VIU Driver Software User Guide

ABSTRACT:
This is the Software User Guide Document for VIU Driver for Linux OS.
KEYWORDS:
User Guide
APPROVED:

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	20-September-16	Cristian Tomescu	First draft
0.2	17-October-16	Cristian Tomescu	Changes after review
1.0	1-February-17	Cristian Tomescu	Update for RTM
1.1	12-March-18	Loc Nguyen	Update for RTM 1.1: use low level driver command instead of ioctl command
1.2	15-Aug-18	Dat Vu Van	Update for RTM 1.2: Update Section 1.5 Document Location, section 3.2.2 Usage

Table of Contents

1 Introduction

The purpose of this document is to describe the VIU driver interface. It is intended to serve as a reference source during the development of VSDK based application.

1.1 Purpose

The purpose of this document is to define VIU driver internal behavior and user space interface. It is intended to serve as a reference source during the driver implementation and future use. For exact definitions and implementation details please check references and source code.

1.2 Audience Description

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

1.3 References

Id	*Title*	*Location*
[1]	*SDI SW User Guide*	*Vision sdk git, folder:* s32v234_sdk\docs\drivers
[2]	*S32v234 Reference Manual*	Available on demand

Table 1: References

1.4 Definitions, Acronyms and Abbreviations

Term/Acronym	*Description*
API	*Application Programming Interface*
HW	*Hardware*
ISP	*Image signal processor (whole image processing system)*
LLD	*Low Level Driver*
SDI	*Sensor Data Interface library*
SW	*Software*
VIU	*Video In Unit*

Table 2: Acronyms

1.5 Document Location

This document is available in VisionSDK directory structure at the following location:

VisionSDK: s32v234_sdk/docs/drivers

2 General Description

The VIU driver software (SW) is intended for kernel space management of both VIULite HW modules, which are designed to be 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 by LLD commands.

3 Functional Description

The VIU driver SW has 2 layers (see Figure 1). The first layer operates in kernel space and implements functionality using all HW resources. Internal behavior of the kernel space layer will be described in detail in section 3.2.

The second layer is implemented as a user space abstraction layer for the low-level kernel driver API. This layer is designated as VIU user library. The provided user level API is explained in section 3.3.

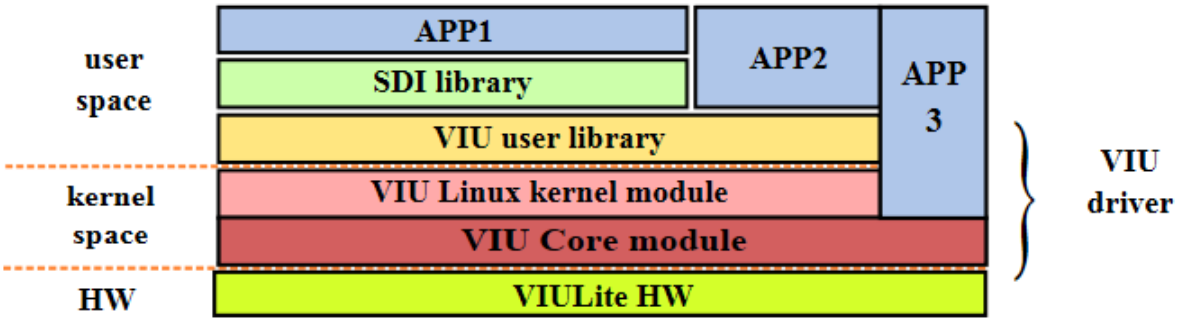

Figure 1: VIU driver software layout

3.1 Data Types Generic

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

- Enumeration `VIU_BOOL`:
  Enumerates possible values for logical variables.
- Structure `VIU_DATA_INTERFACE`:
  Describes the input interface configuring parameters.
- Structure `VIU_INPUT_FORMAT`:
  Describes the input video data parameters.
- Structure `DMA_CONFIG`:
  Describes the parameters of the data transfer.
- Structure `VIU_SYNC_STATUS`:
  Describes the status of the vertical and horizontal synchronization signals.
- Structure `VIU_FRAME_SIZE`:
  Describes the frame parameters: number of lines and number of pixels/line.
- Structure `VIU_IMAGE_PARAMS`:
  Describes the clipped image parameters (origin coordinates, x size and y size).
- Structure `VIU_Registers_t`:

Describes the register bank.

- Enumeration `VIU_ITU_ERR`:
  Enumerates the ITU-656 transfer error codes.
- Enumeration `in_cpp_t`:
  Enumerates the clock ticks/pixel possible values.
- Enumeration `in_mode_t`:
  Enumerates the possible camera data transfer modes: PARALLEL or ITU-656.
- Enumeration `in_width_t`:
  Enumerates the possible values for the data input width.

3.2 Kernel Space

3.2.1API Functions

This section, Table 3, describes functionality exported by the VIU 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 (TBD).

In the Linux environment the VIU driver is associated with special device files `viulite0` and `viulite1`.

Function LLD Command	Description
`viulite_set_videoinputformat` `VIULITE_LLDCMD_SET_VIDEOIN_FORMAT`	Sets the input interface with the camera video data parameters.
`viulite_get_videoinputformat` `VIULITE_LLDCMD_GET_VIDEOIN_FORMAT`	Returns the video input parameters the input interface was previously configured with.
`viulite_set_datainterface` `VIULITE_LLDCMD_SET_DATA_INTERFACE`	Sets the input interface with camera specific output interface parameters: signals (vsync, hsync and pixel clock) polarity and data endianness type.
`viulite_get_datainterface` `VIULITE_LLDCMD_GET_DATA_INTERFACE`	Returns the data interface parameters the input interface was previously configured with.
`viulite_dma_config` `VIULITE_LLDCMD_DMA_CONFIG`	Configures the data transfer parameters.
`viulite_dma_start` `VIULITE_LLDCMD_DMA_START`	Starts of the data transfer.
`viulite_dma_stop` `VIULITE_LLDCMD_DMA_STOP`	Stops the data transfer.

viulite_dma_getstatus VIULITE_LLDCMD_DMA_GET_STATUS	Returns the data transfer status running or stopped.
viulite_sw_reset VIULITE_LLDCMD_SW_RESET	Resets the module internal mechanisms without the registers.
viulite_enable_ituerror VIULITE_LLDCMD_EN_ITU_ERRCODE	Enables/disables the ITU-656 specific errors monitoring when ITU-656 mode is set.
viulite_get_ituerror VIULITE_LLDCMD_GET_ITU_ERRCODE	Returns the ITU-656 error code when the ITU mode is set.
viulite_enable_irqs VIULITE_LLDCMD_CONFIG_IRQS	Enables/disables the interrupts according to the bits mask sent as parameter.
viulite_reset_irqstatus VIULITE_LLDCMD_RESET_IRQSTATUS	Resets the interrupt flags according to the bits mask sent as parameter.
viulite_get_irqstatus VIULITE_LLDCMD_GET_IRQSTATUS	Returns the status of the interrupt flags.
viulite_get_syncsignals VIULITE_LLDCMD_GET_SYNC	Returns the status of the synchronization signals.
viulite_get_fieldnum VIULITE_LLDCMD_GET_FIELDNUM	Returns the field number from the ITU-656 data stream in case this mode is set.
viulite_get_framesize VIULITE_LLDCMD_GET_FRAME_SIZE	Returns the detected parameters of the received image frame (number of pixels and number of lines).
viulite_set_clippingdata VIULITE_LLDCMD_SET_CLIPPING	Sets the origin and the size of the desired clipped area.
viulite_get_clippingdata VIULITE_LLDCMD_GET_CLIPPING	Returns the origin and the size previously set for a clipped area

Table 3: VIU kernel driver API

3.2.2 Usage

The VIU interface can be configured for the used camera using the functions `viulite_set_videoinputformat` to set the HW interface parameters (passed in the `VIU_DATA_INTERFACE` data pointer) and the `viulite_set_datainterface` to set the

image data format (passed in the VIU_INPUT_FORMAT data pointer). The input interface configuration can be checked during using the driver by calling the viulite_get_videoinputformat and the viulite_get_datainterface functions. The data transfer can be configured using the viulite_dma_config function and the parameter values filled in the DMA_CONFIG structure. It is controlled with the viulite_dma_start and the viulite_dma_stop functions. The status of the transfer DMA can be inspected by calling the viulite_dma_getstatus function.
The have a continuous video frames transfer to the memory it is a must to enable the VSYNC interrupt. This is done by the viulite_enable_irqs function. The interrupt status flag can be checked using the viulite_get_irqstatus function and reset using the viulite_reset_irqstatus function. The other interrupts are not used.
If the transfer is done in ITU-656 mode some transfer errors can be checked with the viulite_get_ituerror function if the monitoring of this ITU-656 errors is previously enabled by calling the viulite_enable_ituerror function. Transmission parameters like field number which can be read by calling the viulite_get_fieldnum function and/or the status of the synchronization signals which can be read using the viulite_get_syncsignals function.
If an error occurs the HW has be reset using the viulite_sw_reset function.
Transfer of clipped images can be configured using the viulite_set_clippingdata function and the data in the VIU_IMAGE_PARAMS structure.

3.3 User Space

The VIU driver SW includes a user space library to abstract the kernel space driver from user applications. The user space library invokes the kernel space functionality described in the previous section.

3.3.1 Data Types Specific

The VIU user space driver introduces the following data type:

- Enumeration VIU_IDX:
  Enumerates the index of the VIULite HW units.

3.3.2 API Functions

The VIU driver user level API mentioned in Table 44 is declared in isp_viu.h and defined in viulite_user.cpp file.

Function	Description
VIU_Open	Opens the special device file on Linux (“fsl_viulite0” or “fsl_viulite1”, depending on the parameter value).
VIU_Close	Closes the special device file on Linux (“fsl_viulite0” or “fsl_viulite1”, depending on the parameter value).
VIU_Config	Configures the data input interface (the camera data and the hw

	interface parameters) and the data transfer to the memory.
`VIU_IrqConfig`	Enables/disables the used VIULite module interrupts.
`VIU_DmaStart`	Enables the DMA machine.
`VIU_DmaStop`	Stops the DMA transfer.
`VIU_Start`	Enables the frames transfer.
`VIU_Stop`	Suspends the frames transfer.
`VIU_SwReset`	Resets the used VIULite module.

Table 4: VIU user library exported functions

4 High Level Design

4.1 System Decomposition

The VIU driver belongs 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 *[1]*.

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

4.2 File Structure

The VIU driver code is located in VSDK under s32v234_sdk/libs/isp/viu folder. Internally it has the following structure:

- kernel
  - build-v234ce-gnu-linux-d – build folder for Linux kernel module
    - Makefile.
  - include
    - ov10635_types.h – declaration of data type for OV10635 camera driver,
    - ov10635_viu_config. h – declaration of OV10635 camera registers configuration,
    - viulite_linux.h – declaration of kernel space driver functionality,
    - viulite_core.h – declaration of core driver functionality,
    - viulite_types.h – declaration of data types.
  - src
    - viulite_core.c – core related functionality.
    - viulite_linux.c – kernel space driver related functionality.
- user
  - build-* – build folders for the Linux platform,
    - Makefile.
  - src
    - viulite_user.cpp – definition of user space level public API.
  - include
    - viu_types.h – declaration of user space level public API types.
  - BUILD.mk – defines build details
- Public headers (s32v234_sdk/include):

- isp_viu.h – declaration of user space level public API.

4.3 Module Usage

1. Call the VIU_Open function to open the Linux device file corresponding to the VIULite module to be used.
2. Call the VIU_Config function to set the data input interface parameters and to setup the DMA transfer.
3. Call the VIU_IrqConfig function to enable and/or disable the module interrupts. Mandatory the VSYNC interrupt is enabled for continue image frame transfer.
4. Call the VIU_DmaStart to enable the DMA transfer.
5. Call the VIU_Start to start the continue transfer of image frames.
6. The VIU_Stop function is used to stop the transfer of image frames.
7. The VIU_DmaStop disables the DMA transfer.
8. The VIU_SwReset is used to rest the data transfer mechanism in case of transfer error.
9. Call the VIU_Close function to close the Linux device file corresponding to the VIULite module if it shall be no more used.