The Fastest Verification

# ZeBu ™

# API

Wed Jan 9 15:18:03 2013

# Copyright Notice

## Proprietary Information



## Right to Copy Documentation

The license agreement with EVE permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:
*This document is duplicated with the permission of EVE, for the exclusive use of _____ and its employees. This is copy number ____.*

## Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader s responsibility to determine the applicable regulations and to comply with them.

## Disclaimer

EVE AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

# Contents


**1 Namespace Index** **5**

1.1 Namespace List . . . . . 5

**2 Class Index** **7**

2.1 Class List . . . . . 7

**3 File Index** **9**

3.1 File List . . . . . 9

**4 Namespace Documentation** **11**

4.1 MIPI_CSI Namespace Reference . . . . . 11

4.2 ZEBU_IP Namespace Reference . . . . . 12

4.3 ZEBU_IP::MIPI_CSI Namespace Reference . . . . . 13

**5 Class Documentation** **17**

5.1 ZEBU_IP::MIPI_CSI::CCIStatusRegister_t Union Reference . . . . . 17

5.2 ZEBU_IP::MIPI_CSI::CSI Class Reference . . . . . 18

5.3 ZEBU_IP::MIPI_CSI::CSIEventStatus_t Union Reference . . . . . 54

**6 File Documentation** **57**

6.1 CSI.hh File Reference . . . . . 57

6.2 CSI_Struct.hh File Reference . . . . . 58

# Chapter 1

# Namespace Index

## 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

MIPI_CSI (Name Space Containing MIPI_CSI Transactor description ) . . . . . 11
ZEBU_IP (Name Space Containing all the Zebu Transactor ) . . . . . 12
ZEBU_IP::MIPI_CSI . . . . . 13

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ZEBU_IP::MIPI_CSI::CCIStatusRegister_t (Structure to handle received event from the I2C BUS nion to view status of CSI CCI Call Back ) . . . . . 17
ZEBU_IP::MIPI_CSI::CSI (CSI Transactor Class definition ) . . . . . 18
ZEBU_IP::MIPI_CSI::CSIEventStatus_t (Union to handle Status Of CSI Controller ) . . . . . 54

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

CSI.hh (Include file containing C++ API for MIPI_CSI Transactor ) . . . . . 57
CSI_Struct.hh (Include file containing C++ API Types used by MIPI_CSI Transactor ) . . . . . 58

# Chapter 4

# Namespace Documentation

## 4.1 MIPI_CSI Namespace Reference

Name Space Containing MIPI_CSI Transactor description.

### 4.1.1 Detailed Description

Name Space Containing MIPI_CSI Transactor description. Name Space Containing CSI Transactor description.

## 4.2 ZEBU_IP Namespace Reference

Name Space Containing all the Zebu Transactor.

### Namespaces

- namespace MIPI_CSI

### 4.2.1 Detailed Description

Name Space Containing all the Zebu Transactor.

# 4.3 ZEBU_IP::MIPI_CSI Namespace Reference

## Classes

- class CSI

  *CSI Transactor Class definition.*

- union CSIEventStatus_t

  *Union to handle Status Of CSI Controller.*

- union CCIStatusRegister_t

  *Structure to handle received event from the I2C BUS nion to view status of CSI CCI Call Back.*

## Typedefs

- typedef int32_t CSI_pixel_t
- typedef uint16_t Frame_pixel_t
- typedef uint16_t Frame_line_t

## Enumerations

- enum Pixel_Format_t {
  Pixel_Format_NotSet, RGBFFF, RGB888, RGB666,
  RGB565, RGB555, RGB444, YUV422_10,
  YUV422_8, YUV420_10_CSPS, YUV420_8_CSPS, YUV420_8_legacy,
  YUV420_10, YUV420_8, RAW16, RAW14,
  RAW12, RAW10, RAW8, RAW7,
  RAW6 }

  *Enum for Pixel Format Definition.*

- enum Pixel_File_Format_t {
  File_Format_NotSet, FILE_RGB16, FILE_RGB8, FILE_YUV422_8,
  FILE_RAW16, FILE_RAW8 }

  *Enum for Pixel File Format Pixel Mapping.*

- enum SensorMode_t {
  _SensorMode_NotSet, RGB, YUV, RAW,
  Colorbar }

  *Enum for SensorMode Definition.*

- enum CSI_Packet_Name_t {
  P_Frame_Start, P_Frame_End, P_Line_Start, P_Line_End,
  P_Generic_Short_Packet_Code1, P_Generic_Short_Packet_Code2, P_Generic_Short_Packet_-
  Code3, P_Generic_Short_Packet_Code4,

P_Generic_Short_Packet_Code5, P_Generic_Short_Packet_Code6, P_Generic_Short_Packet_-Code7, P_Generic_Short_Packet_Code8,

P_Null, P_Blanking_Data, P_Embedded_8bit_non_Image_Data, P_YUV420_8bit,

P_YUV420_10bit, P_Legacy_YUV420_8bit, P_YUV420_8bit_Chroma_Shifted_Pixel_Sampling, P_YUV420_10bit_Chroma_Shifted_Pixel_Sampling,

P_YUV422_8bit, P_YUV422_10bit, P_RGB444, P_RGB555,

P_RGB565, P_RGB666, P_RGB888, P_RAW6,

P_RAW7, P_RAW8, P_RAW10, P_RAW12,

P_RAW14, P_User_Defined_8bit_Data_Type1, P_User_Defined_8bit_Data_Type2, P_User_-Defined_8bit_Data_Type3,

P_User_Defined_8bit_Data_Type4, P_User_Defined_8bit_Data_Type5, P_User_Defined_8bit_-Data_Type6, P_User_Defined_8bit_Data_Type7,

P_User_Defined_8bit_Data_Type8 }

*Enum for CSI Packet Name.*

### 4.3.1 Typedef Documentation

#### 4.3.1.1 typedef int32_t ZEBU_IP::MIPI_CSI::CSI_pixel_t

#### 4.3.1.2 typedef uint16_t ZEBU_IP::MIPI_CSI::Frame_line_t

#### 4.3.1.3 typedef uint16_t ZEBU_IP::MIPI_CSI::Frame_pixel_t

### 4.3.2 Enumeration Type Documentation

#### 4.3.2.1 enum ZEBU_IP::MIPI_CSI::CSI_Packet_Name_t

Enum for CSI Packet Name.

**Enumerator:**

***P_Frame_Start***

***P_Frame_End***

***P_Line_Start***

***P_Line_End***

***P_Generic_Short_Packet_Code1***

***P_Generic_Short_Packet_Code2***

***P_Generic_Short_Packet_Code3***

***P_Generic_Short_Packet_Code4***

***P_Generic_Short_Packet_Code5***

***P_Generic_Short_Packet_Code6***

***P_Generic_Short_Packet_Code7***

***P_Generic_Short_Packet_Code8***

***P_Null***

***P_Blanking_Data***

***P_Embedded_8bit_non_Image_Data***

***P_YUV420_8bit***

***P_YUV420_10bit***

***P_Legacy_YUV420_8bit***

***P_YUV420_8bit_Chroma_Shifted_Pixel_Sampling***

***P_YUV420_10bit_Chroma_Shifted_Pixel_Sampling***

***P_YUV422_8bit***

***P_YUV422_10bit***

***P_RGB444***

***P_RGB555***

***P_RGB565***

***P_RGB666***

***P_RGB888***

***P_RAW6***

***P_RAW7***

***P_RAW8***

***P_RAW10***

***P_RAW12***

***P_RAW14***

***P_User_Defined_8bit_Data_Type1***

***P_User_Defined_8bit_Data_Type2***

***P_User_Defined_8bit_Data_Type3***

***P_User_Defined_8bit_Data_Type4***

***P_User_Defined_8bit_Data_Type5***

***P_User_Defined_8bit_Data_Type6***

***P_User_Defined_8bit_Data_Type7***

***P_User_Defined_8bit_Data_Type8***

#### 4.3.2.2 enum ZEBU_IP::MIPI_CSI::Pixel_File_Format_t

Enum for Pixel File Format Pixel Mapping.

**Enumerator:**

***File_Format_NotSet***

***FILE_RGB16*** RGB 16

***FILE_RGB8*** RGB 8

***FILE_YUV422_8*** YUV 422 8 bits

***FILE_RAW16*** RAW 16

***FILE_RAW8*** RAW 8

#### 4.3.2.3 enum ZEBU_IP::MIPI_CSI::Pixel_Format_t

Enum for Pixel Format Definition.

**Enumerator:**

***Pixel_Format_NotSet***

***RGBFFF*** RGB FFF

***RGB888*** RGB 888

***RGB666*** RGB 666

***RGB565*** RGB 565

***RGB555*** RGB 555

***RGB444*** YUV 444

***YUV422_10*** YUV 422 10 bits

***YUV422_8*** YUV 422 8 bits

***YUV420_10_CSPS*** YUV 420 10 bits Chroma Shifted Pixel Sampling

***YUV420_8_CSPS*** YUV 420 8 bits Chroma Shifted Pixel Sampling

***YUV420_8_legacy*** YUV 420 8 bits Legacy

***YUV420_10*** YUV 420 10 bits

***YUV420_8*** YUV 420 8 bits

***RAW16*** RAW 16

***RAW14*** RAW 14

***RAW12*** RAW 12

***RAW10*** RAW 10

***RAW8*** RAW 8

***RAW7*** RAW 7

***RAW6*** RAW 6

#### 4.3.2.4 enum ZEBU_IP::MIPI_CSI::SensorMode_t

Enum for SensorMode Definition.

**Enumerator:**

***_SensorMode_NotSet***

***RGB*** RGB

***YUV*** YUV

***RAW*** RAW

***Colorbar*** Colorbar

# Chapter 5

# Class Documentation

## 5.1 ZEBU_IP::MIPI_CSI::CCIStatusRegister_t Union Reference

Structure to handle received event from the I2C BUS nion to view status of CSI CCI Call Back.

`#include <CSI_Struct.hh>`

### Public Attributes

- bool MONITORING_ENABLE:1
- bool CALLBACK_ENABLE:1

### 5.1.1 Detailed Description

Structure to handle received event from the I2C BUS nion to view status of CSI CCI Call Back.

### 5.1.2 Member Data Documentation

#### 5.1.2.1 bool ZEBU_IP::MIPI_CSI::CCIStatusRegister_t::CALLBACK_ENABLE

Indicates if a CallBack is present

#### 5.1.2.2 bool ZEBU_IP::MIPI_CSI::CCIStatusRegister_t::MONITORING_ENABLE

Indicates if a CCI adr is monitored

The documentation for this union was generated from the following file:

- CSI_Struct.hh

## 5.2 ZEBU_IP::MIPI_CSI::CSI Class Reference

CSI Transactor Class definition.

`#include <CSI.hh>`

### Public Member Functions

- VS_SO_EXPORT CSI ()

  *MIPI_CSI Constructor.*

- VS_SO_EXPORT ~CSI ()

  *MIPI_CSI Destructor.*

- void VS_SO_EXPORT setName (const char *name)

  *set name of MIPI_CSI transactor that will appear in all messages prefixes (for debug purpose)*

- const char *VS_SO_EXPORT getName (void)

  *Return the prefix message name (for debug purpose).*

- void VS_SO_EXPORT setDebugLevel (uint lvl)

  *Set Display Trace Level For Xtor.*

- bool VS_SO_EXPORT setLog (FILE *stream, bool stdoutDup=false)

  *set output stream for transactor debug*

- bool VS_SO_EXPORT setLog (char *fname, bool stdoutDup=false)

  *Create and Open log file.*

- bool VS_SO_EXPORT init (Board *zebu, const char *driverName)

  *Initialization of the transactor.*

- void VS_SO_EXPORT useZebuServiceLoop (bool state)

  *enable call of ZeBu service loop by CSI Xactor when activated, the transactor will call ZeBu Board::serviceLoop() with no arguments instead of the CSIServiceLoop() when the current operation cannot send/receive data to/from hardware See ZeBu API documentation to know more about the ZeBu service loop*

- void VS_SO_EXPORT useZebuServiceLoop (int(*zebuServiceLoopHandler)(void *context, int pending), void *context)

  *enable call of ZeBu service loop by CSI Xactor when activated, the transactor will call ZeBu Board::serviceLoop() with the specified arguments instead of the CSIServiceLoop() when the current operation cannot send/receive data to/from hardware See ZeBu API documentation to know more about the ZeBu service loop*

- void VS_SO_EXPORT useZebuServiceLoop (int(*zebuServiceLoopHandler)(void *context, int pending), void *context, const unsigned int portGroupNumber)

  *enable call of ZeBu service loop by CSI Xactor when activated, the transactor will call ZeBu Board::serviceLoop() with the specified arguments instead of the CSIServiceLoop() when the current operation cannot send/receive data to/from hardware See ZeBu API documentation to know more about the ZeBu service loop*

- void VS_SO_EXPORT setZebuPortGroup (const unsigned int grp)

  *Set the current CSI transactor instance ports group This method allows the user to attach the transactor to a specific ZeBu port group. Then calling Board::serviceLoop() on the specified group will allow the serviceLoop() method to handle the CSI ports.*

- void VS_SO_EXPORT registerUserCB (void(∗userCB)(void ∗context), void ∗context)

  *Register a function that will be called by the transactor instead of the CSI service loop when registered, the user callback will be called each time the current operation is unable to send send/receive data to/from ZeBu port.*

- void VS_SO_EXPORT CSIServiceLoop (int(∗servcieCB)(void ∗, int)=NULL, void ∗context=NULL)

  *Handle the current MIPI_CSI transactor instance incoming and pending messages This method check all the MIPI_CSI ports in order to send pending messages or receive incoming messages when possible. if the service callback pointer is NULL, the method exits right after having handled the messages. if a service callback is registered, it is called after each check of the ports. The pending argument will be set to 0 if no messages could be sent or received, otherwise it will be sent to a different value.*

- void VS_SO_EXPORT enableWatchdog (bool enable=true)

  *enable or disable CSI watchdogs Note that by default watchdogs are enabled*

- void VS_SO_EXPORT setTimeout (uint64_t ms)

  *set watchdogs timeout value*

- void VS_SO_EXPORT registerTimeoutCB (bool(∗timeoutCB)(void ∗context), void ∗context=NULL)

  *register a timeout callback register a function that will be called when watchdog timeout is trigerred if the timeout function return true, current blocking operation are aborted After each timeout, the trigerred watchdog timer is automatically rearmed*

- void VS_SO_EXPORT checkWatchdogs (void)

  *check all watchdogs Go through all Watchdog to check timeouts. If a timeout, the timeout callback is called if registered. if the timeout callback returns true or no callback is registered, the execution of blocking will stop*

- void VS_SO_EXPORT enableSequencer ()

  *Enable hardware Sequencer of the Bfm.*

- bool VS_SO_EXPORT disableSequencer ()

  *Disable hardware Sequencer of the Bfm.*

- bool VS_SO_EXPORT getSequencer ()

  *Return Sequencer status (enable or disable).*

- bool VS_SO_EXPORT runCycle (unsigned int nb_cycle)

  *run CSI_PPI_bytes clock during nb_cycle*

- void VS_SO_EXPORT usleep (int usec)

  *Pause the CSI testbench execution for the given number of Usec.*

- void VS_SO_EXPORT setSensorMode (SensorMode_t SensorMode)

*Define the mode of capture: RGB, YUV, RAW or Colorbar.*

- void VS_SO_EXPORT setVirtualChannelID (unsigned int VirtualChannelID)

  *Define the Virtual Channel Identifier.*

- SensorMode_t VS_SO_EXPORT getSensorMode ()

  *Return the mode of capture: RGB, YUV, RAW or Colorbar.*

- void VS_SO_EXPORT defineFrontPorchSync (Frame_pixel_t HFP, Frame_line_t VFP)

  *Define the Horizontal and Vertical Video Front Porch Profile.*

- void VS_SO_EXPORT getVideoTiming (float ∗VFP, float ∗FrameLength, float ∗VBP, float ∗HFP, float ∗LineLength, float ∗HBP)

  *Return the frame timing in us.*

- void VS_SO_EXPORT setVideoJitter (uint Hjitter_max, uint Vjitter_max, uint seed)

  *Set the maximum horizontal line jitter and the maximum vertical frame jitter.*

- void VS_SO_EXPORT useLineSyncPacket (bool enable)

  *Enable the transmission of the Line start / Line End CSI packets.*

- void VS_SO_EXPORT setPixelPacking (unsigned int nb_LineSplit, unsigned int delay=0)

  *Define the number of split CSI pixel packet.*

- unsigned int VS_SO_EXPORT getPixelPacking ()

  *Get the number of split CSI pixel packet.*

- float VS_SO_EXPORT getFrameRate ()

  *Get the frame rate.*

- void VS_SO_EXPORT setFrameRate (float FPS)

  *Define the frame rate.*

- float VS_SO_EXPORT getRealFrameRate ()

  *Return the Real frame rate from the CSI transactor current transmission.*

- void VS_SO_EXPORT defineRefClkFreq (float RefClkFreq)

  *Define the frequency of the Transcator Reference Clock.*

- bool VS_SO_EXPORT setCSIClkDivider (unsigned int CSIClkDivider)

  *set the CSI clock divider*

- unsigned int VS_SO_EXPORT getCSIClkDivider ()

  *Return the CSI clock divider.*

- float VS_SO_EXPORT getPPIByteClk_Freq ()

  *Return the frequency of PPI Byte clock.*

- float VS_SO_EXPORT getPPIByteClk_MinFreq ()

  *Return the minimum frequency of PPI Byte clock to achieve the required FPS.*

- bool VS_SO_EXPORT checkCSITxCharacteristics ()

  *check the CSI Timing setup to achieve the required FPS with the current image resolution*

- float VS_SO_EXPORT getVirtualPixelClkFreq ()

  *Return the equivalent pixelClock Frequency from the FPS and the image resolution.*

- void VS_SO_EXPORT getCSIBlankingDPHYPeriod (uint *LineBlanking, uint *FrameBlanking)

  *Return the line blanking period in number of cycles of the DPHY source clock.*

- void VS_SO_EXPORT getCSIBlankingTiming (float *LineBlanking, float *FrameBlanking)

  *Return the line blanking period in microseconds.*

- unsigned int VS_SO_EXPORT getHWinfo ()

  *Return information about hardware part of transactor.*

- void VS_SO_EXPORT getVideoMode (Frame_line_t *nb_lines, Frame_pixel_t *nb_pixels, Pixel_Format_t *VideoFormatOut, SensorMode_t *SensorMode)

  *Return the video parameter.*

- void VS_SO_EXPORT setInputFile (string file_path_name, Pixel_File_Format_t Pixel_File_Format, Frame_line_t nb_lines, Frame_pixel_t nb_pixels, bool loop_on_file=true)

  *Define the video file according to its format and resolution.*

- bool VS_SO_EXPORT setImageZoom (float Xzoom, float Yzoom)

  *Define the zoom to apply on the original file.*

- bool VS_SO_EXPORT setImageRotate (unsigned int Rotation)

  *Define the rotation to apply on the original file.*

- bool VS_SO_EXPORT setImageRegion (Frame_pixel_t Start_pixel, Frame_pixel_t Nb_Pixels, Frame_line_t Start_Line, Frame_line_t Nb_Lines)

  *Define the region area to send.*

- void VS_SO_EXPORT getImageRegion (Frame_pixel_t *Start_Pixel, Frame_pixel_t *Nb_Pixels, Frame_line_t *Start_Line, Frame_line_t *Nb_Lines)

  *Get the region area sended.*

- void VS_SO_EXPORT setTransform (Pixel_Format_t VideoFormatIn, Pixel_Format_t VideoFormatOut)

  *Defines the transformation to apply from the input file to the CSI packet.*

- bool VS_SO_EXPORT buildImage ()

  *Build the pixel Frame Buffer according to the current source and its transformation.*

- void VS_SO_EXPORT setColorbar (Pixel_Format_t Pixel_Format, Frame_line_t nb_lines, Frame_pixel_t nb_pixels)

  *Defines the colorbar parameters.*

- void VS_SO_EXPORT getColorbarParam (Pixel_Format_t *Pixel_Format, Frame_line_t *nb_lines, Frame_pixel_t *nb_pixels)

*Return the colorbar parameters.*

- bool VS_SO_EXPORT sendImage ()

  *Send the whole content of the pixel Frame Buffer.*

- bool VS_SO_EXPORT sendLine ()

  *Send a line of the pixel frame buffer.*

- bool VS_SO_EXPORT flushCSIPacket ()

  *Flush HW/SW Fifo pipe.*

- void VS_SO_EXPORT getCSIStatus (CSIEventStatus_t ∗CSIStatus)

  *Return the MIPI_CSI Video Status.*

- unsigned int VS_SO_EXPORT getFrameNumber (void)

  *Indicates the number of frame sent.*

- unsigned int VS_SO_EXPORT getLineInFrame (void)

  *Indicates the number of line sent in the current frame.*

- void VS_SO_EXPORT displayCSIStatus ()

  *Display the field of CSIEventStatus.*

- unsigned long long VS_SO_EXPORT getCurrentCycle (void)

  *Indicates the current cycle number.*

- bool VS_SO_EXPORT sendShortPacket (unsigned int DataID, uint16_t Data_0_1)

  *Send a CSI Short Packet.*

- bool VS_SO_EXPORT sendShortPacket (CSI_Packet_Name_t DataID, uint16_t Data_0_1)

  *Send a CSI Short Packet.*

- bool VS_SO_EXPORT sendLongPacket (unsigned int DataID, uint16_t WordCount, uint8_t ∗DataByte)

  *Send a CSI Long Packet.*

- bool VS_SO_EXPORT sendLongPacket (CSI_Packet_Name_t DataID, uint16_t WordCount, uint8_t ∗DataByte)

  *Send a CSI Long Packet.*

- bool VS_SO_EXPORT sendRawDataPacket (unsigned int nb_byte, uint8_t ∗DataByte)

  *Send a Raw Data Packet without hardware CRC and ECC computed.*

- void VS_SO_EXPORT getPacketStatistics (unsigned int ∗nb_sp, unsigned int ∗nb_lp)

  *Report the number of Short Packet and Long Packet effectively transmitted.*

- bool VS_SO_EXPORT getLaneModelVersion (unsigned int ∗Nb_Lane_Tx, unsigned int ∗Nb_Lane_Rx, char ∗LaneModel_Type, float ∗LaneModel_Version)

  *Return the Lane Model Type.*

- bool VS_SO_EXPORT setNbLaneDPHY (unsigned int Nb_Lane)

  *Define the number of lane enable.*

- unsigned int VS_SO_EXPORT getNbLaneDPHY ()

  *Get the number of DPHY lanes enabled.*

- void VS_SO_EXPORT displayInfoDPHY ()

  *Display the DPHY Lane information on the last access.*

- bool VS_SO_EXPORT enableDPHYInterface (unsigned int Nb_Lanes)

  *WakeUp the D-PPHY Lane Model.*

- bool VS_SO_EXPORT setNbCycleClkRqstDPHY (unsigned int Nb_cycles)

  *Define the number of cycle (DPHY Clock) between the TxRequest_Data and TxRequest_Clk.*

- bool VS_SO_EXPORT setNbCycleClkRqstDPHY (unsigned int Nb_Front_cycles, unsigned int Nb_Back_cycles)

  *Define the number of cycle (DPHY Clock) between the TxRequest_Data and TxRequest_Clk.*

- bool VS_SO_EXPORT openMonitorFile_CSI (char ∗fileName, uint level=0)

  *Open monitor file and start monitoring CSI packets.*

- bool VS_SO_EXPORT closeMonitorFile_CSI ()

  *Stops monitor and close monitor file.*

- bool VS_SO_EXPORT stopMonitorFile_CSI ()

  *Stops monitor.*

- bool VS_SO_EXPORT restartMonitorFile_CSI ()

  *Restarts monitoring on current file.*

- bool VS_SO_EXPORT setCCISlaveAddress (unsigned int Slave_Address)

  *Define the CCI slave address for register.*

- unsigned int VS_SO_EXPORT getCCISlaveAddress ()

  *Return the CCI slave Address.*

- bool VS_SO_EXPORT setCCIAddressMode (unsigned int Address_Mode)

  *Define the I2C sub address mode (8bits or 16bits).*

- unsigned int VS_SO_EXPORT getCCIAddressMode ()

  *Return the CCI sub address (index) mode.*

- bool VS_SO_EXPORT updateCCIRegister (unsigned int addr, unsigned int nb_registers)

  *Update the shadow soft register from the CCI hardware slave register.*

- bool VS_SO_EXPORT updateCCIRegister (void)

  *Read the whole CCI Register from the CCI hardware slave register and update the shadow soft register.*

- bool VS_SO_EXPORT getCCIRegister (unsigned int addr, unsigned int nb_registers, uint8_t *DataRead)

  *Read the CCI Register from the shadow soft register.*

- uint8_t VS_SO_EXPORT getCCIRegister (unsigned int addr)

  *Read one CCI Register from the shadow soft register.*

- bool VS_SO_EXPORT getAllCCIRegister (uint8_t *DataRead)

  *Return the whole CCI Register from the shadow soft register.*

- void VS_SO_EXPORT displayCCIRegister (unsigned int start_addr, unsigned int number)

  *Display the CCI Register from the shadow soft register.*

- bool VS_SO_EXPORT setCCIRegister (unsigned int addr, unsigned int nb_registers, uint8_t *DataToWrite)

  *Initialize CCI Register : Shadow soft register and CCI hardware slave register.*

- bool VS_SO_EXPORT setAllCCIRegister (uint8_t *DataToWrite)

  *Initialize All CCI Register : Shadow soft register and CCI hardware slave register.*

- unsigned int VS_SO_EXPORT getNumberPendingCCI (void)

  *Return the number of Write/Modify event.*

- void VS_SO_EXPORT enableCCIAddr (unsigned int addr, bool enable)

  *Enable the Write/Modify auto signalization for CCI Register at the specified address.*

- void VS_SO_EXPORT enableCCIAddrRange (unsigned int start_addr, unsigned int number, bool enable)

  *Define the address range of the Write/Modify auto signalization.*

- void VS_SO_EXPORT registerCCI_CB_Addr (unsigned int addr, void(*userCCI_ModifyCB)(void *context), void *context)

  *Register a function that will be called by the transactor instead of the CSI. This function will be execut when the API will detect a CCI Write/Modify at the specified address.*

- void VS_SO_EXPORT registerCCI_CB_AddrRange (unsigned int start_addr, unsigned int number, void(*userCCI_ModifyCB)(void *context), void *context)

  *Register a function that will be called by the transactor instead of the CSI. This function will be executed when the API will detect a CCI Write/Modify at the specified Range address.*

- void VS_SO_EXPORT unRegisterCCI_CB_Addr (unsigned int addr)

  *UnRegister the Callback at the specified address.*

- void VS_SO_EXPORT unRegisterCCI_CB_AddrRange (unsigned int start_addr, unsigned int number)

  *Define the address range of the CCI register to UnRegister the Callback.*

- CCIStatusRegister_t VS_SO_EXPORT getCCIStatusRegister (unsigned int addr)

  *Return the information about the CCI Register management (Callback and Monitored).*

- bool VS_SO_EXPORT getNextCCIRegisterModify (unsigned int *addr, uint8_t *data)

*Return the address and the data of CCI register monitored. The data are available while CSIStatus.b.CCI_-WRITE_MODIFY_PENDING is set.*

- bool VS_SO_EXPORT openMonitorFile_CCI (char ∗fileName)

  *Open monitor file and start monitoring CCI Read and Write access.*

- bool VS_SO_EXPORT save (const char ∗clockName)

  *Save MIPI_CSI transactor state Save the state of the transactor before the call to ZEBU_Board::save().*

- bool VS_SO_EXPORT configRestore (const char ∗clockName)

  *Configure MIPI_CSI transactor after restore Sends configuration parameters defined by the user to the transactor after a restore.*

- void VS_SO_OBSOLETE setVideoProfileV (unsigned int Bd2V, unsigned int V2active, unsigned int VActive2Bd)

  *Define the Vertical Video Profile.*

- void VS_SO_OBSOLETE setVideoProfileH (unsigned int Bd2H, unsigned int H2active, unsigned int HActive2Bd)

  *Define the Horizontal Video Profile.*

- void VS_SO_OBSOLETE getVideoSync (unsigned int ∗Bd2V, unsigned int ∗V2active, unsigned int ∗VActive2Bd, unsigned int ∗Bd2H, unsigned int ∗H2active, unsigned int ∗HActive2Bd)

  *Return the Horizontal and Vertical Video Profile.*

## Static Public Member Functions

- static const char ∗VS_SO_EXPORT getXtorVersion ()

  *Get the CSI transactor revision.*

- static bool VS_SO_EXPORT isDriverPresent (ZEBU::Board ∗board)

  *Check MIPI_CSI Transactor presence.*

- static bool VS_SO_EXPORT firstDriver (ZEBU::Board ∗board)

  *Look for first intance of MIPI_CSI Transactor.*

- static bool VS_SO_EXPORT nextDriver (void)

  *Search for next MIPI_CSI Transactor instance Prior to calling this method, the search must be initialized by calling the FirstDriver() method.*

- static const char ∗VS_SO_EXPORT getInstanceName (void)

  *Get name of current MIPI_CSI Transactor instance.*

- static bool VS_SO_OBSOLETE IsDriverPresent (ZEBU::Board ∗board)
- static bool VS_SO_OBSOLETE FirstDriver (ZEBU::Board ∗board)
- static bool VS_SO_OBSOLETE NextDriver (void)
- static const char ∗VS_SO_OBSOLETE GetInstanceName (void)

### 5.2.1 Detailed Description

CSI Transactor Class definition.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 ZEBU_IP::MIPI_CSI::CSI::CSI ()

MIPI_CSI Constructor.

#### 5.2.2.2 ZEBU_IP::MIPI_CSI::CSI::~CSI ()

MIPI_CSI Destructor.

### 5.2.3 Member Function Documentation

#### 5.2.3.1 bool ZEBU_IP::MIPI_CSI::CSI::buildImage ()

Build the pixel Frame Buffer according to the current source and its transformation.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.2 bool ZEBU_IP::MIPI_CSI::CSI::checkCSITxCharacteristics ()

check the CSI Timing setup to achieve the required FPS with the current image resolution

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.3 void ZEBU_IP::MIPI_CSI::CSI::checkWatchdogs (void)

check all watchdogs Go through all Watchdog to check timeouts. If a timeout, the timeout callback is called if registered. if the timeout callback returns true or no callback is registered, the execution of blocking will stop

#### 5.2.3.4 bool ZEBU_IP::MIPI_CSI::CSI::closeMonitorFile_CSI ()

Stops monitor and close monitor file.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.5 bool ZEBU_IP::MIPI_CSI::CSI::configRestore (const char * *clockName*)

Configure MIPI_CSI transactor after restore Sends configuration parameters defined by the user to the transactor after a restore.

**Parameters:**

***clockName*** name of the controlled clock

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.6 void ZEBU_IP::MIPI_CSI::CSI::CSIServiceLoop (int(*)(void *, int) *servcieCB* = NULL, void * *context* = NULL)

Handle the current MIPI_CSI transactor instance incoming and pending messages This method check all the MIPI_CSI ports in order to send pending messages or receive incoming messages when possible. if the service callback pointer is NULL, the method exits right after having handled the messages. if a service callback is registered, it is called after each check of the ports. The pending argument will be set to 0 if no messages could be sent or received, otherwise it will be sent to a different value.

**Parameters:**

***servcieCB*** pointer to the service callback function

***context*** pointer that will be given as argument to the service callback

#### 5.2.3.7 void ZEBU_IP::MIPI_CSI::CSI::defineFrontPorchSync (Frame_pixel_t *HFP*, Frame_line_t *VFP*)

Define the Horizontal and Vertical Video Front Porch Profile.

**Parameters:**

***HFP*** Horizontal Front Porch

***VFP*** Vertical Front Porch

#### 5.2.3.8 void ZEBU_IP::MIPI_CSI::CSI::defineRefClkFreq (float *RefClkFreq*)

Define the frequency of the Transcator Reference Clock.

**Parameters:**

***RefClkFreq*** frequence MHz

#### 5.2.3.9 bool ZEBU_IP::MIPI_CSI::CSI::disableSequencer ()

Disable hardware Sequencer of the Bfm.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if in Progress or Failure

#### 5.2.3.10 void ZEBU_IP::MIPI_CSI::CSI::displayCCIRegister (unsigned int *start_addr*, unsigned int *number*)

Display the CCI Register from the shadow soft register.

**Parameters:**

***start_addr*** Start address of register

***number*** of display

#### 5.2.3.11 void ZEBU_IP::MIPI_CSI::CSI::displayCSIStatus (void)

Display the field of CSIEventStatus.

#### 5.2.3.12 void ZEBU_IP::MIPI_CSI::CSI::displayInfoDPHY ()

Display the DPHY Lane information on the last access.

#### 5.2.3.13 void ZEBU_IP::MIPI_CSI::CSI::enableCCIAddr (unsigned int *addr*, bool *enable*)

Enable the Write/Modify auto signalization for CCI Register at the specified address.

**Parameters:**

***addr*** Address to enable/disable

***enable*** if true : enable the auto signalization, enable if false : disable the auto signalization

#### 5.2.3.14 void ZEBU_IP::MIPI_CSI::CSI::enableCCIAddrRange (unsigned int *start_addr*, unsigned int *number*, bool *enable*)

Define the address range of the Write/Modify auto signalization.

**Parameters:**

***start_addr*** is the start address for auto signalization

***number*** is the number of address for auto signalization

***enable*** if true : enable the auto signalization enable if false : disable the auto signaliation

#### 5.2.3.15 bool ZEBU_IP::MIPI_CSI::CSI::enableDPHYInterface (unsigned int *Nb_Lanes*)

WakeUp the D-PPHY Lane Model.

**Parameters:**

***Nb_Lanes*** is the number of lane to wake up

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.16 void ZEBU_IP::MIPI_CSI::CSI::enableSequencer ()

Enable hardware Sequencer of the Bfm.

#### 5.2.3.17 void ZEBU_IP::MIPI_CSI::CSI::enableWatchdog (bool *enable* = `true`)

enable or disable CSI watchdogs Note that by default watchdogs are enabled

**Parameters:**

***enable*** enable watchdogs when true, disable if false

#### 5.2.3.18 static bool VS_SO_OBSOLETE ZEBU_IP::MIPI_CSI::CSI::FirstDriver (ZEBU::Board * *board*) `[static]`

#### 5.2.3.19 static bool ZEBU_IP::MIPI_CSI::CSI::firstDriver (ZEBU::Board * *board*) `[static]`

Look for first intance of MIPI_CSI Transactor.

**Parameters:**

***board*** ZeBu Board

**Returns:**

bool

**Return values:**

***true*** if first MIPI_CSI Transactor instance was found

***false*** if no MIPI_CSI Transactor was found

#### 5.2.3.20 bool ZEBU_IP::MIPI_CSI::CSI::flushCSIPacket ()

Flush HW/SW Fifo pipe.

#### 5.2.3.21 bool ZEBU_IP::MIPI_CSI::CSI::getAllCCIRegister (uint8_t * *DataRead*)

Return the whole CCI Register from the shadow soft register.

**Parameters:**

***DataRead*** pointer of data

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.22 unsigned int ZEBU_IP::MIPI_CSI::CSI::getCCIAddressMode ()

Return the CCI sub address (index) mode.

**Returns:**

unsigned int

**Return values:**

***8*** means address mode 8 bits

***16*** means address mode 16 bits

#### 5.2.3.23 uint8_t ZEBU_IP::MIPI_CSI::CSI::getCCIRegister (unsigned int *addr*)

Read one CCI Register from the shadow soft register.

**Parameters:**

***addr*** Address of read

**Returns:**

uint8_t

**Return values:**

***data***

#### 5.2.3.24 bool ZEBU_IP::MIPI_CSI::CSI::getCCIRegister (unsigned int *addr*, unsigned int *nb_registers*, uint8_t * *DataRead*)

Read the CCI Register from the shadow soft register.

**Parameters:**

***addr*** is the start address of the first read
***nb_registers*** is the number of byte to read
***DataRead*** pointer of data

**Returns:**

bool

**Return values:**

***true*** if Success
***false*** if Failure

#### 5.2.3.25 unsigned int ZEBU_IP::MIPI_CSI::CSI::getCCISlaveAddress ()

Return the CCI slave Address.

**Returns:**

unsigned int

**Return values:**

***CCI*** slave address

#### 5.2.3.26 void CCIStatusRegister_t ZEBU_IP::MIPI_CSI::CSI::getCCIStatusRegister (unsigned int *addr*)

Return the information about the CCI Register management (Callback and Monitored).

**Parameters:**

***addr***

**Returns:**

CCIStatusRegister_t

**Return values:**

***MONITORING_ENABLE*** Indicates if a CCI address is monitored
***CALLBACK_ENABLE*** Indicates if a CallBack is present

#### 5.2.3.27 void ZEBU_IP::MIPI_CSI::CSI::getColorbarParam (Pixel_Format_t * *Pixel_Format*, Frame_line_t * *nb_lines*, Frame_pixel_t * *nb_pixels*)

Return the colorbar parameters.

**Parameters:**

***Pixel_Format*** pointer of pixel format

***nb_lines*** pointer of number of line per frame

***nb_pixels*** pointer of number of pixel per line

#### 5.2.3.28 void ZEBU_IP::MIPI_CSI::CSI::getCSIBlankingDPHYPeriod (uint * *LineBlanking*, uint * *FrameBlanking*)

Return the line blanking period in number of cycles of the DPHY source clock.

**Parameters:**

**LineBlanking*

**FrameBlanking*

#### 5.2.3.29 void ZEBU_IP::MIPI_CSI::CSI::getCSIBlankingTiming (float * *LineBlanking*, float * *FrameBlanking*)

Return the line blanking period in microseconds.

**Parameters:**

**LineBlanking*

**FrameBlanking*

#### 5.2.3.30 unsigned int ZEBU_IP::MIPI_CSI::CSI::getCSIClkDivider ()

Return the CSI clock divider.

**Returns:**

unsigned int

**Return values:**

***Value*** of CSI clock divider

#### 5.2.3.31 void ZEBU_IP::MIPI_CSI::CSI::getCSIStatus (CSIEventStatus_t * *CSIStatus*)

Return the MIPI_CSI Video Status.

**Parameters:**

***CSIStatus*** :IDLE : Indicates if the CSI is IDLE. CSIStatus :FRAME_SENDING : Indicates if a Frame is sending. CSIStatus :LINE_SENDING : Indicates if a Line is sending. CSIStatus :FRAME_-DONE : Indicates if a Frame is completely send. CSIStatus :LINE_DONE : Indicates if a Line is completely send. CSIStatus :CCI_WRITE_MODIFY_PENDING: Indicates a pending Write Modify event. CSIStatus :bool CCI_UPDATE_DONE : Indicates if the CCI Shadow soft register update is finished.

#### 5.2.3.32 unsigned long ZEBU_IP::MIPI_CSI::CSI::getCurrentCycle (void)

Indicates the current cycle number.

**Returns:**

unsigned long

#### 5.2.3.33 unsigned int ZEBU_IP::MIPI_CSI::CSI::getFrameNumber (void)

Indicates the number of frame sent.

**Returns:**

unsigned int

#### 5.2.3.34 float ZEBU_IP::MIPI_CSI::CSI::getFrameRate ()

Get the frame rate.

**Returns:**

float

**Return values:**

***FPS*** the Frame Per Second

#### 5.2.3.35 unsigned int ZEBU_IP::MIPI_CSI::CSI::getHWinfo ()

Return information about hardware part of transactor.

**Returns:**

unsigned int

**Return values:**

***internal*** value

#### 5.2.3.36 bool ZEBU_IP::MIPI_CSI::CSI::getImageRegion (Frame_pixel_t * *Start_Pixel*, Frame_pixel_t * *Nb_Pixels*, Frame_line_t * *Start_Line*, Frame_line_t * *Nb_Lines*)

Get the region area sended.

**Parameters:**

***Start_Pixel*** is a pointer on the pixel number of the line to begin the region

***Nb_Pixels*** is a pointer one the number of pixel which define the region

***Start_Line*** is a pointer one the line number of the frame to begin the region

***Nb_Lines*** is a pointer one the number of line which define the region

#### 5.2.3.37 static const char* VS_SO_OBSOLETE ZEBU_IP::MIPI_CSI::CSI::GetInstanceName (void) [static]

#### 5.2.3.38 static const char * ZEBU_IP::MIPI_CSI::CSI::getInstanceName (void) [static]

Get name of current MIPI_CSI Transactor instance.

**Returns:**

const char*

**Return values:**

***Link*** name string

#### 5.2.3.39 bool ZEBU_IP::MIPI_CSI::CSI::getLaneModelVersion (unsigned int * *Nb_Lane_Tx*, unsigned int * *Nb_Lane_Rx*, char * *LaneModel_Type*, float * *LaneModel_Version*)

Return the Lane Model Type.

**Parameters:**

***Nb_Lane_Tx*** pointer on number of Tx Lane

***Nb_Lane_Rx*** pointer on number of Rx Lane

***LaneModel_Type*** pointer on Lane Model Type

***LaneModel_Version*** pointer on Lane Model Version

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.40 unsigned int ZEBU_IP::MIPI_CSI::CSI::getLineInFrame (void)

Indicates the number of line sent in the current frame.

**Returns:**

unsigned int

#### 5.2.3.41 const char * ZEBU_IP::MIPI_CSI::CSI::getName (void)

Return the prefix message name (for debug purpose).

**Returns:**

char*

**Return values:**

***Link*** name string

#### 5.2.3.42 unsigned int ZEBU_IP::MIPI_CSI::CSI::getNbLaneDPHY ()

Get the number of DPHY lanes enabled.

**Returns:**

unsigned int

**Return values:**

***number*** of DPHY lane enable

#### 5.2.3.43 bool ZEBU_IP::MIPI_CSI::CSI::getNextCCIRegisterModify (unsigned int * *addr*, uint8_t * *data*)

Return the address and the data of CCI register monitored. The data are available while CSIStatus.b.CCI_-WRITE_MODIFY_PENDING is set.

**Parameters:**

***addr*** pointer of address modified

***data*** pointer of data modified

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.44 unsigned int ZEBU_IP::MIPI_CSI::CSI::getNumberPendingCCI (void)

Return the number of Write/Modify event.

**Returns:**

unsigned int

#### 5.2.3.45 void ZEBU_IP::MIPI_CSI::CSI::getPacketStatistics (unsigned int * *nb_sp*, unsigned int * *nb_lp*)

Report the number of Short Packet and Long Packet effectively transmitted.

**Parameters:**

***nb_sp*** number of Short Packet send

***nb_lp*** number of Long Packet send

#### 5.2.3.46 unsigned int ZEBU_IP::MIPI_CSI::CSI::getPixelPacking ()

Get the number of split CSI pixel packet.

**Returns:**

unsigned int

**Return values:**

***nb_LineSplit*** the number of split in one line

#### 5.2.3.47 float ZEBU_IP::MIPI_CSI::CSI::getPPIByteClk_Freq ()

Return the frequency of PPI Byte clock.

**Returns:**

float

**Return values:**

***Value*** of PPI Byte clock in Mega Hertz

#### 5.2.3.48 float ZEBU_IP::MIPI_CSI::CSI::getPPIByteClk_MinFreq ()

Return the minimum frequency of PPI Byte clock to achieve the required FPS.

**Returns:**

unsigned int

**Return values:**

***Value*** of PPI Byte clock in Mega Hertz

#### 5.2.3.49 float ZEBU_IP::MIPI_CSI::CSI::getRealFrameRate ()

Return the Real frame rate from the CSI transactor current transmission.

**Returns:**

float

**Return values:**

***the*** Frame Rate in Frame/s

#### 5.2.3.50 SensorMode_t ZEBU_IP::MIPI_CSI::CSI::getSensorMode ()

Return the mode of capture: RGB, YUV, RAW or Colorbar.

**Returns:**

SensorMode_t

**Return values:**

***enum*** contaning the current sensore mode

#### 5.2.3.51 void ZEBU_IP::MIPI_CSI::CSI::getSequencer ()

Return Sequencer status (enable or disable).

**Returns:**

bool

**Return values:**

***true*** if Senquencer is enable

***false*** if Sequencer is disable

#### 5.2.3.52 void ZEBU_IP::MIPI_CSI::CSI::getVideoMode (Frame_line_t * *nb_lines*, Frame_pixel_t * *nb_pixels*, Pixel_Format_t * *VideoFormatOut*, SensorMode_t * *SensorMode*)

Return the video parameter.

**Parameters:**

***nb_lines*** pointer on the number of line

***nb_pixels*** pointer on the number of pixel per line

***VideoFormatOut*** pointer on the video format out

***SensorMode*** pointer on capture mode

#### 5.2.3.53 void ZEBU_IP::MIPI_CSI::CSI::getVideoSync (unsigned int * *Bd2V*, unsigned int * *V2active*, unsigned int * *VActive2Bd*, unsigned int * *Bd2H*, unsigned int * *H2active*, unsigned int * *HActive2Bd*)

Return the Horizontal and Vertical Video Profile.

**Parameters:**

***Bd2V*** number of blanking line between the edge of the screen and the frame start packet
***V2active*** number of blanking line between the frame start packet and the first pixel line
***VActive2Bd*** number of blanking line between the last pixel line and the edge of the screen
***Bd2H*** number of blanking pixel between the edge of the screen and the line start packet
***H2active*** number of blanking pixel between the line start packet and the pixel packet
***HActive2Bd*** number of blanking pixel between the line end packet and the edge of the screen

#### 5.2.3.54 void ZEBU_IP::MIPI_CSI::CSI::getVideoTiming (float * *VFP*, float * *FrameLength*, float * *VBP*, float * *HFP*, float * *LineLength*, float * *HBP*)

Return the frame timing in us.

**Parameters:**

***VFP*** Front Porch Vertical Timing
***FrameLength*** frame timing
***VBP*** Back Porch Vertical Timing
***HFP*** Front Porch Horizontal Timing
***LineLength*** line timing
***HBP*** Back Porch Horizontal Timing

#### 5.2.3.55 float ZEBU_IP::MIPI_CSI::CSI::getVirtualPixelClkFreq ()

Return the equivalent pixelClock Frequency from the FPS and the image resolution.

**Returns:**

float

**Return values:**

***the*** value of Pixel Clock Frequence

#### 5.2.3.56 static const char * ZEBU_IP::MIPI_CSI::CSI::getXtorVersion () `[static]`

Get the CSI transactor revision.

**Returns:**

char*

**Return values:**

***string*** containing the current CSI Xactor version

#### 5.2.3.57 bool ZEBU_IP::MIPI_CSI::CSI::init (Board * *zebu*, const char * *driverName*)

Initialization of the transactor.

**Parameters:**

***zebu*** The board object used

***driverName*** the transactor instance name in your DVE

**Returns:**

bool

**Return values:**

***true*** if the initialization sequence is successfull.

***false*** if the transactor initialization sequence failed.

#### 5.2.3.58 static bool VS_SO_OBSOLETE ZEBU_IP::MIPI_CSI::CSI::IsDriverPresent (ZEBU::Board * *board*) [static]

#### 5.2.3.59 static bool ZEBU_IP::MIPI_CSI::CSI::isDriverPresent (ZEBU::Board * *board*) [static]

Check MIPI_CSI Transactor presence.

**Parameters:**

***board*** ZeBu Board

**Returns:**

bool

**Return values:**

***true*** if at least on MIPI_CSI Transactor was found

***false*** if no MIPI_CSI Transactor was found

#### 5.2.3.60 static bool VS_SO_OBSOLETE ZEBU_IP::MIPI_CSI::CSI::NextDriver (void) [static]

#### 5.2.3.61 static bool ZEBU_IP::MIPI_CSI::CSI::nextDriver (void) [static]

Search for next MIPI_CSI Transactor instance Prior to calling this method, the search must be initialized by calling the FirstDriver() method.

**Returns:**

bool

**Return values:**

***true*** if next MIPI_CSI Transactor instance was found

***false*** if no more MIPI_CSI Transactor was found

#### 5.2.3.62 bool ZEBU_IP::MIPI_CSI::CSI::openMonitorFile_CCI (char * *fileName*)

Open monitor file and start monitoring CCI Read and Write access.

**Parameters:**

***fileName*** monitor file name

**Return values:**

***true*** if succesful

#### 5.2.3.63 bool ZEBU_IP::MIPI_CSI::CSI::openMonitorFile_CSI (char * *fileName*, uint *level* = 0)

Open monitor file and start monitoring CSI packets.

**Parameters:**

***fileName*** monitor file name

***level*** Info level (optionnal default is 0) if level = 0 meams no log if level = 1 means Log all Packet exept Payload if level = 2 means Log Payload Packets only if level = 3 means Log all Packet

**Return values:**

***true*** if succesful

#### 5.2.3.64 void ZEBU_IP::MIPI_CSI::CSI::registerCCI_CB_Addr (unsigned int *addr*, void(*)(void *context) *userCCI_ModifyCB*, void * *context*)

Register a function that will be called by the transactor instead of the CSI. This function will be execut when the API will detect a CCI Write/Modify at the specified address.

**Parameters:**

***addr*** attach the CallBack to this address

***userCCI_ModifyCB*** pointer to the function

***context*** pointer that will be given as argument to handler

#### 5.2.3.65 void ZEBU_IP::MIPI_CSI::CSI::registerCCI_CB_AddrRange (unsigned int *start_addr*, unsigned int *number*, void(*)(void *context) *userCCI_ModifyCB*, void * *context*)

Register a function that will be called by the transactor instead of the CSI. This function will be executed when the API will detect a CCI Write/Modify at the specified Range address.

**Parameters:**

***start_addr*** attach the CallBack from this address

***number*** of register attach to the CallBack

***userCCI_ModifyCB*** pointer to the function

***context*** pointer that will be given as argument to handler

#### 5.2.3.66 void ZEBU_IP::MIPI_CSI::CSI::registerTimeoutCB (bool(∗)(void ∗context) *timeoutCB*, void ∗ *context* = NULL)

register a timeout callback register a function that will be called when watchdog timeout is trigerred if the timeout function return true, current blocking operation are aborted After each timeout, the trigerred watchdog timer is automatically rearmed

**Parameters:**

***timeoutCB*** pointer to the timeout function, NULL to unregister previous callback

***context*** pointer that will be given as argument to the timeout callback

#### 5.2.3.67 void ZEBU_IP::MIPI_CSI::CSI::registerUserCB (void(∗)(void ∗context) *userCB*, void ∗ *context*)

Register a function that will be called by the transactor instead of the CSI service loop when registered, the user callback will be called each time the current operation is unable to send send/receive data to/from ZeBu port.

**Parameters:**

***userCB*** pointer to the function or NULL to disable previously recorded callback

***context*** pointer that will be given as argument to handler

#### 5.2.3.68 bool ZEBU_IP::MIPI_CSI::CSI::restartMonitorFile_CSI ()

Restarts monitoring on current file.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.69 bool ZEBU_IP::MIPI_CSI::CSI::runCycle (unsigned int *nb_cycle*)

run CSI_PPI_bytes clock during nb_cycle

**Parameters:**

***nb_cycle*** number of CSI_PPI_bytes clock cycle to run

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if in Progress or Failure

#### 5.2.3.70 bool ZEBU_IP::MIPI_CSI::CSI::save (const char * *clockName*)

Save MIPI_CSI transactor state Save the state of the transactor before the call to ZEBU_Board::save().

**Parameters:**

***clockName*** name of the controlled clock

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.71 bool ZEBU_IP::MIPI_CSI::CSI::sendImage ()

Send the whole content of the pixel Frame Buffer.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.72 bool ZEBU_IP::MIPI_CSI::CSI::sendLine ()

Send a line of the pixel frame buffer.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.73 bool ZEBU_IP::MIPI_CSI::CSI::sendLongPacket (CSI_Packet_Name_t *DataID*, uint16_t *WordCount*, uint8_t * *DataByte*)

Send a CSI Long Packet.

**Parameters:**

***DataID*** CSI_Packet_Name_t

***WordCount*** (16bits)

∗***DataByte*** Pointer of DataByte

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.74 bool ZEBU_IP::MIPI_CSI::CSI::sendLongPacket (unsigned int *DataID*, uint16_t *WordCount*, uint8_t ∗ *DataByte*)

Send a CSI Long Packet.

**Parameters:**

***DataID*** (8bits)

***WordCount*** (16bits)

∗***DataByte*** Pointer of DataByte

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.75 bool ZEBU_IP::MIPI_CSI::CSI::sendRawDataPacket (unsigned int *nb_byte*, uint8_t ∗ *DataByte*)

Send a Raw Data Packet without hardware CRC and ECC computed.

**Parameters:**

***nb_byte*** number of byte to send

∗***DataByte*** Pointer of DataByte

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.76 bool ZEBU_IP::MIPI_CSI::CSI::sendShortPacket (CSI_Packet_Name_t *DataID*, uint16_t *Data_0_1*)

Send a CSI Short Packet.

**Parameters:**

***DataID*** CSI_Packet_Name_t

***Data_0_1*** (16 bits)

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.77 bool ZEBU_IP::MIPI_CSI::CSI::sendShortPacket (unsigned int *DataID*, uint16_t *Data_0_1*)

Send a CSI Short Packet.

**Parameters:**

***DataID*** (8bits)

***Data_0_1*** (16 bits)

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.78 bool ZEBU_IP::MIPI_CSI::CSI::setAllCCIRegister (uint8_t * *DataToWrite*)

Initialize All CCI Register : Shadow soft register and CCI hardware slave register.

**Parameters:**

***DataToWrite*** pointer of data to write

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.79 bool ZEBU_IP::MIPI_CSI::CSI::setCCIAddressMode (unsigned int *Address_Mode*)

Define the I2C sub address mode (8bits or 16bits).

**Parameters:**

***Address_Mode*** = 8 means address mode 8 bits Address_Mode = 16 means address mode 16 bits

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.80 bool ZEBU_IP::MIPI_CSI::CSI::setCCIRegister (unsigned int *addr*, unsigned int *nb_registers*, uint8_t * *DataToWrite*)

Initialize CCI Register : Shadow soft register and CCI hardware slave register.

**Parameters:**

***addr*** is the start address of Initialization

***nb_registers*** is the number of byte write

***DataToWrite*** pointer of data to write

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.81 bool ZEBU_IP::MIPI_CSI::CSI::setCCISlaveAddress (unsigned int *Slave_Address*)

Define the CCI slave address for register.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.82 void ZEBU_IP::MIPI_CSI::CSI::setColorbar (Pixel_Format_t *Pixel_Format*, Frame_line_t *nb_lines*, Frame_pixel_t *nb_pixels*)

Defines the colorbar parameters.

**Parameters:**

***Pixel_Format*** is the pixel format

***nb_lines*** is the number of line per frame

***nb_pixels*** is the number of pixel per line

#### 5.2.3.83 bool ZEBU_IP::MIPI_CSI::CSI::setCSIClkDivider (unsigned int *CSIClkDivider*)

set the CSI clock divider

**Parameters:**

***CSIClkDivider*** clock divider must be a upper than one

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.84 void ZEBU_IP::MIPI_CSI::CSI::setDebugLevel (uint *lvl*)

Set Display Trace Level For Xtor.

**Parameters:**

***lvl*** debug level from 0 to 3, 0:no debug messages and 3: all debug messages

#### 5.2.3.85 void ZEBU_IP::MIPI_CSI::CSI::setFrameRate (float *FPS*)

Define the frame rate.

**Parameters:**

***FPS*** Frame Per Second

#### 5.2.3.86 bool ZEBU_IP::MIPI_CSI::CSI::setImageRegion (Frame_pixel_t *Start_Pixel*, Frame_pixel_t *Nb_Pixels*, Frame_line_t *Start_Line*, Frame_line_t *Nb_Lines*)

Define the region area to send.

**Parameters:**

***Start_Pixel*** define the pixel number on the line to begin the region

***Nb_Pixels*** define the number of pixel which define the region

***Start_Line*** define the line number on the frame to begin the region

***Nb_Lines*** define the number of line which define the region

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.87 bool ZEBU_IP::MIPI_CSI::CSI::setImageRotate (unsigned int *Rotation*)

Define the rotation to apply on the original file.

**Parameters:**

***Rotation*** is the degree to apply on the frame (0, 90, 180, 270);

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.88 bool ZEBU_IP::MIPI_CSI::CSI::setImageZoom (float *Xzoom*, float *Yzoom*)

Define the zoom to apply on the original file.

**Parameters:**

***Xzoom*** is the coeficient to apply on the pixel line

***Yzoom*** is the coeficient to apply on the frame line

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.89 void ZEBU_IP::MIPI_CSI::CSI::setInputFile (string *file_path_name*, Pixel_File_Format_t *Pixel_File_Format*, Frame_line_t *nb_lines*, Frame_pixel_t *nb_pixels*, bool *loop_on_file* = `true`)

Define the video file according to its format and resolution.

**Parameters:**

***file_path_name*** is the file path name

***Pixel_File_Format*** is the pixel format of the file

***nb_lines*** is the number of line per frame

***nb_pixels*** is the number of pixel per line

***loop_on_file*** loop on the begining of the file when end of file is detected

#### 5.2.3.90 bool ZEBU_IP::MIPI_CSI::CSI::setLog (char * *fname*, bool *stdoutDup* = `false`)

Create and Open log file.

**Parameters:**

***fname*** log filename path

***stdoutDup*** Send log on both stdout and specified file

#### 5.2.3.91 bool ZEBU_IP::MIPI_CSI::CSI::setLog (FILE * *stream*, bool *stdoutDup* = `false`)

set output stream for transactor debug

**Parameters:**

***stream*** A handler to the ouput file

***stdoutDup*** Send stream on both stdout and specified file

#### 5.2.3.92 void ZEBU_IP::MIPI_CSI::CSI::setName (const char * *name*)

set name of MIPI_CSI transactor that will appear in all messages prefixes (for debug purpose)

**Parameters:**

***name*** Transactor name

#### 5.2.3.93 bool ZEBU_IP::MIPI_CSI::CSI::setNbCycleClkRqstDPHY (unsigned int *Nb_Front_cycles*, unsigned int *Nb_Back_cycles*)

Define the number of cycle (DPHY Clock) between the TxRequest_Data and TxRequest_Clk.

**Parameters:**

***Nb_Front_cycles*** the number of cycle between the rising edge of request

***Nb_Back_cycles*** the number of cycle between the falling edge of request

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.94 bool ZEBU_IP::MIPI_CSI::CSI::setNbCycleClkRqstDPHY (unsigned int *Nb_cycles*)

Define the number of cycle (DPHY Clock) between the TxRequest_Data and TxRequest_Clk.

**Parameters:**

***Nb_cycles*** the number of cycle

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.95 bool ZEBU_IP::MIPI_CSI::CSI::setNbLaneDPHY (unsigned int *Nb_Lane*)

Define the number of lane enable.

**Parameters:**

***Nb_Lane*** is the number of lane enable

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.96 void ZEBU_IP::MIPI_CSI::CSI::setPixelPacking (unsigned int *nb_LineSplit*, unsigned int *delay* = `0`)

Define the number of split CSI pixel packet.

**Parameters:**

***nb_LineSplit*** = 1 means no split nb_LineSplit = 2 means that the line is divided per two

***delay*** between the two CSI packet (Gap in DPHY cycles)

#### 5.2.3.97 void ZEBU_IP::MIPI_CSI::CSI::setSensorMode (SensorMode_t *SensorMode*)

Define the mode of capture: RGB, YUV, RAW or Colorbar.

#### 5.2.3.98 void ZEBU_IP::MIPI_CSI::CSI::setTimeout (uint64_t *ms*)

set watchdogs timeout value

**Parameters:**

***ms*** timeout value in ms

#### 5.2.3.99 void ZEBU_IP::MIPI_CSI::CSI::setTransform (Pixel_Format_t *VideoFormatIn*, Pixel_Format_t *VideoFormatOut*)

Defines the transformation to apply from the input file to the CSI packet.

**Parameters:**

***VideoFormatIn*** is the orignal format

***VideoFormatOut*** is the format to convert

#### 5.2.3.100 void ZEBU_IP::MIPI_CSI::CSI::setVideoJitter (uint *Hjitter_max*, uint *Vjitter_max*, uint *seed*)

Set the maximum horizontal line jitter and the maximum vertical frame jitter.

**Parameters:**

***Hjitter_max*** the maximum horizontal jitter (DPHY CLOCK)

***Vjitter_max*** the maximum vertical jitter (DPHY CLOCK)

***seed*** for random function

#### 5.2.3.101 void ZEBU_IP::MIPI_CSI::CSI::setVideoProfileH (unsigned int *Bd2H*, unsigned int *H2active*, unsigned int *HActive2Bd*)

Define the Horizontal Video Profile.

**Parameters:**

***Bd2H*** number of blanking pixel between the edge of the screen and the line start packet

***H2active*** number of blanking pixel between the line start packet and the pixel packet

***HActive2Bd*** number of blanking pixel between the line end packet and the edge of the screen

#### 5.2.3.102 void ZEBU_IP::MIPI_CSI::CSI::setVideoProfileV (unsigned int *Bd2V*, unsigned int *V2active*, unsigned int *VActive2Bd*)

Define the Vertical Video Profile.

**Parameters:**

***Bd2V*** number of blanking line between the edge of the screen and the frame start packet

***V2active*** number of blanking line between the frame start packet and the first pixel line

***VActive2Bd*** number of blanking line between the last pixel line and the edge of the screen

#### 5.2.3.103 void ZEBU_IP::MIPI_CSI::CSI::setVirtualChannelID (unsigned int *VirtualChannelID*)

Define the Virtual Channel Identifier.

**Parameters:**

***VirtualChannelID*** define de virtual channel ID

#### 5.2.3.104 void ZEBU_IP::MIPI_CSI::CSI::setZebuPortGroup (const unsigned int *grp*)

Set the current CSI transactor instance ports group This method allows the user to attach the transactor to a specific ZeBu port group. Then calling Board::serviceLoop() on the specified group will allow the serviceLoop() method to handle the CSI ports.

**Parameters:**

***grp*** ZeBu port group

#### 5.2.3.105 bool ZEBU_IP::MIPI_CSI::CSI::stopMonitorFile_CSI ()

Stops monitor.

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.106 void ZEBU_IP::MIPI_CSI::CSI::unRegisterCCI_CB_Addr (unsigned int *addr*)

UnRegister the Callback at the specified address.

**Parameters:**

***addr*** unattach the CallBack to this address

#### 5.2.3.107 void ZEBU_IP::MIPI_CSI::CSI::unRegisterCCI_CB_AddrRange (unsigned int *start_addr*, unsigned int *number*)

Define the address range of the CCI register to UnRegister the Callback.

**Parameters:**

***start_addr*** and number unattach the CallBack for all register between "start_addr" and "start_addr+number"

***number*** of register attach to the CallBack

#### 5.2.3.108 void ZEBU_IP::MIPI_CSI::CSI::updateCCIRegister (void)

Read the whole CCI Register from the CCI hardware slave register and update the shadow soft register.

#### 5.2.3.109 bool ZEBU_IP::MIPI_CSI::CSI::updateCCIRegister (unsigned int *addr*, unsigned int *nb_registers*)

Update the shadow soft register from the CCI hardware slave register.

**Parameters:**

***addr*** is the start address of update

***nb_registers*** is the number of byte to update

**Returns:**

bool

**Return values:**

***true*** if Success

***false*** if Failure

#### 5.2.3.110 void ZEBU_IP::MIPI_CSI::CSI::useLineSyncPacket (bool *enable*)

Enable the transmission of the Line start / Line End CSI packets.

**Parameters:**

***enable*** or disable the Packet Line Synchronisation

#### 5.2.3.111 void ZEBU_IP::MIPI_CSI::CSI::useZebuServiceLoop (int(∗)(void ∗context, int pending) *zebuServiceLoopHandler*, void ∗ *context*, const unsigned int *portGroupNumber*)

enable call of ZeBu service loop by CSI Xactor when activated, the transactor will call ZeBu Board::serviceLoop() with the specified arguments instead of the CSIServiceLoop() when the current operation cannot send/receive data to/from hardware See ZeBu API documentation to know more about the ZeBu service loop

**Parameters:**

***zebuServiceLoopHandler*** pointer to ZeBu service loop handler

***context*** pointer to data that will be given as argument to handler

***portGroupNumber*** ZeBu port group number

#### 5.2.3.112 void ZEBU_IP::MIPI_CSI::CSI::useZebuServiceLoop (int(∗)(void ∗context, int pending) *zebuServiceLoopHandler*, void ∗ *context*)

enable call of ZeBu service loop by CSI Xactor when activated, the transactor will call ZeBu Board::serviceLoop() with the specified arguments instead of the CSIServiceLoop() when the current operation cannot send/receive data to/from hardware See ZeBu API documentation to know more about the ZeBu service loop

**Parameters:**

***zebuServiceLoopHandler*** pointer to ZeBu service loop handler

***context*** pointer that will be given as argument to handler

#### 5.2.3.113 void ZEBU_IP::MIPI_CSI::CSI::useZebuServiceLoop (bool *state*)

enable call of ZeBu service loop by CSI Xactor when activated, the transactor will call ZeBu Board::serviceLoop() with no arguments instead of the CSIServiceLoop() when the current operation cannot send/receive data to/from hardware See ZeBu API documentation to know more about the ZeBu service loop

**Parameters:**

***state*** enable when true, disable when false

#### 5.2.3.114 void VS_SO_EXPORT ZEBU_IP::MIPI_CSI::CSI::usleep (int *usec*)

Pause the CSI testbench execution for the given number of Usec.

**Parameters:**

***usec*** Number of Usec to wait

The documentation for this class was generated from the following file:

- CSI.hh

## 5.3 ZEBU_IP::MIPI_CSI::CSIEventStatus_t Union Reference

Union to handle Status Of CSI Controller.

`#include <CSI_Struct.hh>`

### Public Attributes

- bool IDLE:1
- bool FRAME_SENDING:1
- bool LINE_SENDING:1
- bool FRAME_DONE:1
- bool LINE_DONE:1
- bool CCI_WRITE_MODIFY_PENDING:1
- bool CCI_UPDATE_DONE:1
- bool CSI_SEQ_STATE:1

### 5.3.1 Detailed Description

Union to handle Status Of CSI Controller.

### 5.3.2 Member Data Documentation

#### 5.3.2.1 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::CCI_UPDATE_DONE

Indicates if the CCI Shadow soft register update is finished

#### 5.3.2.2 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::CCI_WRITE_MODIFY_PENDING

Indicates a pending Write Modify event

#### 5.3.2.3 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::CSI_SEQ_STATE

Indicates if the status of sequencer

#### 5.3.2.4 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::FRAME_DONE

Indicates if a Frame is completely send

#### 5.3.2.5 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::FRAME_SENDING

Indicates if a Frame is sending - in progress

#### 5.3.2.6 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::IDLE

Indicates if the CSI is IDLE

#### 5.3.2.7 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::LINE_DONE

Indicates if a Line is completely send

#### 5.3.2.8 bool ZEBU_IP::MIPI_CSI::CSIEventStatus_t::LINE_SENDING

Indicates if a Line is sending - in progress

The documentation for this union was generated from the following file:

- CSI_Struct.hh

# Chapter 6

# File Documentation

## 6.1 CSI.hh File Reference

Include file containing C++ API for MIPI_CSI Transactor. `#include "CSI_Struct.hh"`

### Classes

- class ZEBU_IP::MIPI_CSI::CSI
  *CSI Transactor Class definition.*

### Namespaces

- namespace ZEBU_IP
  *Name Space Containing all the Zebu Transactor.*
- namespace MIPI_CSI
  *Name Space Containing MIPI_CSI Transactor description.*
- namespace ZEBU_IP::MIPI_CSI

### Defines

- #define VS_SO_OBSOLETE __attribute__((deprecated))

### 6.1.1 Detailed Description

Include file containing C++ API for MIPI_CSI Transactor.

### 6.1.2 Define Documentation

#### 6.1.2.1 #define VS_SO_OBSOLETE __attribute__((deprecated))

## 6.2 CSI_Struct.hh File Reference

Include file containing C++ API Types used by MIPI_CSI Transactor.

### Classes

- union ZEBU_IP::MIPI_CSI::CSIEventStatus_t

  *Union to handle Status Of CSI Controller.*

- union ZEBU_IP::MIPI_CSI::CCIStatusRegister_t

  *Structure to handle received event from the I2C BUS nion to view status of CSI CCI Call Back.*

### Namespaces

- namespace ZEBU_IP

  *Name Space Containing all the Zebu Transactor.*

- namespace MIPI_CSI

  *Name Space Containing MIPI_CSI Transactor description.*

- namespace ZEBU_IP::MIPI_CSI

### Typedefs

- typedef int32_t ZEBU_IP::MIPI_CSI::CSI_pixel_t
- typedef uint16_t ZEBU_IP::MIPI_CSI::Frame_pixel_t
- typedef uint16_t ZEBU_IP::MIPI_CSI::Frame_line_t

### Enumerations

- enum ZEBU_IP::MIPI_CSI::Pixel_Format_t {

  ZEBU_IP::MIPI_CSI::Pixel_Format_NotSet, ZEBU_IP::MIPI_CSI::RGBFFF, ZEBU_IP::MIPI_-CSI::RGB888, ZEBU_IP::MIPI_CSI::RGB666,

  ZEBU_IP::MIPI_CSI::RGB565, ZEBU_IP::MIPI_CSI::RGB555, ZEBU_IP::MIPI_CSI::RGB444, ZEBU_IP::MIPI_CSI::YUV422_10,

  ZEBU_IP::MIPI_CSI::YUV422_8, ZEBU_IP::MIPI_CSI::YUV420_10_CSPS, ZEBU_IP::MIPI_-CSI::YUV420_8_CSPS, ZEBU_IP::MIPI_CSI::YUV420_8_legacy,

  ZEBU_IP::MIPI_CSI::YUV420_10, ZEBU_IP::MIPI_CSI::YUV420_8, ZEBU_IP::MIPI_-CSI::RAW16, ZEBU_IP::MIPI_CSI::RAW14,

  ZEBU_IP::MIPI_CSI::RAW12, ZEBU_IP::MIPI_CSI::RAW10, ZEBU_IP::MIPI_CSI::RAW8, ZEBU_IP::MIPI_CSI::RAW7,

  ZEBU_IP::MIPI_CSI::RAW6 }

  *Enum for Pixel Format Definition.*

- enum ZEBU_IP::MIPI_CSI::Pixel_File_Format_t {

  ZEBU_IP::MIPI_CSI::File_Format_NotSet, ZEBU_IP::MIPI_CSI::FILE_RGB16, ZEBU_IP::MIPI_CSI::FILE_RGB8, ZEBU_IP::MIPI_CSI::FILE_YUV422_8,

  ZEBU_IP::MIPI_CSI::FILE_RAW16, ZEBU_IP::MIPI_CSI::FILE_RAW8 }

  *Enum for Pixel File Format Pixel Mapping.*

- enum ZEBU_IP::MIPI_CSI::SensorMode_t {

  ZEBU_IP::MIPI_CSI::_SensorMode_NotSet, ZEBU_IP::MIPI_CSI::RGB, ZEBU_IP::MIPI_CSI::YUV, ZEBU_IP::MIPI_CSI::RAW,

  ZEBU_IP::MIPI_CSI::Colorbar }

  *Enum for SensorMode Definition.*

- enum ZEBU_IP::MIPI_CSI::CSI_Packet_Name_t {

  ZEBU_IP::MIPI_CSI::P_Frame_Start, ZEBU_IP::MIPI_CSI::P_Frame_End, ZEBU_IP::MIPI_CSI::P_Line_Start, ZEBU_IP::MIPI_CSI::P_Line_End,

  ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code1, ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code2, ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code3, ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code4,

  ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code5, ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code6, ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code7, ZEBU_IP::MIPI_CSI::P_Generic_Short_Packet_Code8,

  ZEBU_IP::MIPI_CSI::P_Null, ZEBU_IP::MIPI_CSI::P_Blanking_Data, ZEBU_IP::MIPI_CSI::P_Embedded_8bit_non_Image_Data, ZEBU_IP::MIPI_CSI::P_YUV420_8bit,

  ZEBU_IP::MIPI_CSI::P_YUV420_10bit, ZEBU_IP::MIPI_CSI::P_Legacy_YUV420_8bit, ZEBU_IP::MIPI_CSI::P_YUV420_8bit_Chroma_Shifted_Pixel_Sampling, ZEBU_IP::MIPI_CSI::P_YUV420_10bit_Chroma_Shifted_Pixel_Sampling,

  ZEBU_IP::MIPI_CSI::P_YUV422_8bit, ZEBU_IP::MIPI_CSI::P_YUV422_10bit, ZEBU_IP::MIPI_CSI::P_RGB444, ZEBU_IP::MIPI_CSI::P_RGB555,

  ZEBU_IP::MIPI_CSI::P_RGB565, ZEBU_IP::MIPI_CSI::P_RGB666, ZEBU_IP::MIPI_CSI::P_RGB888, ZEBU_IP::MIPI_CSI::P_RAW6,

  ZEBU_IP::MIPI_CSI::P_RAW7, ZEBU_IP::MIPI_CSI::P_RAW8, ZEBU_IP::MIPI_CSI::P_RAW10, ZEBU_IP::MIPI_CSI::P_RAW12,

  ZEBU_IP::MIPI_CSI::P_RAW14, ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type1, ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type2, ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type3,

  ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type4, ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type5, ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type6, ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type7,

  ZEBU_IP::MIPI_CSI::P_User_Defined_8bit_Data_Type8 }

  *Enum for CSI Packet Name.*

### 6.2.1 Detailed Description

Include file containing C++ API Types used by MIPI_CSI Transactor.

# Index


~CSI
ZEBU_IP::MIPI_CSI::CSI, 26
_SensorMode_NotSet
ZEBU_IP::MIPI_CSI, 16

buildImage
ZEBU_IP::MIPI_CSI::CSI, 26

CALLBACK_ENABLE
ZEBU_IP::MIPI_-CSI::CCIStatusRegister_t, 17
CCI_UPDATE_DONE
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54
CCI_WRITE_MODIFY_PENDING
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54
checkCSITxCharacteristics
ZEBU_IP::MIPI_CSI::CSI, 26
checkWatchdogs
ZEBU_IP::MIPI_CSI::CSI, 26
closeMonitorFile_CSI
ZEBU_IP::MIPI_CSI::CSI, 26
Colorbar
ZEBU_IP::MIPI_CSI, 16
configRestore
ZEBU_IP::MIPI_CSI::CSI, 27
CSI
ZEBU_IP::MIPI_CSI::CSI, 26
CSI.hh, 57
VS_SO_OBSOLETE, 57
CSI_Packet_Name_t
ZEBU_IP::MIPI_CSI, 14
CSI_pixel_t
ZEBU_IP::MIPI_CSI, 14
CSI_SEQ_STATE
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54
CSI_Struct.hh, 58
CSIServiceLoop
ZEBU_IP::MIPI_CSI::CSI, 27

defineFrontPorchSync
ZEBU_IP::MIPI_CSI::CSI, 27
defineRefClkFreq
ZEBU_IP::MIPI_CSI::CSI, 27
disableSequencer
ZEBU_IP::MIPI_CSI::CSI, 28
displayCCIRegister
ZEBU_IP::MIPI_CSI::CSI, 28
displayCSIStatus
ZEBU_IP::MIPI_CSI::CSI, 28
displayInfoDPHY
ZEBU_IP::MIPI_CSI::CSI, 28

enableCCIAddr
ZEBU_IP::MIPI_CSI::CSI, 28
enableCCIAddrRange
ZEBU_IP::MIPI_CSI::CSI, 28
enableDPHYInterface
ZEBU_IP::MIPI_CSI::CSI, 29
enableSequencer
ZEBU_IP::MIPI_CSI::CSI, 29
enableWatchdog
ZEBU_IP::MIPI_CSI::CSI, 29

File_Format_NotSet
ZEBU_IP::MIPI_CSI, 15
FILE_RAW16
ZEBU_IP::MIPI_CSI, 15
FILE_RAW8
ZEBU_IP::MIPI_CSI, 15
FILE_RGB16
ZEBU_IP::MIPI_CSI, 15
FILE_RGB8
ZEBU_IP::MIPI_CSI, 15
FILE_YUV422_8
ZEBU_IP::MIPI_CSI, 15
FirstDriver
ZEBU_IP::MIPI_CSI::CSI, 29
firstDriver
ZEBU_IP::MIPI_CSI::CSI, 29
flushCSIPacket
ZEBU_IP::MIPI_CSI::CSI, 30
FRAME_DONE
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54
Frame_line_t
ZEBU_IP::MIPI_CSI, 14
Frame_pixel_t

ZEBU_IP::MIPI_CSI, 14
FRAME_SENDING
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54

getAllCCIRegister
ZEBU_IP::MIPI_CSI::CSI, 30
getCCIAddressMode
ZEBU_IP::MIPI_CSI::CSI, 30
getCCIRegister
ZEBU_IP::MIPI_CSI::CSI, 30, 31
getCCISlaveAddress
ZEBU_IP::MIPI_CSI::CSI, 31
getCCIStatusRegister
ZEBU_IP::MIPI_CSI::CSI, 31
getColorbarParam
ZEBU_IP::MIPI_CSI::CSI, 31
getCSIBlankingDPHYPeriod
ZEBU_IP::MIPI_CSI::CSI, 32
getCSIBlankingTiming
ZEBU_IP::MIPI_CSI::CSI, 32
getCSIClkDivider
ZEBU_IP::MIPI_CSI::CSI, 32
getCSIStatus
ZEBU_IP::MIPI_CSI::CSI, 32
getCurrentCycle
ZEBU_IP::MIPI_CSI::CSI, 33
getFrameNumber
ZEBU_IP::MIPI_CSI::CSI, 33
getFrameRate
ZEBU_IP::MIPI_CSI::CSI, 33
getHWinfo
ZEBU_IP::MIPI_CSI::CSI, 33
getImageRegion
ZEBU_IP::MIPI_CSI::CSI, 33
GetInstanceName
ZEBU_IP::MIPI_CSI::CSI, 34
getInstanceName
ZEBU_IP::MIPI_CSI::CSI, 34
getLaneModelVersion
ZEBU_IP::MIPI_CSI::CSI, 34
getLineInFrame
ZEBU_IP::MIPI_CSI::CSI, 34
getName
ZEBU_IP::MIPI_CSI::CSI, 35
getNbLaneDPHY
ZEBU_IP::MIPI_CSI::CSI, 35
getNextCCIRegisterModify
ZEBU_IP::MIPI_CSI::CSI, 35
getNumberPendingCCI
ZEBU_IP::MIPI_CSI::CSI, 35
getPacketStatistics
ZEBU_IP::MIPI_CSI::CSI, 36
getPixelPacking
ZEBU_IP::MIPI_CSI::CSI, 36
getPPIByteClk_Freq
ZEBU_IP::MIPI_CSI::CSI, 36
getPPIByteClk_MinFreq
ZEBU_IP::MIPI_CSI::CSI, 36
getRealFrameRate
ZEBU_IP::MIPI_CSI::CSI, 36
getSensorMode
ZEBU_IP::MIPI_CSI::CSI, 37
getSequencer
ZEBU_IP::MIPI_CSI::CSI, 37
getVideoMode
ZEBU_IP::MIPI_CSI::CSI, 37
getVideoSync
ZEBU_IP::MIPI_CSI::CSI, 37
getVideoTiming
ZEBU_IP::MIPI_CSI::CSI, 38
getVirtualPixelClkFreq
ZEBU_IP::MIPI_CSI::CSI, 38
getXtorVersion
ZEBU_IP::MIPI_CSI::CSI, 38

IDLE
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54
init
ZEBU_IP::MIPI_CSI::CSI, 38
IsDriverPresent
ZEBU_IP::MIPI_CSI::CSI, 39
isDriverPresent
ZEBU_IP::MIPI_CSI::CSI, 39

LINE_DONE
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54
LINE_SENDING
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 55

MIPI_CSI, 11
MONITORING_ENABLE
ZEBU_IP::MIPI_-CSI::CCIStatusRegister_t, 17

NextDriver
ZEBU_IP::MIPI_CSI::CSI, 39
nextDriver
ZEBU_IP::MIPI_CSI::CSI, 39

openMonitorFile_CCI
ZEBU_IP::MIPI_CSI::CSI, 39
openMonitorFile_CSI
ZEBU_IP::MIPI_CSI::CSI, 40

P_Blanking_Data

ZEBU_IP::MIPI_CSI, 14
P_Embedded_8bit_non_Image_Data
ZEBU_IP::MIPI_CSI, 14
P_Frame_End
ZEBU_IP::MIPI_CSI, 14
P_Frame_Start
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code1
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code2
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code3
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code4
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code5
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code6
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code7
ZEBU_IP::MIPI_CSI, 14
P_Generic_Short_Packet_Code8
ZEBU_IP::MIPI_CSI, 14
P_Legacy_YUV420_8bit
ZEBU_IP::MIPI_CSI, 15
P_Line_End
ZEBU_IP::MIPI_CSI, 14
P_Line_Start
ZEBU_IP::MIPI_CSI, 14
P_Null
ZEBU_IP::MIPI_CSI, 14
P_RAW10
ZEBU_IP::MIPI_CSI, 15
P_RAW12
ZEBU_IP::MIPI_CSI, 15
P_RAW14
ZEBU_IP::MIPI_CSI, 15
P_RAW6
ZEBU_IP::MIPI_CSI, 15
P_RAW7
ZEBU_IP::MIPI_CSI, 15
P_RAW8
ZEBU_IP::MIPI_CSI, 15
P_RGB444
ZEBU_IP::MIPI_CSI, 15
P_RGB555
ZEBU_IP::MIPI_CSI, 15
P_RGB565
ZEBU_IP::MIPI_CSI, 15
P_RGB666
ZEBU_IP::MIPI_CSI, 15
P_RGB888
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type1
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type2
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type3
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type4
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type5
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type6
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type7
ZEBU_IP::MIPI_CSI, 15
P_User_Defined_8bit_Data_Type8
ZEBU_IP::MIPI_CSI, 15
P_YUV420_10bit
ZEBU_IP::MIPI_CSI, 15
P_YUV420_10bit_Chroma_Shifted_Pixel_-
Sampling
ZEBU_IP::MIPI_CSI, 15
P_YUV420_8bit
ZEBU_IP::MIPI_CSI, 15
P_YUV420_8bit_Chroma_Shifted_Pixel_-
Sampling
ZEBU_IP::MIPI_CSI, 15
P_YUV422_10bit
ZEBU_IP::MIPI_CSI, 15
P_YUV422_8bit
ZEBU_IP::MIPI_CSI, 15
Pixel_Format_NotSet
ZEBU_IP::MIPI_CSI, 16
Pixel_File_Format_t
ZEBU_IP::MIPI_CSI, 15
Pixel_Format_t
ZEBU_IP::MIPI_CSI, 15

RAW
ZEBU_IP::MIPI_CSI, 16
RAW10
ZEBU_IP::MIPI_CSI, 16
RAW12
ZEBU_IP::MIPI_CSI, 16
RAW14
ZEBU_IP::MIPI_CSI, 16
RAW16
ZEBU_IP::MIPI_CSI, 16
RAW6
ZEBU_IP::MIPI_CSI, 16
RAW7
ZEBU_IP::MIPI_CSI, 16
RAW8
ZEBU_IP::MIPI_CSI, 16
registerCCI_CB_Addr
ZEBU_IP::MIPI_CSI::CSI, 40

registerCCI_CB_AddrRange
ZEBU_IP::MIPI_CSI::CSI, 40
registerTimeoutCB
ZEBU_IP::MIPI_CSI::CSI, 40
registerUserCB
ZEBU_IP::MIPI_CSI::CSI, 41
restartMonitorFile_CSI
ZEBU_IP::MIPI_CSI::CSI, 41
RGB
ZEBU_IP::MIPI_CSI, 16
RGB444
ZEBU_IP::MIPI_CSI, 16
RGB555
ZEBU_IP::MIPI_CSI, 16
RGB565
ZEBU_IP::MIPI_CSI, 16
RGB666
ZEBU_IP::MIPI_CSI, 16
RGB888
ZEBU_IP::MIPI_CSI, 16
RGBFFF
ZEBU_IP::MIPI_CSI, 16
runCycle
ZEBU_IP::MIPI_CSI::CSI, 41

save
ZEBU_IP::MIPI_CSI::CSI, 41
sendImage
ZEBU_IP::MIPI_CSI::CSI, 42
sendLine
ZEBU_IP::MIPI_CSI::CSI, 42
sendLongPacket
ZEBU_IP::MIPI_CSI::CSI, 42, 43
sendRawDataPacket
ZEBU_IP::MIPI_CSI::CSI, 43
sendShortPacket
ZEBU_IP::MIPI_CSI::CSI, 43, 44
SensorMode_t
ZEBU_IP::MIPI_CSI, 16
setAllCCIRegister
ZEBU_IP::MIPI_CSI::CSI, 44
setCCIAddressMode
ZEBU_IP::MIPI_CSI::CSI, 44
setCCIRegister
ZEBU_IP::MIPI_CSI::CSI, 45
setCCISlaveAddress
ZEBU_IP::MIPI_CSI::CSI, 45
setColorbar
ZEBU_IP::MIPI_CSI::CSI, 45
setCSIClkDivider
ZEBU_IP::MIPI_CSI::CSI, 46
setDebugLevel
ZEBU_IP::MIPI_CSI::CSI, 46
setFrameRate
ZEBU_IP::MIPI_CSI::CSI, 46
setImageRegion
ZEBU_IP::MIPI_CSI::CSI, 46
setImageRotate
ZEBU_IP::MIPI_CSI::CSI, 47
setImageZoom
ZEBU_IP::MIPI_CSI::CSI, 47
setInputFile
ZEBU_IP::MIPI_CSI::CSI, 47
setLog
ZEBU_IP::MIPI_CSI::CSI, 48
setName
ZEBU_IP::MIPI_CSI::CSI, 48
setNbCycleClkRqstDPHY
ZEBU_IP::MIPI_CSI::CSI, 48, 49
setNbLaneDPHY
ZEBU_IP::MIPI_CSI::CSI, 49
setPixelPacking
ZEBU_IP::MIPI_CSI::CSI, 49
setSensorMode
ZEBU_IP::MIPI_CSI::CSI, 49
setTimeout
ZEBU_IP::MIPI_CSI::CSI, 50
setTransform
ZEBU_IP::MIPI_CSI::CSI, 50
setVideoJitter
ZEBU_IP::MIPI_CSI::CSI, 50
setVideoProfileH
ZEBU_IP::MIPI_CSI::CSI, 50
setVideoProfileV
ZEBU_IP::MIPI_CSI::CSI, 50
setVirtualChannelID
ZEBU_IP::MIPI_CSI::CSI, 51
setZebuPortGroup
ZEBU_IP::MIPI_CSI::CSI, 51
stopMonitorFile_CSI
ZEBU_IP::MIPI_CSI::CSI, 51

unRegisterCCI_CB_Addr
ZEBU_IP::MIPI_CSI::CSI, 51
unRegisterCCI_CB_AddrRange
ZEBU_IP::MIPI_CSI::CSI, 51
updateCCIRegister
ZEBU_IP::MIPI_CSI::CSI, 52
useLineSyncPacket
ZEBU_IP::MIPI_CSI::CSI, 52
useZebuServiceLoop
ZEBU_IP::MIPI_CSI::CSI, 52, 53
usleep
ZEBU_IP::MIPI_CSI::CSI, 53

VS_SO_OBSOLETE
CSI.hh, 57

YUV
ZEBU_IP::MIPI_CSI, 16
YUV420_10
ZEBU_IP::MIPI_CSI, 16
YUV420_10_CSPS
ZEBU_IP::MIPI_CSI, 16
YUV420_8
ZEBU_IP::MIPI_CSI, 16
YUV420_8_CSPS
ZEBU_IP::MIPI_CSI, 16
YUV420_8_legacy
ZEBU_IP::MIPI_CSI, 16
YUV422_10
ZEBU_IP::MIPI_CSI, 16
YUV422_8
ZEBU_IP::MIPI_CSI, 16

ZEBU_IP::MIPI_CSI
_SensorMode_NotSet, 16
Colorbar, 16
File_Format_NotSet, 15
FILE_RAW16, 15
FILE_RAW8, 15
FILE_RGB16, 15
FILE_RGB8, 15
FILE_YUV422_8, 15
P_Blanking_Data, 14
P_Embedded_8bit_non_Image_Data, 14
P_Frame_End, 14
P_Frame_Start, 14
P_Generic_Short_Packet_Code1, 14
P_Generic_Short_Packet_Code2, 14
P_Generic_Short_Packet_Code3, 14
P_Generic_Short_Packet_Code4, 14
P_Generic_Short_Packet_Code5, 14
P_Generic_Short_Packet_Code6, 14
P_Generic_Short_Packet_Code7, 14
P_Generic_Short_Packet_Code8, 14
P_Legacy_YUV420_8bit, 15
P_Line_End, 14
P_Line_Start, 14
P_Null, 14
P_RAW10, 15
P_RAW12, 15
P_RAW14, 15
P_RAW6, 15
P_RAW7, 15
P_RAW8, 15
P_RGB444, 15
P_RGB555, 15
P_RGB565, 15
P_RGB666, 15
P_RGB888, 15
P_User_Defined_8bit_Data_Type1, 15
P_User_Defined_8bit_Data_Type2, 15
P_User_Defined_8bit_Data_Type3, 15
P_User_Defined_8bit_Data_Type4, 15
P_User_Defined_8bit_Data_Type5, 15
P_User_Defined_8bit_Data_Type6, 15
P_User_Defined_8bit_Data_Type7, 15
P_User_Defined_8bit_Data_Type8, 15
P_YUV420_10bit, 15
P_YUV420_10bit_Chroma_Shifted_-Pixel_Sampling, 15
P_YUV420_8bit, 15
P_YUV420_8bit_Chroma_Shifted_Pixel_-Sampling, 15
P_YUV422_10bit, 15
P_YUV422_8bit, 15
Pixel_Format_NotSet, 16
RAW, 16
RAW10, 16
RAW12, 16
RAW14, 16
RAW16, 16
RAW6, 16
RAW7, 16
RAW8, 16
RGB, 16
RGB444, 16
RGB555, 16
RGB565, 16
RGB666, 16
RGB888, 16
RGBFFF, 16
YUV, 16
YUV420_10, 16
YUV420_10_CSPS, 16
YUV420_8, 16
YUV420_8_CSPS, 16
YUV420_8_legacy, 16
YUV422_10, 16
YUV422_8, 16
ZEBU_IP, 12
ZEBU_IP::MIPI_CSI, 13
CSI_Packet_Name_t, 14
CSI_pixel_t, 14
Frame_line_t, 14
Frame_pixel_t, 14
Pixel_File_Format_t, 15
Pixel_Format_t, 15
SensorMode_t, 16
ZEBU_IP::MIPI_CSI::CCIStatusRegister_t, 17
CALLBACK_ENABLE, 17
MONITORING_ENABLE, 17
ZEBU_IP::MIPI_CSI::CSI, 18
~CSI, 26
buildImage, 26

checkCSITxCharacteristics, 26
checkWatchdogs, 26
closeMonitorFile_CSI, 26
configRestore, 27
CSI, 26
CSIServiceLoop, 27
defineFrontPorchSync, 27
defineRefClkFreq, 27
disableSequencer, 28
displayCCIRegister, 28
displayCSIStatus, 28
displayInfoDPHY, 28
enableCCIAddr, 28
enableCCIAddrRange, 28
enableDPHYInterface, 29
enableSequencer, 29
enableWatchdog, 29
FirstDriver, 29
firstDriver, 29
flushCSIPacket, 30
getAllCCIRegister, 30
getCCIAddressMode, 30
getCCIRegister, 30, 31
getCCISlaveAddress, 31
getCCIStatusRegister, 31
getColorbarParam, 31
getCSIBlankingDPHYPeriod, 32
getCSIBlankingTiming, 32
getCSIClkDivider, 32
getCSIStatus, 32
getCurrentCycle, 33
getFrameNumber, 33
getFrameRate, 33
getHWinfo, 33
getImageRegion, 33
GetInstanceName, 34
getInstanceName, 34
getLaneModelVersion, 34
getLineInFrame, 34
getName, 35
getNbLaneDPHY, 35
getNextCCIRegisterModify, 35
getNumberPendingCCI, 35
getPacketStatistics, 36
getPixelPacking, 36
getPPIByteClk_Freq, 36
getPPIByteClk_MinFreq, 36
getRealFrameRate, 36
getSensorMode, 37
getSequencer, 37
getVideoMode, 37
getVideoSync, 37
getVideoTiming, 38
getVirtualPixelClkFreq, 38
getXtorVersion, 38
init, 38
IsDriverPresent, 39
isDriverPresent, 39
NextDriver, 39
nextDriver, 39
openMonitorFile_CCI, 39
openMonitorFile_CSI, 40
registerCCI_CB_Addr, 40
registerCCI_CB_AddrRange, 40
registerTimeoutCB, 40
registerUserCB, 41
restartMonitorFile_CSI, 41
runCycle, 41
save, 41
sendImage, 42
sendLine, 42
sendLongPacket, 42, 43
sendRawDataPacket, 43
sendShortPacket, 43, 44
setAllCCIRegister, 44
setCCIAddressMode, 44
setCCIRegister, 45
setCCISlaveAddress, 45
setColorbar, 45
setCSIClkDivider, 46
setDebugLevel, 46
setFrameRate, 46
setImageRegion, 46
setImageRotate, 47
setImageZoom, 47
setInputFile, 47
setLog, 48
setName, 48
setNbCycleClkRqstDPHY, 48, 49
setNbLaneDPHY, 49
setPixelPacking, 49
setSensorMode, 49
setTimeout, 50
setTransform, 50
setVideoJitter, 50
setVideoProfileH, 50
setVideoProfileV, 50
setVirtualChannelID, 51
setZebuPortGroup, 51
stopMonitorFile_CSI, 51
unRegisterCCI_CB_Addr, 51
unRegisterCCI_CB_AddrRange, 51
updateCCIRegister, 52
useLineSyncPacket, 52
useZebuServiceLoop, 52, 53
usleep, 53
ZEBU_IP::MIPI_CSI::CSIEventStatus_t, 54
CCI_UPDATE_DONE, 54

CCI_WRITE_MODIFY_PENDING, 54
CSI_SEQ_STATE, 54
FRAME_DONE, 54
FRAME_SENDING, 54
IDLE, 54
LINE_DONE, 54
LINE_SENDING, 55