Linux on a Digital Camera

Porting 2.4 Linux kernel to an existing digital camera

Alain Volmat
Ricoh Company Ltd.
avolmat@src.ricoh.co.jp

Shigeki Ouchi
Ricoh Company Ltd
shigeki@src.ricoh.co.jp

Abstract

The RDC-i700 is one of high specs digital camera of Ricoh. Its relatively big size, large amount of different interfaces, input methods (buttons, or touch panel), have made it a good candidate for prototyping the world first Linux embedded digital camera. This paper presents our experiences of porting the 2.4 linux kernel to an existing digital camera. (the RDC-i700 is originally build on top of VxWorks). Eventhough embedded systems running on Linux are getting more and more popular, the digital camera field remains to be unexplored. The paper introduces how digital cameras differ from any other PC-like devices (PDA, HDD recorder...) and what problems, such as timing or software design issues, have to be (have been) solved in order to get the world first linux digital camera running on the linux 2.4 kernel.

1 The hardware

Ricoh's RDC-i700 [1] is a relatively old digital camera (released late 2000 in japan) running on VxWorks, a famous Real-Time OS (RTOS). Some might be asking the reason why we decided to port Linux OS to the camera. The reason is to make it become a *programmable camera*. Once it becomes a programmable device, many VARs or individual programmers may write a lot of useful software for it. Then it will be a good platform for business imaging use.

Figure 1: The RDC-i700 digital camera

The RDC-i700 is one of high specs digital camera of Ricoh. It integrates all peripherals traditional digital camera has, but also several different interfaces, allowing wide range of application to run on it. The VxWorks version allows user to perform various tasks such as taking picture or movie, recording voice memo, browse the Internet, send email or upload picture to a remote server. Its relatively big size, large amount of different interfaces, input methods (buttons, or touch panel), makes it a good candidate for prototyping the world first Linux embedded digital camera.

The RDC-i700 is a 3.2 million pixels digital camera equipped with a Hitachi SH3

[1] http://www.ricohzone.com/product_rdci700.html

(SH7709A) CPU. The SH7709A is a 32 bit RISC CPU which include MMU and several other peripherals such as serial communication interfaces (SCIs), D/A - A/D converters. Around this CPU, traditional digital camera peripherals (CCD, LCD, buttons, Image Processor) but also 1 PCMCIA and 1 CF socket, touch panel, audio input/output interface, USB device controller and a serial port are available. Figure 2 shows a block diagram of the RDC-i700.

2 Digital camera is not "PDA combined with camera function"

Nowadays, embedded Linux has become a very hot topic in the Linux community. More and more Linux gadget are becoming available and the share of embedded related paper published has literally exploded in the last 3 years. Linux seems to be everywhere, lots of devices that were running on RTOS in the past are now running on Linux. However, one field seems to be still unexplored: digital camera. Some might say that a digital camera is just a PDA combined with a CCD (this kind of combination is actually already available, for example the Zaurus CF Digital Camera option), but this is not that simple.

The quality point of view: PDA combined with a digital camera option can take pictures or even movies and in that sense can be compared to a digital camera. But digital cameras still have some advantages that make them irreplaceable. Indeed optical zoom, but also autofocus or strobe are all precious elements that are currently not available on Linux PDA. For example, the Zaurus camera has a focus but this one is manual. Auto-exposure is also another very important part when taking picture; for that, Zaurus PDA has some-kind of gain control but this cannot have same quality as a traditional digital camera auto-exposure system.

The technical point of view: We will see that having digital camera specific peripherals is a very good plus in term of quality, but it also creates lots of problem that traditional Linux PDA doesn't face. Keeping the Zaurus PDA as an example, only few parameters are configurable and the CPU doesn't actually have to perform much work in order to get an image. On the contrary, in case of a fully configurable digital camera, the OS must orchestrate all devices in order to get a picture.

2.1 Zoom and Focus

Several motors are used inside the camera. Two of them are used for zoom and focus in order to adjust the lens position. Due to high precision requirements, those two motors are stepping motors. As the name says, this kind of motors are controlled step by step (at the difference with traditional motors which only have start/stop command). The CPU has to set ports of the motor at a quite fast frequency in order to make the motor turn. In that case the period between two steps is only few milli-seconds.

2.2 Strobe

In case of strobe, the problem is not doing thing at very high speed, but making perfect synchronization between the moment the strobe is going to flash and the moment the CCD sensor will acquire the picture.(see figure 3) For that purpose, we will need precision of only very few milli-seconds.

2.3 Auto-Exposure / White Balance

So called Auto-Exposure is the algorithm in charge of adjusting the exposure time (that is to say the time period while the CCD is exposed to light) in order to have a good image

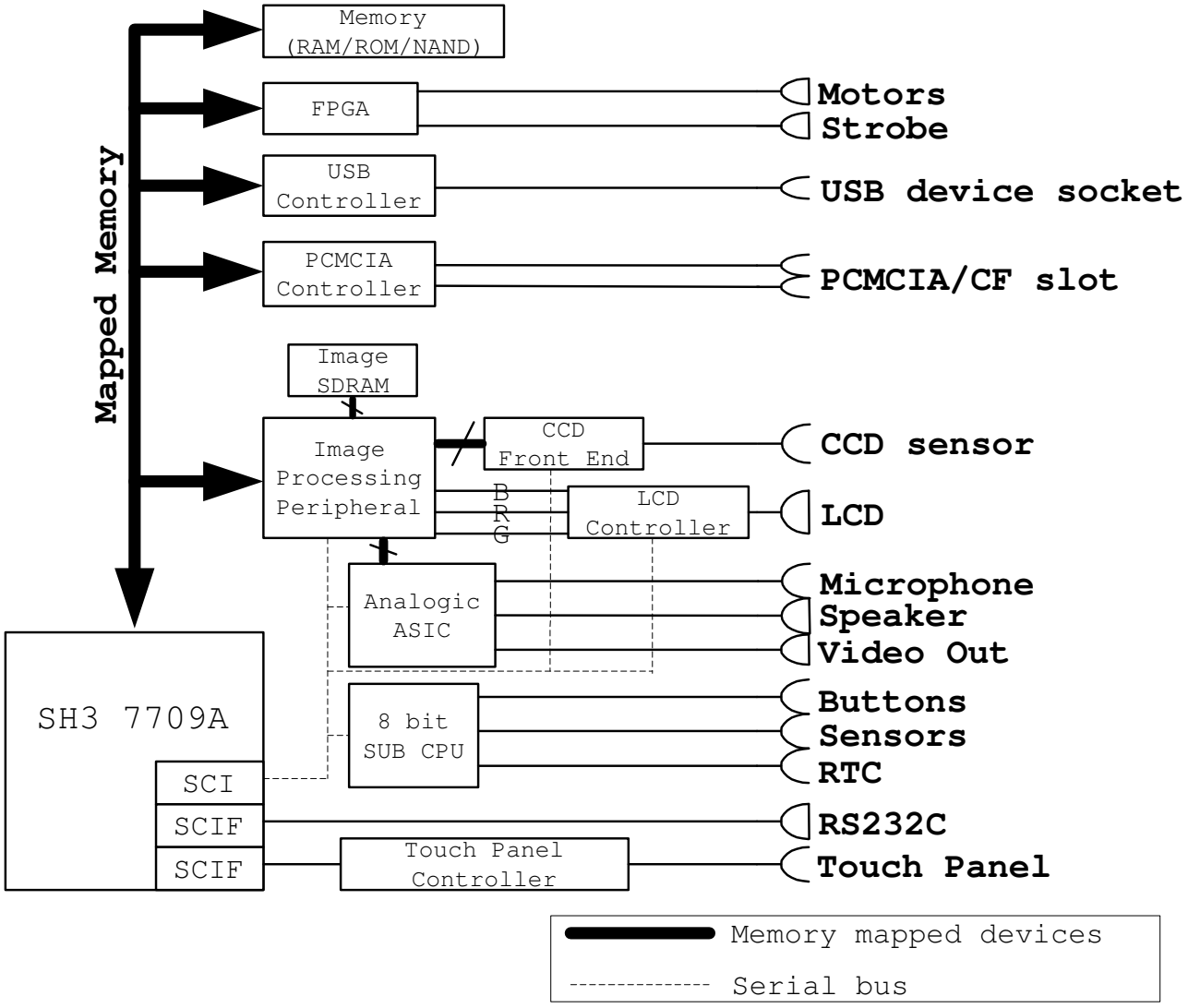

Figure 2: The RDC-i700 peripherals diagram

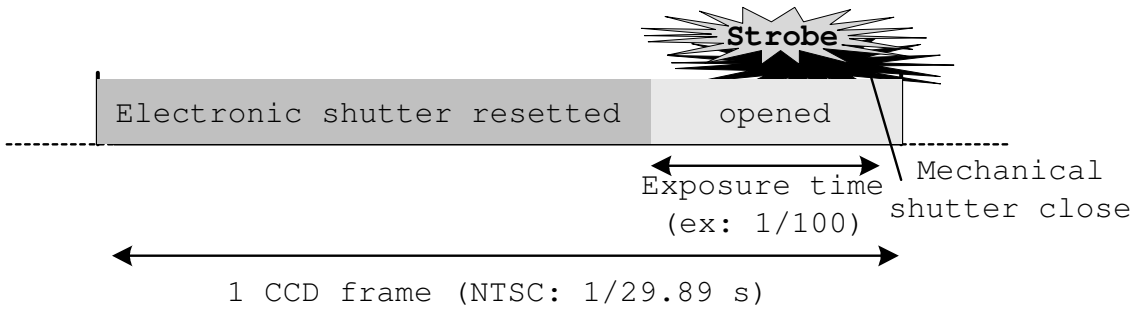

Figure 3: CCD Frame sequence

in both dark and bright conditions. White balance algorithm needs to analyze data coming from the CCD sensor and adjusts the Image Processing Peripheral (IPP) settings in order to have good color matching between the generated image and the reality. Several implementations of those two algorithms can exist (some needs very heavy calculations while others can be very simple), but the main issue is that those algorithms have to be performed very often (in the worst case, every frame of the CCD, that is to say every 33 milli-seconds).

This second part introduced particularity of digital cameras. The next part will discuss how those functions have been implemented into the Linux RDC-i700.

3 Current Support

As the name "Linux on a digital camera" suggests, the RDC-i700 can now run using Linux OS. Although some work remains in some areas, kernel support now exists for most of the hardware and features of the camera. This part explains current status for important features (digital camera related) of the kernel.

3.1 SH-Linux

The RDC-i700 linux kernel is originally based on the work of the SH-Linux [1] team. First tested with the kernel version 2.4.2, the camera is now using the version 2.4.19 of the kernel. SH-Linux kernel already had support for almost all parts of the SH3 7709, but since the RDC-i700 is using the CPU in big endian mode, some modifications were necessary in that field. Source code necessary to run the kernel on this new platform has also been added into `/arch/sh/kernel`.

3.2 RDC-i700 device drivers

RDC-i700 drivers can be separated into two kinds (or two layers). (See figure 4) The lower layer contains so-called **Low level drivers**, or drivers providing control to a specific device (such as focus sensor, IPP ...). All those drivers doesn't have any algorithm included and only provide basic access to the device capabilities. For example in case of the driver controlling motors (MECH driver), only functions provided are to set or get the position of the motor (motors have some predefined positions). RDC-i700 currently has 5 device drivers controlling imaging related devices (we will avoid non-imaging specific drivers here):

- **The CCD F/E (Front End)** which permits to control the CCD parameters (such as exposure time, gain ...)
- **The IPP (Image Processing Peripheral)** which is actually the heart of the camera (almost everything goes through the IPP)
- **The Strobe** driver which allows to charge or flash the strobe
- **The Focus Sensor** which permits to evaluate the distance between the camera and the target
- **Mech** driver which controls all mechanical parts of the camera, that is to say, iris, shutter, zoom and focus.

All those drivers are very system dependent and might change from one camera to another.

On the top of those 5 drivers is what we could called the "Algorithms" layer. This layer contains "intelligent" drivers such as auto focus driver, or auto exposure driver. One more driver, simply called CCD driver, is actually the driver which performs actions such as taking a picture or switching to monitoring mode.

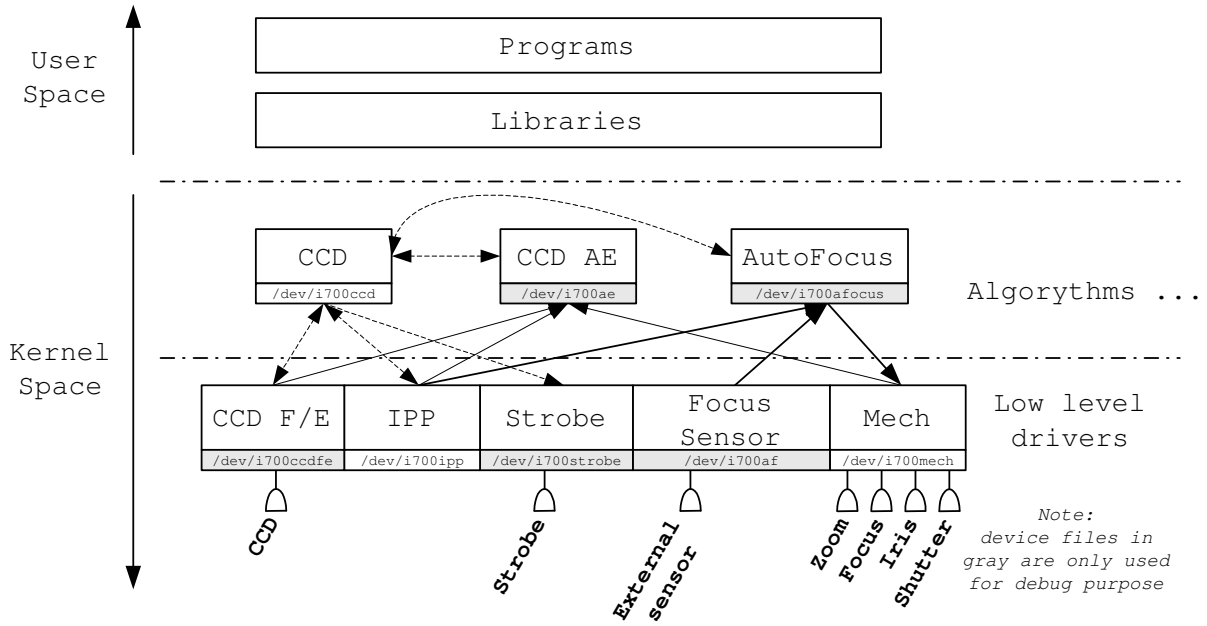

Figure 4: RDC-i700 device drivers

This driver has to access both low level drivers and algorithms drivers. Drivers of the upper layer are "virtually" platform independent. However since the current system lacks of a well defined abstraction layer, upper layer drivers are currently directly accessing lower layer driver which make them unable to work with any other lower driver without having to slightly change the source code. (see Future Work section). Currently device drivers communicate with each others by accessing EXPORTED functions.

All 8 drivers are registered to the kernel as characters drivers and can be accessed from user-level using each device file. Some of those drivers don't actually need to be accessed from user space and in that case device file is only used for debugging purpose. In user space, libraries provide easy access to camera functionalities, avoiding an intensive usage of IOCTL commands.

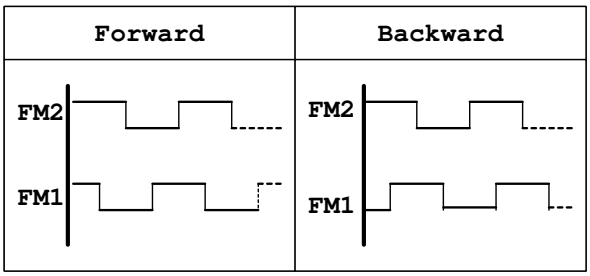

Figure 5: Motors step sequence

3.3 Motors

As the name says, stepping motors are going step by step; the CPU sets 2 I/O ports in order to specify the position of the rotor. Figure 5 shows motor ports state sequence when going forward and backward. Rotation speed is determined by the time between 2 states. Each motor has already predetermined positions, 19 for zoom and 18 for focus; however, if positions for the zoom are fixed, focus position varies depending on the current zoom position. All

those things are handled by the MECH driver and are accessible via IOCTL command such as "Get/Set position." The MECH driver is timer based, that is to say, the delay between 2 steps is performed using a timer (2.8 ms in case of focus, and 1.4 ms in case of zoom); we will see in part 4 the current problems when using this implementation. The driver provides both SYNC and NOSYNC mode; that is to say, in the first one, the ioctl command will hold until the command finish, but in NOSYNC mode, the ioctl will immediately return, allowing to call another ioctl command, even if the motor is still running. This is useful in the case of user's adjustment of the zoom. Since motor needs time to start and stop, it would be inefficient to request each time 1 position change. Instead of this, when the user uses the zoom lever, the first IOCTL command request the motor to go to max position and then when the user release the button, the ioctl STOP command will be requested. In other cases, such as when controlling the FOCUS motor from the auto focus driver, the SYNC mode should be used.

3.4 Auto Exposure

In order to control exposure, the auto exposure driver is accessing 3 different device drivers (IPP, MECH and the CCD front end). The IPP has the ability to divide a CCD image into several block and inform about each block luminance. By looking at those luminance values, the auto exposure algorithm decide how parameters should be modified; it can decide to change iris diameter (MECH driver), or use the strobe (STROBE driver), and in most of the case change parameters of the CCD front end (electronic shutter speed...). The digital camera can work in 2 different modes: the monitoring mode which permits to see in real time what the CCD sensor is targeting, and the still image mode which is used when the user pushes the shutter button to take a still image. The behaviour of the auto exposure module depends on the camera mode:

- **monitoring mode**

in that mode, we adjust CCD F/E parameters every 2 CCD frames. The auto exposure driver starts a kernel thread which needs to be synchronized with the CCD frame (an hardware interrupt is generated by the CCD F/E at every start of frame). Synchronization is achieved by using wait queues. [2] The function which needs to get synchronized creates a wait queue (as follows):

```
struct task_struct *tsk = current;
DECLARE_WAITQUEUE(wait, tsk);
add_wait_queue(&ccd_vd_wq,&wait);
set_current_state(TASK_INTERRUPTIBLE);
schedule();
set_current_state(TASK_RUNNING);
remove_wait_queue(&ccd_vd_wq,&wait);
```

and the wait queue is woken up by the interrupt handler (as follow):

```
wake_up(&ccd_vd_wq);
```

The CCD F/E is controlled using the SCI port of the SH3 which use is shared with some other devices. In some case, it might be necessary to wait for the SCI port availability and, for that reason, the kernel thread implementation has been preferred to some other solutions such as bottom halves (it is not possible to schedule from a bottom halves while kernel thread allows that).

- **still image mode**

in that mode, exposure parameters are only adjusted once before taking the picture. The CCD

[2] This synchronization method is also heavily used by the CCD driver

driver requests information to the auto exposure module which will calculate parameters used to take the picture. After that, the CCD driver will directly control lower driver to set those parameters. Synchronization between all devices is very important and for that reason it is easier to perform everything sequentially from a unique driver. No thread is used and the CCD driver get synchronized with the CCD frame using the same wait queue method as in monitoring mode.

Currently only a very simple algorithm is available for both monitoring and still mode. This algorithm doesn't make use of neither iris nor strobe and the exposure is only controlled using CCD FE's parameters and the mechanical shutter.

3.5 Auto Focus

Compared to the auto exposure, the auto focus driver is quite an easy one. The IPP driver has the ability to determine the "focus level" (the more the focus is correct, the more the value returned by the IPP will be high). In normal mode, the auto-focus driver should get an approximation of the distance to the target by using the focus sensor, then first adjust the focus to this approximation. This permits to perform the "fine focus" (using the IPP capability) to a smaller range. However, the current implementation doesn't use the focus sensor approximation which means that the "fine focus" is performed to the full range of the focus (this is actually the mode which is used in case of MACRO mode). The consequence is that the auto-focus process is much slower than in normal mode. Currently the driver performs the following things:

- check zoom status to calculate focus positions
- retrieve focus level for all focus positions
- go back to the position with the highest focus level

3.6 CCD - IPP

The IPP driver is some kind of library which, except performing initialization of the device, mainly provides a lot of functions, accessible from other drivers and permitting to control the hardware. The driver is quite big since the IPP performs very various things such as

- JPEG compression/decompression
- YUV-RGB conversion
- video output (for the LCD and TV)
- image scaler

The CCD driver is considered as the main driver since almost everything starts from it. It is in charge of coordination between all other drivers. The driver can be controlled using a user land library permitting to control the monitoring mode or to take still image. The driver mainly uses other drivers functions (CCD F/E, IPP, MECH) and performs synchronization using the waitqueue method introduced previously.

3.7 LCD

The RDC-i700 LCD has a fixed resolution of 640x480 pixels. What we could call video card is actually a part of the IPP chip and can control 4 layers of display (1 layer for image/video data, and 3 On Screen Display or OSDs). In the current design the first layer is controlled by the IPP driver and doesn't have direct interface to the user land. Even if 3 OSDs are available, only one is currently used as a framebuffer device. The OSD uses a 8 bit YUV palette (maintained by the IPP device) which

means that `_setcolreg` and `_getcolreg` entry points are used to perform conversion between RGB and YUV color space. This solution allows to use the camera LCD as any traditional Linux console and run any software that usually works on the top of a Linux Framebuffer. One reason why only 1 OSD level is supported is because all OSDs share the same palette which means that it cannot be simply designed as 3 different framebuffer devices. However there is also currently no real need for 3 OSDs so this is not actually a big issue.

3.8 Filesystem

The RDC-i700 has 8 MB of NAND Flash internal memory. The Linux kernel now provides support for this kind of memory by using the Memory Technology Devices (MTD) [2] support. Only a very small layer needs to be written in order to get the camera's NAND work. [3] JFFS2 [4] is usually used on the top of a NAND device, however we will see that in case of digital camera, it might not be the best solution. In our case, internal flash memory is usually exclusively used for storage of compressed data such as JPEG or MPEG. In that case, using JFFS2, which is a compressed filesystem, makes the CPU spend lots of time compressing data which anyway will almost not get compressed more than they are. In such case, YAFFS [5] should be preferred to JFFS2 since it is not a compressed filesystem.(see table 1 for details of tests performed on the RDC-i700)

Time consumption for 1 transaction (secs)		
	YAFFS	JFFS2
JPG (80k)	0.37	0.54
JPG (193k)	0.79	1.32
JPG (547k)	2.21	3.63
MJPG (1463k)	5.6	9.51
*1 transaction = NAND to NAND file copy		

Table 1: YAFFS / JFFS2 tests on RDC-i700

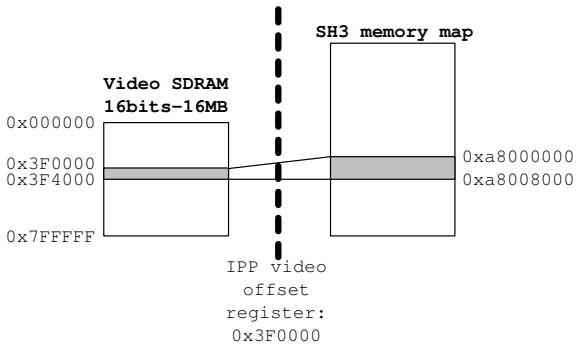

Figure 6: Accessing the video SDRAM

4 Issues

Several problems have been encountered while developing the Linux RDC-i700. Some have been solved but some are still under progress.

4.1 Framebuffer with non-linear memory

The purpose of a framebuffer device is to provide a standard way to access **linearly** video memory from the user space. This is the case of almost (or probably all) video card running on linux. Usually the kernel CPU can directly access any part of the video memory, linearly. However, in case of the RDC-i700, this is not the case. As figure 6 shows, the video SDRAM is not directly accessible from the CPU but is seen through the IPP chip. The IPP provides a 8KB window of memory directly accessible from the SH CPU. By setting a register of the IPP it becomes possible to define which "page" of the SDRAM becomes visible to the CPU. This makes problem with the framebuffer since the video memory is supposed to be linearly accessible, so no method is provided for such kind of system.

Inside a framebuffer device driver, two memory access methods exist:

linux console access mode is performed via function call. 6 functions (read and write for byte, word and long type) are provided. In case of the RDC-i700, the trick is just to overwrite those function in order to set the IPP register to the page needed to be accessed.

mmap access mode is necessary to allow user land application to access the framebuffer memory. The problem in case of mmap access is that the driver doesn't know which part of the memory is being accessed and so it becomes impossible to correctly set the page selector.

To solve this problem, the framebuffer uses a NOPAGE memory handler, combined with the `remap_page_range` function. By always leaving only 1 page mapped at a time, we ensure that the NOPAGE handler will be called everytime the application is trying to access a page which is not mapped. The handler will then unmap the previous page and map the page corresponding to the address to be accessed. One problem remains, which is, if `remap_page_range` allows to map page, it seems there is no function to "unmap" a previously mapped page. The function `zap_page_range` seems to do similar thing and by implementing the nopage handler as follows, the trick seems to work.

```
nopage_handler(...)
{
  calculate pointed addr in SDRAM;
  calculate physical addr;
  if(already_mapped){
    zap_page_range(...);
    flush_tlb_range(...);
  }
  remap_page_range(...);
  already_mapped=1;
}
```

However, some errors occurs time to time when using mmap on the framebuffer and those errors might come from this implementation.

4.2 Timers

We have seen in previous section that several timers are used in various drivers. Some must be very short and for that reason, timer is a very hot topic in our case. Several implementations are possible to perform delay.

busy wait: this solution should be avoided since it would for example almost stop the camera everytime the zoom is adjusted.

kernel timers: kernel timers should be used in order to avoid problems introduced by the busy wait solution. However, in order to achieve such implementation we need first to solve one big problem. While we need about 1ms or less resolution timer, a vanilla 2.4.19 kernel only permits to use timers with resolution of 10ms. In case of stepping motors, this doesn't really make big problem except that motors will just run about 10 times slower than their nominal speed. However, the low accuracy of kernel timers makes problem when used to control other devices such as mechanical shutter, strobe or iris since it can result in low quality image or, even worse, wrong operation performed (narrow instead of large for the iris). In order to solve this problem, several possibilities exist:

- **High Resolution Timer: [7]** is a project hosted on sourceforge in order to add high resolution (nano seconds) capable timer to the Linux kernel. However currently only the i386 architecture is supported, which means that some work is needed in order to get it work on the SH architecture.
- **Hardware Timer:** if such short delay cannot be achieved using software timer,

it still remains the possibility to use hardware timers of the SH3. However, this would make drivers very architecture dependent which should be avoided if possible. Moreover, even if the hardware timer can generate very short period, we need to ensure that the time between the hardware interrupt is generated and the interrupt handler get called is not too long otherwise using hardware timer wouldn't have any meaning. In order to achieve such requirement, it might be necessary to use preemptive kernel.

- **Vanilla kernel with HZ=1000:** changing tick period from 10ms to 1ms allows to use 1ms timer. However, if this solution works fine in some case, we need further experimentation in order to check the accuracy under heavy load condition.

5 Future works

5.1 Design of device driver architecture and user access

Kernel driver layering: the goal is to create a proper abstraction layer permitting to have upper kernel drivers (algorithms) totally independent from the hardware. Currently EXPORTED functions are used to allow function calls between drivers, however it means that upper drivers must understand the behavior of hardware drivers. The abstraction layer needs to define both function prototypes and structures used to access lower driver functionalities. Such interface would permit to easily customize any upper drivers, for example auto exposure algorithm or auto white balance.

Exporting functionalities to user space: currently small libraries are available, permitting to control camera functionalities. However, it doesn't seem reasonable to write new libraries specifically for the camera. Modifying device drivers to make them compatible with some existing standard should be the solution to take advantage of the large amount of existing software. In the camera field, Video For Linux would probably be a good candidate, and especially the second release which is currently under development. The idea would be to provide access to the CCD as a standard video input interface, similar to any USB camera for example. Other functionalities, such as JPEG compression, decompression could be accessed as a CODEC.

5.2 Remaining tasks

Some features still need to be implemented on the camera such as:

Power Management: currently no power management is performed while running Linux on the RDC-i700. This makes the battery life as short as about 25 minutes when using PCMCIA cards. This part should be the next big issue for the linux RDC-i700.

USB controller: the RDC-i700 includes a PDIUSBD12 USB device (slave) controller[3]. The Linux-USB Gadget API [8] allows to easily implement USB device class on the top of controller drivers, however this device controller is currently not supported yet.

6 Conclusion

The linux RDC-i700 has now enough support in order to be used as a digital camera. Most of the constrains due to the architecture and specific hardware have been solved but we still need some more performance testing in order to ensure that everything can run well. But remaining issues are not only technical one. Since we are now preparing for distributing the

[3] http://www.semiconductors.philips.com/pip/PDIUSBD12.html

source codes, we still needs some more coordination in our company. We also have to think of how to make and support a developing community.

References

[1] SH-Linux
http://linuxsh.sourceforge.net

[2] Memory Technology Devices (MTD)
http://www.linux-mtd.infradead.org

[3] MTD's NAND Flash support
http://www.linux-mtd.infradead.org/tech/nand.html

[4] JFFS2 homepage
http://source.redhat.com/jffs2/

[5] YAFFS homepage
http://www.aleph1.co.uk/yaffs/

[6] Linux Framebuffer Driver Writing HOWTO
http://linux-fbdev.sourceforge.net/HOWTO/

[7] High Resolution Timer Project
http://high-res-timers.sourceforge.net/

[8] USB-Gadget API
http://www.linux-usb.org/gadget/

[9] *Linux Device Drivers*, 2nd Edition
Alessandro Rubini & Jonathan Corbet

Proceedings of the Linux Symposium

Volume Two

July 21st–24th, 2004
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Jes Sorensen, *Wild Open Source, Inc.*
Matt Domsch, *Dell*
Gerrit Huizenga, *IBM*
Matthew Wilcox, *Hewlett-Packard*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*