Final Project Report: Pelmanism

Embedded System Design (CSEE 4840)

Prof Stephen Edwards

Can Ilhan (ci2137@columbia.edu)
Chintan Shah (cds2127@columbia.edu)
Sungjun Kim (sk3062@columbia.edu)
Zenan Li (zl2174@columbia.edu)

Contents

1. Overview

The goal of our project was to develop an interactive, picture-based version of the popular memory game played with cards, so called Pelmanism. The basic idea of Pelmanism is that a player should match two same pictures on the screen, recalling what he or she has seen before. In the beginning of the game, the back side of all the pictures is shown, hiding the front image. Then, the player turns over two cards in every turn. If the two pictures match, these are cleared from the screen; otherwise, the pictures are flipped over again. The game is over when the last pair is matched. The objective is to clear the screen in the lowest number of turns. Figure 1.1 illustrates a game of Pelmanism.

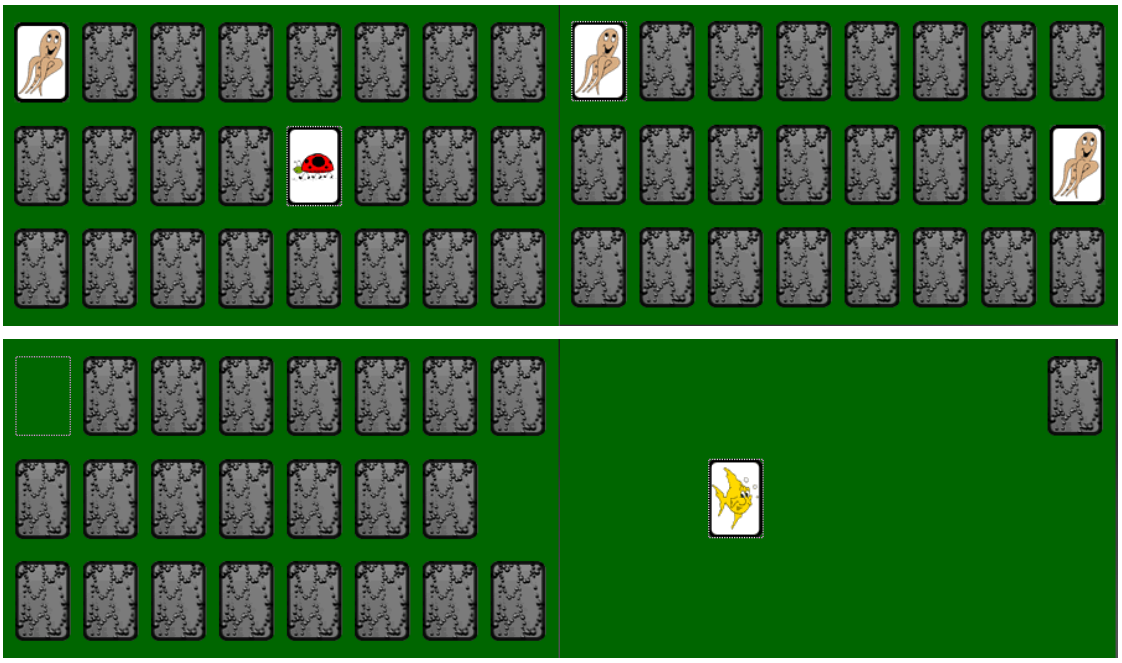


Figure 1.1: An illustration of Pelmanism

In our implementation of the game, we read pictures pre-stored in the SD card in JPEG format. The pictures are then decoded using a JPEG decoder which is implemented in software. The decoded pictures are then transferred to the on-board SRAM. These are then read by the VGA controller and displayed on the screen. One of the features of our game is animation effects during game play which is implemented entirely in hardware.

2. Architecture

In general, the design is divided into the hardware part and the software part. The main structure of the hardware design is shown in Figure 1.2 and the software design is shown in Figure 1.3.

The function of the SD card controller is interface the SD card slot and read pictures from it. The SRAM stores the pictures once it is transferred from the SD card. The VGA controller is the main hardware component. It interacts with the VGA DAC and provides it with proper signals. It reads the data from the SRAM and also contains the animation logic which is triggered by user actions. The PS2 controller's function is to provide interface with the mouse. The SDRAM stores the software program.

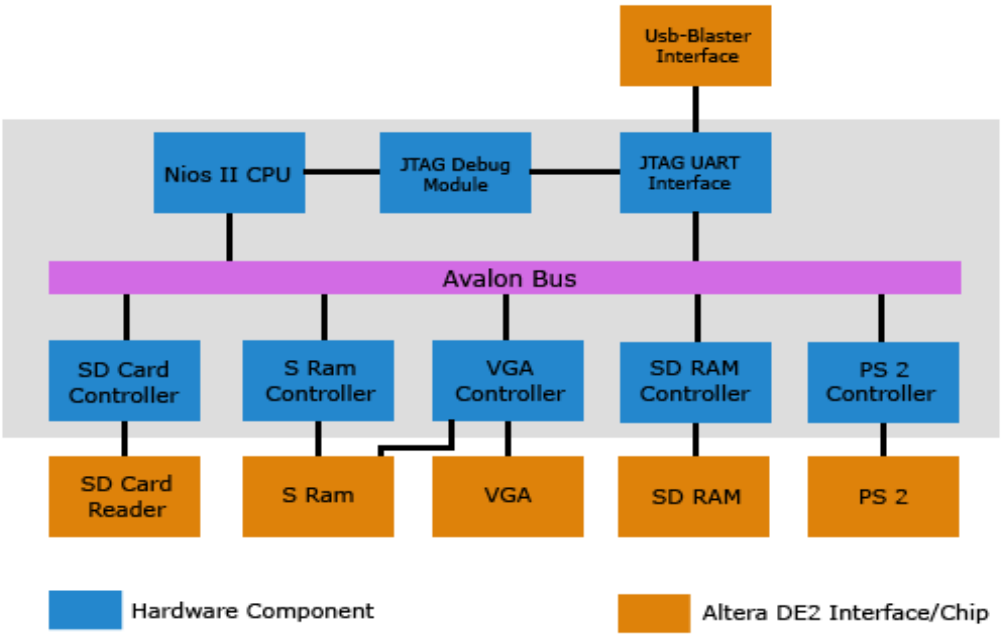


Figure 1.2: Overall structure of our hardware design

The software initializes and controls the peripherals, decodes the pictures with the JPEG decoder and transfers it to the SRAM. It also has the code to read the FAT file-system of the SD card. The mouse is movement is processed by the software and then the actions are communicated to the VGA controller.

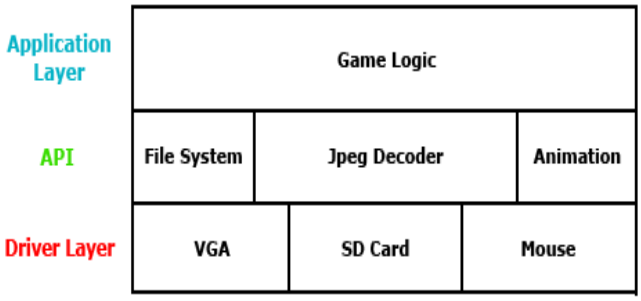


Figure 1.3: Overall architecture of our software design

2. Hardware Design

2.1 VGA Controller

One of the critical **hardware** components that we have implemented was VGA controller which is talking to VGA DAC **and** SRAM presented on DE2 board. Porch values, sync declarations and other important values for 640x480 resolution are already defined in the vhdl file in lab3 and we have used that in our controller. We have added code that helped us to achieve our expectations from VGA controller. We will talk about *cursor drawing*, *SRAM interface*, *pixel drawing* and lastly *animation/visual* effect in this part. The memory mapped I/O can be depicted as below:

Address Value	Processes
0x00	Cursor X & Y pos
0x01	
0x02	SRAM r/w command
0x03	Sprite Status Information (clicked image array)
0x04 - 0x13	Animation - Visual Effect

Fig 2.1 Address Space of VGA-Controller

To make user interface practical, we implemented PS-2 mouse to our system which you can find details about that hardware component in this report. We have defined a two color cursor sprite (the way we defined the ball for lab3) in VGA controller with address 0x00 reserved for it. Note that we use **32bit data bus** for transactions so we can able to send both X and Y positions of the cursor from Nios-2 using only one transaction. The protocol is pretty simple. Nios-2 calculates the new coordinates of a cursor (setting a default starting point 0,0 in the very beginning and calculating the new coordinate using ps-2 controller's output) and puts the x coordinate to the first ten bits and adds y coordinate to the second ten bits of the 32bit array. Therefore 9 down to 0 holds X coordinate and 19 down to 10 keeps Y coordinate value and VGA controller, after receiving this array, splits it properly and draws the cursor depending on the values. Moreover cursor drawing, as usual, has the highest priority (i.e. it is in the highest layer and can overdraw pictures, border etc.).

VGA component, after 16 images are saved into SRAM (check SRAM controller), starts drawing the sprites for 16 predefined rectangle areas. Reading from SRAM and drawing the resulting pixels is actually pretty straightforward. We set "read" flag to "0" from Nios-2, indicating that write phase has been finished, and send that flag to the address 0x02. After that time, any SRAM address value that is being sent by Nios2 will let VGA controller to grab that information from SRAM and show it (with an appropriate animation).

We assume that SDCard holds images that have the same resolution that we have defined for the GUI. You can find the illustration of our GUI below:

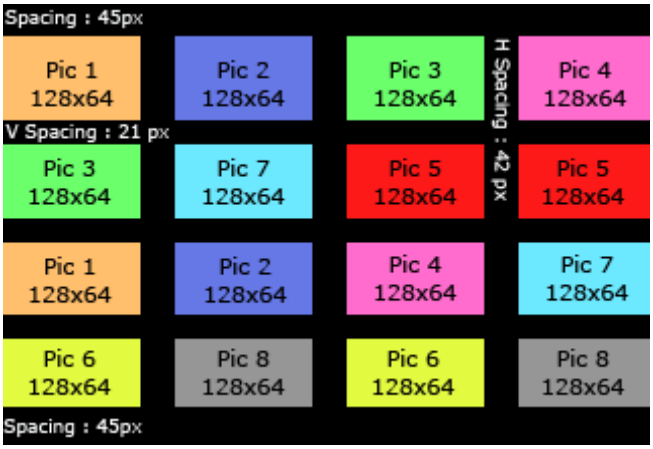


Fig2.2 GUI that depicts the sprites and spacings

We changed the hardcoded resolution to make the animation logic simpler, which is now composed of the two's power (128x64) and any division/multiplication is done by shifting left or right easily. Vertical and Horizontal spacing values are selected such that the aspect ratio is kept 2:1 and there isn't any wasted space horizontally.

Address Value	Processes
0x00	Cursor X & Y pos
0x01	
0x02	SRAM r/w command
0x03	Sprite Status Information (clicked image array)
0x04 - 0x13	Animation - Visual Effect

Fig2.3 Addressing scheme

Whenever the player clicks an image, Nios2 sends a clicked image array to VGA controller's 0x03 address (which is all zeros in default). Clicked image array is a 32bit wide array and the first 16 bits hold information about whether corresponding sprite has been clicked or not. The higher 2 bytes hold whether sprite is still valid (when corresponding bit is zero) or should be removed (when set to one). That is to say, the first sprite will first check the sixteenth bit of the array and if it is set to zero, it assumes that the sprite hasn't been removed yet and will check zero'th bit to check whether it has been clicked or not. If the value is zero, it will show the background image (the last image in the Sram and starting address is hardcoded) otherwise it will show the corresponding image with selected animation (again for each sprite, the Sram address that they should start reading is known apriori). When the user finds two copies of an image, it will be removed from the screen by setting the corresponding bit to '1' by Nios-2 (from the

higher two bytes of clicked image vector). If user clicks two different images, Nios2 will wait some time and folds off both images by showing the background. This is done by setting corresponding clicked image vector to '0' (from the lower two bytes of clicked image vector). One example can be shown as below:

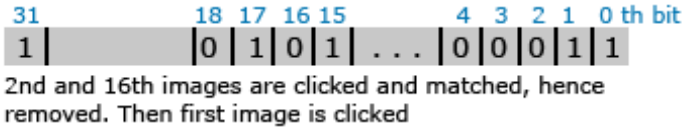


Fig2.4 Sample clicked image array

Note that we are using 5 bits instead of 10 bits for each color channel to save space and make the logic simpler. Therefore 15 bits is enough for a pixel to represent that can be saved and retrieved in one transaction. While sending to VGA DAC we simply copy the higher 5 bits (the actual pixel value that we did get from SRAM) to lower 5 bits and send 10 bits. Each picture will take 128*64*2 (16.384 bytes) bytes of space since each pixel needs 2 bytes. Therefore, total of 128*64*2*17 (16 images and the background) is used from SRAM. It is exactly 272Kbytes and easily fits to 512k SRAM we have.

2.1.1 Animation Effects

Animation Effect:
We have two types of animations in our project implemented entirely in hardware.
1) Expanding and squeezing
2) Fading out.

Expanding and Squeezing:
Whenever the player clicks the back image, it is turned over and it the original picture is revealed.
This animation consists of squeezing the back image and expanding the front image. When the picture needs to be turned to the back side, the reverse process takes place. i.e. expanding the back image and squeezing the front image. When this is done, it gives the impression that the picture is turning over. In order to realize this effect, we implement and offset table of size 30 x 128. The row size is 30 because the animation lasts for30 frames. In each frame, the row corresponding to the animation state is read. For example, in the 15^{th} frame of the animation, the 15^{th} row elements will be read. In the offset table, there are dummy values (255) and real offset values. If the column is 255, then that pixel is displayed as black. Otherwise the offset is added to the current value of the read address of the SRAM. In successive rows, the number of black pixels (255) values is reduced successively, thus the image expands. The reverse process happens for the squeezing effect.

Fading out:
The fading out animation takes place when two pictures are matched and both need to be cleared off the screen. While clearing, instead of just disappearing from the screen, the pictures fade out giving a nice effect. In oorder to implement this animation, we converted RGB value to YUV value. After converting them, we decreased Y value according to the frames, and converted to RGB again. Converting RGB into YUV is through the following matrix transformation.

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

In pseudo code, this can be expressed as,

// Basic Transform
Y' = 66 * R + 129 * G + 25 * B
U = -38 * R - 74 * G + 112 * B
V = 112 * R - 94 * G - 18 * B

// Scale down to 8 bits with rounding
Y' = (Y' + 128) >> 8
U = (U + 128) >> 8
V = (V + 128) >> 8

// Shift values
Y' += 16
U += 128
V += 128
After we convert the signal into YUV, we decreased luminance according to the frame status.
Y = Y * (1 - current_frame / total_frames)
In VHDL, division calculation is too slow to meet VGA clock timing constraints. Thus, we changed division into shift operation.
Y = Y * (1 – current_frame >> 5)
If it goes to negative, RGB is all set to 0, or we convert the YUV again into RGB value.
C = Y' - 16
D = U - 128
E = V - 128
R = clip ((298 * C + 409 * E + 128) >> 8)
G = clip ((298 * C - 100 * D - 208 * E + 128) >> 8)
B = clip ((298 * C + 516 * D + 128) >> 8)

2.2 SRAM Controller

Even tough SRAM Controller is trivial and already defined in SOPC builder before; we would like to mention how it is going to store our images. The overall picture is as follows: After we fetch the 8 images from SD-Card, the JPEG Decoder will convert them into raster format and the result will be stored into our SRAM randomly (so that the first sprite will not be necessarily the first image and so on). You can find below how SRAM looks like after the first 2 pictures (note that they are randomized by software) are sent to SRAM-controller's 0x00 address and after it saves those to SRAM.

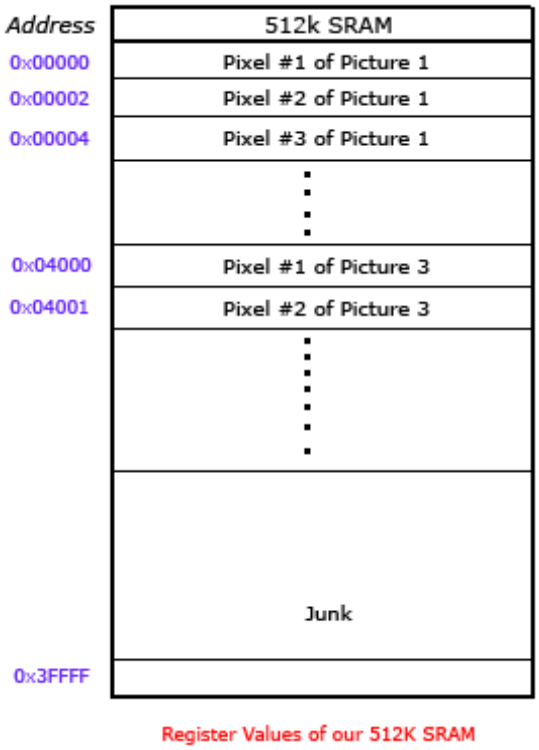


Fig 2.5

2.3 SDRAM Controller

Since we are using SRAM for saving decoded Jpegs (bitmaps), we are not saving our C++ code in the same memory because we find that it would not fit there (after defining Jpeg decoder, file system, drivers and API's in software). That's why we are using off-chip 8Mb SDRAM. The controller definition is trivial since it is already defined in SOPC builder and we don't need to modify it. It is actually using PC100 standard to talk to SDRAM. All of the signals except the Clock signal are already defined in SOPC controller and we have connected our global clock directly to it to make it work. Below you can find the interface between the controller and SDRAM chip itself.

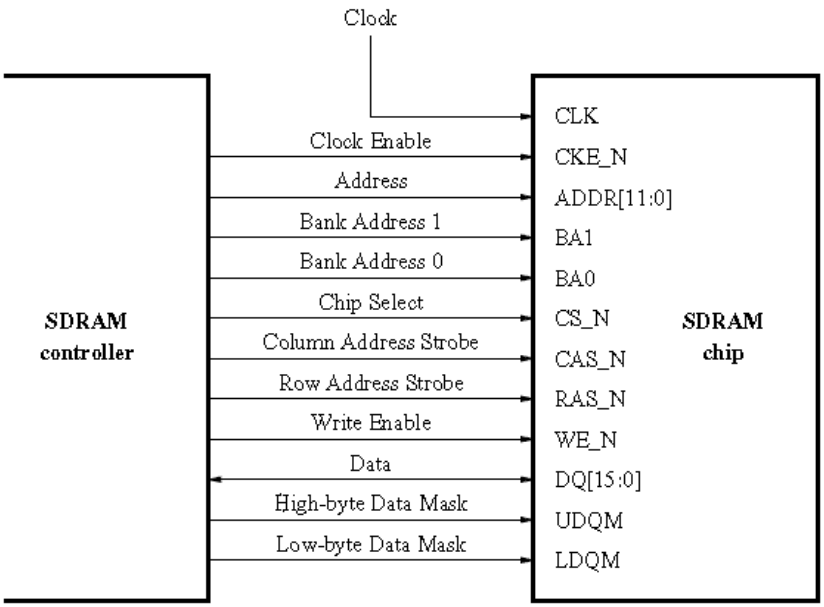


Figure 2. The SDRAM signals.

2.4 PS2 controller

In this project, since PS2 mouse is used, PS2 controller is implemented. Data communication is done according to the clock as follows:

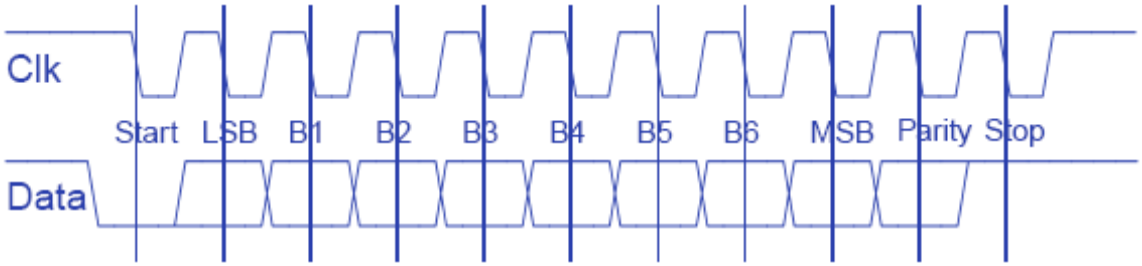


Figure 2.7: The clocks of data communication

Therefore, in the hardware part, PS2 controller reads these data and sends them to the software side. Software mouse driver processes the data: Overflow, Sign, Buttons, X movement and Y movement.

2.5 SD Card Controller

The contents of the SD card are read in the SPI mode. The connection of the FPGA with the SD Card slot is as shown in the diagram below.

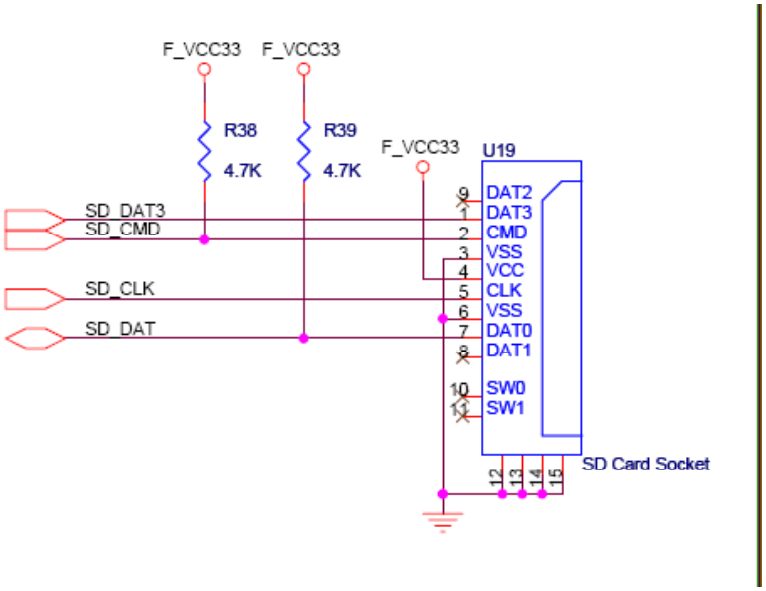


Figure 2.8: Connection with the FPGA

The SD Card supports three protocols which can be divided into two classes, the proprietary SD Mode and the open SPI mode. The DE2 board supports the SPI mode and we will be using this mode. It is to be noted that the SPI communications mode supports only a subset of the full SD communications protocol. However, these functionalities are enough to achieve our purpose. A diagram of the SD card is shown below.

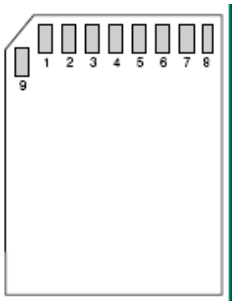


Figure 2.9: SD Card Diagram

The table below lists the pin assignments for the SD card and their functions in the SPI mode.

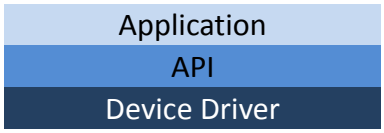
Table 2.1: SD card pin assignments in the SPI mode

Pin	Name	Function (SPI Mode)
1	DAT3/CS	Chip Select/Slave Select (SS)
2	CMD/DI	Master Out Slave In (MOSI)
3	VSS1	Ground
4	VDD	Supply Voltage
5	CLK	Clock (SCK)
6	VSS2	Ground
7	DAT0/DO	Master In Slave Out (MISO)
8	DAT1/IRQ	Unused or IRQ
9	DAT2/NC	Unused

In SPI mode, four signals (clock, data in, data out and chip select) are used for the interface. The clock is used to drive data out on the data out pin and receive data on the data in pin. The software drives commands and data to the SD card over the SD card's data in pin. The software receives response and data from the card on its data out pin. The chip select signal is used to enable the card during data and command transfer. For each of the pins, we have a separate peripheral in hardware. Everything else is controlled by the software.

3. Software Architecture

Our basic software design is 3-layered architecture: application, API, device driver. In the device driver layer, code is written as microcode with C language; in the API and application, it is coded as object-oriented programming with C++ language.



3.1 Device driver layer

Device driver layer abstracts hardware devices such as VGA, SD card and PS2 mouse. In this layer, all the microcode of our game software will be isolated. Basic role of this layer is providing abstracted functionality to the API layer.

3.1.1 VGA driver

VGA driver provides drawing pixels, giving visual effects and cleaning LCD screen or pixel. This driver is interfaced with the hardware VGA interface. Since most of functionalities will be implemented in the hardware side, this driver simply interfaces the hardware

3.1.2 SD card driver

SD card driver communicates with SD card interface in the hardware layer. This driver reads data from SD card or writes data to the card. Additionally, it checks whether SD card is inserted.

3.1.2.1 Protocol

The SD protocol is a simple command-response protocol. All commands are initiated by the master. The SD card responds to the command with a response frame and then, depending on the command, may be followed by a data token indicating the beginning of a bulk data transfer or an error condition. SD commands are issued to the card in a packed command frame, a 6-byte structure sent over the SPI port. The command frame always starts with 01 followed by the 6-bit command number. Next the 4-byte argument is sent, MSB first. The 7-bit CRC with a final stop bit '1' is sent last. All bytes of the command frame are sent over the MOSI pin MSB first. The figure shows the command frame format. The CRC is optional in SPI mode and by default CRC checking is disabled and we would not be enabling it.

Table 3.2: SD Command Format

First Byte			Bytes 2-5	Last Byte	
0	1	Command	Argument (MSB First)	CRC	1

The SD card responds to each command frame with a response. Every command has an expected response type. The type of response used for a particular command depends only on the command number, not on the content of the frame. Three response types are defined for SPI mode: R1, R2, and R3. For example, the R1 response is always 1 byte long and its MSB is always 0. The other bits in the response indicate error conditions.

Table 3.3: R1 Response Format

Bit	7	6	5	4	3	2	1	0
Field	0	Parameter Error	Address Error	Erase Seq Error	Com CRC Error	Illegal Command	Erase Reset	In Idle State

3.1.2.2 Initializing the SD Card in SPI Mode

At power-up, the SD card defaults to the proprietary SD bus protocol. To switch the card to SPI mode, the command 0 (GO_IDLE_STATE) is issued. The SD card detects SPI mode selection by observing that the card select (CS) pin is held low during the GO_IDLE_STATE command. The card responds with response format R1 (as shown above). The idle state bit is set high to signify that the card has entered idle state. The SPI clock rate must not exceed 400 KHz at this stage. Next, at least 74 clocks must be issued by the master before any attempt is made to communicate with the card. This allows the card to initialize any internal state registers before card initialization proceeds. Next, the card is reset by issuing the command CMD0 while holding the CS pin low. This both resets the card and instructs it to enter SPI mode. While the CRC, in general, is ignored in SPI mode, the very first command must be followed by a valid CRC, since the card is not yet in SPI mode. The CRC byte for a CMD0 command with a zero argument is a constant 0x95. For simplicity, this CRC byte is always sent with every command. Next, the card is continuously polled with the commands CMD55 and ACMD41 until the idle bit becomes clear, indicating that the card is fully initialized and ready to respond to general commands.

3.1.2.3 Reading the Images

The SPI mode supports single block read operations only, which will be used to read the images. The READ_SINGLE_BLOCK command with a starting byte address as the argument is to be used. This address must be aligned with the beginning of a block on the media in the valid address range of the card. The SD card then evaluates this byte address and responds back with an R1 command reply. If the read is completed from the SD media without error, a start data token is sent followed by a fixed number of data bytes. The start data token is not sent if the SD card encounters a hardware failure or media read error. Rather, an error token is sent and the data transfer is aborted.

3.1.3 Mouse driver

The PS/2 mouse interface uses a bidirectional serial protocol to transmit movement and button-position data. This driver's input is movement direction, movement speed, mouse click and information whether a mouse is connected to the PS2 port. The standard PS/2 mouse interface supports the following inputs: X (right/left) movement, Y (up/down) movement, left button, middle button, and right button. The mouse reads these inputs at a regular frequency and updates various counters and flags to reflect movement and button states. The mouse has two counters that keep track of movement: the X-movement counter and the Y-movement counter.

The standard PS/2 mouse sends movement/button information to the driver using the following 3-byte packet. The X and Y counters are 9-bit 2's complement values and each has an associated overflow flag. The movement counters represent the amount of movement that has occurred since the last movement data packet was sent to the host (i.e., they do not represent absolute positions.) The movement counters are updated when the mouse reads its input and finds movement has occurred.

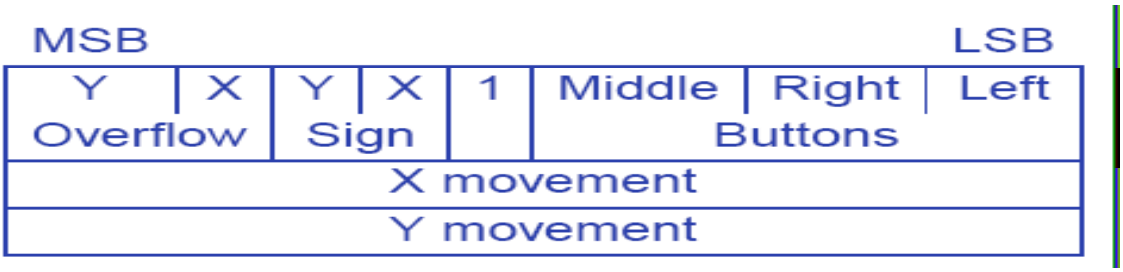


Figure 3.1: The transferring data of PS2 mouse control

Their value is the amount of movement that has occurred since the last movement data packet was sent to the host (i.e., after a packet is sent to the host, the movement counters are reset.) The range of values that can be expressed by the movement counters is -255 to +255. If this range is exceeded, the appropriate overflow bit is set.

The default mode of the mouse is the Stream Mode. However, we are using the Remote Mode of operation. Initially we tried using the mouse in the stream mode but faced some problems. So after the mouse finishes executing the reset sequence, the "Set Remote Mode (0xF0) is sent to the mouse. In this mode the mouse reads its inputs and updates its counters/flags at the current sample rate, but it does not automatically issue data packets when movement has occurred. Instead, the driver polls the mouse using the "Read Data" command. Upon receiving this command the mouse will send a single movement data packet and reset its movement counters. This gives greater control to the software to read the cursor position.

3.2 API layer

API layer gives useful toolsets or specific functionalities to the application layer. Through the API, the application layer can deals with file data, receives mouse status, decodes JPEG image and write text image on the LCD screen.

3.2.1 FAT File-system

The code to read the file system is an altered and simplified version of the FAT module that is part of FreeDos32. The first 512 bytes of the SD card make up the BIOS Parameter Block, which contains the volume information, type of FAT (FAT16 is most common for removable media), location of root directory, location of the FAT table, as well as other information. Files on the disk are broken up into chunks of Clusters (2048 bytes), which contain 8 Sectors (512 bytes) each. The File Allocation Table contains the linked list of all Clusters that make up a complete file. Since we are only interested in reading JPG files on the file system, we only implement the read functionality, and ignore Long File Names (we only read the Short File names in MSDOS 8.3 format). The root directory is read from the file system, and the JPEG files are retrieved. The functions that are used for our purpose are

3.2.2 JPEG decoder

JPEG decoder takes as its input JPEG file and outputs in raster format. Even though it would have been more efficient to be implemented in the hardware side, this decoder is placed in the software side due to the decoding algorithm's complexity. Below is a diagram describing the decoding process.

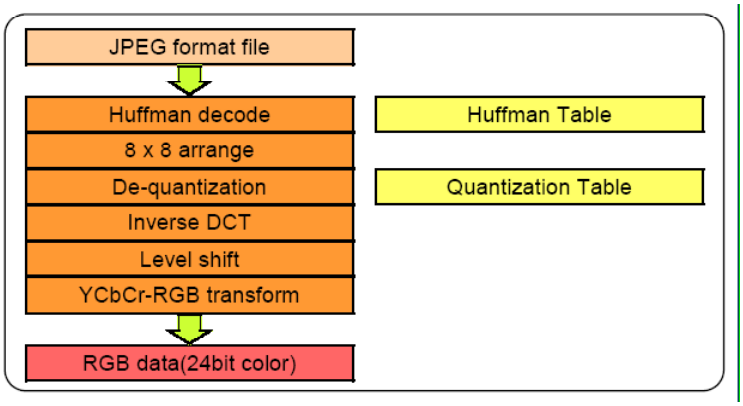


Fig3.2 Jpeg decoding Process

3.3 Application layer

Application layer has game logic. The game logic is simple source codes with deciding whether cards are matched. If they are matched, the cards are disappeared; otherwise, they are flipped back again. Finally, when all the cards are matched, we can restart the game by clicking mouse button again.

4. Who Did What

Can Ilhan:
I worked on VGA-controller component and SRAM controller in the beginning. After I implemented the basics for milestone 1, I have waited the other modules to be complete (sd-card,ps-2, interface). The animation and SRAM interface was non-trivial so after milestone 2, I have worked with my teammates to finalize VGA-controller part. I also worked on debugging software design to check vga-controller is doing its job properly or not.

Chintan Shah:
My main job was to transfer the pictures from the SD card to the SRAM. For this I took help of the sd card controller, the jpeg decoder and the file system implementation of last years imagic project. The porting of the code turned out to be more difficult than expected because we faced some issues in this. First of all, since our application was written in C++, there was some trouble in determining that in order . to combine both C and C++ code we had to define C functions as extern. Initially, we thought that it was a problem related to nios-ide. Also, the imagic code had a bug that instead of reading the boot sector and then using the pointer there to go to the partition, it was directly going to the partition table. Because of this, for each card the offset had to be manually modified. I fixed this part so that it could read from our card. Also the code was put in arranged in a much more intuitive by making classes and putting the functions in the library. After getting the pictures from the sd card, before putting in the SRAM, I implemented a randomizer function so that the cards are arranged in random locations in the screen each time. Once this was completed, I worked with my teammates in debugging problems that arised in the vga controller part and other issues in the software.

Sungjun Kim:
I worked on ps2 mount and SDRAM mount on SOPC in milestone 1. Afterwards, I finished mouse driver and designed software interface in milestone 2. Then, from milestone 3, I switched and focused my work to the hardware side to develop VGA controller. I developed image drawing logic and animation logics in milestone 3. Specifically, my work in milestone 3 was defining and implementing hardware algorithms of reading image data from SRAM, of plotting pixels on the screen, and of giving animation effects (squizing/expanding/fading-out images.) Finally, until final report, I developed game logic with my teammates and fixed VGA controller's bugs.

Zenan Li:
Initially I did the SOPC system generation and configuration in the milestone1. After that, I worked with

my teammates to complete the SDRAM interfacing in milestone2. In the milestone3, when we were working on the animation issues, we were not getting a smooth transition because of problems in the offset table in the expanding and squeezing animation. After careful design of the offset table, I optimized it so that the animation works pretty well and looks highly natural. And in the final part, I implemented some parts of the game logic with my teammates in the software side and tested the whole system to ensure that it works perfectly.

5. Learnings and advice for future students

Over the course of the project, we learnt many things and had some experiences that we would like to share with future students. It is best if the group can have a clear vision of all the aspects of the project from the beginning, because in that case the project design can be easily drawn and separated into independent modules. These can then be divided among the team members. In this way the project can progress efficiently. Initially, we did not have such a clear view as to how we will implement the project. Also, each of the team members had his own opinion of how it could be done. We made the mistake that we did not write down the outputs of our meetings. Because of this, each time we sat down to discuss we had to start from the beginning. So we suggest that during the initial stages, after each meeting the team should write down the summary of the points discussed so that they could proceed from there the next time. Because of our initial mistakes, we could not distribute the work efficiently and this resulted in some delays. This resulted in some situation to case similar to extreme programming where one would write the code and the others would sit around and debug the code. While sometimes useful, overall we think the traditional approach of dividing the project into modules is more efficient. One of our team members is of the view that teams should appoint a team leader who can coordinate the activities of other members. Another important decision is to decide what part of the project should be completed in which milestone. The distribution should be even so that the team is not stretched in anyone of the milestones and it is even better if most of the work can be done in the earlier milestones. Also a key decision in an embedded system project is to decide which components and functionality will be implemented in hardware and which will be implemented in software. This should generally depend on the type of project that one is doing. Hardware is faster than software but at the same time more difficult to implement and debug. For example, in our project we could have implemented the SD card control and the JPEG decoder in hardware instead of software. This would have resulted in much improved performance but with a corresponding increase in complexity. As our application was not so sensitive to timing and speed issues we implemented those in software. But at the same time animation was implemented in hardware considering the above.

6. Code Listing

6.1 VHDL Code

```
--
-- DE2 top-level module that includes the simple VGA raster generator
--
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu
--
-- From an original by Terasic Technology, Inc.
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pelmanism is

    port (
        -- Clocks

        CLOCK_27,                                                       -- 27 MHz
        CLOCK_50,                                                       -- 50 MHz
        EXT_CLOCK : in std_logic;                                       -- External Clock

        -- Buttons and switches

--      KEY : in std_logic_vector(3 downto 0);                  -- Push buttons
--      SW : in std_logic_vector(17 downto 0);                  -- DPDT switches

        -- LED displays

        HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7 -- 7-segment displays
            : out std_logic_vector(6 downto 0);
--      LEDG : out std_logic_vector(8 downto 0);                -- Green LEDs
--      signal LEDR : out std_logic_vector(17 downto 0);            -- Red LEDs

        -- RS-232 interface
C/C
--      UART_TXD : out std_logic;                                       -- UART transmitter
--      UART_RXD : in std_logic;                                        -- UART receiver

        -- IRDA interface

--      IRDA_TXD : out std_logic;                                       -- IRDA Transmitter
--      IRDA_RXD : in std_logic;                                        -- IRDA Receiver

        -- SDRAM

        DRAM_DQ : inout std_logic_vector(15 downto 0); -- Data Bus
        DRAM_ADDR : out std_logic_vector(11 downto 0); -- Address Bus
        DRAM_LDQM,                                                              -- Low-byte Data Mask
        DRAM_UDQM,                                                              -- High-byte Data Mask
        DRAM_WE_N,                                                              -- Write Enable
        DRAM_CAS_N,                                                             -- Column Address Strobe
        DRAM_RAS_N,                                                             -- Row Address Strobe
        DRAM_CS_N,                                                              -- Chip Select
        DRAM_BA_0,                                                              -- Bank Address 0
        DRAM_BA_1,                                                              -- Bank Address 0
        DRAM_CLK,                                                               -- Clock
        DRAM_CKE : out std_logic;                                       -- Clock Enable

--      -- FLASH
--
--      FL_DQ : inout std_logic_vector(7 downto 0);             -- Data bus
--      FL_ADDR : out std_logic_vector(21 downto 0);    -- Address bus
--      FL_WE_N,                                                        -- Write Enable
--      FL_RST_N,                                                       -- Reset
--      FL_OE_N,                                                        -- Output Enable
--      FL_CE_N : out std_logic;                                    -- Chip Enable

        -- SRAM

        SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
        SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
        SRAM_UB_N,                                                              -- High-byte Data Mask
        SRAM_LB_N,                                                              -- Low-byte Data Mask
        SRAM_WE_N,                                                              -- Write Enable
        SRAM_CE_N,                                                              -- Chip Enable
        SRAM_OE_N : out std_logic;                                      -- Output Enable
        -- USB controller

--      OTG_DATA : inout std_logic_vector(15 downto 0); -- Data bus
--      OTG_ADDR : out std_logic_vector(1 downto 0);        -- Address
--      OTG_CS_N,                                                       -- Chip Select
--      OTG_RD_N,                                                       -- Write
--      OTG_WR_N,                                                       -- Read
--      OTG_RST_N,                                                      -- Reset
--      OTG_FSPEED,                                 -- USB Full Speed, 0 = Enable, Z = Disable
--      OTG_LSPEED : out std_logic;         -- USB Low Speed, 0 = Enable, Z = Disable
--      OTG_INT0,                                                       -- Interrupt 0
```

```
--      OTG_INT1,                                                   -- Interrupt 1
--      OTG_DREQ0,                                                  -- DMA Request 0
--      OTG_DREQ1 : in std_logic;                                   -- DMA Request 1
--      OTG_DACK0_N,                                                -- DMA Acknowledge 0
--      OTG_DACK1_N : out std_logic;                                -- DMA Acknowledge 1

        -- 16 X 2 LCD Module
--
        LCD_ON,                                 -- Power ON/OFF
        LCD_BLON,                               -- Back Light ON/OFF
        LCD_RW,                                 -- Read/Write Select, 0 = Write, 1 = Read
        LCD_E,                                  -- Enable
        LCD_RS : out std_logic;         -- Command/Data Select, 0 = Command, 1 = Data
        LCD_DATA : inout std_logic_vector(7 downto 0); -- Data bus 8 bits

        -- SD card interface

        SD_DAT,                                 -- SD Card Data
        SD_DAT3,                                -- SD Card Data 3
        SD_CMD : inout std_logic;       -- SD Card Command Signal
        SD_CLK : out std_logic;         -- SD Card Clock

        -- USB JTAG link

        TDI,                                    -- CPLD -> FPGA (data in)
        TCK,                                    -- CPLD -> FPGA (clk)
        TCS : in std_logic;                 -- CPLD -> FPGA (CS)
        TDO : out std_logic;                -- FPGA -> CPLD (data out)

        -- I2C bus

--      I2C_SDAT : inout std_logic; -- I2C Data
--      I2C_SCLK : out std_logic;       -- I2C Clock

        -- PS/2 port

        PS2_DAT,                                -- Data
        PS2_CLK : inout std_logic;          -- Clock

        -- VGA output

        VGA_CLK,                                                    -- Clock
        VGA_HS,                                                     -- H_SYNC
        VGA_VS,                                                     -- V_SYNC
        VGA_BLANK,                                                  -- BLANK
        VGA_SYNC : out std_logic;                                   -- SYNC
        VGA_R,                                                      -- Red[9:0]
        VGA_G,                                                      -- Green[9:0]
        VGA_B : out std_logic_vector (9 downto 0)                   -- Blue[9:0]

        --    Ethernet Interface
--
--      ENET_DATA : inout std_logic_vector(15 downto 0);        -- DATA bus 16Bits
--      ENET_CMD,                   -- Command/Data Select, 0 = Command, 1 = Data
--      ENET_CS_N,                                                  -- Chip Select
--      ENET_WR_N,                                                  -- Write
--      ENET_RD_N,                                                  -- Read
--      ENET_RST_N,                                                 -- Reset
--      ENET_CLK : out std_logic;                                   -- Clock 25 MHz
--      ENET_INT : in std_logic;                                    -- Interrupt

        -- Audio CODEC

--      AUD_ADCLRCK : inout std_logic;                              -- ADC LR Clock
--      AUD_ADCDAT : in std_logic;                                  -- ADC Data
--      AUD_DACLRCK : inout std_logic;                              -- DAC LR Clock
--      AUD_DACDAT : out std_logic;                                 -- DAC Data
--      AUD_BCLK : inout std_logic;                                 -- Bit-Stream Clock
--      AUD_XCK : out std_logic;                                    -- Chip Clock
--
        -- Video Decoder

--      TD_DATA : in std_logic_vector(7 downto 0);    -- Data bus 8 bits
--      TD_HS,                                                      -- H_SYNC
--      TD_VS : in std_logic;                                   -- V_SYNC
--      TD_RESET : out std_logic;                               -- Reset

        -- General-purpose I/O

--      GPIO_0,                                                     -- GPIO Connection 0
--      GPIO_1 : inout std_logic_vector(35 downto 0) -- GPIO Connection 1

);

end pelmanism;

architecture datapath of pelmanism is
   signal counter : unsigned(15 downto 0);
   signal reset_n : std_logic;

        signal sram_addr_vga : std_logic_vector( 17 downto 0 ) := "00" & X"0000";
        signal sram_ce_n_vga : std_logic := '0';
        signal sram_dq_vga : std_logic_vector( 15 downto 0 ) := X"0000";
        signal sram_lb_n_vga : std_logic := '0';
        signal sram_ub_n_vga : std_logic := '0';
        signal sram_oe_n_vga : std_logic := '0';
        signal sram_we_n_vga : std_logic := '0';
        signal sram_occupy_vga : std_logic := '0';
```

```
        signal sram_addr_sram : std_logic_vector( 17 downto 0 ) := "00" & X"0000";
        signal sram_ce_n_sram : std_logic := '0';
        signal sram_dq_sram : std_logic_vector( 15 downto 0 ) := X"0000";
        signal sram_lb_n_sram : std_logic := '0';
        signal sram_ub_n_sram : std_logic := '0';
        signal sram_oe_n_sram : std_logic := '0';
        signal sram_we_n_sram : std_logic := '0';

begin

SRAM_ADDR <= sram_addr_vga when sram_occupy_vga = '1' else sram_addr_sram;
SRAM_CE_N <= sram_ce_n_vga when sram_occupy_vga = '1' else sram_ce_n_sram;
sram_dq_vga <= SRAM_DQ when sram_occupy_vga = '1' else ( others => 'Z' );
SRAM_DQ <= sram_dq_sram when sram_occupy_vga = '0' else ( others => 'Z' );
SRAM_LB_N <= sram_lb_n_vga when sram_occupy_vga = '1' else sram_lb_n_sram;
SRAM_OE_N <= sram_oe_n_vga when sram_occupy_vga = '1' else sram_oe_n_sram;
SRAM_UB_N <= sram_ub_n_vga when sram_occupy_vga = '1' else sram_ub_n_sram;
SRAM_WE_N <= sram_we_n_vga when sram_occupy_vga = '1' else sram_we_n_sram;

process (CLOCK_50)
   begin
      if rising_edge(CLOCK_50) then
         if counter = x"ffff" then
            reset_n <= '1';
         else
            reset_n <= '0';
            counter <= counter + 1;
         end if;
      end if;
   end process;

   pll: entity work.sdram_pll port map(
        inclk0 => CLOCK_50,
        c0 => DRAM_CLK
   );

   nios: entity work.nios_system port map (
      clk                                          => CLOCK_50,
      reset_n                                   => reset_n,
      SRAM_ADDR_from_the_vga          => sram_addr_vga,
      SRAM_CE_N_from_the_vga          => sram_ce_n_vga,
      SRAM_DQ_to_and_from_the_vga => sram_dq_vga,
      SRAM_LB_N_from_the_vga          => sram_lb_n_vga,
      SRAM_OE_N_from_the_vga          => sram_oe_n_vga,
      SRAM_UB_N_from_the_vga          => sram_ub_n_vga,
      SRAM_WE_N_from_the_vga          => sram_we_n_vga,
                SRAM_OCCUPY_from_the_vga          => sram_occupy_vga,
      SRAM_ADDR_from_the_sram          => sram_addr_sram,
      SRAM_CE_N_from_the_sram          => sram_ce_n_sram,
      SRAM_DQ_to_and_from_the_sram => sram_dq_sram,
      SRAM_LB_N_from_the_sram          => sram_lb_n_sram,
      SRAM_OE_N_from_the_sram          => sram_oe_n_sram,
      SRAM_UB_N_from_the_sram          => sram_ub_n_sram,
      SRAM_WE_N_from_the_sram          => sram_we_n_sram,

--       SRAM_ADDR_from_the_sram          => SRAM_ADDR,
--       SRAM_CE_N_from_the_sram          => SRAM_CE_N,
--       SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
--       SRAM_LB_N_from_the_sram          => SRAM_LB_N,
--       SRAM_OE_N_from_the_sram          => SRAM_OE_N,
--       SRAM_UB_N_from_the_sram          => SRAM_UB_N,
--       SRAM_WE_N_from_the_sram          => SRAM_WE_N,


--clk_to_the_vga                          => CLOCK_50,
        --reset_to_the_vga                          => '0',
      VGA_CLK_from_the_vga          => VGA_CLK,
      VGA_HS_from_the_vga          => VGA_HS,
      VGA_VS_from_the_vga          => VGA_VS,
      VGA_BLANK_from_the_vga                => VGA_BLANK,
      VGA_SYNC_from_the_vga          => VGA_SYNC,
      VGA_R_from_the_vga                    => VGA_R,
      VGA_G_from_the_vga                    => VGA_G,
      VGA_B_from_the_vga                    => VGA_B,
        PS2_CLK_to_and_from_the_ps2=> PS2_CLK,
        PS2_DAT_to_and_from_the_ps2          => PS2_DAT,
        zs_addr_from_the_sdram                => DRAM_ADDR,
        zs_ba_from_the_sdram( 0 )       => DRAM_BA_1,
        zs_ba_from_the_sdram( 1 )       => DRAM_BA_0,
        zs_cas_n_from_the_sdram        => DRAM_CAS_N,
        zs_cke_from_the_sdram          => DRAM_CKE,
        zs_cs_n_from_the_sdram                => DRAM_CS_N,
        zs_dq_to_and_from_the_sdram => DRAM_DQ,
        zs_dqm_from_the_sdram( 0 )    => DRAM_UDQM,
        zs_dqm_from_the_sdram( 1 )    => DRAM_LDQM,
        zs_ras_n_from_the_sdram        => DRAM_RAS_N,
        zs_we_n_from_the_sdram                => DRAM_WE_N,
        CARD_CLK_from_the_sd_card_clk => SD_CLK,
        CARD_CMD_from_the_sd_card_datain => SD_CMD,
        CARD_DAT_to_the_sd_card_dataout => SD_DAT,
        CARD_SD3_from_the_sd_card_ncs => SD_DAT3,
        LCD_RW_from_the_LCD => LCD_RW,
        LCD_E_from_the_LCD => LCD_E,
        LCD_data_to_and_from_the_LCD => LCD_DATA,
        LCD_RS_from_the_LCD => LCD_RS
      );

   HEX7        <= "0001001"; -- Leftmost
   HEX6        <= "0000110";
```

```
    HEX5        <= "1000111";
    HEX4        <= "1000111";
    HEX3        <= "1000000";
    HEX2        <= (others => '1');
    HEX1        <= (others => '1');
    HEX0        <= (others => '1');                 -- Rightmost

        LCD_ON <= '1';
        LCD_BLON <= '1';

--
--    SD_DAT3 <= '1';
--    SD_CMD <= '1';
--    SD_CLK <= '1';

--    SRAM_DQ <= (others => 'Z');
--    SRAM_ADDR <= (others => '0');
--    SRAM_UB_N <= '1';
--    SRAM_LB_N <= '1';
--    SRAM_CE_N <= '1';
--    SRAM_WE_N <= '1';
--    SRAM_OE_N <= '1';

--    UART_TXD <= '0';
--    DRAM_ADDR <= (others => '0');
--    DRAM_LDQM <= '0';
--    DRAM_UDQM <= '0';
--    DRAM_WE_N <= '1';
--    DRAM_CAS_N <= '1';
--    DRAM_RAS_N <= '1';
--    DRAM_CS_N <= '1';
--    DRAM_BA_0 <= '0';
--    DRAM_BA_1 <= '0';
--    DRAM_CLK <= '0';
--    DRAM_CKE <= '0';
--    FL_ADDR <= (others => '0');
--    FL_WE_N <= '1';
--    FL_RST_N <= '0';
--    FL_OE_N <= '1';
--    FL_CE_N <= '1';
--    OTG_ADDR <= (others => '0');
--    OTG_CS_N <= '1';
--    OTG_RD_N <= '1';
--    OTG_RD_N <= '1';
--    OTG_WR_N <= '1';
--    OTG_RST_N <= '1';
--    OTG_FSPEED <= '1';
--    OTG_LSPEED <= '1';
--    OTG_DACK0_N <= '1';
--    OTG_DACK1_N <= '1';
--
--    TDO <= '0';

--    ENET_CMD <= '0';
--    ENET_CS_N <= '1';
--    ENET_WR_N <= '1';
--    ENET_RD_N <= '1';
--    ENET_RST_N <= '1';
--    ENET_CLK <= '0';

--    TD_RESET <= '0';
--
--    I2C_SCLK <= '1';
--
--    AUD_DACDAT <= '1';
--    AUD_XCK <= '1';
--
   -- Set all bidirectional ports to tri-state
--    DRAM_DQ          <= (others => 'Z');
--    FL_DQ            <= (others => 'Z');
--    SRAM_DQ          <= (others => 'Z');
--    OTG_DATA         <= (others => 'Z');
      LCD_DATA         <= (others => 'Z');
--    SD_DAT           <= 'Z';
--    I2C_SDAT         <= 'Z';
----    ENET_DATA       <= (others => 'Z');
--    AUD_ADCLRCK <= 'Z';
--    AUD_DACLRCK <= 'Z';
--    AUD_BCLK         <= 'Z';
--    GPIO_0           <= (others => 'Z');
--    GPIO_1           <= (others => 'Z');

end datapath;
--------------------------------------------------------------------------------
--
-- VGA Controller
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity vga_controller is

    port (
        reset               : in std_logic;
        clk                 : in std_logic;                             -- Should be 25.125 MHz
        read                : in    std_logic;
```

```
    write           : in    std_logic;
    chipselect      : in    std_logic;
      waitrequest : out std_logic;
    address         : in    unsigned(7 downto 0);
    readdata        : out unsigned(31 downto 0);
    writedata       : in    unsigned(31 downto 0); --doubled to 32bit bus

    VGA_CLK,                                    -- Clock
    VGA_HS,                                     -- H_SYNC
    VGA_VS,                                     -- V_SYNC
    VGA_BLANK,                                   -- BLANK
    VGA_SYNC     : out std_logic;              -- SYNC
    VGA_R,                                      -- Red[9:0]
    VGA_G,                                       -- Green[9:0]
    VGA_B                   : out std_logic_vector (9 downto 0); -- Blue[9:0]

      --    Sram Module :
      SRAM_DQ                       : inout unsigned(15 downto 0);
      SRAM_ADDR                 : out unsigned(17 downto 0);
      SRAM_UB_N, SRAM_LB_N : out std_logic;
      SRAM_WE_N, SRAM_CE_N : out std_logic;
      SRAM_OE_N                 : out std_logic;
      SRAM_OCCUPY                                   : out std_logic
    );

end vga_controller;

architecture rtl of vga_controller is

  -- video constants
      constant HTOTAL                       : integer := 800;
      constant HSYNC                        : integer := 96;
      constant HBACK_PORCH                          : integer := 48;
      constant HACTIVE                      : integer := 640;
      constant HFRONT_PORCH                         : integer := 16;
      constant HACTIVE_CNT                  : integer := HSYNC + HBACK_PORCH;

      constant VTOTAL                       : integer := 525;
      constant VSYNC                        : integer := 2;
      constant VBACK_PORCH                          : integer := 33;
      constant VACTIVE                      : integer := 480;
      constant VFRONT_PORCH                         : integer := 10;
      constant VACTIVE_CNT                  : integer := VSYNC + VBACK_PORCH;

      -- sprite constants
      constant N_SPRITE_ANIMATION       : integer := 16;
      constant N_SPRITE_CURSOR          : integer := 1;
      constant N_SPRITE_TOTAL                   : integer := N_SPRITE_ANIMATION + N_SPRITE_CURSOR;
      constant SPRITE_ANIMATION_WIDTH  : integer := 128;
      constant SPRITE_ANIMATION_HEIGHT: integer := 64;
      constant SPRITE_CURSOR_WIDTH      : integer := 9;
      constant SPRITE_CURSOR_HEIGHT     : integer := 12;
      constant Border_VStart                : integer := 336;
      constant Border_Width                 : integer := 5;

      -- screen constants
      constant MARGIN_LEFTMOST          : integer := 0;
      constant MARGIN_RIGHTMOST         : integer := 0;
      constant MARGIN_UPPEREND          : integer := 0;
      constant MARGIN_LOWEREND          : integer := 0;

      -- color constants
      constant COLOR_BIT_ALPHA          : integer := 15;
      constant COLOR_R_START                    : integer := 10;
      constant COLOR_R_END                  : integer := 14;
      constant COLOR_G_START                    : integer := 5;
      constant COLOR_G_END                  : integer := 9;
      constant COLOR_B_START                    : integer := 0;
      constant COLOR_B_END                  : integer := 4;

      -- color data
      signal cursor_color_out                   : unsigned( 15 downto 0 ) := X"0000";
      signal color_out                          : unsigned( 15 downto 0 ) := X"0000";
      signal is_fading_out                  : std_logic := '0';
      signal is_gading_in                   : std_logic := '0';
      signal current_frame                  : integer := 0;

      -- cursor sprite
      signal is_cursor_drawing          : std_logic := '0';
      signal cursor_pos_x                       : unsigned( 9 downto 0 ) := "0001000000";           -- 0 ~ 640
      signal cursor_pos_y                       : unsigned( 9 downto 0 ) := "0011000000";           -- 0 ~ 480

      ------------------------------------------------------------------------------------------

  type HStartMatrix is array (0 to 3) of unsigned(9 downto 0);
  signal Card_HStart : HStartMatrix := ("0000000000","0010101010","0101010100","0111111110");
  type VStartMatrix is array (0 to 3) of integer;
  signal Card_VStart : VStartMatrix := ( 45, 154, 263, 372 );

  -- Signals for the video controller
  signal Hcount : unsigned(9 downto 0);    -- Horizontal position (0-800)
  signal Vcount : unsigned(9 downto 0);    -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;
  signal vga_hblank, vga_hsync, vga_vblank, vga_vsync : std_logic;    -- Sync. signals

  signal pic0 : std_logic := '0'; signal pic1 : std_logic := '0'; signal pic2 : std_logic := '0'; signal pic3 : std_logic := '0';
  signal pic4 : std_logic := '0'; signal pic5 : std_logic := '0'; signal pic6 : std_logic := '0'; signal pic7 : std_logic := '0';
  signal pic8 : std_logic := '0'; signal pic9 : std_logic := '0'; signal pic10 : std_logic := '0'; signal pic11 : std_logic := '0';
  signal pic12 : std_logic := '0'; signal pic13 : std_logic := '0'; signal pic14 : std_logic := '0'; signal pic15 : std_logic := '0';
```

```
    signal border : std_logic := '0';
    signal clk_25 : std_logic := '0';

        signal sprite_status   : unsigned(31 downto 0) := X"00000000";

        -- animation status
        type TYPE_ANIMATION_STATUS is array( 0 to 15 ) of unsigned( 31 downto 0 );
        signal animation_status : TYPE_ANIMATION_STATUS := ( others => X"00000000" );

-- sram signals
signal ram_address          : unsigned( 17 downto 0 )                   := "00" & X"0000";
signal ram_data                         : unsigned( 15 downto 0 )               := X"0000";
signal ram_rw_status        : std_logic                                                 := '0';


begin
    --Feeding the Sram control signals
    ram_data <= SRAM_DQ;
    SRAM_ADDR <= ram_address;
    SRAM_WE_N <= '1';
        SRAM_OE_N <= '0';
    SRAM_UB_N <= '0';
        SRAM_LB_N <= '0';
        SRAM_CE_N <= '0';
        SRAM_OCCUPY <= not ram_rw_status;

-- Halving the clock that is fed to this component since VGA supports 25Mhz instead of 50Mhz
    process (clk)
    begin
       if rising_edge(clk) then
          clk_25 <= not clk_25;
       end if;
    end process;

RECV_SPRITE_ANIMATION : process(clk)
begin
        if rising_edge(clk) then
                if chipselect = '1' and write = '1' then
                        case address is
                                when X"04" => animation_status( 0 ) <= writedata;
                                when X"05" => animation_status( 1 ) <= writedata;
                                when X"06" => animation_status( 2 ) <= writedata;
                                when X"07" => animation_status( 3 ) <= writedata;
                                when X"08" => animation_status( 4 ) <= writedata;
                                when X"09" => animation_status( 5 ) <= writedata;
                                when X"0A" => animation_status( 6 ) <= writedata;
                                when X"0B" => animation_status( 7 ) <= writedata;
                                when X"0C" => animation_status( 8 ) <= writedata;
                                when X"0D" => animation_status( 9 ) <= writedata;
                                when X"0E" => animation_status( 10 ) <= writedata;
                                when X"0F" => animation_status( 11 ) <= writedata;
                                when X"10" => animation_status( 12 ) <= writedata;
                                when X"11" => animation_status( 13 ) <= writedata;
                                when X"12" => animation_status( 14 ) <= writedata;
                                when X"13" => animation_status( 15 ) <= writedata;
                                when others => animation_status( 15 ) <= animation_status( 15 );
                        end case;
                end if;
        end if;
end process RECV_SPRITE_ANIMATION;

RECV_SRAM_STATUS : process (clk)
    begin
        if rising_edge(clk) then
                if chipselect = '1' then
                        if write = '1' then
                                if address = X"02" then -- address reserved for sram r/w
                                        ram_rw_status <= std_logic( writedata( 0 ) );
                                end if;
                        end if;
                end if;
        end if;
end process RECV_SRAM_STATUS;

RECV_SPRITE_STATUS : process (clk)
begin
        if rising_edge(clk) then
                if chipselect = '1' then
                        if write = '1' then
                                if address = X"03" then --address reserved for clicked image array
                                        sprite_status <= writedata(31 downto 0);
                                end if;
                        end if;
                end if;
        end if;
end process RECV_SPRITE_STATUS;

-- Getting the X and Y positions for cursor from Nios2 and writing those coordinates into 2 predefined registers
-- namely cursor_pos_x and cursor_pos_y. Assuming the first 10 bits have cursor_pos_X data and
-- the second 10 bits are holding cursor_pos_Y data
    GetCursorPos : process (clk)
    begin
        if rising_edge(clk) then
                if chipselect = '1' then
                        if write = '1' then
                                if address = X"00" then
                                        cursor_pos_x <= writedata(9 downto 0);
                                        cursor_pos_y <= writedata(19 downto 10);
                                end if;
```

```
                    end if;
                end if;
            end if;
end process GetCursorPos;

-- Horizontal and vertical counters
HCounter : process (clk_25)
begin
   if rising_edge(clk_25) then
      if reset = '1' then
         Hcount <= (others => '0');
      elsif EndOfLine = '1' then
         Hcount <= (others => '0');
      else
         Hcount <= Hcount + 1;
      end if;
   end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (clk_25)
begin
   if rising_edge(clk_25) then
      if reset = '1' then
         Vcount <= (others => '0');
      elsif EndOfLine = '1' then
         if EndOfField = '1' then
            Vcount <= (others => '0');
         else
            Vcount <= Vcount + 1;
         end if;
      end if;
   end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk_25)
begin
   if rising_edge(clk_25) then
      if reset = '1' or EndOfLine = '1' then
         vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
         vga_hsync <= '0';
      end if;
   end if;
end process HSyncGen;

HBlankGen : process (clk_25)
begin
   if rising_edge(clk_25) then
      if reset = '1' then
         vga_hblank <= '1';
      elsif Hcount = HSYNC + HBACK_PORCH then
         vga_hblank <= '0';
      elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
         vga_hblank <= '1';
      end if;
   end if;
end process HBlankGen;

VSyncGen : process (clk_25)
begin
   if rising_edge(clk_25) then
      if reset = '1' then
         vga_vsync <= '1';
      elsif EndOfLine ='1' then
         if EndOfField = '1' then
            vga_vsync <= '1';
         elsif Vcount = VSYNC - 1 then
            vga_vsync <= '0';
         end if;
      end if;
   end if;
end process VSyncGen;

VBlankGen : process (clk_25)
begin
   if rising_edge(clk_25) then
      if reset = '1' then
         vga_vblank <= '1';
      elsif EndOfLine = '1' then
         if Vcount = VSYNC + VBACK_PORCH - 1 then
            vga_vblank <= '0';
         elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
            vga_vblank <= '1';
         end if;
      end if;
   end if;
end process VBlankGen;

      -- Cursor Sprite Generation
      CursorGen : process (clk_25)
      constant cur_inner_color : unsigned( 15 downto 0 ) := X"C86C";
      type TYPE_CURSOR_H_COLOR is array( 0 to ( SPRITE_CURSOR_WIDTH - 1 ) ) of unsigned( 15 downto 0 );
      variable cursor_h_color_0 : TYPE_CURSOR_H_COLOR := ( X"FFFF", X"0000", X"0000", X"0000", X"0000", X"0000", X"0000", X"0000",
```

```
X"0000" );
    variable cursor_h_color_1 : TYPE_CURSOR_H_COLOR := ( X"FFFF", X"FFFF", X"0000", X"0000", X"0000", X"0000", X"0000", X"0000",
X"0000" );
    variable cursor_h_color_2 : TYPE_CURSOR_H_COLOR := ( X"FFFF", cur_inner_color, X"FFFF", X"0000", X"0000", X"0000", X"0000",
X"0000", X"0000" );
    variable cursor_h_color_3 : TYPE_CURSOR_H_COLOR := ( X"FFFF", cur_inner_color, cur_inner_color, X"FFFF", X"0000", X"0000",
X"0000", X"0000", X"0000" );
    variable cursor_h_color_4 : TYPE_CURSOR_H_COLOR := ( X"FFFF", cur_inner_color, cur_inner_color, cur_inner_color, X"FFFF",
X"0000", X"0000", X"0000", X"0000" );
    variable cursor_h_color_5 : TYPE_CURSOR_H_COLOR := ( X"FFFF", cur_inner_color, cur_inner_color, cur_inner_color,
cur_inner_color, X"FFFF", X"0000", X"0000", X"0000" );
    variable cursor_h_color_6 : TYPE_CURSOR_H_COLOR := ( X"FFFF", cur_inner_color, cur_inner_color, cur_inner_color,
cur_inner_color, cur_inner_color, X"FFFF", X"0000", X"0000" );
    variable cursor_h_color_7 : TYPE_CURSOR_H_COLOR := ( X"FFFF", X"FFFF", X"FFFF", cur_inner_color, X"FFFF", X"FFFF", X"FFFF",
X"0000", X"0000" );
    variable cursor_h_color_8 : TYPE_CURSOR_H_COLOR := ( X"0000", X"0000", X"FFFF", cur_inner_color, X"FFFF", X"0000", X"0000",
X"0000", X"0000" );
    variable cursor_h_color_9 : TYPE_CURSOR_H_COLOR := ( X"0000", X"0000", X"0000", X"FFFF", cur_inner_color, X"FFFF", X"0000",
X"0000", X"0000" );
    variable cursor_h_color_10 : TYPE_CURSOR_H_COLOR := ( X"0000", X"0000", X"0000", X"FFFF", cur_inner_color, X"FFFF", X"0000",
X"0000", X"0000" );
    variable cursor_h_color_11 : TYPE_CURSOR_H_COLOR := ( X"0000", X"0000", X"0000", X"0000", X"FFFF", X"FFFF", X"0000", X"0000",
X"0000" );

    variable cursor_h_color_cur : TYPE_CURSOR_H_COLOR := ( others => X"0000" );
    variable cursor_pixel_color_cur : unsigned( 15 downto 0 ) := X"0000";

    begin
    if rising_edge(clk_25) then
        if reset = '1' then
                is_cursor_drawing <= '0';
        -- check whether the current horizontal pixel is pointing cursor area
        elsif ( HACTIVE_CNT + cursor_pos_x - 1 < Hcount and Hcount < HACTIVE_CNT + cursor_pos_x + SPRITE_CURSOR_WIDTH ) and
                ( VACTIVE_CNT + cursor_pos_y - 1 < Vcount and Vcount < VACTIVE_CNT + cursor_pos_y +
SPRITE_CURSOR_HEIGHT ) then

            -- get current cursor line
            case( to_integer( Vcount - VACTIVE_CNT - cursor_pos_y ) ) is
                when 0 => cursor_h_color_cur := cursor_h_color_0;
                when 1 => cursor_h_color_cur := cursor_h_color_1;
                when 2 => cursor_h_color_cur := cursor_h_color_2;
                when 3 => cursor_h_color_cur := cursor_h_color_3;
                when 4 => cursor_h_color_cur := cursor_h_color_4;
                when 5 => cursor_h_color_cur := cursor_h_color_5;
                when 6 => cursor_h_color_cur := cursor_h_color_6;
                when 7 => cursor_h_color_cur := cursor_h_color_7;
                when 8 => cursor_h_color_cur := cursor_h_color_8;
                when 9 => cursor_h_color_cur := cursor_h_color_9;
                when 10 => cursor_h_color_cur := cursor_h_color_10;
                when 11 => cursor_h_color_cur := cursor_h_color_11;
                when others => cursor_h_color_cur := ( others => X"0000" );
            end case;

            -- get current pixel
            cursor_pixel_color_cur := cursor_h_color_cur( to_integer( Hcount - HACTIVE_CNT - cursor_pos_x ) );

            -- set alpha value and color
            is_cursor_drawing <= cursor_pixel_color_cur( COLOR_BIT_ALPHA );
            cursor_color_out <= cursor_pixel_color_cur;
        else
            is_cursor_drawing <= '0';
        end if;
    end if;
  end process CursorGen;

  -- Sprite Generation
    SpriteGen : process (clk_25)
    type TYPE_OFFSET_TABLE is array( integer range 0 to 29, integer range 0 to 127 ) of integer;
    constant offset_table_squizing : TYPE_OFFSET_TABLE :=
(
( 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2 ),
( 255, 255, 255, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2,
2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2,
4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2,
2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 255, 255, 255,
255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2,
2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2,
2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2,
2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2,
2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 255, 255, 255, 255, 255, 255,
255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2,
2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2,
4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4,
2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2,
2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 2, 2, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
```

```
255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255,   2,   2,   2,   2,   2,   4,   2,   2,   4,   2,   2,
4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,
2,   2,   4,   2,   2,   4,   2,   2,   2,   2,   4,   2,   2,   2,   2,   2,   4,   2,   4,   2,   2,   4,   2,   2,   4,
2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   4,   2,
2,   4,   2,   2,   4,   2,   2,   4,   2,   2,   2,   2, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255,   2,   2,   2,   2,   4,   2,   4,   2,
2,   4,   2,   4,   2,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,
2,   4,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,   2,   2,   4,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,
4,   2,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,   2,   4,   2,   4,   2,   2,   4,   2,
4,   2,   2,   4,   2,   4,   2,   2,   2,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255
),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255,   2,   2,   4,   2,   4,
2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,
4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   2,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,
2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,   4,   2,
4,   2,   4,   2,   4,   2, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255,   2,   4,   4,
2,   4,   2,   4,   4,   2,   4,   2,   4,   4,   2,   4,   4,   2,   4,   2,   4,   4,   2,   4,   2,   4,   4,   2,   4,
2,   4,   4,   2,   4,   2,   4,   4,   2,   4,   2,   2,   4,   2,   2,   4,   2,   4,   4,   2,   4,   2,   4,   4,   2,
4,   2,   4,   4,   2,   4,   2,   4,   4,   2,   4,   2,   4,   4,   2,   4,   4,   2,   4,   2,   4,   4,   2,   4,   2,
4,   4, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255,   2,
4,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,
4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,
2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   2,   4,   4,   4,
255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),

( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255,   2,   4,   4,   4,   2,   4,   4,   4,   4,   2,   4,   4,   4,   2,   4,   4,   4,   4,   2,   4,   4,   4,   2,
4,   4,   4,   4,   2,   4,   4,   4,   2,   4,   4,   2,   4,   4,   4,   2,   4,   4,   2,   4,   4,   4,   2,   4,   4,
4,   4,   2,   4,   4,   4,   2,   4,   4,   4,   4,   2,   4,   4,   4,   2,   4,   4,   4,   4,   2,   4,   4,   4,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   2,   4,   4,   4,   4,   4,   2,   4,   4,   4,   4,   4,   4,   2,   4,   4,   4,   4,   4,   4,   2,
4,   4,   4,   4,   4,   4,   2,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   2,   4,   4,   4,   4,
4,   4,   2,   4,   4,   4,   4,   4,   4,   2,   4,   4,   4,   4,   4,   4,   2,   4,   4,   4,   4,   4, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255,   2,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,
4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,
4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4,   4, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255,   2,   4,   4,   4,   4,   4,   4,   4,   6,   4,   4,   4,   4,   4,   4,   6,   4,   4,
4,   4,   4,   6,   4,   4,   4,   4,   4,   6,   4,   4,   4,   4,   4,   6,   4,   4,   4,   4,   4,   6,   4,   4,   4,
4,   4,   6,   4,   4,   4,   4,   4,   4,   6,   4,   4,   4,   4,   4,   4,   4, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255,   2,   4,   4,   6,   6,   4,   6,   4,   6,   4,   4,   4,   6,   4,   4,   4,
6,   4,   4,   4,   6,   4,   4,   4,   6,   4,   4,   4,   4,   4,   4,   4,   6,   4,   4,   4,   6,   4,   4,   4,   6,
4,   4,   4,   6,   4,   4,   4,   6,   4,   6,   4,   6,   6,   4,   4, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255,
255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255,   2,   4,   4,   6,   6,   4,   6,   4,   6,   4,   6,   4,   6,   4,
6,   4,   6,   4,   6,   4,   6,   4,   6,   4,   6,   4,   4,   4,   6,   4,   6,   4,   6,   4,   6,   4,   6,   4,   6,
4,   6,   4,   6,   4,   6,   4,   6,   4,   6,   6,   4,   4,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255,
255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   2,   4,   4,   6,   6,   6,   4,   6,   6,   6,   4,   6,
6,   6,   4,   6,   6,   6,   4,   6,   6,   6,   4,   6,   4,   6,   4,   6,   6,   6,   4,   6,   6,   6,   4,   6,   6,
6,   4,   6,   6,   6,   4,   6,   6,   6,   4,   4, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255,
255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255,   2,   4,   4,   6,   6,   6,   6,   6,   6,   6,
6,   6,   6,   6,   6,   6,   6,   6,   6,   6,   6,   6,   8,   6,   6,   6,   6,   6,   6,   6,   6,   6,   6,   6,   6,
6,   6,   6,   6,   6,   6,   6,   4,   4, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255,
255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255,   2,   4,   4,   8,   6,   6,   8,   6,   6,
8,   6,   6,   8,   6,   6,   8,   6,   6,   8,   6,   8,   6,   8,   6,   6,   8,   6,   6,   8,   6,   6,   8,   6,   6,
8,   6,   6,   8,   4,   4, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255,
255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255,   2,   8,   8,   8,   6,   6,   8,
8,   8,   6,   6,   8,   8,   8,   6,   6,   8,   6,   8,   6,   8,   6,   6,   8,   8,   8,   6,   6,   8,   8,   8,   6,
6,   8,   8,   8, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255,
255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),

( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255,   2,   8,   8,   8,   8,
8,   8,   8,   8,   8,   8,   8,   8,   8,   8,  10,   8,  10,   8,   8,   8,   8,   8,   8,   8,   8,   8,   8,   8,   8,
8,   8,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255,
255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   2,  10,  10,
8,  10,  10,   8,  10,  10,   8,  10,  10,   8,  10,   8,  10,   8,  10,  10,   8,  10,  10,   8,  10,  10,   8,  10,  10,
255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255,
255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255,   2,  12,
10,  12,  10,  12,  10,  12,  10,  12,  10,  10,  12,  10,  10,  12,  10,  12,  10,  12,  10,  12,  10,  12, 255, 255, 255,
255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255,
255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255,
255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255, 255, 255, 255, 255, 255, 255,   255, 255, 255, 255,
2,  14,  14,  12,  14,  14,  12,  14,  14,  12,  12,  12,  14,  14,  12,  14,  14,  12,  14,  14, 255, 255, 255, 255, 255,
```

```
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 2, 16, 18, 16, 18, 16, 18, 16, 16, 16, 18, 16, 18, 16, 18, 16, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 2, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 2, 36, 36, 36, 36, 36, 36, 36, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 2, 84, 84, 84, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 2, 252, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 ),
( 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255 )
);

    variable color_c : integer := 0; -- signed( 31 downto 0 ) := X"00000000";
    variable color_d : integer := 0; -- signed( 31 downto 0 ) := X"00000000";
    variable color_e : integer := 0; -- signed( 31 downto 0 ) := X"00000000";
    variable color_r_faded : signed( 31 downto 0 ) := X"00000000";
    variable color_g_faded : signed( 31 downto 0 ) := X"00000000";
    variable color_b_faded : signed( 31 downto 0 ) := X"00000000";
    type TYPE_FRAME is array( 0 to 29 ) of integer;
    variable frame : TYPE_FRAME := ( others => 0 );
    begin
        if falling_edge( clk_25 ) then
            if reset = '1' then -- reset
                is_fading_out <= '0';
            else
                -- draw sprite 0
                -- within this sprite area and when it is activated
                if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart( 0 ) - 1
                    and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 0 ) + 128 and Vcount < VSYNC + VBACK_PORCH + Card_VStart( 0 ) + 64 then
                    is_fading_out <= '0';
                    case animation_status( 0 ) is
                        when X"00000000" => -- normal showing
                            frame( 0 ) := 0;
                            if sprite_status( 16 ) = '0' then    -- if enabled
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                    ram_address <= ( "00" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                else
                                    pic0 <= '1';
                                    ram_address <= ram_address + 2;
                                end if;
                                if sprite_status( 0 ) = '0' then -- show back
                                    color_out <= X"39DF";
                                else
                                    color_out <= ram_data;
                                end if;
                            end if;
                        when X"00000001" => -- squizing back image and expanding front image
                            -- increase frame
                            if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 0 ) and frame( 0 ) < 59 then
                                frame( 0 ) := frame( 0 ) + 1;
                            end if;
                            if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                ram_address <= ( "00" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                            else
                                pic0 <= '1';
                                if frame( 0 ) < 30 then
                                    -- squize back image
                                    if offset_table_squizing( frame( 0 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                        color_out <= X"0000";
                                    else
                                        color_out <= X"39DF";
                                    end if;
                                else
                                    -- expand front image
                                    if offset_table_squizing( 59 - frame( 0 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                        color_out <= X"0000";
                                    else
                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 0 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
```

```
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        end if;
                when X"00000002" => -- squizing front image and expanding back image
                        -- increase frame
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 0 ) < 59 then
                                frame( 0 ) := frame( 0 ) + 1;
                        end if;
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                ram_address <= ( "00" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 0 ) ) * 256 );
                        else
                                pic0 <= '1';
                                if frame( 0 ) < 30 then
                                        -- squize front image
                                        if offset_table_squizing( frame( 0 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 0 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( frame( 0 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                color_out <= ram_data;
                                        end if;
                                else
                                        -- expand back image
                                        if offset_table_squizing( 59 - frame( 0 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                color_out <= X"39DF";
                                        end if;
                                end if;
                        end if;
                when   X"00000003" => -- fading out
                        is_fading_out <= '1';
                        -- access ram
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                ram_address <= ( "00" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 0 ) ) * 256 );
                        else
                                pic0 <= '1';
                                ram_address <= ram_address + 2;
                                current_frame <= frame( 0 );
                        end if;
                        -- increase frame
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 0 ) < 59 then
                                frame( 0 ) := frame( 0 ) + 1;
                        end if;
                when others =>
                        pic0 <= '0';
        end case;
-- out of the sprite area
else
        pic0 <= '0';
end if;

-- draw sprite 1
-- within this sprite area and when it is activated
if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
0 ) - 1
        and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 1 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 0 ) + 64 then
        is_fading_out <= '0';
        case animation_status( 1 ) is
                when X"00000000" => -- normal showing
                        frame( 1 ) := 0;
                        if sprite_status( 17 ) = '0' then    -- if enabled
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        --if sprite_status( 1 ) = '0' then
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                        --else
                                                ram_address <= ( "00" & X"4000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                        --end if;
                                else
                                        pic1 <= '1';
                                        ram_address <= ram_address + 2;
                                end if;
                                if sprite_status( 1 ) = '0' then -- show back
                                        color_out <= X"39DF";
                                else
                                        color_out <= ram_data;
                                end if;
                        end if;
                when X"00000001" => -- squizing back image and expanding front image
                        -- increase frame
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 1 ) < 59 then
                                frame( 1 ) := frame( 1 ) + 1;
                        end if;
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                --if frame( 1 ) < 30 then
                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                --else
```

```
                                                ram_address <= ( "00" & X"4000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                        --end if;
                                    else
                                        pic1 <= '1';
                                        if frame( 1 ) < 30 then
                                            -- squize back image
                                            if offset_table_squizing( frame( 1 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 1 ) ) ) = 255 then
                                                color_out <= X"0000";
                                            else
                                                ram_address <= ram_address + offset_table_squizing( frame( 1 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                color_out <= X"39DF";
                                            end if;
                                        else
                                            -- expand front image
                                            if offset_table_squizing( 59 - frame( 1 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                color_out <= X"0000";
                                            else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 1 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                color_out <= ram_data;
                                            end if;
                                        end if;
                                    end if;
                                when X"00000002" => -- squizing front image and expanding back image
                                    -- increase frame
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 1 ) < 59 then
                                        frame( 1 ) := frame( 1 ) + 1;
                                    end if;
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        --if frame( 1 ) < 30 then
                                            ram_address <= ( "00" & X"4000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                        --else
                                            --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                        --end if;
                                    else
                                        pic1 <= '1';
                                        if frame( 1 ) < 30 then
                                            -- squize front image
                                            if offset_table_squizing( frame( 1 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 1 ) ) ) = 255 then
                                                color_out <= X"0000";
                                            else
                                                ram_address <= ram_address + offset_table_squizing( frame( 1 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                color_out <= ram_data;
                                            end if;
                                        else
                                            -- expand back image
                                            if offset_table_squizing( 59 - frame( 1 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                color_out <= X"0000";
                                            else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 1 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                color_out <= X"39DF";
                                            end if;
                                        end if;
                                    end if;
                                when   X"00000003" => -- fading out
                                    is_fading_out <= '1';
                                    current_frame <= frame( 1 );
                                    -- access ram
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        ram_address <= ( "00" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 0 ) ) * 256 );
                                    else
                                        pic1 <= '1';
                                        ram_address <= ram_address + 2;
                                    end if;
                                    -- increase frame
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 1 ) < 59 then
                                        frame( 1 ) := frame( 1 ) + 1;
                                    end if;
                                when others =>
                                    pic1 <= '0';
                            end case;
                        -- out of the sprite area
                        else
                            pic1 <= '0';
                        end if;

                        -- draw sprite 2
                        -- within this sprite area and when it is activated
                        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
0 ) - 1
                            and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 2 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 0 ) + 64 then
                            is_fading_out <= '0';
                            case animation_status( 2 ) is
                                when X"00000000" => -- normal showing
                                    frame( 2 ) := 0;
```

```
                                        if sprite_status( 18 ) = '0' then    -- if enabled
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                        --if sprite_status( 2 ) = '0' then -- show back
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                        --else -- show front
                                                                ram_address <= ( "00" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic2 <= '1';
                                                        ram_address <= ram_address + 2;
                                                end if;
                                                if sprite_status( 2 ) = '0' then -- show back
                                                        color_out <= X"39DF";
                                                else
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                when X"00000001" => -- squizing back image and expanding front image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 2 ) < 59 then
                                                frame( 2 ) := frame( 2 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                --if frame( 2 ) < 30 then
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                --else
                                                        ram_address <= ( "00" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                --end if;
                                        else
                                                pic2 <= '1';
                                                if frame( 2 ) < 30 then
                                                        -- squize back image
                                                        if offset_table_squizing( frame( 2 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 2 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 2 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                else
                                                        -- expand front image
                                                        if offset_table_squizing( 59 - frame( 2 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 2 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                color_out <= ram_data;
                                                        end if;
                                                end if;
                                        end if;
                                when X"00000002" => -- squizing front image and expanding back image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 2 ) < 59 then
                                                frame( 2 ) := frame( 2 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                --if frame( 2 ) < 30 then
                                                        ram_address <= ( "00" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                --else
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                --end if;
                                        else
                                                pic2 <= '1';
                                                if frame( 2 ) < 30 then
                                                        -- squize front image
                                                        if offset_table_squizing( frame( 2 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 2 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 2 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                color_out <= ram_data;
                                                        end if;
                                                else
                                                        -- expand back image
                                                        if offset_table_squizing( 59 - frame( 2 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 2 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                end if;
                                        end if;
                                when    X"00000003" => -- fading out
                                        is_fading_out <= '1';
                                        current_frame <= frame( 2 );
                                        -- access ram
```

```
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                ram_address <= ( "00" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 0 ) ) * 256 );
                                        else
                                                pic2 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 2 ) < 59 then
                                                frame( 2 ) := frame( 2 ) + 1;
                                        end if;
                                when others =>
                                        pic2 <= '0';
                        end case;
                -- out of the sprite area
                else
                        pic2 <= '0';
                end if;

                -- draw sprite 3
                -- within this sprite area and when it is activated
                if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
0 ) - 1
                        and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 3 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 0 ) + 64 then
                        is_fading_out <= '0';
                        case animation_status( 3 ) is
                                when X"00000000" => -- normal showing
                                        frame( 3 ) := 0;
                                        if sprite_status( 19 ) = '0' then    -- if enabled
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                        --if sprite_status( 3 ) = '0' then -- show back
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                        --else -- show front
                                                                ram_address <= ( "00" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic3 <= '1';
                                                        ram_address <= ram_address + 2;
                                                end if;
                                                if sprite_status( 3 ) = '0' then -- show back
                                                        color_out <= X"39DF";
                                                else
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                when X"00000001" => -- squizing back image and expanding front image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 3 ) < 59 then
                                                frame( 3 ) := frame( 3 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                --if frame( 3 ) < 30 then
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                --else
                                                        ram_address <= ( "00" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                --end if;
                                        else
                                                pic3 <= '1';
                                                if frame( 3 ) < 30 then
                                                        -- squize back image
                                                        if offset_table_squizing( frame( 3 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 3 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 3 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                else
                                                        -- expand front image
                                                        if offset_table_squizing( 59 - frame( 3 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 3 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                                color_out <= ram_data;
                                                        end if;
                                                end if;
                                        end if;
                                when X"00000002" => -- squizing front image and expanding back image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 3 ) < 59 then
                                                frame( 3 ) := frame( 3 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                --if frame( 3 ) < 30 then
                                                        ram_address <= ( "00" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                                --else
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
```

```
VBACK_PORCH - Card_VStart( 0 ) ) * 256 );
                                        --end if;
                                    else
                                        pic3 <= '1';
                                        if frame( 3 ) < 30 then
                                            -- squize front image
                                            if offset_table_squizing( frame( 3 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 3 ) ) ) = 255 then
                                                color_out <= X"0000";
                                            else
                                                ram_address <= ram_address + offset_table_squizing( frame( 3 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                color_out <= ram_data;
                                            end if;
                                        else
                                            -- expand back image
                                            if offset_table_squizing( 59 - frame( 3 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                color_out <= X"0000";
                                            else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 3 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                color_out <= X"39DF";
                                            end if;
                                        end if;
                                    end if;
                                when   X"00000003" => -- fading out
                                    is_fading_out <= '1';
                                    current_frame <= frame( 3 );
                                    -- access ram
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                        ram_address <= ( "00" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 0 ) ) * 256 );
                                    else
                                        pic3 <= '1';
                                        ram_address <= ram_address + 2;
                                    end if;
                                    -- increase frame
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 0 ) and frame( 3 ) < 59 then
                                        frame( 3 ) := frame( 3 ) + 1;
                                    end if;
                                when others =>
                                    pic3 <= '0';
                            end case;
                        -- out of the sprite area
                        else
                            pic3 <= '0';
                        end if;

                        -- draw sprite 4
                        -- within this sprite area and when it is activated
                        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
1 ) - 1
                            and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 0 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 1 ) + 64 then
                            is_fading_out <= '0';
                            case animation_status( 4 ) is
                                when X"00000000" => -- normal showing
                                    frame( 4 ) := 0;
                                    if sprite_status( 20 ) = '0' then    -- if enabled
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                            --if sprite_status( 4 ) = '0' then -- show back
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                            --else -- show front
                                                ram_address <= ( "01" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                            --end if;
                                        else
                                            pic4 <= '1';
                                            ram_address <= ram_address + 2;
                                        end if;
                                        if sprite_status( 4 ) = '0' then -- show back
                                            color_out <= X"39DF";
                                        else
                                            color_out <= ram_data;
                                        end if;
                                    end if;
                                when X"00000001" => -- squizing back image and expanding front image
                                    -- increase frame
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 1 ) and frame( 4 ) < 59 then
                                        frame( 4 ) := frame( 4 ) + 1;
                                    end if;
                                    if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                        --if frame( 4 ) < 30 then
                                            --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --else
                                            ram_address <= ( "01" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --end if;
                                    else
                                        pic4 <= '1';
                                        if frame( 4 ) < 30 then
                                            -- squize back image
                                            if offset_table_squizing( frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 0 ) ) ) = 255 then
```

```
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                color_out <= X"39DF";
                                        end if;
                                else
                                        -- expand front image
                                        if offset_table_squizing( 59 - frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        end if;
                when X"00000002" => -- squizing front image and expanding back image
                        -- increase frame
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 1 ) and frame( 4 ) < 59 then
                                frame( 4 ) := frame( 4 ) + 1;
                        end if;
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                --if frame( 4 ) < 30 then
                                        ram_address <= ( "01" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                --else
                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                --end if;
                        else
                                pic4 <= '1';
                                if frame( 4 ) < 30 then
                                        -- squize front image
                                        if offset_table_squizing( frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                color_out <= ram_data;
                                        end if;
                                else
                                        -- expand back image
                                        if offset_table_squizing( 59 - frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 4 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                color_out <= X"39DF";
                                        end if;
                                end if;
                        end if;
                when   X"00000003" => -- fading out
                        is_fading_out <= '1';
                        current_frame <= frame( 4 );
                        -- access ram
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                ram_address <= ( "01" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                        else
                                pic4 <= '1';
                                ram_address <= ram_address + 2;
                        end if;
                        -- increase frame
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 1 ) and frame( 4 ) < 59 then
                                frame( 4 ) := frame( 4 ) + 1;
                        end if;
                when others =>
                        pic4 <= '0';
        end case;
    -- out of the sprite area
    else
        pic4 <= '0';
    end if;

    -- draw sprite 5
    -- within this sprite area and when it is activated
    if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart( 1 ) - 1
        and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 1 ) + 128 and Vcount < VSYNC + VBACK_PORCH + Card_VStart( 1 ) + 64 then
        is_fading_out <= '0';
        case animation_status( 5 ) is
                when X"00000000" => -- normal showing
                        frame( 5 ) := 0;
                        if sprite_status( 21 ) = '0' then    -- if enabled
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        --if sprite_status( 5 ) = '0' then -- show back
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --else -- show front
                                                ram_address <= ( "01" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --end if;
```

```
                                        else
                                                pic5 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        if sprite_status( 5 ) = '0' then -- show back
                                                color_out <= X"39DF";
                                        else
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        when X"00000001" => -- squizing back image and expanding front image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 1 ) and frame( 5 ) < 59 then
                                        frame( 5 ) := frame( 5 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        --if frame( 5 ) < 30 then
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --else
                                                ram_address <= ( "01" & X"4000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --end if;
                                else
                                        pic5 <= '1';
                                        if frame( 5 ) < 30 then
                                                -- squize back image
                                                if offset_table_squizing( frame( 5 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 1 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 5 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        else
                                                -- expand front image
                                                if offset_table_squizing( 59 - frame( 5 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 5 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                end if;
                        when X"00000002" => -- squizing front image and expanding back image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 1 ) and frame( 5 ) < 59 then
                                        frame( 5 ) := frame( 5 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        --if frame( 5 ) < 30 then
                                                ram_address <= ( "01" & X"4000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --else
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --end if;
                                else
                                        pic5 <= '1';
                                        if frame( 5 ) < 30 then
                                                -- squize front image
                                                if offset_table_squizing( frame( 5 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 1 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 5 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        else
                                                -- expand back image
                                                if offset_table_squizing( 59 - frame( 5 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 5 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        end if;
                                end if;
                        when    X"00000003" => -- fading out
                                is_fading_out <= '1';
                                current_frame <= frame( 5 );
                                -- access ram
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        ram_address <= ( "01" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 1 ) ) * 256 );
                                else
                                        pic5 <= '1';
                                        ram_address <= ram_address + 2;
                                end if;
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
```

```
VBACK_PORCH + Card_VStart( 1 ) and frame( 5 ) < 59 then
                                        frame( 5 ) := frame( 5 ) + 1;
                                end if;
                        when others =>
                                pic5 <= '0';
                end case;
        -- out of the sprite area
        else
                pic5 <= '0';
        end if;

        -- draw sprite 6
        -- within this sprite area and when it is activated
        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart( 1 ) - 1
                and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 2 ) + 128 and Vcount < VSYNC + VBACK_PORCH + Card_VStart( 1 ) + 64 then
                is_fading_out <= '0';
                case animation_status( 6 ) is
                        when X"00000000" => -- normal showing
                                frame( 6 ) := 0;
                                if sprite_status( 22 ) = '0' then    -- if enabled
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                --if sprite_status( 6 ) = '0' then -- show back
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                                --else -- show front
                                                        ram_address <= ( "01" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                                --end if;
                                        else
                                                pic6 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        if sprite_status( 6 ) = '0' then -- show back
                                                color_out <= X"39DF";
                                        else
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        when X"00000001" => -- squizing back image and expanding front image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 1 ) and frame( 6 ) < 59 then
                                        frame( 6 ) := frame( 6 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                        --if frame( 6 ) < 30 then
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --else
                                                ram_address <= ( "01" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --end if;
                                else
                                        pic6 <= '1';
                                        if frame( 6 ) < 30 then
                                                -- squize back image
                                                if offset_table_squizing( frame( 6 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 6 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        else
                                                -- expand front image
                                                if offset_table_squizing( 59 - frame( 6 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 6 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                end if;
                        when X"00000002" => -- squizing front image and expanding back image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 1 ) and frame( 6 ) < 59 then
                                        frame( 6 ) := frame( 6 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                        --if frame( 6 ) < 30 then
                                                ram_address <= ( "01" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --else
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --end if;
                                else
                                        pic6 <= '1';
                                        if frame( 6 ) < 30 then
                                                -- squize front image
                                                if offset_table_squizing( frame( 6 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
```

```
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 6 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                color_out <= ram_data;
                                                        end if;
                                                else
                                                        -- expand back image
                                                        if offset_table_squizing( 59 - frame( 6 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 6 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                end if;
                                        end if;
                                when    X"00000003" => -- fading out
                                        is_fading_out <= '1';
                                        current_frame <= frame( 6 );
                                        -- access ram
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                ram_address <= ( "01" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 1 ) ) * 256 );
                                        else
                                                pic6 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 1 ) and frame( 6 ) < 59 then
                                                frame( 6 ) := frame( 6 ) + 1;
                                        end if;
                                when others =>
                                        pic6 <= '0';
                        end case;
                -- out of the sprite area
                else
                        pic6 <= '0';
                end if;

                -- draw sprite 7
                -- within this sprite area and when it is activated
                if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
1 ) - 1
                        and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 3 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 1 ) + 64 then
                        is_fading_out <= '0';
                        case animation_status( 7 ) is
                                when X"00000000" => -- normal showing
                                        frame( 7 ) := 0;
                                        if sprite_status( 23 ) = '0' then    -- if enabled
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                        --if sprite_status( 7 ) = '0' then -- show back
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                                        --else  -- show front
                                                                ram_address <= ( "01" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic7 <= '1';
                                                        ram_address <= ram_address + 2;
                                                end if;
                                                if sprite_status( 7 ) = '0' then -- show back
                                                        color_out <= X"39DF";
                                                else
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                when X"00000001" => -- squizing back image and expanding front image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 1 ) and frame( 7 ) < 59 then
                                                frame( 7 ) := frame( 7 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                --if frame( 7 ) < 30 then
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                                --else
                                                        ram_address <= ( "01" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                                --end if;
                                        else
                                                pic7 <= '1';
                                                if frame( 7 ) < 30 then
                                                        -- squize back image
                                                        if offset_table_squizing( frame( 7 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 3 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 7 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                else
                                                        -- expand front image
                                                        if offset_table_squizing( 59 - frame( 7 ), to_integer( Hcount - HSYNC -
```

```
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 7 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        when X"00000002" => -- squizing front image and expanding back image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 1 ) and frame( 7 ) < 59 then
                                        frame( 7 ) := frame( 7 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                        --if frame( 7 ) < 30 then
                                                ram_address <= ( "01" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --else
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 1 ) ) * 256 );
                                        --end if;
                                else
                                        pic7 <= '1';
                                        if frame( 7 ) < 30 then
                                                -- squize front image
                                                if offset_table_squizing( frame( 7 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 3 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 7 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        else
                                                -- expand back image
                                                if offset_table_squizing( 59 - frame( 7 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 7 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        end if;
                                end if;
                        when   X"00000003" => -- fading out
                                is_fading_out <= '1';
                                current_frame <= frame( 7 );
                                -- access ram
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                        ram_address <= ( "01" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 1 ) ) * 256 );
                                else
                                        pic7 <= '1';
                                        ram_address <= ram_address + 2;
                                end if;
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 1 ) and frame( 7 ) < 59 then
                                        frame( 7 ) := frame( 7 ) + 1;
                                end if;
                        when others =>
                                pic7 <= '0';
                end case;
        -- out of the sprite area
        else
                pic7 <= '0';
        end if;

        -- draw sprite 8
        -- within this sprite area and when it is activated
        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
2 ) - 1
                and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 0 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 2 ) + 64 then
                is_fading_out <= '0';
                case animation_status( 8 ) is
                        when X"00000000" => -- normal showing
                                frame( 8 ) := 0;
                                if sprite_status( 24 ) = '0' then    -- if enabled
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                --if sprite_status( 8 ) = '0' then -- show back
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --else -- show front
                                                        ram_address <= ( "10" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --end if;
                                        else
                                                pic8 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        if sprite_status( 8 ) = '0' then -- show back
                                                color_out <= X"39DF";
                                        else
                                                color_out <= ram_data;
                                        end if;
```

```
                                        end if;
                                when X"00000001" => -- squizing back image and expanding front image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 8 ) < 59 then
                                                frame( 8 ) := frame( 8 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                --if frame( 8 ) < 30 then
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --else
                                                        ram_address <= ( "10" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --end if;
                                        else
                                                pic8 <= '1';
                                                if frame( 8 ) < 30 then
                                                        -- squize back image
                                                        if offset_table_squizing( frame( 8 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 0 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 8 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                else
                                                        -- expand front image
                                                        if offset_table_squizing( 59 - frame( 8 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 8 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                color_out <= ram_data;
                                                        end if;
                                                end if;
                                        end if;
                                when X"00000002" => -- squizing front image and expanding back image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 8 ) < 59 then
                                                frame( 8 ) := frame( 8 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                --if frame( 8 ) < 30 then
                                                        ram_address <= ( "10" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --else
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --end if;
                                        else
                                                pic8 <= '1';
                                                if frame( 8 ) < 30 then
                                                        -- squize front image
                                                        if offset_table_squizing( frame( 8 ), to_integer( Hcount - HSYNC - HBACK_PORCH
- Card_HStart( 0 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 8 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                color_out <= ram_data;
                                                        end if;
                                                else
                                                        -- expand back image
                                                        if offset_table_squizing( 59 - frame( 8 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 8 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                end if;
                                        end if;
                                when   X"00000003" => -- fading out
                                        is_fading_out <= '1';
                                        current_frame <= frame( 8 );
                                        -- access ram
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                ram_address <= ( "10" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 2 ) ) * 256 );
                                        else
                                                pic8 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 8 ) < 59 then
                                                frame( 8 ) := frame( 8 ) + 1;
                                        end if;
                                when others =>
                                        pic8 <= '0';
                        end case;
                -- out of the sprite area
                else
                        pic8 <= '0';
```

```
                        end if;

                        -- draw sprite 9
                        -- within this sprite area and when it is activated
                        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart( 2 ) - 1
                                and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 1 ) + 128 and Vcount < VSYNC + VBACK_PORCH + Card_VStart( 2 ) + 64 then
                                is_fading_out <= '0';
                                case animation_status( 9 ) is
                                        when X"00000000" => -- normal showing
                                                frame( 9 ) := 0;
                                                if sprite_status( 25 ) = '0' then    -- if enabled
                                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                                                --if sprite_status( 9 ) = '0' then -- show back
                                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                                --else -- show front
                                                                        ram_address <= ( "10" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                                --end if;
                                                        else
                                                                pic9 <= '1';
                                                                ram_address <= ram_address + 2;
                                                        end if;
                                                        if sprite_status( 9 ) = '0' then -- show back
                                                                color_out <= X"39DF";
                                                        else
                                                                color_out <= ram_data;
                                                        end if;
                                                end if;
                                        when X"00000001" => -- squizing back image and expanding front image
                                                -- increase frame
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 2 ) and frame( 9 ) < 59 then
                                                        frame( 9 ) := frame( 9 ) + 1;
                                                end if;
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                                        --if frame( 9 ) < 30 then
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                        --else
                                                                ram_address <= ( "10" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic9 <= '1';
                                                        if frame( 9 ) < 30 then
                                                                -- squize back image
                                                                if offset_table_squizing( frame( 9 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( frame( 9 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                                        color_out <= X"39DF";
                                                                end if;
                                                        else
                                                                -- expand front image
                                                                if offset_table_squizing( 59 - frame( 9 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 9 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                                        color_out <= ram_data;
                                                                end if;
                                                        end if;
                                                end if;
                                        when X"00000002" => -- squizing front image and expanding back image
                                                -- increase frame
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 2 ) and frame( 9 ) < 59 then
                                                        frame( 9 ) := frame( 9 ) + 1;
                                                end if;
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                                        --if frame( 9 ) < 30 then
                                                                ram_address <= ( "10" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                        --else
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic9 <= '1';
                                                        if frame( 9 ) < 30 then
                                                                -- squize front image
                                                                if offset_table_squizing( frame( 9 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( frame( 9 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                                        color_out <= ram_data;
                                                                end if;
                                                        else
                                                                -- expand back image
                                                                if offset_table_squizing( 59 - frame( 9 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
```

```
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 9 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                color_out <= X"39DF";
                                        end if;
                                end if;
                        when    X"00000003" => -- fading out
                                is_fading_out <= '1';
                                current_frame <= frame( 9 );
                                -- access ram
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        ram_address <= ( "10" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 2 ) ) * 256 );
                                else
                                        pic9 <= '1';
                                        ram_address <= ram_address + 2;
                                end if;
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 9 ) < 59 then
                                        frame( 9 ) := frame( 9 ) + 1;
                                end if;
                        when others =>
                                pic9 <= '0';
                end case;
        -- out of the sprite area
        else
                pic9 <= '0';
        end if;

        -- draw sprite 10
        -- within this sprite area and when it is activated
        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
2 ) - 1
                and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 2 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 2 ) + 64 then
                is_fading_out <= '0';
                case animation_status( 10 ) is
                        when X"00000000" => -- normal showing
                                frame( 10 ) := 0;
                                if sprite_status( 26 ) = '0' then    -- if enabled
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                --if sprite_status( 10 ) = '0' then -- show back
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --else -- show front
                                                        ram_address <= ( "10" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                --end if;
                                        else
                                                pic10 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        if sprite_status( 10 ) = '0' then -- show back
                                                color_out <= X"39DF";
                                        else
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        when X"00000001" => -- squizing back image and expanding front image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 10 ) < 59 then
                                        frame( 10 ) := frame( 10 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                        --if frame( 10 ) < 30 then
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                        --else
                                                ram_address <= ( "10" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                        --end if;
                                else
                                        pic10 <= '1';
                                        if frame( 10 ) < 30 then
                                                -- squize back image
                                                if offset_table_squizing( frame( 10 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 10 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        else
                                                -- expand front image
                                                if offset_table_squizing( 59 - frame( 10 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 10 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                end if;
```

```
                                        when X"00000002" => -- squizing front image and expanding back image
                                                -- increase frame
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 10 ) < 59 then
                                                        frame( 10 ) := frame( 10 ) + 1;
                                                end if;
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                        --if frame( 10 ) < 30 then
                                                                ram_address <= ( "10" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                        --else
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic10 <= '1';
                                                        if frame( 10 ) < 30 then
                                                                -- squize front image
                                                                if offset_table_squizing( frame( 10 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( frame( 10 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                        color_out <= ram_data;
                                                                end if;
                                                        else
                                                                -- expand back image
                                                                if offset_table_squizing( 59 - frame( 10 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 10 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                                        color_out <= X"39DF";
                                                                end if;
                                                        end if;
                                                end if;
                                        when   X"00000003" => -- fading out
                                                is_fading_out <= '1';
                                                current_frame <= frame( 10 );
                                                -- access ram
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                        ram_address <= ( "10" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 2 ) ) * 256 );
                                                else
                                                        pic10 <= '1';
                                                        ram_address <= ram_address + 2;
                                                end if;
                                                -- increase frame
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 10 ) < 59 then
                                                        frame( 10 ) := frame( 10 ) + 1;
                                                end if;
                                        when others =>
                                                pic10 <= '0';
                                end case;
                        -- out of the sprite area
                        else
                                pic10 <= '0';
                        end if;

                        -- draw sprite 11
                        -- within this sprite area and when it is activated
                        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
2 ) - 1
                                and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 3 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 2 ) + 64 then
                                is_fading_out <= '0';
                                case animation_status( 11 ) is
                                        when X"00000000" => -- normal showing
                                                frame( 11 ) := 0;
                                                if sprite_status( 27 ) = '0' then    -- if enabled
                                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                                --if sprite_status( 11 ) = '0' then -- show back
                                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                                --else  -- show front
                                                                        ram_address <= ( "10" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                                                --end if;
                                                        else
                                                                pic11 <= '1';
                                                                ram_address <= ram_address + 2;
                                                        end if;
                                                        if sprite_status( 11 ) = '0' then -- show back
                                                                color_out <= X"39DF";
                                                        else
                                                                color_out <= ram_data;
                                                        end if;
                                                end if;
                                        when X"00000001" => -- squizing back image and expanding front image
                                                -- increase frame
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 11 ) < 59 then
                                                        frame( 11 ) := frame( 11 ) + 1;
                                                end if;
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                        --if frame( 11 ) < 30 then
```

```
                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                    --else
                                        ram_address <= ( "10" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                    --end if;
                                else
                                    pic11 <= '1';
                                    if frame( 11 ) < 30 then
                                        -- squize back image
                                        if offset_table_squizing( frame( 11 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                            color_out <= X"0000";
                                        else
                                            ram_address <= ram_address + offset_table_squizing( frame( 11 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                            color_out <= X"39DF";
                                        end if;
                                    else
                                        -- expand front image
                                        if offset_table_squizing( 59 - frame( 11 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                            color_out <= X"0000";
                                        else
                                            ram_address <= ram_address + offset_table_squizing( 59 - frame( 11 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                            color_out <= ram_data;
                                        end if;
                                    end if;
                                end if;
                            when X"00000002" => -- squizing front image and expanding back image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 11 ) < 59 then
                                    frame( 11 ) := frame( 11 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                    --if frame( 11 ) < 30 then
                                        ram_address <= ( "10" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                    --else
                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 2 ) ) * 256 );
                                    --end if;
                                else
                                    pic11 <= '1';
                                    if frame( 11 ) < 30 then
                                        -- squize front image
                                        if offset_table_squizing( frame( 11 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                            color_out <= X"0000";
                                        else
                                            ram_address <= ram_address + offset_table_squizing( frame( 11 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                            color_out <= ram_data;
                                        end if;
                                    else
                                        -- expand back image
                                        if offset_table_squizing( 59 - frame( 11 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                            color_out <= X"0000";
                                        else
                                            ram_address <= ram_address + offset_table_squizing( 59 - frame( 11 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                            color_out <= X"39DF";
                                        end if;
                                    end if;
                                end if;
                            when   X"00000003" => -- fading out
                                is_fading_out <= '1';
                                current_frame <= frame( 11 );
                                -- access ram
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                    ram_address <= ( "10" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 2 ) ) * 256 );
                                else
                                    pic11 <= '1';
                                    ram_address <= ram_address + 2;
                                end if;
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 2 ) and frame( 11 ) < 59 then
                                    frame( 11 ) := frame( 11 ) + 1;
                                end if;
                            when others =>
                                pic11 <= '0';
                        end case;
                    -- out of the sprite area
                    else
                        pic11 <= '0';
                    end if;

                    -- draw sprite 12
                    -- within this sprite area and when it is activated
                    if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
3 ) - 1
                        and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 0 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 3 ) + 64 then
                        is_fading_out <= '0';
```

```
                                case animation_status( 12 ) is
                                        when X"00000000" => -- normal showing
                                                frame( 12 ) := 0;
                                                if sprite_status( 28 ) = '0' then    -- if enabled
                                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                                --if sprite_status( 12 ) = '0' then -- show back
                                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                                --else -- show front
                                                                        ram_address <= ( "11" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                                --end if;
                                                        else
                                                                pic12 <= '1';
                                                                ram_address <= ram_address + 2;
                                                        end if;
                                                        if sprite_status( 12 ) = '0' then -- show back
                                                                color_out <= X"39DF";
                                                        else
                                                                color_out <= ram_data;
                                                        end if;
                                                end if;
                                        when X"00000001" => -- squizing back image and expanding front image
                                                -- increase frame
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 12 ) < 59 then
                                                        frame( 12 ) := frame( 12 ) + 1;
                                                end if;
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                        --if frame( 12 ) < 30 then
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                        --else
                                                                ram_address <= ( "11" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic12 <= '1';
                                                        if frame( 12 ) < 30 then
                                                                -- squize back image
                                                                if offset_table_squizing( frame( 12 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( frame( 12 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                        color_out <= X"39DF";
                                                                end if;
                                                        else
                                                                -- expand front image
                                                                if offset_table_squizing( 59 - frame( 12 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 12 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                        color_out <= ram_data;
                                                                end if;
                                                        end if;
                                                end if;
                                        when X"00000002" => -- squizing front image and expanding back image
                                                -- increase frame
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 12 ) < 59 then
                                                        frame( 12 ) := frame( 12 ) + 1;
                                                end if;
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                        --if frame( 12 ) < 30 then
                                                                ram_address <= ( "11" & X"0000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                        --else
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic12 <= '1';
                                                        if frame( 12 ) < 30 then
                                                                -- squize front image
                                                                if offset_table_squizing( frame( 12 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( frame( 12 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                        color_out <= ram_data;
                                                                end if;
                                                        else
                                                                -- expand back image
                                                                if offset_table_squizing( 59 - frame( 12 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 0 ) ) ) = 255 then
                                                                        color_out <= X"0000";
                                                                else
                                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 12 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 0 ) ) );
                                                                        color_out <= X"39DF";
                                                                end if;
                                                        end if;
                                                end if;
                                        when    X"00000003" => -- fading out
```

```
                                        is_fading_out <= '1';
                                        current_frame <= frame( 12 );
                                        -- access ram
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 then
                                                ram_address <= ( "11" & X"0000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 3 ) ) * 256 );
                                        else
                                                pic12 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 0 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 12 ) < 59 then
                                                frame( 12 ) := frame( 12 ) + 1;
                                        end if;
                                when others =>
                                        pic12 <= '0';
                        end case;
                -- out of the sprite area
                else
                        pic12 <= '0';
                end if;

                -- draw sprite 13
                -- within this sprite area and when it is activated
                if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
3 ) - 1
                        and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 1 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 3 ) + 64 then
                        is_fading_out <= '0';
                        case animation_status( 13 ) is
                                when X"00000000" => -- normal showing
                                        frame( 13 ) := 0;
                                        if sprite_status( 29 ) = '0' then    -- if enabled
                                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                                        --if sprite_status( 13 ) = '0' then -- show back
                                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                        --else -- show front
                                                                ram_address <= ( "11" & X"4000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                        --end if;
                                                else
                                                        pic13 <= '1';
                                                        ram_address <= ram_address + 2;
                                                end if;
                                                if sprite_status( 13 ) = '0' then -- show back
                                                        color_out <= X"39DF";
                                                else
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                when X"00000001" => -- squizing back image and expanding front image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 13 ) < 59 then
                                                frame( 13 ) := frame( 13 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                                --if frame( 13 ) < 30 then
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                --else
                                                        ram_address <= ( "11" & X"4000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                --end if;
                                        else
                                                pic13 <= '1';
                                                if frame( 13 ) < 30 then
                                                        -- squize back image
                                                        if offset_table_squizing( frame( 13 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( frame( 13 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                                color_out <= X"39DF";
                                                        end if;
                                                else
                                                        -- expand front image
                                                        if offset_table_squizing( 59 - frame( 13 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                                color_out <= X"0000";
                                                        else
                                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 13 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                                color_out <= ram_data;
                                                        end if;
                                                end if;
                                        end if;
                                when X"00000002" => -- squizing front image and expanding back image
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 13 ) < 59 then
                                                frame( 13 ) := frame( 13 ) + 1;
                                        end if;
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                                --if frame( 13 ) < 30 then
                                                        ram_address <= ( "11" & X"4000" ) + to_integer( ( Vcount - VSYNC -
```

```
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                        --else
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                        --end if;
                                else
                                        pic13 <= '1';
                                        if frame( 13 ) < 30 then
                                                -- squize front image
                                                if offset_table_squizing( frame( 13 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 13 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        else
                                                -- expand back image
                                                if offset_table_squizing( 59 - frame( 13 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 1 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 13 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 1 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        end if;
                                end if;
                        when   X"00000003" => -- fading out
                                is_fading_out <= '1';
                                current_frame <= frame( 13 );
                                -- access ram
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 then
                                        ram_address <= ( "11" & X"4000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 3 ) ) * 256 );
                                else
                                        pic13 <= '1';
                                        ram_address <= ram_address + 2;
                                end if;
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 1 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 13 ) < 59 then
                                        frame( 13 ) := frame( 13 ) + 1;
                                end if;
                        when others =>
                                pic13 <= '0';
                end case;
        -- out of the sprite area
        else
                pic13 <= '0';
        end if;

        -- draw sprite 14
        -- within this sprite area and when it is activated
        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart(
3 ) - 1
                and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 2 ) + 128 and Vcount < VSYNC + VBACK_PORCH +
Card_VStart( 3 ) + 64 then
                is_fading_out <= '0';
                case animation_status( 14 ) is
                        when X"00000000" => -- normal showing
                                frame( 14 ) := 0;
                                if sprite_status( 30 ) = '0' then    -- if enabled
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                                --if sprite_status( 14 ) = '0' then -- show back
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                --else  -- show front
                                                        ram_address <= ( "11" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                --end if;
                                        else
                                                pic14 <= '1';
                                                ram_address <= ram_address + 2;
                                        end if;
                                        if sprite_status( 14 ) = '0' then -- show back
                                                color_out <= X"39DF";
                                        else
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        when X"00000001" => -- squizing back image and expanding front image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 14 ) < 59 then
                                        frame( 14 ) := frame( 14 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                        --if frame( 14 ) < 30 then
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                        --else
                                                ram_address <= ( "11" & X"8000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                        --end if;
                                else
                                        pic14 <= '1';
                                        if frame( 14 ) < 30 then
```

```
                                                -- squize back image
                                                if offset_table_squizing( frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        else
                                                -- expand front image
                                                if offset_table_squizing( 59 - frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        end if;
                                end if;
                        when X"00000002" => -- squizing front image and expanding back image
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 3 ) and frame( 14 ) < 59 then
                                        frame( 14 ) := frame( 14 ) + 1;
                                end if;
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                        --if frame( 14 ) < 30 then
                                                ram_address <= ( "11" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                        --else
                                                --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                        --end if;
                                else
                                        pic14 <= '1';
                                        if frame( 14 ) < 30 then
                                                -- squize front image
                                                if offset_table_squizing( frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= ram_data;
                                                end if;
                                        else
                                                -- expand back image
                                                if offset_table_squizing( 59 - frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) ) = 255 then
                                                        color_out <= X"0000";
                                                else
                                                        ram_address <= ram_address + offset_table_squizing( 59 - frame( 14 ), to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 2 ) ) );
                                                        color_out <= X"39DF";
                                                end if;
                                        end if;
                                end if;
                        when   X"00000003" => -- fading out
                                is_fading_out <= '1';
                                current_frame <= frame( 14 );
                                -- access ram
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 then
                                        ram_address <= ( "11" & X"8000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                else
                                        pic14 <= '1';
                                        ram_address <= ram_address + 2;
                                end if;
                                -- increase frame
                                if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 2 ) - 1 and Vcount = VSYNC + VBACK_PORCH + Card_VStart( 3 ) and frame( 14 ) < 59 then
                                        frame( 14 ) := frame( 14 ) + 1;
                                end if;
                        when others =>
                                pic14 <= '0';
                end case;
        -- out of the sprite area
        else
                pic14 <= '0';
        end if;

        -- draw sprite 15
        -- within this sprite area and when it is activated
        if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount > VSYNC + VBACK_PORCH + Card_VStart( 3 ) - 1
                and Hcount < HSYNC + HBACK_PORCH + Card_HStart( 3 ) + 128 and Vcount < VSYNC + VBACK_PORCH + Card_VStart( 3 ) + 64 then
                is_fading_out <= '0';
                case animation_status( 15 ) is
                        when X"00000000" => -- normal showing
                                frame( 15 ) := 0;
                                if sprite_status( 31 ) = '0' then    -- if enabled
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                                --if sprite_status( 15 ) = '0' then -- show back
                                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                                --else  -- show front
```

```
                                                ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                        --end if;
                                else
                                        pic15 <= '1';
                                        ram_address <= ram_address + 2;
                                end if;
                                if sprite_status( 15 ) = '0' then -- show back
                                        color_out <= X"39DF";
                                else
                                        color_out <= ram_data;
                                end if;
                        end if;
                when X"00000001" => -- squizing back image and expanding front image
                        -- increase frame
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 15 ) < 59 then
                                frame( 15 ) := frame( 15 ) + 1;
                        end if;
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                --if frame( 15 ) < 30 then
                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                --else
                                        ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                --end if;
                        else
                                pic15 <= '1';
                                if frame( 15 ) < 30 then
                                        -- squize back image
                                        if offset_table_squizing( frame( 15 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( frame( 15 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                color_out <= X"39DF";
                                        end if;
                                else
                                        -- expand front image
                                        if offset_table_squizing( 59 - frame( 15 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 15 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                color_out <= ram_data;
                                        end if;
                                end if;
                        end if;
                when X"00000002" => -- squizing front image and expanding back image
                        -- increase frame
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 15 ) < 59 then
                                frame( 15 ) := frame( 15 ) + 1;
                        end if;
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                --if frame( 15 ) < 30 then
                                        --ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                --else
                                        ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC -
VBACK_PORCH - Card_VStart( 3 ) ) * 256 );
                                --end if;
                        else
                                pic15 <= '1';
                                if frame( 15 ) < 30 then
                                        -- squize front image
                                        if offset_table_squizing( frame( 15 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( frame( 15 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                color_out <= ram_data;
                                        end if;
                                else
                                        -- expand back image
                                        if offset_table_squizing( 59 - frame( 15 ), to_integer( Hcount - HSYNC -
HBACK_PORCH - Card_HStart( 3 ) ) ) = 255 then
                                                color_out <= X"0000";
                                        else
                                                ram_address <= ram_address + offset_table_squizing( 59 - frame( 15 ),
to_integer( Hcount - HSYNC - HBACK_PORCH - Card_HStart( 3 ) ) );
                                                color_out <= X"39DF";
                                        end if;
                                end if;
                        end if;
                when   X"00000003" => -- fading out
                        is_fading_out <= '1';
                        current_frame <= frame( 15 );
                        -- access ram
                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 then
                                ram_address <= ( "11" & X"C000" ) + to_integer( ( Vcount - VSYNC - VBACK_PORCH -
Card_VStart( 3 ) ) * 256 );
                        else
                                pic15 <= '1';
                                ram_address <= ram_address + 2;
```

```
                                        end if;
                                        -- increase frame
                                        if Hcount = HSYNC + HBACK_PORCH + Card_HStart( 3 ) - 1 and Vcount = VSYNC +
VBACK_PORCH + Card_VStart( 3 ) and frame( 15 ) < 59 then
                                                frame( 15 ) := frame( 15 ) + 1;
                                        end if;
                                when others =>
                                        pic15 <= '0';
                        end case;
                -- out of the sprite area
                else
                        pic15 <= '0';
                end if;

                if Hcount >= HSYNC + HBACK_PORCH + Card_HStart( 3 ) + 128 then
                        is_fading_out <= '0';
                end if;

        end if;
    end if;
end process SpriteGen;


-- Registered video signals going to the video DAC
VideoOut: process (clk_25, reset)
    -- variables for transferring from cursor_ to VGA_R,G,B
    variable color_r : unsigned( 9 downto 0 ) := "0000000000";
    variable color_g : unsigned( 9 downto 0 ) := "0000000000";
    variable color_b : unsigned( 9 downto 0 ) := "0000000000";

    variable sprite_active : std_logic := '0';

    variable color_c : integer := 0; -- signed( 31 downto 0 ) := X"00000000";
    variable color_d : integer := 0; -- signed( 31 downto 0 ) := X"00000000";
    variable color_e : integer := 0; -- signed( 31 downto 0 ) := X"00000000";
    variable color_r_faded : unsigned( 31 downto 0 ) := X"00000000";
    variable color_g_faded : unsigned( 31 downto 0 ) := X"00000000";
    variable color_b_faded : unsigned( 31 downto 0 ) := X"00000000";

begin
        -- split 15-bit color data into 5-bit R, G, B data
        color_r( 9 downto 5 ) := color_out( 14 downto 10 );
        color_g( 9 downto 5 ) := color_out( 9 downto 5 );
        color_b( 9 downto 5 ) := color_out( 4 downto 0 );

        sprite_active := pic0 or pic1 or pic2 or pic3 or pic4 or pic5 or pic6 or pic7
                                                        or pic8 or pic9 or pic10 or pic11 or pic12 or pic13 or pic14 or pic15;

    if reset = '1' then
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
    elsif rising_edge( clk_25 ) then
                if is_cursor_drawing = '1' then
                        color_r( 9 downto 5 ) := cursor_color_out( COLOR_R_END downto COLOR_R_START );
                        color_g( 9 downto 5 ) := cursor_color_out( COLOR_G_END downto COLOR_G_START );
                        color_b( 9 downto 5 ) := cursor_color_out( COLOR_B_END downto COLOR_B_START );
                        VGA_R <= std_logic_vector( color_r );
            VGA_G <= std_logic_vector( color_g );
            VGA_B <= std_logic_vector( color_b );
        elsif sprite_active = '1' then
                        if is_fading_out = '1' then
                                -- convert rgb -> cde
                                color_c := ( 66 * to_integer( ram_data( 14 downto 10 ) ) + 129 * to_integer( ram_data( 9 downto 5 ) ) + 25
* to_integer( ram_data( 4 downto 0 ) ) );
                                color_d := ( -38 * to_integer( ram_data( 14 downto 10 ) ) - 74 * to_integer( ram_data( 9 downto 5 ) ) + 112
* to_integer( ram_data( 4 downto 0 ) ) );
                                color_e := ( 112 * to_integer( ram_data( 14 downto 10 ) ) - 94 * to_integer( ram_data( 9 downto 5 ) ) - 18 *
to_integer( ram_data( 4 downto 0 ) ) );

                                -- decrease luminance
                                color_c := ( color_c - to_integer( to_unsigned( ( color_c * current_frame ), 32 ) srl 5 ) );

                                        -- convert cde -> rgb
                                        color_r_faded := ( to_unsigned( ( 298 * color_c + 409 * color_e ), 32 ) srl 16 );
                                        color_g_faded := ( to_unsigned( ( 298 * color_c - 100 * color_d - 208 * color_e ), 32 ) srl 16 );
                                        color_b_faded := ( to_unsigned( ( 298 * color_c + 516 * color_d ), 32 ) srl 16 );

                                if color_r_faded( 15 ) = '1' or color_g_faded( 15 ) = '1' or color_b_faded( 15 ) = '1' then
                                        VGA_R <= "0000000000";
                                        VGA_G <= "0000000000";
                                        VGA_B <= "0000000000";
                                else
                                        VGA_R <= std_logic_vector( color_r_faded( 4 downto 0 ) ) & "00000";
                                        VGA_G <= std_logic_vector( color_g_faded( 4 downto 0 ) ) & "00000";
                                        VGA_B <= std_logic_vector( color_b_faded( 4 downto 0 ) ) & "00000";
                                end if;
                        else
                                VGA_R <= std_logic_vector( color_out( 14 downto 10 ) ) & "00000";
                                VGA_G <= std_logic_vector( color_out( 9 downto 5 ) ) & "00000";
                                VGA_B <= std_logic_vector( color_out( 4 downto 0 ) ) & "00000";
                        end if;
        elsif vga_hblank = '0' and vga_vblank ='0' then
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
        else
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
```

```
            VGA_B <= "0000000000";
          end if;
       end if;

--       if EndOfField = '1' then
--               readdata(0)<='1';
--       else
--               readdata(0)<='0';
--       end if;
   end process VideoOut;

   VGA_CLK <= clk_25;
   VGA_HS <= not vga_hsync;
   VGA_VS <= not vga_vsync;
   VGA_SYNC <= '0';
   VGA_BLANK <= not (vga_hsync or vga_vsync);
end rtl;



library ieee;
use ieee.std_logic_1164.all;

entity de2_sram_controller is

   port (
      signal chipselect : in std_logic;
      signal write, read : in std_logic;
      signal address    :    in std_logic_vector(17 downto 0);
      signal readdata : out std_logic_vector(15 downto 0);
      signal writedata : in std_logic_vector(15 downto 0);
      signal byteenable : in std_logic_vector(1 downto 0);

      signal SRAM_DQ      : inout std_logic_vector(15 downto 0);
      signal SRAM_ADDR : out std_logic_vector(17 downto 0);
      signal SRAM_UB_N, SRAM_LB_N : out std_logic;
      signal SRAM_WE_N, SRAM_CE_N : out std_logic;
      signal SRAM_OE_N                    : out std_logic
      );

end de2_sram_controller;

architecture dp of de2_sram_controller is
begin

   SRAM_DQ <= writedata when write = '1' else (others => 'Z');
   readdata <= SRAM_DQ;
   SRAM_ADDR <= address;
   SRAM_UB_N <= not byteenable(1);
   SRAM_LB_N <= not byteenable(0);
   SRAM_WE_N <= not write;
   SRAM_CE_N <= not chipselect;
   SRAM_OE_N <= not read;

end dp;


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity sd_card_clk_controller is
port (

clk             :       in std_logic;
write           :       in std_logic;
chipselect      :       in std_logic;
address                 :       in    unsigned(7 downto 0);
writedata       :       in    unsigned(31 downto 0);

CARD_CLK    :    out std_logic
);

end sd_card_clk_controller;

architecture card of sd_card_clk_controller is
begin
   process(clk)
  begin
   if clk'event and clk ='1' then
      if chipselect = '1' then
               if write = '1' then
                        CARD_CLK <= writedata(0);
               end if;
        end if;
   end if;
end process;
end architecture card;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sd_card_datain_controller is
port (

clk             :       in std_logic;
write           :       in std_logic;
```

```
chipselect    :        in std_logic;
address               :        in    unsigned(7 downto 0);
writedata     :        in    unsigned(31 downto 0);

CARD_CMD  :    out std_logic
);

end sd_card_datain_controller;

architecture card of sd_card_datain_controller is
begin
   process(clk)
  begin
   if clk'event and clk ='1' then
      if chipselect = '1' then
                if write = '1' then
                        CARD_CMD <= writedata(0);
                end if;
        end if;
   end if;
end process;
end architecture card;


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sd_card_dataout_controller is
port (

clk               :        in std_logic;
read              :        in std_logic;
chipselect    :        in std_logic;
address               :        in    unsigned(7 downto 0);
readdata      :        out unsigned(31 downto 0);

CARD_DAT   :    in std_logic
);

end sd_card_dataout_controller;

architecture card of sd_card_dataout_controller is
begin
   process(clk)
  begin
   if clk'event and clk ='1' then
      if chipselect = '1' then
                if read = '1' then
                  readdata(0)<= CARD_DAT;
                end if;
        end if;
   end if;
end process;
end architecture card;
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sd_card_ncs_controller is
port (

clk               :        in std_logic;
write             :        in std_logic;
chipselect    :        in std_logic;
address               :        in    unsigned(7 downto 0);
writedata     :        in    unsigned(31 downto 0);

CARD_SD3   :    out std_logic
);

end sd_card_ncs_controller;

architecture card of sd_card_ncs_controller is
begin
   process(clk)
  begin
   if clk'event and clk ='1' then
      if chipselect = '1' then
                if write = '1' then
                        CARD_SD3 <= writedata(0);
                end if;
        end if;
   end if;
end process;
end architecture card;
```

6.2 C/C++ Code

6.2.1 Application

```
#include <stdio.h>
#include <time.h>
#include "driver/mouse.h"
#include "driver/vga.h"
#include "api/jpeg_decoder.h"
#include "api/LCD.h"
#include <system.h>
#include <io.h>
#include <stdlib.h>

static enum
{
	STATE_INIT,
	STATE_GAME,
	STATE_END
}	GAME_STATE;

extern "C" void fat_loop_files();

extern "C"
{
	void show( );
	int init_file_system( );
	int mount_file_system( );
	const unsigned char * read_next_file( const char * ext, char * filename, int * size );
	void	LCD_Init();
	void	LCD_Show_Text(char* Text);
	void	LCD_Line2();
	void	LCD_Test();

}


void write_image( const int& sprite_id, const int& image_id );

void delay( int time )
{
	while( time-- );
}

int get_clicked_sprite_id( int position_x, int position_y )
{
	if ( position_x > 0 && position_x < 129 && position_y > 45 && position_y < 110 )
	{
		return 0;
	}
	if ( position_x > 170 && position_x < 299 && position_y > 45 && position_y < 110 )
	{
		return 1;
	}
	if ( position_x > 340 && position_x < 469 && position_y > 45 && position_y < 110 )
	{
		return 2;
	}
	if ( position_x > 510 && position_x < 639 && position_y > 45 && position_y < 110 )
	{
		return 3;
	}
	if ( position_x > 0 && position_x < 129 && position_y > 154 && position_y < 219 )
	{
		return 4;
	}
	if ( position_x > 170 && position_x < 299 && position_y > 154 && position_y < 219 )
	{
		return 5;
	}
	if ( position_x > 340 && position_x < 469 && position_y > 154 && position_y < 219 )
	{
		return 6;
	}
	if ( position_x > 510 && position_x < 639 && position_y > 154 && position_y < 219 )
	{
		return 7;
	}
	if ( position_x > 0 && position_x < 129 && position_y > 263 && position_y < 328 )
	{
		return 8;
	}
	if ( position_x > 170 && position_x < 299 && position_y > 263 && position_y < 328 )
	{
		return 9;
	}
	if ( position_x > 340 && position_x < 469 && position_y > 263 && position_y < 328 )
	{
		return 10;
	}
	if ( position_x > 510 && position_x < 639 && position_y > 263 && position_y < 328 )
	{
		return 11;
	}
	if ( position_x > 0 && position_x < 129 && position_y > 372 && position_y < 437 )
```

```
	{
		return 12;
	}
	if ( position_x > 170 && position_x < 299 && position_y > 372 && position_y < 437 )
	{
		return 13;
	}
	if ( position_x > 340 && position_x < 469 && position_y > 372 && position_y < 437 )
	{
		return 14;
	}
	if ( position_x > 510 && position_x < 639 && position_y > 372 && position_y < 437 )
	{
		return 15;
	}
	return -1;
}

extern "C" void write_sram( const int address, const int data );

void write_sram( const int address, const int data )
{
	IOWR_32DIRECT( SRAM_BASE, address * 4, data );
}

void shuffle_cards( int * random_array )
{
	int flag=0,value,index=0,i;
	//int random_array[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

	/* initialize random seed: */
	while(index <=15)
	{
		/* generate random number: */
		value= rand()%16+1;
		//printf( "value=%d    ",value);

		for(i=0;i<=index;i++)
		{
			if(random_array[i]==value)
			{
				flag=1;
				break;
			}
		}

		if(flag==0)
		{
			random_array[index]=value;
			index++;
		}

		flag =0;

	}
	for( i =0; i<=15;i++)
	{
		random_array[i]--;
		random_array[i] /= 2;
	}

	for( i =0; i<=15;i++)
	{
		//printf("%d	", random_array[i]);
	}
}


int main()
{
	printf("Hello from Nios II!\n");
//		void LCD_Test();
	  LCD_Test();

	srand( IORD_32DIRECT( SDRAM_BASE, 0xF000 ) );

	char buf[ 100 ] = { 0, };
	int i = 0;
	sprintf( buf, "%d clicked", i );
	//LCD_Show_Text(buf);
	const unsigned int TIMEOUT = 0xFFFF;

	Mouse& m = Mouse::create( );
	Vga& v = Vga::create( );

	JPEG_Decoder d;

	m.init();
	v.set_resolution( 640, 480 );
	v.locate_cursor( 400, 400 );


/*
	char ext[ 4 ] = "JPG";
	char filename[ 100 ] = { 0, };
	int size = 0;


	init_file_system( );
```

```
	mount_file_system( );
	read_next_file( ext, filename, &size );
	printf( "name: %s, size: %d\n", filename, size );

	read_next_file( ext, filename, &size );
	printf( "name: %s, size: %d\n", filename, size );
//*/

//	for( int i = 0; i < 100; i++ )
//	{
//	  printf( "BEFORE: %02X\n", IORD_8DIRECT( SRAM_BASE, i ) );
//	}
//
//	for( int i = 0; i < 100; i++ )
//	{
//	  IOWR_8DIRECT( SRAM_BASE, i, i );
//	}
//
//	for( int i = 0; i < 100; i++ )
//	{
//	  printf( "AFTER: %02X\n", IORD_8DIRECT( SRAM_BASE, i ) );
//	}
//
//	  return 0;

	v.show_back_image_all( );

	IOWR_32DIRECT( VGA_BASE, 0x0C, 0x00 );
	IOWR_32DIRECT( VGA_BASE, 0x08, 0xFFFFFFFF );

	//while( true )
	{
		fat_loop_files( );
	}

	IOWR_32DIRECT( VGA_BASE, 0x08, 0x00 );

//	  for( int i = 0; i < 16; i++ )
//	  {
//		write_image( i, i / 2 );
//	  }

//
//	  IOWR_32DIRECT( VGA_BASE, 0x0C, 0x7C );
//
//	  IOWR_32DIRECT( VGA_BASE, 0x10, 0x00 );


//	  int flag = 0;
//	  while( true )
//	  {
//		IOWR_32DIRECT( VGA_BASE, 0x0C, 0xFFFF );
//
//		IOWR_32DIRECT( VGA_BASE, 0x10, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x14, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x18, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x1C, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x20, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x24, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x28, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x2C, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x30, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x34, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x38, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x3C, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x40, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x44, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x48, 0x00 );
//		IOWR_32DIRECT( VGA_BASE, 0x4C, 0x00 );
//		delay( 10000000 );
//		IOWR_32DIRECT( VGA_BASE, 0x10, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x14, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x18, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x1C, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x20, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x24, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x28, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x2C, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x30, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x34, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x38, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x3C, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x40, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x44, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x48, flag % 3 + 1 );
//		IOWR_32DIRECT( VGA_BASE, 0x4C, flag % 3 + 1 );
//
//		flag++;
//		delay( 10000000 );
//		//IOWR_32DIRECT( VGA_BASE, 0x0C, 0x01007F );
//		//IOWR_32DIRECT( VGA_BASE, 0x14, 0x00 );
//		//delay( 100000 );
//		printf( " " );
//		//IOWR_32DIRECT( VGA_BASE, 0x14, 0x04 );
//		//delay( 6000000 );
//	  }
```

```
	v.enter_read_mode( );
	//v.show_back_image( 0 );
	//v.show_front_image( 0 );
//	while( true )
//	{
//		v.turn_back_image( 7 );
//		v.turn_back_image( 11 );
//		v.turn_front_image( 7 );
//		v.turn_front_image( 11 );
//	}
//	v.turn_back_image( 0 );
//	v.turn_front_image( 0 );
//
//	while( true )
//	{
//		for( int i = 0; i < 16; i++ )
//		{
//			IOWR_32DIRECT( VGA_BASE, 0x0C, 0x00 );
//			delay( 10000000 );
//			IOWR_32DIRECT( VGA_BASE, 0x10 + 4 * i, 0x00 );
//			delay( 10000000 );
//			IOWR_32DIRECT( VGA_BASE, 0x10 + 4 * i, 0x01 );
//			delay( 10000000 );
//			IOWR_32DIRECT( VGA_BASE, 0x0C, 0x01 << i );
//			IOWR_32DIRECT( VGA_BASE, 0x10 + 4 * i, 0x01 );
//			delay( 10000000 );
//			IOWR_32DIRECT( VGA_BASE, 0x10 + 4 * i, 0x02 );
//			delay( 10000000 );
//		}
//	}

	// shuffle cards
	int random_array[ 16 ] = { 0, };
	shuffle_cards( random_array );
	// distribute cards
	for( int i = 0; i < 16; i++ )
	{
		write_image( i, random_array[ i ]    );
	}

	int turn = 0;
	int selected_sprite[ 2 ] = { -1, };
	bool matched[ 8 ] = { false, };
	int n_matched = 0;
	int state = STATE_GAME;
	while( true )
	{
		int dir_x = 0;
		int dir_y = 0;
		int pos_x = 0;
		int pos_y = 0;

		if( m.update( TIMEOUT ) )
		{
			m.get_direction( dir_x, dir_y );
			v.move_cursor( dir_x, dir_y );
			if( m.get_status( Mouse::LEFT ) == Mouse::PRESSED )
			{
				printf( "pressed\n" );
				switch( state )
				{
					case STATE_INIT:
					state = STATE_GAME;
					v.show_back_image_all( );
					for( int i = 0; i < 8; i++ )
					{
						matched[ i ] = false;
					}
					n_matched = 0;
					// shuffle cards
					shuffle_cards( random_array );
					// distribute cards
					for( int i = 0; i < 16; i++ )
					{
						write_image( i, random_array[ i ]    );
					}
					printf( "image loaded again" );
					break;
					case STATE_GAME:
					// check whether it is same as previous or already matched
					v.get_cursor_position( pos_x, pos_y );
					if( selected_sprite[ 0 ] == get_clicked_sprite_id( pos_x, pos_y ) || matched[ random_array[ get_clicked_sprite_id(
pos_x, pos_y ) ] ] )
					{
						continue;
					}

					v.turn_back_image( get_clicked_sprite_id( pos_x, pos_y ) );

					// check whether it is first matching or second matching
					if( turn == 0 )
					{
						turn++;
						selected_sprite[ 0 ] = get_clicked_sprite_id( pos_x, pos_y );
					}
					else
					{
						// check matching
						turn = 0;
```

```
                                if( random_array[ selected_sprite[ 0 ] ] == random_array[ get_clicked_sprite_id( pos_x, pos_y ) ] )
                                {
                                        matched[ random_array[ selected_sprite[ 0 ] ] ] = true;
                                        n_matched++;
                                        v.fade_out_image( selected_sprite[ 0 ] );
                                        v.fade_out_image( get_clicked_sprite_id( pos_x, pos_y ) );
                                }
                                else
                                {
                                        delay( 7000000 );
                                        v.turn_front_image( selected_sprite[ 0 ] );
                                        v.turn_front_image( get_clicked_sprite_id( pos_x, pos_y ) );
                                }
                                selected_sprite[ 0 ] = -1;
                        }

                        // end check
                        if( n_matched == 8 )
                        {
                                state = STATE_END;
                        }
                        break;
                        case STATE_END:
                        state = STATE_INIT;
                        break;
                }
            }
        }
        else
        {
                m.init( );
        }
    }

    return 0;
```

6.2.2 API

```
#include "api/file.h"

#define NULL        0

extern unsigned char * input_buffer;
extern "C"
{
        int init_file_system( );
        int mount_file_system( );
        const unsigned char * read_next_file( const char * ext, char * filename, int * size );
}

File::File( )
{
        m_n_current_fd = 0;
        init_file_system( );
}

bool File::open( int& fd, const char * filename )
{
        return true;
}

bool File::close( const int& fd )
{
        return true;
}

void read( const char * filename, unsigned char * buf, const int& length )
{
        if( filename == NULL )
        {
                return;
        }

        mount_file_system( );
}

void write( const char * filename, const unsigned char * buf, const int& length )
{
}

#ifndef FILE_H_
#define FILE_H_

class File
{
        public:
        File( );
        bool open( int& fd, const char * filename );
        bool close( const int& fd );
        void read( const char * filename, unsigned char * buf, const int& length );
        void write( const char * filename, const unsigned char * buf, const int& length );

        private:
        static const int N_MAX_FD                       = 100;    //TBD
        int m_n_current_fd;
};
```

```
#endif /*FILE_H_*/


#include "api/jpeg_decoder.h"
#include "api/jpeg_decoder_port.h"

extern unsigned char *FrameBuffer;

void JPEG_Decoder::decode()
{
	jpeg_decoder();
}

#ifndef JPEG_DECODER_H_
#define JPEG_DECODER_H_

class JPEG_Decoder
{
	public:
	void decode( );

	private:
};

#endif /*JPEG_DECODER_H_*/

#ifndef JPEG_PORT_H_
#define JPEG_PORT_H_

extern "C" int jpeg_decoder( );

#endif /*JPEG_PORT_H_*/
```

6.2.3 Driver

```
#include <stdio.h>
#include <io.h>
#include <system.h>
#include "driver/sd_card.h"

bool SD_Card::read_data( unsigned char& buf )
{
	buf = IORD_32DIRECT(SD_CARD_DATAOUT_BASE,0);

	return true;
}

bool SD_Card::write_data( const unsigned char& source )
{
	return true;
}

void SD_Card::clock_high( )
{
	//printf( "clock high\n" );
	IOWR_16DIRECT(SD_CARD_CLK_BASE,0,1);
}

void SD_Card::clock_low( )
{
	//printf( "clock low\n" );
	IOWR_16DIRECT(SD_CARD_CLK_BASE,0,0);
}

void SD_Card::chip_select_high(    )
{
	IOWR_16DIRECT(SD_CARD_NCS_BASE,0,1);
}

void SD_Card::chip_select_low( )
{
	IOWR_16DIRECT(SD_CARD_NCS_BASE,0,0);
}

void SD_Card::data_in_high( )
{
	IOWR_16DIRECT(SD_CARD_DATAIN_BASE,0,1);
}

void SD_Card::data_in_low( )
{
	IOWR_16DIRECT(SD_CARD_DATAIN_BASE,0,0);
}

unsigned char SD_Card::read_data( )
{
	unsigned char result = 0;

	return result;
}

void SD_Card::write_data_into_sd_card( const unsigned char& in )
{
}

void SD_Card::write_command_into_sd_card( const unsigned char& command )
{
}
```

```
unsigned char SD_Card::read_data_from_sd_card( )
{
	unsigned char result = 0;

	return result;
}

//#include <mmc_header.h>
#include <stdio.h>
#include "driver/sd_card.h"

#define CMD buffer
#define ARG (buffer+1)
#define MAX (buffer+5)

using namespace std;

extern "C"
{
	void * memcpy ( void * destination, const void * source, size_t num );
	void send_clks(int a,int b);
	void mmc_init();
	void mmc_disable();
	void send_cmd(unsigned	char cmd, unsigned char* arg);
	unsigned char recv_resp();
	unsigned char send_recv_cmd(unsigned char cmd, unsigned char * arg);
	unsigned char send_recv_cmd(unsigned char cmd, unsigned char * arg);
	int read_data(unsigned int address, unsigned char * data_buf, int size);
	void read_data_block(unsigned int address, unsigned char * data_buf);
	void mmc_super_init(char *arg1);
}

//*
static unsigned char buffer[] = {
	0x40, 0x00, 0x00, 0x00, 0x00, 0x95
};

//*
static unsigned char read_buffer[4096];
static int read_buffer_addr = -1;
//*/
static void delay(int i)
{
	int a;
	for(a=0;a<i;a++)
	{
		int b = 0;
	}
}

void send_clks(int a,int b)
{
/*
	int i;
	for(i=0;i<a;i++)
	{
		delay(b);
		CLK_HI;
		delay(b);
		CLK_LO;
	}
//*/

	SD_Card& s = SD_Card::create( );

	for( int i = 0; i < a; i++ )
	{
		delay( b );
		s.clock_high( );

		delay( b );
		s.clock_low( );
	}
}

void mmc_init()
{
/*
	CLK_LO;
	DIN_HI;
	NCS_HI;
	send_clks(80,60);
//*/

	//printf( "mmc init\n" );

	SD_Card& s = SD_Card::create( );

	s.clock_low( );
	s.data_in_high( );
	s.chip_select_high( );
	send_clks( 80, 60 );
}

void mmc_disable()
{
/*
	CLK_LO;
```

```
	DIN_HI;
	NCS_HI;
//*/

	//printf( "mmc disable\n" );

	SD_Card& s = SD_Card::create( );

	s.clock_low( );
	s.data_in_high( );
	s.chip_select_high( );
}

void send_cmd(unsigned    char cmd, unsigned char* arg)
{
//*
	//printf( "send cmd\n" );

	SD_Card& s = SD_Card::create( );

    unsigned char *temp;
    unsigned char data;
    int i,j;

    //NCS_LO;
    s.chip_select_low( );

    *buffer = 0x40 | cmd;
    for(i=0;i<4;i++)
      *(ARG + i) = *(arg + i);
      temp = CMD;
      for(i=0;i<6;i++)
      {
            for(j=7;j>-1;j--)
            {
                  data = *temp;
                  data = data >> j;
                  data = data & 0x01;
                  //printf("-> %x",data);
                  if( data != 0 )
                  {
                        s.data_in_high( );
                  }
                  else
                  {
                        s.data_in_low( );
                  }

                  send_clks( 1, 60 );
            }
            temp++;
      }

      //DIN_HI;
      s.data_in_high( );
//*/
}

unsigned char recv_resp()
{
//*

	//printf( "recv_resp\n" );

	SD_Card& s = SD_Card::create( );

	unsigned char resp=0x00,temp=0x00;
	int i;

	//temp=IORD_32DIRECT(MMC_DATAOUT1_BASE,0);
	s.read_data( temp );

	while(temp == 1)
	{
		send_clks(1,60);
		//temp=IORD_32DIRECT(MMC_DATAOUT1_BASE,0);
		s.read_data( temp );
	}
	for(i=7;i>-1;i--)
	{
		temp = temp << i;
		resp = resp | temp;
		send_clks(1,60);
		//temp=IORD_32DIRECT(MMC_DATAOUT1_BASE,0);
		s.read_data( temp );
	}
	return resp;
//*/
}

unsigned char send_recv_cmd(unsigned char cmd, unsigned char * arg)
{
	//printf( "send_recv_cmd\n" );

	send_cmd(cmd, arg);
	return recv_resp();
}

int read_data(unsigned int address, unsigned char * data_buf, int size)
```

```
{
	address += 0xF3*512;

	unsigned char temp_buf[512];
	int i, bytes;
	if (size == 4096 && ( ( ( unsigned int ) read_buffer_addr ) == address )) {
		memcpy((unsigned char *) data_buf, read_buffer, 4096);
		return size;
	}
	else {
		//printf("reading %d bytes at addr=%d\n", size, address - 57*512);
		read_buffer_addr = address;
	}

	for (i=0; i<size; i+=512) {
		bytes = (size-i)>512?512:(size-i);
		//printf("copying addr:%d, size=%d\n", i, bytes);
		read_data_block(address + i, temp_buf);
		memcpy((unsigned char *) data_buf + i, temp_buf, bytes);
		memcpy((unsigned char *) read_buffer + i, temp_buf, bytes);
	}

	return size;
}

void read_data_block(unsigned int address, unsigned char * data_buf)
{
   SD_Card& s = SD_Card::create( );

   //printf(".....reading block %d\n", address);
   unsigned char arg[3];
   unsigned char resp=0x11, temp = 0x00;
   int i,j;
   //printf("\n address in hex %x\n",address);
   for(i=3;i>-1;i--)
   {
      //printf("\n -> %x",address);
      //printf(" -> %x", address & 255);
      arg[i]= (address) & 255;
      address = address >> 8;
   }
         //printf("\n %x %x %x %x \n",arg[0],arg[1],arg[2],arg[3]);
   resp = send_recv_cmd(resp,arg);
   if(resp != 0x00) {
      //printf("Error sending data Request %x\n",resp);
   }

   //temp=IORD_32DIRECT(MMC_DATAOUT1_BASE,0);
      s.read_data( temp );


   while(temp == 1)
   {
         send_clks(1,60);
         //temp=IORD_32DIRECT(MMC_DATAOUT1_BASE,0);
         s.read_data( temp );

   }
   //printf("Out of while\n");
   for(j=0;j<512;j++)
   {
      resp = 0x00;
      for(i=7;i>-1;i--)
      {
            send_clks(1,0);
            //temp=IORD_32DIRECT(MMC_DATAOUT1_BASE,0);
            s.read_data( temp );
            temp = temp << i;
            resp = resp | temp;
      }
      data_buf[j]=resp;
   }
   send_clks(8,0);
}
void mmc_super_init(char *arg1)
{
   mmc_init();
   unsigned char resp;
   unsigned char arg[4] = { 0x00, 0x00, 0x00, 0x00 };
   resp = 0x00;
   resp = send_recv_cmd(resp, arg);
   send_clks(8,60);
   //printf("%x This is the response\n",resp);
   resp = 0x01;
   resp = send_recv_cmd(resp, arg);
   send_clks(8,60);
   while(resp == 0x01)
   {
      //printf ("Resp after cmd1 == %x == \n",resp);
      resp = send_recv_cmd(resp,arg);
      send_clks(8,60);
   }
   //printf ("\nLooks like I did it! == %x == \n",resp);
   resp=0x10;
   resp = send_recv_cmd(resp, ( unsigned char * ) arg1);
   send_clks(8,60);
   //printf(" Set block length resp == %x == \n",resp);
}
```

```
#include "vga.h"
#include "system.h"
#include "io.h"
#include <stdio.h>

void Vga::enter_read_mode( )
{
	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_RAM_WRITE, 0x00	);
}

void Vga::locate_cursor( const int& x, const int& y )
{
	unsigned int command = 0;

	// build command
	command = ( x ) + ( y << 10 );

	// send the command to the VGA memory-mapped I/O address
	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_MOVE_CURSOR, command );
}

void Vga::get_cursor_position( int& pos_x, int& pos_y )
{
	pos_x = m_cursor_pos_x;
	pos_y = m_cursor_pos_y;
}

void Vga::move_cursor( const int& dir_x, const int& dir_y )
{
	// apply location
	m_cursor_pos_x += dir_x;
	m_cursor_pos_y += dir_y;

	// check whether the direction exceeds the corner location
	if( m_cursor_pos_x < 0 )
	{
		m_cursor_pos_x = 0;
	}
	else if( m_cursor_pos_x > m_width )
	{
		m_cursor_pos_x = m_width;
	}

	if( m_cursor_pos_y < 0 )
	{
		m_cursor_pos_y = 0;
	}
	else if( m_cursor_pos_y > m_height )
	{
		m_cursor_pos_y = m_height;
	}

	// send the command
	locate_cursor( m_cursor_pos_x, m_cursor_pos_y );
}

void Vga::set_resolution( const int& width, const int& height )
{
	m_width = width;
	m_height = height;
}

void Vga::store_image( const int& sprite_id, const unsigned short * buf, const int& len )
{
	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_RAM_WRITE, 0xFFFFFFFF );

	for( int i = 0; i < len * 4; i += 4 )
	{
		IOWR_32DIRECT( SRAM_BASE, ADDRESS_IMAGE_SPRITE_0 + sprite_id * IMAGE_SIZE + i, *( buf + i / 4 ) );
	}

	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_RAM_WRITE, 0x00 );
}

void Vga::hide_image( const int& sprite_id )
{
	//int data = 0;
	//data = IORD_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET	);
	m_data |= 1 << ( 16 + sprite_id	);
	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET, m_data );
	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x00 );
}

void Vga::hide_image_all( )
{
	m_data = 0xFFFF0000;
	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET, m_data );
	for( int i = 0; i < 16; i++ )
	{
		IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * i, 0x00 );
	}
}

void Vga::show_back_image_all(	)
{
	m_data = 0x00;
	IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET, m_data );
	for( int i = 0; i < 16; i++ )
	{
```

```
            IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * i, 0x00 );
        }
}

void Vga::show_front_image_all( )
{
        m_data = 0xFFFF;
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET, m_data );
        for( int i = 0; i < 16; i++ )
        {
            IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * i, 0x00 );
        }
}

void Vga::show_back_image( const int& sprite_id )
{
        //int data = 0;
        m_data = IORD_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET );
        m_data = ~m_data;
        m_data |= 1 << ( 16 + sprite_id    );
        m_data |= 1 << sprite_id;
        m_data = ~m_data;
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET, m_data );
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x00 );
}

void Vga::show_front_image( const int& sprite_id )
{
        //int data = 0;
        //data = IORD_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET );
        //printf( "data after read: %02X\n", data );
        m_data |= 1 << sprite_id;
        //printf( "data after revert: %02X\n", data );
        m_data = ~m_data;
        //printf( "data after bit operation over 16 bits: %02X\n", data );
        m_data |= 1 << ( 16 + sprite_id    );
        //printf( "data after bit operation: %02X\n", data );
        m_data = ~m_data;
        //printf( "data after bit operation: %02X\n", data );
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_NORMAL_IMAGE_SET, m_data );
        //printf( "data to be written: %02X\n", data );
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x00 );
}

void Vga::turn_back_image( const int& sprite_id )
{
        //hide_image( sprite_id );
        show_back_image( sprite_id );

        //IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x00 );
        delay( 70000 );
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x01 );
        delay( 7000 );
}

void Vga::turn_front_image( const int& sprite_id )
{
        //hide_image( sprite_id );
        show_front_image( sprite_id );

        //IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x00 );
        delay( 70000 );
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x02 );
        delay( 7000 );
}

void Vga::blink_back_image( const int& sprite_id )
{
        for( int i = 0; i < 1; i++ )
        {
            show_back_image( sprite_id );
            delay( 700000 );
            hide_image( sprite_id );
            delay( 700000 );
        }
}

void Vga::blink_front_image( const int& sprite_id )
{
        for( int i = 0; i < 3; i++ )
        {
            show_front_image( sprite_id );
            delay( 700000 );
            hide_image( sprite_id );
            delay( 700000 );
        }
}

void Vga::fade_out_image( const int& sprite_id )
{
        show_front_image( sprite_id );
        delay( 100000 );
        IOWR_32DIRECT( VGA_BASE, ADDRESS_OFFSET_EFFECT_SPRITE_0 + 4 * sprite_id, 0x03 );
        //delay( 10000000 );
}

#ifndef MOUSE_H_
#define MOUSE_H_
```

```
#include "system.h"
#include "io.h"

class Mouse
{
	public:
	~Mouse( ){ }
	typedef enum
	{
		LEFT,
		RIGHT,
		N_BUTTON
	}	TYPE_BUTTON;
	typedef enum
	{
		PRESSED,
		RELEASED,
		N_BUTTON_STATUS
	}	TYPE_BUTTON_STATUS;

	static const int MOUSE_LEFT = LEFT;
	static const int MOUSE_RIGHT = RIGHT;

	static const int BUTTON_PRESSED = PRESSED;
	static const int BUTTON_RELEASED = RELEASED;

	static Mouse& create( ){ static Mouse m; return m; }
	void init( );
	bool update( const int& timeout );
	void get_direction( int& x, int& y ) const;
	const TYPE_BUTTON_STATUS get_status( const TYPE_BUTTON& button ) const;

	private:
	Mouse( ){ }
	static const int COMMAND_RESET                          = 0xFF;
	static const int COMMAND_RESEND                         = 0xFE;
	static const int COMMAND_SET_DEFAULT                    = 0xF6;
	static const int COMMAND_DISABLE_DATA_REPORTING = 0xF5;
	static const int COMMAND_ENABLE_DATA_REPORTING    = 0xF4;
	static const int COMMAND_SET_SAMPLE_RATE          = 0xF3;
	static const int COMMAND_GET_DEVICE_ID            = 0xF2;
	static const int COMMAND_SET_REMOTE_MODE            = 0xF0;
	static const int COMMAND_SET_WRAP_MODE              = 0xEE;
	static const int COMMAND_RESET_WRAP_MODE            = 0xEC;
	static const int COMMAND_READ_DATA                = 0xEB;
	static const int COMMAND_SET_STREAM_MODE            = 0xEA;
	static const int COMMAND_STATUS_REQUEST           = 0xE9;
	static const int COMMAND_SET_RESOLUTION           = 0xE8;
	static const int COMMAND_SET_SCALING_2_TO_1       = 0xE7;
	static const int COMMAND_SET_SCALING_1_TO_1       = 0xE6;

	static const int STATUS_ACK                             = 0xFA;
	static const int STATUS_ERROR                           = 0x00;

	int read( );
	int read( const unsigned int& timeout );
	void write( int data );
	char m_dir_x;
	char m_dir_y;
	TYPE_BUTTON_STATUS m_status[ N_BUTTON ];
};


#endif /*MOUSE_H_*/

#ifndef SD_CARD_H_
#define SD_CARD_H_

class SD_Card
{
	public:
	~SD_Card( )
	{
		m_command_buffer[ 0 ] = 0x40;
		m_command_buffer[ 1 ] = 0x00;
		m_command_buffer[ 2 ] = 0x00;
		m_command_buffer[ 3 ] = 0x00;
		m_command_buffer[ 4 ] = 0x00;
		m_command_buffer[ 5 ] = 0x95;

		m_address_read_buffer = -1;
	}
	static SD_Card& create( ){ static SD_Card card; return card; }
	bool read_data( unsigned char& buf );
	bool write_data( const unsigned char& source );
	void clock_high( );
	void clock_low( );
	void chip_select_high(    );
	void chip_select_low( );
	void data_in_high( );
	void data_in_low( );
	unsigned char read_data( );

	private:
	SD_Card( ){ }
	typedef enum
	{
		N_STATE
	}	STATE;
```

```
        static const int COMMAND_LENGTH =    6;
        static const int SIZE_READ_BUFFER = 4096;

        void write_data_into_sd_card( const unsigned char& in );
        void write_command_into_sd_card( const unsigned char& command );
        unsigned char read_data_from_sd_card( );

        unsigned char m_command_buffer[ COMMAND_LENGTH ];
        unsigned char m_read_buffer[ SIZE_READ_BUFFER ];
        int m_address_read_buffer;

};

#endif /*SD_CARD_H_*/

#ifndef VGA_H_
#define VGA_H_

class Vga
{
        public:
        ~Vga( ){ }

        static Vga& create( ){ static Vga m_vga;    return m_vga; }

        void enter_read_mode( );
        void locate_cursor( const int& x, const int& y );
        void get_cursor_position( int& pos_x, int& pos_y );
        void move_cursor( const int& dir_x, const int& dir_y );
        void store_image( const int& sprite_id, const unsigned short * buf, const int& len );
        void hide_image( const int& sprite_id );
        void hide_image_all( );
        void show_back_image_all(    );
        void show_front_image_all( );
        void show_back_image( const int& sprite_id );
        void show_front_image( const int& sprite_id );
        void turn_back_image( const int& sprite_id );
        void turn_front_image( const int& sprite_id );
        void blink_back_image( const int& sprite_id );
        void blink_front_image( const int& sprite_id );
        void fade_out_image( const int& sprite_id );

        void set_resolution( const int& width, const int& height );

        private:
        Vga( ){ m_data = 0; }

        static const unsigned int N_SPRITE = 16;
        static const unsigned int IMAGE_SIZE = 0x8000;

        static const unsigned int ADDRESS_OFFSET_MOVE_CURSOR                = 0x00;
        static const unsigned int ADDRESS_OFFSET_RAM_WRITE                  = 0x08;
        static const unsigned int ADDRESS_OFFSET_NORMAL_IMAGE_SET       = 0x0C;
//      static const unsigned int ADDRESS_OFFSET_EFFECT_SPRITE[ N_SPRITE ] =
//      {
//              0x10,
//              0x14,
//              0x18,
//              0x1C,
//              0x20,
//              0x24,
//              0x28,
//              0x2C,
//              0x30,
//              0x34,
//              0x38,
//              0x3C,
//              0x40,
//              0x44,
//              0x48,
//              0x4C
//      };
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_0         = 0x10;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_1         = 0x14;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_2         = 0x18;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_3         = 0x1C;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_4         = 0x20;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_5         = 0x24;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_6         = 0x28;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_7         = 0x2C;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_8         = 0x30;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_9         = 0x34;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_10        = 0x38;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_11        = 0x3C;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_12        = 0x40;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_13        = 0x44;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_14        = 0x48;
        static const int ADDRESS_OFFSET_EFFECT_SPRITE_15        = 0x4C;

//      static unsigned int ADDRESS_IMAGE_SPRITE[ N_SPRITE ] =
//      {
//              0x00000000,
//              0x00008000,
//              0x00010000,
//              0x00018000,
//              0x00020000,
//              0x00028000,
//              0x00030000,
```

```
//              0x00038000,
//              0x00040000,
//              0x00048000,
//              0x00050000,
//              0x00058000,
//              0x00060000,
//              0x00068000,
//              0x00070000,
//              0x00078000
//      };

        static const unsigned int ADDRESS_IMAGE_SPRITE_0        = 0x00000000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_1        = 0x00008000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_2        = 0x00010000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_3        = 0x00018000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_4        = 0x00020000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_5        = 0x00028000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_6        = 0x00030000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_7        = 0x00038000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_8        = 0x00040000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_9        = 0x00048000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_10       = 0x00050000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_11       = 0x00058000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_12       = 0x00060000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_13       = 0x00068000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_14       = 0x00070000;
        static const unsigned int ADDRESS_IMAGE_SPRITE_15       = 0x00078000;

        int m_width;
        int m_height;
        int m_cursor_pos_x;
        int m_cursor_pos_y;
        int m_data;

        void delay( int time ){ volatile int delay = time; while( delay-- ); }

};

#endif /*VGA_H_*/

#include "mouse.h"
#include <stdio.h>

int Mouse::read( )
{
        int data = 0;

        while( true )
        {
                data = IORD_32DIRECT( PS2_BASE, 0 );
                if( ( ( data >> 16) & 0xFFFF ) != 0 )
                {
                        return data;
                }
        }

        return data;
}

int Mouse::read( const unsigned int& timeout )
{
        int data = 0;
        volatile unsigned int t = timeout;

        while( t-- )
        {
                data = IORD_32DIRECT( PS2_BASE, 0 );
                if( ( ( data >> 16) & 0xFFFF ) != 0 )
                {
                        return data;
                }
        }

        // if non-valid data until timeout, return useless data
        return STATUS_ERROR;
}

void Mouse::write( int data )
{
        IOWR_8DIRECT( PS2_BASE, 0, data );
}

void Mouse::init( )
{
        write( COMMAND_RESET );
        read( );

        write( COMMAND_RESET );
        read( );

        write( COMMAND_RESET );
        read( );

        write( COMMAND_SET_SAMPLE_RATE );
        read( );

        write( 0xC8 );
        read( );
```

```
	write( COMMAND_SET_SAMPLE_RATE );
	read( );

	write( 0x64 );
	read( );

	write( COMMAND_SET_SAMPLE_RATE );
	read( );

	write( 0x50 );
	read( );

	write( COMMAND_GET_DEVICE_ID );
	read( );

	write( COMMAND_SET_RESOLUTION );
	read( );

	write( 0x01 );
	read( );

	write( COMMAND_SET_SCALING_1_TO_1 );
	read( );

	write( COMMAND_SET_SAMPLE_RATE );
	read( );

	write( 0xC8 );
	read( );

	write( COMMAND_SET_REMOTE_MODE );
	read( );

}

bool Mouse::update( const int& timeout )
{
	int x_sign = 0;
	int y_sign = 0;
	int data[ 10 ] = { 0, };

	// request position data
	IOWR_32DIRECT( PS2_BASE, 0, COMMAND_READ_DATA );

	// wait for ack
	while( read( ) & 0xFF != STATUS_ACK );

	// if new ack is received while receiving data, refresh
	int loop = 3;
	while( loop )
	{
		data[ 3 - loop ] = read( timeout );

		if( data[ 3 - loop ] == STATUS_ERROR )
		{
			return false;
		}
		if( ( unsigned char ) ( data[ 3 - loop ] & 0xFF ) == ( unsigned char ) STATUS_ACK )
		{
			loop = 3;
		}
		else
		{
			loop--;
		}
	}

	// set button
	if( data[ 0 ] & 0x01 == 1 )
	{
		m_status[ LEFT ] = PRESSED;
	}
	else
	{
		m_status[ LEFT ] = RELEASED;
	}

	if( ( data[ 0 ] >> 1 ) & 0x01 == 1 )
	{
		m_status[ RIGHT ] = PRESSED;
	}
	else
	{
		m_status[ RIGHT ] = RELEASED;
	}

	// set position
	if( ( data[ 0 ] >> 4 ) & 0x01 == 1 )
	{
		x_sign = 1;
	}
	if( ( data[ 0 ] >>5 ) & 0x01 == 1 )
	{
		y_sign = 1;
	}

	if( x_sign == 1 )
	{
		m_dir_x = data[ 1 ] & 0xFF;
```

```
        }
        else
        {
                m_dir_x = data[ 1 ] & 0xFF;
        }

        if( y_sign == 0 )
        {
                m_dir_y = 0 - ( data[ 2 ] & 0xFF );
        }
        else
        {
                m_dir_y = ( ~( data[ 2 ] & 0xFF ) + 1 ) & 0xFF;
        }

        // remove mouse's pop data
        if( m_dir_x >= 6 )
        {
                m_dir_x = 0;
        }
        else if( m_dir_x <= -6 )
        {
                m_dir_x = 0;
        }

        if( m_dir_y >= 6 )
        {
                m_dir_y = 0;
        }
        else if( m_dir_y <= -6 )
        {
                m_dir_y = 0;
        }

        // increase mouse speed
        m_dir_x <<= 2;
        m_dir_y <<= 2;

        return true;
}

void Mouse::get_direction( int& x, int& y ) const
{
        x = m_dir_x;
        y = m_dir_y;
}

const Mouse::TYPE_BUTTON_STATUS Mouse::get_status( const TYPE_BUTTON& button ) const
{
        return m_status[ button ];
}
```

4.2.4 Library

```
/* File : jpeg.h, header for all jpeg code */
/* Author: Pierre Guerrier, march 1998          */
/*                                                      */
/* 19/01/99     Edited by Koen van Eijk             */

/*#define SPY*/
/* Leave structures in memory,output something and dump core in the event
     of a failure: */
#define DEBUG 0



/*---------------------------------*/
/* JPEG format parsing markers here */
/*---------------------------------*/

#define SOI_MK      0xFFD8                  /* start of image     */
#define APP_MK      0xFFE0                  /* custom, up to FFEF */
#define COM_MK      0xFFFE                  /* commment segment    */
#define SOF_MK      0xFFC0                  /* start of frame     */
#define SOS_MK      0xFFDA                  /* start of scan       */
#define DHT_MK      0xFFC4                  /* Huffman table    */
#define DQT_MK      0xFFDB                  /* Quant. table                */
#define DRI_MK      0xFFDD                  /* restart interval   */
#define EOI_MK      0xFFD9                  /* end of image                */
#define MK_MSK      0xFFF0

#define RST_MK(x) ( (0xFFF8&(x)) == 0xFFD0 )
                        /* is x a restart interval ? */



/*------------------------------------------------------------ */
/* all kinds of macros here                                     */
/*------------------------------------------------------------ */

#define first_quad(c)       ((c) >> 4)              /* first 4 bits in file order */
#define second_quad(c)     ((c) & 15)

#define HUFF_ID(hclass, id)             (2 * (hclass) + (id))

#define DC_CLASS                0
#define AC_CLASS                1


/*-------------------------------------------------------------*/
```

```
/* JPEG data types here                                          */
/*--------------------------------------------------------------*/

typedef union {               /* block of pixel-space values */
   unsigned char    block[8][8];
   unsigned char    linear[64];
} PBlock;

typedef union {               /* block of frequency-space values */
   int block[8][8];
   int linear[64];
} FBlock;


/* component descriptor structure */

typedef struct {
   unsigned char    CID;   /* component ID */
   unsigned char    IDX;   /* index of first block in MCU */

   unsigned char    HS;    /* sampling factors */
   unsigned char    VS;
   unsigned char    HDIV;  /* sample width ratios */
   unsigned char    VDIV;

   char             QT;    /* QTable index, 2bits       */
   char             DC_HT;       /* DC table index, 1bit */
   char             AC_HT;       /* AC table index, 1bit */
   int          PRED; /* DC predictor value */
} cd_t;


/*--------------------------------------------------*/
/* global variables here                            */
/*--------------------------------------------------*/

extern unsigned char * input_buffer;
extern int input_buf_size;
extern cd_t      comp[3]; /* for every component, useful stuff */

extern PBlock *MCU_buff[10];    /* decoded component buffer */
                                /* between IDCT and color convert */
extern int       MCU_valid[10];    /* for every DCT block, component id then -1 */

extern PBlock *QTable[4];         /* three quantization tables */
extern int       QTvalid[4];         /* at most, but seen as four ... */

extern FILE *fi;
extern FILE *fo;
extern int input_pointer;

/* picture attributes */
extern int x_size, y_size;   /* Video frame size          */
extern int rx_size, ry_size;/* down-rounded Video frame size */
                                /* in pixel units, multiple of MCU */
extern int MCU_sx, MCU_sy;       /* MCU size in pixels      */
extern int mx_size, my_size;       /* picture size in units of MCUs */
extern int n_comp;          /* number of components 1,3 */

/* processing cursor variables */
extern int in_frame, curcomp, MCU_row, MCU_column;
                        /* current position in MCU unit */

/* RGB buffer storage */
extern unsigned char *ColorBuffer;      /* MCU after color conversion */
extern unsigned char *FrameBuffer;      /* complete final RGB image */
extern PBlock               *PBuff;
extern FBlock               *FBuff;

/* process statistics */
extern int stuffers;  /* number of stuff bytes in file */
extern int passed;   /* number of bytes skipped looking for markers */

extern int verbose;

extern int length, width;
/*------------------------------------------*/
/* prototypes from utils.c                  */
/*------------------------------------------*/

extern void show_FBlock(FBlock *S);
extern void show_PBlock(PBlock *S);
extern void bin_dump(FILE *fi);

extern int           ceil_div(int N, int D);
extern int           floor_div(int N, int D);
extern void          reset_prediction();
extern int     reformat(unsigned long S, int good);
extern void          free_structures();
extern void   suicide();
extern void   aborted_stream(FILE *fi, FILE *fo);
extern void   RGB_save(FILE *fo);

/*------------------------------------------*/
/* prototypes from parse.c                  */
/*------------------------------------------*/
extern int    my_get(FILE *fi);
extern void   clear_bits();
extern unsigned long       get_bits(FILE *fi, int number);
```

```
extern unsigned char       get_one_bit(FILE *fi);
extern unsigned intget_size(FILE *fi);
extern unsigned int        get_next_MK(FILE *fi);
extern int     load_quant_tables(FILE *fi);
extern int           init_MCU();
extern void         skip_segment(FILE *fi);
extern int          process_MCU(FILE *fi);

/*----------------------------------------------*/
/* prototypes from fast_idct.c                  */
/*----------------------------------------------*/

extern void   IDCT(const FBlock *S, PBlock *T);

/*----------------------------------------------*/
/* prototypes from color.c                      */
/*----------------------------------------------*/

extern void   color_conversion();

/*----------------------------------------------*/
/* prototypes from table_vld.c or tree_vld.c */
/*----------------------------------------------*/

extern int     load_huff_tables(FILE *fi);
extern unsigned char       get_symbol(FILE *fi, int select);

/*----------------------------------------------*/
/* prototypes from huffman.c                    */
/*----------------------------------------------*/

extern void   unpack_block(FILE *fi, FBlock *T, int comp);
                  /* unpack, predict, dequantize, reorder on store */

/*----------------------------------------------*/
/* prototypes from spy.c                */
/*----------------------------------------------*/

extern void   trace_bits(int number, int type);
extern void   output_stats(char *dumpfile);


#include "mouse.h"
#include <stdio.h>

int Mouse::read( )
{
      int data = 0;

      while( true )
      {
            data = IORD_32DIRECT( PS2_BASE, 0 );
            if( ( ( data >> 16) & 0xFFFF ) != 0 )
            {
                  return data;
            }
      }

      return data;
}

int Mouse::read( const unsigned int& timeout )
{
      int data = 0;
      volatile unsigned int t = timeout;

      while( t-- )
      {
            data = IORD_32DIRECT( PS2_BASE, 0 );
            if( ( ( data >> 16) & 0xFFFF ) != 0 )
            {
                  return data;
            }
      }

      // if non-valid data until timeout, return useless data
      return STATUS_ERROR;
}

void Mouse::write( int data )
{
      IOWR_8DIRECT( PS2_BASE, 0, data );
}

void Mouse::init( )
{
      write( COMMAND_RESET );
      read( );

      write( COMMAND_RESET );
      read( );

      write( COMMAND_RESET );
      read( );

      write( COMMAND_SET_SAMPLE_RATE );
      read( );

      write( 0xC8 );
```

```
	read( );

	write( COMMAND_SET_SAMPLE_RATE );
	read( );

	write( 0x64 );
	read( );

	write( COMMAND_SET_SAMPLE_RATE );
	read( );

	write( 0x50 );
	read( );

	write( COMMAND_GET_DEVICE_ID );
	read( );

	write( COMMAND_SET_RESOLUTION );
	read( );

	write( 0x01 );
	read( );

	write( COMMAND_SET_SCALING_1_TO_1 );
	read( );

	write( COMMAND_SET_SAMPLE_RATE );
	read( );

	write( 0xC8 );
	read( );

	write( COMMAND_SET_REMOTE_MODE );
	read( );

}

bool Mouse::update( const int& timeout )
{
	int x_sign = 0;
	int y_sign = 0;
	int data[ 10 ] = { 0, };

	// request position data
	IOWR_32DIRECT( PS2_BASE, 0, COMMAND_READ_DATA );

	// wait for ack
	while( read( ) & 0xFF != STATUS_ACK );

	// if new ack is received while receiving data, refresh
	int loop = 3;
	while( loop )
	{
		data[ 3 - loop ] = read( timeout );

		if( data[ 3 - loop ] == STATUS_ERROR )
		{
			return false;
		}
		if( ( unsigned char ) ( data[ 3 - loop ] & 0xFF ) == ( unsigned char ) STATUS_ACK )
		{
			loop = 3;
		}
		else
		{
			loop--;
		}
	}

	// set button
	if( data[ 0 ] & 0x01 == 1 )
	{
		m_status[ LEFT ] = PRESSED;
	}
	else
	{
		m_status[ LEFT ] = RELEASED;
	}

	if( ( data[ 0 ] >> 1 ) & 0x01 == 1 )
	{
		m_status[ RIGHT ] = PRESSED;
	}
	else
	{
		m_status[ RIGHT ] = RELEASED;
	}

	// set position
	if( ( data[ 0 ] >> 4 ) & 0x01 == 1 )
	{
		x_sign = 1;
	}
	if( ( data[ 0 ] >>5 ) & 0x01 == 1 )
	{
		y_sign = 1;
	}

	if( x_sign == 1 )
```

```
        {
                m_dir_x = data[ 1 ] & 0xFF;
        }
        else
        {
                m_dir_x = data[ 1 ] & 0xFF;
        }

        if( y_sign == 0 )
        {
                m_dir_y = 0 - ( data[ 2 ] & 0xFF );
        }
        else
        {
                m_dir_y = ( ~( data[ 2 ] & 0xFF ) + 1 ) & 0xFF;
        }

        // remove mouse's pop data
        if( m_dir_x >= 6 )
        {
                m_dir_x = 0;
        }
        else if( m_dir_x <= -6 )
        {
                m_dir_x = 0;
        }

        if( m_dir_y >= 6 )
        {
                m_dir_y = 0;
        }
        else if( m_dir_y <= -6 )
        {
                m_dir_y = 0;
        }

        // increase mouse speed
        m_dir_x <<= 2;
        m_dir_y <<= 2;

        return true;
}

void Mouse::get_direction( int& x, int& y ) const
{
        x = m_dir_x;
        y = m_dir_y;
}

const Mouse::TYPE_BUTTON_STATUS Mouse::get_status( const TYPE_BUTTON& button ) const
{
        return m_status[ button ];
}



/* The FreeDOS-32 FAT Driver version 2.0
  * Copyright (C) 2001-2005     Salvatore ISAJA
  *
  * This file "ondisk.h" is part of the FreeDOS-32 FAT Driver (the Program).
  *
  * The 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.
  *
  * The 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 the Program; see the file GPL.txt; if not, write to
  * the Free Software Foundation, Inc.,
  * 59 Temple Place, Suite 330, Boston, MA     02111-1307     USA
  */
/**
  * \file
  * \brief Format for the on-disk data structures for the FAT file system.
  */
/**
  * \addtogroup fat
  * @{
  */
#ifndef __FAT_ONDISK_H
#define __FAT_ONDISK_H

#ifndef PACKED
  #define PACKED __attribute__ ((packed))
#endif


/* EOC (End Of Clusterchain) check macros.
  * These expressions are true (nonzero) if the value of a FAT entry is
  * an EOC for the FAT type. An EOC indicates the last cluster of a file.
  */
#define    IS_FAT12_EOC(entry_value)    (entry_value >= 0x0FF8)
#define    IS_FAT16_EOC(entry_value)    (entry_value >= 0xFFF8)
#define    IS_FAT32_EOC(entry_value)    (entry_value >= 0x0FFFFFF8)
```

```
/* File attributes */
#define FAT_ARDONLY    0x01 /* Read only */
#define FAT_AHIDDEN    0x02 /* Hidden */
#define FAT_ASYSTEM    0x04 /* System */
#define FAT_AVOLID     0x08 /* Volume label */
#define FAT_ADIR       0x10 /* Directory */
#define FAT_AARCHIV    0x20 /* Modified since last backup */
#define FAT_ALFN       0x0F /* Long file name directory slot (R+H+S+V) */
#define FAT_AALL       0x3F /* Select all attributes */
#define FAT_ANONE      0x00 /* Select no attributes */
#define FAT_ANOVOLID 0x37 /* All attributes but volume label */


/* Because of the FAT??_BAD markers, the following are the max cluster
  * number for a FAT file system:
  * FAT12: 4086 (0x0FF6), FAT16: 65526 (0xFFF6), FAT32: 0x0FFFFFF6.
  */
enum
{
        /* Set a FAT entry to FATxx_BAD to mark the cluster as bad. */
        FAT12_BAD = 0x0FF7,
        FAT16_BAD = 0xFFF7,
        FAT32_BAD = 0x0FFFFFF7,
        /* The default End Of Clusterchain marker */
        FAT12_EOC = 0x0FFF,
        FAT16_EOC = 0xFFFF,
        FAT32_EOC = 0x0FFFFFFF,
        /* Special codes for the first byte of a directory entry */
        FAT_FREEENT    = 0xE5, /* The directory entry is free                        */
        FAT_ENDOFDIR = 0x00, /* This and the following entries are free */
        /* Signatures for the FSInfo sector */
        FAT_FSI_SIG1 = 0x41615252,
        FAT_FSI_SIG2 = 0x61417272,
        FAT_FSI_SIG3 = 0xAA550000,
        FAT_FSI_NA     = 0xFFFFFFFF,    /* FSInfo value Not Available */

        FAT_SFN_MAX    = 12, /* Max characters in a short file name (excluding the null terminator) */
};


/* Boot Sector and BIOS Parameter Block for FAT12 and FAT16 */
struct fat16_bpb
{
        /* These fields are common to FAT12, FAT16 and FAT32 */
        uint8_t    jump[3];                    /* assembly JMP to boot code */
        uint8_t    oem_name[8];                /* who formatted the volume */
        uint16_t bytes_per_sector;
        uint8_t    sectors_per_cluster;
        uint16_t reserved_sectors;
        uint8_t    num_fats;
        uint16_t root_entries;              /* size of the FAT12/FAT16 root directory */
        uint16_t num_sectors_16;            /* 0 if it does not fit in 16 bits */
        uint8_t    media_id;
        uint16_t fat_size_16;               /* 0 if it does not fit in 16 bits */
        uint16_t sectors_per_track;
        uint16_t num_heads;
        uint32_t hidden_sectors;
        uint32_t num_sectors_32;            /* if num_sectors_16 is 0 */
        /* The following fields are present also in FAT32, but at offset 64 */
        uint8_t    bios_drive;
        uint8_t    reserved1;
        uint8_t    boot_sig;
        uint32_t serial_number;
        uint8_t    volume_label[11];
        uint8_t    fs_name[8];             /* a descriptive file system name */
} PACKED;


/* Boot Sector and BIOS Parameter Block for FAT32 */
struct fat32_bpb
{
        /* These fields are common to FAT12, FAT16 and FAT32 */
        uint8_t    jump[3];                    /* assembly JMP to boot code */
        uint8_t    oem_name[8];                /* who formatted the volume */
        uint16_t bytes_per_sector;
        uint8_t    sectors_per_cluster;
        uint16_t reserved_sectors;
        uint8_t    num_fats;
        uint16_t root_entries;              /* size of the FAT12/FAT16 root directory */
        uint16_t num_sectors_16;            /* 0 if it does not fit in 16 bits */
        uint8_t    media_id;
        uint16_t fat_size_16;               /* 0 if it does not fit in 16 bits */
        uint16_t sectors_per_track;
        uint16_t num_heads;
        uint32_t hidden_sectors;
        uint32_t num_sectors_32;            /* if num_sectors_16 is 0 */
        /* Here start the FAT32 specific fields (offset 36) */
        uint32_t fat_size_32;             /* if fat_size_16 is 0 */
        uint16_t ext_flags;
        uint16_t fs_version;
        uint32_t root_cluster;
        uint16_t fsinfo_sector;
        uint16_t bootsector_backup; /* sector containing the backup */
        uint8_t    reserved[12];
        /* The following fields are present in a FAT12/FAT16 BPB too,
          * but at offset 36. In a FAT32 BPB they are at offset 64 instead. */
        uint8_t    bios_drive;
```

```
        uint8_t    reserved1;
        uint8_t    boot_sig;
        uint32_t serial_number;
        uint8_t    volume_label[11];
        uint8_t    fs_name[8];             /* a descriptive file system name */
} PACKED;


/* FAT32 FSInfo Sector structure */
struct fat_fsinfo
{
        uint32_t sig1;                    /* FAT_FSI_SIG1 */
        uint8_t    reserved1[480]; /* zero */
        uint32_t sig2;                    /* FAT_FSI_SIG2 */
        uint32_t free_clusters;    /* count of free clusters, or FAT_FSI_NA */
        uint32_t next_free;           /* hint for the next free cluster, or FAT_FSI_NA */
        uint8_t    reserved2[12];    /* zero */
        uint32_t sig3;                    /* FAT_FSI_SIG3 */
} PACKED;


/* 32-byte Directory Entry structure */
struct fat_direntry
{
        uint8_t    name[11];
        uint8_t    attr;
        uint8_t    nt_case;
        uint8_t    cre_time_hund;
        uint16_t cre_time;
        uint16_t cre_date;
        uint16_t acc_date;
        uint16_t first_cluster_hi;
        uint16_t mod_time;
        uint16_t mod_date;
        uint16_t first_cluster_lo;
        uint32_t file_size;
} PACKED;

#endif /* #ifndef __FAT_ONDISK_H */

/* @} */



/* The FreeDOS-32 FAT Driver version 2.0
 * Copyright (C) 2001-2005    Salvatore ISAJA
 *
 * This file "open.c" is part of the FreeDOS-32 FAT Driver (the Program).
 *
 * The 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.
 *
 * The 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 the Program; see the file GPL.txt; if not, write to
 * the Free Software Foundation, Inc.,
 * 59 Temple Place, Suite 330, Boston, MA    02111-1307    USA
 */
/**
 * \file
 * \brief Facilities to open and close files.
 */
/**
 * \addtogroup fat
 * @{
 */
#include "fat.h"


static void file_init(File *f, Volume *v, const struct fat_direntry *de, Sector de_sector, unsigned de_secofs)
{
        memcpy(&f->de, de, sizeof(struct fat_direntry));
        f->de_changed       = false;
        f->de_sector        = de_sector;
        f->de_secofs        = de_secofs;
        f->v                = v;
        f->first_cluster = ((Cluster) de->first_cluster_hi << 16) | (Cluster) de->first_cluster_lo;
        f->references       = 0;
        //list_push_front(&v->files_open, (ListItem *) f);
}


/* Gets and initializes a state structure for a file.
 * If the specified File is already open, increase its reference count.
 * Fot the latter task, comparing the first cluster is not enough, since
 * all zero length files have the first cluster equal to 0.
 */
static File *file_get(Volume *v, const struct fat_direntry *de, Sector de_sector, unsigned de_secofs)
{
        File *f;
//      Cluster first = ((Cluster) de->first_cluster_hi << 16) | (Cluster) de->first_cluster_lo;
//      for (f = (File *) v->files_open.begin; f; f = f->next)
//      {
```

```
//              if (first && (f->first_cluster == first)) break;
//              if (!first && (f->de_sector == de_sector) && (f->de_secofs == de_secofs)) break;
//      }
//      if (!f)
//      {
//              f = slabmem_alloc(&v->files_slab);
                f = (File *) malloc(sizeof(File));
                file_init(f, v, de, de_sector, de_secofs);
//      }
        f->references++;
        return f;
}


/* Releases a state structure for a file. */
static void file_put(File *f)
{
        assert (f->references);
        if (--f->references == 0)
        {
//              list_erase(&f->v->files_open, (ListItem *) f);
//              slabmem_free(&f->v->files_slab, f);
        }
}


/**
 * \brief Increases the reference count of a cached directory node.
 * \param d the cached directory node; the behavior is undefined if it is not valid.
 */
void fat_dget(Dentry *d)
{
        d->references++;
        d->v->num_dentries++;
}


static void dentry_init(Dentry *d, Dentry *parent, unsigned de_entcnt, Sector de_sector, unsigned attr)
{
//      d->prev               = NULL;
//      d->next               = NULL;
        d->parent           = parent;
//      list_init(&d->children);
        d->references    = 0;
        d->v                  = parent->v;
        d->de_sector      = de_sector;
        d->attr              = attr;
        d->de_entcnt      = (uint16_t) de_entcnt;
        fat_dget(parent);
//      list_push_front(&parent->children, (ListItem *) d);
}


/* Gets a cached directory node matching the specified parameters,
 * allocating a new one of there is not already one.
 */
//static Dentry *dentry_get(Dentry *parent, unsigned de_entcnt, Sector de_sector, unsigned attr)
Dentry *dentry_get(Dentry *parent, unsigned de_entcnt, Sector de_sector, unsigned attr)
{
        Dentry *d;
//      for (d = (Dentry *) parent->children.begin; d; d = d->next)
//              if (de_entcnt == d->de_entcnt) break;
//      if (!d)
//      {
//              d = slabmem_alloc(&parent->v->dentries_slab);
                d = (Dentry *) malloc(sizeof(Dentry));
                if (!d) return NULL;
                dentry_init(d, parent, de_entcnt, de_sector, attr);
//      }
        fat_dget(d);
        return d;
}


/**
 * \brief      Releases a cached directory node.
 * \param      d the cached directory node; the behavior is undefined if it is not valid.
 * \remarks The reference count for the cached directory node shall be
 *                decreased. If it reaches zero, the directory node itself shall
 *                be deallocated, and the procedure shall be repeated recursively
 *                for any parent cached directory node of the file system volume.
 */
void fat_dput(Dentry *d)
{
        Dentry *parent;
        Volume *v = d->v;
        while (d)
        {
                assert(d->v == v);
                assert(d->references);
                assert(v->num_dentries);
                d->references--;
                v->num_dentries--;
                if (d->references) break;
//              assert(!d->children.size);
                parent = d->parent;
                if (parent)
                {
//                      list_erase(&parent->children, (ListItem *) d);
```

```
//                      slabmem_free(&v->dentries_slab, d);
                }
                d = parent;
        }
}


static void channel_init(Channel *c, File *f, Dentry *d, int flags)
{
        c->file_pointer      = 0;
        c->f                 = f;
        c->magic             = FAT_CHANNEL_MAGIC;
        c->flags             = flags;
        c->references        = 1;
        c->cluster_index     = 0;
        c->cluster           = 0;
        c->dentry            = d;
//      list_push_front(&f->v->channels_open, (ListItem *) c);
}


/* Allocates and initializes an open instance of a file, and increases
 * the reference count of the associated cached directory node.
 */
static Channel *channel_get(File *f, Dentry *d, int flags)
{
//      Channel *c = slabmem_alloc(&f->v->channels_slab);
        Channel *c = (Channel *) malloc(sizeof(Channel));
        if (c)
        {
                channel_init(c, f, d, flags);
                if (d) fat_dget(d);
        }
        return c;
}

/**
 * \brief    Opens an existing file.
 * \param    dentry    cached directory node of the file to open;
 * \param    flags     opening flags;
 * \param    channel to receive the pointer to the open file description.
 * \return 0 on success, or a negative error (\c channel unchanged).
 * \remarks On success, the cached directory node shall be associated to the
 *                  open file description, thus its reference count shall be increased.
 */
int fat_open(Dentry *dentry, int flags, Channel **channel)
{
        struct fat_direntry de;
        Volume    *v = dentry->v;
        Buffer    *b = NULL;
        File      *f;
        Channel *c;
        int          res;
        unsigned de_secofs = ((off_t) dentry->de_entcnt * sizeof(struct fat_direntry))
                                        & (v->bytes_per_sector - 1);

        /* Check for permissions */
        if ((flags & O_WRONLY) || (flags & O_RDWR) || (flags & O_TRUNC))
        {
                if (dentry->attr & FAT_ARDONLY) return -EACCES;
                if (!(flags & O_DIRECTORY) && (dentry->attr & FAT_ADIR)) return -EISDIR;
        }
        if ((flags & O_DIRECTORY) && !(dentry->attr & FAT_ADIR)) return -ENOTDIR;
        /* Fetch the directory entry, or synthesize one for the root */
        if (dentry->parent)
        {
                res = fat_readbuf(v, dentry->de_sector, &b, false);
                if (res < 0) return res;
                memcpy(&de, b->data + res + de_secofs, sizeof(struct fat_direntry));
        }
        else
        {
                memset(&de, 0, sizeof(struct fat_direntry));
                de.attr = FAT_ADIR;
                de.first_cluster_hi = (uint16_t) (v->root_cluster >> 16);
                de.first_cluster_lo = (uint16_t) v->root_cluster;
        }
        /* Open a file description */
        f = file_get(v, &de, dentry->de_sector, de_secofs);
        if (!f) return -ENOMEM;
        c = channel_get(f, dentry, flags);
        if (!c)
        {
                file_put(f);
                return -ENOMEM;
        }
        assert(dentry->references >= 2);
        *channel = c;
        return 0;
}


#define mode_to_attributes(x) x

/* Opens a directory knowing its first cluster for read, and seeks to
 * the offset corresponding to the specified 32-byte directory entry.
 * Only used for DOS-style FindFirst and FindNext services.
 */
int fat_reopen_dir(Volume *v, Cluster first_cluster, unsigned entry_count, Channel **channel)
```

```
{
        struct fat_direntry de;
        File *f;
        Channel *c;
        memset(&de, 0, sizeof(struct fat_direntry));
        de.attr = FAT_ADIR;
        de.first_cluster_hi = (uint16_t) (first_cluster >> 16);
        de.first_cluster_lo = (uint16_t) first_cluster;
        f = file_get(v, &de, 0, 0);
        if (!f) return -ENOMEM;
        c = channel_get(f, 0, O_RDONLY | O_DIRECTORY); //TODO: Missing Dentry
        if (!c)
        {
                file_put(f);
                return -ENOMEM;
        }
        c->file_pointer = (off_t) entry_count << 5;
        *channel = c;
        return FD32_OROPEN;
}

/**
 * \brief      Backend for the "close" POSIX system call.
 * \param      c open instance of the file to close.
 * \return     0 on success, or a negative error.
 * \remarks If there are no other open instances, the file description and the
 *                  associated cached directory node shall be released even on I/O error.
 */
int fat_close(Channel *c)
{
        File *f;
        Volume *v;
        int res = 0;
        if ((c->magic != FAT_CHANNEL_MAGIC) || !c->references) return -EBADF;
        f = c->f;
        v = f->v;
        c->references--;
      free(c->f);
      free(c->dentry);

        if (!c->references)
        {
//              if (c->dentry) fat_dput(c->dentry);
//              file_put(f);
//              c->magic = 0;
//              list_erase(&v->channels_open, (ListItem *) c);
//              slabmem_free(&v->channels_slab, c);
        }

        return res;
}

/* @} */



/*-------------------------------------------*/
/* File : parse.c, utilities for jfif view */
/* Author : Pierre Guerrier, march 1998              */
/*-------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "jpeg.h"


/*-----------------------------------------------------------------------*/

int input_pointer = 0;

/*Overloading my_get*/
int my_get(FILE *fi)
{


       int temp = input_buffer[input_pointer];
     if ((input_pointer == input_buf_size) )
                return -1;

         input_pointer++;
         return (unsigned int) temp;
}


int my_seek(FILE * fi, int offset, int whence)
{

   input_pointer = input_pointer+ offset;

}

/* utility and counter to return the number of bits from file */
/* right aligned, masked, first bit towards MSB's                */

static unsigned char bit_count;   /* available bits in the window */
static unsigned char window;
```

```
unsigned long
get_bits(FILE *fi, int number)
{
    int i, newbit;
    unsigned long result = 0;
    unsigned char aux, wwindow;

    if (!number)
        return 0;

    for (i = 0; i < number; i++) {
        if (bit_count == 0) {
            wwindow = my_get(fi);

            if (wwindow == 0xFF)
          switch (aux = my_get(fi)) {           /* skip stuffer 0 byte */
          case EOF:
          case 0xFF:
              printf("\tERROR:\tRan out of bit stream\n");
              aborted_stream(fi, fo);
              break;

          case 0x00:
              stuffers++;
              break;

          default:
              if (RST_MK(0xFF00 | aux))
           {
                printf("%ld:\tERROR:\tSpontaneously found restart!\n", ftell(fi));
           }
              printf("%ld:\tERROR:\tLost sync in bit stream\n", ftell(fi));
              aborted_stream(fi, fo);
              break;
          }

            bit_count = 8;
        }
        else wwindow = window;
        newbit = (wwindow>>7) & 1;
        window = wwindow << 1;
        bit_count--;
        result = (result << 1) | newbit;
    }
    return result;
}


void
clear_bits(void)
{
    bit_count = 0;
}


unsigned char
get_one_bit(FILE *fi)
{
    int newbit;
    unsigned char aux, wwindow;

    if (bit_count == 0) {
        wwindow = my_get(fi);

        if (wwindow == 0xFF)
            switch (aux = my_get(fi)) {         /* skip stuffer 0 byte */
            case EOF:
            case 0xFF:
          printf("\tERROR:\tRan out of bit stream\n");
          aborted_stream(fi, fo);
          break;

            case 0x00:
          stuffers++;
          break;

            default:
          if (RST_MK(0xFF00 | aux))
        {
              printf("\tERROR:\tSpontaneously found restart!\n");
        }
          printf("\tERROR:\tLost sync in bit stream\n");
          aborted_stream(fi, fo);
          break;
           }

        bit_count = 8;
    }
    else
        wwindow = window;

    newbit = (wwindow >> 7) & 1;
    window = wwindow << 1;
    bit_count--;
    return newbit;
}

/*-----------------------------------------------------------*/
```

```
unsigned int
get_size(FILE *fi)
{
    unsigned char aux;

    aux = my_get(fi);
    return (aux << 8) | my_get(fi);          /* big endian */
}


/*--------------------------------------------------------------*/


void
skip_segment(FILE *fi)      /* skip a segment we don't want */
{
    unsigned int size;
    char         tag[5];
    int i;

    size = get_size(fi);
    if (size > 5) {
        for (i = 0; i < 4; i++)
            tag[i] = my_get(fi);
        tag[4] = '\0';
        if (verbose)
        {
            printf("\tINFO:\tTag is %s\n", tag);
        }
        size -= 4;
    }
    my_seek(fi, size-2, SEEK_CUR);
}


/*-------------------------------------------------------------------*/
/* find next marker of any type, returns it, positions just after */
/* EOF instead of marker if end of file met while searching ...          */
/*-------------------------------------------------------------------*/

unsigned int
get_next_MK(FILE *fi)
{
    unsigned int c;
    int ffmet = 0;
    int locpassed = -1;

    passed--;  /* as we fetch one anyway */

    while ((c = my_get(fi)) != (unsigned int) EOF) {
        switch (c) {
        case 0xFF:
            ffmet = 1;
            break;
        case 0x00:
            ffmet = 0;
            break;
        default:
            if (locpassed > 1) printf("NOTE: passed %d bytes\n", locpassed);
            if (ffmet)
          return (0xFF00 | c);
            ffmet = 0;
            break;
        }
        locpassed++;
        passed++;
    }

    return (unsigned int) EOF;
}


/*--------------------------------------------------------------*/
/* loading and allocating of quantization table                 */
/* table elements are in ZZ order (same as unpack output)       */
/*--------------------------------------------------------------*/

int
load_quant_tables(FILE *fi)
{
    char aux;
    unsigned int size, n, i, id, x;

    size = get_size(fi); /* this is the tables' size */
    n = (size - 2) / 65;

    for (i = 0; i < n; i++) {
        aux = my_get(fi);
        if (first_quad(aux) > 0) {
            printf("\tERROR:\tBad QTable precision!\n");
            return -1;
        }
        id = second_quad(aux);
        if (verbose)
        {
            printf("\tINFO:\tLoading table %d\n", id);
        }
        QTable[id] = (PBlock *) malloc(sizeof(PBlock));
```

```
        if (QTable[id] == NULL) {
            printf("\tERROR:\tCould not allocate table storage!\n");
            exit(1);
        }
        QTvalid[id] = 1;
        for (x = 0; x < 64; x++)
            QTable[id]->linear[x] = my_get(fi);
            /*
                -- This is useful to print out the table content --
                for (x = 0; x < 64; x++)
                printf("%d\n", QTable[id]->linear[x]);
            */
    }
    return 0;
}


/*-----------------------------------------------------------*/
/* initialise MCU block descriptors                          */
/*-----------------------------------------------------------*/

int
init_MCU(void)
{
    int i, j, k, n, hmax = 0, vmax = 0;

    for (i = 0; i < 10; i++)
        MCU_valid[i] = -1;

    k = 0;

    for (i = 0; i < n_comp; i++) {
        if (comp[i].HS > hmax)
            hmax = comp[i].HS;
        if (comp[i].VS > vmax)
            vmax = comp[i].VS;
        n = comp[i].HS * comp[i].VS;

        comp[i].IDX = k;
        for (j = 0; j < n; j++) {
            MCU_valid[k] = i;
            MCU_buff[k] = (PBlock *) malloc(sizeof(PBlock));
            if (MCU_buff[k] == NULL) {
          printf("\tERROR:\tCould not allocate MCU buffers!\n");
          exit(1);
            }
            k++;
            if (k == 10) {
          printf("\tERROR:\tMax subsampling exceeded!\n");
          return -1;
            }
        }
    }

    MCU_sx = 8 * hmax;
    MCU_sy = 8 * vmax;
    for (i = 0; i < n_comp; i++) {
        comp[i].HDIV = (hmax / comp[i].HS > 1); /* if 1 shift by 0 */
        comp[i].VDIV = (vmax / comp[i].VS > 1);  /* if 2 shift by one */
    }

    mx_size = ceil_div(x_size,MCU_sx);
    my_size = ceil_div(y_size,MCU_sy);
    rx_size = MCU_sx * floor_div(x_size,MCU_sx);
    ry_size = MCU_sy * floor_div(y_size,MCU_sy);

    return 0;
}


/*-----------------------------------------------------------*/
/* this takes care for processing all the blocks in one MCU */
/*-----------------------------------------------------------*/

int
process_MCU(FILE *fi)
{
    int    i;
    long offset;
    int    goodrows, goodcolumns;

    if (MCU_column == mx_size) {
        MCU_column = 0;
        MCU_row++;
        if (MCU_row == my_size) {
            in_frame = 0;
            return 0;
        }
        if (verbose)
        {
              printf("\tINFO:\tProcessing stripe %d/%d\n", MCU_row+1, my_size);
        }
    }

    for (curcomp = 0; MCU_valid[curcomp] != -1; curcomp++) {
        unpack_block(fi, FBuff, MCU_valid[curcomp]); /* pass index to HT,QT,pred */
        IDCT(FBuff, MCU_buff[curcomp]);
    }
```

```
    /* YCrCb to RGB color space transform here */
    if (n_comp > 1)
        color_conversion();
    else
        memmove(ColorBuffer, MCU_buff[0], 64);

    /* cut last row/column as needed */
    if ((y_size != ry_size) && (MCU_row == (my_size - 1)))
        goodrows = y_size - ry_size;
    else
        goodrows = MCU_sy;

    if ((x_size != rx_size) && (MCU_column == (mx_size - 1)))
        goodcolumns = x_size - rx_size;
    else
        goodcolumns = MCU_sx;

    offset = n_comp * (MCU_row * MCU_sy * x_size + MCU_column * MCU_sx);

    for (i = 0; i < goodrows; i++)
        memmove(FrameBuffer + offset + n_comp * i * x_size,
                ColorBuffer + n_comp * i * MCU_sx,
                n_comp * goodcolumns);

    MCU_column++;
    return 1;
}


/*---------------------------------------*/
/* File: spy.c, instrumentation functions */
/*       to monitor activity of JPEG code    */
/* Author: Pierre Guerrier, march 1998    */
/*---------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include "jpeg.h"


static int      loc_code_bits       = 0;    /* catch peak length */
static int      max_code[2]         = { 1 , 1 };      /* Luma then Chroma */

static long     code_bits   = 0;    /* huffman codes */
static long     code_words          = 0;

static long     coef_bits    = 0;    /* non zero dct coefs */
static long     coef_words[2]       = { 0 , 0 };

static long     Luma_bits  = 0;
static long     Chroma_bits         = 0;


/* number of bits */
/* type: digit or code */

void
trace_bits(int number, int type)
{
    if (MCU_valid[curcomp] > 0)
        Chroma_bits += number;
    else
        Luma_bits += number;

    if (type == 0) {                /* code word */
        code_bits += number;
        loc_code_bits += number;
        if (!number) {              /* done with one code word */
            if (loc_code_bits>max_code[MCU_valid[curcomp]>0])
          max_code[MCU_valid[curcomp]>0] = loc_code_bits;
            loc_code_bits = 0;
            code_words ++;
        }
    }
    else {
        coef_bits += number;
        coef_words[MCU_valid[curcomp]>0]++;
    }
}


/* we know number of Luma and chroma blocks from picture size and format */
/* we know total number of coefficients because of fixed block size */
/* we know number of bits in L and C because of counters */
/* hence the averages */

/* we know number of code words because of counters (mind EOB and ZRL) */


void
output_stats(char *filename)
{
    long          tot_coefs, Lblocks, Cblocks, Tcoef_words;
    int n_tot, n_luma, i;

    n_luma = 0;
    for (i=0; MCU_valid[i] != -1; i++)
        if (MCU_valid[i] == 0)
            n_luma = i+1;
```

```
    n_tot = i;
    tot_coefs = 64*mx_size*my_size*n_tot;
    Tcoef_words = coef_words[0] + coef_words[1];

    printf("Traces for %s\n\n\n", filename);

    /* totalizers */
    printf("Total code bits\t\t%ld\n", code_bits);
    printf("Total code words\t%ld\n", code_words);
    printf("Total coeff bits\t%ld\n", coef_bits);
    printf("Total NZ coeff words\tLuma %ld\tChroma %ld\n", coef_words[0], coef_words[1]);
    printf("\nTotal coeff %ld in %ld blocks (%ld Luma, %ld Chroma)\n", tot_coefs, tot_coefs/64, Lblocks = mx_size*my_size*n_luma, Cblocks
= mx_size*my_size*(n_tot-n_luma) );

    /* block averages */
    printf("\n\nAverage block bits\tLuma %f\tChroma %f\n", (double)Luma_bits/Lblocks, (double)Chroma_bits/Cblocks);
    printf("Average NZ coefs/block\tLuma %f\tChroma %f\n", (double)64*coef_words[0]/(Lblocks * 64), (double)64*coef_words[1]/(Cblocks
* 64)    );

    /* code averages */
    printf("Average total bits per NZ coef\t%f\n", (double)(code_bits+coef_bits)/Tcoef_words);
    printf("Peak code length\tLuma %d\tChroma %d\n", max_code[0], max_code[1]);
    printf("\nAverage code length\t%f\n", (double)code_bits/code_words);
    printf("Percent lone words\t%f\n", (double)100*(code_words-Tcoef_words)/code_words);
    printf("Average digits (when some)\t%f\n", (double)coef_bits/Tcoef_words);
}


/*--------------------------------------------*/
/* File : tree_vld.c, utilities for jfif view */
/* Author : Pierre Guerrier, march 1998       */
/*--------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>

#include "jpeg.h"



/*--------------------------------------*/
/* private huffman.c defines and macros */
/*--------------------------------------*/

/* Number of HTable words sacrificed to bookkeeping: */
#define GLOB_SIZE           32

/* Memory size of HTables: */
#define MAX_SIZE(hclass)        ((hclass)?384:64)

/* Available cells, top of storage: */
#define MAX_CELLS(hclass)       (MAX_SIZE(hclass) - GLOB_SIZE)

/* for Huffman tree descent */
/* lower 8 bits are for value/left son */

#define GOOD_NODE_FLAG              0x100
#define GOOD_LEAF_FLAG              0x200
#define BAD_LEAF_FLAG           0x300
#define SPECIAL_FLAG            0x000
#define HUFF_FLAG_MSK           0x300

#define HUFF_FLAG(c)            ((c) & HUFF_FLAG_MSK)
#define HUFF_VALUE(c)           ((unsigned char)( (c) & (~HUFF_FLAG_MSK) ))


/*--------------------------------------*/
/* some static structures for storage   */
/*--------------------------------------*/

static unsigned int  DC_Table0[MAX_SIZE(DC_CLASS)],
                     DC_Table1[MAX_SIZE(DC_CLASS)];

static unsigned int  AC_Table0[MAX_SIZE(AC_CLASS)],
                     AC_Table1[MAX_SIZE(AC_CLASS)];

static unsigned int      *HTable[4] = {
                             &DC_Table0[0], &DC_Table1[0],
                             &AC_Table0[0], &AC_Table1[0]
                         };


/*-------------------------------------------------------*/
/* Loading of Huffman table, with leaves drop ability    */
/*-------------------------------------------------------*/

int load_huff_tables(FILE *fi)
{
    char aux;
    int size, hclass, id;
    int LeavesN, NodesN, CellsN;
    int MaxDepth, i, k, done;
    int NextCellPt;   /* where shall we put next cell */
    int NextLevelPt;  /* where shall node point to */
    unsigned int flag;

    size = get_size(fi); /* this is the tables' size */
```

```
size -= 2;

while (size>0) {

    aux = my_get(fi);
    hclass = first_quad(aux);     /* AC or DC */
    id = second_quad(aux);        /* table no */
    if (id>1) {
        printf("\tERROR:\tBad HTable identity %d!\n",id);
        return -1;
    }
    id = HUFF_ID(hclass, id);
    if (verbose)
        printf("\tINFO:\tLoading Table %d\n", id);
    size--;
    CellsN = NodesN = 1;            /* the root one */
    LeavesN = 0;

    for (i=0; i<MAX_CELLS(hclass); i++)
        HTable[id][i] = SPECIAL_FLAG;   /* secure memory with crash value */

    /* first load the sizes of code elements */
    /* and compute contents of each tree level */
    /* Adress        Content              */
    /* Top           Leaves 0       */
    /* Top-1         Nodes    0     */
    /* .....  .......                */
    /* Top-2k        Leaves k       */
    /* Top-2k-1      Nodes    k     */

    MaxDepth = 0;
    for (i=0; i<16; i++) {
        LeavesN = HTable[id][MAX_SIZE(hclass)-2*i-1] = my_get(fi);
        CellsN = 2*NodesN; /* nodes is old */
        NodesN = HTable[id][MAX_SIZE(hclass)-2*i-2] = CellsN-LeavesN;
        if (LeavesN) MaxDepth = i;
    }
    size-=16;

    /* build root at address 0, then deeper levels at */
    /* increasing addresses until MAX_CELLS reached */

    HTable[id][0] = 1 | GOOD_NODE_FLAG;           /* points to cell _2_ ! */
    /* we give up address one to keep left brothers on even adresses */
    NextCellPt = 2;
    i = 0;              /* this is actually length 1 */

    done = 0;

    while (i<= MaxDepth) {

        /* then load leaves for other levels */
        LeavesN = HTable[id][MAX_SIZE(hclass)-2*i-1];
        for (k = 0; k<LeavesN; k++)
      if (!done) {
          HTable[id][NextCellPt++] = my_get(fi) | GOOD_LEAF_FLAG;
          if (NextCellPt >= MAX_CELLS(hclass)) {
              done = 1;
              printf("\tWARNING:\tTruncating Table at depth %d\n", i+1);
          }
      }
      else my_get(fi);      /* throw it away, just to keep file sync */
        size -= LeavesN;

        if (done || (i == MaxDepth)) { i++; continue; }
        /* skip useless node building */

        /* then build nodes at that level */
        NodesN = HTable[id][MAX_SIZE(hclass)-2*i-2];

        NextLevelPt = NextCellPt+NodesN;
        for (k = 0; k<NodesN; k++) {
      if (NextCellPt >= MAX_CELLS(hclass)) { done = 1; break; }

      flag = ((NextLevelPt|1) >=
              MAX_CELLS(hclass)) ? BAD_LEAF_FLAG : GOOD_NODE_FLAG;
      /* we OR by 1 to check even right brother within range */
      HTable[id][NextCellPt++] = (NextLevelPt/2) | flag;
      NextLevelPt += 2;
        }

        i++;   /* now for next level */
    } /* nothing left to read from file after maxdepth */

    if (verbose)
        printf("\tINFO:\tUsing %d words of table memory\n", NextCellPt);

    /*
        -- this is useful for displaying the uploaded tree --
        for(i=0; i<NextCellPt; i++) {
        switch (HUFF_FLAG(HTable[id][i])) {
        case GOOD_NODE_FLAG:
        fprintf(stderr, "Cell %X: Node to %X and %X\n", i,
        HUFF_VALUE(HTable[id][i])*2,
        HUFF_VALUE(HTable[id][i])*2 +1);
        break;
        case GOOD_LEAF_FLAG:
        fprintf(stderr, "Cell %X: Leaf with value %X\n", i,
        HUFF_VALUE(HTable[id][i]) );
```

```
            break;
            case SPECIAL_FLAG:
            fprintf(stderr, "Cell %X: Special flag\n", i);
            break;
            case BAD_LEAF_FLAG:
            fprintf(stderr, "Cell %X: Bad leaf\n", i);
            break;
            }
            }
            */

    }     /* loop on tables */
    return 0;
}

/*-----------------------------------*/
/* extract a single symbol from file */
/* using specified huffman table ... */
/*-----------------------------------*/

unsigned char
get_symbol(FILE *fi, int select)
{
    int cellPt;

    cellPt = 0; /* this is the root cell */

    while (HUFF_FLAG(HTable[select][cellPt]) == GOOD_NODE_FLAG)
        cellPt = get_one_bit(fi) | (HUFF_VALUE(HTable[select][cellPt])<<1);

    switch (HUFF_FLAG(HTable[select][cellPt])) {
    case SPECIAL_FLAG:
        printf("\tERROR:\tFound forbidden Huffman symbol !\n");
        aborted_stream(fi, fo);

        break;

    case GOOD_LEAF_FLAG:
        return HUFF_VALUE(HTable[select][cellPt]);
        break;

    case BAD_LEAF_FLAG:
        /* how do we fall back in case of truncated tree ? */
        /* suggest we send an EOB and warn */
        return 0;
        break;

    default:
        break;
    }
    return 0;
}


/*-----------------------------------------*/
/* File : utils.c, utilities for jfif view */
/* Author : Pierre Guerrier, march 1998    */
/*-----------------------------------------*/

//#include <io.h>
//#include <system.h>
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>

#include "jpeg.h"
#define RAM_SIZE (2<<16) // in number of words

#define SRAM_BASE 0x2200000
#define VGA_BASE 0x1283000

//#define IOWR_RECT_DATA(base, offset, data) \
    IOWR_16DIRECT(base, (offset)*2, data)

void write_sram( const int address, const int data );
void write_buffer( const unsigned int address, const unsigned int data );

/* Prints a data block in frequency space. */
void
show_FBlock(FBlock *S)
{
    int i,j;

    for (i=0; i<8; i++) {
        for (j=0; j<8; j++)
            printf("\t%d", S->block[i][j]);
        printf("\n");
    }
}

/* Prints a data block in pixel space. */
void
show_PBlock(PBlock *S)
{
    int i,j;

    for (i=0; i<8; i++) {
        for (j=0; j<8; j++)
            printf("\t%d", S->block[i][j]);
```

```
        printf("\n");
    }
}

/* Prints the next 800 bits read from file `fi'. */
void
bin_dump(FILE *fi)
{
    int i;

    for (i=0; i<100; i++) {
        unsigned int bitmask;
        int c = fgetc(fi);

        for (bitmask = 0x80; bitmask; bitmask >>= 1)
            printf("\t%1d", !!(c & bitmask));
        printf("\n");
    }
}


/*-------------------------------------------*/
/* core dump generator for forensic analysis */
/*-------------------------------------------*/

void
suicide(void)
{
    int *P;

    fflush(stdout);
    fflush(stderr);
    P = NULL;
    *P = 1;
}


/*-------------------------------------------*/


void
aborted_stream(FILE *fi, FILE *fo)
{

    printf("\tOops! Aborted Stream\n");

    printf("\tINFO:\tTotal skipped bytes %d, total stuffers %d\n", passed, stuffers);


    if (DEBUG) RGB_save(fo); else free_structures();


    if (DEBUG) suicide(); else return;
}


/*-------------------------------------------------------------*/

/* Returns ceil(N/D). */
int
ceil_div(int N, int D)
{
    int i = N/D;

    if (N > D*i) i++;
    return i;
}


/* Returns floor(N/D). */
int
floor_div(int N, int D)
{
    int i = N/D;

    if (N < D*i) i--;
    return i;
}


/*-------------------------------------------------------------*/

/* For all components reset DC prediction value to 0. */
void
reset_prediction(void)
{
    int i;

    for (i=0; i<3; i++) comp[i].PRED = 0;
}


/*-------------------------------------------------------------*/


/* Transform JPEG number format into usual 2's-complement format. */
int
reformat(unsigned long S, int good)
```

```
{
    int St;

    if (!good)
        return 0;
    St = 1 << (good-1);         /* 2^(good-1) */
    if (S < (unsigned long) St)
        return (S+1+((-1) << good));
    else
        return S;
}


/*-------------------------------------------------------------*/

void
free_structures(void)
{
    int i;

    for (i=0; i<4; i++) if (QTvalid[i]) free(QTable[i]);

    //printf("Freeing memory of FrameBuffer and ColorBuffer\n");
    free(ColorBuffer);
    free(FrameBuffer);

    for (i=0; MCU_valid[i] != -1; i++) free(MCU_buff[i]);
      free(input_buffer);
}


/*-----------------------------------------------*/
/* this is to save final RGB image to disk      */
/* using the sunraster uncompressed format      */
/*-----------------------------------------------*/

void delay( int t )
{
        while( t-- );
}

/* Sun raster header */

typedef struct {
    unsigned long     MAGIC;
    unsigned long     Width;
    unsigned long     Heigth;
    unsigned long     Depth;
    unsigned long     Length;
    unsigned long     Type;
    unsigned long     CMapType;
    unsigned long     CMapLength;
} sunraster;

unsigned int addr =0;
void
RGB_save(FILE *fo)
{
      /*fo = stdout;*/


//          printf ("in RGB save\n");
    sunraster *FrameHeader;
    int i, j, k;
    unsigned long bigendian_value;

    FrameHeader = (sunraster *) malloc(sizeof(sunraster));
    FrameHeader->MAGIC          = 0x59a66a95L;
    FrameHeader->Width          = x_size; //2 * ceil_div(x_size, 2); /* round to 2 more */
    FrameHeader->Heigth         = y_size;
    FrameHeader->Depth          = (n_comp>1) ? 24 : 8;
    FrameHeader->Length         = 0;      /* not required in v1.0 */
    FrameHeader->Type           = 0;      /* old one */
    FrameHeader->CMapType       = 0;      /* truecolor */
    FrameHeader->CMapLength = 0;      /* none */

    /* Frameheader must be in Big-Endian format */
    int sample_rate = 1;
    unsigned int new_width = FrameHeader->Width ;
    unsigned int new_height = FrameHeader->Heigth;

    while ((new_width * new_height)> (RAM_SIZE))
    {
            sample_rate++;
            new_width = FrameHeader->Width /sample_rate;
            new_height = FrameHeader->Heigth/sample_rate;
    }

      //printf("x size: %d ", x_size* n_comp);
      //printf("y size: %d\n", y_size);

/*#if 1

#define MACHINE_2_BIGENDIAN(value)\
  ((( (value) & (unsigned long)(0x000000FF)) << 24) | \
    (( (value) & (unsigned long)(0x0000FF00)) << 8) | \
    (( (value) & (unsigned long)(0x00FF0000)) >> 8) | \
    (( (value) & (unsigned long)(0xFF000000)) >> 24))
```

```
    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->MAGIC);
    fwrite(&bigendian_value, 4, 1, fo);

    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Width);
    fwrite(&bigendian_value, 4, 1, fo);

    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Heigth);
    fwrite(&bigendian_value, 4, 1, fo);

    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Depth);
    fwrite(&bigendian_value, 4, 1, fo);

    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Length);
    fwrite(&bigendian_value, 4, 1, fo);

    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->Type);
    fwrite(&bigendian_value, 4, 1, fo);

    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->CMapType);
    fwrite(&bigendian_value, 4, 1, fo);

    bigendian_value = MACHINE_2_BIGENDIAN(FrameHeader->CMapLength);
    fwrite(&bigendian_value, 4, 1, fo);

#else
    fwrite(FrameHeader, sizeof(sunraster), 1, fo);
#endif*/

// if(FrameHeader->Width * FrameHeader->Heigth < 150 * 150)
//     {
//             printf ("%s","\nPausing");
//               for (i =0; i< 1000; i++)
//                       for (j= 0; j< 10000; j++);
//     }

//*    SJK(0414)
    //IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0003, (0x0001));
    //IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0004, (0x0001));


//IOWR_32DIRECT( VGA_BASE, 0x08, 0xFFFFFFFF );
//int whiteCount = 0;
//unsigned int count =0;
    for (i=0; i<FrameHeader->Heigth; i++)
        for (j=0; j<FrameHeader->Width; j++)
            {
              unsigned int to_out=0;
                  for (k =0; k<n_comp;k++)
                        {
                                unsigned char temp = FrameBuffer[i*x_size*n_comp+j*n_comp+k];// temp is 8 bits
                                to_out = (temp>> 3)<< (k*5) | to_out;
                        }

                        //Address 1 & 2
                    if(i%sample_rate== (sample_rate -1 ) && j%sample_rate==(sample_rate -1))
                    {
                        //IOWR_16DIRECT( SRAM_BASE, addr * 4, to_out );
//                        if(to_out > 0x7000)
//                        {
//                              whiteCount++;
//                              printf( "%02X ", to_out );
//                        }

                        //do
                        //{
                             //write_sram( addr, to_out );
                             write_buffer( addr, to_out );
                             //}
//                        if( *( volatile unsigned int * ) ( SRAM_BASE + addr * 4 ) != ( to_out & 0xFFFF ) )
//                        {
//                              printf( "%02X == %02X\n", *( volatile unsigned int * ) ( SRAM_BASE + addr * 4 ), to_out & 0xFFFF );
//                        }

                    //IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0001,((2*addr)& 0xFFFF));
                    //IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0002,((2*addr)>>16));
                    //IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0000, to_out);
                    addr++;
                    }
                    //count++;
                   // printf("%d ", count);
                    //(FrameBuffer+n_comp*i*x_size, n_comp, FrameHeader->Width, fo);
            }

//       printf( "%d\n",whiteCount);
      //IOWR_32DIRECT( VGA_BASE, 0x08, 0x00 );

//       *( ( volatile unsigned int * ) ( VGA_BASE + 0x0C ) ) = 0xFF;

    //IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0003, (new_width ));
    //IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0004, (new_height));

          // clean_up();
//*/
}

/* The FreeDOS-32 FAT Driver version 2.0
  * Copyright (C) 2001-2005    Salvatore ISAJA
  *
  * This file "volume.c" is part of the FreeDOS-32 FAT Driver (the Program).
```

```
 *
 * The 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.
 *
 * The 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 the Program; see the file GPL.txt; if not, write to
 * the Free Software Foundation, Inc.,
 * 59 Temple Place, Suite 330, Boston, MA    02111-1307    USA
 */
/**
 * \file
 * \brief Low level part of the driver dealing with the hosting block device.
 */
/**
 * \addtogroup fat
 * @{
 */
#include "fat.h"
Volume vol;

/* Buffer flags */
enum
{
        BUF_VALID = 1 << 0, /* Buffer contains valid data */
        BUF_DIRTY = 1 << 1    /* Buffer contains data to be written */
};


/* Searches for a volume buffer contaning the specified sector.
 * Returns the buffer address on success, or NULL on failure.
 */

/* Performs a buffered read from the hosting block device.
 * If the requested sector is already buffered, use that buffer.
 * If the sector is not buffered, read it from the hosting block device:
 * if "buffer" is NULL, flush the least recently used buffer and use it,
 * if "buffer" is not NULL, flush that buffer and use it (this can be useful
 * for sequential access).
 * On success, puts the address of the buffer in "buffer" and returns
 * the byte offset of the requested sector in the data of that buffer.
 * TODO: enable read_through
 */

int fat_readbuf(Volume *v, Sector sector, Buffer **buffer, bool read_through)
{
      int res;
        Buffer *b;
        //v->buf_hit++;
        //b = find_buffer(v, sector);
        //if (!b)
        //{
        //      v->buf_hit--;
        //      v->buf_miss++;
                //b = *buffer;
        //      if (!b) b = (Buffer *) v->buffers_lru.begin;
                b = &v->buffer;       //use temp buffer instead of linked list of buffers
                b->flags = 0;
                b->sector = sector & ~(v->sectors_per_buffer - 1);
                //b->data = (uint8_t *) malloc(v->sectors_per_buffer * v->bytes_per_sector);
                res = fat_blockdev_read(&v->blk, b->data, v->sectors_per_buffer, b->sector);
                if ((res < 0) || (sector >= b->sector + res)) return -EIO;
                b->count = res;
                b->flags = BUF_VALID;
        //}
        //v->buf_access++;
        //list_erase(&v->buffers_lru, (ListItem *) b);
        //list_push_back(&v->buffers_lru, (ListItem *) b);
        *buffer = b;
        return (sector - b->sector) << v->log_bytes_per_sector;
}


/* Collects volume garbage on unmount or mount error */
static void free_volume(Volume *v)
{
        if (v)
        {
                //slabmem_destroy(&v->files_slab);
                //slabmem_destroy(&v->channels_slab);
                /*
                if (v->buffers)
                {
                        if (v->buffer_data)
                                mfree(v->buffer_data, v->bytes_per_sector * v->sectors_per_buffer * v->num_buffers);
                        mfree(v->buffers, sizeof(Buffer) * v->num_buffers);
                }
                */
                //if (v->nls) v->nls->release();
                #if FAT_CONFIG_FD32
                if (v->blk.is_open)
                {
                        v->blk.bops->close(v->blk.handle);
```

```
                v->blk.is_open = false;
            }
            if (v->blk.bops) v->blk.bops->request(REQ_RELEASE);
            #else
            if (v->blk) fclose(v->blk);
            #endif
            v->magic = 0; /* Invalidate signature against bad pointers */
            mfree(v, sizeof(Volume));
        }
}


/* Unmounts a FAT volume releasing its state structure */
/* TODO: Add forced unmount */
int fat_unmount(Volume *v)
{
        #if 0 //FAT_CONFIG_FD32
        int res;
        #endif
        //if (v->channels_open.size) return -EBUSY;
        #if 0 //FAT_CONFIG_FD32
        /* Restore the original device data for the hosting block device */
        res = fd32_dev_replace(v->blk.handle, v->blk.request, v->blk.devid);
        if (res < 0) return res;
        #endif
        free_volume(v);
        LOG_PRINTF(("[FAT2] Volume succesfully unmounted.\n"));
        return 0;
}

/* A simple macro that prints a log error message and aborts the mount function */
#define ABORT_MOUNT(e) { LOG_PRINTF(e); goto abort_mount; }


/* Mounts a FAT volume initializing its state structure */
int fat_mount(const char *blk_name, Volume **volume)
{
        #if FAT_CONFIG_FD32
        BlockMediumInfo bmi;
        #endif
        struct fat16_bpb *bpb;
        Volume    *v = &vol; //NULL;
        uint8_t *secbuf = NULL;
        unsigned block_size = 512;
        Sector     vol_sectors, total_blocks = UINT32_MAX;
        int        res;

        /* Allocate the FAT volume structure */
        //v = (Volume *) malloc(sizeof(Volume));
        if (!v) return -ENOMEM;
        //memset(v, 0, sizeof(Volume));
        v->magic = FAT_VOL_MAGIC;

        /* Initialize files */
        //slabmem_create(&v->dentries_slab, sizeof(Dentry));
        v->root_dentry.v = v;
        v->root_dentry.attr = FAT_ADIR;
        v->root_dentry.references = 1; /* mounted root */
        v->num_dentries = 1; /* mounted root */
        //slabmem_create(&v->files_slab, sizeof(File));
        //slabmem_create(&v->channels_slab, sizeof(Channel));

        /* Open the block device, allocate a sector buffer and read the boot sector */
        res = -EIO;
      //NOT using file system anymore!!
      //MMC INITIALIZATION
      unsigned char arg[4] = { 0x00, 0x00, 0x02, 0x00 };
      mmc_super_init(arg);
      send_clks(8,60);



        //v->blk = fopen(blk_name, "rb");
        //if (!v->blk) ABORT_MOUNT(("[FAT2] Cannot open the disk image\n"));
        res = -ENOMEM;
        secbuf = (uint8_t *) malloc(block_size);
        if (!secbuf) ABORT_MOUNT(("[FAT2] Cannot allocate a sector buffer\n"));
        res = fat_blockdev_read(&v->blk, secbuf, 1, 0);
        if (res < 0)
        {
                res = -EIO;
                ABORT_MOUNT(("[FAT2] Cannot read the boot sector\n"));
        }
        #if 0
        /* Read the first block of the device */
        res = -ENOMEM;
        secbuf = (uint8_t *) malloc(block_size);
        if (!secbuf) ABORT_MOUNT(("[FAT2] Cannot allocate a sector buffer\n"));
        res = fat_blockdev_read(&v->blk, secbuf, 1, 0);
        if (res < 0)
        {
                res = -EIO;
                ABORT_MOUNT(("[FAT2] Cannot read the boot sector\n"));
        }
        #endif

        /* Check if the BPB is valid */
        res = -ENODEV; /* FIXME: Use custom error code for "unknown/invalid file system" */
        bpb = (struct fat16_bpb *) secbuf;
```

```
  if (*((uint16_t *) &secbuf[510]) != 0xAA55)
          ABORT_MOUNT(("[FAT2] Boot sector signature 0xAA55 not found\n"));
  vol_sectors = bpb->num_sectors_16;
  if (!vol_sectors) vol_sectors = bpb->num_sectors_32;
  if (!vol_sectors) ABORT_MOUNT(("[FAT2] Both num_sectors_16 and num_sectors_32 in BPB are zero\n"));
if (vol_sectors > total_blocks)
          ABORT_MOUNT(("[FAT2] num_sectors in BPB larger than block device size: %u > %u\n", vol_sectors, total_blocks));
  switch (bpb->bytes_per_sector)
  {
          case 512 : v->log_bytes_per_sector = 9;    break;
          case 1024: v->log_bytes_per_sector = 10; break;
          case 2048: v->log_bytes_per_sector = 11; break;
          case 4096: v->log_bytes_per_sector = 12; break;
          default: ABORT_MOUNT(("[FAT2] Invalid bytes_per_sector in BPB: %u\n", bpb->bytes_per_sector));
  }

  v->bytes_per_sector = 1 << v->log_bytes_per_sector;
  switch (bpb->sectors_per_cluster)
  {
          case 1    : v->log_sectors_per_cluster = 0; break;
          case 2    : v->log_sectors_per_cluster = 1; break;
          case 4    : v->log_sectors_per_cluster = 2; break;
          case 8    : v->log_sectors_per_cluster = 3; break;
          case 16 : v->log_sectors_per_cluster = 4; break;
          case 32 : v->log_sectors_per_cluster = 5; break;
          case 64 : v->log_sectors_per_cluster = 6; break;
          case 128: v->log_sectors_per_cluster = 7; break;
          default: ABORT_MOUNT(("[FAT2] Invalid sectors_per_cluster in BPB: %u\n", bpb->sectors_per_cluster));
  }
  v->sectors_per_cluster = 1 << v->log_sectors_per_cluster;
  /* The media_id can be 0xF0, 0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF */
  if ((bpb->media_id != 0xF0) && (bpb->media_id < 0xF8))
          ABORT_MOUNT(("[FAT2] Invalid media_id in BPB: %u\n", bpb->media_id));

  v->fat_size = bpb->fat_size_16;
  if (!v->fat_size) v->fat_size = ((struct fat32_bpb *) secbuf)->fat_size_32;
  v->fat_start      = bpb->reserved_sectors;
  v->num_fats       = bpb->num_fats;
  v->root_sector = v->fat_start + v->num_fats * v->fat_size;
  v->root_size      = (bpb->root_entries * sizeof(struct fat_direntry)
                           + (v->bytes_per_sector - 1)) >> v->log_bytes_per_sector;
  v->data_start    = v->root_sector + v->root_size;
  v->data_clusters = (vol_sectors - v->data_start) >> v->log_sectors_per_cluster;

  /* Determine the FAT type */
  if (v->data_clusters < 4085)
  {
          v->fat_type    = FAT12;
          v->fat_read    = fat12_read;
  }
  else if (v->data_clusters < 65525)
  {
          v->fat_type    = FAT16;
          v->fat_read    = fat16_read;
  }
  else
  {
          v->fat_type    = FAT32;
          v->fat_read    = fat32_read;
  }

  /* Initialize buffers */
  res = -ENOMEM;
  //v->num_buffers = FAT_NUM_BUFFERS;
  v->sectors_per_buffer = FAT_READ_AHEAD;
  //v->buffers = (Buffer *) malloc(sizeof(Buffer) * v->num_buffers);
  //if (!v->buffers) ABORT_MOUNT(("[FAT2] Could not allocate buffers\n"));
  //v->buffer_data = (uint8_t *) malloc(v->bytes_per_sector * v->sectors_per_buffer * v->num_buffers);
  //if (!v->buffer_data) ABORT_MOUNT(("[FAT2] Could not allocate buffer data\n"));
  //memset(v->buffers, 0, sizeof(Buffer) * v->num_buffers);
  //for (k = 0, p = v->buffer_data; k < v->num_buffers; k++, p += v->bytes_per_sector * v->sectors_per_buffer)
  //{
  //      v->buffers[k].v = v;
  //      v->buffers[k].data = p;
  //      list_push_back(&v->buffers_lru, (ListItem *) &v->buffers[k]);
  //}

  /* More pracalcs if FAT32 */
  v->serial_number = bpb->serial_number;
  memcpy(v->volume_label, bpb->volume_label, sizeof(v->volume_label));
  v->free_clusters = FAT_FSI_NA;
  v->next_free = FAT_FSI_NA;
  if (v->fat_type == FAT32)
  {
          struct fat32_bpb *bpb32 = (struct fat32_bpb *) secbuf;
          struct fat_fsinfo *fsi;
          Buffer *b = NULL;
          res = -ENODEV; /* FIXME: Use custom error code for "unknown/invalid file system" */
          if (bpb32->fs_version) ABORT_MOUNT(("[FAT2] Unknown FAT32 version in BPB: %04xh\n", bpb32->fs_version));
          res = fat_readbuf(v, bpb32->fsinfo_sector, &b, false);
          if (res < 0) ABORT_MOUNT(("[FAT2] Error reading the FAT32 FSInfo sector\n"));
          fsi = (struct fat_fsinfo *) &b->data[res];
          res = -ENODEV; /* FIXME: Use custom error code for "unknown/invalid file system" */
          //if ((fsi->sig1 != FAT_FSI_SIG1) || (fsi->sig2 != FAT_FSI_SIG2) || (fsi->sig3 != FAT_FSI_SIG3))
          //      ABORT_MOUNT(("[FAT2] Wrong signatures in the FAT32 FSInfo sector\n"));
          v->free_clusters = fsi->free_clusters;
          v->next_free = fsi->next_free;
          v->active_fat = bpb32->ext_flags & 0x0F; /* TODO: FAT mirroring enabled if !(ext_flags & 0x80) */
          v->root_cluster = bpb32->root_cluster;
```

```
		v->serial_number = bpb32->serial_number;
		memcpy(v->volume_label, bpb32->volume_label, sizeof(v->volume_label));
	}

       //allocate memory for the buffer once!!
       (v->buffer).data = (uint8_t *) malloc(v->sectors_per_buffer * v->bytes_per_sector);

	/* Scan free clusters if the value is unknown or invalid */
//	if ((v->free_clusters == FAT_FSI_NA) || (v->free_clusters > v->data_clusters + 1))
//	{
//		LOG_PRINTF(("[FAT2] Free cluster count not available. Scanning the FAT...\n"));
//		res = scan_free_clusters(v);
//		if (res < 0) ABORT_MOUNT(("[FAT2] Error %i while scanning free clusters\n", res));
//		v->free_clusters = (Cluster) res;
//	}
//	LOG_PRINTF(("[FAT2] %u/%u clusters available\n", v->free_clusters, v->data_clusters));
	//res = nls_get("default", OT_NLS_OPERATIONS, (void **) &v->nls);
	//if (res < 0) ABORT_MOUNT(("[FAT2] Could not get NLS operations"));

	#if 0 //FAT_CONFIG_FD32
	/* Request function and DeviceId of the hosting block device are backed up
	  * in v->blk_request and v->blk_devid, so we replace the device. */
	res = fd32_dev_replace(blk_handle, mounted_request, v);
	if (res < 0) ABORT_MOUNT(("[FAT2] Could not replace the block device\n"));
	#endif

	#if FAT_CONFIG_DEBUG
	LOG_PRINTF(("[FAT2] "));
	switch (v->fat_type)
	{
		case FAT12 : LOG_PRINTF(("FAT12")); break;
		case FAT16 : LOG_PRINTF(("FAT16")); break;
		case FAT32 : LOG_PRINTF(("FAT32")); break;
	}
	LOG_PRINTF((" volume successfully mounted on device '%s'\n", blk_name));
	#endif
	mfree(secbuf, block_size);
	*volume = v;
	return 0;

abort_mount:
	if (secbuf) mfree(secbuf, block_size);
	free_volume(v);
	return res;
}



/* Partition types supported by the FAT driver */
static const struct { unsigned id; const char *name; } partition_types[] =
{
	{ 0x01, "FAT12"                                    },
	{ 0x04, "FAT16 up to 32 MB"                      },
	{ 0x06, "FAT16 over 32 MB"                       },
	{ 0x0B, "FAT32"                                    },
	{ 0x0C, "FAT32 using LBA BIOS"                   },
	{ 0x0E, "FAT16 using LBA BIOS"                   },
	{ 0x1B, "Hidden FAT32"                             },
	{ 0x1C, "Hidden FAT32 using LBA BIOS" },
	{ 0x1E, "Hidden VFAT"                              },
	{ 0, 0 }
};

/* Checks if the passed partition signature is supported by the FAT driver.
  * Returns zero if supported, nonzero if not.
  */
int fat_partcheck(unsigned id)
{
	unsigned k;
	for (k = 0; partition_types[k].id; k++)
		if (partition_types[k].id == id)
		{
			LOG_PRINTF(("[FAT2] Partition type is %02xh:%s\n", k, partition_types[k].name));
			return 0;
		}
	return -1;
}


ssize_t fat_blockdev_read(BlockDev *bd, void *data, size_t count, Sector from)
{

	int res;
       res = read_data(from * BYTES_PER_SECTOR, data, count * BYTES_PER_SECTOR);

	//res = fseek(*bd, from * BYTES_PER_SECTOR, SEEK_SET);
	//res = fread(data, BYTES_PER_SECTOR, count, *bd);
	return res;
}



//custom functions for reading files in root directory
//initializes the FAT volume
int fat_init(char *path, Volume **v, Channel **c) {
	int res;
	res = fat_mount(path, v);
	if (res < 0) return res;
```

```
        res = fat_open(&(*v)->root_dentry, O_RDONLY | O_DIRECTORY, c);
        if (res < 0) return res;
        return res;
}

//points to the next file in the root directory
//stores the filename in <filename>
//returns the file size
int fat_nextfile(Volume *v, Channel *c, Channel **c2, char filename[]) {
        //printf("In fat_nextfile\n");
      int res, sfn_length;
        res = fat_do_readdir(c, &v->lud);
      //printf("Directory read result = %d\n", res);
      //fat_close(*c2);

        if (res < 0) return res;
        while (v->lud.cde.attr & FAT_ADIR ||
                  v->lud.cde.attr & FAT_ASYSTEM ||
                  v->lud.cde.attr & FAT_AVOLID) {       //skip directories
                res = fat_do_readdir(c, &v->lud);
                if (res < 0) return res;
        }

        //copy the filename
        //*filename = malloc(v->lud.sfn_length * sizeof(char));
      //if(*filename == NULL)
      //{
      // printf("File allocation failed\n");
      //}
        strncpy(filename, v->lud.sfn, v->lud.sfn_length);
        filename[v->lud.sfn_length] = '\0';

        //open the file
        Dentry *d = dentry_get(&v->root_dentry, v->lud.de_dirofs / sizeof(struct fat_direntry), v->lud.de_sector, v->lud.cde.attr);
        res = fat_open (d, O_RDONLY, c2);
        free(d);
        if (res < 0) return res;
        return v->lud.cde.file_size;
}

void fat_getfilext(char filename[], char ext[]) {
        int len = strlen(filename);
        int i=0, index = 0;
        while (filename[index] != '.' && index < len) index++;
        for (index++; index <= len && i <= 3; index++, i++) {
                ext[i] = filename[index];
        }
        ext[i] = '\0';
}


/* @} */


/* The FreeDOS-32 FAT Driver version 2.0
 * Copyright (C) 2001-2005    Salvatore ISAJA
 *
 * This file "alloc.c" is part of the FreeDOS-32 FAT Driver (the Program).
 *
 * The 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.
 *
 * The 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 the Program; see the file GPL.txt; if not, write to
 * the Free Software Foundation, Inc.,
 * 59 Temple Place, Suite 330, Boston, MA    02111-1307    USA
 */
/**
 * \addtogroup fat
 * @{
 */
/**
 * \file
 * \brief Manages allocation of clusters and the File Allocation Table.
 */
#include "fat.h"


/*****************************************************************************
 * Seek functions.
 * Find the location of the "n"-th cluster entry in the file allocation table.
 * On success, put the number of the sector containing the entry in "sector"
 * and return the byte offset of the entry in that sector.
 *****************************************************************************/

static int fat12_seek(Volume *v, Cluster n, unsigned fat_num, Sector *sector)
{
        Cluster offset;
        if (n > v->data_clusters + 1) return -ENXIO;
        offset = n + (n >> 1); /* floor(n * 1.5) */
        *sector = fat_num * v->fat_size + v->fat_start + (offset >> v->log_bytes_per_sector);
        return offset & (v->bytes_per_sector - 1); /* % bytes_per_sector */
```

```
}


static int fat16_seek(Volume *v, Cluster n, unsigned fat_num, Sector *sector)
{
        Cluster offset;
        if (n > v->data_clusters + 1) return -ENXIO;
        offset = n << 1;
        *sector = fat_num * v->fat_size + v->fat_start + (offset >> v->log_bytes_per_sector);
        return offset & (v->bytes_per_sector - 1); /* % bytes_per_sector */
}


static int fat32_seek(Volume *v, Cluster n, unsigned fat_num, Sector *sector)
{
        Cluster offset;
        if (n > v->data_clusters + 1) return -ENXIO;
        offset = n << 2;
        *sector = fat_num * v->fat_size + v->fat_start + (offset >> v->log_bytes_per_sector);
        return offset & (v->bytes_per_sector - 1); /* % bytes_per_sector */
}


/*************************************************************************
 * Read functions.
 * Read the value of the specified cluster entry of the file allocation table.
 * On success, return the not negative entry value.
 *************************************************************************/


/**
 * \brief    Reads a cluster entry from the active copy of the FAT.
 * \param    v the volume the FAT to read belongs to;
 * \param    n number of the cluster entry to read.
 * \return The non-negative value of the cluster entry, or a negative error.
 */
static int32_t read_fat_entry(Volume *v, Cluster n)
{
        return v->fat_read(v, n, v->active_fat);
}


int32_t fat12_read(Volume *v, Cluster n, unsigned fat_num)
{
        int         res, offset;
        Sector    sector;
        Cluster c;
        Buffer *b = NULL;

        offset = fat12_seek(v, n, fat_num, &sector);
        if (offset < 0) return offset;
        res = fat_readbuf(v, sector, &b, false);
        if (res < 0) return res;
        if (offset < v->bytes_per_sector - 1) {
                //c = (Cluster) *((uint16_t *) &b->data[offset + res]);
            c = (Cluster) b->data[offset + res] + ((Cluster) b->data[offset + res+1] << 8) ;
            if (c==0) {
                c = (Cluster) *((uint16_t *) &b->data[offset + res]);
              //        if (c==0)
                 //          c =     *((uint16_t *) &b->data[offset + res]);
                  //c = (Cluster) b->data[offset + res];
                  printf("->-> zero 1, c=%d, offset=%d, res=%d\n", c, offset, res);
            }

       }
        else /* the entry spans across two sectors */
        {
                Buffer *b2 = NULL;
                int res2 = fat_readbuf(v, sector + 1, &b2, false);
                if (res2 < 0) return res2;
                c = (Cluster) b->data[offset + res] + ((Cluster) b2->data[res2] << 8);
                 if (c==0) {
                        printf("->-> zero 1, c=%d, offset=%d, res=%d\n", c, offset, res);
            }


        }
        /* If the entry number is odd we need the highest 12 bits of "c",
           * whereas if it's even we need the lowest 12 bits. */
        if (n & 1) c >>= 4; else c &= 0x0FFF;
           // printf("Value of    c =%d", c);
        if (IS_FAT12_EOC(c)) c = FAT_EOC;
     // printf("Value of    c =%d", c);

        return c;
}


int32_t fat16_read(Volume *v, Cluster n, unsigned fat_num)
{
        int         res, offset;
        Sector    sector;
        Cluster c;
        Buffer *b = NULL;

        offset = fat16_seek(v, n, fat_num, &sector);
        if (offset < 0) return offset;
        res = fat_readbuf(v, sector, &b, false);
        if (res < 0) return res;
```

```
        c = (Cluster) *((uint16_t *) &b->data[offset + res]);
        if (IS_FAT16_EOC(c)) c = FAT_EOC;
        return c;
}


int32_t fat32_read(Volume *v, Cluster n, unsigned fat_num)
{
        int          res, offset;
        Sector    sector;
        Cluster c;
        Buffer *b = NULL;

        offset = fat32_seek(v, n, fat_num, &sector);
        if (offset < 0) return offset;
        res = fat_readbuf(v, sector, &b, false);
        if (res < 0) return res;
        c = (Cluster) *((uint32_t *) &b->data[offset + res]) & 0x0FFFFFFF;
        if (IS_FAT32_EOC(c)) c = FAT_EOC;
        return c;
}


/**
 * \brief      Gets the address of a sector of a file.
 * \param      c                    the channel to get the sector of;
 * \param      sector_index index of the sector of the file to get the address of;
 * \param      sector             to receive the address of the sector corresponding to \c sector_index;
 * \return     0 on success, >0 on EOF, or a negative error.
 * \remarks \c c->cluster_index and \c c->cluster are updated to cache the address of the last
 *                  cluster of the file reached by this function. On EOF, they relate to the last
 *                  cluster of the file, if any.
 */
int fat_get_file_sector(Channel *c, Sector sector_index, Sector *sector)
{
        int res = 0;
        File      *f = c->f;
        Volume *v = f->v;
        if (!f->de_sector && !f->first_cluster) /* FAT12/FAT16 root directory */
        {
                if (sector_index >= v->root_size) return 1;
                *sector = sector_index + v->root_sector;
        }
        else /* File/directory with linked allocation */
        {
                Cluster    cluster_index          = sector_index >> v->log_sectors_per_cluster;
                unsigned sector_in_cluster = sector_index & (v->sectors_per_cluster - 1);
                if (!f->first_cluster) return 1;
                if (!c->cluster_index || (cluster_index < c->cluster_index))
                {
                        c->cluster_index = 0;
                        c->cluster = f->first_cluster;
                }
                while (c->cluster_index < cluster_index)
                {
                        int32_t nc = read_fat_entry(v, c->cluster);
                        if (nc < 0) {
                         printf("2. NC= %d\n", nc);
                         return nc;
                   }
                        if (nc == FAT_EOC) return 1;
                        if ((nc < 2) || (nc > v->data_clusters + 1)) {
                         printf("3. nc = %d, v->data_clusters = %d\n", nc, v->data_clusters);
                         return -EIO;
                   }
                        c->cluster_index++;
                        c->cluster = (Cluster) nc;
                }
                *sector = ((c->cluster - 2) << v->log_sectors_per_cluster)
                                + v->data_start + sector_in_cluster;
        }
        return res;
}

/* @} */



/*---------------------------------------*/
/* File : color.c, utilities for jfif view */
/* Author : Pierre Guerrier, march 1998                */
/*---------------------------------------*/

#include <stdlib.h>
#include <stdio.h>

#include "jpeg.h"

/* Ensure number is >=0 and <=255                                */
#define Saturate(n)((n) > 0 ? ((n) < 255 ? (n) : 255) : 0)

/*-------------------------------------------------------------*/


/* internal color conversion utility */
/*
static unsigned char
get_comp(int n, int i, int j)
{
```

```
    int ip = i >> comp[n].VDIV;      / * get coordinates in n-th comp pixels * /
    int jp = j >> comp[n].HDIV;      / * within the MCU * /

    return MCU_buff[comp[n].IDX+comp[n].HS*(ip>>3)+(jp>>3)]->block[ip&7][jp&7];
    / * this is the right block in MCU, and this right sample * /
}
*/
/* and alternate macro, a little faster */
/*
#define get_comp(t,n,i,j) { int ip = i >> comp[n].VDIV; \
                            int jp = j >> comp[n].HDIV; \
t = MCU_buff[comp[n].IDX+comp[n].HS*(ip>>3)+(jp>>3) ]->block[ip&7][jp& 7]; \
                          }
*/

/*----------------------------------------*/
/* rules for color conversion:                      */
/*    r = y              +1.402          v          */
/*    g = y -0.34414u          -0.71414v       */
/*    b = y +1.772    u                           */
/* Approximations: 1.402 # 7/5 = 1.400   */
/*                 .71414 # 357/500 = 0.714 */
/*                 .34414 # 43/125      = 0.344          */
/*                 1.772    = 443/250    */
/*----------------------------------------*/
/* Approximations: 1.402 # 359/256 = 1.40234 */
/*                 .71414 # 183/256 = 0.71484 */
/*                 .34414 # 11/32 = 0.34375 */
/*                 1.772 # 227/128 = 1.7734 */
/*-------------------------------------*/

void
color_conversion(void)
{
    int    i, j;
    unsigned char y,cb,cr;
    signed char rcb, rcr;
    long r,g,b;
    long offset;

    for (i = 0; i < MCU_sy; i++)      /* pixel rows */
       {
          int ip_0 = i >> comp[0].VDIV;
          int ip_1 = i >> comp[1].VDIV;
          int ip_2 = i >> comp[2].VDIV;
          int inv_ndx_0 = comp[0].IDX + comp[0].HS * (ip_0 >> 3);
          int inv_ndx_1 = comp[1].IDX + comp[1].HS * (ip_1 >> 3);
          int inv_ndx_2 = comp[2].IDX + comp[2].HS * (ip_2 >> 3);
          int ip_0_lsbs = ip_0 & 7;
          int ip_1_lsbs = ip_1 & 7;
          int ip_2_lsbs = ip_2 & 7;
          int i_times_MCU_sx = i * MCU_sx;

          for (j = 0; j < MCU_sx; j++)      /* pixel columns */
         {
             int jp_0 = j >> comp[0].HDIV;
             int jp_1 = j >> comp[1].HDIV;
             int jp_2 = j >> comp[2].HDIV;

             y    = MCU_buff[inv_ndx_0 + (jp_0 >> 3)]->block[ip_0_lsbs][jp_0 & 7];
             cb = MCU_buff[inv_ndx_1 + (jp_1 >> 3)]->block[ip_1_lsbs][jp_1 & 7];
             cr = MCU_buff[inv_ndx_2 + (jp_2 >> 3)]->block[ip_2_lsbs][jp_2 & 7];

             rcb = cb - 128;
             rcr = cr - 128;

             r = y + ((359 * rcr) >> 8);
             g = y - ((11 * rcb) >> 5) - ((183 * rcr) >> 8);
             b = y + ((227 * rcb) >> 7);

             offset = 3 * (i_times_MCU_sx + j);
             ColorBuffer[offset + 2] = Saturate(r);
             ColorBuffer[offset + 1] = Saturate(g);
             ColorBuffer[offset + 0] = Saturate(b);
             /* note that this is SunRaster color ordering */
         }
       }
}



/* The FreeDOS-32 FAT Driver version 2.0
 * Copyright (C) 2001-2005    Salvatore ISAJA
 *
 * This file "dir.c" is part of the FreeDOS-32 FAT Driver (the Program).
 *
 * The 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.
 *
 * The 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 the Program; see the file GPL.txt; if not, write to
 * the Free Software Foundation, Inc.,
```

```
 * 59 Temple Place, Suite 330, Boston, MA    02111-1307    USA
 */
/**
 * \file
 * \brief Facilities to access directories.
 */
/**
 * \addtogroup fat
 * @{
 */
#include "fat.h"


/************************************************************************
 *
 * Facilities to manage short file names
 *
 ************************************************************************/

static const char *invalid_sfn_characters = "\"+,./:;<=>[\\]|*?";

/**
 * \brief Expands a file name in FCB format to a wide character string.
 * \param nls             NLS operations to use for character set conversion;
 * \param dest            array to hold the converted name, not null terminated;
 * \param dest_size max number of wide characters to write in \c dest;
 * \param source        11-bytes array holding the name in FCB format to convert.
 * \return The length in wide characters of the expanded name, or a negative error.
 */
//static int expand_fcb_name(const struct nls_operations *nls, wchar_t *dest, size_t dest_size, const uint8_t *source)
static int expand_fcb_name(char *dest, size_t dest_size, const uint8_t *source)
{
        const uint8_t *name_end, *ext_end;
        const uint8_t *s = source;
        char aux[13], *a = aux;
        int res;

        /* Skip padding spaces at the end of the name and the extension */
        for (name_end = source + 7; *name_end == ' '; name_end--)
                if (name_end == source) return -EINVAL;
        for (ext_end = source + 10; *ext_end == ' '; ext_end--)
                if (ext_end == source) return -EINVAL;

        /* Copy name dot extension in aux */
        if (*s == 0x05) *a++ = (char) FAT_FREEENT, s++;
        for (; s <= name_end; *a++ = (char) *s++);
        if (source + 8 <= ext_end) *a++ = '.';
        for (s = source + 8; s <= ext_end; *a++ = (char) *s++);
        *a = 0;

        /* Convert aux to a wide character string */
        for (a = aux, res = 0; *a; dest++, dest_size--, res++)
        {
                int skip;
                if (!dest_size) return -ENAMETOOLONG;
//              skip = nls->mbtowc(dest, a, aux + sizeof(aux) - a);
                //added manually
                *dest = (unsigned char) *a;
                skip = 1;
                //end manual add
                if (skip < 0) return skip;
                a += skip;
        }
        return res;
}

/************************************************************************
 *
 * Facilities to read directory entries
 *
 ************************************************************************/


/**
 * \brief      Computes the location of a directory entry.
 * \param      c      the directory containing the directory entry to locate;
 * \param      lud LookupData structure to update with the directory entry location.
 * \remarks It is assumed that this function is called right after reading or
 *                writing the directory entry. Updates the \c de_dirofs,
 *                \c de_sector and \c de_secofs fields of \c lud.
 */
static void direntry_location(const Channel *c, LookupData *lud)
{
        const File      *f = c->f;
        const Volume *v = f->v;
        lud->de_dirofs = c->file_pointer - sizeof(struct fat_direntry);
        lud->de_sector = lud->de_dirofs >> v->log_bytes_per_sector;
        if (!f->de_sector && !f->first_cluster)
                lud->de_sector += v->root_sector;
        else
                lud->de_sector = (lud->de_sector & (v->sectors_per_cluster - 1))
                                        + ((c->cluster - 2) << v->log_sectors_per_cluster) + v->data_start;
        lud->de_secofs = lud->de_dirofs & (v->bytes_per_sector - 1);
}


/**
 * \brief    Backend for the readdir and find services.
 * \param    c      the open instance of the directory to read;
```

```
 * \param    lud LookupData structure to fill with the read data.
 * \return 0 on success, or a negative error.
 */
int fat_do_readdir(Channel *c, LookupData *lud)
{
        File     *f = c->f;
        Volume *v = f->v;
        if (!(f->de.attr & FAT_ADIR)) return -EBADF;
        for (;;)
        {
                struct fat_direntry *de = &lud->cde;
                int num_read = fat_read(c, de, sizeof(struct fat_direntry));
                if (num_read < 0) return num_read;
                if (!num_read) break; /* EOF */
                if (num_read != sizeof(struct fat_direntry)) return -EFTYPE; /* malformed directory */
                if (de->name[0] == FAT_ENDOFDIR) break;
                if ((de->name[0] != FAT_FREEENT) && (!(de->attr & FAT_AVOLID) || (de->attr == FAT_AVOLID)))
                {
                        int res;
//                      res = expand_fcb_name(v->nls, lud->sfn, FAT_SFN_MAX, de->name);
                        res = expand_fcb_name(lud->sfn, FAT_SFN_MAX, de->name);
                        if (res < 0) return res;
                        lud->sfn_length = res;
                        direntry_location(c, lud);
                        return 0;
                }

        }
        return -ENMFILE;
}

/* @} */


/*-----------------------------------------------*/
/* File : fast_idct.c, utilities for jfif view */
/* Author : Pierre Guerrier, march 1998                 */
/* IDCT code by Geert Janssen                   */
/*-----------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>

#include "jpeg.h"

#define Y(i,j)          Y[8*i+j]
#define X(i,j)          (output->block[i][j])

/* This version is IEEE compliant using 16-bit arithmetic. */

/* The number of bits coefficients are scaled up before 2-D IDCT: */
#define S_BITS                  3
/* The number of bits in the fractional part of a fixed point constant: */
#define C_BITS                  14

#define SCALE(x,n) ((x) << (n))

/* This version is vital in passing overall mean error test. */
#define DESCALE(x, n)       (((x) + (1 << ((n)-1)) - ((x) < 0)) >> (n))

#define ADD(x, y)   ((x) + (y))
#define SUB(x, y)   ((x) - (y))
#define CMUL(C, x)(((C) * (x) + (1 << (C_BITS-1))) >> C_BITS)

/* Butterfly: but(a,b,x,y) = rot(sqrt(2),4,a,b,x,y) */
#define but(a,b,x,y)          { x = SUB(a,b); y = ADD(a,b); }

/* Inverse 1-D Discrete Cosine Transform.
     Result Y is scaled up by factor sqrt(8).
     Original Loeffler algorithm.
*/
static void
idct_1d(int *Y)
{
   int z1[8], z2[8], z3[8];

   /* Stage 1: */
   but(Y[0], Y[4], z1[1], z1[0]);
   /* rot(sqrt(2), 6, Y[2], Y[6], &z1[2], &z1[3]); */
   z1[2] = SUB(CMUL( 8867, Y[2]), CMUL(21407, Y[6]));
   z1[3] = ADD(CMUL(21407, Y[2]), CMUL( 8867, Y[6]));
   but(Y[1], Y[7], z1[4], z1[7]);
   /* z1[5] = CMUL(sqrt(2), Y[3]);
        z1[6] = CMUL(sqrt(2), Y[5]);
   */
   z1[5] = CMUL(23170, Y[3]);
   z1[6] = CMUL(23170, Y[5]);

   /* Stage 2: */
   but(z1[0], z1[3], z2[3], z2[0]);
   but(z1[1], z1[2], z2[2], z2[1]);
   but(z1[4], z1[6], z2[6], z2[4]);
   but(z1[7], z1[5], z2[5], z2[7]);

   /* Stage 3: */
   z3[0] = z2[0];
```

```
    z3[1] = z2[1];
    z3[2] = z2[2];
    z3[3] = z2[3];
    /* rot(1, 3, z2[4], z2[7], &z3[4], &z3[7]); */
    z3[4] = SUB(CMUL(13623, z2[4]), CMUL( 9102, z2[7]));
    z3[7] = ADD(CMUL( 9102, z2[4]), CMUL(13623, z2[7]));
    /* rot(1, 1, z2[5], z2[6], &z3[5], &z3[6]); */
    z3[5] = SUB(CMUL(16069, z2[5]), CMUL( 3196, z2[6]));
    z3[6] = ADD(CMUL( 3196, z2[5]), CMUL(16069, z2[6]));

    /* Final stage 4: */
    but(z3[0], z3[7], Y[7], Y[0]);
    but(z3[1], z3[6], Y[6], Y[1]);
    but(z3[2], z3[5], Y[5], Y[2]);
    but(z3[3], z3[4], Y[4], Y[3]);
}

/* Inverse 2-D Discrete Cosine Transform. */
void
IDCT(const FBlock *input, PBlock *output)
{
    int Y[64];
    int k,l;

    /* Pass 1: process rows. */
    for (k = 0; k < 8; k++) {

        /* Prescale k-th row: */
        for (l = 0; l < 8; l++)
            Y(k,l) = SCALE(input->block[k][l], S_BITS);

        /* 1-D IDCT on k-th row: */
        idct_1d(&Y(k,0));
        /* Result Y is scaled up by factor sqrt(8)*2^S_BITS. */
    }

    /* Pass 2: process columns. */
    for (l = 0; l < 8; l++) {
        int Yc[8];

        for (k = 0; k < 8; k++) Yc[k] = Y(k,l);
        /* 1-D IDCT on l-th column: */
        idct_1d(Yc);
        /* Result is once more scaled up by a factor sqrt(8). */
        for (k = 0; k < 8; k++) {
            int r = 128 + DESCALE(Yc[k], S_BITS+3); /* includes level shift */

            /* Clip to 8 bits unsigned: */
            r = r > 0 ? (r < 255 ? r : 255) : 0;
            X(k,l) = r;
        }
    }
}



/* The FreeDOS-32 FAT Driver version 2.0
 * Copyright (C) 2001-2005    Salvatore ISAJA
 *
 * This file "fat.h" is part of the FreeDOS-32 FAT Driver (the Program).
 *
 * The 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.
 *
 * The 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 the Program; see the file GPL.txt; if not, write to
 * the Free Software Foundation, Inc.,
 * 59 Temple Place, Suite 330, Boston, MA    02111-1307    USA
 */
/**
 * \file
 * \brief Declarations of the facilities provided by the FAT driver.
 */
/**
 * \defgroup fat FAT file system driver
 *
 * The FreeDOS-32 FAT driver is a highly portable software to gain access to
 * media formatted with the FAT file system.
 *
 * @{ */
#ifndef __FD32_FAT_DRIVER_H
#define __FD32_FAT_DRIVER_H

/* Compile time options for the FAT driver */
//#define FAT_CONFIG_LFN              1 /* Enable Long File Names */
#define FAT_CONFIG_LFN              0 /* Disable Long File Names */
#define FAT_CONFIG_WRITE            0 /* Disable write facilities */
//#define FAT_CONFIG_REMOVABLE 1 /* Enable support for media change */
#define FAT_CONFIG_REMOVABLE 0 /* Disable support for media change */
//#define FAT_CONFIG_FD32             1 /* Enable FD32 devices */
#define FAT_CONFIG_FD32             0 /* Disable FD32 devices */
#define FAT_CONFIG_DEBUG            0 /* Enable log output */
```

```
#define BYTES_PER_SECTOR 512

// copied from <stdint.h>
typedef signed char          int8_t;
typedef short int            int16_t;
typedef int                  int32_t;
typedef unsigned char        uint8_t;
typedef unsigned short int   uint16_t;
typedef unsigned int         uint32_t;
# define UINT32_MAX          (4294967295U)


#include <stdio.h>
//#include <sys/types.h>
#include <errno.h>
//#define EFTYPE    2000
//#define ENMFILE 2001
#define __USE_GNU
#define O_NOATIME (1 << 31)
//#include <string.h>
#include "fcntl.h"
//#include <stdint.h>
//#include <time.h>
//#include <sys/time.h>
//#include <sys/stat.h>
#define mfree(p, size) free(p)
#define FD32_OROPEN    1
#define FD32_ORCREAT 2
#define FD32_ORTRUNC 3
#if FAT_CONFIG_DEBUG
  #define LOG_PRINTF(s) printf s
#else
  #define LOG_PRINTF(s)
#endif

#ifndef __cplusplus
typedef enum { false = 0, true = !false } bool;
#endif
//#include <nls/nls.h>
//#include <unicode/unicode.h>
//#include <filesys.h>
//#include "list.h"
//#include <slabmem.h>
#include "ondisk.h"
#include <assert.h>


#if FAT_CONFIG_FD32
typedef struct BlockDev BlockDev;
struct BlockDev
{
        BlockOperations *bops;
        void *handle;
        bool is_open;
};
#else
typedef FILE* BlockDev;
#endif


/* Macros to check whether or not an open file is readable or writable */
#if 1 /* Old-style, zero-based access modes (O_RDONLY = 0, O_WRONLY = 1, O_RDWR = 2) */
  #define IS_NOT_READABLE(c)    (((c->flags & O_ACCMODE) != O_RDONLY) && ((c->flags & O_ACCMODE) != O_RDWR))
  #define IS_NOT_WRITEABLE(c) (((c->flags & O_ACCMODE) != O_RDWR) && ((c->flags & O_ACCMODE) != O_WRONLY))
#else /* New-style, bitwise-distinct access modes (O_RDWR = O_RDONLY | O_WRONLY) */
  #define IS_NOT_READABLE(c)    (!(c->flags & O_RDONLY))
  #define IS_NOT_WRITEABLE(c) (!(c->flags & O_WRONLY))
#endif


/* The character to use as path component separator */
/* TODO: Replace the backslash with a forward slash for internal operation. Use backslash as a quotation character. */
#define FAT_PATH_SEP '\\'

enum
{
        FAT_VOL_MAGIC = 0x46415456, /* "FATV": valid FAT volume signature */
        FAT_CHANNEL_MAGIC = 0x46415446, /* "FATF": valid FAT file signature */
        FAT_EOC = 0x0FFFFFFF,
        /* TODO: these should be command line options */
        FAT_NUM_BUFFERS = 30,
        FAT_READ_AHEAD    = 8,
        FAT_UNLINKED = 1 /* value of de_sector if file unlinked */
};


typedef enum { FAT12, FAT16, FAT32 } FatType;
typedef uint32_t Sector;
typedef uint32_t Cluster;
typedef struct LookupData LookupData;
typedef struct Buffer          Buffer;
typedef struct Dentry          Dentry;
typedef struct Volume          Volume;
typedef struct File            File;
typedef struct Channel         Channel;


/// Lookup data block for the internal "readdir" function.
```

```
struct LookupData
{
//      wchar_t    sfn[FAT_SFN_MAX];
        char   sfn[FAT_SFN_MAX];
        unsigned sfn_length;
        struct fat_direntry cde;
        Sector      de_sector;
        unsigned de_secofs;
        off_t       de_dirofs;
};


/// A buffer containing one or more contiguous sectors.
struct Buffer
{
//      Buffer    *prev;      /* From ListItem, less recently used */
//      Buffer    *next;      /* From ListItem, more recently used */
        Volume    *v;          /* Volume owning this buffer */
        Sector     sector; /* First sector of this buffer */
        unsigned count;    /* Sectors in this buffer */
        int          flags;   /* Buffer state */
        uint8_t *data;      /* Raw data of the buffered sectors */
};


/// A node in the cached directory tree, to locate directory entries.
struct Dentry
{
//      Dentry    *prev;              /* From ListItem, previous at the same level */
//      Dentry    *next;              /* From ListItem, next at the same level */
        Dentry    *parent;            /* The parent Dentry */
//      List        children;       /* List of children of this node */
        unsigned references;    /* Number of Channels and Dentries referring to this Dentry */
        Volume    *v;                   /* FAT volume containing this Dentry */
        Sector      de_sector;      /* Sector containing the directory entry.
                                       * 0 if no directory entry (root directory),
                                       * FAT_UNLINKED if referring to an unlinked file. */
        uint16_t de_entcnt;     /* Offset of the short name entry in the parent in sizeof(struct fat_direntry) units */
        uint8_t    attr;            /* Attributes of the directory entry */
        uint8_t    lfn_entries; /* Number of LFN entries occupied by this entry. Undefined if !FAT_CONFIG_LFN */
};


/// A structure storing the state of a mounted FAT volume.
struct Volume
{
        BlockDev blk;
        //struct nls_operations *nls;

        /* Some precalculated data */
        uint32_t magic; /* FAT_VOL_MAGIC */
        FatType    fat_type;
        unsigned num_fats;
        unsigned bytes_per_sector;
        unsigned log_bytes_per_sector;
        unsigned sectors_per_cluster;
        unsigned log_sectors_per_cluster;
        unsigned active_fat;
        uint32_t serial_number;
        uint8_t    volume_label[11];
        Sector      fat_size;
        Sector      fat_start;
        Sector      root_sector;
        Sector      root_size;
        Sector      data_start;
        Cluster    root_cluster;
        Cluster    data_clusters;
        Cluster    free_clusters;
        Cluster    next_free;

        /* Functions to access the file allocation table */
        int32_t (*fat_read) (Volume *v, Cluster n, unsigned fat_num);

        /* Buffers */
        unsigned    sectors_per_buffer;
//      unsigned    buf_access; /* statistics */
//      unsigned    buf_miss;      /* statistics */
//      unsigned    buf_hit;       /* statistics */
//      unsigned    num_buffers;
//      Buffer      *buffers;
        Buffer       buffer;   //a single buffer
//      List          buffers_lru;

        /* Files */
        unsigned    num_dentries; /* statistics */
//      slabmem_t dentries_slab;
//      slabmem_t files_slab;
//      slabmem_t channels_slab;
        Dentry       root_dentry;
//      List          files_open;
//      List          channels_open;

        /* A per-volume LookupData to avoid using too much stack */
        LookupData lud;
};


/// The state of a file (shared by open instances).
struct File
```

```
{
//      File *prev; /* from ListItem */
//      File *next; /* from ListItem */
        struct fat_direntry de;
        bool        de_changed;
        Sector      de_sector; /* Sector containing the directory entry.
                                * 0 if no directory entry (root directory),
                                * FAT_UNLINKED if file queued for deletion. */
        unsigned de_secofs; /* Byte offset of the directory entry in de_sector */
        Volume    *v;
        Cluster    first_cluster;
        unsigned references;
};


/// An open instance of a file (called a "channel" in glibc documentation).
struct Channel
{
//      Channel    *prev;                /* From ListItem */
//      Channel    *next;                /* From ListItem */
        off_t       file_pointer;    /* The one set by lseek and updated on r/w */
        File       *f;                   /* State of this file */
        uint32_t    magic;               /* FAT_CHANNEL_MAGIC */
        int         flags;              /* Opening flags of this file instance */
        unsigned    references;        /* Number of times this instance is open */
        Cluster     cluster_index; /* Cached cluster position (0 = N/A) */
        Cluster     cluster;            /* Cached cluster address (undefined if cluster_index==0) */
        Dentry      *dentry;             /* Cached directory node for this open instance */
};


/* alloc.c */
int32_t fat12_read(Volume *v, Cluster n, unsigned fat_num);
int32_t fat16_read(Volume *v, Cluster n, unsigned fat_num);
int32_t fat32_read(Volume *v, Cluster n, unsigned fat_num);
int         fat_get_file_sector(Channel *c, Sector sector_index, Sector *sector);

/* dir.c */
//int fat_build_fcb_name(const struct nls_operations *nls, uint8_t *dest, const char *src, size_t src_size, bool wildcards);
int fat_do_readdir(Channel *c, LookupData *lud);

/* dos.c */
//int fat_findfirst(Dentry *dparent, const char *fn, size_t fnsize, int attr, fd32_fs_dosfind_t *df);
//int fat_findnext (Volume *v, fd32_fs_dosfind_t *df);
//int fat_findfile (Channel *c, const char *fn, size_t fnsize, int flags, fd32_fs_lfnfind_t *lfnfind);

/* file.c */
off_t       fat_lseek       (Channel *c, off_t offset, int whence);
ssize_t fat_read        (Channel *c, void *buffer, size_t size);
//int         fat_get_attr (Channel *c, fd32_fs_attr_t *a);

/* open.c */
void fat_dget    (Dentry *d);
void fat_dput    (Dentry *d);
int    fat_open    (Dentry *dentry, int flags, Channel **channel);
int    fat_create(Dentry *dparent, const char *fn, size_t fnsize, int flags, mode_t mode, Channel **channel);
int    fat_reopen_dir(Volume *v, Cluster first_cluster, unsigned entry_count, Channel **channel);
int    fat_close (Channel *c);
int    fat_lookup(Dentry **dentry, const char *fn, size_t fnsize);
//added for testing
Dentry *dentry_get(Dentry *parent, unsigned de_entcnt, Sector de_sector, unsigned attr);

/* volume.c */
//int fat_readbuf (Volume *v, Sector sector, Buffer **buffer, bool read_through);
int fat_mount(const char *blk_name, Volume **volume);
int fat_unmount(Volume *v);
int fat_partcheck(unsigned id);
//custom functions
int fat_init(char *path, Volume **v, Channel **c);
int fat_nextfile(Volume *v, Channel *c, Channel **c2, char filename[]);
void fat_getfilext(char *filename, char ext[]);

#if FAT_CONFIG_REMOVABLE && FAT_CONFIG_FD32
int fat_handle_attention(Volume *v);
#endif

/* Portability */
//ssize_t fat_blockdev_read(Volume *v, void *data, size_t count, Sector from);
ssize_t fat_blockdev_read(BlockDev *bd, void *data, size_t count, Sector from);
ssize_t fat_blockdev_write(Volume *v, const void *data, size_t count, Sector from);
int         fat_blockdev_test_unit_ready(Volume *v);

/* Functions with pathname resolution */
int fat_open_pr    (Dentry *dentry, const char *pn, size_t pnsize, int flags, mode_t mode, Channel **channel);
int fat_unlink_pr(Dentry *dentry, const char *pn, size_t pnsize);
int fat_rename_pr(Dentry *odentry, const char *on, size_t onsize,
                  Dentry *ndentry, const char *nn, size_t nnsize);
int fat_rmdir_pr (Dentry *dentry, const char *pn, size_t pnsize);
int fat_mkdir_pr (Dentry *dentry, const char *pn, size_t pnsize, mode_t mode);
//int fat_findfirst_pr(Dentry *dentry, const char *pn, size_t pnsize, int attr, fd32_fs_dosfind_t *df);

/* @} */
#endif /* #ifndef __FD32_FAT_DRIVER_H */


/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
```

```
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */

#define MAX_INPUT 1024
#include "mmc_header.h"
#include "stdio.h"
#include "fat.h"
#include "jpeg.h"

int input_buf_size =0;

static Volume *v;
static Channel c1d, c2d;
static Channel *c1=&c1d, *c2=&c2d;


int init_file_system( )
{
	unsigned char arg[4] = { 0x00, 0x00, 0x02, 0x00 };
	mmc_super_init(arg);
	send_clks(8,60);

	return 0;
}

int mount_file_system( )
{
	int res = 0;
	res = fat_init ("", &v, &c1);
	if (res < 0)
	{
		printf("Mount Error: %i (%s)\n", res, strerror(-res));
		return -1;
	}

	return 0;
}

int seek_file( const char * filename, const char * ext, int * size )
{
	if( filename == NULL || ext == NULL || size == NULL )
	{
		return -1;
	}

	return 0;
}

static unsigned char * file_buffer = NULL;
const unsigned char * read_next_file( const char * ext, char * filename, int * size )
{
	// check parameters
	if( ext == NULL || filename == NULL || size == NULL )
	{
		return NULL;
	}

	// check parameter integrity
	if( strlen( ext ) <= 0 )
	{
		return NULL;
	}

	// go to next file
	*size = fat_nextfile( v, c1, &c2, filename);
	if( *size <= 0 )
	{
		return NULL;
	}
	printf( "filename: %s\n", filename );

	// check file extension
	char file_ext[ 10 ] = { 0, };
	fat_getfilext( filename, file_ext );
	if( strncmp( file_ext, ext, strlen( file_ext ) ) != 0 )
	{
		return NULL;
	}

	// allocate buffer
	if( input_buffer == NULL )
	{
		input_buffer = ( unsigned char * ) malloc( ( * size ) * sizeof( unsigned char ) + 1 );
	}
	if( input_buffer == NULL )
	{
		return NULL;
	}

	// read file
```

```
	input_buf_size = *size;
	input_buffer[ *size ] = -1;
	int res = 0;
	res = fat_read( c2, input_buffer, *size );
	if( res < 0 )
	{
		free( input_buffer );
		input_buffer = NULL;
		return NULL;
	}

	return input_buffer;

}

void fat_loop_files()
{

   //printf ("Inside fat loop\n");
unsigned int image_counting=0;
   unsigned char arg[4] = { 0x00, 0x00, 0x02, 0x00 };
   mmc_super_init(arg);
   send_clks(8,60);

  // do
   {

	int res, size;//, i, j;
	char filename[13];
	  char ext[4];
	char buffer[512];

		res = fat_init ("", &v, &c1);
		if (res < 0) {
			 printf("Mount Error: %i (%s)\n", res, strerror(-res));
			 return 0;
		}

		//loop thru all files in the root directory
		while((size = fat_nextfile(v, c1, &c2, filename)) >= 0 && image_counting < 8 ) {
		    printf("filename = %s, size=%d\n", filename, size);
			fat_getfilext(filename, ext);
		  {
			if (strncmp(ext, "JPG", 3) == 0) {
			    //printf("JPEG Found!\n");
			    input_buffer = (unsigned char*) malloc (size * sizeof(unsigned char) + 1);
			    if(input_buffer == NULL)
			    {
			        printf("\nError in allocating memory for input buffer\n");
			    }
				//    if small file, print it's contents
				  //if (size < 512) {
				    input_buf_size = size;
				    input_buffer[size] = -1;
				    int    read =0;
				    //while(read < size)
				    //{

				    //int a;
				    //char *b = malloc(100000);
				    //printf("Stack = %d, Heap=%d\n", &a, b);
				    //free(b);
				   res = fat_read(c2, input_buffer+read, size);
				     //printf ("Bytes read = %d, total=%d \n", res, res);
				     if (res < 0) {
					    printf("Read error: %i (%s)\n", res, strerror(-res));
					    //break;
				     }
				     //read += res;
				    //} //end while


				      if (res >= 0)
				      {


					    //    for(i=0; i<size; i++)
					   // printf("%d\n", input_buffer[i]);
					     //printf ("Before Decoding\n");

					      //exit (1);
					    jpeg_decoder();
					    image_counting= image_counting+1;
				    }
				    else
				     free (input_buffer);

			    //}
			}

		}
//


}

  }
  //while(1);
```

```
}


/* FreeDOS-32 open/fcntl constants
 * by Salvo Isaja, May 2005
 */
#ifndef __FD32_FCNTL_H
#define __FD32_FCNTL_H

/* The following are derived rrom the DOS API (see RBIL table 01782) */
#define O_ACCMODE      (3 << 0)
#define O_RDONLY       (0 << 0)
#define O_WRONLY       (1 << 0)
#define O_RDWR         (2 << 0)
//#define O_NOATIME      (1 << 2)    /* Do not update last access timestamp */
#define O_NOINHERIT (1 << 7)    /* Not inherited from child processes */
#define O_DIRECT       (1 << 8)    /* Direct (not buffered) disk access */
#define O_SYNC         (1 << 14) /* Commit after every write */
#define O_FSYNC        O_SYNC
/* The following are extensions */
#define O_DIRECTORY (1 << 16) /* Must be a directory */
#define O_LINK         (1 << 17) /* Do not follow links */
#define O_NOFOLLOW     O_LINK
#define O_CREAT        (1 << 18)
#define O_EXCL         (1 << 19)
#define O_TRUNC        (1 << 20)
#define O_APPEND       (1 << 21)
#define O_NOTRANS      (1 << 22)
#define O_SHLOCK       (1 << 23)
#define O_EXLOCK       (1 << 24)

#endif /* #ifndef __FD32_FCNTL_H */


/* The FreeDOS-32 FAT Driver version 2.0
 * Copyright (C) 2001-2005    Salvatore ISAJA
 *
 * This file "file.c" is part of the FreeDOS-32 FAT Driver (the Program).
 *
 * The 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.
 *
 * The 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 the Program; see the file GPL.txt; if not, write to
 * the Free Software Foundation, Inc.,
 * 59 Temple Place, Suite 330, Boston, MA    02111-1307    USA
 */
/**
 * \file
 * \brief Facilities to access open files.
 */
/**
 * \addtogroup fat
 * @{
 */
#include "fat.h"

/**
 * \brief    Backend for the "lseek" POSIX system call.
 * \param    c           the file instance to seek into;
 * \param    offset new byte offset for the file pointer according to \c whence;
 * \param    whence can be \c SEEK_SET, \c SEEK_CUR or \c SEEK_END; the latter
 *                       is not allowed for directories.
 * \return On success, the new byte offset from the beginning of the file,
 *                or a negative error.
 */
off_t fat_lseek(Channel *c, off_t offset, int whence)
{
        off_t res = -EINVAL;
        if (!c) return -EFAULT;
        if (c->magic != FAT_CHANNEL_MAGIC) return -EBADF;
        switch (whence)
        {
                case SEEK_SET: res = offset; break;
                case SEEK_CUR: res = offset + c->file_pointer; break;
                case SEEK_END:
                        if (!(c->f->de.attr & FAT_ADIR))
                                res = offset + c->f->de.file_size;
                        break;
        }
        if (res < 0) return -EINVAL;
        c->file_pointer = res;
        return res;
}


/**
 * \brief    Backend for the "read" POSIX system call.
 * \param    c           the file instance to read from;
 * \param    buffer pointer to a buffer to receive the data;
 * \param    size      the number of bytes to read;
```

```
 * \return The number of bytes read on success (may be less than \c size, 0 at EOF), or a negative error.
 */
ssize_t fat_read(Channel *c, void *buffer, size_t size)
{
        off_t       offset;
        size_t      k = 0, count;
        unsigned byte_in_sector;
        int         res;
        Sector      sector, sector_index;
        Buffer    *b = NULL;
        File        *f = c->f;
        Volume    *v = f->v;

        if (!c || !buffer) return -EFAULT;
        if (c->magic != FAT_CHANNEL_MAGIC) return -EBADF;
        assert(c->file_pointer >= 0);
        if (IS_NOT_READABLE(c)) return -EBADF;
        offset = c->file_pointer;
      //printf("size to read = %d\n", size);
        while (k < size)
        {
                if (!(f->de.attr & FAT_ADIR) && (offset >= f->de.file_size)) break; /* EOF */
                /* Locate the current sector and byte in sector position */
                sector_index = offset >> v->log_bytes_per_sector;
                res = fat_get_file_sector(c, sector_index, &sector);
                if (res < 0) {
                  printf("4. Failed to get file sector: %d\n", sector);
                  return res;
            }
                if (res > 0) break; /* EOF */
                byte_in_sector = offset & (v->bytes_per_sector - 1);

                /* Read as much as we can in that sector with a single operation.
                 * However, don't read past EOF or more than "size" bytes. */
                count = v->bytes_per_sector - byte_in_sector;
                if (!(f->de.attr & FAT_ADIR) && (offset + count >= f->de.file_size))
                        count = f->de.file_size - offset;
                if (k + count > size)
                        count = size - k;

                /* Fetch the sector, read data and continue */
                res = fat_readbuf(v, sector, &b, false);
                if (res < 0) {
                  printf("5. Failed to read sector: %d\n", sector);
                  return res;
            }
                memcpy((uint8_t *) buffer + k, b->data + res + byte_in_sector, count);
                offset += count;
                k      += count;
          // printf("k = %d\n", k);
        }
        /* Successful exit, update file last-access time-stamp if required */
        c->file_pointer = offset;
        return (ssize_t) k;
}

/* @} */


/*--------------------------------------------*/
/* File : huffman.c, utilities for jfif view */
/* Author : Pierre Guerrier, march 1998       */
/*--------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>

#include "jpeg.h"


/*---------------------------------------*/
/* private huffman.c defines and macros */
/*---------------------------------------*/

#define HUFF_EOB            0x00
#define HUFF_ZRL            0xF0

/*--------------------------------------------*/
/* some constants for on-the-fly IQ and IZZ */
/*--------------------------------------------*/

static const int G_ZZ[] = {
     0,  1,  8, 16,  9,  2,  3, 10,
    17, 24, 32, 25, 18, 11,  4,  5,
    12, 19, 26, 33, 40, 48, 41, 34,
    27, 20, 13,  6,  7, 14, 21, 28,
    35, 42, 49, 56, 57, 50, 43, 36,
    29, 22, 15, 23, 30, 37, 44, 51,
    58, 59, 52, 45, 38, 31, 39, 46,
    53, 60, 61, 54, 47, 55, 62, 63
};

/*-----------------------------------------------*/
/* here we unpack, predict, unquantify and reorder */
/* a complete 8*8 DCT block ...                  */
/*-----------------------------------------------*/

void
unpack_block(FILE *fi, FBlock *T, int select)
```

```
{
   unsigned int i, run, cat;
   int value;
   unsigned char     symbol;

   /* Init the block with 0's: */
   for (i=0; i<64; i++) T->linear[i] = 0;

   /* First get the DC coefficient: */
   symbol = get_symbol(fi, HUFF_ID(DC_CLASS, comp[select].DC_HT));
   value = reformat(get_bits(fi, symbol), symbol);

#ifdef SPY
   trace_bits(symbol, 1);
#endif

   value += comp[select].PRED;
   comp[select].PRED = value;
   T->linear[0] = value * QTable[comp[select].QT]->linear[0];

   /* Now get all 63 AC values: */
   for (i=1; i<64; i++) {
      symbol = get_symbol(fi, HUFF_ID(AC_CLASS, comp[select].AC_HT));
      if (symbol == HUFF_EOB) break;
      if (symbol == HUFF_ZRL) { i += 15; continue; }
      cat = symbol & 0x0F;
      run = (symbol>>4) & 0x0F;
      i += run;
      value = reformat(get_bits(fi, cat), cat);

#ifdef SPY
      trace_bits(cat, 1);
#endif

      /* Dequantify and ZigZag-reorder: */
      T->linear[G_ZZ[i]] = value * QTable[comp[select].QT]->linear[i];
   }
}
```

7. References

[1]Stephen A. Edwards CSEE 4840 Embedded System Design. http://www1.cs.columbia.edu/~sedwards/classes/2008/4840/index.html
[2] Interfacing a MultiMediaCard with ADSP-2126x SHARC Processors (EE-264)
[3] SD Card Association. http://sdcard.org/
[4] Jun Li, Sharp. Interfacing a MultiMediaCard to the LH79520 System-On-Chip
[5] Secure Digital Card Interface for the MSP430, Michigan State University, 2004
[6] Imagic project , embedded systems design class, spring 2007.
[7] http://en.wikipedia.org/wiki/YUV