```
                        SN9C10x PC Camera Controllers
                               Driver for Linux
                        =============================

                              - Documentation -
```

Index
=====
1. Copyright
2. Disclaimer
3. License
4. Overview and features
5. Module dependencies
6. Module loading
7. Module parameters
8. Optional device control through "sysfs"
9. Supported devices
10. How to add plug-in's for new image sensors
11. Notes for V4L2 application developers
12. Video frame formats
13. Contact information
14. Credits

1. Copyright
============
Copyright (C) 2004-2005 by Luca Risolia <luca.risolia@studio.unibo.it>

2. Disclaimer
=============
SONiX is a trademark of SONiX Technology Company Limited, inc.
This software is not sponsored or developed by SONiX.

3. License
==========
This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

4. Overview and features
========================
This driver attempts to support the video and audio streaming capabilities of
the devices mounting the SONiX SN9C101, SN9C102 and SN9C103 PC Camera
Controllers.

It's worth to note that SONiX has never collaborated with the author during the
development of this project, despite several requests for enough detailed
specifications of the register tables, compression engine and video data format
of the above chips. Nevertheless, these informations are no longer necessary,
becouse all the aspects related to these chips are known and have been
described in detail in this documentation.

The driver relies on the Video4Linux2 and USB core modules. It has been
designed to run properly on SMP systems as well.

The latest version of the SN9C10x driver can be found at the following URL:
http://www.linux-projects.org/

Some of the features of the driver are:

- full compliance with the Video4Linux2 API (see also "Notes for V4L2
  application developers" paragraph);
- available mmap or read/poll methods for video streaming through isochronous
  data transfers;
- automatic detection of image sensor;
- support for any window resolutions and optional panning within the maximum
  pixel area of image sensor;
- image downscaling with arbitrary scaling factors from 1, 2 and 4 in both
  directions (see "Notes for V4L2 application developers" paragraph);
- two different video formats for uncompressed or compressed data in low or
  high compression quality (see also "Notes for V4L2 application developers"
  and "Video frame formats" paragraphs);
- full support for the capabilities of many of the possible image sensors that
  can be connected to the SN9C10x bridges, including, for istance, red, green,
  blue and global gain adjustments and exposure (see "Supported devices"
  paragraph for details);
- use of default color settings for sunlight conditions;
- dynamic I/O interface for both SN9C10x and image sensor control and
  monitoring (see "Optional device control through 'sysfs'" paragraph);
- dynamic driver control thanks to various module parameters (see "Module
  parameters" paragraph);
- up to 64 cameras can be handled at the same time; they can be connected and
  disconnected from the host many times without turning off the computer, if
  your system supports hotplugging;
- no known bugs.

5. Module dependencies
======================
For it to work properly, the driver needs kernel support for Video4Linux and
USB.

The following options of the kernel configuration file must be enabled and
corresponding modules must be compiled:

    # Multimedia devices
    #
    CONFIG_VIDEO_DEV=m

    # USB support
    #
    CONFIG_USB=m

In addition, depending on the hardware being used, the modules below are
necessary:

    # USB Host Controller Drivers
    #
    CONFIG_USB_EHCI_HCD=m
    CONFIG_USB_UHCI_HCD=m
    CONFIG_USB_OHCI_HCD=m

And finally:

    # USB Multimedia devices
    #
    CONFIG_USB_SN9C102=m

6. Module loading
=================
To use the driver, it is necessary to load the "sn9c102" module into memory
after every other module required: "videodev", "usbcore" and, depending on
the USB host controller you have, "ehci-hcd", "uhci-hcd" or "ohci-hcd".

Loading can be done as shown below:

    [root@localhost home]# modprobe sn9c102

At this point the devices should be recognized. You can invoke "dmesg" to
analyze kernel messages and verify that the loading process has gone well:

```
[user@localhost home]$ dmesg
```


7. Module parameters
====================
Module parameters are listed below:

Name: video_nr
Type: int array (min = 0, max = 64)
Syntax: <-1|n[,...]>
Description: Specify V4L2 minor mode number:
                -1 = use next available
                 n = use minor number n
                You can specify up to 64 cameras this way.
                For example:
                video_nr=-1,2,-1 would assign minor number 2 to the second
                recognized camera and use auto for the first one and for every
                other camera.
Default: -1

Name: force_munmap;
Type: bool array (min = 0, max = 64)
Syntax: <0|1[,...]>
Description: Force the application to unmap previously mapped buffer memory
                before calling any VIDIOC_S_CROP or VIDIOC_S_FMT ioctl's. Not
                all the applications support this feature. This parameter is
                specific for each detected camera.
                0 = do not force memory unmapping"
                1 = force memory unmapping (save memory)"
Default: 0

Name: debug
Type: int
Syntax: <n>
Description: Debugging information level, from 0 to 3:
                0 = none (use carefully)
                1 = critical errors
                2 = significant informations
                3 = more verbose messages
                Level 3 is useful for testing only, when only one device
                is used. It also shows some more informations about the
                hardware being detected. This parameter can be changed at
                runtime thanks to the /sys filesystem.
Default: 2
----

8. Optional device control through "sysfs" [1]
==
It is possible to read and write both the SN9C10x and the image sensor
registers by using the "sysfs" filesystem interface.

Every time a supported device is recognized, a write-only file named "green" is
created in the /sys/class/video4linux/videoX directory. You can set the green
channel's gain by writing the desired value to it. The value may range from 0
to 15 for SN9C101 or SN9C102 bridges, from 0 to 127 for SN9C103 bridges.
Similarly, only for SN9C103 controllers, blue and red gain control files are
available in the same directory, for which accepted values may range from 0 to
127.

There are other four entries in the directory above for each registered camera:
"reg", "val", "i2c_reg" and "i2c_val". The first two files control the
SN9C10x bridge, while the other two control the sensor chip. "reg" and
"i2c_reg" hold the values of the current register index where the following
reading/writing operations are addressed at through "val" and "i2c_val". Their
use is not intended for end-users. Note that "i2c_reg" and "i2c_val" will not
be created if the sensor does not actually support the standard I2C protocol or
its registers are not 8-bit long. Also, remember that you must be logged in as
root before writing to them.

As an example, suppose we were to want to read the value contained in the
register number 1 of the sensor register table - which is usually the product
identifier - of the camera registered as "/dev/video0":

```
[root@localhost #] cd /sys/class/video4linux/video0
[root@localhost #] echo 1 > i2c_reg
[root@localhost #] cat i2c_val
```

Note that "cat" will fail if sensor registers cannot be read.

Now let's set the green gain's register of the SN9C101 or SN9C102 chips to 2:

```
[root@localhost #] echo 0x11 > reg
[root@localhost #] echo 2 > val
```

Note that the SN9C10x always returns 0 when some of its registers are read.
To avoid race conditions, all the I/O accesses to the above files are
serialized.

The sysfs interface also provides the "frame_header" entry, which exports the
frame header of the most recent requested and captured video frame. The header
is 12-bytes long and is appended to every video frame by the SN9C10x
controllers. As an example, this additional information can be used by the user
application for implementing auto-exposure features via software.

The following table describes the frame header:

Byte #	Value	Description
0x00	0xFF	Frame synchronisation pattern.
0x01	0xFF	Frame synchronisation pattern.
0x02	0x00	Frame synchronisation pattern.
0x03	0xC4	Frame synchronisation pattern.
0x04	0xC4	Frame synchronisation pattern.
0x05	0x96	Frame synchronisation pattern.
0x06	0x00 or 0x01	Unknown meaning. The exact value depends on the chip.
0x07	0xXX	Variable value, whose bits are ff00uzzc, where ff is a frame counter, u is unknown, zz is a size indicator (00 = VGA, 01 = SIF, 10 = QSIF) and c stands for "compression enabled" (1 = yes, 0 = no).
0x08	0xXX	Brightness sum inside Auto-Exposure area (low-byte).
0x09	0xXX	Brightness sum inside Auto-Exposure area (high-byte). For a pure white image, this number will be equal to 500 times the area of the specified AE area. For images that are not pure white, the value scales down according to relative whiteness.
0x0A	0xXX	Brightness sum outside Auto-Exposure area (low-byte).
0x0B	0xXX	Brightness sum outside Auto-Exposure area (high-byte). For a pure white image, this number will be equal to 125 times the area outside of the specified AE area. For images that are not pure white, the value scales down according to relative whiteness.

The AE area (sx, sy, ex, ey) in the active window can be set by programming the
registers 0x1c, 0x1d, 0x1e and 0x1f of the SN9C10x controllers, where one unit
corresponds to 32 pixels.

[1] The frame header has been documented by Bertrik Sikken.

9. Supported devices
====================
None of the names of the companies as well as their products will be mentioned
here. They have never collaborated with the author, so no advertising.

From the point of view of a driver, what unambiguously identify a device are
its vendor and product USB identifiers. Below is a list of known identifiers of
devices mounting the SN9C10x PC camera controllers:

```
Vendor ID  Product ID
---------  ----------
```

```
0x0c45     0x6001
0x0c45     0x6005
0x0c45     0x6009
0x0c45     0x600d
0x0c45     0x6024
0x0c45     0x6025
0x0c45     0x6028
0x0c45     0x6029
0x0c45     0x602a
0x0c45     0x602b
0x0c45     0x602c
0x0c45     0x602d
0x0c45     0x6030
0x0c45     0x6080
0x0c45     0x6082
0x0c45     0x6083
0x0c45     0x6088
0x0c45     0x608a
0x0c45     0x608b
0x0c45     0x608c
0x0c45     0x608e
0x0c45     0x608f
0x0c45     0x60a0
0x0c45     0x60a2
0x0c45     0x60a3
0x0c45     0x60a8
0x0c45     0x60aa
0x0c45     0x60ab
0x0c45     0x60ac
0x0c45     0x60ae
0x0c45     0x60af
0x0c45     0x60b0
0x0c45     0x60b2
0x0c45     0x60b3
0x0c45     0x60b8
0x0c45     0x60ba
0x0c45     0x60bb
0x0c45     0x60bc
0x0c45     0x60be
```

The list above does not imply that all those devices work with this driver: up
until now only the ones that mount the following image sensors are supported;
kernel messages will always tell you whether this is the case:

```
Model        Manufacturer
-----        ------------
HV7131D      Hynix Semiconductor, Inc.
MI-0343      Micron Technology, Inc.
OV7630       OmniVision Technologies, Inc.
PAS106B      PixArt Imaging, Inc.
PAS202BCB    PixArt Imaging, Inc.
TAS5110C1B   Taiwan Advanced Sensor Corporation
TAS5130D1B   Taiwan Advanced Sensor Corporation
```

All the available control settings of each image sensor are supported through
the V4L2 interface.

Donations of new models for further testing and support would be much
appreciated. Non-available hardware will not be supported by the author of this
driver.

10. How to add plug-in's for new image sensors
==
It should be easy to write plug-in's for new sensors by using the small API
that has been created for this purpose, which is present in "sn9c102_sensor.h"
(documentation is included there). As an example, have a look at the code in
"sn9c102_pas106b.c", which uses the mentioned interface.

At the moment, possible unsupported image sensors are: CIS-VF10 (VGA),
OV7620 (VGA), OV7630 (VGA).

11. Notes for V4L2 application developers
==
This driver follows the V4L2 API specifications. In particular, it enforces two
rules:

- exactly one I/O method, either "mmap" or "read", is associated with each
file descriptor. Once it is selected, the application must close and reopen the
device to switch to the other I/O method;

- although it is not mandatory, previously mapped buffer memory should always
be unmapped before calling any "VIDIOC_S_CROP" or "VIDIOC_S_FMT" ioctl's.
The same number of buffers as before will be allocated again to match the size
of the new video frames, so you have to map the buffers again before any I/O
attempts on them.

Consistently with the hardware limits, this driver also supports image
downscaling with arbitrary scaling factors from 1, 2 and 4 in both directions.
However, the V4L2 API specifications don't correctly define how the scaling
factor can be chosen arbitrarily by the "negotiation" of the "source" and
"target" rectangles. To work around this flaw, we have added the convention
that, during the negotiation, whenever the "VIDIOC_S_CROP" ioctl is issued, the
scaling factor is restored to 1.

This driver supports two different video formats: the first one is the "8-bit
Sequential Bayer" format and can be used to obtain uncompressed video data
from the device through the current I/O method, while the second one provides
"raw" compressed video data (without frame headers not related to the
compressed data). The compression quality may vary from 0 to 1 and can be
selected or queried thanks to the VIDIOC_S_JPEGCOMP and VIDIOC_G_JPEGCOMP V4L2
ioctl's. For maximum flexibility, both the default active video format and the
default compression quality depend on how the image sensor being used is
initialized (as described in the documentation of the API for the image sensors
supplied by this driver).

12. Video frame formats [1]
========================
The SN9C10x PC Camera Controllers can send images in two possible video
formats over the USB: either native "Sequential RGB Bayer" or Huffman
compressed. The latter is used to achieve high frame rates. The current video
format may be selected or queried from the user application by calling the
VIDIOC_S_FMT or VIDIOC_G_FMT ioctl's, as described in the V4L2 API
specifications.

The name "Sequential Bayer" indicates the organization of the red, green and
blue pixels in one video frame. Each pixel is associated with a 8-bit long
value and is disposed in memory according to the pattern shown below:

```
B[0]   G[1]   B[2]   G[3]   ...   B[m-2]        G[m-1]
G[m]   R[m+1] G[m+2] R[m+2] ...   G[2m-2]       R[2m-1]
...
...                               B[(n-1)(m-2)] G[(n-1)(m-1)]
...                               G[n(m-2)]     R[n(m-1)]
```

The above matrix also represents the sequential or progressive read-out mode of
the (n, m) Bayer color filter array used in many CCD/CMOS image sensors.

One compressed video frame consists of a bitstream that encodes for every R, G,
or B pixel the difference between the value of the pixel itself and some
reference pixel value. Pixels are organised in the Bayer pattern and the Bayer
sub-pixels are tracked individually and alternatingly. For example, in the
first line values for the B and G1 pixels are alternatingly encoded, while in
the second line values for the G2 and R pixels are alternatingly encoded.

The pixel reference value is calculated as follows:
- the 4 top left pixels are encoded in raw uncompressed 8-bit format;
- the value in the top two rows is the value of the pixel left of the current
  pixel;
- the value in the left column is the value of the pixel above the current
  pixel;
- for all other pixels, the reference value is the average of the value of the

  pixel on the left and the value of the pixel above the current pixel;
- there is one code in the bitstream that specifies the value of a pixel
  directly (in 4-bit resolution);
- pixel values need to be clamped inside the range [0..255] for proper
  decoding.

The algorithm purely describes the conversion from compressed Bayer code used
in the SN9C10x chips to uncompressed Bayer. Additional steps are required to
convert this to a color image (i.e. a color interpolation algorithm).

The following Huffman codes have been found:
0: +0 (relative to reference pixel value)
100: +4
101: -4?
1110xxxx: set absolute value to xxxx.0000
1101: +11
1111: -11
11001: +20
110000: -20
110001: ??? - these codes are apparently not used

[1] The Huffman compression algorithm has been reverse-engineered and
    documented by Bertrik Sikken.

13. Contact information
=======================
The author may be contacted by e-mail at <luca.risolia@studio.unibo.it>.

GPG/PGP encrypted e-mail's are accepted. The GPG key ID of the author is
'FCE635A4'; the public 1024-bit key should be available at any keyserver;
the fingerprint is: '88E8 F32F 7244 68BA 3958 5D40 99DA 5D2A FCE6 35A4'.

14. Credits
===========
Many thanks to following persons for their contribute (listed in alphabetical
order):

- Luca Capello for the donation of a webcam;
- Joao Rodrigo Fuzaro, Joao Limirio, Claudio Filho and Caio Begotti for the
  donation of a webcam;
- Jon Hollstrom for the donation of a webcam;
- Carlos Eduardo Medaglia Dyonisio, who added the support for the PAS202BCB
  image sensor;
- Stefano Mozzi, who donated 45 EU;
- Andrew Pearce for the donation of a webcam;
- Bertrik Sikken, who reverse-engineered and documented the Huffman compression
  algorithm used in the SN9C10x controllers and implemented the first decoder;
- Mizuno Takafumi for the donation of a webcam;
- An "anonymous" donator (who didn't want his name to be revealed) for the
  donation of a webcam.