 IP

Outputs

None

Return Value

None

Description

Loads the IME3 code from SL2 memory to program memory of IME3.

7.4 HDVICP20_TI_IME3_Config

Name

HDVICP20_TI_IME3_Config()

Synopsis

```
void HDVICP20_TI_IME3_Config(tHdvicp20Ime3InitStruct *pFrmPrm)
```

Arguments

Inputs

pFrmprm	Pointer to the tHdvicp20Ime3InitStruct structure which contains the parameters needed for the initialization of iME3 IP

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the iME3 IP module. The function does the following:
Write the address of Stack parameters which is stored in SL2 to IME3 registers
Configures the Circular Buffer

8. IPE3 Functions

The IPE3 accelerator is used to perform the intra prediction estimation in video encode. It supports two modes, depending on the video standard:

- Spatial Intra Prediction estimation for H.264 and AVS. It creates intra prediction macro blocks with given intra modes from original macroblock and provides cost estimation between the original macroblock and each pseudo intra prediction macroblock, and then choose a mode with the smallest cost to recommend it as the optimal intra prediction mode.
- Spatial Activity for MPEG-1/2/4 and VC-1. It calculates the spatial activity of the original luminance samples with the specified block size. This mode intends to provide information of the original luma pixels. The values can be used to decide the coding parameters of the macroblock such as the coding mode and the quantization parameter.

The number of prediction modes to be searched can be programmed and its performance depends upon the number of modes being searched.

IPE3 embeds its own SyncBox module for synchronizing tasks with other HWAs. IPE3 is not capable of working with shared memory hence it has a Load/Store unit from/to internal memories to/from internal memories to Shared L2 memory. IPE3's SyncBox is a two-task syncbox and its tasks are: "Load data from SL2 to IPE3 internal memory" and "Store data from IPE3 internal memory to SL2".

The following are the APIs related to ECD3 available in the IVA-HD API Library. The ECD3-related APIs are codec-specific and are typically used for frame-level initialization of ECD3 before the encoding pipeline is set up and triggered.

8.1 List of functions

1. HDVICP20_TI_IPE3_H264E_FrameInit

8.2 HDVICP20_TI_IPE3_H264E_FrameInit

Name

```
HDVICP20_TI_IPE3_H264E_FrameInit()
```

Synopsis

```
void HDVICP20_TI_IPE3_H264E_FrameInit
(
     tHdvicp20IPE3InputParams *pIPEInputParams
)
```

Arguments

Inputs

pIPEInputParams	Pointer to the tHdvicp20IPE3InputParams structure in which IPE3 parameters needtio set for H.264 encoder

Outputs

None

Return Value

None

Description

Configures the IPE3 IPE_CTRL register and IPE input paramters at frame level for H.264 encoder.

9. MC3 Functions

The MC3 accelerator is used to perform motion compensation. It requires a region in SL2 which contains sufficient pixels to perform motion compensation and some means of motion vector within that region. It creates an inter prediction macroblock.

MC3 embeds its own SyncBox module for synchronizing tasks with other HWAs. MC3 is not capable of working with shared memory hence it has a Load/Store unit to transfer data from/to internal memories to/from Shared L2 memory. MC3's SyncBox is a two-task syncbox and its tasks are: "Load data from SL2 to MC3 internal memory" and "Store data from MC3 internal memory to SL2".

The following are the APIs related to MC3 available in the IVA-HD API Library. The MC3-related APIs are codec-specific and are typically used for frame-level initialization of MC3 before the coding/decoding pipeline is set up and triggered.

9.1 List of functions

1. HDVICP20_TI_MC3_H264D_FrameInit
2. HDVICP20_TI_MC3_H264E_FrameInit
3. HDVICP20_TI_MC3_MPEG4D_FrameInit
4. HDVICP20_TI_MC3_MPEG4E_FrameInit
5. HDVICP20_TI_MC3_VC1D_FrameInit
6. HDVICP20_TI_MC3_MPEG2D_FrameInit

9.2 HDVICP20_TI_MC3_H264D_FrameInit

Name

```
HDVICP20_TI_MC3_H264D_FrameInit ()
```

Synopsis

```
void HDVICP20_TI_MC3_H264D_FrameInit
(
     tHdvicp20mc3H264dFrameinit *pFrmPrm
)
```

Arguments

Inputs

pFrmPrm	Pointer to MC3 specific context structure, initialized at Frame level

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the MC3 IP module. The function does the following:

1. Initializes the MC_CNT (counter is disabled) and MC_PARAM0 (codec type is set as H.264, initialize weighted bipredict ion field) registers.
2. Sets the base addresses of reference luma and chroma data.

9.3 HDVICP20_TI_MC3_H264E_FrameInit

Name

```
HDVICP20_TI_MC3_H264E_FrameInit ()
```

Synopsis

```
void HDVICP20_TI_MC3_H264E_FrameInit
     (tHdvicp20mc3H264eFrameinit *pFrmPrm)
```

Arguments

Inputs

pFrmPrm	Pointer to MC3 specific context structure, initialized at Frame level

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the MC3 IP module. The function does the following:

1. Initializes the MC_CNT (counter is disabled) and MC_PARAM0 (codec type is set as H.264, disable weighted bipredict ion) registers.

2. Sets the base addresses of reference luma and chroma data.

9.4 HDVICP20_TI_MC3_MPEG4D_FrameInit

Name

```
HDVICP20_TI_MC3_MPEG4D_FrameInit ()
```

Synopsis

```
void HDVICP20_TI_MC3_MPEG4D_InitCmd
(
      tHdvicp20MPEG4DFrameInfo *pFrameInfo,
)
```

Arguments

Inputs

*pFrameInfo	Pointer to the mpeg4 decoder state structure

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the MC3 IP module. The function does the following:

- Initializes all the MC3 registers.
- Sets the base addresses of reference luma chroma data.
- Copies the slice information into the MC buffer.

9.5 HDVICP20_TI_MC3_MPEG4E_FrameInit

Name

```
HDVICP20_TI_MC3_MPEG4E_FrameInit()
```

Synopsis

```
void HDVICP20_TI_MC3_MPEG4E_FrameInit
(
      tHdvicp20MC3MPEG4ENCInitStruct *Hdvicp20MC3MPEG4ENCInitParams
)
```

Arguments

Inputs

Hdvicp20MC3MPEG4ENCInitParams	Pointer to the tHdvicp20Calc3MPEG4ENCInitStruct structure which contains the parameters needed in the MPEG4Enc MC3 Init function.

Outputs

None

Return Value

None

Description

This function will do the frame-level initialization/programming of the MC3 IP module. The function does the following:

1. Programs the MC3 IPGW module to enable interrupts.
2. Programs the MC3 Buffer Switch module to set up the ping-pong buffers.
3. Initializes the MC_CNT (counter is disabled) and MC_PARAM0 (codec type is set as MPEG-4) registers.
4. Sets the base addresses of reference luma and chroma data.

9.6 HDVICP20_TI_MC3_VC1D_FrameInit

Name

```
HDVICP20_TI_MC3_VC1D_FrameInit()
```

Synopsis

```
void HDVICP20_TI_MC3_VC1D_InitCmd
(
      tHdvicp20VC1DDecState *decstate
)
```

Arguments

Inputs

*decstate	Pointer to the vc1 decoder state structure

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the MC3 IP module. The function does the following:

1. Initializes all the MC3 registers.
2. Sets the base addresses of reference luma and chroma data.
3. Initializes the buffer switch registers.
4. Copies the boundary information to SL2 memory.

9.7 HDVICP20_TI_MC3_MPEG2D_FrameInit

Name

```
HDVICP20_TI_MC3_MPEG2D_FrameInit ()
```

Synopsis

```
void HDVICP20_TI_MC3_MPEG2D_FrameInit
(
      tHdvicp20MPEG2DFrameInfo *pFrameInfo
)
```

Arguments

Inputs

*pFrameInfo	Pointer to the mpeg2 decoder state structure

Outputs

None

Return Value

None

Description

This function is used for frame-level initialization/programming of the MC3 IP module. The function does the following:

1. Initializes all the MC3 registers.
2. Sets the base addresses of reference luma chroma data.
3. Copies the slice information into the MC buffer.
4. DMA offsets and attributes which does not change throughout the frame are set.
5. Prepares the VDMA transfer descriptors.
6. Prepares the MC3 commands and moves that to SL2.

THIS PAGE IS INTENTIONALLY LEFT BLANK

10. Data Mover Functions

The Data Mover (DM) is a sub-module present in iCONT1 and iCONT2. The ICONTx Data Mover is used to perform data movement between ICONT internal memories (ITCM and DTCM), external shared L2 memory and VDMA transversal ports. It is a 1 physical channel with 2 queues, 4 logic channels basic DMA.

10.1 List of Macros

1. HDVICP20_TI_DM_SetContextParams
2. HDVICP20_TI_DM_SetSourceAddress
3. HDVICP20_TI_DM_SetDestinationAddress
4. HDVICP20_TI_DM_GetCurrentStatus
5. HDVICP20_TI_DM_Wait

10.2 HDVICP20_TI_DM_SetContextParams

Name

```
HDVICP20_TI_DM_SetContextParams()
```

Synopsis

```
HDVICP20_TI_DM_SetContextParams
(
     U32 *NodeAddrBase,
     U8  ContextId,
     U8  St_Type,
     U8  End_Type,
     U8  Cmd_Type,
     U16 Bytes
)
```

Arguments

Inputs

NodeAddrBase	Value contains base address of DataMover
ContextId	Value specifies the Context ID of which the attributes are getting set. There are 4 contexts Data Mover can support.
St_Type	Specifies how the start of the transfer is scheduled 0: Transfer is scheduled on write to ICONT_DMCONTEXT register 1: Transfer is scheduled on logical channel start notification
End_Type	Specifies how the end of the transfer is notified 0: Do not send End Of Task notification on transfer completion 1: Send End Of Logical channel notification on transfer completion
Cmd_Type	Type of transfer 0: Transfer from SL2 to DTCM 1: Transfer from SL2 to ITCM 2: Transfer from DTCM to transversal port of VDMA 3: Transfer from SL2 to Transversal port of VDMA 4: Transfer from DTCM to SL2
Bytes	Number of Bytes to transfer, must be a multiple of 128 b transfer

Outputs

None

Return Value

Not applicable as this a macro

Description

This API is used to set the following transfer parameters for a given context of the Data Mover: Number of bytes to transfer, type of transfer (SL2 to DTCM, DTCM to VDMA,. etc.), specifies how the start of the transfer is scheduled, specifies how the end of the transfer is notified, if an interrupt needs to be generated upon completion of the transfer. To set up a DM transfer, this function is to be used along with HDVICP20_TI_DM_SetSourceAddress and HDVICP20_TI_DM_SetDestinationAddress. The transfer is to be scheduled through the mechanism specified in the St_Type argument.

10.3 HDVICP20_TI_DM_SetSourceAddress

Name

```
HDVICP20_TI_DM_SetSourceAddress()
```

Synopsis

```
HDVICP20_TI_DM_SetSourceAddress
(
     U32 *NodeAddrBase,
     U8  ContextId,
     U32 SrcAddr
)
```

Arguments

Inputs

NodeAddrBase	Value contains base address of DataMover
ContextId	Value specifies the Context ID of which the attributes are getting set. There are 4 contexts Data Mover can support.
SrcAddr	Source address of the transfer (18 bit) (byte address, must be aligned on 128-bit boundary)

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to set the source address of the transfer for the given Data Mover context. To set up a DM transfer, this function is to be used along with HDVICP20_TI_DM_SetContextParams and HDVICP20_TI_DM_SetDestinationAddress. The transfer is to be scheduled through the mechanism specified in the St_Type argument passed to HDVICP20_TI_DM_SetContextParams.

10.4 HDVICP20_TI_DM_SetDestinationAddress

Name

```
HDVICP20_TI_DM_SetDestinationAddress()
```

Synopsis

```
HDVICP20_TI_DM_SetDestinationAddress
(
     U32 *NodeAddrBase,
     U8  ContextId,
     U32 DstAddr
)
```

Arguments

Inputs

NodeAddrBase	Value contains base address of DataMover
ContextId	Value specifies the Context ID of which the attributes are getting set. There are 4 contexts Data Mover can support.
DstAddr	Destination address of the transfer (18 bit) (byte address, must be aligned on 128-bit boundary)

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to set the destination address of the transfer for the given Data Mover context. To set up a DM transfer, this function is to be used along with HDVICP20_TI_DM_SetContextParams and HDVICP20_TI_DM_SetSourceAddress. The transfer is to be scheduled through the mechanism specified in the St_Type argument passed to HDVICP20_TI_DM_SetContextParams.

10.5 HDVICP20_TI_DM_GetCurrentStatus

Name

```
HDVICP20_TI_DM_GetCurrentStatus()
```

Synopsis

```
HDVICP20_TI_DM_GetCurrentStatus
(
     U32 *NodeAddrBase,
     U8  ContextId
)
```

Arguments

Inputs

NodeAddrBase	Value contains base address of DataMover
ContextId	Value specifies the Context ID of which the status getting read. There are 4 contexts Data Mover can support.

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to get the current status of Data Mover for the given context. This helps to check for the completion of a DM transfer.

10.6 HDVICP20_TI_DM_Wait

Name

```
HDVICP20_TI_DM_Wait()
```

Synopsis

```
HDVICP20_TI_DM_Wait
(
     U32 *NodeAddrBase,
     U8  ContextId
)
```

Arguments

Inputs

NodeAddrBase	Value contains base address of DataMover
ContextId	Value specifies the Context ID for which wait is applicable. There are 4 contexts Data Mover can support.

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to wait for the DM transfer to get completed for the given context.

10.7 DM API Usage Example

The following is a illustrative example to program the DM for a particular task.

```
/*-------------------------------------------------------*/
/* Data Mover Programming Example                        */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Wait till Data Mover context is available             */
/*-------------------------------------------------------*/

  While ( (HDVICP20_TI_DM_GetCurrentStatus
           (puiNodeAddrBase, ubContextId)) != 0);


/*-------------------------------------------------------*/
/* Set source address for the DM transfer                */
/*-------------------------------------------------------*/
          HDVICP20_TI_DM_SetSourceAddress
          (
              puiNodeAddrBase,
              ubContextId,
              uiSourceAddress
          );

/*-------------------------------------------------------*/
/* Set destination address for the DM transfer           */
/*-------------------------------------------------------*/

          HDVICP20_TI_DM_SetDestinationAddress
          (
              puiNodeAddrBase,
              ubContextId,
              uiDestinationAddress
          );

/*-------------------------------------------------------*/
/* Set the context parameters                            */
/*-------------------------------------------------------*/

          HDVICP20_TI_DM_SetContextParams
          (
                puiNodeAddrBase,
                ubContextId,
```

```
        ubStartTypeOfTransfer,

        ubEndTypeofTransfer,
        ubTransferType,
        usNumBytesToTransfer
    );
```

11. Load-Store Engine Functions

The Load-Store Engine (LSE) is a sub-module present in the IPE3, CALC3, MC3 and ECD3 accelerators. Each of these accelerators have their internal memories which contain one or more buffers which could typically be used as ping-pong buffers. The LSE is used to transfer data between the SL2 memory and the IP internal buffers. The LSE is also communicates with the IP core, the SyncBox associated with the IP, and the buffer-switch (BFSW) module associated with the IP.

11.1 List of functions

1. HDVICP20_TI_LSE_Init
2. HDVICP20_TI_LSE_SetParamAddr
3. HDVICP20_TI_LSE_SetLoadAddress
4. HDVICP20_TI_LSE_SetStoreAddress
5. HDVICP20_TI_LSE_Trigger
6. HDVICP20_TI_LSE_PrepareCommandCopy1D
7. HDVICP20_TI_LSE_PrepareCommandCopy2D
8. HDVICP20_TI_LSE_PrepareCommandUpdateCopyAddress
9. HDVICP20_TI_LSE_PrepareCommandRegWrite
10. HDVICP20_TI_LSE_PrepareCommandNop
11. HDVICP20_TI_LSE_END
12. HDVICP20_TI_LSE_PrepareMultiCopy1D
13. HDVICP20_TI_LSE_PrepareCopy2DwithUpdateCopyAddress
14. HDVICP20_TI_LSE_PrepareCommandEnd
15. HDVICP20_TI_LSE_PrepareCommandUpdateAdptvFlagAddr
16. HDVICP20_TI_LSE_PrepareCommandAdptvAdd
17. HDVICP20_TI_LSE_AdptvFlagAddrCntrUpdate
18. HDVICP20_TI_LSE_CopyAddrCntrUpdate

11.2 HDVICP20_TI_LSE_Init

Name

```
HDVICP20_TI_LSE_Init()
```

Synopsis

```
void HDVICP20_TI_LSE_Init
(
   U8 ubNodeId,
   U8 uiSyncboxBypass,
   U8 uiBfswChg,
   U8 uiInteosthru
)
```

Arguments

Inputs

ubNodeId	This specifies the NodeID. Please refer to enumerated data type "eHWAnode"
uiSyncboxBypass	Execute syncbox in Bypass mode or normal mode. 1- Syncbox bypass mode 0- Normal mode
uiBfswChg	Enable or disable buffer switch. 1 – Disable buffer switch 0 – Enable buffer switch
uiInteosthru	Int_eos through bit 0 : LSE does process for slice boundary after receiving int_eos 1 : int_eos is passed through to SB without the process for slice boundary.

Outputs

None

Return Value

None

Description

This API initializes the LSE's mode of operation. In a typical optimized codec scenario, the SyncBox executes in the normal mode and buffer switch is enabled. SyncBox bypass mode is used mainly for debug purposes.

11.3 HDVICP20_TI_LSE_SetParamAddr

Name

```
HDVICP20_TI_LSE_SetParamAddr()
```

Synopsis

```
void HDVICP20_TI_LSE_SetParamAddr
(
    U8  ubNodeId,
    U32 uiLoadParamStart,
    U32 uiStoreParamStart
)
```

Arguments

Inputs

ubNodeId	This specifies the NodeID. Please refer to enumerated data type "eHWAnode"
uiLoadParamStart	The start address of LSE parameters/commands in SL2 for load
uiStoreParamStart	The start address of LSE parameters/commands in SL2 for store

Outputs

None

Return Value

None

Description

This API sets the SL2 LSE parameters/commands start address for load and store in the format LSE expects and SyncBox expects. The LSE, on triggering, starts reading the commands from this address and executes them in order to complete the required 'load' or 'store' task. The 'load' task transfers the input data that the IP requires for processing from the SL2 memory to the IP internal memory. The 'store' task transfers the processed output data of the IP core from the IP internal memory to the SL2 memory.

11.4 HDVICP20_TI_LSE_SetLoadAddress

Name

```
HDVICP20_TI_LSE_SetLoadAddress()
```

Synopsis

```
void HDVICP20_TI_LSE_SetLoadAddress
(
     U8  ubNodeId,
     U32 uiLoadParamStart
)
```

Arguments

Inputs

ubNodeId	This specifies the NodeID. Please refer to enumerated data type "eHWAnode"
uiLoadParamStart	The start address of LSE parameters/commands in SL2 for load

Outputs

None

Return Value

None

Description

This API sets the SL2 LSE parameters/commands start address for load in the format LSE expects and Sync-Box expects. The LSE, on triggering, starts reading the commands from this address and executes them in order to complete the required 'load' task. The 'load' task transfers the input data that the IP requires for processing from the SL2 memory to the IP internal memory.

11.5 HDVICP20_TI_LSE_SetStoreAddress

Name

```
HDVICP20_TI_LSE_SetStoreAddress()
```

Synopsis

```
void HDVICP20_TI_LSE_SetStoreAddress
(
   U8  ubNodeId,
   U32 uiStoreParamStart
)
```

Arguments

Inputs

ubNodeId	This specifies the NodeID. Please refer to enumerated data type "eHWAnode"
uiStoreParamStart	The start address of LSE parameters/commands in SL2 for store

Outputs

None

Return Value

None

Description

This API sets the SL2 LSE parameters/commands start address for store in the format LSE expects and Sync-Box expects. The LSE, on triggering, starts reading the commands from this address and executes them in order to complete the required 'store' task. The 'store' task transfers the processed output data of the IP core from the IP internal memory to the SL2 memory.

11.6 HDVICP20_TI_LSE_Trigger

Name

```
HDVICP20_TI_LSE_Trigger()
```

Synopsis

```
void  HDVICP20_TI_LSE_Trigger
(
     U8 ubNodeId,
     U8 uiSyncboxBypass,
     U8 uiLoadCompStore,
     U8 uiBfswChg,
)
```

Arguments

Inputs

ubNodeId	This specifies the NodeID. Please refer to enumerated data type "eHWAnode"
uiSyncboxBypass	Execute syncbox in Bypass mode or normal mode. 1-sync box bypass mode 0 - Normal mode
uiLoadCompStore	This value stores the type of Copy/engine to be triggered by LSE: Load, Compute or Store
uiBfswChg	Enable or disable Buffer switch '1' Diasble buffer switch

	'0' Enable buffer switch

Outputs

None

Return Value

None

Description

This API sets the LSE's mode of operation and triggers the type of LSE Copy (load or store) / IP Core. In a typical optimized codec scenario, the SyncBox executes in the normal mode and buffer switch is enabled. SyncBox bypass mode is used mainly for debug purposes.

11.7 HDVICP20_TI_LSE_PrepareCommandCopy1D

Name

```
HDVICP20_TI_LSE_PrepareCommandCopy1D()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandCopy1D
(
     pU8 pubSl2BaseParam,
     tHdvicp20LSECopy1DParamStruct *Copy1DParamStruct
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
Copy1DParamStruct	This structure defines the parameters that are required for LSE 1D Copy command preparation. The structure needs to be populated before calling this API.

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API prepares the one dimensional copy command of LSE for copying of image/parameter data among the dedicated memories and SL2 memory. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.8 HDVICP20_TI_LSE_PrepareCommandCopy2D

Name

```
HDVICP20_TI_LSE_PrepareCommandCopy2D()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandCopy2D
(
     pU8 pubSl2BaseParam,
     tHdvicp20LSECopy2DParamStruct *Copy2DParamStruct
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
Copy2DParamStruct	This structure defines the parameters that are required for LSE 2D Copy command preparation. The structure needs to be populated before calling this API.

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API prepares the two dimensional copy command of LSE for copying of image/parameter data among the dedicated memories and SL2 memory. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.9 HDVICP20_TI_LSE_PrepareCommandUpdateCopyAddress

Name

```
HDVICP20_TI_LSE_PrepareCommandUpdateCopyAddress()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandUpdateCopyAddress
(
     pU8 pubSl2BaseParam,
     tHdvicp20LSEUpdateCopyAddrParamStruct
                                    * UpdateCopyAddressParameters
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
UpdateCopyAddressParameters	This structure defines the parameters that are required for preparation of update copy address command. The structure needs to be populated before calling this API.

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API Updates the source and destination addresses for the following copy commands, 1D_Copy() or 2D_Copy, in a circular manner. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.10 HDVICP20_TI_LSE_PrepareCommandRegWrite

Name

```
HDVICP20_TI_LSE_PrepareCommandRegWrite()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandRegWrite
(
     pU8 pubSl2BaseParam,
     U32 uiValue,
     U16 ucAddr,
     U8  ubCondBit,
     U8  ubCondEna
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
uiValue	Value to be given to IP core
ucAddr	Address given to the IP Core
ubCondBit	Conditional copy bit for conditional copy
ubCondEna	Enable/disable conditional copy

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API prepares the LSE command that writes a value to the specified IP core register. This enables the LSE to write commands for IP core/BFSW MMR (eg., to trigger the IP core after completion of load). This LSE command (RegWrite) is used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.11 HDVICP20_TI_LSE_PrepareCommandNop

Name

```
HDVICP20_TI_LSE_PrepareCommandNop()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandNop
(
     pU8 pubSl2BaseParam
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API prepares Nop() (no operation & skips to the next command address in SL2) LSE command. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.12 HDVICP20_TI_LSE_END

Name

```
HDVICP20_TI_LSE_END()
```

Synopsis

```
U32 HDVICP20_TI_LSE_END
(
     pU8 pubSl2BaseParam,
     eLSETask uiLoadCompStore
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
uiLoadCompStore	The value store the type of Copy/engine to be stopped by LSE: Load, Compute or Store

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API prepares the LSE command to enable the stopping of copy/computationl engine by LSE. This is used as the last command in the set of LSE commands that define a 'load' or 'store' task. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.13 HDVICP20_TI_LSE_PrepareMultiCopy1D

Name

```
HDVICP20_TI_LSE_PrepareMultiCopy1D()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareMultiCopy1D
(
     pU8 pubSl2BaseParam,
     tHdvicp20LSEUpdateCopyAddrParamStruct *UpdateCopyAddressParameters,
     tHdvicp20LSECopy1DParamStruct *Copy1DParamStruct
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
UpdateCopyAddressParameters	This structure defines the parameters that are required for preparation of update copy address command. The structure needs to be populated before calling this API.
Copy1DParamStruct	This structure defines the parameters that are required for LSE 1D Copy command preparation. The Structure needs to be populated before calling this API.

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This function has been defined to do multiple 1D copies when size exceeds 256 bytes. This function will also take care of UpdateCopyAddress for each transfer. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.14 HDVICP20_TI_LSE_PrepareCopy2DwithUpdateCopyAddress

Name

```
HDVICP20_TI_LSE_PrepareCopy2DwithUpdateCopyAddress()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCopy2DwithUpdateCopyAddress
(
     pU8 pubSl2BaseParam,
     tHdvicp20LSECopy2DParamStruct *LSECopy2DParams,
     tHdvicp20LSEUpdateCopyAddrParamStruct *LSEUpdateCopyAddrParams
 )
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
LSECopy2DParams	Structure defines the parameters that are required for LSE 2D Copy command preparation. The Structure needs to be populated before calling this API.
LSEUpdateCopyAddrParams	This structure defines the parameters that are required for preparation of update copy address command. The structure needs to be populated before calling this API.

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

Function to do 2D copy when either Src or Dst are ping-pong. This function will also take care of UpdateCopyAddress for each transfer. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.15 HDVICP20_TI_LSE_PrepareCommandEnd

Name

```
HDVICP20_TI_LSE_PrepareCommandEnd()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandEnd
(
     pU8 pubSl2BaseParam,
     U8  uiLoadCompStore
)
```

Arguments

Inputs

pubSl2BaseParam	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
uiLoadCompStore	The value store the type of Copy/engine to be stopped by LSE: Load, Compute or Store

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API has the same functionality as the HDVICP20_TI_LSE_END() API. In addition, This takes care of the extra Nops for byte alignment. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.16 HDVICP20_TI_LSE_PrepareCommandUpdateAdptvFlagAddr

Name

```
HDVICP20_TI_LSE_PrepareCommandUpdateAdptvFlagAddr()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandUpdateAdptvFlagAddr
(
     pU8 sl2_base_param,
     U16 src_cnt,
     U16 src_max,
     U16 src_jmp
 )
```

Arguments

Inputs

	Pointer to the LSE parameters in SL2. LSE starts decoding

sl2_base_param	instruction from this location.
src_cnt	Counter value for SRC_BASE_ADD
src_max	Maximum number of MBs in source buffer
src_jmp	Address gap between two adjacent flags (in bits)

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API updates the source address for the Apdtv_add() command in a circular manner. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.17 HDVICP20_TI_LSE_PrepareCommandAdptvAdd

Name

```
HDVICP20_TI_LSE_PrepareCommandAdptvAdd()
```

Synopsis

```
U32 HDVICP20_TI_LSE_PrepareCommandAdptvAdd
(
     pU8 sl2_base_param,
     U32 mbinfo_byte_addr,
     U8  bit_num
)
```

Arguments

Inputs

sl2_base_param	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
mbinfo_byte_addr	Target MBInfo byte address
bit_num	Bit number 0-7. Target data can obtained from MBInfo byte address[20 bits] & Bit_num[3 bits] (total 23 bits)

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API prepares the LSE AdptvAdd() command. Some data (especially MC reference input) depends on intra/inter, P/B, block size, etc. The constant Load/Store of these data causes increase unnecessary SL2 data traffic. In order to reduce the unnecessary data traffic, data transfer is desirable to be adaptive to ECD MB info. This data is extracted from SL2 MEM for both Load and Store Task. These LSE commands are used in defining the corresponding 'load' and 'store' tasks for each IP that contains a LSE.

11.18 HDVICP20_TI_LSE_AdptvFlagAddrCntrUpdate

Name

```
HDVICP20_TI_LSE_AdptvFlagAddrCntrUpdate()
```

Synopsis

```
U32 HDVICP20_TI_LSE_AdptvFlagAddrCntrUpdate
(
     pU8 sl2_base_param,
     U32 mb_number,
     U16  src_max
 )
```

Arguments

Inputs

sl2_base_param	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
mb_number	MB number
src_max	Maximum number of buffers for source

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API updates the Source address counter for the Apdtv_add() command in a circular manner.

11.19 HDVICP20_TI_LSE_CopyAddrCntrUpdate

Name

```
HDVICP20_TI_LSE_CopyAddrCntrUpdate()
```

Synopsis

```
U32 HDVICP20_TI_LSE_CopyAddrCntrUpdate
(
    pU8 sl2_base_param,
    U32 src_cnt,
    U32 dst_cnt,
    U8  src_max,
    U8  dst_max
)
```

Arguments

Inputs

sl2_base_param	Pointer to the LSE parameters in SL2. LSE starts decoding instruction from this location.
src_cnt	Counter value for Source base address
dst_cnt	Counter value for destintaion base address
src_max	Maximum number of buffers for source
dst_max	Maximum number of buffers for destination

Outputs

None

Return Value

Returns the size (in bytes) of the command

Description

This API updates the Source & destination address counter for the LSE command in a circular manner.

11.20 LSE API Example Usage

This section explains LSE example usage for 1D data transfer from SL2 to external memory, external memory to SL2, and triggering IP core from LSE in SyncBox Normal mode.

```
/*-------------------------------------------------------*/
 /* Example: LSE operation using LSE 1D Copy APIs        */
/*-------------------------------------------------------*/

 /*-------------------------------------------------------*/
/* Initialize LSE's mode of operation e.g. Syncbox in By-*/
 /* Pass or normal mode, BFSW enabled or disabled, etc.   */
 /* Note Init needs to be done only once for Frame/Picture*/
 /* This example is for Sync Box normal mode.            */
 /*-------------------------------------------------------*/

          HDVICP20_TI_LSE_Init
          (
              ubNodeId,
              uiSyncboxBypass,
              uiBfswChg,
              uiAdptv,
              uiInteosthru
          );

 /*-------------------------------------------------------*/
/* Prepare LSE parameters/commands in SL2 e.g. required  */
 /* source and destination addresses and copy set        */
/*-------------------------------------------------------*/
 /*-------------------------------------------------------*/
/* LSE parameters/commands for load from SL2 to dedicated*/
 /* Memory                                               */
/*-------------------------------------------------------*/
 Offset = 0; /* Counter for size of LSE commands         */

 Offset += HDVICP20_TI_LSE_PrepareCommandCopy1D
             (
               pubSl2BaseParam + Offset,
               Copy1DParamStruct
             );

 Offset += HDVICP20_TI_LSE_PrepareCommandEnd
            (
               pubSl2BaseParam + Offset,
               uiLoadCompStore
             );


 /*-------------------------------------------------------*/
/* Prepare required IP Core data for LSE to write into IP */
 /* core using HDVICP20_TI_LSE_PrepareCommandRegWrite()   */
```

```
 /* API.                                                      */
/*-----------------------------------------------------------*/

 Offset += HDVICP20_TI_LSE_PrepareCommandRegWrite
             (
               pubSl2BaseParam + Offset,
               uiValue,
               ucAddr,
               ubCondBit,
               ubCondEna
              );

 /* The following command also takes care of extra Nops    */
 /* for byte alignment.                                    */
 Offset += HDVICP20_TI_LSE_PrepareCommandEnd
            (
                pubSl2BaseParam + Offset,
                uiLoadCompStore
             );

 /*-----------------------------------------------------------*/
/* Prepare LSE Store parameters/commands.                    */
 /* Sync Box triggers LSE Store after LSE receives CORE    */
 /* completion notification.                                */
/*-----------------------------------------------------------*/
/*-----------------------------------------------------------*/
/* LSE parameter/commands for Store: SL2 to external Mem */
/*-----------------------------------------------------------*/

 Offset += HDVICP20_TI_LSE_PrepareCommandCopy1D
              (
                pubSl2BaseParam + Offset,
                Copy1DParamStruct
              );

 Offset += HDVICP20_TI_LSE_PrepareCommandEnd
            (
                pubSl2BaseParam + Offset,
                uiLoadCompStore
             );

 /* These tasks are triggered when appropriate SyncBox  */
 /* messages are received.                              */
```

12. SyncBox Functions

The Video Synchronization Box module is a simple messaging layer for low-overhead synchronization of the parallel computing units and is independent from the host processor. It is a piece of hardware to be instantiated in all IVA-HD accelerators. It allows frame level autonomy for the accelerator sub system. The SyncBox defines all aspects of synchronization, data sharing and parameters passing between accelerators. The SyncBox also controls the power consumption by offering the possibility to stop HWA clocks when no task is scheduled.

12.1 List of functions

1. HDVICP20_TI_SB_FillCommonHWACFGParams
2. HDVICP20_TI_SB_FillAsyncHWACFGParams
3. HDVICP20_TI_SB_FillTaskHWACFGParam
4. HDVICP20_TI_SB_ConfigureHWARegisters
5. HDVICP20_TI_SB_FillCommonCPUCFGParams
6. HDVICP20_TI_SB_FillTaskCPUCFGParam
7. HDVICP20_TI_SB_ConfigureCPURegisters
8. HDVICP20_TI_SB_SyncboxClearStatus
9. HDVICP20_TI_SB_SyncBoxSWReset
10. HDVICP20_TI_SB_WriteSyncBoxRegister
11. HDVICP20_TI_SB_ReadSyncBoxRegister

12.2 HDVICP20_TI_SB_FillCommonHWACFGParams

Name

```
HDVICP20_TI_SB_FillCommonHWACFGParams()
```

Synopsis

```
void HDVICP20_TI_SB_FillCommonHWACFGParams
(
     pU8 pubHWAParams,
     U8  uiNode,
```

```
        U8  ubNumTasks,
        U16 usErrorMsgDest,
        U8  ubParamAddressModulo
)
```

Arguments

Inputs

pubHWAParams	Pointer to a structure of type tHdvicp20SBInitHWAParamStruct that contains the HWA SyncBox configuration parameters.
uiNode	Node ID of the HWA SyncBox. Refee to enumerated data type "eHWANode"
ubNumTasks	Number of tasks associated with the node
usErrorMsgDest	The 9-bit value contains the node and task identifier to send an activation message when an error is detected
ubParamAddressModulo	Number of MB configuration arrays in pipeline

Outputs

None

Return Value

None

Description

This API fills up the HWA SyncBox's common configuration parameters (independent of task) in the HWAParams structure.

12.3 HDVICP20_TI_SB_FillAsyncHWACFGParams

Name

```
HDVICP20_TI_SB_FillAsyncHWACFGParams()
```

Synopsis

```
void HDVICP20_TI_SB_FillAsyncHWACFGParams
(
        pU8 pubHWAParams,
        U8  ubNumAsycLines,
        U8  ubAsyncEventAckReq,
        U8  ubAsyncActivationDest1,
        U8  ubAsyncActivationDest2,
        U8  ubAsyncActivationDest3,
        U8  ubAsyncActivationDest4
)
```

Arguments

Inputs

pubHWAParams	Pointer to a structure of type tHdvicp20SBInitHWAParamStruct that contains the HWA SyncBox configuration parameters.
ubNumAsycLines	Number of asynchronous lines
ubAsyncEventAckReq	Defines if asynchronous line event needs an acknowledge upon task completion
ubAsyncActivationDest1	Value containing Async destination activation

ubAsyncActivationDest2	Value containing Async destination activation
ubAsyncActivationDest3	Value containing Async destination activation
ubAsyncActivationDest4	Value containing Async destination activation

Outputs

None

Return Value

None

Description

This API fills up the HWA SyncBox's asynchronous event configuration parameters (independent of task) in the HWAParams structure.

12.4 HDVICP20_TI_SB_FillTaskHWACFGParam

Name

```
HDVICP20_TI_SB_FillTaskHWACFGParam()
```

Synopsis

```
void HDVICP20_TI_SB_FillTaskHWACFGParam
(
     pU8 pubHWAParams,
     U8  ubtaskid,
     U8  ubNumSuccessors,
     U8  ubSyncActivationDest1,
     U8  ubSyncActivationDest2,
     U8  ubSyncActivationDest3,
     U8  ubSyncActivationDest4,
     U32 uiActivationMask,
     U16 usParamAddressBase,
     U16 usParamAddressInc
)
```

Arguments

Inputs

pubHWAParams	Pointer to a structure of type tHdvicp20SBInitHWAParamStruct that contains the HWA SyncBox configuration parameters.
ubtaskid	This value contains the task ID
ubNumSuccessors	Number of successor tasks to be activated on completion of this task
ubSyncActivationDest1	Destination node ID for successor activation
ubSyncActivationDest2	Destination node ID for successor activation
ubSyncActivationDest3	Destination node ID for successor activation
ubSyncActivationDest4	Destination node ID for successor activation
uiActivationMask	Value to indicate the Activation mask of Task ID & corresponding NodeID
usParamAddressBase	Base offset into SL2 of the configuration parameters structure for a given task. Pointer value is expressed as a 128 bit aligned address.
usParamAddressInc	Contains the increment to be added to the current pointer

Outputs

None

Return Value

None

Description

This API fills up the HWA SyncBox's task specific configuration parameters in the HWAParams structure. This is the API to be used to build the processing pipeline by defining appropriate successor tasks and nodes to the present task.

12.5 HDVICP20_TI_SB_ConfigureHWARegisters

Name

```
HDVICP20_TI_SB_ConfigureHWARegisters()
```

Synopsis

```
void HDVICP20_TI_SB_ConfigureHWARegisters
(
    pU8 pubHWAParams
)
```

Arguments

Inputs

pubHWAParams	Pointer to a structure of type tHdvicp20SBInitHWAParamStruct that contains the HWA SyncBox configuration parameters.

Outputs

None

Return Value

None

Description

This API configures the necessary HWA SyncBox registers from the HWAParamStruct structure. Briefly these include setting task specific parameters for every successor/node, async parameters and common parameters like param modulo, event ack, Error logs, etc. The processing pipeline is built by defining appropriate successor tasks and nodes to the tasks in the IP (Use the HDVICP20_TI_SB_FillTaskHWACFGParam API).

12.6 HDVICP20_TI_SB_FillCommonCPUCFGParams

Name

```
HDVICP20_TI_SB_FillCommonCPUCFGParams()
```

Synopsis

```
void HDVICP20_TI_SB_FillCommonCPUCFGParams
(
     pU8 pubCPUParams,
     u8  uiNode,
     U8  ubNumTasks,
     U16 usErrorMsgDest,
     U8  ubTaskSyncMode
)
```

Arguments

Inputs

pubCPUParams	Pointer to a structure of type tHdvicp20SBInitCPUParamStruct that contains the CPU SyncBox configuration parameters.
uiNode	Node ID the CPU /iCONT SyncBox.Refer to enumerated data type "eCPU_ID"
ubNumTasks	Number of tasks associated with the node
usErrorMsgDest	The 9-bit value contains the node and task identifier to send an activation message when an error is detected
ubTaskSyncMode	This value contains the task synchronization mode

Outputs

None

Return Value

None

Description

This API fills up the CPU SyncBox's common configuration parameters (independent of task) in the CPUParams structure.

12.7 HDVICP20_TI_SB_FillTaskCPUCFGParam

Name

```
HDVICP20_TI_SB_FillTaskCPUCFGParam()
```

Synopsis

```
void HDVICP20_TI_SB_FillTaskCPUCFGParam
(
     pU8 pubCPUParams,
     U8 ubtaskid,
     U8 ubNumSuccessors,
     U8 ubSyncActivationDest1,
     U8 ubSyncActivationDest2,
     U8 ubSyncActivationDest3,
     U8 ubSyncActivationDest4,
     U32 uiActivationMask,
)
```

Arguments

Inputs

pubCPUParams	Pointer to a structure of type tHdvicp20SBInitCPUParamStruct that contains the CPU/ICONT SyncBox configuration parameters.
ubtaskid	This value contains the task ID
ubNumSuccessors	Number of successor tasks to be activated on completion of this task
ubSyncActivationDest1	Destination node ID for successor activation
ubSyncActivationDest2	Destination node ID for successor activation
ubSyncActivationDest3	Destination node ID for successor activation
ubSyncActivationDest4	Destination node ID for successor activation

uiActivationMask	Value to indicate the Activation mask of Task ID & corresponding NodeID

Outputs

None

Return Value

None

Description

This API fills up the CPU SyncBox's Task specific configuration parameters in the CPUParams structure. This is the API to be used to build the processing pipeline by defining appropriate successor tasks and nodes to the present task.

12.8 HDVICP20_TI_SB_ConfigureCPURegisters

Name

```
HDVICP20_TI_SB_ConfigureCPURegisters()
```

Synopsis

```
void HDVICP20_TI_SB_ConfigureCPURegisters
(
    pU8 pubCPUParams
)
```

Arguments

Inputs

pubCPUParams	Pointer to a structure of type tHdvicp20SBInitCPUParamStruct that contains the CPU/iCONT SyncBox configuration parameters.

Outputs

None

Return Value

None

Description

This API fills up the CPU SyncBox's Task specific configuration parameters in the CPUParams structure. This is the API to be used to build the processing pipeline by defining appropriate successor tasks and nodes to the present task This API configures the necessary CPU SynBox registers from CPUParamStruct structure. Briefly, these inclkude setting task specific parameters for successor/node and common parameters (independent of task). The processing pipeline is built by defining appropriate successor tasks and nides to the tasks (Use the HDVICP20_TI_SB_FillTaskCPUCFGParam API).

12.9 HDVICP20_TI_SB_ReadSyncBoxRegister

Name

```
HDVICP20_TI_SB_ReadSyncBoxRegister()
```

Synopsis

```
U32 HDVICP20_TI_SB_ReadSyncBoxRegister
(
     U32 sb_base_addr,
     U32 addr_reg
```

```
)
```

Arguments

Inputs

sb_base_addr	HWA Sync box base address to which register need to be read
addr_reg	Specific register address (offset from base address)

Outputs

None

Return Value

None

Description

Function to read the syncbox registers

12.10 HDVICP20_TI_SB_WriteSyncBoxRegister

Name

```
HDVICP20_TI_SB_WriteSyncBoxRegister()
```

Synopsis

```
void HDVICP20_TI_SB_WriteSyncBoxRegister
(
     U32 sb_base_addr,
     U32 addr_reg,
     U32 value
)
```

Arguments

Inputs

sb_base_addr	HWA Sync box base address to which register need to be write
addr_reg	Specific register address (offset from base address)
value	Value to be written to sync box register

Outputs

None

Return Value

None

Description

API to write a value to the syncbox registers.

12.11 HDVICP20_TI_SB_SyncBoxSWReset

Name

```
HDVICP20_TI_SB_SyncBoxSWReset()
```

Synopsis

```
U32 HDVICP20_TI_SB_SyncBoxSWReset(U32 sb_base_addr)
```

Arguments

Inputs

sb_base_addr	HWA Sync box base address which needs to be reset

Outputs

None

Return Value

Status code : TRUE/FALSE indicating the reset status of the sync box

Description

This function performs a Software Reset for sync box provided

12.12 HDVICP20_TI_SB_SyncboxClearStatus

Name

```
HDVICP20_TI_SB_SyncboxClearStatus()
```

Synopsis

```
U32 HDVICP20_TI_SB_SyncboxClearStatus(U32 sb_base_addr)
```

Arguments

Inputs

sb_base_addr	HWA Sync box base address which needs to be cleared

Outputs

None

Return Value

Status code : TRUE/FALSE indicating the clear status of the sync box

Description

This function clears the status of the sync box provided.

12.13 SyncBox API Example Usage

This section illustrates the programming of a HWA SyncBox using IVA-HD APIs.

```
/*-------------------------------------------------------*/
/* HWA SyncBox Programming Example                       */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Prepare SyncBox common parameters                     */
/*-------------------------------------------------------*/
          HDVICP20_TI_SB_FillCommonHWACFGParams
          (
              pubHWAParams,
              uiNodeId,
              ubNumTasks,
              usErrorMsgDest,
              ubParamAddressModulo
          );

/*-------------------------------------------------------*/
/* Prepare SyncBox Async parameters                      */
/*-------------------------------------------------------*/
```

```
            HDVICP20_TI_SB_FillAsyncHWACFGParams
            (
                pubHWAParams,
                ubNumAsycLines,
                ubAsyncEventAckReq,
                ubAsyncActivationDest1,
                ubAsyncActivationDest2,
                ubAsyncActivationDest3,
                ubAsyncActivationDest4
            );

    /*-------------------------------------------------------*/
    /* Prepare SyncBox Task specific parameters: Task 0      */
    /*-------------------------------------------------------*/


            HDVICP20_TI_SB_FillTaskHWACFGParam
             (
                  pubHWAParams,
                  ubtaskid,
                  ubNumSuccessors,
                  ubSyncActivationDest1,
                  ubSyncActivationDest2,
                  ubSyncActivationDest3,
                  ubSyncActivationDest4,
                  uiActivationMask,
                  usParamAddressBase,
                  usParamAddressInc
             );

    /*-------------------------------------------------------*/
    /* Prepare SyncBox Task specific parameters: Task 1      */
    /*-------------------------------------------------------*/

             HDVICP20_TI_SB_FillTaskHWACFGParam
             (
                  pubHWAParams,
                  ubtaskid,
                  ubNumSuccessors,
                  ubSyncActivationDest1,
                  ubSyncActivationDest2,
                  ubSyncActivationDest3,
                  ubSyncActivationDest4,
                  uiActivationMask,
                  usParamAddressBase,
                  usParamAddressInc
             );

     /*-------------------------------------------------------*/
    /* Configure HWA SyncBox registers                       */
     /* based on the parameters prepared above                */
    /*-------------------------------------------------------*/

             HDVICP20_TI_SB_ConfigureHWARegisters
             (
                  pubHWAParams
```

```
);
```

THIS PAGE IS INTENTIONALLY LEFT BLANK

13. SyncBox Handler Functions

The SyncBox Handler module is a part of iCONT1 and iCONT2. SBH is used for the management of multiple tasks. iCONTx is a multi-task node. So, it should have a multi-task sync box. To handle the different tasks and variations in the tasks we need a small piece of logic, called SyncBox Handler (only attached to iCONTx). SBH carries out task association to sub-activities carried out by the Data Mover, ARM968E and the VDMA. It also carries out sequencing of these sub-activities.

13.1 List of functions & Macros

1. HDVICP20_TI_SBH_ConfigureCtrlParams
2. HDVICP20_TI_SBH_ReadActvTaskListRegister
3. HDVICP20_TI_SBH_ConfigureTaskAck
4. HDVICP20_TI_SBH_ConfigureToNotifyEndOfTask
5. HDVICP20_TI_SBH_ConfigureTaskControlParams
6. HDVICP20_TI_SBH_UpdateDMLglChannelEnable
7. HDVICP20_TI_SBH_UpdatevDMAGroupEnable
8. HDVICP20_TI_SBH_FillTaskControlParams
9. HDVICP20_TI_SBH_MapvDMAGrouptoDMLglChannel
10. HDVICP20_TI_SBH_vDMAGroupStartInBypassMode
11. HDVICP20_TI_SBH_ReadvDMAGroupEndInSBBypass
12. HDVICP20_TI_SBH_IsTaskActivated
13. HDVICP20_TI_SBH_SetIRQMask
14. HDVICP20_TI_SBH_ResetIRQMask
15. HDVICP20_TI_SBH_SaveIRQMask
16. HDVICP20_TI_SBH_ReadIRQStatus
17. HDVICP20_TI_SBH_SetEndofTask
18. HDVICP20_TI_SBH_SendAckofTask
19. HDVICP20_TI_SBH_SetCounter

13.2 HDVICP20_TI_SBH_ConfigureCtrlParams

Name

```
HDVICP20_TI_SBH_ConfigureCtrlParams()
```

Synopsis

```
S32 HDVICP20_TI_SBH_ConfigureCtrlParams
(
     U32 *puiSBHAddrBase,
     U8 ubSyncBoxBypassMode,
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.
ubSyncBoxBypassMode	This Field defines if SyncBox is used to synchronize Task between the different HWAs. 0 : Sync Box used 1 : SyncBox Bypass mode

Outputs

None

Return Value

Status code : SUCCESS/FAIL

Description

This API configures the SyncBox Handler Control Parameters. The SyncBox handler enables to connect the 6 Task Syncbox to the iCont embedded interrupt controller,to Data Mover and to the VDMA.

13.3 HDVICP20_TI_SBH_ReadActvTaskListRegister

Name

```
HDVICP20_TI_SBH_ReadActvTaskListRegister()
```

Synopsis

```
S32 HDVICP20_TI_SBH_ReadActvTaskListRegister
(
     U32 *puiSBHAddrBase
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.

Outputs

None

Return Value

Returns the tasks that are interrupting the CPU

Description

This API logs reference of the task interrupting CPU. When 1, Task x has generated an interrupt to the CPU. Reading clears register.

13.4 HDVICP20_TI_SBH_ConfigureTaskAck

Name

```
HDVICP20_TI_SBH_ConfigureTaskAck()
```

Synopsis

```
S32 HDVICP20_TI_SBH_ConfigureTaskAck
(
    U32 *puiSBHAddrBase,
    U8 ubEnableAutoAck,
    U8 uiSetTaskAck
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.
ubEnableAutoAck	Enable Automatic ACKx 0 : Automatic ACKx is disabled 1 : Automatic ACKx is enabled.
uiSetTaskAck	Mask to Set Task Ack (0-5). e.g. bit 0 corresponds to Task Number 0

Outputs

None

Return Value

Status code : SUCCESS/FAIL

Description

This API configures Acknowledge Task register, if Automatic ACKx is to be enabled or not upon receiving. The API needs to be used only when ubEnableAutoAck == 0; If ubEnableAutoAck is disabled or == 0 then Task n is automatically acknowledged on writing 1 to these task Bits else if ubEnableAutoAck == 1 then this API does nothing. It returns an error message reporting illegal usage of the API.

13.5 HDVICP20_TI_SBH_ConfigureToNotifyEndOfTask

Name

```
HDVICP20_TI_SBH_ConfigureToNotifyEndOfTask ()
```

Synopsis

```
S32 HDVICP20_TI_SBH_ConfigureTaskAck
(
    U32 *puiSBHAddrBase,
    U8 uiSetTaskEOT
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.
uiSetTaskEOT	Mask to Set Task EndOfTask (0-5). e.g. bit 0 corresponds to Task Number 0

Outputs

None

Return Value

Staus Code : SUCCESS/FAIL

Description

This API configures the SBH to notify End of task to the SyncBox. Register is automatically cleared.

13.6 HDVICP20_TI_SBH_ConfigureTaskControlParams

Name

```
HDVICP20_TI_SBH_ConfigureTaskControlParams()
```

Synopsis

```
S32 HDVICP20_TI_SBH_ConfigureTaskAck
(
    U32 *puiSBHAddrBase,
    U8  ubTaskId,
    tHdvicp20SBHTaskCtrlParamStruct *TaskCtrlParamStruct
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.
ubTaskId	Value contains the task number. e.g. Bit 0 corresponds to Task Id 0
TaskCtrlParamStruct	Pointer to structure containing the task specific control parameters.

Outputs

None

Return Value

Status code : SUCCESS/FAIL

Description

This API Configures task specific control Registers. The parameters that are configured are CPU Task Type (eg., CPU only, DM then vDMA, etc.), DM logical channel enable mask, vDMA Group mask, Counter Enable Mask, Counter Selector, VDMA Super Group Selector, VDMA Group Selector, Initial Counter Value.

13.7 HDVICP20_TI_SBH_UpdateDMLglChannelEnable

Name

```
HDVICP20_TI_SBH_UpdateDMLglChannelEnable()
```

Synopsis

```
S32 HDVICP20_TI_SBH_UpdateDMLglChannelEnable
```

```
(
    U32 *puiSBHAddrBase,
    U8  ubTaskId,
    U8  ubDMLglChannelEnable
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.
ubTaskId	Value contains the task number. e.g. Bit 0 corresponds to Task Id 0
ubDMLglChannelEnable	Mask to Enable Data Mover logical Channel (0-3). Bit 0 : Priority0 DMLgl channel Bit 1 : Priority1 DMLgl channel Bit 2 : Priority2 DMLgl channel Bit 3 : Priority3 DMLgl channel

Outputs

None

Return Value

Status code : SUCCESS/FAIL

Description

This API updates the DM logical channel enable mask in the Task Config Control Register corresponding to the task.

13.8 HDVICP20_TI_SBH_FillTaskControlParams

Name

```
HDVICP20_TI_SBH_FillTaskControlParams()
```

Synopsis

```
void HDVICP20_TI_SBH_FillTaskControlParams
       (
           tHdvicp20SBHTaskCtrlParamStruct *TaskCtrlParamStruct,
           U8 ubTaskType,
           U8 ubDMLglChannelEnable,
           U8 uiVDMAGroupEnable,
           U8 ubCounterEnable,
           U8 ubCounterId
       )
```

Arguments

Inputs

TaskCtrlParamStruct	Pointer to structure containing the task specific control parameters
ubTaskType	Value to indicate the Task type. Refer to enumerated data type "eSBHTaskType" for 0x0: Immediate Task End 0x1: CPU only activated 0x2: VDMA only activated 0x3: DM then CPU activated

	0x4: DM then VDMA only activated 0x5: CPU then VDMA only activated 0x6: CPU then DM activated 0x7: CPU then DM then VDMA activated
ubDMLglChannelEnable	Mask to Enable Data Mover logical Channel (0-3). Bit 0 : Priority0 DMLgl channel Bit 1 : Priority1 DMLgl channel Bit 2 : Priority2 DMLgl channel Bit 3 : Priority3 DMLgl channel
uiVDMAGroupEnable	Mask to Enable vDMA Group (0-31) e.g. vDMA bit 0 corresponds to channel 0
ubCounterEnable	Enable SBH counter flag. 1: Enable, 0: Disable
ubCounterId	SBH counter Identifier : 0 to 3. Refer to enumerated data type "eSBHCounterId"

Outputs

None

Return Value

None

Description

This API prepares the Sync Box Handler task specific control parameters in the TaskCtrlParamStruct which could then be passed as an argument to HDVICP20_TI_SBH_ConfigureTaskControlParams.

13.9 HDVICP20_TI_SBH_MapvDMAGrouptoDMLglChannel

Name

```
HDVICP20_TI_SBH_MapvDMAGrouptoDMLglChannel()
```

Synopsis

```
S32 HDVICP20_TI_SBH_MapvDMAGrouptoDMLglChannel
(
    U32 *puiSBHAddrBase,
    U8  ubDMLglChannelId,
    U32 ubvDMAGroupId
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.
ubDMLglChannelId	Data Mover Logical Channel Identifier Bit 0 : Priority0 DMLgl channel Bit 1 : Priority1 DMLgl channel Bit 2 : Priority2 DMLgl channel Bit 3 : Priority3 DMLgl channel
ubvDMAGroupId	vDMA Group identifier (0-31) e.g. vDMA bit 0 corresponds to channel 0

Outputs

None

Return Value

Status code : SUCCESS/FAIL

Description

This API maps the vDMA Group identifier to the Data Mover Logical channel.

13.10 HDVICP20_TI_SBH_vDMAGroupStartInBypassMode

Name

```
HDVICP20_TI_SBH_vDMAGroupStartInBypassMode()
```

Synopsis

```
S32 HDVICP20_TI_SBH_vDMAGroupStartInBypassMode
(
    U32 *puiSBHAddrBase,
    U32 uivDMAGroupStart
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.
uivDMAGroupStart	Setting '1' to a bit will kick the respective vDMA Group in the Bypass mode e.g. vDMA bit 0 corresponds to channel 0.

Outputs

None

Return Value

Status code : SUCCESS/FAIL

Description

This API is used only in Bypass mode to kick the vDMA group.

13.11 HDVICP20_TI_SBH_ReadvDMAGroupEndInSBBypass

Name

```
HDVICP20_TI_SBH_ReadvDMAGroupEndInSBBypass()
```

Synopsis

```
U32 HDVICP20_TI_SBH_ReadvDMAGroupEndInSBBypass
(
    U32 *puiSBHAddrBase
)
```

Arguments

Inputs

puiSBHAddrBase	Pointer contains iCONT1/iCONT2 SBH Register base address.

Outputs

None

Return Value

vDMA Group End Register value

Description

This API is used only in Bypass mode to check end of vDMA group transfer.

13.12 HDVICP20_TI_SBH_IsTaskActivated

Name

```
HDVICP20_TI_SBH_IsTaskActivated()
```

Synopsis

```
U32 HDVICP20_TI_SBH_IsTaskActivated
(
     U8 uiTaskId,
     U8 ubTaskList
)
```

Arguments

Inputs

uiTaskId	TaskId for which activate/not actiovated status need to be found: Task Id can 0 to 6
ubTaskList	Task list with updated status

Outputs

None

Return Value

SBH_TASK_ACTIVATED /SBH_TASK_NOT_ACTIVATED

Description

This API is used check if the task is activated or not.

13.13 HDVICP20_TI_SBH_SetIRQMask

Name

```
HDVICP20_TI_SBH_SetIRQMask ()
```

Synopsis

```
HDVICP20_TI_SBH_SetIRQMask
(
     pU8 icont_sbh_regs,
     U32 uiIRQMask
)
```

Arguments

Inputs

icont_sbh_regs	Base address of the corresponding iCONTx SBH MMR
uiIRQMask	-When uiIRQMask n-bit is 1, corresponding event/interrupt n is set/enabled -When uiIRQMask n-bit is 0, there won't be effect any impact on corresponding event/interrupt n

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API (preprocessor macro) is used to set/enable icont interrupts/events using IRQ mask

13.14 HDVICP20_TI_SBH_ResetIRQMask

Name

```
HDVICP20_TI_SBH_ResetIRQMask ()
```

Synopsis

```
HDVICP20_TI_SBH_ResetIRQMask
(
     pU8 icont_sbh_regs,
     U32 uiIRQResetMask
)
```

Arguments

Inputs

icont_sbh_regs	Base address of the corresponding iCONTx SBH MMR
uiIRQResetMask	-When uiIRQResetMask n-bit is 1, corresponding event/interrupt n is cleared/disabled -When uiIRQResetMask n-bit is 0, there won't be effect any impact on corresponding event/interrupt n

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API (preprocessor macro) is used to clear/disable icont interrupts/events using IRQ mask

13.15 HDVICP20_TI_SBH_SaveIRQMask

Name

```
HDVICP20_TI_SBH_SaveIRQMask ()
```

Synopsis

```
HDVICP20_TI_SBH_SaveIRQMask
(
     pU8 icont_sbh_regs,
)
```

Arguments

Inputs

icont_sbh_regs	Base address of the corresponding iCONTx SBH MMR

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API (preprocessor macro) is used to read & save current IRQ mask

13.16 HDVICP20_TI_SBH_ReadIRQStatus

Name

```
HDVICP20_TI_SBH_ReadIRQStatus ()
```

Synopsis

```
HDVICP20_TI_SBH_ReadIRQStatus
(
     pU8 icont_sbh_regs,
)
```

Arguments

Inputs

icont_sbh_regs	Base address of the corresponding iCONTx SBH MMR

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to read the current IRQ Status

13.17 HDVICP20_TI_SBH_SetEndofTask

Name

```
HDVICP20_TI_SBH_SetEndofTask ()
```

Synopsis

```
HDVICP20_TI_SBH_SetEndofTask
(
     pU8 icont_sbh_regs,
     U32 act_mask,
     U32 task_id
)
```

Arguments

Inputs

icont_sbh_regs	Base address of the corresponding iCONTx SBH MMR
act_mask	Mask to Set Task EOT
task_id	Task number

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to send end of Synbox Task.

13.18 HDVICP20_TI_SBH_SendAckofTask

Name

```
HDVICP20_TI_SBH_SendAckofTask ()
```

Synopsis

```
HDVICP20_TI_SBH_SendAckofTask
(
     pU8 icont_sbh_regs,
     U32 act_mask,
     U32 task_id
)
```

Arguments

Inputs

icont_sbh_regs	Base address of the corresponding iCONTx SBH MMR
act_mask	Mask to Set task acknowledgement
task_id	Task number

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to send acknolegement to the Syncbox task

13.19 HDVICP20_TI_SBH_SetCounter

Name

```
HDVICP20_TI_SBH_SetCounter ()
```

Synopsis

```
HDVICP20_TI_SBH_SetCounter
(
     pU8 icont_sbh_regs,
     U32 counter_id,
     U32 count
)
```

Arguments

Inputs

icont_sbh_regs	Base address of the corresponding iCONTx SBH MMR
counter_id	Counter number
count	Count value to be set

Outputs

None

Return Value

Not applicable as this is a macro

Description

This API is used to set SBH counter to given value

13.20 SBH API Usage Example

The following is a illustrative example to program the SBH for a particular task.

```
/*-------------------------------------------------------*/
/* SyncBox Handler Programming Example                   */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Configure the SBH control parameters                  */
/*-------------------------------------------------------*/

          HDVICP20_TI_SBH_ConfigureCtrlParams
          (
                puiSBHAddrBase,
                ubSyncBoxBypassMode,
                ubEnableInterruptOnvDMAGroups,
                ubEnableAutoAck,
                ubErrorStatus,
                ubEnableErrorInterrupt
          );

/*-------------------------------------------------------*/
/* Prepare SyncBox Handler parameters for the specific   */
/* task                                                  */
/*-------------------------------------------------------*/
          HDVICP20_TI_SBH_FillTaskControlParams
          (
              TaskCtrlParamStruct,
              ubTaskType,
              ubDMLglChannelEnable,
              uiVDMAGroupEnable,
              ubCounterEnable,
              ubCounterId
          );

/*-------------------------------------------------------*/
/* Configure the Task Config Control Register pertaining */
/* to the task in the SBH using the parameters prepared  */
/* above                                                 */
/*-------------------------------------------------------*/

          HDVICP20_TI_SBH_ConfigureTaskControlParams
          (
              puiSBHAddrBase,
              ubTaskId,
              TaskCtrlParamStruct
          );

/* These tasks are triggered when appropriate SyncBox    */
/* messages are received.                                */
```

THIS PAGE IS INTENTIONALLY LEFT BLANK

14. VDMA Functions

VDMA is a DMA engine designated for IVA-HD IP video data transfers from/to IVA-HD external memory from/to IVA-HD internal SL2 memory and some on-the-fly data processing (padding, decimation, UV interleaving, etc.).

14.1 List of functions

1. HDVICP20_TI_VDMA_Open
2. HDVICP20_TI_VDMA_PushTrfDescNonDet
3. HDVICP20_TI_VDMA_SWReset
4. HDVICP20_TI_VDMA_TriggerGroupDet
5. HDVICP20_TI_VDMA_Wait
6. HDVICP20_TI_VDMA_Clear
7. HDVICP20_TI_VDMA_Configure1DCopyDet
8. HDVICP20_TI_VDMA_Configure1DCopyWithAutoIncDet
9. HDVICP20_TI_VDMA_Configure2DCopyDet
10. HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDet
11. HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncPaddingDet
12. HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDataProcDet
13. HDVICP20_TI_VDMA_Update1DCopyWithAutoIncDet
14. HDVICP20_TI_VDMA_PrepTrfDesc1DCopyNonDetShort
15. HDVICP20_TI_VDMA_PrepTrfDesc2DCopySL2ToSL2NonDetShort
16. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDetShort
17. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyDDRToDDRNonDet
18. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDet
19. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithPaddingNonDet
20. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDecimationNonDet
21. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithUVInterleavingNonDet
22. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithCompressionNonDet
23. HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDataProcessingNonDet

14.2 List of Macros

1. HDVICP20_TI_VDMA_UpdateYLength
2. HDVICP20_TI_VDMA_UpdateHeight
3. HDVICP20_TI_VDMA_UpdateYLengthNonDet
4. HDVICP20_TI_VDMA_UpdateHeightNonDet

5. HDVICP20_TI_VDMA_UpdateSrcPitch
6. HDVICP20_TI_VDMA_UpdateSrcPitchNonDet
7. HDVICP20_TI_VDMA_UpdateDstModulo
8. HDVICP20_TI_VDMA_UpdateSrcModulo
9. HDVICP20_TI_VDMA_UpdateSrcByteInc
10. HDVICP20_TI_VDMA_UpdateNumBytes
11. HDVICP20_TI_VDMA_UpdateDst
12. HDVICP20_TI_VDMA_UpdateSrc
13. HDVICP20_TI_VDMA_UpdateYOffset
14. HDVICP20_TI_VDMA_UpdateXOffset
15. HDVICP20_TI_VDMA_UpdateDstPitch
16. HDVICP20_TI_VDMA_UpdateDstPitchNonDet
17. HDVICP20_TI_VDMA_UpdateDstNonDet
18. HDVICP20_TI_VDMA_UpdateSrcNonDet
19. HDVICP20_TI_VDMA_UpdateYOffsetNonDet
20. HDVICP20_TI_VDMA_UpdateXOffsetNonDet
21. HDVICP20_TI_VDMA_SetFirstFlagNonDet
22. HDVICP20_TI_VDMA_SetLastFlagNonDet
23. HDVICP20_TI_VDMA_ResetFirstFlagNonDet
24. HDVICP20_TI_VDMA_ResetLastFlagNonDet
25. HDVICP20_TI_VDMA_TurnOffPaddingDet
26. HDVICP20_TI_VDMA_TurnOffPaddingNonDet

14.3 HDVICP20_TI_VDMA_Open

Name

```
HDVICP20_TI_VDMA_Open()
```

Synopsis

```
U32 HDVICP20_TI_VDMA_Open
(
     tHdvicp20VDMAParametersStruct       *pVDMAParamStruct,
     tHdvicp20VDMAStateStruct            *pVDMAStatusStruct
)
```

Arguments

Inputs

*pVDMAParamStruct	Pointer to a VDMA parameter structure which contains the parameters that are required for initializing certain control registers of VDMA eg., No of contexts for sync/aysnc transfers etc.
*pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.

Outputs

None

Return Value

Error Code (PASS or TRFEXCEEDMAX)

Description

Routine to configure control registers and group properties of the VDMA. This API is used during VDMA initialization.

14.4 HDVICP20_TI_VDMA_PushTrfDescNonDet

Name

```
HDVICP20_TI_VDMA_PushTrfDescNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PushTrfDescNonDet
(
     pU32   puiTrfDesc,
     U8     ubDescLength,
     U32    ubNumTrfDescriptors
)
```

Arguments

Inputs

pU32 puiTrfDesc	Pointer to the transfer descriptor
U8 ubDescLength	Flag indicating the descriptor length (long / short)
U32 ubNumTrfDescriptors	Number of transfer descriptors to be pushed

Outputs

None

Return Value

None

Description

Routine to push the transfer descriptor for a non-deterministic transfer to the VDMA.

14.5 HDVICP20_TI_VDMA_SWReset

Name

```
HDVICP20_TI_VDMA_SWReset()
```

Synopsis

```
void HDVICP20_TI_VDMA_SWReset(void)
```

Arguments

Inputs

None

Outputs

None

Return Value

None

Description

Routine to do SW Reset of the vDMA engine.

14.6 HDVICP20_TI_VDMA_TriggerGroupDet

Name

```
HDVICP20_TI_VDMA_TriggerGroupDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_TriggerGroupDet(U32 uigroupID)
```

Arguments

Inputs

U32 uigroupID	Group ID of the group to be triggered

Outputs

None

Return Value

None

Description

Routine to trigger a group transfer.

14.7 HDVICP20_TI_VDMA_Wait

Name

```
HDVICP20_TI_VDMA_Wait()
```

Synopsis

```
void HDVICP20_TI_VDMA_Wait(U32 uigroupID)
```

Arguments

Inputs

U32 uigroupID	Group ID of the group

Outputs

None

Return Value

None

Description

Routine to wait for completion of a group transfer.

14.8 HDVICP20_TI_VDMA_Clear

Name

```
HDVICP20_TI_VDMA_Clear()
```

Synopsis

```
void HDVICP20_TI_VDMA_Clear(U32 uigroupID)
```

Arguments

Inputs

U32 uigroupID	Group ID of the group

Outputs

None

Return Value

None

Description

Routine to clear the status of a group transfer.

14.9 HDVICP20_TI_VDMA_Configure1DCopyDet

Name

```
HDVICP20_TI_VDMA_Configure1DCopyDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_Configure1DCopyDet
(
     tHdvicp20VDMAStateStruct   *pVDMAStatusStruct,
     U32                        uiGroupId,
     U32                        uiTransferId,
     U32                        uiDirection,
     pU8                        pubSourceAddress,
     pU8                        pubDestinationAddress,
     U32                        uiSize
)
```

Arguments

Inputs

*pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.
uiGroupId	The Group ID for the current transfer
uiTransferId	The transfer id for each transfer types
uiDirection	Direction of Data transfer
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered

uiSize	Number of bytes to be transferred for each trigger of vDMA

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a Deterministic 1D transfer of data.

14.10 HDVICP20_TI_VDMA_Configure1DCopyWithAutoIncDet

Name

```
HDVICP20_TI_VDMA_Configure1DCopyWithAutoIncDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_Configure1DCopyWithAutoIncDet
(
     tHdvicp20VDMAStateStruct   *pVDMAStatusStruct,
     U32                        uiGroupId,
     U32                        uiTransferId,
     U32                        uiDirection,
     pU8                        pubSourceAddress,
     pU8                        pubDestinationAddress,
     U32                        uiSize,
     tHdvicp20VDMA1DObjAutoIncParamStruct *pAutoIncParam
)
```

Arguments

Inputs

*pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.
uiGroupId	The Group ID for the current transfer
uiTransferId	The transfer id for each transfer types
uiDirection	Direction of Data transfer
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
uiSize	Number of bytes to be transferred for each trigger of vDMA
pAutoIncParam	Pointer to the auto increment parameter structure

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a Deterministic 1D transfer of data with Auto Increment feature.

14.11 HDVICP20_TI_VDMA_Configure2DCopyDet

Name

HDVICP20_TI_VDMA_Configure2DCopyDet()

Synopsis

```
void HDVICP20_TI_VDMA_Configure2DCopyDet
(
     tHdvicp20VDMAStateStruct  *pVDMAStatusStruct,
     U32                       uiGroupId,
     U32                       uiTransferId,
     U32                       uiDirection,
     pU8                       pubSourceAddress,
     pU8                       pubDestinationAddress,
     U32                       uiXoffset,
     U32                       uiYOffset,
     U32                       uiWidth,
     U32                       uiHeight,
     U32                       uiSourcePitch,
     U32                       uiDestinationPitch
)
```

Arguments

Inputs

*pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.
uiGroupId	The Group ID for the current transfer
uiTransferId	The transfer id for each transfer types
uiDirection	Direction of Data transfer
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the X-direction where the 2D object is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a Deterministic 2D transfer of data.

14.12 HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDet

Name

HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDet()

Synopsis

```
void HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDet
(
     tHdvicp20VDMAStateStruct  *pVDMAStatusStruct,
     U32                       uiGroupId,
     U32                       uiTransferId,
     U32                       uiDirection,
     pU8                       pubSourceAddress,
     pU8                       pubDestinationAddress,
     U32                       uiXoffset,
     U32                       uiYOffset,
     U32                       uiWidth,
     U32                       uiHeight,
     U32                       uiSourcePitch,
     U32                       uiDestinationPitch ,
     tHdvicp20VDMA2DObjAutoIncParamStruct *pAutoIncParam
)
```

Arguments

Inputs

pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.
uiGroupId	The Group ID for the current transfer
uiTransferId	The transfer id for each transfer types
uiDirection	Direction of Data transfer
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the X-direction where the 2D object is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pAutoIncParam	Pointer to the auto increment parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a Deterministic 2D transfer of data with Auto Increment feature.

14.13 HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncPaddingDet

Name

```
HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncPaddingDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncPaddingDet
(
     tHdvicp20VDMAStateStruct  *pVDMAStatusStruct,
     U32                       uiGroupId,
     U32                       uiTransferId,
     U32                       uiDirection,
     pU8                       pubSourceAddress,
     pU8                       pubDestinationAddress,
     U32                       uiXoffset,
     U32                       uiYOffset,
     U32                       uiWidth,
     U32                       uiHeight,
     U32                       uiSourcePitch,
     U32                       uiDestinationPitch ,
     tHdvicp20VDMA2DObjAutoIncParamStruct *pAutoIncParam,
     tHdvicp20VDMAPaddingParamStruct      *pPaddingParams
)
```

Arguments

Inputs

*pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.
uiGroupId	The Group ID for the current transfer
uiTransferId	The transfer id for each transfer types
uiDirection	Direction of Data transfer
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the X-direction where the 2D object is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pAutoIncParam	Pointer to the auto increment parameter structure.
pPaddingParams	Pointer to the padding parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a Deterministic 2D transfer of data with padding, Auto Increment feature.

14.14 HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDataProcDet

Name

```
HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDataProcDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_Configure2DCopyWithAutoIncDataProcDet
(
     tHdvicp20VDMAStateStruct  *pVDMAStatusStruct,
     U32                       uiGroupId,
     U32                       uiTransferId,
     U32                       uiDirection,
     pU8                       pubSourceAddress,
     pU8                       pubDestinationAddress,
     U32                       uiXoffset,
     U32                       uiYOffset,
     U32                       uiWidth,
     U32                       uiHeight,
     U32                       uiSourcePitch,
     U32                       uiDestinationPitch ,
     tHdvicp20VDMA2DObjAutoIncParamStruct *pAutoIncParam,
     tHdvicp20VDMAPaddingParamStruct      *pPaddingParams ,
     tHdvicp20VDMADecimationParamStruct     *pDecimationParams,
     tHdvicp20VDMAUVInterleavingParamStruct *pUVInterleaveParams,
     tHdvicp20VDMACompressionParamStruct    *pCompressionParams
)
```

Arguments

Inputs

*pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.
uiGroupId	The Group ID for the current transfer
uiTransferId	The transfer id for each transfer types
uiDirection	Direction of Data transfer
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the X-direction where the 2D object is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pAutoIncParam	Pointer to the auto increment parameter structure.
pPaddingParams	Pointer to the padding parameter structure.
pDecimationParams	Pointer to the decimation parameter structure.
pUVInterleaveParams	Pointer to the UV interleave parameter structure.
pCompressionParams	Pointer to the compression parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a Deterministic 2D transfer of data with data processing operations.

14.15 HDVICP20_TI_VDMA_Update1DCopyWithAutoIncDet

Name

```
HDVICP20_TI_VDMA_Update1DCopyWithAutoIncDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_Update1DCopyWithAutoIncDet
(
     tHdvicp20VDMAStateStruct *pVDMAStatusStruct,
     U32                      uiGroupId,
     U32                      uiTransferId,
     pU8                      pubSourceAddress,
     pU8                      pubDestinationAddress,
     U32                      bytesToTransfer,
     U32                      srcCntr,
     U32                      dstCntr
)
```

Arguments

Inputs

pVDMAStatusStruct	Pointer to a VDMA state structure which has all the information regarding group definition, group transfer descriptor address etc.
uiGroupId	The Group ID for the current transfer
uiTransferId	The transfer id for each transfer types
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
bytesToTransfer	Number of bytes to be transferred
srcCntr	Source counter value to be updated
dstCntr	Destination counter value to be updated

Outputs

None

Return Value

None

Description

Routine to update the source and destination address parameters of a deterministic 1D transfer with auto increment.

14.16 HDVICP20_TI_VDMA_PrepTrfDesc1DCopyNonDetShort

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc1DCopyNonDetShort()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc1DCopyNonDetShort
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
```

```
        U32  uiLastFlag,
        U32  uiDirection,
        pU8  pubSourceAddress,
        pU8  pubDestinationAddress,
        U32  uiSize
    )
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
uiDirection	Direction of Data transfer
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
uiSize	Number of bytes to be transferred for each trigger of vDMA

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a short non-deterministic 1D transfer of data.

14.17 HDVICP20_TI_VDMA_PrepTrfDesc2DCopySL2ToSL2NonDetShort

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopySL2ToSL2NonDetShort()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopySL2ToSL2NonDetShort
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag,
     pU8  pubSourceAddress,
     pU8  pubDestinationAddress,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.

uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
pubSourceAddress	Source address from where the data to be transfered
pubDestinationAddress	Destination address to where the data to be transfered
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a short non-deterministic 2D transfer of data from SL2 to SL2 memory.

14.18 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDetShort

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDetShort()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDetShort
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     U32  uiDirection,
     pU8  pubSL2Address,
     pU8  pubDDRAddress,
     U32  uiXoffset,
     U32  uiYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSL2Pitch,
     U32  uiDDRPitch
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.

uiDirection	Direction of Data transfer.
pubSL2Address	The SL2 address from where the data of our interest starts.
pubDDRAddress	The DDR Base address.
uiXoffset	DDR Offset in the X-direction where the 2D object is placed.
uiYOffset	DDR Offset in the Y-direction where the 2D object is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSL2Pitch	Pitch in SL2 side.
uiDDRPitch	Pitch in DDR side.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a short non-deterministic 2D transfer of data.

14.19 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyDDRToDDRNonDet

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyDDRToDDRNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyDDRToDDRNonDet
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     pU8  pubSourceAddress,
     U32  uiSourceXoffset,
     U32  uiSourceYOffset,
     pU8  pubDestinationAddress,
     U32  uiDestinationXoffset,
     U32  uiDestinationYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
pubSourceAddress	Source address from where the data to be transfered.

uiSourceXoffset	Source Offset in the X-direction where the 2D object is placed.
uiSourceYOffset	Source Offset in the Y-direction where the 2D object is placed.
pubDestinationAddress	Destination address to where the data to be transfered.
uiDestinationXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiDestinationYOffset	Destination Offset in the Y-direction where the 2Dobject is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a long non-deterministic 2D transfer of data from DDR to DDR.

14.20 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDet

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyNonDet
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     U32  uiDirection,
     pU8  pubSourceAddress,
     pU8  pubDestinationAddress,
     U32  uiXoffset,
     U32  uiYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.

uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
uiDirection	Direction of Data transfer.
pubSourceAddress	Source address from where the data to be transfered.
pubDestinationAddress	Destination address to where the data to be transfered.
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the Y-direction where the 2Dobject is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a long non-deterministic 2D transfer of data.

14.21 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithPaddingNonDet

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithPaddingNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithPaddingNonDet
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     U32  uiDirection,
     pU8  pubSourceAddress,
     pU8  pubDestinationAddress,
     U32  uiXoffset,
     U32  uiYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch,
     tHdvicp20VDMAPaddingParamStruct *pPaddingParams
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.

uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
uiDirection	Direction of Data transfer.
pubSourceAddress	Source address from where the data to be transfered.
pubDestinationAddress	Destination address to where the data to be transfered.
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the Y-direction where the 2Dobject is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pPaddingParams	Pointer to the padding parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a long non-deterministic 2D transfer of data with padding.

14.22 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDecimationNonDet

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDecimationNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDecimationNonDet
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     U32  uiDirection,
     pU8  pubSourceAddress,
     pU8  pubDestinationAddress,
     U32  uiXoffset,
     U32  uiYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch,
     tHdvicp20VDMADecimationParamStruct *pDecimationParams
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer

uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
uiDirection	Direction of Data transfer.
pubSourceAddress	Source address from where the data to be transfered.
pubDestinationAddress	Destination address to where the data to be transfered.
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the Y-direction where the 2Dobject is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pDecimationParams	Pointer to the decimation parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a long non-deterministic 2D transfer of data with decimation.

14.23 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithUVInterleavingNonDet

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithUVInterleavingNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithUVInterleavingNonDet
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     U32  uiDirection,
     pU8  pubSourceAddress,
     pU8  pubDestinationAddress,
     U32  uiXoffset,
     U32  uiYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch,
     tHdvicp20VDMAUVInterleavingParamStruct *pUVInterleaveParams
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of

	data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
uiDirection	Direction of Data transfer.
pubSourceAddress	Source address from where the data to be transfered.
pubDestinationAddress	Destination address to where the data to be transfered.
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the Y-direction where the 2Dobject is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pUVInterleaveParams	Pointer to the UV interleave parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a long non-deterministic 2D transfer of data with UV interleaving.

14.24 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithCompressionNonDet

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithCompressionNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithCompressionNonDet
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     U32  uiDirection,
     pU8  pubSourceAddress,
     pU8  pubDestinationAddress,
     U32  uiXoffset,
     U32  uiYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch,
     tHdvicp20VDMACompressionParamStruct *pCompressionParams
)
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
uiDirection	Direction of Data transfer.
pubSourceAddress	Source address from where the data to be transfered.
pubDestinationAddress	Destination address to where the data to be transfered.
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the Y-direction where the 2Dobject is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pCompressionParams	Pointer to the compression parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a long non-deterministic 2D transfer of data with Compression.

14.25 HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDataProcessingNonDet

Name

```
HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDataProcessingNonDet()
```

Synopsis

```
void HDVICP20_TI_VDMA_PrepTrfDesc2DCopyWithDataProcessingNonDet
(
     pU32 puiDescBufferAddress,
     U32  uiGroupId,
     U32  uiFirstFlag,
     U32  uiLastFlag ,
     U32  uiDirection,
     pU8  pubSourceAddress,
     pU8  pubDestinationAddress,
     U32  uiXoffset,
     U32  uiYOffset,
     U32  uiWidth,
     U32  uiHeight,
     U32  uiSourcePitch,
     U32  uiDestinationPitch,
     tHdvicp20VDMAPaddingParamStruct *pPaddingParams,
     tHdvicp20VDMADecimationParamStruct *pDecimationParams,
     tHdvicp20VDMAUVInterleavingParamStruct *pUVInterleaveParams,
```

```
        tHdvicp20VDMACompressionParamStruct *pCompressionParams
    )
```

Arguments

Inputs

puiDescBufferAddress	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiGroupId	The Group ID for the current transfer
uiFirstFlag	This tells whether the current transfer is the first transfer in this group or not.
uiLastFlag	This tells whether the current transfer is the first transfer in this group or not.
uiDirection	Direction of Data transfer.
pubSourceAddress	Source address from where the data to be transfered.
pubDestinationAddress	Destination address to where the data to be transfered.
uiXoffset	Destination Offset in the X-direction where the 2D object is placed.
uiYOffset	Destination Offset in the Y-direction where the 2Dobject is placed.
uiWidth	This is the width of the 2D object to be transferred.
uiHeight	This is the height of the 2D object to be transferred.
uiSourcePitch	The Offset to jump at the source location after copying each line in the 2D object.
uiDestinationPitch	The Offset to jump at the destination location after copying each line in the 2D object.
pPaddingParams	Pointer to the padding parameter structure.
pDecimationParams	Pointer to the decimation parameter structure.
pUVInterleaveParams	Pointer to the UV interleave parameter structure.
pCompressionParams	Pointer to the compression parameter structure.

Outputs

None

Return Value

None

Description

Routine to prepare the transfer descriptor for a long non-deterministic 2D transfer of data with data processing capabilities.

14.26 HDVICP20_TI_VDMA_ UpdateDstModulo

Name

```
HDVICP20_TI_VDMA_ UpdateDstModulo()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateDstModulo(pubBaseAddr, ubVal)
(
     pU8  pubBaseAddr,
     U8   ubVal
)
```

Arguments

Inputs

pubBaseAddr	Pointer to a transfer descriptor for the current transfer of data

	through vDMA engine.
ubVal	Value for the destination modulo

Outputs

None

Return Value

Not Applicable as this is a macro

Description

Preprocessor macro to update the destination modulo parameter in VDMA descriptor.

14.27 HDVICP20_TI_VDMA_ UpdateSrcModulo

Name

```
HDVICP20_TI_VDMA_ UpdateSrcModulo()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateSrcModulo(pubBaseAddr, ubVal)
(
     pU8  pubBaseAddr,
     U8   ubVal
)
```

Arguments

Inputs

pubBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
ubVal	Value for the source modulo

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the source modulo parameter in VDMA descriptor

14.28 HDVICP20_TI_VDMA_ UpdateSrcByteInc

Name

```
HDVICP20_TI_VDMA_ UpdateSrcByteInc ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateSrcByteInc(pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through

	vDMA engine.
usVal	Value for the source byte increment

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the source byte increment parameter in VDMA descriptor

14.29 HDVICP20_TI_VDMA_ UpdateNumBytes

Name

```
HDVICP20_TI_VDMA_ UpdateNumBytes ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateNumBytes (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
usVal	Value for number of bytes

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the number of bytes parameter in VDMA descriptor

14.30 HDVICP20_TI_VDMA_ UpdateDst

Name

```
HDVICP20_TI_VDMA_ UpdateDst ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateDst (pBaseAddr, uiVal)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through

	vDMA engine.
uiVal	Destination address

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the destination address parameter in VDMA descriptor.

14.31 HDVICP20_TI_VDMA_ UpdateSrc

Name

```
HDVICP20_TI_VDMA_ UpdateSrc ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateSrc (pBaseAddr, uiVal)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiVal	Source address

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the source address parameter in VDMA descriptor

14.32 HDVICP20_TI_VDMA_ UpdateYOffset

Name

```
HDVICP20_TI_VDMA_ UpdateYOffset ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateYOffset (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through

	vDMA engine.
usVal	Value for Y offset

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the Y offset parameter in VDMA descriptor

14.33 HDVICP20_TI_VDMA_ UpdateXOffset

Name

```
HDVICP20_TI_VDMA_ UpdateXOffset()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateXOffset (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
usVal	Value for X offset

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the X offset in VDMA descriptor

14.34 HDVICP20_TI_VDMA_ UpdateDstPitch

Name

```
HDVICP20_TI_VDMA_ UpdateDstPitch ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateDstPitch (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data

	through vDMA engine.
usVal	Value for destination pitch

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the destination pitch parameter in VDMA descriptor

14.35 HDVICP20_TI_VDMA_ UpdateDstPitchNonDet

Name

```
HDVICP20_TI_VDMA_ UpdateDstPitchNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateDstPitchNonDet (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
usVal	Value for destination pitch

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the destination pitch of a non-deterministic VDMA descriptor

14.36 HDVICP20_TI_VDMA_ UpdateDstNonDet

Name

```
HDVICP20_TI_VDMA_ UpdateDstNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateDstNonDet (pBaseAddr, uiVal)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data

	through vDMA engine.
uiVal	Destination address

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the destination address of a non-deterministic VDMA transfer descriptor

14.37 HDVICP20_TI_VDMA_ UpdateSrcNonDet

Name

```
HDVICP20_TI_VDMA_ UpdateSrcNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateSrcNonDet (pBaseAddr, uiVal)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
uiVal	Source address

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the source address of a non-deterministic transfer descriptor

14.38 HDVICP20_TI_VDMA_ UpdateYOffsetNonDet

Name

```
HDVICP20_TI_VDMA_ UpdateYOffsetNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateYOffsetNonDet (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
usVal	Value for Y offset

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to update the Y offset in a non-deterministic VDMA transfer descriptor

14.39 HDVICP20_TI_VDMA_ UpdateXOffsetNonDet

Name

```
HDVICP20_TI_VDMA_ UpdateXOffsetNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ UpdateXOffsetNonDet (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
usVal	Value for X offset

Outputs

None

Return Value

None

Description

Preprocessor macro to update the X offset in a non-deterministic transfer descriptor

14.40 HDVICP20_TI_VDMA_SetFirstFlagNonDet

Name

```
HDVICP20_TI_VDMA_ SetFirstFlagNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ SetFirstFlagNonDet (pBaseAddr)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to set first flag of non deterministic transfer descriptor

14.41 HDVICP20_TI_VDMA_ SetLastFlagNonDet

Name

```
HDVICP20_TI_VDMA_ SetLastFlagNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ SetLastFlagNonDet (pBaseAddr)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to set last flag of non deterministic transfer descriptor

14.42 HDVICP20_TI_VDMA_ ResetFirstFlagNonDet

Name

```
HDVICP20_TI_VDMA_ ResetFirstFlagNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ ResetFirstFlagNonDet (pBaseAddr)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to reset first flag of non deterministic transfer descriptor

14.43 HDVICP20_TI_VDMA_ ResetLastFlagNonDet

Name

```
HDVICP20_TI_VDMA_ ResetLastFlagNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_ ResetLastFlagNonDet (pBaseAddr)
(
     pU32  pBaseAddr,
     U32   uiVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.

Outputs

None

Return Value

Not applicable as this is macro.

Description

Preprocessor macro to reset last flag of non deterministic transfer descriptor

14.44 HDVICP20_TI_VDMA_TurnOffPaddingDet

Name

```
HDVICP20_TI_VDMA_TurnOffPaddingDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_TurnOffPaddingDet(baseAddr, val)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
usVal	Value for padding flag

Outputs

None

Return Value

None

Description

Preprocessor macro to update padding flag in a deterministic transfer VDMA descriptor

14.45 HDVICP20_TI_VDMA_TurnOffPaddingNonDet

Name

```
HDVICP20_TI_VDMA_TurnOffPaddingNonDet ()
```

Synopsis

```
HDVICP20_TI_VDMA_TurnOffPaddingNonDet (pBaseAddr, usVal)
(
     pU16  pBaseAddr,
     U16   usVal
)
```

Arguments

Inputs

pBaseAddr	Pointer to a transfer descriptor for the current transfer of data through vDMA engine.
usVal	Value for padding flag

Outputs

None

Return Value

Not applicable as this is macro

Description

Preprocessor macro to update the padding flag in a non-deterministic transfer VDMA descriptor

14.46 VDMA API Usage Example

This section illustrates VDMA API example usage to program deterministic and non-deterministic transfers in the VDMA.

```
   if ( (HDVICP20_TI_VDMA_Open
    (pVDMAParamStruct, pVDMAStatusStruct) == TRFEXCEEDMAX)
                   report error and return;

 /*-------------------------------------------------------*/
  /* Deterministic Transfer Example                       */
 /*-------------------------------------------------------*/

 /*-------------------------------------------------------*/
/* Deterministic transfers are programmed once at frame  */
 /* level                                                 */
/*-------------------------------------------------------*/

   HDVICP20_TI_VDMA_Configure1DCopyDet
```

```
    (
          pVDMAStatusStruct,
          uiGroupId,
          uiTransferId,
          uiDirection,
          pubSourceAddress,
          pubDestinationAddress,
          uiSize
    );

  /* The group transfer can be triggered either by using    */
  /* the HDVICP20_TI_VDMA_TriggerGroupDet() API or through  */
  /* programming the SBH to directly trigger the VDMA       */

  /* Use the following API to ascertain the completion      */
  /* of the group transfer                                  */
    HDVICP20_TI_VDMA_Wait(uiGroupId);

  /* Clear the status on completion                         */
    HDVICP20_TI_VDMA_Clear(uiGroupId);


  /*--------------------------------------------------------*/
  /* Non-Deterministic Transfer Example                     */
  /*--------------------------------------------------------*/

  /*--------------------------------------------------------*/
  /* Prepare non-deterministic transfer descriptor          */
  /*--------------------------------------------------------*/

    HDVICP20_TI_VDMA_PrepTrfDesc1DCopyNonDetShort
    (
        puiDescBufferAddress,
        uiGroupId,
        uiFirstFlag,
        uiLastFlag,
        uiDirection,
        pubSourceAddress,
        pubDestinationAddress,
        uiSize
    );

  /*--------------------------------------------------------*/
  /* Push the descriptor to VDMA memory space               */
  /*--------------------------------------------------------*/

    HDVICP20_TI_VDMA_PushTrfDescNonDet
    (
       puiTrfDesc,
       ubDescLength,
       ubNumTrfDescriptors
    );

  /* Use the following API to ascertain the completion      */
  /* of the group transfer                                  */
    HDVICP20_TI_VDMA_Wait(uiGroupId);

  /* Clear the status on completion                         */
```

```
HDVICP20_TI_VDMA_Clear(uiGroupId);
```

15. Mailbox Functions

The IVA-HD Mailbox is a module that enables inter-processor communication. It supports 3 users and 6 mailboxes (FIFOs) - each FIFO capable of holding upto four 32-bit messages. Sending and receiving messages could be either polling-based or interrupt-based.

15.1 List of functions

1. HDVICP20_TI_MBx_SoftReset
2. HDVICP20_TI_MBx_IRQInterruptEnable
3. HDVICP20_TI_MBx_IRQInterruptDisable
4. HDVICP20_TI_MBx_GetIRQStatus
5. HDVICP20_TI_MBx_ClearIRQStatus
6. HDVICP20_TI_MBx_GetMessageStatus
7. HDVICP20_TI_MBx_GetFIFOStatus
8. HDVICP20_TI_MBx_WriteMessage
9. HDVICP20_TI_MBx_ReadMessage

15.2 HDVICP20_TI_MBx_SoftReset

Name

`HDVICP20_TI_MBx_SoftReset()`

Synopsis

```
void HDVICP20_TI_MBx_SoftReset(U32 *puiMBxAddrBase)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address

Outputs

None

Return Value

None

Description

This API fills up the HWA SyncBox's common configuration parameters (independent of task) in the HWAParams structure.

15.3 HDVICP20_TI_MBx_IRQInterruptEnable

Name

```
HDVICP20_TI_MBx_IRQInterruptEnable()
```

Synopsis

```
void HDVICP20_TI_MBx_IRQInterruptEnable
(
     U32 *puiMBxAddrBase,
     U8  uiUserId,
     U8  uiMailBoxId,
     U8  uiNewMessageEnable,
     U8  uiNotFullEnable
)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
uiUserId	This argument specifies the user.
uiMailBoxId	This argument specifies the mailbox.
uiNewMessageEnable	Flag to enable interrupt to the user on generation of a new message. Set 1 to enable the interrupt.
uiNotFullEnable	Flag to enable interrupt to the user when mailbox is not full. Set 1 to enable the interrupt.

Outputs

None

Return Value

None

Description

This API enables interrupts for the specified user on the specified mailbox on arrival of a new message and/or when the mailbox is not full.

15.4 HDVICP20_TI_MBx_IRQInterruptDisable

Name

```
HDVICP20_TI_MBx_IRQInterruptDisable()
```

Synopsis

```
void HDVICP20_TI_MBx_IRQInterruptDisable
(
     U32 *puiMBxAddrBase,
     U8  uiUserId,
     U8  uiMailBoxId,
     U8  uiNewMessageDisable,
     U8  uiNotFullDisable
)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
uiUserId	This argument specifies the user.
uiMailBoxId	This argument specifies the mailbox.
uiNewMessageDisable	Flag to disable interrupt to the user on generation of a new message. Set 1 to disable the interrupt.
uiNotFullDisable	Flag to disable interrupt to the user when mailbox is not full. Set 1 to disable the interrupt.

Outputs

None

Return Value

None

Description

This API disables interrupts for the specified user on the specified mailbox on arrival of a new message and/or when the mailbox is not full.

15.5 HDVICP20_TI_MBx_GetIRQStatus

Name

```
HDVICP20_TI_MBx_IRQInterruptDisable()
```

Synopsis

```
void HDVICP20_TI_MBx_IRQInterruptDisable
(
     U32 *puiMBxAddrBase,
     U8  uiUserId,
     U8  uiMailBoxId,
     U8  uiNewMessageDisable,
     U8  uiNotFullDisable
)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
uiUserId	This argument specifies the user.
uiMailBoxId	This argument specifies the mailbox.
uiNewMessageDisable	Flag to disable interrupt to the user on generation of a new message. Set 1 to disable the interrupt.
uiNotFullDisable	Flag to disable interrupt to the user when mailbox is not full. Set 1 to disable the interrupt.

Outputs

None

Return Value

None

Description

This API disables interrupts for the specified user on the specified mailbox on arrival of a new message and/or when the mailbox is not full.

15.6 HDVICP20_TI_MBx_ClearIRQStatus

Name

```
HDVICP20_TI_MBx_ClearIRQStatus()
```

Synopsis

```
void HDVICP20_TI_MBx_ClearIRQStatus
(
     U32 *puiMBxAddrBase,
     U8  ubUserId,
     U8  uiMailBoxId,
     U8  usIRQClear,
     U8  usNotFullClear
)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
ubUserId	This argument specifies the user.
uiMailBoxId	This argument specifies the mailbox.
usIRQClear	Flag to clear interrupt to the user on generation of a new message. Set 1 to clear the interrupt.
usNotFullClear	Flag to clear interrupt to the user when mailbox is not full. Set 1 to clear the interrupt.

Outputs

None

Return Value

None

Description

This API clears interrupts for the specified user on the specified mailbox on arrival of a new message and/or when the mailbox is not full.

15.7 HDVICP20_TI_MBx_GetMessageStatus

Name

```
HDVICP20_TI_MBx_GetMessageStatus()
```

Synopsis

```
U8  HDVICP20_TI_MBx_GetMessageStatus
(
     U32 *puiMBxAddrBase,
     U8  ubMBxId
)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
ubMBxId	This argument specifies the mailbox.

Outputs

None

Return Value

Returns the number of unread messages in the mailbox.

Description

This API is used to read the number of unread messages in the mailbox.

15.8 HDVICP20_TI_MBx_GetFIFOStatus

Name

```
HDVICP20_TI_MBx_GetFIFOStatus()
```

Synopsis

```
U8  HDVICP20_TI_MBx_GetFIFOStatus
(
     U32 *puiMBxAddrBase,
     U8  ubMBxId
)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
ubMBxId	This argument specifies the mailbox.

Outputs

None

Return Value

Returns the FIFO "Full" status
0: Not full
1: Full

Description

This API is used to read FIFO "Full" status.

15.9 HDVICP20_TI_MBx_WriteMessage

Name

```
HDVICP20_TI_MBx_WriteMessage()
```

Synopsis

```
U8 HDVICP20_TI_MBx_WriteMessage
(
     U32 *puiMBxAddrBase,
```

```
        U8  uiMailBoxId,
        U32 uiMessage
    )
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
uiMailBoxId	This argument specifies the mailbox.
uiMessage	32-bit message to be written to the mailbox.

Outputs

None

Return Value

Returns whether message posting in MailBox is successful or not. If posting is unsuccessful, the API won't write the message into MailBox.
* 0: Not Full (and write messsage is Successful).
* 1: Full (and write message Failed)

Description

This API writes a 32-bit message to the specified mailbox.

15.10 HDVICP20_TI_MBx_ReadMessage

Name

```
HDVICP20_TI_MBx_ReadMessage()
```

Synopsis

```
U32 HDVICP20_TI_MBx_ReadMessage
(
     U32 *puiMBxAddrBase,
     U8  uiMailBoxId
)
```

Arguments

Inputs

puiMBxAddrBase	Pointer containing MailBox Register base address
uiMailBoxId	This argument specifies the mailbox.

Outputs

None

Return Value

32-bit message read from the specified mailbox.

Description

This API reads a 32-bit message from the specified mailbox.

15.11 Mailbox API Usage Example

This section illustrates the use of Mailbox APIs for sending and receiving messages.

```
/*-------------------------------------------------------*/
/* Example 1: Sending a message                          */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Check if FIFO is full?                                */
/*-------------------------------------------------------*/
     if ( (HDVICP20_TI_MBx_GetFIFOStatus(MAILBOX,ubMBxId)) == 1 )
     {

/*-------------------------------------------------------*/
/* Enable interrupt to notify when FIFO is not full      */
/*-------------------------------------------------------*/

          HDVICP20_TI_MBx_IRQInterruptEnable
          (
              MAILBOX,
              uiUserId,
              uiMailBoxId,
              uiNewMessageEnable,
              uiNotFullEnable    /* Should be 1 */
          );

/*-------------------------------------------------------*/
/* Wait for interrupt. The user can perform other tasks  */
/* during this time.                                     */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* When interrupt is received…                           */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Write message to FIFO                                 */
/*-------------------------------------------------------*/

          HDVICP20_TI_MBx_WriteMessage
          (
              MAILBOX,
              uiMailBoxId,
              uiMessage
          );

/*-------------------------------------------------------*/
/* Clear the interrupt                                   */
/*-------------------------------------------------------*/

          HDVICP20_TI_MBx_ClearIRQStatus
          (
             MAILBOX,
             uiUserId,
             uiMailBoxId,
             usIRQClear,
```

```
            usNotFullClear   /* Should be 1 */
        );

    }
    else
    {
/*-------------------------------------------------------*/
/* FIFO is not full. So, write message to FIFO           */
/*-------------------------------------------------------*/

        HDVICP20_TI_MBx_WriteMessage
        (
            MAILBOX,
            uiMailBoxId,
            uiMessage
        );
    }

/*-------------------------------------------------------*/
/* END OF EXAMPLE 1                                      */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Example 2: Receiving a message                        */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Enable interrupt to notify when a new message arrives */
/*-------------------------------------------------------*/

        HDVICP20_TI_MBx_IRQInterruptEnable
        (
            MAILBOX,
            uiUserId,
            uiMailBoxId,
            uiNewMessageEnable,  /* Should be 1 */
            uiNotFullEnable
        );

/*-------------------------------------------------------*/
/* Wait for interrupt. The user can perform other tasks  */
/* during this time.                                     */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* When interrupt is received…                           */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/* Check if number of messages not equal to zero?        */
/*-------------------------------------------------------*/

    if ( (HDVICP20_TI_MBx_GetMessageStatus(MAILBOX,ubMBxId)) != 0 )
     {
```

```
/*-------------------------------------------------------*/
/* Read the message                                      */
/*-------------------------------------------------------*/

     message = HDVICP20_TI_MBx_ReadMessage(MAILBOX,uiMailBoxId);

     }
     else
     {

/*-------------------------------------------------------*/
/* Clear the interrupt                                   */
/*-------------------------------------------------------*/

          HDVICP20_TI_MBx_ClearIRQStatus
          (
             MAILBOX,
             uiUserId,
             uiMailBoxId,
             usIRQClear,      /* Should be 1 */
             usNotFullClear
          );
     }

/*-------------------------------------------------------*/
/* END OF EXAMPLE 2                                      */
/*-------------------------------------------------------*/
```

THIS PAGE IS INTENTIONALLY LEFT BLANK

16. SMSET Functions

The IVA-HD SMSET is a module which keeps trace of the System hardware events such as HWA Start and Stop events etc and Software Messages which is useful in debugging and profiling the system. SMSET can be programmed to selectively trace a few system events among 52 system events because of the limited trace buffer inside the Hardware.

16.1 List of functions

1. HDVICP20_TI_SMSET_GetOwnership
2. HDVICP20_TI_SMSET_Enable
3. HDVICP20_TI_SMSET_ResetStartAddress
4. HDVICP20_TI_SMSET_EnableAllEvents
5. HDVICP20_TI_SMSET_StartTrace

16.2 HDVICP20_TI_SMSET_GetOwnership

Name

```
HDVICP20_TI_SMSET_GetOwnership()
```

Synopsis

```
S32 HDVICP20_TI_SMSET_GetOwnership(void)
```

Arguments

Inputs

None

Outputs

None

Return Value

A Zero value implies successful operation and a non-zero value implies failure.

Description

This API acquires the ownership of the SMSET.

16.3 HDVICP20_TI_SMSET_Enable

Name

```
HDVICP20_TI_SMSET_Enable()
```

Synopsis

```
void HDVICP20_TI_SMSET_Enable(void)
```

Arguments

Inputs

None

Outputs

None

Return Value

None

Description

This API enables the module of SMSET.

16.4 HDVICP20_TI_SMSET_ResetStartAddress

Name

```
HDVICP20_TI_SMSET_ ResetStartAddress ()
```

Synopsis

```
void HDVICP20_TI_SMSET_ResetStartAddress
   (
     U32  uiTraceStartAddress
   )
```

Arguments

Inputs

uiTraceStartAddress	This argument specifies the start address of the Trace buffer.

Outputs

None

Return Value

None.

Description

This API to the SMSET MMR where the start of Trace Buffer is.

16.5 HDVICP20_TI_SMSET_EnableAllEvents

Name

```
HDVICP20_TI_SMSET_EnableAllEvents()
```

Synopsis

```
void HDVICP20_TI_SMSET_EnableAllEvents(void)
```

Arguments

Inputs

None

Outputs

None

Return Value

None

Description

This API enables all the 52 System events traced by SMSET.

16.6 HDVICP20_TI_SMSET_StartTrace

Name

```
HDVICP20_TI_SMSET_StartTrace()
```

Synopsis

```
void HDVICP20_TI_SMSET_StartTrace( U32 uiSamplingWindow)
```

Arguments

Inputs

uiSamplingWindow	This argument specifies the Sampling window for the SMSET Trace.

Outputs

None

Return Value

None

Description

This API makes the SMSET to start the trace mechanism with the sampling window.

THIS PAGE IS INTENTIONALLY LEFT BLANK

17. HOST (M3) side APIs

This chapter describes some of the VDMA and DM APIs that may be required on host/M3 side for code and dat loading from DDR to iCONT1/iCONT2's ITCM/DTCM memories. Generally .

17.1 List of functions

1. HDVICP20_TI_VDMA_Open_Extmem_SL2
2. HDVICP20_TI_VDMA_Prepare_Extmem_SL2
3. HDVICP20_TI_VDMA_Trigger_Extmem_SL2
4. HDVICP20_TI_VDMA_WaitForData_Extmem_SL2
5. HDVICP20_TI_DM_Prepare_SL2_TCM
6. HDVICP20_TI_DM_Trigger_SL2_TCM
7. HDVICP20_TI_DM_WaitForData_SL2_TCM

17.2 HDVICP20_TI_VDMA_Open_Extmem_SL2

Name

```
HDVICP20_TI_VDMA_Open_Extmem_SL2()
```

Synopsis

```
void HDVICP20_TI_VDMA_Open_Extmem_SL2
(
     U32 configBase,
     U8 VDMA_TBA
)
```

Arguments

Inputs

configBase	Config base address of IVAHD
VDMA_TBA	Tiler base address for VDMA

Outputs

None

Return Value

None

Description

This function opens vDMA for DDR to SL2 transfer. In this function we do following steps
- Wait till all transactions are completed by the vDMA
- Do software reset of vDMA
- Put the tiler base address
- Put end of group descriptions
- Number of contexts to be used for synchronous transfers
- Group definition for the descriptors

17.3 HDVICP20_TI_VDMA_Prepare_Extmem_SL2

Name

```
HDVICP20_TI_VDMA_Prepare_Extmem_SL2()
```

Synopsis

```
void HDVICP20_TI_VDMA_Prepare_Extmem_SL2
(
     void* dst,
     void* src,
     U32   size,
     U8    Dir,
     U32   Id,
     U32   configBase
)
```

Arguments

Inputs

Dst	Destiniation address of the transfer. If this address is pointing to external memory and then it is assumed to be physical address seen by vDMA
Src	Source address of the transfer. If this address is pointing to external memory and then it is assumed to be physical address seen by vDMA
Size	Size of data that needs to be transfered
Dir	Direction of the transfer. Possible values are DDRTOSL2 or SL2TODDR
Id	vDMA group ID of the transfer
configBase	Config base address of IVAHD

Outputs

None

Return Value

None

Description

This function prepares vDMA descriptors for given transfer. The assumption made in this routine is group id for which user asked to prepre the descriptors is idle. i.e no transfer is on going in this group id. The sequence of steps that are followed in this function are

- Determin the src and dest address depending on drection
- Get the physical address of DDR address
- If deterministic, obtain the address in deterministic memory that needs to be updated.
- write the descriptor in the memory

17.4 HDVICP20_TI_VDMA_Trigger_Extmem_SL2

Name

```
HDVICP20_TI_VDMA_Trigger_Extmem_SL2()
```

Synopsis

```
void HDVICP20_TI_VDMA_Trigger_Extmem_SL2(U32 configBase, U32 groupID)
```

Arguments

Inputs

groupID	vDMA group ID of the transfer
configBase	Config base address of IVAHD

Outputs

None

Return Value

None

Description

This function triggers the VDMA transfers (DDR to SL2 or SL2 to DDR) that are prepared earlier.

17.5 HDVICP20_TI_VDMA_WaitForData_Extmem_SL2

Name

HDVICP20_TI_VDMA_WaitForData_Extmem_SL2

Synopsis

```
void HDVICP20_TI_VDMA_WaitForData_Extmem_SL2(U32 Id, U32 configBase)
```

Arguments

Inputs

Id	vDMA group ID of the transfer
configBase	Config base address of IVAHD

Outputs

None

Return Value

None

Description

This function checks the status of the group in vDMA registers, and if the group is busy then it waits for the group to complete. Once the transfer is completed, It clears the corresponding status bit.

17.6 HDVICP20_TI_DM_Prepare_SL2_TCM

Name

HDVICP20_TI_DM_Prepare_SL2_TCM()

Synopsis

```
U32 HDVICP20_TI_DM_Prepare_SL2_TCM
(
     void* dst,
     void* src,
     U32   size,
     U8    Dir,
     U8    Id,
     U32   configBase,
     U8    ubContextId
 )
```

Arguments

Inputs

Dst	Destination address (SL2/TCM address) for the data transfer
Src	Source address (SL2/TCM address) for the data transfer
Size	Size of the data in bytes to be transferred
Dir	Direction of transfer (SL2 to TCM or TCM to SL2)
Id	id of iCont to be triggered (iCONT1 or iCONT2)
configBase	IVAHD Config base address
ubContextId	DM context or logical channel ID (0 to 3)

Outputs

None

Return Value

32 bit word which needs to put in DM registers

Description

This function takes the src, dst etc info and prepares a 32 bit word for SL2 to ITCM/DTCM or vice versa data trasfer. This word needs to be written into DM trigger register for the transfer to be initiated.

17.7 HDVICP20_TI_DM_Trigger_SL2_TCM

Name

```
HDVICP20_TI_DM_Trigger_SL2_TCM()
```

Synopsis

```
void HDVICP20_TI_DM_Trigger_SL2_TCM
(
     U32 data,
     U8  Id,
     U32 configBase,
     U8  ubContextId
)
```

Arguments

Inputs

Data	Trigger data transfer word to be written to DM register
Id	id of iCont to be triggered (iCONT1 or iCONT2)
configBase	IVAHD Config base address
ubContextId	DM context or logical channel ID (0 to 3)

Outputs

None

Return Value

None

Description

This function triggers the DM transfer from SL2 to ITCM/DTCM of iCONT1/iCONT2 by writing trigger word to DM context register. Once trigger word is written the transfer will automatically start.

17.8 HDVICP20_TI_DM_WaitForData_SL2_TCM

Name

```
HDVICP20_TI_DM_WaitForData_SL2_TCM()
```

Synopsis

```
void HDVICP20_TI_DM_WaitForData_SL2_TCM
(
     U8 Id,
     U32 configBase,
     U8 ubContextId
)
```

Arguments

Inputs

Id	id of iCont to be triggered (iCONT1 or iCONT2)
configBase	IVAHD Config base address
ubContextId	DM context or logical channel ID (0 to 3)

Outputs

None

Return Value

None

Description

This function waits for the DM transfer from SL2 to ITCM/DTCM of iCONT1/iCONT2 to get completed

18. Appendix A

18.1 Usage of Compute IP APIs

This appendix illustrates the usage of the APIs provided for the Compute IPs (CALC3, ECD3, iLF3, iME3, IPE3 and MC3) in a codec implementation.
The following is an illustration of the decoder flow to decode a frame and the usage of IVA-HD APIs in it. Note that the APIs are to be used in Step 3. Steps 1 and 2 are initial steps executed on either of the iCONTs. During Step 4, all HWAs are fully functional and processing is all in hardware (based on the software programming in Step 3) except the iCONT tasks.

DecodeFrameIVAHD / Start of IVA-HD processing */*
{

1. *Initialize the codec variables and arrays with their default values which will be used for setting up H/W IP's and the decoder. Preliminary initializations even before parsing first slice header happens here.*
2. *Parse the slice headers.*
3. ***Initialize the HWAs using the APIs provided in the IVA-HD API Library.***
4. *Trigger the macroblock processing pipeline. This "macroblock processing loop" decodes the whole frame.*

}

The following is an illustration of the encoder flow to encode a frame and the usage of IVA-HD APIs in it. Note that the APIs are to be used in Step 3. Steps 1 and 2 are initial steps executed on either of the iCONTs. During Step 4, all HWAs are fully functional and processing is all in hardware (based on the software programming in Step 3) except the iCONT tasks.

EncodeFrameIVAHD / Start of IVA-HD processing */*
{

1. *Initialize the codec parameter variables and arrays with their initialization values which will be used for setting up H/W IP's and the encoder.*
2. *Encode the sequence/picture/slice headers.*
3. ***Initialize the HWAs using the APIs provided in the IVA-HD API Library.***
4. *Trigger the macroblock peorcessing pipeline. This "macroblock processing loop" encodes the whole frame.*

}