H264Encoder Driver User Guide

ABSTRACT:
This is the Software User Guide Document for H264 Encoder driver for Linux OS.
KEYWORDS:
H264, Encoder, ISP, Driver, S32V234
APPROVED:

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	20-November-16	Roman Kubica	First draft
1.0	11-July-17	Roman Kubica	Updated for RTM
1.1	20-August-18	Khang Ba Tran	Updated for VSDK 1.2 RTM release

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this document is to define H.264 Encoder driver internal behaviour and user space interface. It is intended to serve as a reference source during the driver implementation and future use.

1.2 Audience Description

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

1.3 References

Id	*Title*	*Location*
[1]	*S32V234 Reference Manual*	*Sharepoint*
[2]	*SDI SW User Guide*	*Vision sdk git*, *folder:* s23v234_sdk\docs\drivers\

Table 1: References Table

1.4 Definitions, Acronyms, and Abbreviations

Term/Acronym	*Description*
ADD	*Architecture Design Document*
API	*Application Programming Interface*
DDR	*Double Data Rate DRAM*
HW	*Hardware*
IP	*Intellectual Property*
SDI	*Sensor Data Interface library*
SDS	*Software Design Specification*
SoC	*System on Chip*
SRAM	*Static Random Access Memory*
SW	*Software*

Table 2: Acronyms Table

1.5 Document Location

This document is available in VisionSDK directory structure at the following location:
s32v234_sdk\docs\drivers\H264_Encoder_User_Guide.pdf

1.6 Problem Reporting Instructions

Problems with or corrections to this document should be reported by email to Roman Kubica roman.kubica@nxp.com.

2 General Description

The H.264 Encoder driver software (SW) is intended for kernel space management of the H.264 Encoder 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 (LLDCMD commands, etc.).

2.1 H.264 Encoder HW

The H.264 Encoder HW was designed for S32V234 SoC to compress digital video data coming from camera or loaded from memory.

Main features are as follows:

- Encoding of H.264 Intra frame (I-frame) – only stream
- Color resolution: 8, 10 or 12 bits per pixel.
- Color format: gray level images, YUV 420, Data Mode Chroma
- Image geometry of up to 2M pixels (e.g., 1920x1080@30)

2.2 H.264 Encoder Driver Goals

From the SW point of view it is expected, the H.264 Encoder driver will be used mainly by the Sensor Data Interface (SDI) library [2] (see Figure 1). In such case the H.264 Encoder functionality is abstracted completely by the SDI library and user applications do not have to access the underlying SW layers (H.264 Encoder driver) directly. Therefore the H.264 Encoder driver design is aimed at accommodating the SDI needs as much as possible.

On the other hand there has to be a possibility for the user applications to directly utilize the H.264 Encoder user-space library and operate the H.264 Encoder HW on their own.

The main goals of the H.264 Encoder driver can be summarized as follows:

- To wrap the H.264 Encoder HW configuration;
- To accommodate SDI specific requirements;
- To report errors signaled by the H.264 Encoder receiver HW;

3 Functional Description

The H.264 Encoder driver SW has two layers (see Figure 1). The first layer operates in kernel space and accomplishes most of the driver's functionality. 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 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.

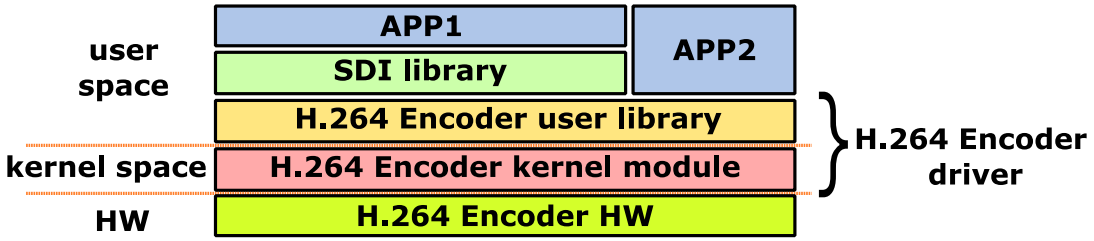

Figure 1: H.264 Encoder driver software layout

3.1 Data types

The H.264 Encoder driver introduces the following data types and containers:

- Structure H264ENC_Buffers_t:
  Describes input and output buffers (addresses and sizes).
- Structure VISION_MemBuf_t:
  Describes one memory buffer
- Structure H264ENC_Config_t:
  Describes all configuration parameters for H.264 Encoder HW setup.
- Structure H264ENC_Status_t:
  Describes current status of encoding.
- Structure H264ENC_IrqNums_t:
  Describes IRQ numbers for corresponding interrupts.
- Structure H264ENC_DrvInfo:
  Stores Encoder control block registers and irq numbers.
- Structure H264ENC_Regs_t:
  Stores Encoder registers.

3.2 Kernel Space

The internal functionality of the H.264 Encoder kernel module and its API manage the low level HW communication and make the H.264 Encoder HW features available for user applications.

To satisfy the goals stated in section 2.2, the H.264 Encoder driver provides the following functionality:

- Initialization
  - Reset of the H.264 HW module and its peripherals;
  - Configuration of H.264 Encoder registers;
- H.264 Encoder event handling
  - IRQs handling and error detection;
- Status reporting
  - Profiling information.

3.2.1API functions

This section, Table 3, describes functionality exported by the H.264 Encoder driver module. It is intended to be used by upper layer SW such as IO control interface creation in case of Linux environment.

In Linux environment the H.264 Encoder driver is associated with special device file `h264enc`.

Function, LLDCMD	Description
`Open()` `---`	First time use initialization. To enable HW interaction and setup internal structures. In Linux invoked when h264enc device file opened
`Release()` `---`	To terminate driver operations. HW reset, release of all resources, reset of internal structures. In Linux invoked when h264enc device file closed
`H264ENC_Stop()` `H264_ENC_STOP`	Reset current encoder operation.
`H264ENC_Config()` `H264_ENC_CONFIG_SET`	Sets H264 Encoder HW configuration. **Parameters:** Structure containing parameters for H264 Encoder HW configuration.
`H264ENC_BuffersSet()` `H264_ENC_BUFFERS_SET`	Sets input and output buffers (sizes and addresses). **Parameters:** Structure containing parameters buffers setup.
`H264ENC_LinesFetch()` `H264_ENC_FETCH`	Fetches macroblock lines for encoding and starts. **Parameters:** Number of lines to be fetched.
`H264ENC_BitstreamAlarmShift` `H264_ENC_BS_ALARM_SHIFT`	Reprogram alarm address **Parameters:** uint32_t `aAlarmShift`
`H264ENC_StatusGet()` `H264_ENC_STATUS_GET`	Gives information of current encoding status. **Parameters:** Structure `h264_enc_status`

`H264ENC_ResetVars()` `LLDCMD_H264_ENC_RESET_VARS`	Resets all global variables and H.264 Encoder HW
`H264ENC_RowDoneStatusGet()` `H264_ENC_BS_ROW_DONE_GET`	Checks that complete macroblock has been encoded. **Parameters:** uint8_t returns 1 if macroblock encoded

Table 3: H.264 Encoder driver API

3.2.2Initialization

The initialization process is schematically displayed in **Figure** 2. When loaded the H.264 Encoder driver kernel module first maps the "H.264 Encoder control/event block" registers and ensures the H.264 Encoder HW is in reset. After that the internal data structures are initialized.

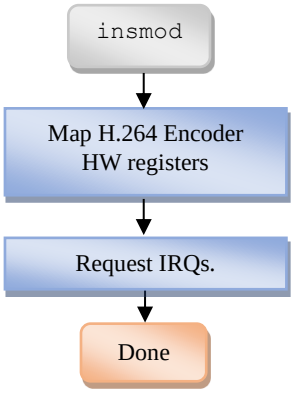

Figure 2: H.264 Encoder kernel module initialization

3.2.3Usage

The H.264 Encoder interface parameters can be configured using the H264ENC_Config() function, where the parameters are passed in a `H264ENC_Config_t` structure pointer. `BuffersSet()` sets SRAM buffers addresses and sizes.

Data needs to be transferred into the SRAM memory. Once (at least) macroblock-row (16 lines of luma and 8 lines of each of the chroma components) are in the SRAM memory the host invokes the function `H264ENC_LinesFetch ()` to start encoding. Alternatively, the sequencer can assert a side band signal to trigger the wrapper and the encoder.

Encoder stops to write encoded data into SRAM memory when no more data to fetch are available.

3.3 User Space

The H.264 Encoder driver SW includes a thin user space library to abstract kernel space driver from user space. The available data types have been mentioned in section 3.1 and functions exported by the user space library are listed in Table 4.

3.3.1API

Besides the user-space counter parts of `Config()`, `BuffersSet()`, `LinesFetch()`, `BitstreamAlarmShift()`,`ResetVars()`, `StatusGet()`, `RowDoneStatusGet()`, `Stop()` functions, the H.264 Encoder user library exports functionality as summarized in the Table 4.

Function	Description
`H264Enc_Reserve`	Opens H.264 Encoder special device file on Linux.
`H264Enc_Release`	Closes H.264 Encoder special device file on Linux.
`H264ENC_FrameIrqDone`	Checks that whole frame has been encoded.
`H264ENC_SwReset`	Resets Encoder core and cleans all registers used in application.
`H264ENC_NewRowIrq`	Returns row done status
`H264ENC_LastFrameWait`	Waits till the last frame all whole sequence is encoded.
`H264ENC_EventHandlerSet`	Internal signal handler setup.

Table4: H.264 Encoder user library exported functions

3.4 H.264 Encoder user level workflow

From the user application point of view there are two ways to use the H.264 Encoder functionality:

- through the SDI library. This is a preferred method.
- directly through the H.264 Encoder driver user-space library.

To enable the use of the H.264 Encoder driver services on Linux the H.264 Encoder kernel module has to be loaded (`insmod` command) and a special device file for the driver has to be created (`mknod` command).

When the `insmod` command has succeeded the user applications can begin to use the H.264 Encoder HW accelerated data preprocessing.

To start using the H.264 Encoder driver an application has to call `H264Enc_Reserve()` function. This function opens the driver special device file and stores its descriptor for future use. `EncodingConfigure()` function configures Encoder HW parameters as data mode, color

format, frame width/height, quantization parameters and more. `H264ENC_BuffersSet()` sets SRAM buffers addresses and sizes.

Once (at least) macroblock-row (16 lines of luma and 8 lines of each of the chroma components) are in the SRAM memory the host calls function `Lines_fetch()` to start. Alternatively, the sequencer can assert a side band signal to trigger the wrapper and the encoder core.

Encoding status can be checked by `StatusGet()` function providing addresses of last frame end, address where encoder wrote last data in SRAM. Those information can be used to copy encoded data from SRAM circular buffer to DDR.

`H264Enc_Release()` function is called to close H264 Encoder device file.

4 High Level Design

4.1 System Decomposition

The H.264 Encoder 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 H.264 Encoder functionality in a user application is to use H.264 Encoder isp graphs together with the SDI library services. The SDI library provides complete abstraction of the H.264 Encoder driver interface and thanks to utilization of the H.264 Encoder HW the data flow management load for the host CPU is minimized.

4.2 File Structure

H264 Encoder driver code is located in VSDK under s3234_sdk/libs/arm/isp/h264enc folder. Internally it has the following structure:

- kernel
  - build-v234ce-gnu-linux-d – build folder for Linux kernel module
    - Makefile
  - include
    - h264enc_func.h – declaration of H.264 Encoder driver functionality
    - h264enc _lldcmd.h – declaration of LLDCMD macros and defines
    - h264enc _types.h – declaration of H.264 Encoder related data types
    - h264enc.h – general H.264 Encoder related declarations/definitions
  - src
    - h264enc _core.c – Linux module related functionality
    - h264enc _func.c – definition of the H.264 Encoder driver functionality
    - h264enc _lldcmd.c – definition of LLDCMD handling
- user
  - build-* – build folders for supported platforms (Linux)
    - Makefile
  - src
    - h264enc _user.cpp – definition of user space level public API,
  - BUILD.mk – defines build details
- Public include
  - isp_h264enc.h – declaration of user space level public API

4.3 Module Usage

4.3.1H264_encoder_cv demo example

After setting input frame parameters through `H264Encoder_setup()` and initialization of SRAM and DDR buffers in `Initialize()` method, each input frame channel (YUV) has its own buffer allocated corresponding to buffers configuration inside the H264Encoder class constructor.

`EncoderFrame()` is called to start encoding. Conversion from RGB to supported YUV 4:2:0 data type is realized inside the method as well as loading macroblock row into the input SRAM circular buffers(YUV) from where are data being encoded. After each macroblock encoding copying from SRAM output buffer to DDR is realized and then data are saved into a file.

`Close()` method resets internal variables and frees used memory.