Lab 8: CMOS image

1. Objective

- **Understand CMOS sensor behavior and web server.**

2. Prerequisite

- **Find out how to translate raw data of image to bmp(or jpeg).**

3. CMOS image

3.1 Compile and load modules

Rebuild your kernel and rootfile system to support creator-pxa270-cmos.ko module:

- Get the source code of creator-pxa270-cmos.ko:
  Download Creator_PXA270_CMOS_Device_Driver.src.tar.gz from E3 website and decompressed it to your kernel source.

```
SHELL>    cd ~
SHELL>    tar xzvf Creator_PXA270_CMOS_Device_Driver.src.tar.gz
```

- Configure kernel source:

```
SHELL>    cd ~/microtime/linux
SHELL>    make mrproper
SHELL>    make menuconfig
```

  - In the window of "Linux Kernel Configuration", select "Load an Alternate Configure from File" and load the configuration file arch/arm/configs/creator_pxa270_defconfig.
  - Select "Device Drivers" → "Character devices" and mark "Creator-pxa270 CMOS" as [M].
  - Save and exit kernel configuration.

- Make Image:
  Compile Linux kernel and creator-pxa270-cmos.ko module.

```
SHELL>    make clean
SHELL>    make
```

  The creator-pxa270-cmos.ko module will be placed at microtime/linux/drivers/char/.

- Make new root filesystem:
  Copy creator-pxa270-lcd.ko module into root filesystem.

```
SHELL>    cp ~/microtime/linux/drivers/char/creator-pxa270-cmos.ko
~/microtime/rootfs/lib/modules/2.6.15.3/kernel/drivers/char/
```

  Then,rebuild and flash root filesystem.

- Load modules

Type the following command to load the creator-pxa270-cmos.ko on PXA270

```
>   insmod /lib/modules/2.6.15.3/kernel/drivers/char/creator-pxa270-cmos.ko
```

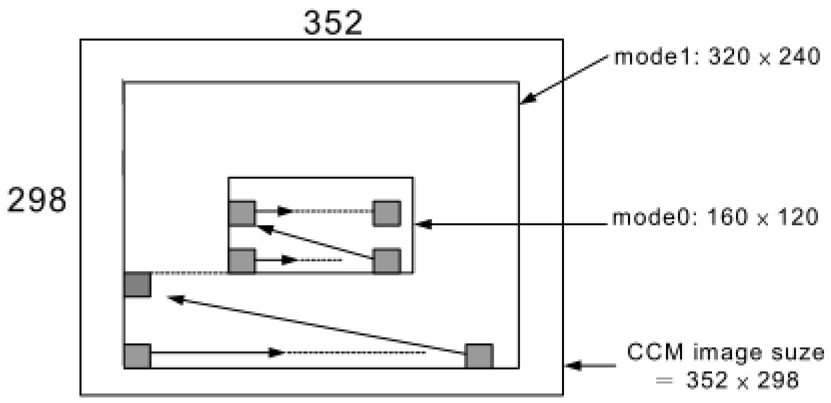
3.2 Read CMOS image

CMOS Camera Module(以下簡稱 CCM)，CCM 功用是將 sensor 所感知的訊號轉為數位之資料，資料再由應用程式處理。在 Creator 上的 CCM，它的處理週期可依序分為曝光時間與影像轉換，而曝光時間影響影像的明亮度，數值範圍為 0-255，數值越大曝光時間越長，使得影像整體亮度變大。影像轉換時間由影像 mode 所決定，mode 有 mode 0 與 mode 1，其大小如下：

Mode	Size
0	160*120
1	320*240

- Image format

CCM 的 sensor 可視範圍為 352*298，而透過 CCM Controller 可設定 CCM 的影像模式為 160*120 或 320*240。



而資料讀取之方式為由左到右，由下到上(參考上圖)。Device driver 所傳回的影像 raw data 之排列如下:

```
byte      1 : row(1),col(1)        第 1 列第 1 點
byte      2 : row(1),col(2)        第 1 列第 2 點
…
byte 19200 : row(160),col(120)     第 160 列第 120 點
```

每一個點占用 1 個 byte，因此當影像 Mode 0=160*120 時，raw data 總共 160*120=19200bytes。

- CMOS image programming guide

Header file:

```
#include "asm-arm/arch-pxa/lib/creator_pxa270_cmos.h"
```

Commands:

int ioctl(int src, int command, struct CMOS_INFO *data)： 設定參數 Set Parameters

src	：進入節點(node) enter the node
command = CMOS_ON	：啟動 CCM 開始做資料轉換且在 data 中

Start CCM started doing data conversion and specify the image Mode in the data , and returns the length and width of the image data

		指定影像 Mode，並且傳回影像長與寬到 data。
= CMOS_OFF	:	停止 CCM 資料轉換。Stop CCM data conversion
= CMOS_PARAMETER	:	改變曝光時間，數值由 data 之 HighRef 設定。
= CMOS_GET_STATUS	:	傳回 CCM 的狀態值，型態為 cmos_status_e。

Status:

CMOS_IMG_READY	: CCM 已經起動且影像資料準備好。
CMOS_IMG_EMPTY	: CCM 已經起動但影像資料沒準備好。/

Sample code:

```
/*cmos.c*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/fcntl.h>
#include "asm-arm/arch-pxa/lib/creator_pxa270_cmos.h"
main()
{
  int terminate = 1;
  int dwTotalReadSize, dwImageSize, nRead;
  int count;
  int fd_cmos;
  cmos_info_t CMOSInfo;
  unsigned char *pImageBuffer=NULL;

  dwImageSize = 160*120;
  pImageBuffer = malloc(dwImageSize);
  fd_cmos = open("/dev/cmos",O_RDWR);
  CMOSInfo.ImageSizeMode = CMOS_SIZE_MODE_160_120;
  CMOSInfo.HighRef = 120;
  ioctl(fd_cmos, CMOS_PARAMETER, &CMOSInfo);
  ioctl(fd_cmos, CMOS_ON, &CMOSInfo);
  while (terminate)
  {
    if (ioctl(fd_cmos, CMOS_GET_STATUS, &CMOSInfo) >= 0)
            if (CMOSInfo.Status == CMOS_IMG_READY)
            {
                    dwTotalReadSize = 0;
                    count = dwImageSize;
                    do
                    {
                       if (count + dwTotalReadSize > dwImageSize)
```

```
                count = dwImageSize - dwTotalReadSize;
            if ((nRead = read(fd_cmos, pImageBuffer + dwTotalReadSize, count)) > 0)
                dwTotalReadSize += nRead;
            else if (nRead == 0)
                break; /* EOF */
           }
           while (dwTotalReadSize < dwImageSize);
           /* now your image raw data is in the buffer. */
       }
 }

 if (pImageBuffer)
     free(pImageBuffer);
 ioctl(fd_cmos, CMOS_OFF, &CMOSInfo);
 close(fd_cmos);
}
```

Add the header search path when compile cmos.c.

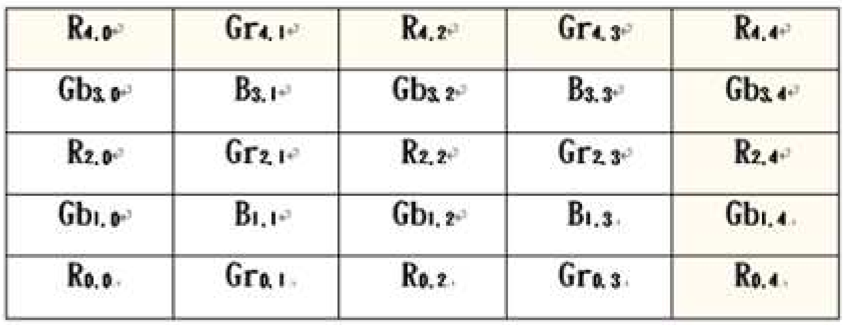
```
SHELL>    arm-unknown-linux-gnu-gcc -o cmos cmos.c
-L /opt/arm-unknown-linux-gnu/arm-unknown-linux-gnu/lib/
-I /opt/arm-unknown-linux-gnu/arm-unknown-linux-gnu/include/
-I /home/lab616/microtime/linux/include/
```

If the error “nfs: server is not responding, still trying” shows up when nfs mounting, you may use the following command to fix the error.

```
>    mount -t nfs –o nolock –o tcp <ip2>:<absolute path to export point> /mnt
```

3.3 Image interpolation

CMOS 所感知每一點資料陣列是使用 color filter array(CFA)和 microlens(ML)所組成，CFA 的排列規則是 RGB Mosoc Bayer’s pattern，第一列的有效資料為 RGRGRGRG…，第二列有效資料為 GBGBGBGB…，如圖所示。然後透過「bayer geometry」可將影像還原成 RGB。「bayer geometry」請參考附件。



$R_{4,0}$	$Gr_{4,1}$	$R_{4,2}$	$Gr_{4,3}$	$R_{4,4}$
$Gb_{3,0}$	$B_{3,1}$	$Gb_{3,2}$	$B_{3,3}$	$Gb_{3,4}$
$R_{2,0}$	$Gr_{2,1}$	$R_{2,2}$	$Gr_{2,3}$	$R_{2,4}$
$Gb_{1,0}$	$B_{1,1}$	$Gb_{1,2}$	$B_{1,3}$	$Gb_{1,4}$
$R_{0,0}$	$Gr_{0,1}$	$R_{0,2}$	$Gr_{0,3}$	$R_{0,4}$

Color coding

3.3 Save BMP file

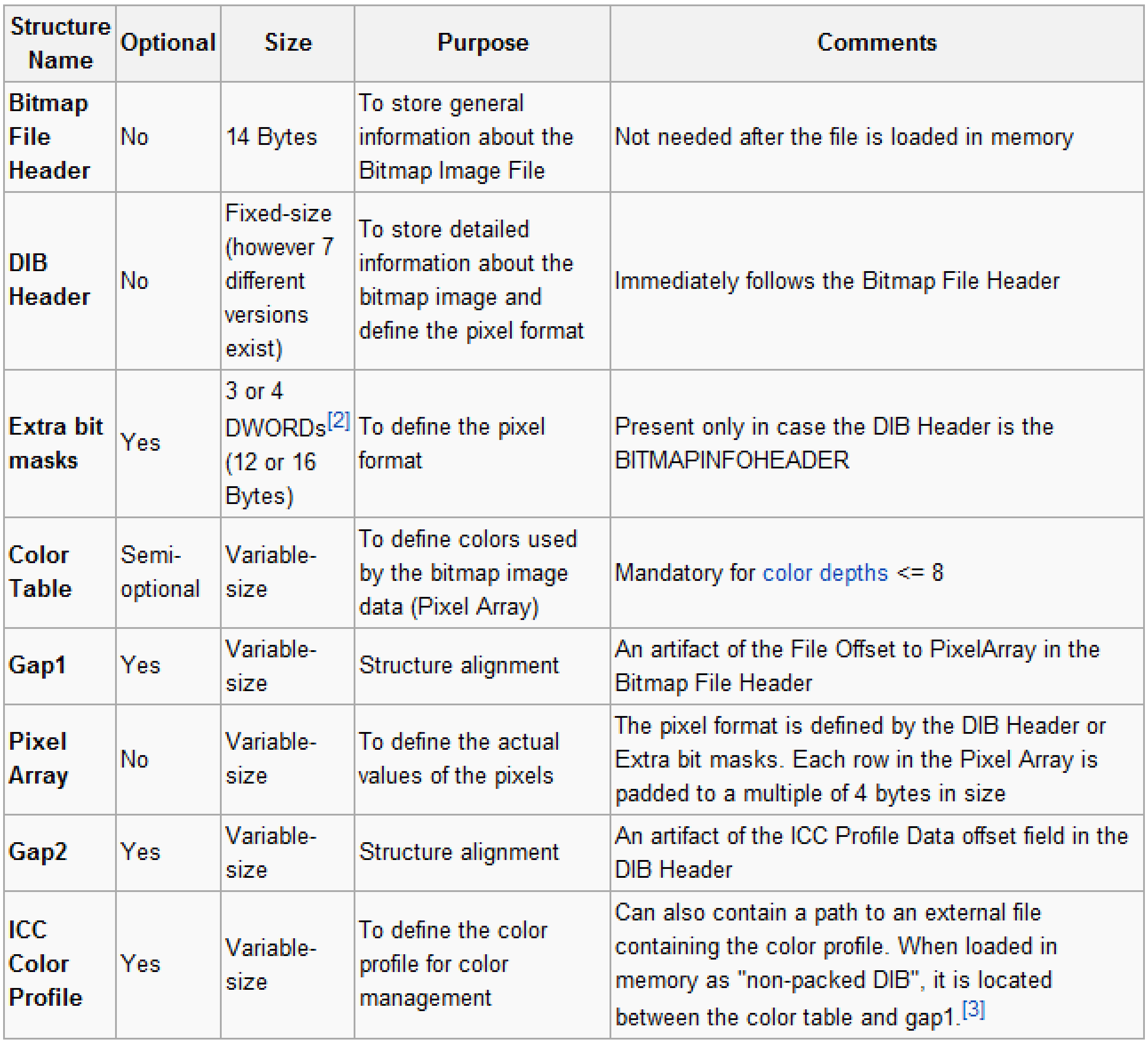
The BMP file format, also known as bitmap image file or Device Independent Bitmap (DIB) file format or

simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS/2 operating systems.
The BMP file format is capable of storing 2D digital images of arbitrary width, height, and resolution, both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles.

- File structure

  The bitmap image file consists of fixed-size structures (headers) as well as variable-size structures appearing in a predetermined sequence. Many different versions of some of these structures can appear in the file, due to the long evolution of this file format.

  The bitmap file is composed of structures in the following order:



Structure Name	Optional	Size	Purpose	Comments
Bitmap File Header	No	14 Bytes	To store general information about the Bitmap Image File	Not needed after the file is loaded in memory
DIB Header	No	Fixed-size (however 7 different versions exist)	To store detailed information about the bitmap image and define the pixel format	Immediately follows the Bitmap File Header
Extra bit masks	Yes	3 or 4 DWORDs[2] (12 or 16 Bytes)	To define the pixel format	Present only in case the DIB Header is the BITMAPINFOHEADER
Color Table	Semi-optional	Variable-size	To define colors used by the bitmap image data (Pixel Array)	Mandatory for color depths <= 8
Gap1	Yes	Variable-size	Structure alignment	An artifact of the File Offset to PixelArray in the Bitmap File Header
Pixel Array	No	Variable-size	To define the actual values of the pixels	The pixel format is defined by the DIB Header or Extra bit masks. Each row in the Pixel Array is padded to a multiple of 4 bytes in size
Gap2	Yes	Variable-size	Structure alignment	An artifact of the ICC Profile Data offset field in the DIB Header
ICC Color Profile	Yes	Variable-size	To define the color profile for color management	Can also contain a path to an external file containing the color profile. When loaded in memory as "non-packed DIB", it is located between the color table and gap1.[3]

4. Web server

A web server is needed on the embedded Linux to server the image. There are several open source projects of small-size web servers which you can use.

4.1 Boa

Boa is a single-tasking HTTP server. That means that unlike traditional web servers, it does not fork for each incoming connection, nor does it fork many copies of itself to handle multiple connections. It internally multiplexes all of the ongoing HTTP connections, and forks only for CGI programs (which must be separate processes), automatic directory generation, and automatic file gunzipping.For more information, visit: http://www.boa.org/

- Setting up

Download boa.tar.gz from E3 website and move it to the <tftproot>.Then use tftp command to transfer it to target board and decompressed it.

```
>   cd /tmp
>   tftp –gr boa.tar.gz <ip2>
```

```
>   tar zxvf boa.tar.gz
```

Create necessary directory and move file.

```
>   mkdir –p /etc/boa
>   mkdir –p /var/log/boa
>   mkdir –p /var/www/cgi-bin
>   mv boa /bin
>   mv boa.conf /etc/boa
>   mv mime.types /etc
>   mv index /var/www
```

Change boa mode. Then start the boa server.

```
>   cd /bin
>   chmod 777 boa
>   boa
```

Use browser to connect <ip3> on Linux. If your boa server works normally, you will see “Hello world”.

4.2 cgi program

The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

For example, let's say that you wanted to "hook up" your Linux database to the World Wide Web, to allow people from all over the world to query it. Basically, you need to create a CGI program that the Web daemon will execute to transmit information to the database engine, and receive the results back again and display them to the client. This is an example of a *gateway*, and this is where CGI, currently version 1.1, got its origins.

The database example is a simple idea, but most of the time rather difficult to implement. There really is no limit as to what you can hook up to the Web. The only thing you need to remember is that whatever your CGI program does, it should not take too long to process. Otherwise, the user will just be staring at their browser waiting for something to happen.

- Sample code:

```
/*cgi.c*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void header(void);
void footer(void);

int main(void){
        header();
        printf(“Hello World\n\n”);

        footer();
        return 0;
}

void header(void){
        printf("Content-type: text/html\n\n");
        printf("<html>\n<body>\n\n");
        return;
}

void footer(void){
        printf("</body>\n</html>\n");
        return;
}
```

Use cross-compiler to build up executable file for PXA270:

```
SHELL> arm-unknown-linux-gnu-gcc –o cgi cgi.c
```

Move cgi.cgi to the <tftproot>.Then use tftp command to transfer it to target board and move it.

```
>   cd /tmp
>   tftp –gr cgi <ip2>
>   mv cgi /var/www/cgi-bin
```

Change cgi.cgi mode.

```
>   cd /var/www/cgi-bin
>   chmod 777 cgi
```

Use browser to connect <ip3>/cgi-bin/cgi on Linux. If your cgi works normally, you will see “Hello world”.

Lab 8 Assignments

Connect the CMOS camera to PXA270.

1. Read the **raw image** (without any image format) from the CMOS camera.
   Transform the raw image to **BMP color** image (xxx.bmp).

2. Compress BMP color image to **JPEG image** (xxx.jpg).

3. Start **BOA web server** and write a **CGI program** that generates a webpage.
   Show the JPEG(or BMP) image.