Digital Picture-Frame

W4840 Embedded Systems Design
Spring 2006

Mothler Dalexis
Alan Rabinowitz
Rafael Guevara

1. Introduction

Image representation has progressed steadily through the years, beginning with cave drawings, moving on to paintings, and finally photographic pictures. Until recently pictures captured on photographic film needed to be processed by a professional with chemicals, a time consuming process. During the last couple of years, however, technology has significantly evolved making these procedures obsolete. Now, a digital camera is all that is needed to allow for images to be seen seconds later. Still, difficulties arise in attempting to share the captured images. Portable devices still need to be developed in order to allow for quick and easy access to them.

Using this as our motivation we intended to build a digital picture frame. We would accomplish this using an existing JPEG file, a JPEG decoder, and an LCD screen. The resulting image would have a resolution of 640x480. This would not have a user-interface, thus making it a limited device.

2. Implementation

While we were initially assuming that the JPEG file that we wished to display would already be given on the FPGA board, we attempted to implement the serial port to allow for more flexibility in choosing what picture the user wanted to display. In order to allow for this, we used test232, a program which took data from a host computer and sent it through the serial port onto the MicroBlaze processor. We had an alternative method of sending data, which was to use minicom. However, we wanted to understand the steps involved in the data transmission and minicom would not allow this. Test232's

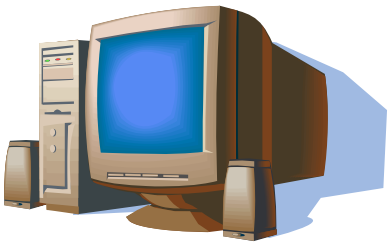
flexibility, on the other hand, allowed for a more precise as to how exactly the image was read.

While having the serial port send our image to the processor was one possible way of displaying the JPEG, it was also possible to simply remove the serial port from the entire scenario. This entailed already having the image on the processor. To do so, a simple C program was written that would convert the JPEG image into an array of decimals that would then be transmitted to the frame. Naturally, this method did not make using test232 necessary, since the image would be directly fed to the monitor.

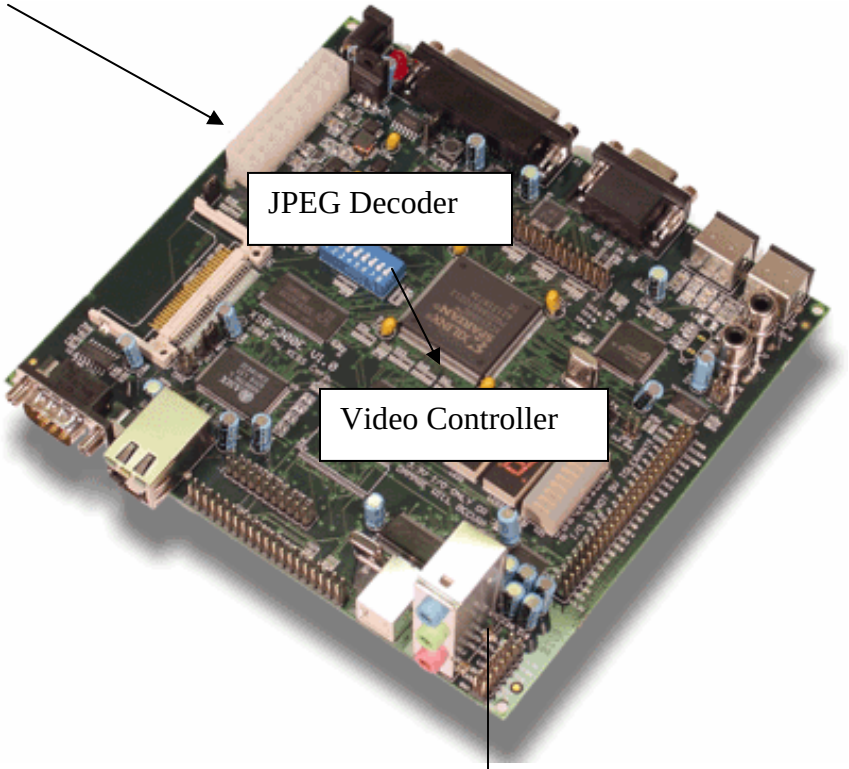
Following last year's group, we initially decided to use an open-source JPEG decoder found at the Independent JPEG group website and streamline the code to allow it to fit on the MicroBlaze processor. However, after actually attempting to do this, we discovered it was much too difficult to understand the program, let alone to attempt to simplify it. Therefore, a different JPEG decoder was provided, which ultimately proved to be much easier to comprehend and to streamline.

Once the JPEG decoder was cut down to fit onto the processor, the bulk of the project was complete. All that was left to do was to simply display the image onto the frame, the VGA monitor. To do this, the video controller on MicroBlaze was used to the send the bytes from memory to video.

3. Design



Host Computer



VGA Display

4. Problems Encountered

Once we were able to get each piece of the project working separately hardware and software we then put all together. That is when we faced our problems, the software would behave one way when ran on Linux and crash when ran on the Microblaze.

Also once we were able to get the software running correctly on the microblaze we encountered problems sending the image files through the serial port. The UARTlite controller was reading the wrong data at certain points and causing the decoder to crash.

5. Lessons Learned

We learned a number of things while working our way through this project. While many of us have debugged programs during the course of their writing, debugging a complete and supposedly working program was something new. We learned how we could increasingly limit the scope that our debugging statements encompassed in order to narrow down the exact line that was causing the problem.

Due to the complications that we experienced with the serial port we were forced to come up with an alternative solution, thankfully Prof. Edwards had one in mind. The episode was not as bad as it could have been, but it taught us the need to be prepared for unexpected complications.

One last lesson learned was the incredibly utility of periodic deadlines. It took an imminent deadline requiring our project to be 75% complete to really get ourselves going. Without it we don't know what would have made us start working.

6. Division of Labor

With only three of us in the group we equally worked on the software and hardware components of the project. Everyone was able to chip in and cover for each other when needed.

7. Advice to Others

Most importantly, start early, you never know what kind of problems you are going to run into and the extra time can be a life saver.

A project of this scope is incredibly difficult to do on ones own. There are a number of resources available to the students and they would best be served by taking full advantage of them. These include the TAs and the professor, all of whom have a wealth of experience which can be incredibly helpful. Additionally, there are pieces of software and hardware, meticulously worked through by earlier students and TAs, which can serve some necessary functions with most of the bugs already worked out.

We made a big mistake in the course of our project. We achieved a working solution and then decided to change it. When our modifications broke the program we were unable to reverse the changes. Had we made a backup we wouldn't have had a problem, but we didn't so we were forced to start all over. The moral of this story: MAKE BACKUPS!

There are a few last points to be made. For one, the people you work with can make or break the project, choose your partners wisely. Don't be overly ambitious, the professor will likely help you in this respect, but it is good to keep in mind that a really cool project may be more than you can handle. Lastly, once you've gotten yourself working its important to work consistently, leaving the work till the end is a bad idea, it is hard enough when you are not swamped with finals and end of term projects.

8. Relevant Source Code

a. opb_xsb300.vhd

```
------------------------------------------------------------------------------
--
-- OPB bus bridge for the XESS XSB-300E board
--
-- Includes a memory controller, a VGA framebuffer, and glue for the SRAM
--
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity opb_xsb300 is
  generic (
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector := X"2000_0000";
    C_HIGHADDR      : std_logic_vector := X"2000_00FF");

  port (
    OPB_Clk : in std_logic;
    OPB_Rst : in std_logic;
    OPB_ABus : in std_logic_vector (31 downto 0);
    OPB_BE : in std_logic_vector (3 downto 0);
    OPB_DBus : in std_logic_vector (31 downto 0);
    OPB_RNW : in std_logic;
    OPB_select : in std_logic;
    OPB_seqAddr : in std_logic;
    pixel_clock : in std_logic;
    UIO_DBus : out std_logic_vector (31 downto 0);
    UIO_errAck : out std_logic;
    UIO_retry : out std_logic;
    UIO_toutSup : out std_logic;
    UIO_xferAck : out std_logic;
    PB_A : out std_logic_vector (19 downto 0);
    PB_UB_N : out std_logic;
    PB_LB_N : out std_logic;
    PB_WE_N : out std_logic;
    PB_OE_N : out std_logic;
    RAM_CE_N : out std_logic;
    VIDOUT_CLK : out std_logic;
    VIDOUT_RCR : out std_logic_vector (9 downto 0);
    VIDOUT_GY : out std_logic_vector (9 downto 0);
    VIDOUT_BCB : out std_logic_vector (9 downto 0);
    VIDOUT_BLANK_N : out std_logic;
    VIDOUT_HSYNC_N : out std_logic;
    VIDOUT_VSYNC_N : out std_logic;
    PB_D : inout std_logic_vector (15 downto 0));
end opb_xsb300;

architecture Behavioral of opb_xsb300 is

  constant C_MASK : integer := 0; -- huge address window as we are a bridge

  signal addr_mux : std_logic_vector(19 downto 0);
  signal video_addr : std_logic_vector (19 downto 0);
```

```
signal video_data : std_logic_vector (15 downto 0);
signal video_req : std_logic;
signal video_ce : std_logic;
signal i : integer;
signal cs : std_logic;

signal onecycle : std_logic ;
signal videocycle, amuxsel, hihalf : std_logic;
signal rce0, rce1, rreset : std_logic;
signal xfer : std_logic;
signal pb_wr, pb_rd : std_logic;

signal sram_ce : std_logic;

signal rnw : std_logic;

signal addr : std_logic_vector (23 downto 0);

signal be : std_logic_vector (3 downto 0);
signal pb_bytesel : std_logic_vector (1 downto 0);

signal wdata : std_logic_vector (31 downto 0);
signal wdata_mux : std_logic_vector (15 downto 0);

signal rdata : std_logic_vector (15 downto 0); -- register data read - FDRE

component vga
  port (
    clk : in std_logic;
    pix_clk : in std_logic;
    rst : in std_logic;
    video_data : in std_logic_vector(15 downto 0);
    video_addr : out std_logic_vector(19 downto 0);
    video_req : out std_logic;
    vidout_clk : out std_logic;
    vidout_RCR : out std_logic_vector(9 downto 0);
    vidout_GY : out std_logic_vector(9 downto 0);
    vidout_BCB : out std_logic_vector(9 downto 0);
    vidout_BLANK_N : out std_logic;
    vidout_HSYNC_N : out std_logic;
    vidout_VSYNC_N : out std_logic);
end component;

component memoryctrl
  port (
    rst : in std_logic;
    clk : in std_logic;
    cs : in std_logic;
    select0 : in std_logic;
    rnw : in std_logic;
    vreq : in std_logic;
    onecycle : in std_logic;
    videocycle : out std_logic;
    hihalf : out std_logic;
    pb_wr : out std_logic;
    pb_rd : out std_logic;
    xfer : out std_logic;
    ce0 : out std_logic;
    ce1 : out std_logic;
    rres : out std_logic;
    video_ce : out std_logic);
end component;
```

```
  component pad_io
    port (
      clk : in std_logic;
      rst : in std_logic;
      PB_A : out std_logic_vector(19 downto 0);
      PB_UB_N : out std_logic;
      PB_LB_N : out std_logic;
      PB_WE_N : out std_logic;
      PB_OE_N : out std_logic;
      RAM_CE_N : out std_logic;
      PB_D : inout std_logic_vector(15 downto 0);
      pb_addr : in std_logic_vector(19 downto 0);
      pb_ub : in std_logic;
      pb_lb : in std_logic;
      pb_wr : in std_logic;
      pb_rd : in std_logic;
      ram_ce : in std_logic;
      pb_dread : out std_logic_vector(15 downto 0);
      pb_dwrite : in std_logic_vector(15 downto 0));
  end component;

begin

  -- Framebuffer

  vga1 : vga
    port map (
      clk => OPB_Clk,
      pix_clk => pixel_clock,
      rst => OPB_Rst,
      video_addr => video_addr,
      video_data => video_data,
      video_req => video_req,
      VIDOUT_CLK => VIDOUT_CLK,
      VIDOUT_RCR => VIDOUT_RCR,
      VIDOUT_GY => VIDOUT_GY,
      VIDOUT_BCB => VIDOUT_BCB,
      VIDOUT_BLANK_N => VIDOUT_BLANK_N,
      VIDOUT_HSYNC_N => VIDOUT_HSYNC_N,
      VIDOUT_VSYNC_N => VIDOUT_VSYNC_N);

-- Memory control/arbitration state machine

  memoryctrl1 : memoryctrl port map (
    rst => OPB_Rst,
    clk => OPB_Clk,
    cs => cs,
    select0 => OPB_select,
    rnw => rnw,
    vreq => video_req,
    onecycle => onecycle,
    videocycle => videocycle,
    hihalf => hihalf,
    pb_wr => pb_wr,
    pb_rd => pb_rd,
    xfer => xfer,
    ce0 => rce0,
    ce1 => rce1,
    rres => rreset,
    video_ce => video_ce);

-- I/O pads
```

```
pad_io1 : pad_io port map (
  clk => OPB_Clk,
  rst => OPB_Rst,
  PB_A => PB_A,
  PB_UB_N => PB_UB_N,
  PB_LB_N => PB_LB_N,
  PB_WE_N => PB_WE_N,
  PB_OE_N => PB_OE_N,
  RAM_CE_N => RAM_CE_N,
  PB_D => PB_D,
  pb_addr => addr_mux,
  pb_rd => pb_rd,
  pb_wr => pb_wr,
  pb_ub => pb_bytesel(1),
  pb_lb => pb_bytesel(0),
  ram_ce => sram_ce,
  pb_dread  => rdata,
  pb_dwrite => wdata_mux);

sram_ce <= pb_rd or pb_wr;

amuxsel <= videocycle;

addr_mux <= video_addr when (amuxsel = '1')
            else (addr(20 downto 2) & (addr(1) or hihalf));

onecycle <= (not be(3)) or (not be(2)) or (not be(1)) or (not be(0));

wdata_mux <= wdata(15 downto 0) when ((addr(1) or hihalf) = '1')
             else wdata(31 downto 16);

process(videocycle, be, addr(1), hihalf, pb_rd, pb_wr)
begin
  if videocycle = '1' then
    pb_bytesel <= "11";
  elsif pb_rd='1' or pb_wr='1' then
    if addr(1)='1' or hihalf='1' then
      pb_bytesel <= be(1 downto 0);
    else
      pb_bytesel <= be(3 downto 2);
    end if;
  else
    pb_bytesel <= "00";
  end if;
end process;

cs <= OPB_select when OPB_ABus(31 downto 20) = X"008" else '0';

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      rnw <= '0';
    else
      rnw <= OPB_RNW;
    end if;
  end if;
end process;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_RST = '1' then
```

```
      addr <= X"000000";
    else
      addr <= OPB_ABus(23 downto 0);
    end if;
  end if;
end process;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      be <= "0000";
    else
      be <= OPB_BE;
    end if;
  end if;
end process;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      wdata <= X"00000000";
    else
      wdata <= OPB_DBus;
    end if;
  end if;
end process;

process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if video_ce = '1' then
      video_data <= rdata;
    end if;
  end if;
end process;

-- Write the low two bytes if rce0 or rce1 is enabled

process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    UIO_DBus(15 downto 0) <= X"0000";
  elsif OPB_Clk'event and OPB_Clk = '1' then
    if rreset = '1' then
      UIO_DBus(15 downto 0) <= X"0000";
    elsif (rce1 or rce0) = '1' then
      UIO_DBus(15 downto 0) <= rdata(15 downto 0);
    end if;
  end if;
end process;

-- Write the high two bytes if rce0 is enabled

process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    UIO_DBus(31 downto 16) <= X"0000";
  elsif OPB_Clk'event and OPB_Clk = '1' then
    if rreset = '1' then
      UIO_DBus(31 downto 16) <= X"0000";
    elsif rce0 = '1' then
```

```
        UIO_DBus(31 downto 16) <= rdata(15 downto 0);
      end if;
    end if;
  end process;

  -- unused outputs

  UIO_errAck <= '0';
  UIO_retry <= '0';
  UIO_toutSup <= '0';

  UIO_xferAck <= xfer;

end Behavioral;
```

b. vga.vhd

```
----------------------------------------------------------------------------------
--
-- VGA video generator
--
-- Uses the vga_timing module to generate hsync etc.
-- Massages the RAM address and requests cycles from the memory controller
-- to generate video using one byte per pixel
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity vga is
  port (
    clk            : in std_logic;
    pix_clk        : in std_logic;
    rst            : in std_logic;
    video_data     : in std_logic_vector(15 downto 0);
    video_addr     : out std_logic_vector(19 downto 0);
    video_req      : out std_logic;
    VIDOUT_CLK     : out std_logic;
    VIDOUT_RCR     : out std_logic_vector(9 downto 0);
    VIDOUT_GY      : out std_logic_vector(9 downto 0);
    VIDOUT_BCB     : out std_logic_vector(9 downto 0);
    VIDOUT_BLANK_N : out std_logic;
    VIDOUT_HSYNC_N : out std_logic;
    VIDOUT_VSYNC_N : out std_logic);
end vga;

architecture Behavioral of vga is

  -- Fast low-voltage TTL-level I/O pad with 12 mA drive

  component OBUF_F_12
    port (
      O : out STD_ULOGIC;
      I : in STD_ULOGIC);
  end component;

  -- Basic edge-sensitive flip-flop
```

```
  component FD
    port (
      C : in std_logic;
      D : in std_logic;
      Q : out std_logic);
  end component;

  -- Force instances of FD into pads for speed

  attribute iob : string;
  attribute iob of FD : component is "true";

  component vga_timing
    port (
      h_sync_delay         : out std_logic;
      v_sync_delay         : out std_logic;
      blank                : out std_logic;
      vga_ram_read_address : out std_logic_vector (19 downto 0);
      pixel_clock          : in std_logic;
      reset                : in std_logic);
  end component;

  signal r                      : std_logic_vector (9 downto 0);
  signal g                      : std_logic_vector (9 downto 0);
  signal b                      : std_logic_vector (9 downto 0);
  signal blank                  : std_logic;
  signal hsync                  : std_logic;
  signal vsync                  : std_logic;
  signal vga_ram_read_address   : std_logic_vector(19 downto 0);
  signal vreq                   : std_logic;
  signal vreq_1                 : std_logic;
  signal load_video_word        : std_logic;
  signal vga_shreg              : std_logic_vector(15 downto 0);

begin

  st : vga_timing port map (
    pixel_clock => pix_clk,
    reset => rst,
    h_sync_delay => hsync,
    v_sync_delay => vsync,
    blank => blank,
    vga_ram_read_address => vga_ram_read_address);

  -- Video request is true when the RAM address is even

  -- FIXME: This should be disabled during blanking to reduce memory traffic

  vreq <= not vga_ram_read_address(0);

  -- Generate load_video_word by delaying vreq two cycles

  process (pix_clk)
  begin
    if pix_clk'event and pix_clk='1' then
      vreq_1 <= vreq;
      load_video_word <= vreq_1;
    end if;
  end process;

  -- Generate video_req (to the RAM controller) by delaying vreq by
  -- a cycle synchronized with the pixel clock
```

```
process (clk)
begin
  if clk'event and clk='1' then
    video_req <= pix_clk and vreq;
  end if;
end process;

-- The video address is the upper 19 bits from the VGA timing generator
-- because we are using two pixels per word and the RAM address counts words

video_addr <= '0' & vga_ram_read_address(19 downto 1);

-- The video shift register: either load it from RAM or shift it up a byte

process (pix_clk)
begin
  if pix_clk'event and pix_clk='1' then
    if load_video_word = '1' then
      vga_shreg <= video_data;
    else
      -- Shift the low byte of read video data into the high byte
      vga_shreg <= vga_shreg(7 downto 0) & "00000000";
    end if;
  end if;
end process;

-- Copy the upper byte of the video word to the color signals
-- Note that we use three bits for red and green and two for blue.

r(9 downto 7) <= vga_shreg (15 downto 13);
r(6 downto 0) <= "0000000";
g(9 downto 7) <= vga_shreg (12 downto 10);
g(6 downto 0) <= "0000000";
b(9 downto 8) <= vga_shreg (9 downto 8);
b(7 downto 0) <= "00000000";

-- Video clock I/O pad to the DAC

vidclk : OBUF_F_12 port map (
  O => VIDOUT_clk,
  I => pix_clk);

-- Control signals: hsync, vsync, and blank

hsync_ff : FD port map (
  C => pix_clk,
  D => not hsync,
  Q => VIDOUT_HSYNC_N );

vsync_ff : FD port map (
  C => pix_clk,
  D => not vsync,
  Q => VIDOUT_VSYNC_N );

blank_ff :  FD port map (
  C => pix_clk,
  D => not blank,
  Q => VIDOUT_BLANK_N );

-- Three digital color signals

rgb_ff : for i in 0 to 9 generate
```

```
    r_ff : FD port map (
      C => pix_clk,
      D => r(i),
      Q => VIDOUT_RCR(i) );

    g_ff : FD port map (
      C => pix_clk,
      D => g(i),
      Q => VIDOUT_GY(i) );

    b_ff : FD port map (
      C => pix_clk,
      D => b(i),
      Q => VIDOUT_BCB(i) );

  end generate;

end Behavioral;
```

c. vga_timing.vhd

```
--------------------------------------------------------------------------------
--
-- VGA timing and address generator
--
-- Fixed-resolution address generator.  Generates h-sync, v-sync, and blanking
-- signals along with a 20-bit RAM address.  H-sync and v-sync signals are
-- delayed two cycles to compensate for the DAC pipeline.
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga_timing is
  port (
    pixel_clock  : in std_logic;
    reset        : in std_logic;
    h_sync_delay : out std_logic;
    v_sync_delay : out std_logic;
    blank        : out std_logic;
    vga_ram_read_address : out std_logic_vector(19 downto 0));
end vga_timing;

architecture Behavioral of vga_timing is

  constant SRAM_DELAY : integer := 3;

-- 640 X 480 @ 60Hz with a 25.175 MHz pixel clock
  constant H_ACTIVE      : integer := 640;
  constant H_FRONT_PORCH : integer := 16;
  constant H_BACK_PORCH  : integer := 48;
  constant H_TOTAL       : integer := 800;

  constant V_ACTIVE      : integer := 480;
  constant V_FRONT_PORCH : integer := 11;
  constant V_BACK_PORCH  : integer := 31;
  constant V_TOTAL       : integer := 524;
```

```
  signal line_count  : std_logic_vector (9 downto 0);   -- Y coordinate
  signal pixel_count : std_logic_vector (10 downto 0);  -- X coordinate

  signal h_sync : std_logic;     -- horizontal sync
  signal v_sync : std_logic;     -- vertical sync

  signal h_sync_delay0 : std_logic; -- h_sync delayed 1 clock
  signal v_sync_delay0 : std_logic; -- v_sync delayed 1 clock

  signal h_blank : std_logic;           -- horizontal blanking
  signal v_blank : std_logic;           -- vertical blanking

  -- flag to reset the ram address during vertical blanking
  signal reset_vga_ram_read_address : std_logic;

  -- flag to hold the address during horizontal blanking
  signal hold_vga_ram_read_address : std_logic;

  signal ram_address_counter : std_logic_vector (19 downto 0);

begin

  -- Pixel counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      pixel_count <= "00000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_TOTAL - 1) then
        pixel_count <= "00000000000";
      else
        pixel_count <= pixel_count + 1;
      end if;
    end if;
  end process;

  -- Horizontal sync

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      h_sync <= '0';
    elsif pixel_clock'event and pixel_clock = '1' then
      if pixel_count = (H_ACTIVE + H_FRONT_PORCH - 1) then
        h_sync <= '1';
      elsif pixel_count = (H_TOTAL - H_BACK_PORCH - 1) then
        h_sync <= '0';
      end if;
    end if;
  end process;

  -- Line counter

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      line_count <= "0000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if ((line_count = V_TOTAL - 1) and (pixel_count = H_TOTAL - 1)) then
        line_count <= "0000000000";
      elsif pixel_count = (H_TOTAL - 1) then
```

```
      line_count <= line_count + 1;
    end if;
  end if;
end process;

-- Vertical sync

process ( pixel_clock, reset )
begin
  if reset = '1' then
    v_sync <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if line_count = (V_ACTIVE + V_FRONT_PORCH -1) and
       pixel_count = (H_TOTAL - 1) then
      v_sync <= '1';
    elsif line_count = (V_TOTAL - V_BACK_PORCH - 1) and
          pixel_count = (H_TOTAL - 1) then
      v_sync <= '0';
    end if;
  end if;
end process;

-- Add two-cycle delays to h/v_sync to compensate for the DAC pipeline

process ( pixel_clock, reset )
begin
  if reset = '1' then
    h_sync_delay0 <= '0';
    v_sync_delay0 <= '0';
    h_sync_delay  <= '0';
    v_sync_delay  <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    h_sync_delay0 <= h_sync;
    v_sync_delay0 <= v_sync;
    h_sync_delay  <= h_sync_delay0;
    v_sync_delay  <= v_sync_delay0;
  end if;
end process;

-- Horizontal blanking

-- The constants are offset by two to compensate for the delay
-- in the composite blanking signal

process ( pixel_clock, reset )
begin
  if reset = '1' then
    h_blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if pixel_count = (H_ACTIVE - 2) then
      h_blank <= '1';
    elsif pixel_count = (H_TOTAL - 2) then
      h_blank <= '0';
    end if;
  end if;
end process;

-- Vertical Blanking

-- The constants are offset by two to compensate for the delay
-- in the composite blanking signal

process ( pixel_clock, reset )
```

```
begin
  if reset = '1' then
    v_blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if line_count = (V_ACTIVE - 1) and pixel_count = (H_TOTAL - 2) then
      v_blank <= '1';
    elsif line_count = (V_TOTAL - 1) and pixel_count = (H_TOTAL - 2) then
      v_blank <= '0';
    end if;
  end if;
end process;

-- Composite blanking

process ( pixel_clock, reset )
begin
  if reset = '1' then
    blank <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if (h_blank or v_blank) = '1' then
      blank <= '1';
    else
      blank <= '0';
    end if;
  end if;
end process;

-- RAM address counter

-- Two control signals:

-- reset_ram_read_address is active from the end of each field until the
-- beginning of the next

-- hold_vga_ram_read_address is active from the end of each line to the
-- start of the next

process ( pixel_clock, reset )
begin
  if reset = '1' then
    reset_vga_ram_read_address <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if line_count = V_ACTIVE - 1 and
       pixel_count = ( (H_TOTAL - 1 ) - SRAM_DELAY ) then
      -- reset the address counter at the end of active video
      reset_vga_ram_read_address <= '1';
    elsif line_count = V_TOTAL - 1 and
          pixel_count =((H_TOTAL -1) - SRAM_DELAY) then
      -- re-enable the address counter at the start of active video
      reset_vga_ram_read_address <= '0';
    end if;
  end if;
end process;

process ( pixel_clock, reset )
begin
  if reset = '1' then
    hold_vga_ram_read_address <= '0';
  elsif pixel_clock'event and pixel_clock = '1' then
    if pixel_count = ((H_ACTIVE - 1) - SRAM_DELAY) then
      -- hold the address counter at the end of active video
      hold_vga_ram_read_address <= '1';
    elsif pixel_count = ((H_TOTAL - 1) - SRAM_DELAY) then
```

```
        -- re-enable the address counter at the start of active video
        hold_vga_ram_read_address <= '0';
      end if;
    end if;
  end process;

  process ( pixel_clock, reset )
  begin
    if reset = '1' then
      ram_address_counter <= "00000000000000000000";
    elsif pixel_clock'event and pixel_clock = '1' then
      if reset_vga_ram_read_address = '1' then
        ram_address_counter <= "00000000000000000000";
      elsif hold_vga_ram_read_address = '0' then
        ram_address_counter <= ram_address_counter + 1;
      end if;
    end if;
  end process;

  vga_ram_read_address <= ram_address_counter;

end Behavioral;
```

d. pad_io.vhd

```
--------------------------------------------------------------------------------
--
-- I/O Pads and associated circuity for the XSB-300E board
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-- Pad drivers, a little glue logic, and flip-flops for most bus signals
--
-- All signals, both control and the data and address busses, are registered.
-- FDC and FDP flip-flops are forced into the pads to do this.
--
-- Only the data bus is mildly complex: it latches data in both directions
-- as well as delaying the tristate control signals a cycle.
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity pad_io is
  port (
    clk : in std_logic;
    rst : in std_logic;
    PB_A : out std_logic_vector(19 downto 0);
    PB_UB_N : out std_logic;
    PB_LB_N : out std_logic;
    PB_WE_N : out std_logic;
    PB_OE_N : out std_logic;
    RAM_CE_N : out std_logic;
    PB_D : inout std_logic_vector(15 downto 0);
    pb_addr : in std_logic_vector(19 downto 0);
    pb_ub : in std_logic;
    pb_lb : in std_logic;
    pb_wr : in std_logic;
```

```
    pb_rd : in std_logic;
    ram_ce : in std_logic;
    pb_dread : out std_logic_vector(15 downto 0);
    pb_dwrite : in std_logic_vector(15 downto 0));
end pad_io;

architecture Behavioral of pad_io is

  -- Flip-flop with asynchronous clear

  component FDC
    port (
      C : in std_logic;
      CLR : in std_logic;
      D : in std_logic;
      Q : out std_logic);
  end component;

  -- Flip-flop with asynchronous preset

  component FDP
    port (
      C : in std_logic;
      PRE : in std_logic;
      D : in std_logic;
      Q : out std_logic);
  end component;

  -- Setting the iob attribute to "true" ensures that instances of these
  -- components are placed inside the I/O pads and are therefore very fast

  attribute iob : string;
  attribute iob of FDC : component is "true";
  attribute iob of FDP : component is "true";

  -- Fast off-chip output buffer, low-voltage TTL levels, 24 mA drive
  -- I is the on-chip signal, O is the pad

  component OBUF_F_24
    port (
      O : out STD_ULOGIC;
      I : in STD_ULOGIC);
  end component;

  -- Fast off-chip input/output buffer, low-voltage TTL levels, 24 mA drive
  -- T is the tristate control input, IO is the pad,

  component IOBUF_F_24
    port (
      O : out STD_ULOGIC;
      IO : inout STD_ULOGIC;
      I : in STD_ULOGIC;
      T : in STD_ULOGIC);
  end component;

  signal pb_addr_1: std_logic_vector(19 downto 0);
  signal pb_dwrite_1: std_logic_vector(15 downto 0);
  signal pb_tristate: std_logic_vector(15 downto 0);
  signal pb_dread_a: std_logic_vector(15 downto 0);
  signal we_n, pb_we_n1: std_logic;
  signal oe_n, pb_oe_n1: std_logic;
  signal lb_n, pb_lb_n1: std_logic;
  signal ub_n, pb_ub_n1: std_logic;
```

```
  signal ramce_n, ram_ce_n1: std_logic;
  signal dataz : std_logic;

begin

  -- Write enable

  we_n <= not pb_wr;

  we_ff : FDP port map (
    C => clk, PRE => rst,
    D => we_n,
    Q => pb_we_n1);

  we_pad : OBUF_F_24 port map (
    O => PB_WE_N,
    I => pb_we_n1);

  -- Output Enable

  oe_n <= not pb_rd;

  oe_ff : FDP port map (
    C => clk,
    PRE => rst,
    D => oe_n,
    Q => pb_oe_n1);

  oe_pad : OBUF_F_24 port map (
    O => PB_OE_N,
    I => pb_oe_n1);

  -- RAM Chip Enable

  ramce_n <= not ram_ce;

  ramce_ff : FDP port map (
    C => clk, PRE => rst,
    D => ramce_n,
    Q => ram_ce_n1);

  ramce_pad : OBUF_F_24 port map (
    O => RAM_CE_N,
    I => ram_ce_n1);

  -- Upper byte enable

  ub_n <= not pb_ub;

  ub_ff : FDP port map (
    C => clk, PRE => rst,
    D => ub_n,
    Q => pb_ub_n1);

  ub_pad : OBUF_F_24 port map (
    O => PB_UB_N,
    I => pb_ub_n1);

  -- Lower byte enable

  lb_n <= not pb_lb;

  lb_ff : FDP port map (
```

```
    C => clk,
    PRE => rst,
    D => lb_n,
    Q => pb_lb_n1);

  lb_pad : OBUF_F_24 port map (
    O => PB_LB_N,
    I => pb_lb_n1);

  -- 20-bit address bus

  addressbus : for i in 0 to 19 generate
    address_ff : FDC port map (
      C => clk, CLR => rst,
      D => pb_addr(i),
      Q => pb_addr_1(i));

    address_pad : OBUF_F_24 port map (
      O => PB_A(i),
      I => pb_addr_1(i));
  end generate;

  -- 16-bit data bus

  dataz <= (not pb_wr) or pb_rd;

  databus : for i in 0 to 15 generate
    dtff : FDP port map (               -- Trisate enable
      C => clk,
      PRE => rst,
      D => dataz,
      Q => pb_tristate(i));

    drff : FDP port map (
      C => clk,
      PRE => rst,
      D => pb_dread_a(i),
      Q => pb_dread(i));

    dwff : FDP port map (
      C => clk,
      PRE => rst,
      D => pb_dwrite(i),
      Q => pb_dwrite_1(i));

    data_pad : IOBUF_F_24 port map (
      O => pb_dread_a(i),
      IO => PB_D(i),
      I => pb_dwrite_1(i),
      T => pb_tristate(i));
  end generate;

end Behavioral;
```

e. memoryctrl.vhd

```
----------------------------------------------------------------------------------
--
-- Memory controller state machine
--
-- Arbitrates between requests from the processor and video system
-- The video system gets priority
```

```
-- Also handles 32- to 16-bit datapath width conversion (sequencing)
--
-- Cristian Soviani, Dennis Lim, and Stephen A. Edwards
--
-------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity memoryctrl is
  port (
    rst : in std_logic;
    clk : in std_logic;
    cs : in std_logic;                  -- chip select (address valid)
    select0 : in std_logic;             -- select (true through whole cycle)
    rnw : in std_logic;                 -- read/write'
    vreq : in std_logic;                -- video request
    onecycle : in std_logic;            -- one- or two-byte access (not four)

    videocycle : out std_logic;         -- acknowledges vreq
    hihalf : out std_logic;             -- doing bytes 2,3 of 32-bit access
    pb_wr : out std_logic;              -- Write request to off-chip memory
    pb_rd : out std_logic;              -- Read request to off-chip memory
    xfer : out std_logic;               -- Transfer acknowledge for OPB

    ce0 : out std_logic;                -- lower 16 bit OPB data latch enable
    ce1 : out std_logic;                -- upper 16 bit OPB data latch enable
    rres : out std_logic;               -- clear OPB data output latches

    video_ce : out std_logic);          -- video buffer latch enable
end memoryctrl;

architecture Behavioral of memoryctrl is
  -- State machine bits
  -- largely one-hot, but one ra and one rb bit can be true simultaneously
  signal r_idle : std_logic;
  signal r_common : std_logic;
  signal r_ra1 : std_logic;
  signal r_ra2 : std_logic;
  signal r_rb1 : std_logic;
  signal r_rb2 : std_logic;
  signal r_rb3 : std_logic;
  signal r_w : std_logic;
  signal r_xfer : std_logic;

  signal vcycle_1 : std_logic;
  signal vcycle_2 : std_logic;

  signal r_idle_next_state : std_logic;
  signal r_common_next_state : std_logic;
  signal r_ra1_next_state : std_logic;
  signal r_ra2_next_state : std_logic;
  signal r_rb1_next_state : std_logic;
  signal r_rb2_next_state : std_logic;
  signal r_rb3_next_state : std_logic;
  signal r_w_next_state : std_logic;
  signal r_xfer_next_state : std_logic;

begin

  -- Sequential process for the state machine

  process (clk, rst)
```

```
begin
  if rst = '1' then
    r_idle <= '1';
    r_common <= '0';
    r_ra1 <= '0';
    r_ra2 <= '0';
    r_rb1 <= '0';
    r_rb2 <= '0';
    r_rb3 <= '0';
    r_w <= '0';
    r_xfer <= '0';
  elsif clk'event and clk='1' then
    r_idle <= r_idle_next_state;
    r_common <= r_common_next_state;
    r_ra1 <= r_ra1_next_state;
    r_ra2 <= r_ra2_next_state;
    r_rb1 <= r_rb1_next_state;
    r_rb2 <= r_rb2_next_state;
    r_rb3 <= r_rb3_next_state;
    r_w <= r_w_next_state;
    r_xfer <= r_xfer_next_state;
  end if;
end process;

-- Combinational next-state logic

r_idle_next_state <= (r_idle and (not cs)) or r_xfer or (not select0);
r_common_next_state <= select0 and
                       ((r_idle and cs) or (r_common and vreq));
r_ra1_next_state <= select0 and
                    r_common and (not vreq) and rnw;
r_ra2_next_state <= select0 and
                    r_ra1;
r_rb1_next_state <= select0 and
                    ((r_common and not onecycle and not vreq and rnw) or
                     (r_rb1 and vreq));
r_rb2_next_state <= select0 and
                    (r_rb1 and (not vreq));
r_rb3_next_state <= select0 and
                    r_rb2;
r_w_next_state <= select0 and
                  ((r_common and (not rnw) and (not vreq) and
                    (not onecycle)) or
                   (r_w and vreq));
r_xfer_next_state <= select0 and
                     ((r_common and onecycle and
                       (not rnw) and (not vreq)) or
                      (r_w and (not vreq)) or (r_ra2 and onecycle) or r_rb3);

-- Combinational output logic

pb_wr <= (r_common and (not rnw) and (not vreq)) or (r_w and (not vreq));
pb_rd <= vreq or (r_common and (not vreq) and rnw) or (r_rb1 and (not vreq));

hihalf <= (r_w and (not vreq)) or (r_rb1 and (not vreq));

ce0 <= r_ra2;
ce1 <= r_rb3;
rres <= r_xfer;
xfer <= r_xfer;

-- Two-cycle delay of video request
-- (implicitly assumes video cycles always succeed)
```

```
  process (clk, rst)
  begin
    if(rst = '1') then
      vcycle_1 <= '0';
      vcycle_2 <= '0';
    elsif clk'event and clk='1' then
      vcycle_1 <= vreq;
      vcycle_2 <= vcycle_1;
    end if;
  end process;

  videocycle <= vreq;
  video_ce <= vcycle_2;

end Behavioral;
```

f. system.mhs

```
# Parameters
PARAMETER VERSION = 2.0.0

# Global Ports

PORT PB_A = PB_A, DIR = OUT, VEC = [19:0]
PORT PB_D = PB_D, DIR = INOUT, VEC = [15:0]
PORT PB_LB_N = PB_LB_N, DIR = OUT
PORT PB_UB_N = PB_UB_N, DIR = OUT
PORT PB_WE_N = PB_WE_N, DIR = OUT
PORT PB_OE_N = PB_OE_N, DIR = OUT
PORT RAM_CE_N = RAM_CE_N, DIR = OUT
PORT VIDOUT_CLK = VIDOUT_CLK, DIR = OUT
PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N, DIR = OUT
PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N, DIR = OUT
PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N, DIR = OUT
PORT VIDOUT_RCR = VIDOUT_RCR, DIR = OUT, VEC = [9:0]
PORT VIDOUT_GY = VIDOUT_GY, DIR = OUT, VEC = [9:0]
PORT VIDOUT_BCB = VIDOUT_BCB, DIR = OUT, VEC = [9:0]
PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN
PORT AU_CSN_N =  AU_CSN_N, DIR=OUT
PORT AU_BCLK =  AU_BCLK, DIR=OUT
PORT AU_MCLK = AU_MCLK, DIR=OUT
PORT AU_LRCK = AU_LRCK, DIR=OUT
PORT AU_SDTI = AU_SDTI, DIR=OUT
PORT AU_SDTO0 = AU_SDTO0, DIR=IN

# Sub Components


BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_ADDR_TAG_BITS = 6
 PARAMETER C_CACHE_BYTE_SIZE = 2048
 PARAMETER C_ICACHE_BASEADDR = 0x00860000
 PARAMETER C_ICACHE_HIGHADDR = 0x0087FFFF
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
```

```
 PORT Interrupt = intr
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

BEGIN opb_intc
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFFFF0000
 PARAMETER C_HIGHADDR = 0xFFFF00FF
 PORT OPB_Clk = sys_clk
 PORT Intr =  uart_intr
 PORT Irq =   intr
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_xsb300
 PARAMETER INSTANCE = xsb300
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF
 PORT PB_A = PB_A
 PORT PB_D = PB_D
 PORT PB_LB_N = PB_LB_N
 PORT PB_UB_N = PB_UB_N
 PORT PB_WE_N = PB_WE_N
 PORT PB_OE_N = PB_OE_N
 PORT RAM_CE_N = RAM_CE_N
 PORT OPB_Clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT VIDOUT_CLK = VIDOUT_CLK
 PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N
 PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N
 PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N
 PORT VIDOUT_RCR = VIDOUT_RCR
 PORT VIDOUT_GY = VIDOUT_GY
 PORT VIDOUT_BCB = VIDOUT_BCB
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

BEGIN lmb_lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
```

```
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 PORT Interrupt = uart_intr
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END
```

g. mylinkerscript

```
/*
 * Instructions for the linker about where in memory to locate various
 * portions of the program.  See "info ld" for details.
 */

/* List of memory blocks */
MEMORY {
  BRAM : ORIGIN = 0x00000000, LENGTH = 0x01000 /* On the FPGA */
  SRAM : ORIGIN = 0x00860000, LENGTH = 0x40000 /* SRAM past the framebuffer */
}
```

```
/* Symbol where the program will start executing */
ENTRY(_start)

/* Instructions on where to locate each segment of the program */
SECTIONS
{

  /*
   * Critical code to place on the FPGA: initialization, interrupts, etc.
   */

  .bram_text : {

       /usr/cad/xilinx/gnu/lib/gcc-lib/microblaze/2.95.3-4/crt0.o(.text)
       /usr/cad/xilinx/gnu/lib/gcc-lib/microblaze/2.95.3-4/crtinit.o(.text)

       ./mymicroblaze/lib/libxil.a(.text)
       ./mymicroblaze/lib//libxil.a(.text)

  } > BRAM

  /*
   * The stack
   */

  . = ALIGN(4);
  .stack : {
     _STACK_SIZE = 0x600;
     . += _STACK_SIZE;
     . = ALIGN(4);
  } > BRAM
  _stack = .;  /* It starts here and grows downward */

  /*
   * Code to be placed in SRAM: the rest of the program
   */

  . = ALIGN(4);
  .sram_text : {
    *(.text)
  } > SRAM

  /*
   * Small initialized read-only memory
   */

  . = ALIGN(8);
  _ss_small = .;
  .sdata2 . : {
    *(.sdata2)
  } > SRAM

  /*
   * Small initialized memory
   */

  . = ALIGN(8);
  .sdata : {
    *(.sdata)
  } > SRAM

  . = ALIGN(8);
```

```
  _es_small = .;
  _ssize_small = _es_small - _ss_small;
  _SDA2_BASE_ = _ss_small + (_ssize_small / 2 );

  /*
   * Initialized read-only memory
   */

  . = ALIGN(4);
  .rodata . : {
    *(.rodata)
  } > SRAM

  /*
   * Initialized memory
   */

  . = ALIGN(4);
  .data : {
    *(.data)
  } > SRAM

  /* SBSS and BSS */

  . = ALIGN(8);
  __sbss_start = .;
  .sbss : {
     *(.sbss)
  } > SRAM
  . = ALIGN(8);
  __sbss_end = .;

  __sbss_size = __sbss_end - __sbss_start;
  _SDA_BASE_ = __sbss_start + (__sbss_size / 2);

  /*
   * The heap
   */

  . = ALIGN(4);
  .sram_bss : {
    __bss_start = .;
    *(.bss)
    *(COMMON)
    . = ALIGN(4);
    __bss_end = . ;
    _heap = .;
  } > SRAM

  _end = .;

}
```

h. Makefile

```
SYSTEM = system

MICROBLAZE_OBJS = \
       djpeg/color.o \
       djpeg/fast_int_idct.o \
       djpeg/huffman.o \
```

```
	djpeg/jpeg.o \
	djpeg/parse.o \
	djpeg/tree_vld.o \
	djpeg/utils.o \
	djpeg/columbia.o

LIBRARIES = mymicroblaze/lib/libxil.a

ELF_FILE = $(SYSTEM).elf

NETLIST = implementation/$(SYSTEM).ngc

# Bitstreams for the FPGA

FPGA_BITFILE = implementation/$(SYSTEM).bit
MERGED_BITFILE = implementation/download.bit

# Files to be downloaded to the SRAM

SRAM_BINFILE = implementation/sram.bin
SRAM_HEXFILE = implementation/sram.hex

MHSFILE = $(SYSTEM).mhs
MSSFILE = $(SYSTEM).mss

FPGA_ARCH = spartan2e
DEVICE = xc2s300epq208-6

LANGUAGE = vhdl
PLATGEN_OPTIONS = -p $(FPGA_ARCH) -lang $(LANGUAGE)
LIBGEN_OPTIONS = -p $(FPGA_ARCH) $(MICROBLAZE_LIBG_OPT)

# Paths for programs

XILINX = /usr/cad/xilinx/ise6.1i
ISEBINDIR = $(XILINX)/bin/lin
ISEENVCMDS = LD_LIBRARY_PATH=$(ISEBINDIR) XILINX=$(XILINX) PATH=$(ISEBINDIR)

XILINX_EDK = /usr/cad/xilinx/edk3.2

MICROBLAZE = /usr/cad/xilinx/gnu
MBBINDIR = $(MICROBLAZE)/bin
XESSBINDIR = /usr/cad/xess/bin

# Executables

XST = $(ISEENVCMDS) $(ISEBINDIR)/xst
XFLOW = $(ISEENVCMDS) $(ISEBINDIR)/xflow
BITGEN = $(ISEENVCMDS) $(ISEBINDIR)/bitgen
DATA2MEM = $(ISEENVCMDS) $(ISEBINDIR)/data2mem
XSLOAD = $(XESSBINDIR)/xsload
XESS_BOARD = XSB-300E

MICROBLAZE_CC = $(MBBINDIR)/microblaze-gcc
MICROBLAZE_CC_SIZE = $(MBBINDIR)/microblaze-size
MICROBLAZE_OBJCOPY = $(MBBINDIR)/microblaze-objcopy

# External Targets

all :
	@echo "Makefile to build a Microprocessor system :"
	@echo "Run make with any of the following targets"
	@echo "  make libs     : Configures the sw libraries for this system"
```

```
	@echo "  make program  : Compiles the program sources for all the
processor instances"
	@echo "  make netlist  : Generates the netlist for this system
($(SYSTEM))"
	@echo "  make bits     : Runs Implementation tools to generate the
bitstream"
	@echo "  make init_bram: Initializes bitstream with BRAM data"
	@echo "  make download : Downloads the bitstream onto the board"
	@echo "  make netlistclean: Deletes netlist"
	@echo "  make hwclean  : Deletes implementation dir"
	@echo "  make libsclean: Deletes sw libraries"
	@echo "  make programclean: Deletes compiled ELF files"
	@echo "  make clean    : Deletes all generated files/directories"
	@echo " "
	@echo "  make <target> : (Default)"
	@echo "      Creates a Microprocessor system using default
initializations"
	@echo "      specified for each processor in MSS file"


bits : $(FPGA_BITFILE)

netlist : $(NETLIST)

libs : $(LIBRARIES)

program : $(ELF_FILE)

init_bram : $(MERGED_BITFILE)

clean : hwclean libsclean programclean
	rm -f bram_init.sh
	rm -f _impact.cmd

hwclean : netlistclean
	rm -rf implementation synthesis xst hdl
	rm -rf xst.srp $(SYSTEM).srp

netlistclean :
	rm -f $(FPGA_BITFILE) $(MERGED_BITFILE) \
	  $(NETLIST) implementation/$(SYSTEM)_bd.bmm

libsclean :
	rm -rf mymicroblaze/lib

programclean :
	rm -f $(ELF_FILE) $(SRAM_BITFILE) $(SRAM_HEXFILE)

#
# Software rules
#

MICROBLAZE_MODE = executable

# Assemble software libraries from the .mss and .mhs files

$(LIBRARIES) : $(MHSFILE) $(MSSFILE)
	PATH=$$PATH:$(MBBINDIR) XILINX=$(XILINX) XILINX_EDK=$(XILINX_EDK) \
	 perl -I $(XILINX_EDK)/bin/nt/perl5lib $(XILINX_EDK)/bin/nt/libgen.pl \
	  $(LIBGEN_OPTIONS) $(MSSFILE)

# Compilation
```

```
MICROBLAZE_CC_CFLAGS =
MICROBLAZE_CC_OPT = -O3 #-mxl-gp-opt
MICROBLAZE_CC_DEBUG_FLAG =#  -gstabs
MICROBLAZE_INCLUDES = -I./mymicroblaze/include/ # -I
MICROBLAZE_CFLAGS = \
       $(MICROBLAZE_CC_CFLAGS)\
       -mxl-barrel-shift \
       $(MICROBLAZE_CC_OPT) \
       $(MICROBLAZE_CC_DEBUG_FLAG) \
       $(MICROBLAZE_INCLUDES)

$(MICROBLAZE_OBJS) : %.o : %.c
       PATH=$(MBBINDIR) $(MICROBLAZE_CC) $(MICROBLAZE_CFLAGS) -c $< -o $@

# Linking

# Uncomment the following to make linker print locations for everything
#MICROBLAZE_LD_FLAGS = -Wl,-M
MICROBLAZE_LINKER_SCRIPT = -Wl,-T -Wl,mylinkscript
MICROBLAZE_LIBPATH = -L./mymicroblaze/lib/
MICROBLAZE_CC_START_ADDR_FLAG= -Wl,-defsym -Wl,_TEXT_START_ADDR=0x00000000
MICROBLAZE_CC_STACK_SIZE_FLAG=  -Wl,-defsym -Wl,_STACK_SIZE=0x1000
MICROBLAZE_LFLAGS = \
        -xl-mode-$(MICROBLAZE_MODE) \
       $(MICROBLAZE_LD_FLAGS) \
       $(MICROBLAZE_LINKER_SCRIPT) \
       $(MICROBLAZE_LIBPATH) \
       $(MICROBLAZE_CC_START_ADDR_FLAG) \
       $(MICROBLAZE_CC_STACK_SIZE_FLAG)

$(ELF_FILE) :  $(LIBRARIES) $(MICROBLAZE_OBJS)
       PATH=$(MBBINDIR) XILINX_EDK="" $(MICROBLAZE_CC) $(MICROBLAZE_LFLAGS) \
              $(MICROBLAZE_OBJS) -lxil -o $(ELF_FILE)
       $(MICROBLAZE_CC_SIZE) -A $(ELF_FILE)
       mb-strip -s $(ELF_FILE)
       $(MICROBLAZE_CC_SIZE) -A $(ELF_FILE)


#
# Hardware rules
#

# Hardware compilation : optimize the netlist, place and route

$(FPGA_BITFILE) : $(NETLIST) \
              etc/fast_runtime.opt etc/bitgen.ut data/$(SYSTEM).ucf
       cp -f etc/bitgen.ut implementation/
       cp -f etc/fast_runtime.opt implementation/
       cp -f data/$(SYSTEM).ucf implementation/$(SYSTEM).ucf
       $(XFLOW) -wd implementation -p $(DEVICE) -implement fast_runtime.opt \
              $(SYSTEM).ngc
       cd implementation; $(BITGEN) -f bitgen.ut $(SYSTEM)

# Hardware assembly: Create the netlist from the .mhs file

$(NETLIST) : $(MHSFILE)
       XILINX=$(XILINX) XILINX_EDK=$(XILINX_EDK) \
       perl -I $(XILINX_EDK)/bin/nt/perl5lib $(XILINX_EDK)/bin/nt/platgen.pl \
         $(PLATGEN_OPTIONS) -st xst $(MHSFILE)
       perl synth_modules.pl < synthesis/xst.scr > xst.scr
       $(XST) -ifn xst.scr
       rm -r xst xst.scr
       $(XST) -ifn synthesis/$(SYSTEM).scr
```

```
#
# Downloading
#

# Add software code to the FPGA bitfile

$(MERGED_BITFILE) : $(FPGA_BITFILE) $(ELF_FILE)
	$(DATA2MEM) -bm implementation/$(SYSTEM)_bd \
	  -bt implementation/$(SYSTEM) \
	  -bd $(ELF_FILE) tag bram -o b $(MERGED_BITFILE)

# Create a .hex file with data for the SRAM

$(SRAM_HEXFILE) : $(ELF_FILE)
	$(MICROBLAZE_OBJCOPY) \
	       -j .sram_text -j .sdata2 -j .sdata -j .rodata -j .data \
	     -O binary $(ELF_FILE) $(SRAM_BINFILE)
	./bin2hex  -a 60000 < $(SRAM_BINFILE) > $(SRAM_HEXFILE)

# Download the files to the target board

download : $(MERGED_BITFILE) $(SRAM_HEXFILE)
	$(XSLOAD) -ram -b $(XESS_BOARD) $(SRAM_HEXFILE)
	$(XSLOAD) -fpga -b $(XESS_BOARD) $(MERGED_BITFILE)
```

i. jpeg.c

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

#ifndef LINUX

#include "xbasic_types.h"
#include "xio.h"


#endif


/* 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; /* MCU after color conversion */

FBlock       *FBuff;    /* scratch frequency buffer */
PBlock       *PBuff;    /* 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;

unsigned char ColorBuffer_stat [ MCU_SX * MCU_SY * N_COMP];
unsigned char FBuff_stat [sizeof(FBlock)];
unsigned char PBuff_stat [sizeof(PBlock)];
PBlock QTable_stat[4];
PBlock MCU_buff_stat[10];

unsigned char filein[INFILE_SIZE], *fip, *fieof;

extern unsigned char myimage[];
extern int myimage_size;

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

#ifdef LINUX

void print(char *s)
{
  printf("%s",s);
}

void putnum(unsigned x)
{
  printf("%08x", x);
}

#endif

#define MAGIC 1883

/*

void readfile()
{
  int i;
  fieof=fip=filein;

  for(i=0;i<MAGIC;i++)
    *fieof++ = getchar();
```

```
  print("File read\n");

  for(i=0;i<MAGIC;i++){
    putnum(MAGIC-1-i);print(":");putnum(filein[i]); print("\n\r");
  }

}

*/


int myfgetc(FILE *f)
{
  unsigned char result;
  if(fip == fieof) {
    print("EOF!");
    return EOF;
    }
  result = *fip++;
  /*  putnum(fieof-fip); print("=");putnum(result); print(":r\r\n");  */
  return result;
}

int
main(int argc, char* argv[])
{

  unsigned int aux, mark;
  int n_restarts, restart_interval, leftover; /* RST check */
  int i,j;
  int r,g,b;

  print("hello!\n\r");

  /*  readfile();  */


  fip =  myimage;
  fieof = myimage + myimage_size;


#ifndef LINUX

  for(i=0;i<640*480;i++)
    XIo_Out8(0x00800000 + i, 0);

#endif



  for(i=0;i<4;i++)
    QTable[i] = &QTable_stat[i];
  for(i=0;i<10;i++)
    MCU_buff[i] = &MCU_buff_stat[i];

/* dimension scan buffer for YUV->RGB conversion */

      ColorBuffer = ColorBuffer_stat;
      FBuff = (FBlock*) FBuff_stat;
      PBuff = (PBlock*) PBuff_stat;

      /*  print("hello 2!\n\r"); */
```

```
    fi = fip;
    /*  fo = stdout;  */

    /* print("hello 3!\n\r"); */

/* First find the SOI marker: */
aux = get_next_MK(fi);

if (aux != SOI_MK)
  aborted_stream(fi, fo);

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:
    print("SOF_MK\r\n");
    in_frame = 1;
    get_size(fi); /* header size, don't care */

    /* load basic image parameters */
    myfgetc(fi);  /* precision, 8bit, don't care */
    y_size = get_size(fi);
    x_size = get_size(fi);

    /*      fprintf(stderr,"x=%d y=%d\n", x_size, y_size); */
    /*  putnum(x_size); putnum(y_size); print("x=y=\n\r"); */

    /*
    if(x_size > X_SIZE || y_size > Y_SIZE){
     exit(-1);
    }
    */

    n_comp = myfgetc(fi);      /* # of components */

    for (i = 0; i < n_comp; i++) {

     comp[i].CID = myfgetc(fi);
     aux = myfgetc(fi);
     comp[i].HS = first_quad(aux);
     comp[i].VS = second_quad(aux);
     comp[i].QT = myfgetc(fi);
    }



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


    break;
```

```
  case DHT_MK:
print("DHT_MK\r\n");
    if (load_huff_tables(fi) == -1)
     aborted_stream(fi, fo);
    break;

  case DQT_MK:
    print("DQT_MK\r\n");
    if (load_quant_tables(fi) == -1)
     aborted_stream(fi, fo);
    break;

  case DRI_MK:
    get_size(fi); /* skip size */
    restart_interval = get_size(fi);

    break;

  case SOS_MK:           /* lots of things to do here */
      print("SOS_MK\r\n");
    get_size(fi); /* don't care */
    aux = myfgetc(fi);
    if (aux != (unsigned int) n_comp) {

     aborted_stream(fi, fo);
    }

    for (i = 0; i < n_comp; i++) {
     aux = myfgetc(fi);
     if (aux != comp[i].CID) {

       aborted_stream(fi, fo);
     }
     aux = myfgetc(fi);
     comp[i].DC_HT = first_quad(aux);
     comp[i].AC_HT = second_quad(aux);
    }
    get_size(fi); myfgetc(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);

       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:
  print("EOI_MK\r\n");
      if (in_frame)
       aborted_stream(fi, fo);


      /*      fclose(fi);*/
      /*      RGB_save(fo); */
      /*      fclose(fo); */

      /* fwrite(FrameBuffer+n_comp*i*x_size, n_comp, FrameHeader->Width, fo);
*/

      print("Done\n\r");
      exit(0);
      break;

    case COM_MK:
      skip_segment(fi);
      break;

    case EOF:
      aborted_stream(fi, fo);

    default:
      print("default\r\n");
      if ((mark & MK_MSK) == APP_MK) {

       skip_segment(fi);

       break;
      }

      /* if all else has failed ... */

      aborted_stream(fi, fo);
      break;
    } /* end switch */

  }
  while (1);

  return 0;
}
```

j. jpeg.h

```
/* 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 ? */

/*#define X_SIZE  100
#define Y_SIZE  100
*/
#define N_COMP  3
#define MCU_SX 16
#define MCU_SY 16

#define INFILE_SIZE 1
extern unsigned char filein[], *fip, *fieof;
extern int myfgetc(FILE *f);

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

/* 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;


/*-----------------------------------------*/
/* 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 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);
```

k. utils.c

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

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

#include "jpeg.h"



/* 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 = myfgetc(fi);
  }
}


/*-------------------------------------------*/
/* 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)
{

}


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

/* 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;
  /*  print("in reformat\n\r");*/
  if (!good)
    {

      return 0;
    }
  St = 1 << (good-1);      /* 2^(good-1) */
  if (S < (unsigned long) St)
    {
      /*      print("s<(unsigned long) st)\n\r");*/
      return (S+1+((-1) << good));
    }
  else
    {
      /*      print("else\n\r");*/
      return S;
    }
}
```

l. parse.c

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

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

#include "jpeg.h"

#ifndef LINUX

#include "xbasic_types.h"
#include "xio.h"


#endif


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


/* 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 = myfgetc(fi);

      if (wwindow == 0xFF)
       switch (aux = myfgetc(fi)) {     /* skip stuffer 0 byte */
       case EOF:
       case 0xFF:
         /*fprintf(stderr, "%ld:\tERROR:\tRan out of bit stream\n",
ftell(fi));*/

         aborted_stream(fi, fo);
         break;

       case 0x00:
         stuffers++;
         break;

       default:


         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 = myfgetc(fi);

    if (wwindow == 0xFF)
      switch (aux = myfgetc(fi)) {      /* skip stuffer 0 byte */
      case EOF:
      case 0xFF:

       aborted_stream(fi, fo);
       break;

      case 0x00:
       stuffers++;
       break;

      default:

       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 = myfgetc(fi);
  return (aux << 8) | myfgetc(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] = myfgetc(fi);
    tag[4] = '\0';

    size -= 4;
  }

}


/*-----------------------------------------------------------------*/
/* 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 = myfgetc(fi)) != (unsigned int) EOF) {

    switch (c) {
    case 0xFF:
      ffmet = 1;

      break;
    case 0x00:
      ffmet = 0;

      break;
    default:

      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 = myfgetc(fi);
    if (first_quad(aux) > 0) {

      return -1;
    }
    id = second_quad(aux);



    QTvalid[id] = 1;
    for (x = 0; x < 64; x++)
      QTable[id]->linear[x] = myfgetc(fi);

  }
  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;


      k++;
      if (k == 10) {

       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, pix, r, g, b;
  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;
    }

  }

  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      r = *(ColorBuffer + n_comp * i * MCU_sx + pix *3
); 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++) {

    for(pix=0; pix<goodcolumns; pix++){
      b = *(ColorBuffer + n_comp * i * MCU_sx + pix *3 );
      g = *(ColorBuffer + n_comp * i * MCU_sx + pix *3 + 1);
      r = *(ColorBuffer + n_comp * i * MCU_sx + pix *3 + 2);

      g = g >> 1;

      XIo_Out8( 0x00800000 +
             (MCU_row * MCU_sy + i)*640 +
             MCU_column * MCU_sx + pix,
             (r& 0xE0) | ((g&0xE0) >> 3) | ((b&0xC0) >> 6)
             );
    }
  }

  MCU_column++;
  return 1;
}
```

m. dumper.c

```
#include <stdio.h>


int main ()
{

  int c,i;


  i=-1;

  while ((c=getchar())!=EOF){
    i++;
    printf("%3d,", (unsigned char)c);
    if((i%16)==15) printf("\n");
  }

  return 0;
}
```

n. test232.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <fcntl.h>
#include <termios.h>
#include <string.h>
#include <errno.h>

#define DEBUG 0

int main()
{

  char rxbuf[1024], txbuf[1024];

  int fd, nbrx, nbtx, ptx, prx, nb;
  fd_set rfdsin, rfdsout;

  struct termios my_termios;

  fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY);


  // configure the serial port
  tcgetattr(fd, &my_termios);
  tcflush(fd, TCIFLUSH);
  my_termios.c_cflag = B19200 | CS8 |CREAD | CLOCAL | HUPCL;
  cfmakeraw(&my_termios);
  cfsetospeed(&my_termios, B19200);
  tcsetattr(fd, TCSANOW, &my_termios);

  nbtx=nbrx=0; ptx=prx=0;

  // the following code tries to be as "nice" as possible

  FD_SET(0, &rfdsin); // stdin
  FD_SET(fd, &rfdsin); // rs232 rx

  while(1){

    FD_ZERO(&rfdsin);
    FD_ZERO(&rfdsout);

    if(nbtx==0) FD_SET(0, &rfdsin); else FD_SET(fd, &rfdsout);
    if(nbrx==0) FD_SET(fd, &rfdsin); else FD_SET(1, &rfdsout);

    select(5, &rfdsin, &rfdsout, NULL, NULL);

    if(nbtx == 0 && FD_ISSET(0, &rfdsin)){ // tx buf empty and data on stdin
      ptx=0;
      nbtx=read(0,txbuf,1024);
      DEBUG && fprintf(stderr,"stdin: read %d\n",nbtx);
    }

    if(nbtx > 0 && FD_ISSET(fd, &rfdsout)){ // have tx data and rs232 free
      nb= write(fd,txbuf+ptx, nbtx);
      DEBUG && fprintf(stderr,"rs232: wrote %d\n",nb);
```

```
    ptx += nb;
    nbtx -= nb;
  }

  if(nbrx == 0 && FD_ISSET(fd, &rfdsin)){ // rx buf empty and data on rs232
    prx=0;
    nbrx=read(fd,rxbuf,1024);
    DEBUG && fprintf(stderr,"rs232: read %d\n",nbrx);
  }

  if(nbrx > 0 && FD_ISSET(1, &rfdsout)){ // have rx data and stdout free
    nb=write(1, rxbuf+prx, nbrx);
    DEBUG && fprintf(stderr,"stdout: wrote %d\n",nb);
    prx+= nb;
    nbrx -= nb;
  }
}

close(fd);
}
```