AVR32413: AVR32 AP7 Image sensor interface driver

32-bit AVR® Microcontrollers

Application Note

Features

- **V4l2 introduction**
- **Image sensor interface driver**
- **Camera driver**

1 Introduction

The image sensor interface (ISI) is available on the Atmel AVR®32AP700x microcontroller series and can be used to connect various image sensors. This application note describes how the ISI module can be used with Linux®.

Rev. 32084B-AVR32-08/08

2 V4L2 introduction

The video for Linux 2 specification (V4L2) is the predecessor of the old video for Linux specification (V4L). It specifies a standard API for a variety of devices but only some of them are can really be labeled as “video devices”. The specification is available from http://www.v4l2spec.bytesex.org. This specification describes the use of the V4L2 API from the user-space. The documentation of V4L2 API in the kernel is unfortunately not that good documented. Never the less an article series on http://www.lwn.net/Articles/203924/ gives a short overview of all necessary parts and in addition a driver skeleton (drivers/media/video/vivi.c) is available in the kernel sources. This is a virtual driver that generates test patterns and does not need to interface any hardware. This source code is also usable as template for an own V4L2 driver implementation.

V4L2 provides, among other things, a video capture interface which provides the ability to grab video data from a tuner or a camera device.

The V4L2 API defines two different ways of transferring video data between user-space and a driver. These are memory mapping and simple data copying from kernel to user-space. Capturing uses the read() (data copy) interface of a driver to copy the grabbed data to user space. This is slow due to the copy transaction and is therefore only suited for a “single capture”. If a video stream should be acquired the memory mapped interface of the V4L2 specification should be used to get the best performance. The mmap() interface is used by a V4L2 driver to map either buffers provided by the driver to user-space or buffers provided by a user-space application to the driver.

3 Image sensor interface capabilities

3.1 Image processing paths

Whereas the ISI can process various input formats the output formats are limited to three, depending on the configuration and the processing path. Two processing paths are available and their capabilities are as described in following chapters.

3.1.1 Codec path

The codec path transforms the sensor data to a YCbCr 4:2:2 format. The exact ordering in memory is described in Table 3-1. Each piece of pixel information Y, Cb and Cr consists of a byte. The codec data path is mainly used to capture a single frame and encode it afterwards in a user-space application. To get a frame from the codec path use the capture interface of the driver.

This path has no support for linked list DMA operations and is therefore not optimal for video streaming. Anyway streaming video is still possible if enough bandwidth is available on the system and the DMA address of the frame buffer is switched in time.

Table 3-1 Codec path pixel data ordering

MSB			**LSB**
$Y_{(i+1)}$	$Cr_{(i)}$	$Y_{(i)}$	$Cb_{(i)}$

3.1.2 Preview path

The preview path has the ability to store the incoming frames in a linked list of buffers. This is also described in detail in the datasheet. Because of this buffer handling, this processing path is preferred if video streaming is needed. The main purpose of this path is to feed the LCD controller with incoming video frames and the video data is therefore converted to RGB 5:5:5. The maximum output resolution on the preview path is VGA (640x480). Larger input images can be scaled down by the module. The data ordering of the preview path in memory is as described in Table 3-2:

Table 3-2 Preview path pixel data representation

MSB															**LSB**
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	R4	R3	R2	R1	R0	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

The RBG output format of the preview path is not optimal for creating a video stream or encoding and therefore the on-chip pixel co-processor (PICO) can be used to do a conversion. This may be implemented in a user-space application.

In grayscale mode only the preview path is active and captures the image data without any processing in memory. Table 3-3 shows the data ordering in memory in grayscale mode. The unused bits are filled with 0.

Table 3-3 Grayscale mode output with 12-bit data input

GS mode	**DATA[31-24]**	**DATA[32-16]**	**DATA[15-8]**	**DATA[7-0]**
0	Data$_{(ii)}$[11:4]	Data$_{(ii)}$[3:0],0x0	Data$_{(i+1)}$[11:4]	Data$_{(i+1)}$[3:0],0x0
1	Data$_{(ii)}$[11:4]	Data$_{(ii)}$[3:0],0x0	0x00	0x00

3.2 Input formats

3.2.1 RGB input

All possible RGB input combinations are listed in the device datasheet. These are the RGB 8:8:8 and the RGB 5:6:5 formats in various ordering variations. The output of the different processing paths is as described in the chapters before.

3.2.2 YCbCr input

Possible YCbCr inputs are listed in Table 3-4. The grey rows show input and ISI configurations that can be used with codec and preview path at the same time. All other combinations lead to a color corruption in the preview path.

If only the codec path is used it can be beneficial to use one of the unusual configurations in order to get a specific V4L2 format without doing a software swap.

Table 3-4 YCbCr codec path output order

ISI input	**ISI configuration**	**ISI codec path output**	**V4L format**
$Cb_{(i)}Y_{(i)}Cr_{(i)}Y_{(i+1)}$	UYVY (default)	$Y_{(i+1)}Cr_{(i)}Y_{(i)}Cb_{(i)}$	
$Cb_{(i)}Y_{(i)}Cr_{(i)}Y_{(i+1)}$	VYUY (mode1)	$Y_{(i+1)}Cb_{(i)}Y_{(i)}Cr_{(i)}$	
$Cb_{(i)}Y_{(i)}Cr_{(i)}Y_{(i+1)}$	YUYV (mode2)	$Cr_{(i)}Y_{(i+1)}Cb_{(i)}Y_{(i)}$	
$Cb_{(i)}Y_{(i)}Cr_{(i)}Y_{(i+1)}$	YVYU (mode3)	$Cr_{(i)}Y_{(i)}Cb_{(i)}Y_{(i+1)}$	V4L2_PIX_FMT_VYUY

ISI input	ISI configuration	ISI codec path output	V4L format
$Cr_{(i)}Y_{(i)}Cb_{(i)}Y_{(i+1)}$	UYVY (default)	$Y_{(i+1)}Cb_{(i)}Y_{(i)}Cr_{(i)}$	
$Cr_{(i)}Y_{(i)}Cb_{(i)}Y_{(i+1)}$	VYUY (mode1)	$Y_{(i+1)}Cr_{(i)}Y_{(i)}Cb_{(i)}$	
$Cr_{(i)}Y_{(i)}Cb_{(i)}Y_{(i+1)}$	YUYV (mode2)	$Cb_{(i)}Y_{(i+1)}Cr_{(i)}Y_{(i)}$	
$Cr_{(i)}Y_{(i)}Cb_{(i)}Y_{(i+1)}$	YVYU (mode3)	$Cb_{(i)}Y_{(i)}Cr_{(i)}Y_{(i+1)}$	V4L2_PIX_FMT_UYVY
$Y_{(i)}Cb_{(i)}Y_{(i+1)}Cr_{(i)}$	UYVY (default)	$Cr_{(i)}Y_{(i+1)}Cb_{(i)}Y_{(i)}$	
$Y_{(i)}Cb_{(i)}Y_{(i+1)}Cr_{(i)}$	VYUY (mode1)	$Cb_{(i)}Y_{(i+1)}Cr_{(i)}Y_{(i)}$	
$Y_{(i)}Cb_{(i)}Y_{(i+1)}Cr_{(i)}$	YUYV (mode2)	$Y_{(i+1)}Cr_{(i)}Y_{(i)}Cb_{(i)}$	
$Y_{(i)}Cb_{(i)}Y_{(i+1)}Cr_{(i)}$	YVYU (mode3)	$Y_{(i+1)}Cb_{(i)}Y_{(i)}Cr_{(i)}$	
$Y_{(i)}Cr_{(i)}Y_{(i+1)}Cb_{(i)}$	UYVY (default)	$Cb_{(i)}Y_{(i+1)}Cr_{(i)}Y_{(i)}$	
$Y_{(i)}Cr_{(i)}Y_{(i+1)}Cb_{(i)}$	VYUY (mode1)	$Cr_{(i)}Y_{(i+1)}Cb_{(i)}Y_{(i)}$	
$Y_{(i)}Cr_{(i)}Y_{(i+1)}Cb_{(i)}$	YUYV (mode2)	$Y_{(i+1)}Cb_{(i)}Y_{(i)}Cr_{(i)}$	
$Y_{(i)}Cr_{(i)}Y_{(i+1)}Cb_{(i)}$	YVYU (mode3)	$Y_{(i+1)}Cr_{(i)}Y_{(i)}Cb_{(i)}$	

3.2.3 Grayscale input

By selecting this mode the codec path is disabled and the data is stored directly without any processing in the internal buffers.

Since the ISI module uses 12-bit in grayscale mode a sensor with 8-bit or lower data width should be connected to the ISI module by starting from line 4. This has the result that the data is aligned to a byte boundary. Table … shows how 8-bit data is ordered in memory if the sensor is connected to the ISI module like discussed before. Since the unused pins can be pulled low or high the state depends on the hardware setup. This is represented by a “X” in the table.

Table 3-5 Grayscale mode output with 8-bit data input

GS mode	DATA[31-24]	DATA[32-16]	DATA[15-8]	DATA[7-0]
0	$Data_{(i)}$[7:0]	0xX0	$Data_{(i+1)}$[7:0]	0xX0
1	$Data_{(i)}$[7:0]	0xX0	0x00	0x00

4 ISI Linux driver

The ISI Linux driver is available from avr32linux.org (if not included in the mainline kernel) and the patch series, when applied, will add the platform device to the system. In order to set up the correct values for the actually used input device the board code has to be edited or a camera needs to be hooked to the ISI driver that provides the needed data. Hooking a camera driver to the ISI module is optional and is discussed in chapter 5.

The ISI image sensor driver provides two interfaces to grab video data. These interfaces represent the two different capture paths in hardware. The read() interface can be used to capture a single frame and the streaming interface to grab a video stream.

4.1 Board setup code

The ISI driver needs a valid platform device with the configuration information. In order to add the platform device to the system the board setup code needs to be edited.

In the board setup code add the includes:

```
#include <media/atmel-isi.h>
#include <linux/videodev2.h>
```

Then fill out the platform data structure.

```
static struct isi_platform_data __initdata isi_data = {
        .image_hsize    = 320,
        .image_vsize    = 240,
        .prev_hsize     = 320,
        .prev_vsize     = 240,
        .pixfmt         = ATMEL_ISI_PIXFMT_CbYCrY,
        .capture_v4l2_fmt = V4L2_PIX_FMT_YUYV,
        .streaming_v4l2_fmt = V4L2_PIX_FMT_RGB555X,
        .cr1_flags              = ISI_FULL | ISI_EMB_SYNC,
};
```

Available configuration options are listed in the above mentioned include files and some of them are described in Table 4-1.

Table 4-1 ISI module parameters

Parameter name	Parameter description	Example values
image_hsize	Horizontal size of the sensor input image.	320
image_vsize	Vertical size of the sensor input image.	320
video_buffers	Video buffers used.	4
prev_hsize	Image horizontal size of the preview path output.	320
prev_vsize	Image vertical size of the preview path output.	240
pix_fmt	Input format of the sensor. This is related to the data ordering tables in the datasheet.	ATMEL_ISI_PIXFMT_CbYCrY
cr1_flags	Signal polarity and other configurations from ISI register CR1.	ISI_EMB_SYNC | ISI_FULL
capture_v4l2_fmt	Use this parameter to pretend a 4VL2 output format for the capture interface. Take a look at the header file include/linux/videodev2.h for valid values. Set this value to 0 if the default format V4L2_PIX_FMT_YUYV should be used.	0

Parameter name	Parameter description	Example values
streaming_v4l2_fmt	Use this parameter to pretend a V4L2 format for the streaming interface. For now this is an integer value. Take a look at the header file include/linux/videodev2.h for valid values. Set this value to 0 if the default format V4L2_PIX_FMT_RGB555X should be used.	0

The last step is to add the platform device by calling the function:

```
at32_add_device_isi(0, &isi_data);
```

More information about board setup code and how to edit it is described in the application note "AVR32744 AVR32 Linux Custom Board Support" available from the Atmel website.

4.2 Accessing the ISI driver form user-space

The ISI driver registers two V4L2 devices. They can be found in the /dev directory as video0 and video1 device files. The numbering may be different if other video devices are available on the system. To obtain the correct device go to the directory /sys/class/video4linux/. Here are two directories, video0 and video1. Move into the directory of interest and read out the data of the "name" file there (do a "cat name" on the shell for instance). Two names are possible for the ISI V4L2 devices:

- **atmel_isi_capture**: This is the ISI V4L2 capture device. It uses the codec path of the ISI module to capture single frames.
- **atmel_isi_streaming**: This is the ISI V4L2 streaming device. It uses the preview path of the ISI module. It is also possible to configure this interface to use the codec path instead but this is still experimental.

When you have decided which of these interfaces you want to use open the corresponding file in the /dev directory. An example application that captures a single frame is available on http://www.avr32linux.org. Further examples which include streaming are available from http://www.thedirks.org/v4l2.

5 Camera driver

The ISI driver functionality was split into two parts in order to separate camera related things from the basic capturing/streaming. It is optionally to hook a camera driver to the ISI driver because some devices that provide the data do not have any use for an interface to the ISI driver. This approach is necessary to be able to connect different cameras without to rewrite the V4L2 driver. The interface between camera driver and V4L2 ISI driver is able to provide various features such as video format negotiation or starting and stopping of a capture. A user-space application can therefore use the V4L2 API to interact with the camera and gain control of basic settings. All camera specific functions can be implemented in a camera driver, such as talking to the device over I2C, power up sequence and so forth.

The more advanced and camera specific configurations must be done in the camera driver or from a user-space application by using a camera driver interface. This is implementation specific and therefore just examples on how this can be done will be mentioned here. A good starting point is the drivers for a Micron (MT9M112) and

Atmel (AT76C451BC-MY15AT) camera. The patches are available on http://www.avr32linux.org.

The ISI module API is defined in drivers/media/video/atmel_isi.h and provides following two functions for a camera driver.

```
int atmel_isi_register_camera(struct atmel_isi_camera *cam);
```

```
void atmel_isi_unregister_camera(struct atmel_isi_camera *cam);
```

The function atmel_isi_register_camera function can be called from a camera driver to register itself. The atmel_isi_unregister_camera function removes the driver from the ISI internal camera list. The parameter of both functions is a structure that needs to be filled out from the camera driver. This structure looks as follows:

```
struct atmel_isi_camera {
        const char              *name;
        struct module            *owner;
        struct list_head        list;
        unsigned int            hsync_act_low:1;
        unsigned int            vsync_act_low:1;
        unsigned int            pclk_act_falling:1;
        unsigned int            has_emb_sync:1;
        /* ISI supports up to 17 formats */
        unsigned int            pixelformats[17];
        int (*get_format)(struct atmel_isi_camera *cam, struct atmel_isi_format *fmt);
        int (*set_format)(struct atmel_isi_camera *cam, struct atmel_isi_format *fmt);
        int (*start_capture)(struct atmel_isi_camera *cam);
        int (*stop_capture)(struct atmel_isi_camera *cam);
        struct atmel_isi        *isi;
};
```

Fields that need to be filled out by the camera driver are.

- name: Name of the camera driver.
- hsync_act_low: Set to 1 if horizontal synchronization signal is active low. If active high set it to 0.
- vsync_act_low: Set to 1 if vertical synchronization is active low. If not, set it to 0.
- pclk_act_falling: If the pixel data is valid at the falling edge of the pixel clock set it to 1. Otherwise to 0.
- has_emb_sync: If the embedded synchronization mechanism is used, set it this value to 1. If horizontal and vertical synchronization signals are used instead set this value to 0.

In addition four functions must be implemented in the camera driver and the function pointers in the above structure must be initialized to point to these functions. These are functions that start/stop capturing (start_capture/stop_capture) and set/get the camera video format (get_format/set_format). More functions may be implemented later if more functionality of the V4L2 specification needs to be passed to the camera driver.

The get_format function updates the fmt structure with the current camera settings. This is all that needs to be done in this function. The set_format function must check the provided fmt structure to decide whether it supports the requested settings or not. If the format is valid it should configure the camera respectively. If the values are not valid the camera driver should set these to the closest possible match.

6 Using the ISI V4L2 driver with other software

Because the ISI driver is using the 4VL2 framework it is compatible with many video players, streamer, grabber or encoder. This makes it possible to process the data from the ISI in various open source applications. VLC (http://www.videolan.org) and MPlayer are good candidates for this purpose. MPlayer is already available in the AVR32 Buildroot build system.

Since the ISI output formats of both processing paths are not standard V4L2 formats usually a swap of the pixel data has to be done before using these applications. The on chip pixel co-processor is good for this task. Another possible way is to use “Table 3-4 **YCbCr codec path output order**” to choose a swapping that provides a valid V4L2 format. Be aware that choosing a “non standard” will work only for the codec path, the preview path will provide garbage data in this configuration since the YUV to RGB conversion expects other input data.

6.1 Capture example application

On avr32linux.org an example program is available that shows how the capture interface of the V4L2 images sensor interface can be used.

Following command line captures an image and saves it as “image.ppm” (make sure that the capture interface is /dev/video0 because it may vary if you have other video drivers installed):

```
capture –d/dev/video0 –o image.ppm
```

More information about the command line arguments is available on http://avr32linux.org/twiki/bin/view/Main/AtmelIsiDriver.

6.2 Webcam example application

A simple command line tool that provides web-camera functionality is “webcam” from the xawtv toolset (http://linux.bytesex.org/xawtv/).

The tool needs a valid configuration file to work properly. Following configuration captures each second an image from /dev/video0 and places it into the folder /tmp.

```
[grab]
device = /dev/video0
width = 320
height = 240
delay = 1
quality = 75
trigger = 0
rotate = 0

[ftp]
host = localhost
user = nobody
pass = xxxxxx
dir = /tmp
file = webcam.png
tmp = imageup.png
```

```
local = 1
```

It is important to select a place that resides in RAM and additionally it should not be mirrored/synced to flash. This ensures that the flash is not worn out by many updates of the image. More information about the configuration is available on the website. To start the application run it with the configuration file as argument. For example like:

```
webcam /etc/webcam.conf
```

To view the image on the web a valid html page is needed that provides a browser with the link to the image. This file could look like this:

```
<head>
<meta http-equiv="refresh" content="1"; URL="http://example.net/">
</head>

<body>
<p><img src="/tmp/webcam.png" alt="test image"></p>
</body>
```

The html file must be stored in a directory that the webserver is aware of and is usually named index.html.

Some other useful tools are also available within the package like streamer and v4lctl.

6.3 Using the ISI driver with MEncoder

MEncoder is an encoding application that is included in the MPlayer project. (http://www.mplayerhq.hu). It can encode a video stream to various formats (raw output, MPEG-2, DivX …) and encapsulates them in different container formats such as AVI, ASF or MP4. MEncoder is able to use a V4L2 device, such as the ISI Linux driver, as input.

To capture a video over the codec path following command line for Mencoder is possible:

```
mencoder –tv \
driver=v4l2:device=/dev/video1:width=320:height=240:noaudio tv:// \
-fps 15 -ofps 15 -ovc raw -o video.avi
```

This command will capture QVGA frames from the ISI and put them in an AVI container format. The output stream has 15 frames per second (-ofps). The input depends on the sensor output but here an estimation is made of 15 frames per second (-fps). More information is available in the MEncoder, or rather MPlayer, documentation on the web.

To get all available output formats run

mencoder –tv outfmt=help

To list all codec libraries use:

mencoder –ovc help

Typical output switches are:

- raw: Output uncompressed video
- lavc: Use the libavcodecs library

All video container formats can be listed by:

mencoder –of help

Typical container formats are AVI and MPEG.

7 References

FOURCC codes: http://www.fourcc.org

VLC player: http://www.videolan.org

V4L2 wiki: http://www.linuxtv.org

V4L2 specification: http://www.v4l2spec.bytesex.org

V4L2 article http://www.lwn.net/Articles/203924/

Atmel camera patches. http://www.avr32linux.org

Webcam and other V4L tools: http://linux.bytesex.org/xawtv/

AVR32744 AVR32 Linux Custom Board Support: http://www.atmel.com

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
Avr32@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2008 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, AVR® and others, are the registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.