Embedded Systems Lab CSEE 4840 : IMAGIC Project Report

Abhilash Itharaju   Nalini Vasudevan   Vitaliy Shchupak   Walter Dearing

Abstract

The Imagic Project has successfully been completed, and the goals set at the start of the project have been reached. The system is able to read the contents of a MMC card and then decode and display all the JPEG images on the screen one after the other as a slideshow. The system can do this quickly, while keeping the transition between images smooth and altering the background to match the picture.

1 Introduction

The basic idea is to have two peripherals, 1) to control the onboard SD card reader and 2) to control the VGA DAC. The function of first peripheral is to talk to MMC card reader and to read things from it. The other peripheral is to interact with VGA DAC and provide it with the proper signals (pixel data, H_SYNC, V_SYNC, BLANKING). The main function of the software is to initialize and control the peripherals and also to decode the JPEG image once it is read from the SD card. The top level idea is to have two memory locations. One is where the program sits (SDRAM) and the other (SRAM) is where the image buffer is kept so that the video peripheral can quickly read from it.

2 Hardware Design

The hardware is broken into four main components:

1. SDRAM
2. VGA Peripheral
3. SD Card Controller
4. Nios II System

The four main components communicate over the Avalon Bus. The image is stored on the SD Card. The Nios II processor reads in this image and decompresses it. It then sends the image to the VGA peripheral. The VGA peripheral stores the data sent from the NIOS into SRAM. The VGA peripheral then reads the SRAM every 25 MHz for a new pixel to display on the screen.

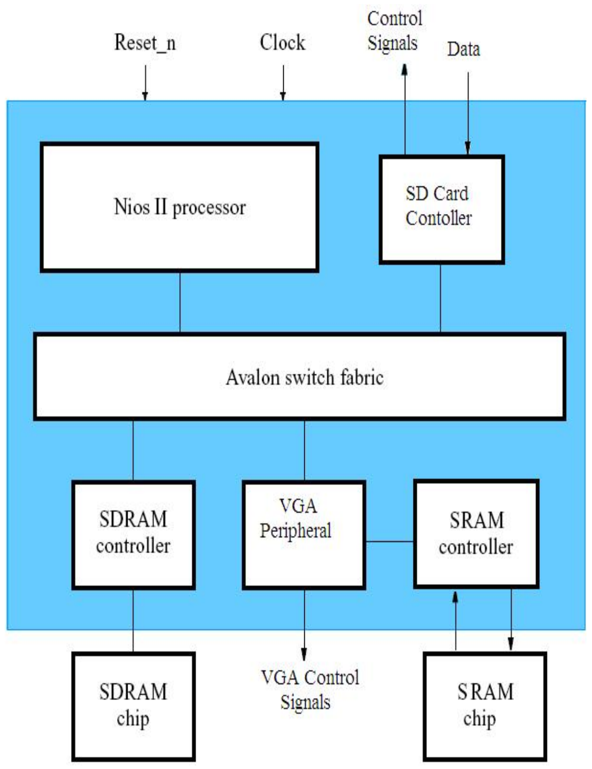


Figure 1: Block Diagram of Imagic

Because each component of the system can be treated as a black box, each item will be discussed in detail separately.

2.1 Interface with MMC/SD Card

We will be communicating with the MMC/SD Card in the SPI mode. Since SD Card is backward compatible with MMC card, the interface is the same. This is how we can make the NIOS-II processor talk to the MMC card.

The following figure shows how the FPGA is connected to the SD Card slot.

For the SPI mode we need all the 4 pins.

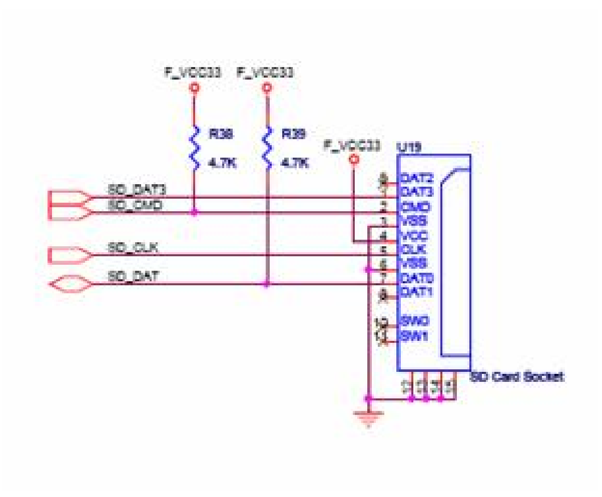
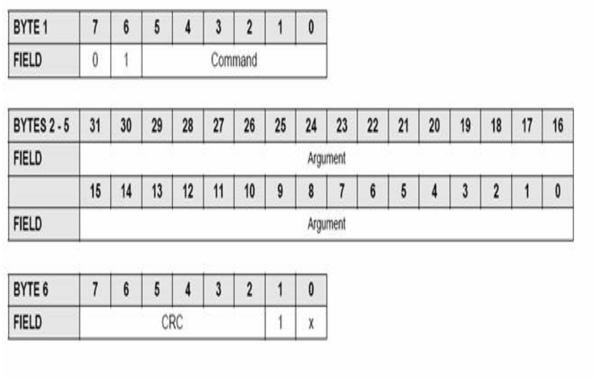


Figure 2: FPGA Connection to the SD Card Slot

2.1.1 SPI Commands

Communications between the microcontroller and the MMC are initiated by different commands sent from the FPGA. All commands are 6 bytes long and are transmitted MSB first.



BYTE 1	7	6	5	4	3	2	1	0
FIELD	0	1	Command					

BYTES 2 - 5	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FIELD	Argument															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIELD	Argument															

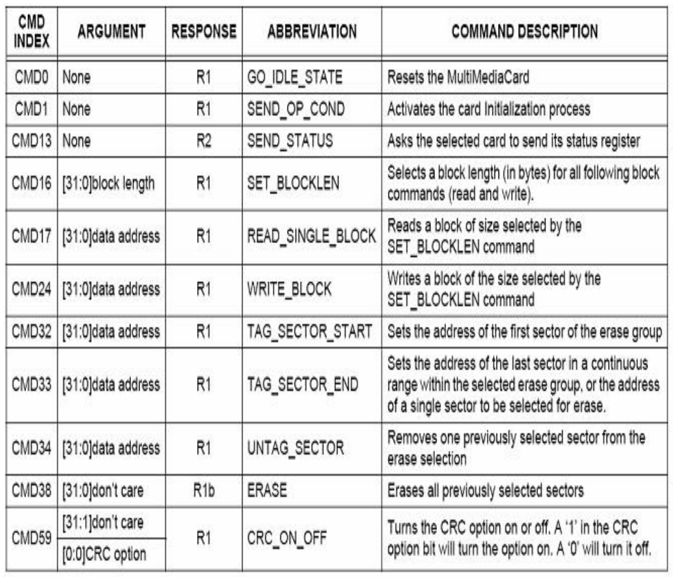
BYTE 6	7	6	5	4	3	2	1	0
FIELD	CRC						1	x

Figure 3: SPI Commands

Figure 4 is a list of commands that can be sent to the MMC Card.

2.1.2 CRC Calculation

The CRC bit calculation is performed:
7 bit CRC Calculation: G(x) = x7 + x3 + 1
M(x) = (start bit) x39 + (second bit) x38+ ...+ (last bit before CRC) x0
CRC[6..0] = Remainder[(M(x) x7)/G(x)]



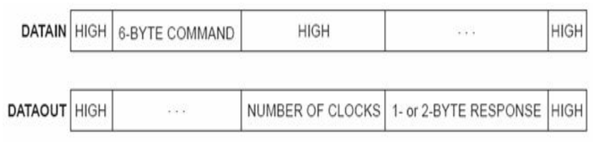
CMD INDEX	ARGUMENT	RESPONSE	ABBREVIATION	COMMAND DESCRIPTION
CMD0	None	R1	GO_IDLE_STATE	Resets the MultiMediaCard
CMD1	None	R1	SEND_OP_COND	Activates the card Initialization process
CMD13	None	R2	SEND_STATUS	Asks the selected card to send its status register
CMD16	[31:0]block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read and write).
CMD17	[31:0]data address	R1	READ_SINGLE_BLOCK	Reads a block of size selected by the SET_BLOCKLEN command
CMD24	[31:0]data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command
CMD32	[31:0]data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group
CMD33	[31:0]data address	R1	TAG_SECTOR_END	Sets the address of the last sector in a continuous range within the selected erase group, or the address of a single sector to be selected for erase.
CMD34	[31:0]data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection
CMD38	[31:0]don't care	R1b	ERASE	Erases all previously selected sectors
CMD59	[31:1]don't care [0:0]CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A '1' in the CRC option bit will turn the option on. A '0' will turn it off.

Figure 4: FPGA Connection to the SD Card Slot

2.1.3 Response Format R1

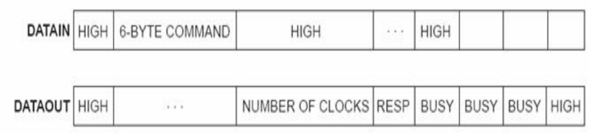
This response token is sent by the card after every command with the exception of SEND_STATUS commands. It is 1 byte long; the MSB is always set to zero and the other bits are error indications. A 1 signals an error.

Here is the timing of various commands and responses:



DATAIN	HIGH	6-BYTE COMMAND	HIGH	...	HIGH
DATAOUT	HIGH	...	NUMBER OF CLOCKS	1- or 2-BYTE RESPONSE	HIGH

Figure 5: Command and Response when card is not busy



DATAIN	HIGH	6-BYTE COMMAND	HIGH	...	HIGH			
DATAOUT	HIGH	...	NUMBER OF CLOCKS	RESP	BUSY	BUSY	BUSY	HIGH

Figure 6: Command and Response when card is busy

2.1.4 Clock Control

The SPI bus clock signal can be used by the SPI host to set the cards to energy saving mode or to control data flow (to avoid under-run or over-run conditions) on the bus. The host

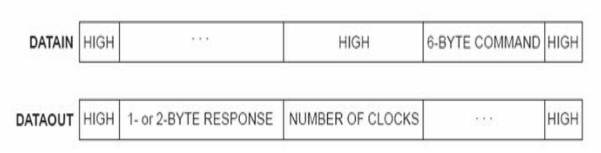


Figure 7: Card response to command from host

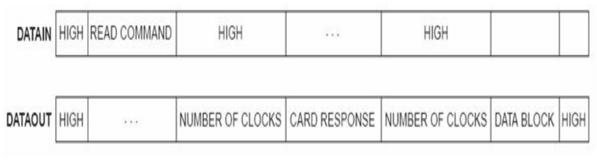


Figure 8: Data read command and response

is allowed to change the clock frequency or stop it altogether. There are a few restrictions the SPI host must follow:

- The bus frequency can be changed at any time, but only up to the maximum data transfer frequency, defined by the MultiMediaCards.
- It is an obvious requirement that the clock must be running for the MultiMediaCard to output data or response tokens. After the last SPI bus transaction, the host is required to provide 8 clock cycles for the card to complete the operation before shutting down the clock. During this 8-clock period, the state of the CS signal is irrelevant. It can be asserted or de-asserted.

2.2 SPI BUS TRANSACTIONS

Here is a list of the various SPI bus transactions:

- A command/response sequence. Eight clocks must be output after the card response end bit. The CS signal can be asserted or de-asserted during these 8 clocks.
- A read data transaction. Eight clocks must be output after the end bit of the last data block.
- A write data transaction. Eight clocks must be output after the end bit of the last data block.
- A write data transaction. Eight clocks must be output after the CRC status token.
- The host is allowed to stop the clock of a BUSY card. The MultiMediaCard will complete the programming operation regardless of the host clock. However, the host must provide a clock edge for the card to turn off its BUSY signal. Without a clock edge, the MultiMediaCard (unless previously disconnected by deasserting the CS signal) will force the DataOut line LOW and hold it there.

2.2.1 MODE SELECTION

The MultiMediaCards SPI mode is the mode used for this Application Note. All transactions described in this Application Note are based on the SPI mode. The MultiMediaCard wakes up in the MultiMedia- Card mode. It will enter SPI mode if the CS signal (pin1 of the MMC) is asserted LOW during the reception of the Reset command (CMD0). If the card is in MultiMediaCard mode, it will not respond to SPI-based commands.

If SPI mode is requested, the card will switch to SPI mode and respond with the SPI mode R1 response. To return to the MultiMediaCard mode, power cycle the card. In SPI mode, the MultiMediaCard protocol state machine is not observed. MultiMediaCard commands supported in SPI mode are always available. Since the card defaults to MultiMediaCard mode after a power cycle, Pin 1 (CS) must be pulled LOW and CMD0 (followed by a valid CRC byte) must be sent on the CMD (DataIn, Pin 2) line for the card to enter SPI mode. In SPI mode, CRC checking is disabled by default. However, since the card always powers up in MultiMediaCard mode, CMD0 must be followed by a valid CRC byte (even though the command is sent using the SPI structure). Once the card enters SPI mode, CRCs are disabled by default. CMD0 is a static command and always generates the same 7-bit CRC of 4Ah. Adding the 1 end bit (bit 0) to the CRC creates a CRC byte of 95h. The following hexadecimal sequence can be used to send CMD0 in all situations for SPI mode, since the CRC byte (although required) is ignored once in SPI mode. The entire CMD0 appears as: 40 00 00 00 00 95 (hexadecimal).

CMD0 is a static command and always generates the same 7-bit CRC of 4Ah. Adding the 1 end bit (bit 0) to the CRC creates a CRC byte of 95h. The following hexadecimal sequence can be used to send CMD0 in all situations for SPI mode, since the CRC byte (although required) is ignored once in SPI mode. The entire CMD0 appears as: 40 00 00 00 00 95 (hexadecimal).

2.2.2 RESET SEQUENCE

The initialization command is described in the following sequence:

1. Send 80 clocks to start bus communication
2. Assert nCS LOW

3. Send CMD0
4. Send 8 clocks for delay
5. Wait for a valid response
6. If there is no response, back to step 4
7. Send 8 clocks of delay
8. Send CMD1
9. Send 8 clocks of delay
10. Wait for valid response
11. Send 8 clocks of delay
12. Repeat from step 9 until the response shows READY.

It will take a large number of cycles for CMD1 to finish its sequence. After every power cycle, the MMC will be in Idle state (not active), the Idle bit in its response will be 1 if using CMD13 (SEND_STATUS) to check the status. Once the CMD1 process is finished, the Idle bit in the response is cleared. Only after MMC is fully up from Idle mode to Active, can it be read and written.

2.2.3 Data Read

The SPI mode supports single block read operations only. Upon reception of a valid Read command, the card will respond with a Response token followed by a Data token in the length defined by a previous SET_BLOCK_LENGTH command. The start address can be any byte address in the valid address range of the card. Every block however, must be contained in a single physical card sector. After the Data Read command is sent from FPGA to the card, the FPGA will need to monitor the data stream input and wait for Data Token 0xFE. Since the response start bit 0 can happen any time in the clock stream, its necessary to use software to align the bytes being read.

2.2.4 Implementation

Right now, there is peripheral for each of the pins. So, there are four peripherals in total connected to the NIOS-II processor. Everything else is software controlled. Each of the pins is either set high or low by the software directly and that is how the above interface protocol is followed.

2.3 SDRAM

The original plan was to use the SRAM to store the picture and the C code used to retrieve the image from the SD Card and decompress the image. While in development, it was realized early on that the C code needed for the project was going to be bigger than expected. Therefore, the decision was made to use the SDRAM to store the C code, so that the code size would not be an issue. The SRAM would still be used to store the image.

The SDRAM was setup using the tutorial supplied by Altera [1]. The directions were straight forward. The setup specifies to instantiate the SDRAM controller in the SOPC builder when building the NIOS II System. The next step was to connect the SDRAM chip to the top level VHDL file, so that the SDRAM controller could be connected to the SDRAM. Using the top level design file and pin assignment file from Lab3, the only thing that had to be done was route the SDRAM to the NIOS II system in the corresponding port map. Most of the pin assignments were straightforward; however, the clock needed one special step. Because there may be potential clock skew between the SDRAM and NIOS system due to the physical characteristics of the board, a phase-lock loop is needed so that the SDRAM clock signal leads the NIOS II System by 3 nano-seconds. Again, using the tutorial, this was straight forward. A separate phase locked loop component was produced that took the 50 Mhz clock as an input and returned the clock signal that could be routed to the SDRAM.

2.4 VGA Peripheral

The VGA peripheral is responsible for controlling the VGA(using the VGA controller) and is also responsible for reading/writing the SRAM. The peripheral accepts data from the NIOS system. The NIOS II System can write four different types of data to the peripheral. The VGA peripheral distinguishes the different type of data by what address the NIOS II writes to.

Address Written	Data Type
0x00	Data for SRAM
0x01	Address to write to in SRAM
0x02	Address to write to in SRAM
0x03	Picture Width
0x04	Picture Length

Table 1: Specifying data to the VGA Peripheral

As already indicated, the VGA peripheral has sole control of the SRAM. The first thing that the NIOS system will send is a picture width of 1. This is used as an indication to the peripheral that a new image is being written to the SRAM (since it is assumed no image widths are 1). Therefore, the peripheral will blank out the screen so that the user will not see the SRAM being written. At the same time, a flag is set to indicate that the background 'complexion' should be changed. The background color is set to match the picture, by making the background color the first pixel in the image. Next the NIOS will send the first address that should be written in the SRAM. This data will also be latched to be used later. Finally the NIOS will send the first data value. This data will be written to the SRAM at whatever address was previously specified. The NIOS will then continuously send SRAM addresses then SRAM data until the entire image is written to SRAM. When the entire image is written to the SRAM, the NIOS will send the actual image width and height. This is an indication that a valid image is ready in the SRAM and it should be displayed. The image height and width will be stored in registers (using VHDL SIGNAL), so that it can be used over and over again.

The data sent from the NIOS II system is a 15 bit value that is used to specify one pixel point. The VGA controller expects RGB values where 9 bits specify each color. To save space in the SRAM, only five bits are sent to the VGA peripheral for each color. When being sent to the VGA controller, the peripheral will copy the five bits stored in the SRAM to the upper and lower portion of the 9 bit value the VGA controller expects.

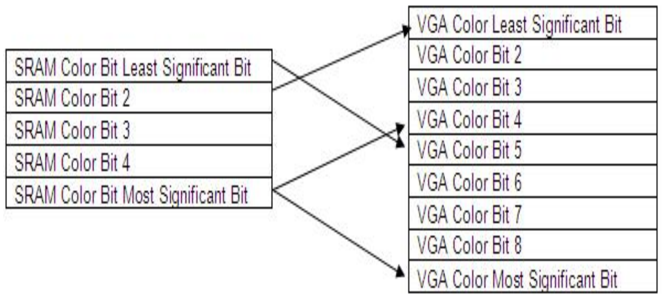
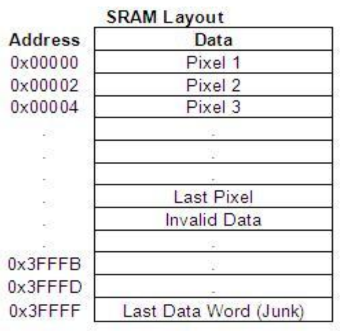


Figure 9: Bit Mapping between SRAM data and VGA data

The image is arranged in the SRAM as follows:

There is 512K available in the SRAM. Therefore 256K pixels can be stored in the SRAM. Therefore approx a 546 x 408 size image can fit into SRAM max. The image will be centered and the rest of the screen will display the background color.

The timing between the SRAM and VGA is of particular interest. Three timing diagrams are important when determining the timing needed for Imagic: 1) the timing constraints



SRAM Layout

Address	Data
0x00000	Pixel 1
0x00002	Pixel 2
0x00004	Pixel 3
.	.
.	.
.	Last Pixel
.	Invalid Data
.	.
0x3FFFB	.
0x3FFFD	.
0x3FFFF	Last Data Word (Junk)

Figure 10: SRAM Layout

needed to read from the SRAM[2], 2) the timing constraints needed to write to the SRAM[2], and 3) the timing constraints needed to force the slave-write transfer to stall one cycle[3].

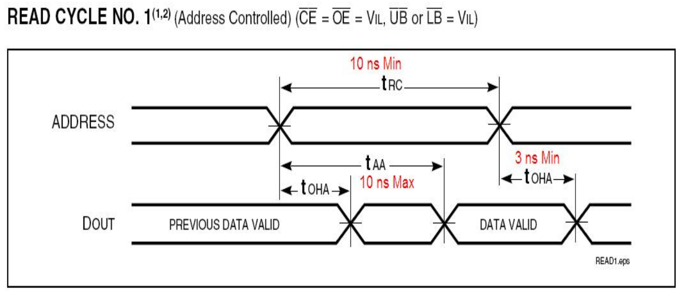


Figure 11: Timing constraints when reading from the SRAM

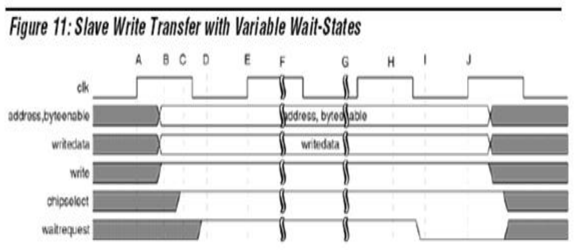


Figure 12: Timing constraints to force the slave write to stall

There are a few signals to consider when discussing the interaction between the VGA peripheral, the Avalon bus, and the SRAM:

- avs_s1_clk : 50 MHz clock
- Clk : 25 MHz clock (split from 50 Mhz clock)
- ADDRESS_TO_SRAM : When writing, what address data should be written. When reading, what address should be output from SRAM.

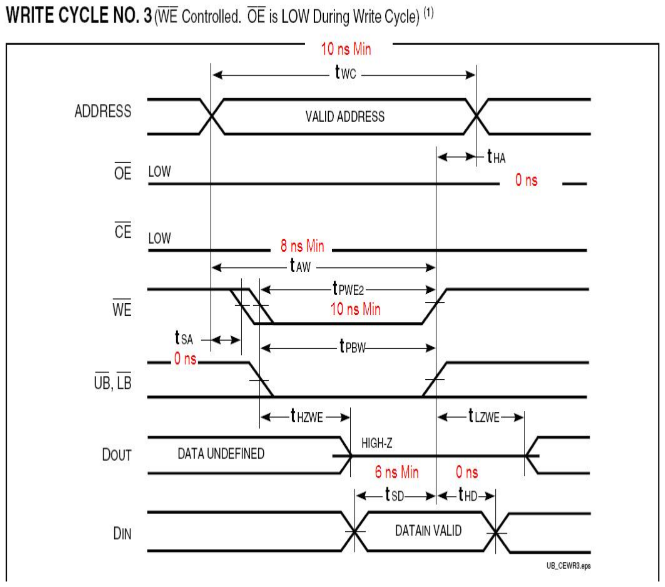


Figure 13: Timing constraints when writing to the SRAM

- RAM_READ_ADDR : What address should be output from SRAM.
- RAM_WRITE_ADDR : What address should be written to SRAM.
- WRITEENABLE_TO_SRAM : Indicates when the SRAM should write data to SRAM.
- WRITEDATA_TO_SRAM : Image Data to be written to SRAM
- avs_s1_write = '1' : Signal received through Avalon protocol the master (NIOS II) wants to send the VGA slave data.
- avs_s1_chipselect : Specifies the NIOS II is selecting the VGA peripheral for I/O
- avs_s1_writedata : Data received from NIOS processor that is to be written to SRAM.
- avs_s1_address : Indicates what type of data is being received from NIOS
- avs_s1_waitrequest : Used to stall the Avalon interconnect when the VGA peripheral is not ready to retrieve and right data to the SRAM.
- READDATA_FROM_SRAM : Pixel Point read from SRAM
- IMAGE_RED : Internal SIGNAL used to store RED component of VGA signal
- IMAGE_GREEN : Internal SIGNAL used to store GREEN component of VGA signal
- IMAGE_BLUE : Internal SIGNAL used to store BLUE component of VGA signal
- write_data : Flag used to indicate that SRAM has been previously been written, and SRAM should not read next pixel.
- wirte_background : Flag used to indicate background should be changed.

Using these guidelines and variables, basic components of the algorithm can be constructed.

The first thing that will be described is how the address that specifies what data should be read from the SRAM is controlled. A register (VHDL SIGNAL) is created named RAM_READ_ADDR. This signal is 18 bits and initialized to zero. This variable is incremented every time a pixel from the image is read from SRAM. So every time the VGA peripheral displays a pixel in the rectangle defined by the width and height of the image, the RAM_READ_ADDR variable will be incremented. This ensures that the VGA peripheral reads a new value from the SRAM every 25 MHz. Once the VGA controller reaches the bottom right pixel (the end of the screen), the RAM_READ_ADDR variable is reset to zero, so that the start of the image can be read. This works because the image is stored continuously in SRAM.

Next the way the SRAM is read/written is described,

```
if avs_s1_clkevent and avs_s1_clk = 1
   if clk = 1   -- Read Data
     -- Tell SRAM to output data
     -- NOTE:  This also commits SRAM data that
               may have been specified during WRITE
               phase
     WRITEENABLE_TO_SRAM <= '1';
     -- Specify what address to output
     ADDRESS_TO_SRAM <= RAM_ADDR;
   else         -- Write Data
     write_data = '0';
     if avs_s1_chipselect = '1' and avs_s1_write = '1'
       -- Data Type
       if avs_s1_address = "00000" then
         -- Tell SRAM where to write
         ADDRESS_TO_SRAM <= RAM_WRITE_ADDR;
          -- Write Data
         WRITEDATA_TO_SRAM <= avs_s1_writedata;
         WRITEENABLE_TO_SRAM <= '0'
         -- Indicate not to read SRAM
         write_data = '1';
         if write_background = '1'
          BG_RED <= READDATA_FROM_SRAM(14 downto 10);
          BG_GREEN <= READDATA_FROM_SRAM(9 downto 5);
          BG_BLUE <= READDATA_FROM_SRAM(4 downto 0);
        -- Address Type
        elsif avs_s1_address = "00001" then
          -- Grab SRAM write ADDR
          RAM_WRITE_ADDR <= avs_s1_writedata;
        -- Width Type
        elsif avs_s1_address = "00011" then
           -- Grab Image Width
           IMAGE_HEND <= avs_s1_writedata;
           -- Get new background
           write_background = '1';
        -- Height Type
        elsif avs_s1_address = "00100" then
           -- Grab Image Height
           IMAGE_VEND <= avs_s1_writedata;
elsif avs_s1_clk'event and avs_s1_clk = '0' then
   if clk = '1' and write_data = 0 then
    -- Latch data from SRAM
    IMAGE_RED <= READDATA_FROM_SRAM(14 downto 10);
    IMAGE_GREEN <= READDATA_FROM_SRAM(9 downto 5);
    IMAGE_BLUE <= READDATA_FROM_SRAM(4 downto 0);
    -- Indicate that we can write data to SRAM next cycle
    avs_s1_waitrequest <= 0;
   else if clk = 0
    -- Indicate that we cant write data
       to the SRAM next cycle and we need to stall.
    avs_s1_waitrequest <= 1;
```

Many aspects of this algorithm need to be analyzed. First it should be noted that avs_s1_clkevent is used to control when this code is evaluated. Also note that both the rising _s1_clk edge and the falling avs_s1_clk edge are important. This is the 50 MHz clock. Next, notice that clk is used to control what action is being performed on the SRAM. If clk is = 1, then the SRAM will be read from. If clk = 0, then the SRAM can be written do (if desired). Clk is the 25 MHz clock. Therefore, there are fourr distinct states available:

1. Rising edge of avs_s1_clk and Clk = 1
2. Falling edge of avs_s1_clk and Clk = 1
3. Rising edge of avs_s1_clk and Clk = 0
4. Falling edge of avs_s1_clk and Clk = 0

Now for a more detailed analysis of the read cycle (when clk = 1).

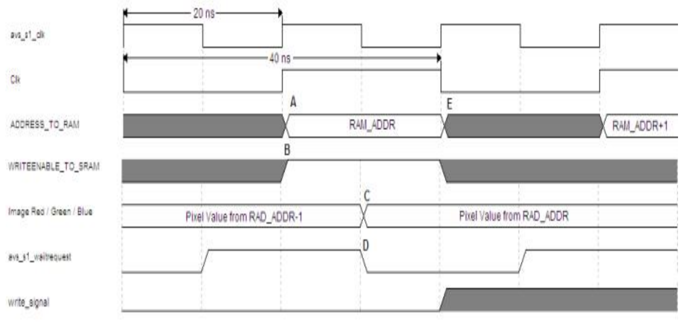


Figure 14: Read Cycle

Notes on Diagram:

- A : Output to the SRAM the address of the pixel point to be read from the SRAM.
- B : Raise Write Enable Signal, since this is the read phase. This will cause SRAM to output data. This also will actually write any data specified during write cycle of SRAM.
- C : As specified by the timing diagram from the SRAM datasheet [2], valid data will be output from the SRAM after 10 ns. At this point it has been 10ns since a valid address was supplied. Therefore, the data from the SRAM is latched and fed into the appropriate IMAGE_RED, IMAGE_GREEN, and IMAGE_BLUE registers.
- D : During the next rising avs_s1_clk edge, the VGA peripheral will be able to write the SRAM, so there is no need to raise the wait request signal.
- E : The RAM_ADDR and WRITENABLE_TO_SRAM values may change, since this is now the write phase.

Now for a more detailed analysis of the write cycle (when clk = 0).

Notes on Diagram:

- A : If chipselect signal and write signal is high, then the NIOS is writing data to VGA peripheral.
- B : This is used to distinguish what type of data is being sent to VGA peripheral. In this case its 0x00, so therefore its a pixel value to be written to the SRAM
- C : Set SRAM address that the pixel should be written to.

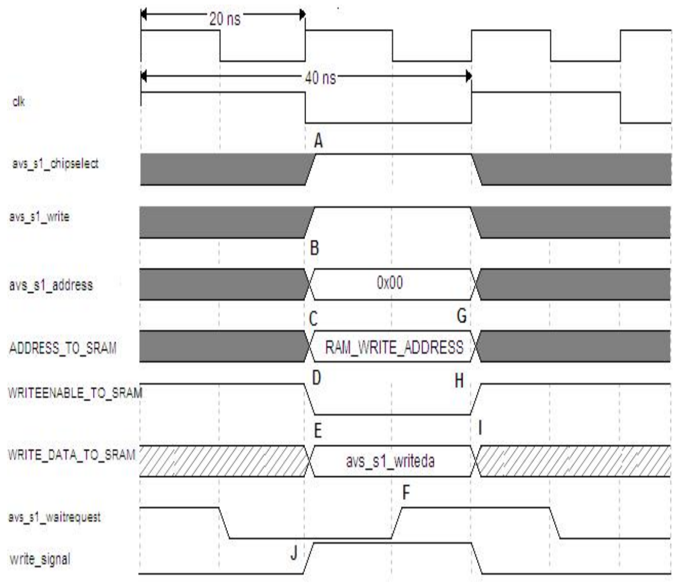


Figure 15: Write Cycle

- D : Set WRITEENABLE_TO_SRAM to indicate that data should be written to SRAM.
- E : Write the pixel value to the SRAM. This is the value received over Avalon bus.
- F : During the next rising avs_s1_clk edge, the VGA peripheral will not be able to write the SRAM, so tell the Avalon interface to wait one clock cycle by raising the wait request signal.
- H : As specified by the timing diagram from the SRAM datasheet [2], the address for the SRAM can change immediately after valid data is written to the SRAM
- G : As specified by the timing diagram from the SRAM datasheet [2], the address and data must be valid for 10 ns before the data can be written to SRAM. At point H it has been 10 ns since there were valid address and data. Also note that the start of the read cycle will ALWAYS raise WRITEENABLE_TO_SRAM, so the data is guaranteed to be written to the SRAM.
- I : As specified by the timing diagram from the SRAM datasheet [2], the data for the SRAM can change immediately after valid data is written to the SRAM. This is let float, since the SRAM chip should be able to drive the data bus with its own data during the read phase.
- J : Once data is written to SRAM, raised the write_signal. This will indicate that data should not be read immediatly after the write, because the SRAM will not have enough time to drive the output bus.

Note that the timing diagram is not produced for the cases where the image width, image height, and SRAM write address are received from NIOS are produced, because these are simplified cases of the timing diagram above. Once the data is received from the Avalon bus, the data is immediately latched. It still occurs only during the write phase.

Some other considerations are the signals that have not been discussed regarding the SRAM chip. These pins are hard coded to the following values:

```
SRAM_UB_N <= '0';
SRAM_LB_N <= '0';
SRAM_CE_N <= '0';
SRAM_OE_N <= '0';
```

This will specify that the SRAM is always enabled, will always output data during the read phase, and every read/write will be a full word.

The image location is calculated in hardware. The image location is determined by starting with the center of the display (for a 640 x 480 display). From this pixel point, the upper left corner of the image is placed on the display by subtracting 1/2 width and 1/2 the height from the center pixel. This centers the image.

It has been described how to read and write an image to SRAM. Once the data is available for display, it is a trivial task to display it. Using the video display lab (lab 3) as a template, only minor modifications have to be made to display the image. Namely, instead of a constant color the RBG values will be updated every 25 MHz. Also, instead of a constant height or width for the rectangle, the height and width are now variable. Every other detail (involving syncs, porches, etc.) are exactly the same as lab3 and dont require any modification.

3 Software Design

3.1 Reading the 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 MS-DOS 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:

- fat_init() initialized the file system. Returns pointer to the file system structure.
- fat_nextfile() opens the next file in the root directory, returns the filename (8.3 format), file size, and pointer to access the file, or -1 when end of directory is reached.
- fat_read() reads the next n bytes from the given file pointer and fills a given buffer with its contents

3.2 JPEG Decoding

A JPEG image consists of a number of 8*8 pixel data block units known as Minimum Coded Unit or the MCU. The unit is converted to its frequency domain using a Discrete Cosine Transform. The high frequency components are filtered. The low pass filter is characterized by the Quantization Tables which determines the quality and the compression ratio of the JPEG image. Finally the JPEG decoder is coded using Huffman codes to allow more frequent values to be stored as shorter codes.

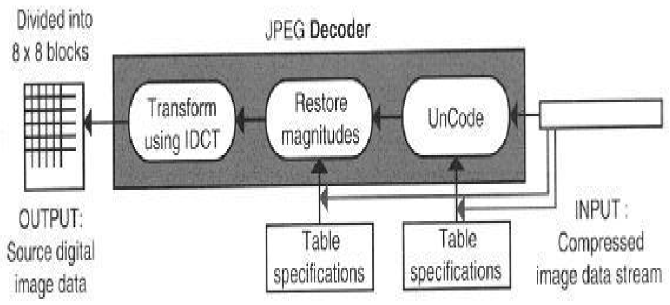


Figure 16: JPEG Decoding

The jpeg decoder is the inverse process. The 8*8 unit of information is retrieved from the encoded data. The coded data is decoded using the Huffmans algorithm using the data provided in the Huffman tables. The frequency components can be extracted using the quantization table. The result is a zig-zag (ZZ) vector which is reordered into an 8x8 block Finally, the inverse discrete cosine transform (IDCT) is applied to the frequency domain to get back the 8*8 MCU blocks.

The JPEG decoder will call the fat_nextfile() function and get a character array as input. It then outputs the image in raster format, and the corresponding RGB values are fed to the vga raster component.

3.3 Resolution

At a time, there is one image in the SRAM. If the size of the RGB image exceeds the size of the SRAM, the pixels are sampled depending on the size of the image that fit into the SRAM.

4 Who Did What?

- Abhilash - MMC Card Interface
- Vitaliy - FAT File System
- Nalini - JPEG Decoding, Resolution
- Walter - Video Display

5 Lessons Learned

It seemed like proper design choices early on led to the success of the project overall. By making the proper decisions, our time and energy was best utilized during the entire design process. The value of starting early was really demonstrated here. Because we had the time to consider all options, and not make decisions based on desperation, the best possible decisions were made. Being able to understand how VHDL maps to hardware was a major themed learned. Also, always taking into account timing constraints was stressed.

6 Advice for future students

6.1 Project Management

The first and the foremost advice is to start early. Difficulties can be solved slowly and steadily and wiser decisions can be made. Secondly, choose a project which neither too hard nor too easy. Thirdly, divide the work among the members, such that each person can work independently of and concurrently with each other.

6.2 Implementation Related

The design has to be clear before the project is implemented. Attention has to be paid to timing diagrams. Triple check the timing diagrams before you implement them. Completely understand the hardware constraints in terms of memory, timing and hardware capabilities before designing or implementing anything.

7 Code Listing

```
#include <mmc_header.h>
#include <stdio.h>

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

void delay(int i)
{
    int a;
    for(a=0;a<i;a++)
    {

    }
}

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

}

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

void mmc_disable()
{
    CLK_LO;
    DIN_HI;
    NCS_HI;
}

void send_cmd(unsigned  char cmd, unsigned char* arg)
{
   unsigned char *temp;
   unsigned char data;
   int i,j;
   NCS_LO;
   *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);
            data?DIN_HI:DIN_LO;
            send_clks(1,60);
        }
        temp++;
    }
    DIN_HI;
}

unsigned char recv_resp()
{
    unsigned char resp=0x00,temp=0x00;
    int i;
    temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
    while(temp == 1)
    {
        send_clks(1,60);
        temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
    }
    for(i=7;i>-1;i--)
    {
        temp = temp << i;
        resp = resp | temp;
        send_clks(1,60);
        temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
    }
    return resp;
}

unsigned char send_recv_cmd(unsigned char cmd, unsigned char * arg)
{
    send_cmd(cmd, arg);
    return recv_resp();
}

int read_data(unsigned int address, unsigned char * data_buf, int size)
{
    address += 57*512;
    unsigned char temp_buf[512];
    int i, bytes;
    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);
    }

    return size;
}

void read_data_block(unsigned int address, unsigned char * data_buf)
{
    printf("........reading block.........\n");
  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_16DIRECT(MMC_DATAOUT1_BASE,0);
  while(temp == 1)
  {
      send_clks(1,60);
      temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
  }
  //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_16DIRECT(MMC_DATAOUT1_BASE,0);
        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,arg1);
  send_clks(8,60);
  //printf(" Set block length resp == %x == \n",resp);
}
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MMC_CLK is

  port (

    avs_s1_clk        : in  std_logic;
    avs_s1_write      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_address    : in  std_logic_vector(4 downto 0);
    avs_s1_writedata  : in  std_logic_vector(31 downto 0);

        SD_CLK : out std_logic
  );

end MMC_CLK;

architecture rtl of MMC_CLK is
begin
  process (avs_s1_clk)
  begin
    if avs_s1_clk'event and avs_s1_clk = '1' then
        if avs_s1_chipselect = '1' then
            if avs_s1_write = '1' then
              SD_CLK <= avs_s1_writedata(0);
            end if;
        end if;
     end if;
  end process;
end architecture rtl;
```

```
-
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MMC_DATAIN is

  port (

    avs_s1_clk        : in  std_logic;
    avs_s1_write      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_address    : in  std_logic_vector(4 downto 0);
    avs_s1_writedata  : in  std_logic_vector(31 downto 0);

        SD_CMD                          : out std_logic
  );

end MMC_DATAIN;

architecture rtl of MMC_DATAIN is
begin
  process (avs_s1_clk)
  begin
    if avs_s1_clk'event and avs_s1_clk = '1' then
        if avs_s1_chipselect = '1' then
            if avs_s1_write = '1' then
              SD_CMD <= avs_s1_writedata(0);
            end if;
        end if;
     end if;
  end process;
end architecture rtl;




library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MMC_DATAOUT is

  port (

    avs_s1_clk        : in  std_logic;
    avs_s1_read      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_address    : in  std_logic_vector(4 downto 0);
    avs_s1_readdata  : out  std_logic_vector(31 downto 0);

        SD_DAT                         : in std_logic
  );

end MMC_DATAOUT;

architecture rtl of MMC_DATAOUT is
begin
  process (avs_s1_clk)
  begin
    if avs_s1_clk'event and avs_s1_clk = '1' then
        if avs_s1_chipselect = '1' then
            if avs_s1_read = '1' then
              avs_s1_readdata(0) <= SD_DAT;
            end if;
        end if;
     end if;
  end process;
end architecture rtl;



#ifndef MMC_HEADER_H
#define MMC_HEADER_H 1

#define CLK_LO IOWR_16DIRECT(MMC_CLK1_BASE,0,0)
#define CLK_HI IOWR_16DIRECT(MMC_CLK1_BASE,0,1)
#define DIN_LO IOWR_16DIRECT(MMC_DATAIN1_BASE,0,0)
#define DIN_HI IOWR_16DIRECT(MMC_DATAIN1_BASE,0,1)
#define NCS_LO IOWR_16DIRECT(MMC_NCS1_BASE,0,0)
#define NCS_HI IOWR_16DIRECT(MMC_NCS1_BASE,0,1)

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


#include<io.h>
#include<system.h>

void mmc_super_init(char *arg1);
int read_data(unsigned int address, unsigned char * data_buf, int size);
void read_data_block(unsigned int address, unsigned char * data_buf);
void delay(int i);
void send_clks(int a,int b);


#endif /*MMC_HEADER_H_*/




library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MMC_NCS is

  port (

    avs_s1_clk        : in  std_logic;
    avs_s1_write      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_address    : in  std_logic_vector(4 downto 0);
    avs_s1_writedata  : in  std_logic_vector(31 downto 0);

        SD_DAT3                         : out std_logic
  );

end MMC_NCS;

architecture rtl of MMC_NCS is
begin
  process (avs_s1_clk)
  begin
    if avs_s1_clk'event and avs_s1_clk = '1' then
        if avs_s1_chipselect = '1' then
            if avs_s1_write = '1' then
              SD_DAT3 <= avs_s1_writedata(0);
            end if;
        end if;
     end if;
  end process;
end architecture rtl;



/* 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;
```

```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;
}

/* @} */



/* 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_
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;
}

/* @} */



/* 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)
        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;

void fat_loop_files()
{

    //printf ("Inside fat loop\n");

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

        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) {
                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();
                           }
                           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;
}

/* @} */



#include <mmc_header.h>
#include <stdio.h>

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

unsigned char read_buffer[4096];
int read_buffer_addr = -1;

void delay(int i)
{
    int a;
    for(a=0;a<i;a++)
    {

    }
}

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

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

void mmc_disable()
{
    CLK_LO;
    DIN_HI;
    NCS_HI;
}

void send_cmd(unsigned  char cmd, unsigned char* arg)
{
   unsigned char *temp;
   unsigned char data;
   int i,j;
   NCS_LO;
   *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);
            data?DIN_HI:DIN_LO;
            send_clks(1,60);
        }
        temp++;
    }
    DIN_HI;
}

unsigned char recv_resp()
{
    unsigned char resp=0x00,temp=0x00;
    int i;
    temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
    while(temp == 1)
    {
        send_clks(1,60);
        temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
    }
    for(i=7;i>-1;i--)
    {
        temp = temp << i;
        resp = resp | temp;
        send_clks(1,60);
        temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
    }
    return resp;
}

unsigned char send_recv_cmd(unsigned char cmd, unsigned char * arg)
{
    send_cmd(cmd, arg);
    return recv_resp();
}

int read_data(unsigned int address, unsigned char * data_buf, int size)
{
    //printf("reading %d bytes at addr=%d\n", size, address);
    address += 57*512;
    unsigned char temp_buf[512];
    int i, bytes;
    if (size == 4096 && 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)
{
  //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_16DIRECT(MMC_DATAOUT1_BASE,0);
  while(temp == 1)
  {
      send_clks(1,60);
      temp=IORD_16DIRECT(MMC_DATAOUT1_BASE,0);
  }
  //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_16DIRECT(MMC_DATAOUT1_BASE,0);
        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,arg1);
  send_clks(8,60);
  //printf(" Set block length resp == %x == \n",resp);
}
```

```
#ifndef MMC_HEADER_H
#define MMC_HEADER_H 1

#define CLK_LO IOWR_16DIRECT(MMC_CLK1_BASE,0,0)
#define CLK_HI IOWR_16DIRECT(MMC_CLK1_BASE,0,1)
#define DIN_LO IOWR_16DIRECT(MMC_DATAIN1_BASE,0,0)
#define DIN_HI IOWR_16DIRECT(MMC_DATAIN1_BASE,0,1)
#define NCS_LO IOWR_16DIRECT(MMC_NCS1_BASE,0,0)
#define NCS_HI IOWR_16DIRECT(MMC_NCS1_BASE,0,1)

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


#include<io.h>
#include<system.h>

void mmc_super_init(char *arg1);
int read_data(unsigned int address, unsigned char * data_buf, int size);
void read_data_block(unsigned int address, unsigned char * data_buf);
void delay(int i);
void send_clks(int a,int b);


#endif /*MMC_HEADER_H_*/
```

```
/* 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;
}
```

```c
#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;
}

/* @} */



/* 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_b
                        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_se
        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';
}


/* @} */



/* 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 int        get_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);




/*---------------------------------------------*/
/* 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;
    }
  }
}



#include <io.h>
#include <system.h>
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include "jpeg.h"
#define MAX_X 640
#define MAX_Y 480

//int width, length;

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

#define SLOWNESS 200          // How slow to move the box (the bigge the number, the slower)



int recurse (int n)
{
 int a;
 int b[1000];
 printf("%ld ", &a);

 if(n ==0)
 return 1;

 return recurse(n-1);


}



int main()
{
  int x = 0, y = 0;
  int down = 1, right = 1;    // Flags Used to Tell what direction the box is moving in
  int r, g, b;
  int i, j;
  /* FILE * fp = fopen ("philips.ras", "r");
 if (fp == NULL)
    {
     printf("oooooooops file error");
    }
  int  i;
  char ch;
  for (i =0; i<320; i++)
  {

   ch =fgetc(fp); /*Skip first 32 bytes*/
   //printf("%c", ch);
  //}*/
                             // down = 0 (means UP), right = 0(means left)
    // unsigned short int width, height;



  unsigned int add =0 ;
 /* for(i=0; i<10; i++)
  {
   int *a = (int*)malloc(1000000);
    printf("%ld ",a );
    //free(a);

  }*/
/*  printf("\n");
  recurse(1000);

    for(i=0; i<10; i++)
  {
   int *a = (int*)malloc(1000000);
    printf("%ld ",a );
    //free(a);

  }*/
//  while(1)
    fat_loop_files();

 /*while(1)
 {

 printf("Starting to Decode!\n");
 jpeg_decoder();

 }*/


  printf("Goodbye!\n");
  return 0;
}



/*-------------------------------------------*/
/* 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];
  }
```

```
}




/*-----------------------------------------*/
/* File : jpeg.c, main for jfif decoder    */
/* Author : Pierre Guerrier, march 1998    */
/*                                         */
/* 19/01/99  Edited by Koen van Eijk       */
/*-----------------------------------------*/

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

#include "jpeg.h"
//#include "input_array.c"

/* real declaration of global variables here */
/* see jpeg.h for more info         */

cd_t   comp[3];       /* descriptors for 3 components */
PBlock *MCU_buff[10]; /* decoded DCT blocks buffer */
int    MCU_valid[10]; /* components of above MCU blocks */

PBlock *QTable[4];  /* quantization tables */
int    QTvalid[4];

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

unsigned char *ColorBuffer = NULL; /* MCU after color conversion */
unsigned char *FrameBuffer = NULL; /* complete final RGB image */
FBlock        *FBuff = NULL;        /* scratch frequency buffer */
PBlock        *PBuff = NULL;        /* scratch pixel buffer */

int in_frame, curcomp;   /* frame started ? current component ? */
int MCU_row, MCU_column; /* current position in MCU unit */

FILE    *fi;            /* input  File stream pointer   */
FILE    *fo;            /* output File stream pointer   */


int stuffers = 0;   /* stuff bytes in entropy coded segments */
int passed = 0;     /* bytes passed when searching markers */

int verbose = 0;


void initialize()
{
    input_pointer = 0;
    stuffers = 0;   /* stuff bytes in entropy coded segments */
    passed = 0;     /* bytes passed when searching markers */

    verbose = 0;
}

/*-----------------------------------------------------------------*/
/*      MAIN        MAIN        MAIN            */
/*-----------------------------------------------------------------*/

int
jpeg_decoder(void)
{
    // printf ("Start of jpeg decoder\n");
    initialize();

    /*  char fnam[256];
        char *p;*/
    unsigned int aux, mark;
    int n_restarts, restart_interval, leftover; /* RST check */
    int i,j;

    /*if (argc != 2) {
      fprintf(stderr, "Please provide a JPEG file as argument.\n");
      exit(0);
      }

      Check for presence of .jpg file extension:
      if ((p = strrchr(argv[1], '.')) && !strcmp(p, ".jpg"))
      Indeed such extension; remove it:
      *p = '\0';
      sprintf(fnam, "%s.jpg", argv[1]);*/


    /*fi = fopen(fnam,"rb");
      if (fi == NULL) {
      fprintf(stderr, "Could not open input file %s.\n", fnam);
      exit(0);
      }

      Prepare raster format output file:/
      sprintf(fnam, "%s.ras", argv[1]);
      fo=fopen(fnam,"wb");
      if (fo == NULL) {
      fprintf(stderr, "Could not open output file %s.\n", fnam);
      exit(0);
      }
      sprintf(fnam, "%s.spy", argv[1]);
      */
    /* First find the SOI marker: */
    aux = get_next_MK(fi);
    if (aux != SOI_MK)
    {
        aborted_stream(fi, fo);
        return;
    }

    if (verbose)
        fprintf(stderr, "\tINFO:\tFound the SOI marker!\n");
    in_frame = 0;
    restart_interval = 0;
    for (i = 0; i < 4; i++)
        QTvalid[i] = 0;

    /* Now process segments as they appear: */
    do {
        mark = get_next_MK(fi);

        switch (mark) {
```

```
case SOF_MK:
    if (verbose)
        fprintf(stderr, "\tINFO:\tFound the SOF marker!\n");
    in_frame = 1;
    get_size(fi);   /* header size, don't care */

    /* load basic image parameters */
    my_get(fi); /* precision, 8bit, don't care */
    y_size = get_size(fi);
    x_size = get_size(fi);
    if (verbose)
        fprintf(stderr, "\tINFO:\tImage size is %d by %d\n", x_size, y_size);

    n_comp = my_get(fi);    /* # of components */
    if (verbose) {
        fprintf(stderr, "\tINFO:\t");
        switch (n_comp)
        {
            case 1:
                fprintf(stderr, "Monochrome");
                break;
            case 3:
                fprintf(stderr, "Color");
                break;
            default:
                fprintf(stderr, "Not a");
                break;
        }
        fprintf(stderr, " JPEG image!\n");
    }

    for (i = 0; i < n_comp; i++) {
        /* component specifiers */
        comp[i].CID = my_get(fi);
        aux = my_get(fi);
        comp[i].HS = first_quad(aux);
        comp[i].VS = second_quad(aux);
        comp[i].QT = my_get(fi);
    }
    if ((n_comp > 1) && verbose)
        fprintf(stderr, "\tINFO:\tColor format is %d:%d:%d, H=%d\n",
                comp[0].HS * comp[0].VS,
                comp[1].HS * comp[1].VS,
                comp[2].HS * comp[2].VS,
                comp[1].HS);

    if (init_MCU() == -1)
    {
        aborted_stream(fi, fo);
        return;
    }

    /* dimension scan buffer for YUV->RGB conversion */
    FrameBuffer =
        (unsigned char *) malloc( (size_t) x_size * y_size * n_comp);
    ColorBuffer =
        (unsigned char *) malloc( (size_t) MCU_sx * MCU_sy * n_comp);
    /* if(FrameBuffer== NULL)
       {
       printf("Trying to malloc\n");
       FrameBuffer =
       (unsigned char *) malloc(500000);
       ColorBuffer =
       (unsigned char *) malloc(500000);
       }*/
    FBuff = (FBlock *) malloc(sizeof(FBlock));
    PBuff = (PBlock *) malloc(sizeof(PBlock));

    if (FrameBuffer == NULL) {
        fprintf(stderr, "\tERROR:\tCould not allocate %d bytes for FrameBuffer!\n", x_size * y_s
        exit(1);
    }
    if (ColorBuffer == NULL) {
        fprintf(stderr, "\tERROR:\tCould not allocate pixel storage for ColorBuffer!\n");
        exit(1);
    }
    if (FBuff == NULL) {
        fprintf(stderr, "\tERROR:\tCould not allocate pixel storage for FBuff\n");
        exit(1);
    }
    if (PBuff == NULL) {
        fprintf(stderr, "\tERROR:\tCould not allocate pixel storage for PBuff\n");
        exit(1);
    }
    break;

case DHT_MK:
    if (verbose)
        fprintf(stderr, "\tINFO:\tDefining Huffman Tables\n");
    if (load_huff_tables(fi) == -1)
    {
        aborted_stream(fi, fo);
        return;
    }
    break;

case DQT_MK:
    if (verbose)
        fprintf(stderr,
                "\tINFO:\tDefining Quantization Tables\n");
    if (load_quant_tables(fi) == -1)
    {
        aborted_stream(fi, fo);
        return;
    }
    break;

case DRI_MK:
    get_size(fi);   /* skip size */
    restart_interval = get_size(fi);
    if (verbose)
        fprintf(stderr, "%\tINFO:\tDefining Restart Interval %d\n", restart_interval);
    break;

case SOS_MK:        /* lots of things to do here */
    if (verbose)
        fprintf(stderr, "\tINFO:\tFound the SOS marker!\n");
    get_size(fi); /* don't care */
    aux = my_get(fi);
    if (aux != (unsigned int) n_comp) {
        fprintf(stderr, "\tERROR:\tBad component interleaving!\n");
        aborted_stream(fi, fo);
        return;
    }
```

```
                for (i = 0; i < n_comp; i++) {
                    aux = my_get(fi);
                    if (aux != comp[i].CID) {
                        fprintf(stderr, "\tERROR:\tBad Component Order!\n");
                        aborted_stream(fi, fo);
                        return;
                    }
                    aux = my_get(fi);
                    comp[i].DC_HT = first_quad(aux);
                    comp[i].AC_HT = second_quad(aux);
                }
                get_size(fi); my_get(fi);   /* skip things */

                MCU_column = 0;
                MCU_row = 0;
                clear_bits();
                reset_prediction();

                /* main MCU processing loop here */
                if (restart_interval) {
                    n_restarts = ceil_div(mx_size * my_size, restart_interval) - 1;
                    leftover = mx_size * my_size - n_restarts * restart_interval;
                    /* final interval may be incomplete */

                    for (i = 0; i < n_restarts; i++) {
                        for (j = 0; j < restart_interval; j++)
                            process_MCU(fi);
                        /* proc till all EOB met */

                        aux = get_next_MK(fi);
                        if (!RST_MK(aux)) {
                            fprintf(stderr, "\tERROR:\tLost Sync after interval!\n");
                            aborted_stream(fi, fo);
                            return;
                        }
                        else if (verbose)
                            fprintf(stderr, "\tINFO:\tFound Restart Marker\n");

                        reset_prediction();
                        clear_bits();
                    }       /* intra-interval loop */
                }
                else
                    leftover = mx_size * my_size;

                /* process till end of row without restarts */
                for (i = 0; i < leftover; i++)
                    process_MCU(fi);

                in_frame = 0;
                break;

            case EOI_MK:
                if (verbose)
                    fprintf(stderr, ":\tINFO:\tFound the EOI marker!\n");
                if (in_frame)
                {
                    aborted_stream(fi, fo);
                    return;
                }

                if (verbose)
                    fprintf(stderr, "\tINFO:\tTotal skipped bytes %d, total stuffers %d\n",
                            passed, stuffers);
                /*fclose(fi);*/
                RGB_save(fo);
                /*fclose(fo);*/
                free_structures();
#ifdef SPY
                output_stats(fnam);
#endif
                fprintf(stderr, "\nDone.\n");
                return;
                break;

            case COM_MK:
                if (verbose)
                    fprintf(stderr, "\tINFO:\tSkipping comments\n");
                skip_segment(fi);
                break;

            case EOF:
                if (verbose)
                    fprintf(stderr, "\tERROR:\tRan out of input data!\n");
                aborted_stream(fi, fo);
                return;

            default:
                if ((mark & MK_MSK) == APP_MK) {
                    if (verbose)
                        fprintf(stderr, "\tINFO:\tSkipping application data\n",
                                ftell(fi));
                    skip_segment(fi);
                    break;
                }
                if (RST_MK(mark)) {
                    reset_prediction();
                    break;
                }
                /* if all else has failed ... */
                fprintf(stderr, "\tWARNING:\tLost Sync outside scan, %d!\n", mark);
                aborted_stream(fi, fo);
                return;
                break;
        } /* end switch */
    }
    while (1);

    return 0;
}




/*-----------------------------------------*/
/* 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:
          fprintf(stderr, "\tERROR:\tRan out of bit stream\n");
          aborted_stream(fi, fo);
          break;

        case 0x00:
          stuffers++;
          break;

        default:
          if (RST_MK(0xFF00 | aux))
            fprintf(stderr, "%ld:\tERROR:\tSpontaneously found restart!\n",
                    ftell(fi));
          fprintf(stderr, "%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:
        fprintf(stderr, "\tERROR:\tRan out of bit stream\n");
        aborted_stream(fi, fo);
        break;

      case 0x00:
        stuffers++;
        break;

      default:
        if (RST_MK(0xFF00 | aux))
          fprintf(stderr, "\tERROR:\tSpontaneously found restart!\n");
        fprintf(stderr, "\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)
      fprintf(stderr, "\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)
        fprintf(stderr, "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) {
      fprintf(stderr, "\tERROR:\tBad QTable precision!\n");
      return -1;
    }
    id = second_quad(aux);
    if (verbose)
      fprintf(stderr, "\tINFO:\tLoading table %d\n", id);
    QTable[id] = (PBlock *) malloc(sizeof(PBlock));
    if (QTable[id] == NULL) {
      fprintf(stderr, "\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++)
         fprintf(stderr, "%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) {
        fprintf(stderr, "\tERROR:\tCould not allocate MCU buffers!\n");
        exit(1);
      }
      k++;
      if (k == 10) {
        fprintf(stderr, "\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)
      fprintf(stderr, "\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 : 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: */
```

```c
#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) {
      fprintf(stderr, "\tERROR:\tBad HTable identity %d!\n",id);
      return -1;
    }
    id = HUFF_ID(hclass, id);
    if (verbose)
      fprintf(stderr, "\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;
            fprintf(stderr, "\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)
      fprintf(stderr, "\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:
    fprintf(stderr, "\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 */
    fprintf(stderr, "%ld:\tWARNING:\tFalling out of truncated tree !\n");
    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 IOWR_RECT_DATA(base, offset, data) \
  IOWR_16DIRECT(base, (offset)*2, 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++)
      fprintf(stderr, "\t%d", S->block[i][j]);
    fprintf(stderr, "\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++)
      fprintf(stderr, "\t%d", S->block[i][j]);
    fprintf(stderr, "\n");
  }
}
```

```c
/* 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)
      fprintf(stderr, "\t%1d", !!(c & bitmask));
    fprintf(stderr, "\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");

  fprintf(stderr, "\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);
}
```

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

/* 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;






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("%d ", x_size* n_comp);
 //  printf("%d", 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++);
  }
  IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0003, (0x0001));
  IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0004, (0x0001));


unsigned int count =0;
unsigned int addr =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_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);
        }
  IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0003, (new_width ));
  IOWR_RECT_DATA(VGA_RASTER_0_BASE, 0x0004, (new_height));

      // clean_up();
}
```

```
--
-- Imagic top-level module.  This handles SDRAM and Nios system routing
--
-- From code produced by 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;

entity Imagic 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
    LEDR : out std_logic_vector(17 downto 0);      -- Red LEDs

    -- RS-232 interface

    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_EN,                    -- 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 : in 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 Imagic;

architecture rtl of Imagic is

  component sdram_pll is
        PORT
        (
                signal inclk0  : IN STD_LOGIC;
                signal c0           : OUT STD_LOGIC
        );
  end component;

  component nios_system is
   port (
-- ++ WED FINAL PROJECT
   -- Add PORTs to nios_system created by SOPC
   -- Global signals
   signal clk      : IN STD_LOGIC;
   signal reset_n  : IN STD_LOGIC;

   -- the_mmc_clk1
   signal SD_CLK_from_the_mmc_clk1 : OUT STD_LOGIC;

   -- the_mmc_datain1
   signal SD_CMD_from_the_mmc_datain1 : OUT STD_LOGIC;

   -- the_mmc_dataout1
   signal SD_DAT_to_the_mmc_dataout1 : IN STD_LOGIC;

   -- the_mmc_ncs1
   signal SD_DAT3_from_the_mmc_ncs1 : OUT STD_LOGIC;

   -- the_sram
   signal SRAM_ADDR_from_the_vga_raster_0        : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
   signal SRAM_CE_N_from_the_vga_raster_0        : OUT STD_LOGIC;
   signal SRAM_DQ_to_and_from_the_vga_raster_0  : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
   signal SRAM_LB_N_from_the_vga_raster_0       : OUT STD_LOGIC;
   signal SRAM_OE_N_from_the_vga_raster_0       : OUT STD_LOGIC;
   signal SRAM_UB_N_from_the_vga_raster_0       : OUT STD_LOGIC;
   signal SRAM_WE_N_from_the_vga_raster_0       : OUT STD_LOGIC;

   -- the_sdram
   signal zs_addr_from_the_sdram : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
   signal zs_ba_from_the_sdram : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
   signal zs_cas_n_from_the_sdram : OUT STD_LOGIC;
   signal zs_cke_from_the_sdram : OUT STD_LOGIC;
   signal zs_cs_n_from_the_sdram : OUT STD_LOGIC;
   signal zs_dq_to_and_from_the_sdram : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
   signal zs_dqm_from_the_sdram : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
   signal zs_ras_n_from_the_sdram : OUT STD_LOGIC;
   signal zs_we_n_from_the_sdram : OUT STD_LOGIC;
   -- the_vga_raster_0
   signal VGA_BLANK_from_the_vga_raster_0 : OUT STD_LOGIC;
   signal VGA_B_from_the_vga_raster_0     : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
   signal VGA_CLK_from_the_vga_raster_0   : OUT STD_LOGIC;
   signal VGA_G_from_the_vga_raster_0     : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
   signal VGA_HS_from_the_vga_raster_0    : OUT STD_LOGIC;
   signal VGA_R_from_the_vga_raster_0     : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
   signal VGA_SYNC_from_the_vga_raster_0  : OUT STD_LOGIC;
   signal VGA_VS_from_the_vga_raster_0    : OUT STD_LOGIC
-- -- WED FINAL PROJECT
  );
  end component;

  signal clk25 : std_logic := '0';

begin

  nios: nios_system port map (
-- ++ WED FINAL PROJECT
-- Route data to nios_system (for VGA Periphial + SDRAM)
    reset_n => '1',
    clk => CLOCK_50,

        SD_CLK_from_the_mmc_clk1 => SD_CLK,
    SD_CMD_from_the_mmc_datain1 => SD_CMD,
    SD_DAT_to_the_mmc_dataout1 => SD_DAT,
        SD_DAT3_from_the_mmc_ncs1 => SD_DAT3,

    VGA_CLK_from_the_vga_raster_0     => VGA_CLK,
    VGA_HS_from_the_vga_raster_0      => VGA_HS,
    VGA_VS_from_the_vga_raster_0      => VGA_VS,
    VGA_BLANK_from_the_vga_raster_0   => VGA_BLANK,
    VGA_SYNC_from_the_vga_raster_0    => VGA_SYNC,
    VGA_R_from_the_vga_raster_0       => VGA_R,
    VGA_G_from_the_vga_raster_0       => VGA_G,
    VGA_B_from_the_vga_raster_0       => VGA_B,
    SRAM_ADDR_from_the_vga_raster_0   => SRAM_ADDR,
    SRAM_CE_N_from_the_vga_raster_0   => SRAM_CE_N,
    SRAM_DQ_to_and_from_the_vga_raster_0   => SRAM_DQ,
    SRAM_LB_N_from_the_vga_raster_0   => SRAM_LB_N,
    SRAM_OE_N_from_the_vga_raster_0   => SRAM_OE_N,
    SRAM_UB_N_from_the_vga_raster_0   => SRAM_UB_N,
    SRAM_WE_N_from_the_vga_raster_0   => SRAM_WE_N,

    -- the_sdram
    zs_addr_from_the_sdram => DRAM_ADDR,
    zs_ba_from_the_sdram(1) => DRAM_BA_1,
    zs_ba_from_the_sdram(0) => 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(1) => DRAM_UDQM,
    zs_dqm_from_the_sdram(0) => DRAM_LDQM,
    zs_ras_n_from_the_sdram => DRAM_RAS_N,
    zs_we_n_from_the_sdram => DRAM_WE_N
-- -- WED LAB3
  );

  pll_clock: sdram_pll port map (inclk0 => CLOCK_50,
                                 c0 => DRAM_CLK);

  HEX7     <= "0001001";                -- Leftmost
  HEX6     <= "0000110";
  HEX5     <= "1000111";
  HEX4     <= "1000111";
  HEX3     <= "1000000";
  HEX2     <= (others => '1');
  HEX1     <= (others => '1');
  HEX0     <= (others => '1');          -- Rightmost
  LEDG     <= (others => '1');
  LEDR     <= (others => '1');
  LCD_ON   <= '1';
  LCD_BLON <= '1';

  -- Set all bidirectional ports to tri-state
  FL_DQ       <= (others => 'Z');
 -- SRAM_DQ     <= (others => 'Z');  -- ++ WED FINAL PROJECT -- Controlled in periphial
  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 rtl;



-------------------------------------------------------------------------------
--
-- VGA raster display for Imagic
-- This also acts as an SRAM controller
-- Since the VGA must read from the SRAM at a rate
-- of 25 MHZ, the VGA controller is given full
-- control of the SRAM.  If this nios wants to write
-- data to the SRAM, it must ask the VGA controller to do so.
--
--
--  From code produced by Stephen A. Edwards,
--  Columbia University, sedwards@cs.columbia.edu
--
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity vga_raster is

  port (
-- ++ WED FINAL PROJECT
    -- Add Bus Interface
    avs_s1_clk        : in  std_logic;
    avs_s1_reset_n    : in  std_logic;
    avs_s1_read       : in  std_logic;
    avs_s1_write      : in  std_logic;
    avs_s1_chipselect : in  std_logic;
    avs_s1_waitrequest      : out  std_logic;
    avs_s1_address    : in  std_logic_vector(4 downto 0);
    avs_s1_readdata   : out std_logic_vector(15 downto 0);
    avs_s1_writedata  : in  std_logic_vector(15 downto 0);

    -- Control Singals for SRAM
    SRAM_DQ   : inout std_logic_vector(15 downto 0);
    SRAM_ADDR : out std_logic_vector(17 downto 0);
    SRAM_UB_N,
    SRAM_LB_N,
    SRAM_WE_N,
    SRAM_CE_N,
    SRAM_OE_N : out std_logic;
-- -- WED FINAL PROJECT

    reset : in std_logic;

    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]
  );

end vga_raster;

architecture rtl of vga_raster is
  -- Video parameters

  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;
  constant MID_X                : integer := 320;
  constant MID_Y        : integer := 240;

  -- Signals for the video controller
  signal Hcount : std_logic_vector(9 downto 0);  -- Horizontal position (0-800)
  signal Vcount : std_logic_vector(9 downto 0);  -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;

-- ++ WED FINAL_PROJECT
  signal IMAGE_HEND      : std_logic_vector(8 downto 0) := "000000001";   -- Image Width from NIOS
  signal IMAGE_VEND      : std_logic_vector(8 downto 0) := "000000001";   -- Image Height from NIOS
  signal IMAGE_RED       : std_logic_vector(4 downto 0) := "00000";       -- Red Pixel data from NIOS
  signal IMAGE_GREEN     : std_logic_vector(4 downto 0) := "00000";       -- Green Pixel data from NIOS
  signal IMAGE_BLUE      : std_logic_vector(4 downto 0) := "00000";       -- Blue Pixel data from NIOS
  signal BG_RED          : std_logic_vector(4 downto 0) := "00000";       -- Red background data from NIOS
  signal BG_GREEN        : std_logic_vector(4 downto 0) := "00000";       -- Green background data from NIOS
  signal BG_BLUE         : std_logic_vector(4 downto 0) := "00000";       -- Blue background data from NIOS
  signal IMAGE_HSTART : integer;                   -- Used during display processing
  signal IMAGE_VSTART : integer;                   -- Used during display processing
  signal image_h, image_v, image : std_logic;     -- Image area
  signal clk                         : std_logic := '0';    -- Internally used 25MHz clock
  signal RAM_ADDR                    : std_logic_vector(17 downto 0) := "000000000000000000";
  signal RAM_WRITE_ADDR              : std_logic_vector(17 downto 0) := "000000000000000000";
  signal write_data, write_background  : std_logic := '0';  -- Flag
  -- BUFFERS for SRAM
  signal ADDRESS_TO_SRAM             : std_logic_vector(17 downto 0) := "000000000000000000";
  signal WRITEDATA_TO_SRAM           : std_logic_vector(15 downto 0) := "0000000000000000";
  signal READDATA_FROM_SRAM          : std_logic_vector(15 downto 0) := "0000000000000000";
  signal WRITEENABLE_TO_SRAM         : std_logic := '0';
-- -- WED FINAL_PROJECT

  signal vga_hblank, vga_hsync,
         vga_vblank, vga_vsync : std_logic;  -- Sync. signals

begin

-- ++ WED FINAL_PROJECT
  -- If writing data, force SRAM bus, otherwise let it float
  SRAM_DQ <= WRITEDATA_TO_SRAM when WRITEENABLE_TO_SRAM = '0' else
             (others => 'Z');
  -- If data is on SRAM bus, store it
  READDATA_FROM_SRAM <= SRAM_DQ;
  -- Address going out to SRAM
  SRAM_ADDR <= ADDRESS_TO_SRAM;
  -- Write Enable Signal going out to SRAM
  SRAM_WE_N <= WRITEENABLE_TO_SRAM;
  -- These can all be set to zero.  Only need simple mode of operation.
  SRAM_UB_N <= '0';
  SRAM_LB_N <= '0';
  SRAM_CE_N <= '0';
  SRAM_OE_N <= '0';

  -- Split the clock for the VGA (from 50 Mhz to 25 Mhz)
  process (avs_s1_clk)
  begin
    if avs_s1_clk'event and avs_s1_clk = '1' then
      clk <= not clk;
    end if;
  end process;

  -- State Machine handling SRAM control and also accepting commands from
  -- NIOS
  -- There are three 'periods' of operation
  --   Period 1)  Change SRAM read address
  --              Raise any flags set during write period.  The low to high
  --              transition is what actually writes the data to the SRAM
  --   Period 2)  Accept commands from NIOS.
  --              This includes accepting data to be written to SRAM, accepting
  --              image height/width that is written to buffers, and SRAM
  --              write address that is also stored in a buffer.
  --              The actual writing of SRAM is initiated here as well.
  --   Period 3)  Read pixel data from SRAM.  If the SRAM has not recently be written
  --              read data from SRAM to be displayed on screen.  If the SRAM has
  --              been written, there is not enough time for the SRAM to drive the bus
  --              so we just display the previous pixel again (the human eye can't tell)
  SRAM_CONTROLLER : process(avs_s1_clk, avs_s1_reset_n, reset)
  begin
    if reset = '1' then
      avs_s1_readdata <= (others => '0');
    elsif avs_s1_clk'event and avs_s1_clk = '1' then
       if clk = '1' then   --(period 1)
         WRITEENABLE_TO_SRAM <= '1';   -- This is the READ portion, so disable writes
         avs_s1_waitrequest <= '0';    -- Tell bus during next cycle data can be processed
         write_data <= '0';            -- Always reset flag, this is READ portion
         ADDRESS_TO_SRAM <= RAM_ADDR;  -- Update the pixel that will be read during down tick
      else                --(period 2)
        -- check if data should be written
        if avs_s1_write = '1' then              -- They want to write something
          if avs_s1_address = "00000" then      -- Data
            ADDRESS_TO_SRAM <= RAM_WRITE_ADDR;  -- Send SRAM address we had stored in buffer
            write_data <= '1';                  -- Raise flag, DONT read anything, we have to drive SRAM bus
            WRITEDATA_TO_SRAM <= avs_s1_writedata(15 downto 0);     --What SRAM Data?
            WRITEENABLE_TO_SRAM <= '0';         -- Tell SRAM to start writing, we can do this right away
            if write_background = '1' then
                write_background <= '0';
            -- If this is the first pixel of the image, save it for the background
                BG_RED <= avs_s1_writedata(14 downto 10);
                BG_GREEN <= avs_s1_writedata(9 downto 5);
                BG_BLUE <= avs_s1_writedata(4 downto 0);
            end if;
          elsif avs_s1_address = "00001" then    -- Address (lower 16 bits)
            RAM_WRITE_ADDR(15 downto 0) <= avs_s1_writedata(15 downto 0); -- store SRAM address they want to write to buffer
                  elsif avs_s1_address = "00010" then    -- Address (upper 2 bits)
            RAM_WRITE_ADDR(17 downto 16) <= avs_s1_writedata(1 downto 0);
          elsif avs_s1_address = "00011" then    -- Image Width
            IMAGE_HEND <= avs_s1_writedata(8 downto 0);
            -- If the image size is '1', assume the image is being written to SRAM
            -- and next data point will be the first pixel point, and that will be used as the backgroud.
            if avs_s1_writedata(8 downto 0) = "000000001" then
               write_background <= '1';
            end if;
          elsif avs_s1_address = "00100" then    -- Image Height
            IMAGE_VEND <= avs_s1_writedata(8 downto 0);
          end if;
        end if;
      avs_s1_waitrequest <= '1';   -- Tell bus during next cycle data can NOT be processed, they have to wait
      end if;
   elsif avs_s1_clk'event and avs_s1_clk = '0' then
     if clk = '1' and write_data = '0' then   -- Read Data from SRAM (period 3)
       IMAGE_RED <= READDATA_FROM_SRAM(14 downto 10);
       IMAGE_GREEN <= READDATA_FROM_SRAM(9 downto 5);
       IMAGE_BLUE <= READDATA_FROM_SRAM(4 downto 0);
     end if;
   end if;
  end process SRAM_CONTROLLER;
  -- Convert data read from NIOS into a useful form for us to use.
  IMAGE_HSTART <= MID_X - conv_integer(IMAGE_HEND(8 downto 1));
  IMAGE_VSTART <= MID_Y - conv_integer(IMAGE_VEND(8 downto 1));
-- -- WED FINAL_PROJECT

  HCounter : process (clk, reset)
  begin
    if reset = '1' then
      Hcount <= (others => '0');
    elsif clk'event and clk = '1' then
      if 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, reset)
  begin
    if reset = '1' then
      Vcount <= (others => '0');
    elsif clk'event and clk = '1' then
      if 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, reset)
  begin
    if reset = '1' then
      vga_hsync <= '1';
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
      end if;
    end if;
  end process HSyncGen;

  HBlankGen : process (clk, reset)
  begin
    if reset = '1' then
      vga_hblank <= '1';
    elsif clk'event and clk = '1' then
      if 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, reset)
  begin
    if reset = '1' then
      vga_vsync <= '1';
    elsif clk'event and clk = '1' then
      if 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, reset)
  begin
    if reset = '1' then
      vga_vblank <= '1';
    elsif clk'event and clk = '1' then
      if 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;

  -- Control the position of the image

  ImageHGen : process (clk, reset)
  begin
    if reset = '1' then
      image_h <= '1';
    elsif clk'event and clk = '1' then
-- ++ WED FINAL_PROJECT
      if Hcount = HSYNC + HBACK_PORCH + IMAGE_HSTART
      then
        -- Starting displaying image
        image_h <= '1';
      elsif Hcount = HSYNC + HBACK_PORCH + IMAGE_HSTART + conv_integer(IMAGE_HEND) then
        -- Stop displaying image
        image_h <= '0';
-- -- WED FINAL_PROJECT
      end if;
    end if;
  end process ImageHGen;

  ImageVGen : process (clk, reset)
  begin
    if reset = '1' then
      image_v <= '0';
    elsif clk'event and clk = '1' then
      if EndOfLine = '1' then
-- ++ WED FINAL_PROJECT
        if Vcount = VSYNC + VBACK_PORCH - 1 + IMAGE_VSTART
        then
          -- Starting displaying image
          image_v <= '1';
        elsif Vcount = VSYNC + VBACK_PORCH - 1 + IMAGE_VSTART + conv_integer(IMAGE_VEND) then
          -- Stop displaying image
          image_v <= '0';
-- -- WED FINAL_PROJECT
        end if;
      end if;
    end if;
  end process ImageVGen;

-- ++ WED FINAL_PROJECT
  -- If the pixel that is being written to the
  -- screen is within image region, set image flag
  image <= image_h and image_v;

  -- Every 25 MHz, if image is being display, increment
  -- the address in the SRAM.  Since the pixels
  -- are being display continously, this gets the next
  -- pixel that needs to be displayed.  (increment by
  -- two since address is byte addressed).  When
  -- the screen reachs the lower right pixel, reset
  -- the ram address.
  RamAddr : process (clk, reset)
  begin
    if clk'event and clk = '0' then
       if EndOfField = '1' and EndofLine= '1' then
          RAM_ADDR <= "000000000000000000";  -- reset address
       elsif image_h = '1' and image_v = '1' then
          RAM_ADDR <= RAM_ADDR + 2;          -- increment
       end if;
    end if;
  end process RamAddr;
-- -- WED FINAL_PROJECT

  -- Registered video signals going to the video DAC

  VideoOut: process (clk, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk'event and clk = '1' then
-- ++ WED FINAL_PROJECT
      if image = '1' then
        if IMAGE_HEND = "000000001" then
            -- if image height is 1, assume background should be shown
            VGA_R(8 downto 4) <= BG_RED;
            VGA_G(8 downto 4) <= BG_GREEN;
            VGA_B(8 downto 4) <= BG_BLUE;
            VGA_R(3 downto 0) <= BG_RED(4 downto 1);
            VGA_G(3 downto 0) <= BG_GREEN(4 downto 1);
            VGA_B(3 downto 0) <= BG_BLUE(4 downto 1);
        else
            -- Copy Data from image into lower and upper halves of pixel output
            VGA_R(8 downto 4) <= IMAGE_RED;
            VGA_G(8 downto 4) <= IMAGE_GREEN;
            VGA_B(8 downto 4) <= IMAGE_BLUE;
            VGA_R(3 downto 0) <= IMAGE_RED(4 downto 1);
            VGA_G(3 downto 0) <= IMAGE_GREEN(4 downto 1);
            VGA_B(3 downto 0) <= IMAGE_BLUE(4 downto 1);
        end if;
      elsif vga_hblank = '0' and vga_vblank ='0' then
        -- Set background of image to first pixel value of image
        VGA_R(8 downto 4) <= BG_RED;
        VGA_G(8 downto 4) <= BG_GREEN;
        VGA_B(8 downto 4) <= BG_BLUE;
        VGA_R(3 downto 0) <= BG_RED(4 downto 1);
        VGA_G(3 downto 0) <= BG_GREEN(4 downto 1);
        VGA_B(3 downto 0) <= BG_BLUE(4 downto 1);
-- -- WED FINAL_PROJECT
      else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;

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

end rtl;
```

8 Acknowledgements

JPEG Decoder originally written by Pierre et al., has been reused in this project. A part of Freedos32 from sourceforge has been reused for the FAT file system code.

The VGA/SRAM periphial is based on code supplied from Stephen Edwards for Embedded Systems Lab 3.

References

[1] Using the SDRAM Memory on Alteras DE2 Board.

[2] 256K x 16 High Speed Asynchronous CMOS Static RAM With 3.3V Supply: Reference Manual

[3] Avalon Memory-Mapped Interface Specification

[4] Wikipedia - File Allocation Table

[5] FreeDOS-32: FAT file system driver project page from SourceForge

[6] J. Jones, JPEG Decoder Design, Sr. Design Document EE175WS00-11, Electrical Engineering Dept., University of California, Riverside, CA, 2000

[7] Jun Li, Interfacing a MultiMediaCard to the LH79520 System-On-Chip

[8] Engineer-to-Engineer Note Interfacing MultiMediaCard with ADSP-2126x SHARC Processors