MIPI-CSI2 Driver Software User Guide

ABSTRACT:

This is the Software User Guide Document for MIPI-CSI2 driver for Linux OS.

KEYWORDS:

MIPI, CSI, ISP, Driver, S32V234

APPROVED:

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	24-March-16	Tomas Babinec	Document creation.
0.2	10-August-16	Tomas Babinec	Updated based on review findings.
0.3	31-August-16	Tomas Babinec	Updated for VSDK 0.9.5 release.
0.4	5-April-17	Roman Kubica	Updated for VSDK RTM
1.0	15-June-17	Roman Kubica	Updated for VSDK RTM
1.1	23-October-17	Tomas Babinec	Added soft reset description
1.2	01-February-18	Cuong Nguyen	Updated to LLDCMD
1.3	15-Aug-18	Dat Vu Van	Update for VSDK RTM 1.2.0: section 3.1 Data type: add structures data type for CSI, 3.2 Kernel Space: Add APIs for kernel space, 3.3 User Space: add APIs for user space

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe MIPI-CSI2 driver user space interface. It is intended to serve as a reference source during the development of VSDK based application. For exact definitions and implementation details please refer to [2].

1.2 Audience Description

This document is intended for internal use by S23V234 Vision SDK developers.

1.3 References

Id	***Title***	***Location***
[1]	*MIPI-CSI2 driver source code documentation*	*VisionSDK folder:* s32v234_sdk\libs\isp\csi
[2]	*SDI SW User Guide*	*VisionSDK folder:* s32v234_sdk\docs\drivers\
[3]	*s32v234 Reference manual*	Available on demand

Table 1: References Table

1.4 Definitions, Acronyms, and Abbreviations

Term/Acronym	***Description***
API	*Application Programming Interface*
CSI2	*Camera Serial Interface 2*
DDR	*Double Data Rate DRAM*
DRAM	*Dynamic Random Access Memory*
fDMA	*fast Direct Memory Access HW block*
HW	*Hardware*
IP	*Intellectual Property*
ISP	*Image signal processor (whole image processing system)*
SDI	*Sensor Data Interface library*
Sequencer	*Microprocessor to control the ISP engines and memory*

Sequencer Graph	*SW description of data flow and processing which can be executed by the ISP*
SoC	*System on Chip*
SW	*Software*
TD	*Transaction Descriptor*

Table 2: Acronyms Table

1.5 Document Location

This document is available in VisionSDK directory structure at the following location s32v234_sdk\docs\drivers\.

2 General Description

The MIPI-CSI2 (further referred to as MIPI or CSI) driver software (SW) is intended for kernel space management of the MIPI-CSI2 HW block, which is designed to be a part of the S32V234 SoC. An integral part of the driver is also a user space library providing an API for the user applications. This API wraps the kernel space interface of the driver (LLDCMDs, etc.).

2.1 MIPI-CSI2 HW

There are two MIPI-CSI2 interfaces available in the S32V234 SoC to capture data from external sensors. Each interface incorporates a four lane physical layer and one clock lane, compliant with the MIPI Alliance Standard for D-PHY.

Main features are as follows:

- Up to 1.5 Gbits/s on each individual D-PHY receive lane.
- Frame and line synchronization packet signaling.
- Input data types: RGB888, YUV422 8bit/10bit, RAW8, RAW10, RAW12, RAW14, user defined and embedded data.
- Support of up to 4 virtual channels (VC).
- Signal trigger to sequencer after programmable number of image lines written to SRAM.
- Frame Start/End IRQs to host CPU per VC.

3 Functional Description

The MIPI-CSI2 driver SW has two layers (see Figure 1). The first layer operates in kernel space and accomplishes most of the driver's functionality.

The second layer is implemented as a user space library creating a thin interface for user level SW (SDI or directly a user application) to access the kernel part functionality. The provided user level API is explained in section 3.3.

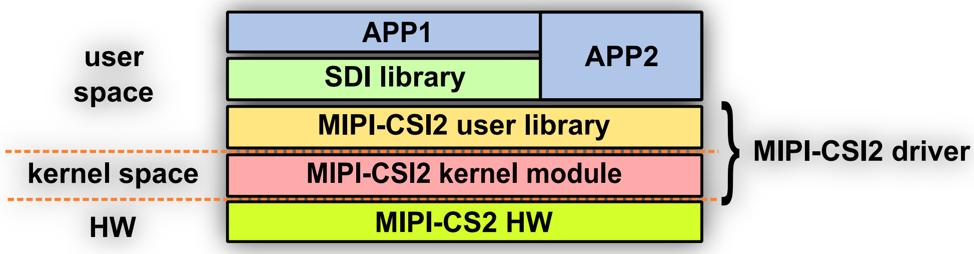

Figure 1: MIPI-CSI2 driver software layout

3.1 Data types CSI

The MIPI-CSI2 driver introduces the following data types and containers (see [1] for full definitions):

- Enum `CSI_IDX`:
  Enumerates possible indexes of the CSI receiver interface.
- Enums `CSI_VCID`:
  Enumerates possible indexes of the CSI virtual channels.
- Union CSI_IdKey:
  32 bits to define what CSI virtual channels are used.
- Enum `CSI_IRQ_TYPE`:
  Enumerates supported IRQs.
- Enum *CSI_EMBD_IRQ_TYPE*

  Enumerates possible indexes of the CSI embedded IRQ
- Enum *CSI_PPERR_IRQ_TYPE*

  Enumerates possible indexes error type PPERR IRQ
- Enum *CSI_EMBD_STATUS*

  Enumerates possible indexes of the CSI embedded data status
- Structure `CSI_VcCfg`:
  Describes configuration parameters of one CSI virtual channels.
- Structure `CSI_EmbdCfg`:

Describes embedded data reception configuration.

- Structure `CSI_Cfg`:
  Describes configuration parameters of on CSI receiver interface.
- Structure *CSI_EmbdStatus*

  Describes configuration parameters of on CSI embedded status
- Structure CSI_ErrStatus

  Describes configuration parameters of on CSI error status
- Structure `CSI_IrqNums`:
  Used to store current status of CSI ISR registers.
- Structure `CSI_DrvInfo`:
  Used to store current status of the CSI driver including CSI registers.

3.2 Kernel Space

The internal functionality of the MIPI-CSI2 kernel module and its API manage the low level HW communication and make the MIPI-CSI2 HW features available for user applications.

3.2.1 API functions

This section, Table 3, describes functionality exported by the MIPI-CSI2 driver module. It is intended to be used by upper layer SW such as IO control interface creation in case of Linux environment or directly by the user library in case of a standalone setup. Exact function headers declarations can be found in `csi_func.h`.

In Linux environment the MIPI-CSI2 driver is associated with special device file `csi`.

Function; LLDCMDs	Description
`CSI_DRV_Setup`	First time use initialization. To enable HW interaction and setup internal structures. In Linux invoked when `csi` device file opened.
`CSI_DRV_Close`	To terminate driver operations. Release of all resources, reset of internal structures. In Linux invoked when `csi` device file closed.
`CSI_DRV_Config`	Sets up the CSI registers based on the `CSI_Cfg_t` structure provided as parameter.
`CSI_DRV_EmbdConfig`	Configures MIPI-CSI2 embedded data reception based on the `CSI_EmbdCfg_t` structure provided as parameter.
`CSI_DRV_EmbdRecap`	Requests new embedded data to be captured on particular CSI interface index provided as parameter.
`CSI_DRV_EmbdStatusGet`	Gets status of embedded data capturing on particular CSI interface index provided as parameter. **Returns:** CSI_EMBD_STATUS_NA if nothing available combination of flags CSI_EMBD_STATUS_1 and/or CSI_EMBD_STATUS_2 if some data captured.
`CSI_DRV_ErrStatusGet`	Gets error registers status.
`CSI_DRV_ErrClear`	Clear all so far captured errors.
`CSI_DRV_Start`	Turns on the MIPI receiver functionality. **NOTE:** MIPI CSI receiver should be enabled before the cam starts to generate data.
`CSI_DRV_SwReset`	Requests a soft reset of the CSI IP. All registers are reset and all FIFOs are flushed.
`CSI_DRV_Stop`	Disables MIPI-CSI2 receiver operation
CSI_IrqHandlerRegister	Registers new IRQ handler for particular CSI interface.

Table 3: MIPI-CSI2 driver API

3.2.2 Usage CSI

Before configuring the MIPI-CSI2 interface for a new data reception session, CSI_DRV_SwReset is expected to be called, to ensure that there are no pending or paused CSI receptions.

The MIPI-CSI2 interface parameters can be configured using the `CSI_DRV_Config` function, where the parameters are passed in a `CSI_Cfg_t` structure pointer.

For embedded data reception, the `CSI_EmbdCfg_t` structure has to be filled in. It requires a SRAM buffer to be allocated to the expected size of the data configured at the sensor side. To apply the prefilled embedded configuration `CSI_DRV_EmbdConfig` functions has to be invoked.

Other kernel space SW can use the `CSI_DRV_IrqHandlerSet` function to register callbacks to be executed as part of the CSI IRQ handling.

To start receiving the camera data the `CSI_DRV_Start` function has to be invoked. Its parameter specifies the CSI interface index and virtual channels to be enabled. It is important that the camera stream is on before the virtual channels on receiver side are enabled.

While the CSI stream is on embedded data capture can be requested using `CSI_DRV_EmbdRecap` function. This function is non-blocking. To get status information about embedded data capturing the `CSI_DRV_EmbdStatusGet` function can be used. It returns `CSI_EMBD_STATUS_NA` in case no data have are available yet or the capturing is in progress. `CSI_EMBD_STATUS_1` is returned in case embedded data at the beginning of the frame have been captured or `CSI_EMBD_STATUS_2` for data at the end have been captured.

To stop the data being received and written to the SRAM the `CSI_DRV_Stop` has to be invoked.

3.3 User Space

The MIPI-CSI2 driver SW includes a thin user space library to abstract kernel space driver from user space. The user space library functions invoke the kernel space functionality described in previous sections.

3.3.1 API

Function	Description
`CSI_Open`	Opens CSI special device file on Linux or calls `CSI_DRV_Setup` in case of standalone environment.
`CSI_Close`	Closes CSI special device file on Linux or calls `CSI_DRV_Close` in case of standalone environment.
`CSI_Config`	Configures the CSI registers based on the provided `CSI_Cfg` structure.
`CSI_EmbdConfig`	Configures CSI embedded data reception based on the `CSI_EmbdCfg_t` structure provided as parameter.
`CSI_EmbdRecap`	Requests new embedded data to be captured.

CSI_EmbdStatusGet	Gets status of embedded data capturing.
CSI_ErrStatusGet	Gets CSI error status.
CSI_ErrClear	Clears all CSI error captured so far and reenables detection. Note: Only first detected error is signaled through RT signal. CSI_ErrClear re-enables the signaling.
CSI_Start	Enables CSI receiver.
CSI_SwReset	Requests CSI soft reset.
CSI_Stop	Disables CSI receiver.
CSI_EventHandlerSet	Register event handler.
CSI_IrqHandlerSet	Standalone environment only! Register event handler object.

Table 4: MIPI-CSI2 user library exported functions

4 High Level Design

4.1 System Decomposition

The MIPI-CSI2 HW and its driver belong to the complex data preprocessing subsystem of the s32v234 SoC that is wrapped and controlled by the SDI library. Part of this subsystem is visualized in Figure 1. For more information about SDI and data preprocessing please refer to [2].

The preferred way to use the MIPI-CSI2 functionality in a user application is to use Sequencer graphs together with the SDI library services. The SDI library provides complete abstraction of the MIPI-CSI2 driver interface and thanks to utilization of the Sequencer HW the data flow management load for the host CPU is minimized.

At the moment the CSI driver is compiled into one library together with the code for all supported camera functionality. These camera drivers are out of scope of this document and their source code will be separated in the future.

4.2 File Structure

MIPI-CSI2 driver code is located in VSDK under s3234_sdk/libs/isp/csi folder. Internally it has the following structure:

- kernel
  - build-v234ce-gnu-linux-d – build folder for Linux kernel module
    - Makefile
  - include
    - csi_func.h – declaration of MIPI-CSI2 driver functionality
    - csi_ lldcmd.h – declaration of LLDCMD codes
    - csi_types.h – declaration of MIPI-CSI2 related data types
    - csi.h – general MIPI-CSI2 related declarations/definitions
  - src
    - csi_core.c – Linux module related functionality
    - csi_func.c – definition of the MIPI-CSI2 driver functionality
    - csi_ lldcmd.c – definition of LLDCMD handling
- user
  - build-* – build folders for supported platforms (standalone and Linux)
    - Makefile
  - src
    - csi_user.cpp – definition of user space level public API,
  - BUILD.mk – defines build details
- Public headers (s32v234_sdk/include):

- isp_csi.h – declaration of user space level public API,

4.3 Module Usage

Not available