Pivoting Object Tracking System

[CSEE 4840 Project Final Report - May 2009]

Damian Ancukiewicz
Applied Physics and Applied Mathematics Department
da2260@columbia.edu

Arjun Roy
Computer Science Department
ar2409@columbia.edu

Jinglin Shen
Electrical Engineering Department
js3478@columbia.edu

Baolin Shao
Computer Science Department
bs2530@columbia.edu

Abstract

This project implements an object recognition system, where a camera tracks the position of an object. The camera is mounted on an iRobot Create two-wheeled robot, which rotates according to the control signal generated by our object tracking algorithm. Meanwhile, it displays 320×200 color video on a VGA display. We use a simple object recognition algorithm based on color information of the image coming from the camera. In our tests, the system is able track objects of single colors such as white, red, orange or blue if there is sufficient contrast between the object and background.

1. System Overview

In Figure 1, we give an overview of all hardware components in our system, which are all hooked up to the Avalon bus. Our system works as following: a video camera is connected to the Altera DE2 and sends NTSC analog signals to the board. An Analog Devices ADV7181 converts analog video signals from a camera to digital signals in YUV format. The converter has an I^2C interface, which allows for the output format and other parameters to be configured. The ADV7181 decoder controller takes digital video input from the ADV each pixel from YUV to 16-bit RGB. A double line buffer in the FPGA's block RAM is used used for data transfer between the 27 MHz frequency domain of the video controller and the 50 MHz frequency domain of the Avalon bus. Each line of the image data is sent to a buffer in the SDRAM using DMA (direct memory access) controller. The Nios II then performs processing on the buffer in SDRAM in order to find the center of the object we are tracking and to mark up the image. Additionally, the Nios II uses a serial interface to command the iRobot Create to turn in the appropriate direction if necessary. Subsequently, the buffer in SDRAM is sent to the VGA controller again by using DMA, which in turn transfers the buffer to SRAM. This buffer in SRAM is used to display the marked-up image on a VGA screen.

2. Sensing the Environment

2.1 ADV7181 Controller Design

In figure 2, we describe the design of the ADV7181 controller. The ADV7181 works on a 27 MHz clock and outputs data in the YUV format on an 8-bit parallel bus. The order used for transferring the information is YUYV, where each Y represents the luma (brightness) component of a pixel, while the U and V components represent chroma (color) and are shared between the two pixels. Using the recommended settings, each line of video takes up 1716 clock cycles, or 858 pixels, although the first 276 cycles consist of the horizontal blanking interval, in which no video information is sent. In the first two cycles, the `HS` signal is pulled low. A frame of video consists of 525 such lines, and is output in interlaced form. The first 20 lines consist of the vertical blanking interval, followed by 242 lines of active video which represent alternating lines of the frame. This is followed by another blanking interval of 21 lines, followed by 242 more lines of active video, representing the other set of alternating lines of the frame. The `VS` signal is pulled low for three lines in both vertical blanking intervals. Thus, the resolution that the ADV7181 outputs is $(1716-276)/2 \times 242 \times 2 = 720 \times 484$ pixels. This is later downscaled to 320×200 pixels due to the limitations imposed by the timing of the VGA framebuffer, as elaborated in the next section.

The ADV7181 is highly configurable and has an I^2C interface for that purpose. To configure the ADV7181 to output data in the correct format, an I^2C controller, written in Verilog, was imported from a lab assignment previously given to us. The controller, when started, sends a set of recommended settings to the ADV7181.

The ADV7181 interface module is clocked at the same speed as the ADV7181 itself. It reads the ADV7181's paral-

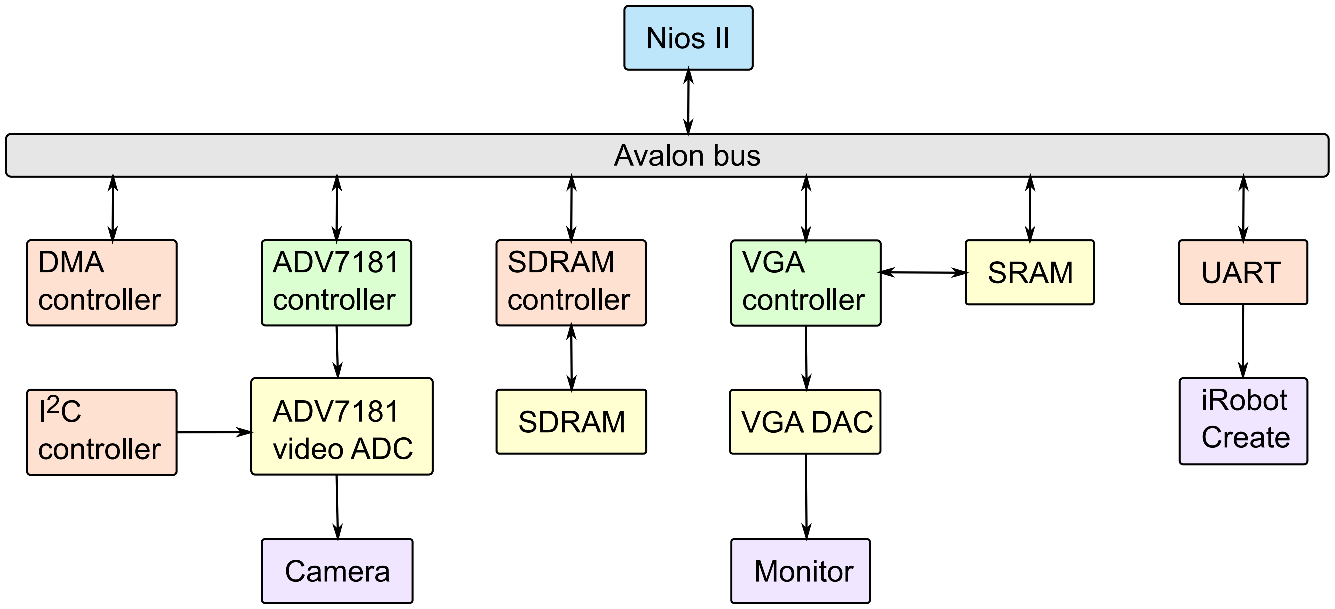

Figure 1. System design

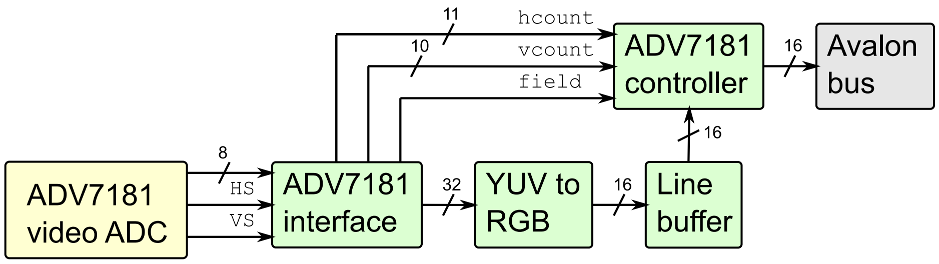

Figure 2. Design of the ADV7181 controller

lel bus on the rising edge of each clock cycle and increments its horizontal counter to keep track of the clock count. Every four cycles, 32 bits of information, representing two pixels, are output by the interface module. The module also reads the state of the `HS` and `VS` signals. When it detects that `HS` is pulled low, it resets the horizontal counter and increments the vertical counter. When it detects that `VS` is pulled low, it resets the vertical counter and changes the state of its field signal. Thus, at all times the decoder interface knows the horizontal position, line and field of the raster.

The YUV to RGB conversion module takes as input the 32 bits in YUYV format representing two pixels from the decoder interface and outputs a single 16-bit pixel in RGB format. The first 5 bits represent the green component, the middle 6 bits are the blue component, and the last 5 bits are the green component. Since two pixels are converted into one, this effectively halves the horizontal resolution. Because there are two Y components for one output pixel, only the first Y component is used. The conversion is done using the following formula:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 298 & 0 & 409 \\ 298 & -100 & -208 \\ 298 & 516 & 0 \end{bmatrix} \begin{bmatrix} Y-16 \\ U-128 \\ V-128 \end{bmatrix} + \begin{bmatrix} 128 \\ 128 \\ 128 \end{bmatrix}$$

The R, G and B values are then divided by 256, clipped to a range of between 0 and 255, and packed into the 16-bit 5:6:5 format. Because this algorithm can be done entirely using integer multiplication and bit shifting, it was easily implemented in VHDL.

The RGB data then needs to be transferred through the Avalon bus to a buffer on the SDRAM so that the Nios II can perform object recognition. This task is made tricky by two complications. First of all, the ADV7181 interface and Nios system operate in different clock domains: the former runs off of the same 27 MHz clock as the ADV7181, while the latter runs off of the 50 MHz system clock. Second of all, the SDRAM has latencies that are inherent to its design, and thus it is not guaranteed that a write will occur during a given system clock cycle. Because of these two problems, some form of buffering is needed. A FIFO buffer was first implemented for this task. A buffer was created in

`address[10..9]`	**Output**
00	Line buffer (memory-mapped)
01	Current field (0 or 1)
10	Horizontal clock count
11	Line count

Table 1. The output of the ADV7181 interface module with different address bit settings

the FPGA's block RAM, a write pointer would increment when the buffer was written to by the ADV7181 interface and a read pointer would increment when the buffer was read by the the Avalon bus respectively. Avalon flow control was implemented using the `dataavailable` signal, in order to stop the buffer from being read from when empty. However, because the ADV7181 and Avalon bus work in different clock domains, the design of the FIFO became very complex. Gray counters were used for the pointers so that consecutive addresses would only differ by one bit in order to minimize the effects of collision errors. Despite this, the FIFO buffer proved problematic, and pixels were lost from each line.

The second and current implementation is a double line buffer. Two buffers were created with enough space to hold a line of pixels. At any given time, the line buffer writes to one line while the other line is memory-mapped to the Avalon bus. This prevents clock collisions from occurring since the write and read processes are separated.

The ADV7181 interface presents a line of video data to the Avalon bus. The address is 11 bits wide, while the bus itself is 16 bits wide, and thus each word represents an RGB pixel. In order to allow the software to synchronize with the hardware, the horizontal clock count, line number and current field are also sent through the Avalon bus and accessed by changing the state of the upper two address bits. Table 1 describes the output produced by the module with different address bit settings.

2.2 VGA Framebuffer

Our vision for the POTS system would be to use it as a remote sentry that can be programmed to automatically track objects of a certain color to the best of its ability, and relay a camera feed to a human observer. To that end, a video output was required. Since the Altera DE2 supports VGA output, and since the VGA standard is fairly simple to implement, we decided to use it as our output.

The VGA standard accepts a stream of pixel data, synchronized on a horizontal blank period after every line and a vertical blank period after every complete frame. This gives us a choice of either streaming video data to the controller or using a framebuffer.

Since we have to perform image processing on a frame by frame basis, we designed our video output system to use a framebuffer. The Altera DE2 board allows us to choose between SDRAM and SRAM memory. Since the SDRAM is slower and more complicated than the SRAM, we based our buffer within the SRAM alone.

One issue with the choice of SRAM is that it is single ported; only one device can either read from or write to it at a time. Thus the VGA output section of the device would have exclusive active during the output of a frame, and the Nios II CPU would only be allowed to write to it during the vertical blank section.

In order to implement this behavior, we made the framebuffer an Avalon peripheral that used flow control. The peripheral asserts a `readyfordata` signal during the vertical blank portion between frames, ensuring the Nios II can only write data when the SRAM wasn't tied up outputting its contents to the VGA module. To prevent frame tearing, a typical solution is to use double buffering so that the integrity of each frame is assured. However, since we were performing transfers to the high performance SRAM, we assumed that the DMA transfer would push two bytes of data per clock cycle (the width of the SRAM data port and consequently, the width of each individual DMA transfer). Since we have a resolution of 320×200 pixels worth of image data, with each pixel requiring two bytes, we estimated needing 64000 clock cycles (50 MHz system clock) to transfer a frame.

VGA uses a 25 MHz clock. We calculated that the vertical blank interval lasts for 200,000 cycles of the 50 MHz system clock. There are 800 horizontal pixels per line, and 125 lines of video where there is no active video. $800 \times 125 = 100,000$ clocks at 25 MHz, or $200,000$ cycles at 50 MHz.

However, we found out during testing that the visible frame tearing did occur. The most apparent example is when filling the entire screen with the same color, and then switching immediately to another one. We hypothesize that even though a transfer should fit within a vertical blank interval, it is possible that a transfer could start close to the end of an interval and be only be partially complete when the next frame is sent to the VGA output, resulting in tearing. When the screen displays video from the camera, however, tearing is not very noticeable.

The implementation in VHDL for the framebuffer device is simple; to the processor it appears just as a buffer with flow control. During the active interval of the VGA signal, it tracks which column and row on the VGA output it is supposed to be displaying and queries the SRAM memory for the appropriate pixel data. The SDRAM stores each pixel as a 16 bit 5-6-5 RGB value The most significant 5 bits store the most significant bits of red, and the following 6 and 5 bits store the most significant 6 and 5 bits of green and blue. A simple transformation converts it back to 24 bits of RGB data destined for the VGA output.

3. Reacting to the Environment

3.1 DMA

To transfer data from the ADV7181 decoder to the SDRAM, and from the SDRAM to the VGA Framebuffer, we used a single DMA controller in order to bypass the Nios II.

In addition, the DMA works with Avalon flow control which enables us to write to the VGA framebuffer only when it is ready for data - when the VGA output is not polling it for pixel data. With flow control, the slave device (the ADV7181 decoder or VGA framebuffer) can assert a signal that it is ready for reading and/or writing. Thus, any data transfer can occur at the speed and timing that the slave requires. A slave peripheral can drive the readyfordata signal high to indicate that it is ready to begin a transfer, and it can drive the `dataavailable` signal high to indicate that it is ready to be written to. With this, we can let the decoder alert the DMA controller when a new video frame has begun so the SDRAM can be written to, and we can let the VGA controller alert the when the visible part of the VGA frame has ended so that the SDRAM can be read.

In addition to having Avalon master ports to facilitate the data transfer, the DMA controller also has an Avalon slave port which is used by the Nios II to initiate the transfer and set the memory locations to be transferred and the length of the transferred data.

While the DMA controller is easy to instantiate and configure, we ran into a strange video corruption problem. Specifically, whenever we modified the image buffer read into SDRAM, it did not show up in a predictable spot when displayed on screen. However, we were able to identify the reason of this as being a race condition between the Nios II data cache updating the SDRAM with the modified value and the DMA transfer between the SDRAM and the framebuffer, and fixed it by using the IO macros to directly force a write to SDRAM.

3.2 Robot

The platform we use to mount our camera is the popular iRobot Create, a simple robot that is capable of moving in a 2-dimensional plane using differential drive. However, we leave it tethered by serial cable to the FPGA board and only use its rotational capabilities about the vertical axis to track targets.

Commanding the robot is accomplished using an RS-232 interface and a custom cable that comes with the robot. The robot provides operation codes to basic commands, such as rotating in a given direction at a given speed till the given angle has been spanned. Since a serial device is a ready made peripheral for the Nios II processor, we use it and treat it as a black box that we drop into our system architecture. The opcodes themselves are simply byte values transmitted in a certain order.

Our image processing software analyzes an image and determines the orientation of the target compared to the center of the screen, and then issues a command to turn either right or left to the robot until the tracked object is in the center of the camera's field of view.

The only complication we ran into with the robot was with the physical RS-232 port. Both the robot cable and the FPGA board have female interfaces, requiring us to implement a null modem ourselves.

4. Hardware-Software Integration

4.1 Nios II Software

Figure 3 shows the software structure of the system. The software polls the status of the ADV7181 controller until it encounters the beginning of a frame of video in the correct field. It then transfers a line of video at a time, using the DMA controller, to a buffer in SDRAM. For each line, it waits for the proper horizontal clock in order to copy the line at the appropriate time. By changing the number of lines copied and the number of bytes to copy from the line buffer to the SDRAM for each line, the resolution of the image can be tuned; currently it is set to 320×200. Only the first field of video is copied, since it already contains every alternating line and only 200 out of every 484 lines of video are needed. After copying the frame of video to the buffer, the object tracking algorithm is used to find the center of the object. The details of the tracking algorithm are described in the next section. The center of the object is then marked up with a crosshair, and the UART is used to move the robot either left or right, with the speed of the robot increasing with increasing distance from the object center to the center of the frame. Finally, the marked-up image is copied using the DMA controller to the SRAM buffer in the VGA controller.

4.2 Recognition Algorithm

Our object tracking algorithm is designed to recognize objects according to their colors which differentiate them from the background. The input of the algorithm is the 320×200 buffer of RGB 6:5:6 format image data we store in SDRAM. The return value of the algorithm is the approximate center of the object, which is used to command the robot to turn and track the object. Due to the limitations of the Nios II and timing constraints imposed by real-time video display, we use a very simple object tracking algorithm, which only works under certain assumptions:

- The object has a pre-defined color
- The object has regular shape
- Background is simple and has a much different color than the target object

For each input frame, we first divided the whole frame into small blocks (16×16 in our final implementation). Then we calculate how many pixels have the color of the target object, which is the color the center of the object we

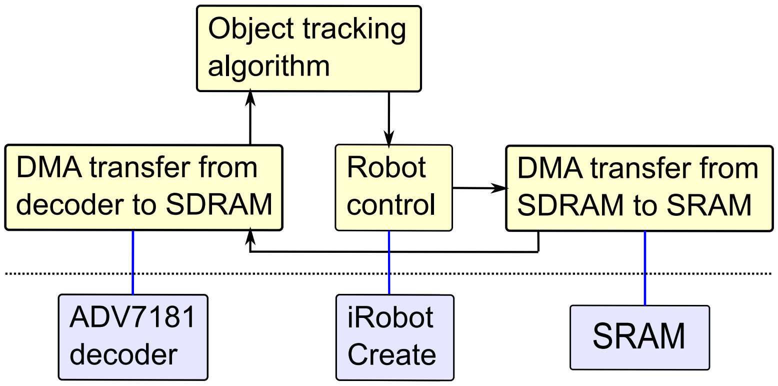
Object tracking algorithm
DMA transfer from decoder to SDRAM
Robot control
DMA transfer from SDRAM to SRAM
ADV7181 decoder
iRobot Create
SRAM

Figure 3. Software design

Algorithm 1 track_object(image, center_row, center_col)

```
1:  top = image's last row, bottom = image's first row
2:  left = image's last column, right = image's first column
3:  for image's each $16 \times 16$ block, $b_i$ do
4:    for each pixel $p$ in $b_i$ do
5:      let (r,g,b) = p
6:      if $Target\_R - 10 \leq r \geq Target\_R + 10$ then
7:        $sum_r$++
8:      end if
9:      if $Target\_G - 10 \leq g \geq Target\_G + 10$ then
10:       $sum_g$++
11:     end if
12:     if $Target\_B - 10 \leq b \geq Target\_B + 10$ then
13:       $sum_b$++
14:     end if
15:   end for
16:   if $(sum_r \geq Block\_Threshold) \&\& (sum_g \geq Block\_Threshold) \&\& (sum_b \geq Block\_Threshold)$ then
17:     let $(b_x, b_y) = b_i$'s center
18:     if $b_x \leq top$ then
19:       top = $b_x$
20:     end if
21:     if $b_x \geq bottom$ then
22:       bottom = $b_x$
23:     end if
24:     if $b_y \leq left$ then
25:       left = $b_y$
26:     end if
27:     if $b_y \geq right$ then
28:       right = $b_y$
29:     end if
30:   end if
31: end for
32: $new\_row = (top + bottom)/2$
33: $new\_col = (left + right)/2$
34: return (new_row,new_col)
```

get from last frame For the first frame, we simply use the center of the screen as the object center. To decide whether a pixel has the same color as the object, we separately compare each pixel's R,G, and B with the target's value within a predefined threshold. Afterwards, we judge if this block belongs to the target object based on if the majority of its pixels are of the same color as the object. This is done for every block, and so we know which blocks belong to the object. After that, we average the positions of the highest, lowest, leftmost and rightmost blocks to calculate the center of the object. The center value is saved and used for finding blocks of the object in next frame.

By processing the image block by block, we reduce the number of membership decisions from one pixel a time to one block a time. This technique also has a reasonable ability to tolerate noise in the image. However, due to the Nios II's limited computational power, we cannot do multiplication for each pixel in each frame as fast as we need. Thus, we turn every possible integer multiplication into integer shifts and additions. For example, $x \times 320 = (x \ll 8) + (x \ll 6)$. In addition, we implement division in terms of multiplication and addition:

$$\frac{x}{5} = \frac{x}{4+1} = \frac{\frac{x}{4}}{1+\frac{1}{4}} = \frac{x}{4} \times \left(1 - \frac{1}{4} + \frac{1}{16} - \frac{1}{64} + \frac{1}{256} \cdots\right)$$

5. Conclusion

Over the course of a semester, we designed and implemented an object tracking system, configurable to track regular shaped objects of various colors, from a collection of disparate hardware peripherals and software components. Over the course of the semester, we learned several lessons that apply to designing and programming embedded systems and to project management in general.

Chief among these is managing time. Verifying a design by compiling it on an FPGA board takes far longer than testing out a software algorithm, so one has to be careful to avoid mistakes. In addition, designing hardware that needs to process video requires that one pay attention to timing requirements, since video is real time and image quality is very sensitive to pulling data at the right time.

In order to make development easier, we learned that dividing the system into smaller, faster compiling, and more easily testable blocks turned a complicated project into a far more manageable one. It also provided a natural breakdown of responsibilities so everyone could contribute to the final project, allowing us to use our available manpower as efficiently as possible.

We also acquired an appreciation for the differences inherent in doing computation in hardware and on a general purpose processor. On hardware we are guaranteed that a given design will finish a computation in a predictable fashion, while software timing is very soft. For that reason, it is difficult to integrate software components with a hardware design.

Future directions for this project would involve tweaking our image recognition, and building a better interface for our system.

6. File Listings

6.1 tv_controller.vhd

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

entity tv_controller is
  port (
    clk           : in std_logic;          -- 50 MHz clock (system)
    reset_n       : in std_logic;
    read          : in std_logic;
    write         : in std_logic;
    chipselect    : in std_logic;
    address       : in unsigned(10 downto 0);
    readdata      : out unsigned(15 downto 0);
    writedata     : in unsigned(15 downto 0);
    clk27         : in std_logic;        -- 27 MHz clock (video)
    td_data       : in unsigned(7 downto 0);
    td_hs         : in std_logic;
    td_vs         : in std_logic
  );

end tv_controller;

architecture rtl of tv_controller is

  component adv_interface is
    port (
      clk27     : in  std_logic;
      reset     : in  std_logic;
      out_data  : out unsigned(31 downto 0);
      data_good : out std_logic;
      field_out : out std_logic;
      hcount    : out unsigned(10 downto 0);
      vcount    : out unsigned(9 downto 0);
      td_data   : in  unsigned(7 downto 0);
      td_hs     : in  std_logic;
      td_vs     : in  std_logic
    );
  end component;

  component yuv2rgb is
    port (
      yuv   : in unsigned (31 downto 0);
      rgb   : out unsigned (15 downto 0)
      );
  end component;


  component line_buffer is
    port (
      reset           : in std_logic;
      wclk       : in std_logic;
      rclk       : in std_logic;
      write_enable    : in std_logic;
      write_data      : in unsigned(15 downto 0);
      write_address   : in unsigned(8 downto 0);
      read_data       : out unsigned(15 downto 0);
      read_address    : in unsigned(8 downto 0);
      page            : in std_logic
    );
  end component;

  -- the 32 bits at a time (in YUV format) that the interface sends out
  signal yuv : unsigned(31 downto 0);
  -- the converted 16 RGB bits
  signal rgb : unsigned(15 downto 0);
  -- tells the line buffer when to accept the data from the video interface
  signal data_good : std_logic;

  signal hcount : unsigned(15 downto 0);
  signal vcount : unsigned(15 downto 0);

  signal buffer_out : unsigned(15 downto 0);
  signal field_out : unsigned(15 downto 0);

begin

  adv_interface0 : adv_interface port map (
    clk27     => clk27,
    reset     => not reset_n,
    out_data  => yuv,
    data_good => data_good,
    field_out => field_out(0),
    hcount    => hcount(10 downto 0),
    vcount    => vcount(9 downto 0),
    td_data   => td_data,
    td_hs     => td_hs,
    td_vs     => td_vs
  );

  yuv2rgb0 : yuv2rgb port map (
    yuv => yuv,
    rgb => rgb
  );

  line_buffer0 : line_buffer port map (
    reset         => not reset_n,
    wclk          => clk27,
    rclk          => clk,
    write_enable  => data_good,
    write_data    => rgb,
    write_address => hcount(10 downto 2),
    read_data     => buffer_out,
    read_address  => address(8 downto 0),
    page          => vcount(0)        -- flip the buffers in each
                                      -- vertical line
  );
  readdata <= buffer_out when address(10 downto 9) = "00" else
              field_out when address(10 downto 9) = "01" else
              hcount when address(10 downto 9) = "10" else
              vcount;
end rtl;
```

6.2 adv_interface.vhd

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

entity adv_interface is

  port (
    clk27     : in std_logic;
    reset     : in std_logic;

    -- data is output two pixels (= 32 bits) at a time
    out_data  : out unsigned(31 downto 0);
    -- this signal is asserted to indicate that the data in
    -- out_data will be valid two rising edges of td_clk27 later
    data_good : out std_logic;
    field_out   : out std_logic;
    hcount : out unsigned(10 downto 0);
    vcount: out unsigned(9 downto 0);
    -- ADV7181 decoder signals
    td_data   : in unsigned(7 downto 0);
    td_hs     : in std_logic;
    td_vs     : in std_logic
  );

end adv_interface;

architecture rtl of adv_interface is


  -- data is temporarily put here before being copied to out_data
  signal data_buffer   : unsigned(31 downto 0) := (others => '0');

  -- keeps track of the horizontal raster position
  signal h_counter     : unsigned(10 downto 0);
  -- keeps track of the current line
  signal v_counter     : unsigned(9 downto 0);
    -- 0 if this is the first interlaced field, 1 if this is the second
  signal field         : std_logic := '0';

  -- high whenever horizontal/vertical sync is pulled low; low whenever
  -- horizontal/vertical sync is pulled high again. Ensures that certain
  -- actions are done only once on the falling edge of horiz/vertical sync.
  signal hit_hsync : std_logic := '0';
  signal hit_vsync : std_logic := '0';
begin

  -- continually copy data from ADV7181 into buffer and output it;
  -- also, update the horizontal and vertical counters as well as
  -- the current field
  GetData : process (clk27)
  begin
    if rising_edge(clk27) then
      if reset = '1' then
```

```
        data_buffer <= (others => '0');
        out_data <= (others => '0');
        data_good <= '0';
        h_counter <= (others => '0');
        v_counter <= (others => '0');
        hit_hsync <= '0';
        hit_vsync <= '0';
        field <= '0';
      else
        if h_counter(1 downto 0) = "00" then
          data_buffer(31 downto 24) <= td_data;
          data_good <= '1';
        elsif h_counter(1 downto 0) = "01" then
          data_buffer(23 downto 16) <= td_data;
        elsif h_counter(1 downto 0) = "10" then
          data_buffer(15 downto 8) <= td_data;
          data_good <= '0';
        else
          data_buffer(7 downto 0) <= td_data;
          out_data <= data_buffer;
        end if;
        h_counter <= h_counter + "00000000001";
        -- Hit hsync? Record that it happened, reset horizontal
        -- counter and increment vertical counter
        if td_hs = '0' and hit_hsync = '0' then
          hit_hsync <= '1';
          h_counter <= "00000000000";
          v_counter <= v_counter + "0000000001";
        elsif td_hs = '1' then
          hit_hsync <= '0';
        end if;
        -- Hit vsync? Record that it happened, reset vertical counter,
        -- and change the field
        if td_vs = '0' and hit_vsync = '0' then
          hit_vsync <= '1';
          v_counter <= (others => '0');
          field <= not field;
        end if;
        if td_vs = '1' then
          hit_vsync <= '0';
        end if;
      end if;
    end if;
  end process GetData;

  hcount <= h_counter;
  vcount <= v_counter;
  field_out <= field;

end rtl;
```

6.3 yuv2rgb.vhd

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

entity yuv2rgb is
  port (
     -- 31         0
     -- y   u   y  v
     yuv : in unsigned (31 downto 0):=x"00000000";
     -- r:5 g:6 b:5
     rgb : out unsigned (15 downto 0):=x"0000"
  );
end yuv2rgb;

architecture y2r of yuv2rgb is
signal y: integer := 0;
signal u: integer := 0;
signal v: integer := 0;
signal r: integer := 0;
signal g: integer := 0;
signal b: integer := 0;


begin
  r <= y*298+v*409+128;
  g <= y*298-u*100-v*208+128;
  b <= y*298+u*516+128;

  y <= to_integer(yuv(31 downto 24))-16;
  u <= to_integer(yuv(7 downto 0))-128;
  v <= to_integer(yuv(23 downto 16))-128;

  rgb(15 downto 11) <= "11111" when r > 65535 else
                       "00000" when r < 0 else
                       to_unsigned(r/2048, 5);
  rgb(10 downto 5) <= "111111" when g > 65535 else
                      "000000" when g < 0 else
                      to_unsigned(g/1024, 6);
  rgb(4 downto 0) <= "11111" when b > 65535 else
                     "00000" when b < 0 else
                     to_unsigned(b/2048, 5);
end y2r;
```

6.4 line_buffer.vhd

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

entity line_buffer is
  port (
    reset   : in std_logic;
    wclk   : in std_logic;
    rclk   : in std_logic;
    write_enable  : in std_logic;
    write_data   : in unsigned(15 downto 0);
    write_address : in unsigned(8 downto 0);
    read_data   : out unsigned(15 downto 0);
    read_address  : in unsigned(8 downto 0);
    -- page = 0: write to line1, read from line2
    -- page = 1: write to line2, read from line1
    page          : in std_logic
  );
end line_buffer;

architecture RTL of line_buffer is

  type ram_type is array(0 to 450) of unsigned(15 downto 0);

  signal line1: ram_type;
  signal line2: ram_type;

begin

  WriteLine : process (wclk)
  begin
    if rising_edge(wclk) then
      if reset = '1' then
        line1 <= (others => x"0000");
        line2 <= (others => x"0000");
      elsif write_enable = '1' then
        if page = '0' then
          line1(to_integer(write_address)) <= write_data;
        else
          line2(to_integer(write_address)) <= write_data;
        end if;
      end if;
    end if;
  end process;

  ReadLine : process (rclk)
  begin
    if rising_edge(rclk) then
      if page = '0' then
        read_data <= line2(to_integer(read_address));
      else
        read_data <= line1(to_integer(read_address));
      end if;
    end if;
  end process;

end RTL;
```

6.5 vga_fb.vhd

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

-- During vblank, we transfer image to SRAM using DMA and Avalon Flow Control
-- During vactive, display whatever is in SRAM to VGA screen

entity vga_fb is

port (
reset_n : in std_logic;
clk   : in std_logic;  -- 50 Mhz; we downscale to 25 MHz ourselves

        -- Avalon Bus signals
signal chipselect : in std_logic;
signal write, read : in std_logic;
signal address  :  in std_logic_vector(17 downto 0);
signal readdata : out std_logic_vector(15 downto 0);
signal writedata : in std_logic_vector(15 downto 0);
signal byteenable : in std_logic_vector(1 downto 0);
signal readyfordata : out std_logic;

        -- Signals for the framebuffer in SRAM
signal SRAM_DQ   : inout std_logic_vector(15 downto 0);
signal SRAM_ADDR : out std_logic_vector(17 downto 0);
signal SRAM_UB_N, SRAM_LB_N : out std_logic;
signal SRAM_WE_N, SRAM_CE_N : out std_logic;
signal SRAM_OE_N            : out std_logic;

        -- VGA Output signals
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 unsigned(9 downto 0) -- Blue[9:0]

);

end vga_fb;

architecture dp of vga_fb is

-- Video parameters
    constant HRES         : integer := 320;
    constant VRES         : integer := 200;

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 FB_MIN_ROW   : integer := 0;
constant FB_MAX_ROW   : integer := VRES - 1;
constant FB_MIN_COL   : integer := 0;
constant FB_MAX_COL   : integer := HRES - 1;

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

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

  -- 25 MHz clock for all video signal control
    signal video_clk : std_logic;

signal reset : std_logic;

signal row, col : unsigned(17 downto 0);
signal FB_Addr : unsigned (17 downto 0);

signal flowready : std_logic := '0';

begin

  reset <= not reset_n;

  -- Get address of SRAM for current pixel
  FB_Addr <= to_unsigned( ((to_integer(ROW) * HRES) + to_integer(COL)) , 18);

  -- downscale 50 MHz clock to 25 MHz video clock
  VideoClock: process (clk)
  begin
    if rising_edge(clk) then
      video_clk <= not video_clk;
    end if;
  end process VideoClock;

SRAM_DQ <= writedata when flowready = '1' and write = '1'
                 else (others => 'Z');
readdata <= SRAM_DQ;
SRAM_ADDR <= address when flowready = '1'
                 else std_logic_vector(FB_Addr);
SRAM_UB_N <= not byteenable(1) when flowready = '1' else '0';
SRAM_LB_N <= not byteenable(0) when flowready = '1' else '0' ;
SRAM_WE_N <= not write when flowready = '1' else '1';
```

```
SRAM_CE_N <= not chipselect when flowready = '1' else '0';
SRAM_OE_N <= not read when flowready = '1' else '0';

  -- AvalonValid lets avalon know when we are ready for data transfer
AvalonValid : process (clk)
begin
if rising_edge(clk) then
if Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
flowready <= '1';
readyfordata <= '1';
elsif Vcount = VSYNC + VBACK_PORCH - 1 then
flowready <= '0';
readyfordata <= '0';
end if;
end if;
end process AvalonValid;

-- Horizontal and vertical counters

HCounter : process (video_clk)
variable c : integer;
  begin
    if rising_edge(video_clk) then
      if reset = '1' then
        Hcount <= (others => '0');
COL <= (others => '0');
      elsif EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
      c := (to_integer(Hcount) - (HSYNC + HBACK_PORCH)) / 2;
      if c > FB_MAX_COL then
          c := FB_MAX_COL;
      elsif c < FB_MIN_COL then
          c := FB_MIN_COL;
      end if;
      COL <= to_unsigned(c, 18);
    end if;
  end process HCounter;

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

VCounter: process (video_clk)
variable r : integer;
  begin
    if rising_edge(video_clk) then
        if reset = '1' then
          Vcount <= (others => '0');
  ROW <= (others => '0');
        elsif EndOfLine = '1' then
            if EndOfField = '1' then
              Vcount <= (others => '0');
            else
```

```
              Vcount <= Vcount + 1;
            end if;
    r := ((to_integer(Vcount) - (VSYNC + VBACK_PORCH) -
                          (VACTIVE/2 - VRES)) / 2);
            if r > FB_MAX_ROW then
                r := FB_MAX_ROW;
            elsif r < FB_MIN_ROW then
                r := FB_MIN_ROW;
            end if;
            ROW <= to_unsigned(r, 18);
        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 (video_clk)
  begin
    if rising_edge(video_clk) then
      if reset = '1' or EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
      end if;
    end if;
  end process HSyncGen;

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

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

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

  -- Registered video signals going to the video DAC

  VideoOut: process (video_clk, reset)
  variable r, b : std_logic_vector(4 downto 0);
  variable g : std_logic_vector(5 downto 0);
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif video_clk'event and video_clk = '1' then
      if vga_hblank = '0' and vga_vblank ='0' then
if vcount > vsync + vback_porch + ((vactive/2) - vres) and
vcount < vsync + vback_porch +
                          ((vactive/2) - vres) + (vres * 2) - 1 then
r := SRAM_DQ(15 downto 11);
g := SRAM_DQ(10 downto 5);
b := SRAM_DQ(4 downto 0);

VGA_R(9 downto 5) <= unsigned(r);
VGA_G(9 downto 4) <= unsigned(g);
VGA_B(9 downto 5) <= unsigned(b);

VGA_R(4 downto 0) <= "00000";
VGA_G(3 downto 0) <= "0000";
VGA_B(4 downto 0) <= "00000";
else
        VGA_R <= "0000000000";
VGA_G <= "0000000000";
VGA_B <= "0000000000";
end if;
      else
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;

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

end dp;
```

6.6 pots.vhd

```
-- Top-level entity for the POTS project. Note: KEY(0) resets the
-- ADV7181, while KEY(2) resets the I2C controller. When first starting
-- the board, it is necessary to press KEY(0) and then KEY(2) to reset both.
-- This sometimes needs to be done multiple times before the ADV7181
-- produces valid data.

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

entity pots is
  port (
    -- Clocks

    CLOCK_27,                                      -- 27 MHz
    CLOCK_50 : in std_logic;                       -- 50 MHz

    -- 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

    -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0);         -- Data bus
    SRAM_ADDR : out std_logic_vector(17 downto 0);         -- Address bus
    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

    -- 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]
    -- TV input

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

    -- Buttons and switches

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

    -- I2C bus

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

    -- UART

    UART_TXD : out std_logic;
    UART_RXD : in std_logic
  );
end pots;

architecture datapath of pots is

component nios_system is
    port (
      -- global signals
      signal clk     : in std_logic;
  signal reset_n : in std_logic;

      -- 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;

      -- VGA framebuffer
      signal sram_addr_from_the_vga      : out std_logic_vector (17 downto 0);
      signal sram_ce_n_from_the_vga      : out std_logic;
      signal sram_dq_to_and_from_the_vga : inout std_logic_vector (15 downto 0);
      signal sram_lb_n_from_the_vga : out std_logic;
      signal sram_oe_n_from_the_vga : out std_logic;
      signal sram_ub_n_from_the_vga : out std_logic;
      signal sram_we_n_from_the_vga : out std_logic;
      signal vga_blank_from_the_vga : out std_logic;
      signal vga_b_from_the_vga : out std_logic_vector (9 downto 0);
      signal vga_clk_from_the_vga : out std_logic;
      signal vga_g_from_the_vga : out std_logic_vector (9 downto 0);
      signal vga_hs_from_the_vga : out std_logic;
      signal vga_r_from_the_vga : out std_logic_vector (9 downto 0);
      signal vga_sync_from_the_vga : out std_logic;
      signal vga_vs_from_the_vga : out std_logic;

      -- TV input
      signal clk27_to_the_tv_in   : in std_logic;
      signal td_data_to_the_tv_in : in unsigned(7 downto 0);
      signal td_hs_to_the_tv_in : in std_logic;
      signal td_vs_to_the_tv_in : in std_logic;

      -- UART
      signal rxd_to_the_uart : in std_logic;
      signal txd_from_the_uart : out std_logic
    );
  end component;

  component sdram_pll
    port (
      inclk0 : in std_logic;
      c0 : out std_logic;
      c1 : out std_logic
    );
  end component;

  component de2_i2c_av_config is
    port (
      iCLK : in std_logic;
      iRST_N : in std_logic;
      I2C_SCLK : out std_logic;
      I2C_SDAT : inout std_logic
    );
  end component;

  signal ba  : std_logic_vector(1 downto 0);
  signal dqm : std_logic_vector(1 downto 0);

  signal pll_c1 : std_logic;

  signal sram_addr_conv : std_logic_vector(17 downto 0);
  signal sram_dq_conv   : std_logic_vector(15 downto 0);

  signal vga_r_conv : std_logic_vector(9 downto 0);
  signal vga_g_conv : std_logic_vector(9 downto 0);
  signal vga_b_conv : std_logic_vector(9 downto 0);

begin
  TD_RESET <= KEY(0);

  dram_ba_1 <= ba(1);
  dram_ba_0 <= ba(0);
  dram_udqm <= dqm(1);
  dram_ldqm <= dqm(0);
  nios: nios_system port map (
    clk => pll_c1,
    reset_n => KEY(1),

    zs_addr_from_the_sdram => DRAM_ADDR,
    zs_ba_from_the_sdram => BA,
    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 => DQM,
    zs_ras_n_from_the_sdram => DRAM_RAS_N,
    zs_we_n_from_the_sdram => DRAM_WE_N,

    sram_addr_from_the_vga => SRAM_ADDR,
    sram_ce_n_from_the_vga => SRAM_CE_N,
    sram_dq_to_and_from_the_vga => SRAM_DQ,
    sram_lb_n_from_the_vga => SRAM_LB_N,
    sram_oe_n_from_the_vga => SRAM_OE_N,
    sram_ub_n_from_the_vga => SRAM_UB_N,
    sram_we_n_from_the_vga => SRAM_WE_N,

    vga_blank_from_the_vga => VGA_BLANK,
    vga_b_from_the_vga => VGA_B,
    vga_clk_from_the_vga => VGA_CLK,
    vga_g_from_the_vga => VGA_G,
    vga_hs_from_the_vga => VGA_HS,
    vga_r_from_the_vga => VGA_R,
    vga_sync_from_the_vga => VGA_SYNC,
    vga_vs_from_the_vga => VGA_VS,

    clk27_to_the_tv_in => CLOCK_27,
    td_data_to_the_tv_in => TD_DATA,
    td_hs_to_the_tv_in => TD_HS,
    td_vs_to_the_tv_in => TD_VS,

    rxd_to_the_uart => uart_rxd,
    txd_from_the_uart => uart_txd
  );

  neg_3ns : sdram_pll port map (CLOCK_50, DRAM_CLK, pll_c1);

  i2c: de2_i2c_av_config port map (
    iCLK => CLOCK_50,
    iRST_N => KEY(2),
    I2C_SCLK => I2C_SCLK,
    I2C_SDAT => I2C_SDAT
  );
end datapath;
```

6.7 pots.c

```
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <sys/alt_dma.h>
#include <io.h>
#include "img_tracking.h"

#define VMAX 400 // maximum velocity allowed by the robot

#define CROSSCOLOR 0x0000

#define IORD_TV_IN_FIELD(base)  \
    IORD_16DIRECT(base, 1024)
#define IORD_TV_IN_HCOUNT(base) \
    IORD_16DIRECT(base, 2048)
#define IORD_TV_IN_VCOUNT(base) \
    IORD_16DIRECT(base, 3072)

#define HRES 320
#define VRES 200

static volatile int rx_done = 0;
static volatile int tx_done = 0;

FILE *serial;

static void snd_done(void *handle)
{
    tx_done = 1;
}

static void rcv_done(void *handle, void *data)
{
    rx_done = 1;
}

static void draw_crosshair(short *buf, int x, int y) {
   IOWR_SDRAM(buf+y*HRES+x, 0xffff);
    if (y < VRES-1) {
       IOWR_SDRAM(buf+(y+1)*HRES+x, CROSSCOLOR);
    }
    if (y > 1) {
     IOWR_SDRAM(buf+(y-1)*HRES+x, CROSSCOLOR);
    }
    if (x < HRES-1) {
     IOWR_SDRAM(buf+y*HRES+(x+1), CROSSCOLOR);
    }
    if (x > 1) {
       IOWR_SDRAM(buf+y*HRES+(x-1), CROSSCOLOR);
    }
}
```

```
static inline void initSerial()
{
    serial = fopen("/dev/uart", "r+");
    if (!serial) {
        perror("fopen");
        exit(1);
    }
    printf("Serial is opened!\n");
}

static inline void writeSerial(unsigned char *buf, int bytes)
{
    int written;

    if (!serial) {
        printf("Serial port not opened, returning...\n");
        return;
    }

    written = fwrite(buf, 1, bytes, serial);
    if (written == -1) {
        perror("write");
    }
}

static inline void startRobot()
{
    unsigned char data[2];
    data[0] = (unsigned char) 128;
    data[1] = (unsigned char) 131;
    writeSerial(data, 2);
}

static inline void turnRight(int speed)
{
    unsigned char data[5];

    if (speed < 1 || speed > VMAX)
        return;

    data[0] = 145;  //Direct Drive command
    data[1] = (unsigned char) ((-speed) >> 8 & 0x00FF);  //[Right velocity high byte]
    data[2] = (unsigned char) ((-speed) & 0x00FF);    //[Right velocity low byte]
    data[3] = (unsigned char) ((speed) >> 8 & 0x00FF);  //[Left velocity high byte]
    data[4] = (unsigned char) ((speed) & 0x00FF);    //[Left velocity low byte]

    writeSerial(data, 5);
}


static inline void turnLeft(int speed)
{
    unsigned char data[5];
    if (speed < 1 || speed > VMAX)
        return;

    data[0] = 145;  //Direct Drive command
    data[1] = (unsigned char) ((speed) >> 8 & 0x00FF);  //[Right velocity high byte]
    data[2] = (unsigned char) ((speed) & 0x00FF);    //[Right velocity low byte]
    data[3] = (unsigned char) ((-speed) >> 8 & 0x00FF);  //[Left velocity high byte]
    data[4] = (unsigned char) ((-speed) & 0x00FF);    //[Left velocity low byte]

    writeSerial(data, 5);
}

static inline void stopRobot()
{
    unsigned char data[5];

    data[0] = 145;  //Direct Drive command
    data[1] = (unsigned char) 0;  //[Right velocity high byte]
    data[2] = (unsigned char) 0;    //[Right velocity low byte]
    data[3] = (unsigned char) 0;  //[Left velocity high byte]
    data[4] = (unsigned char) 0;    //[Left velocity low byte]

    writeSerial(data, 5);
}

int main()
{
  int i, j = 0, rc, ret, speed;
  unsigned short hc, vc, vc2;

  int row=-1, col=-1;

  alt_dma_txchan txchan;
  alt_dma_rxchan rxchan;

  printf("started running!\n");

  initSerial();
  startRobot();

  printf("Initialized robot!\n");

  short *buf = malloc(sizeof(short) * (HRES*VRES));
  if (buf == NULL) {
    printf("Could not alloc!\n");
    exit(1);
  }

  // clear buffer at first
  for (i=0; i < HRES*VRES; ++i) {
    IOWR_SDRAM(buf + i, 0);
  }

  // set up DMA
  if ((txchan = alt_dma_txchan_open("/dev/dma")) == NULL) {
    printf("Failed to open transmit channel.\n");
    exit(1);
  }

  if ((rxchan = alt_dma_rxchan_open("/dev/dma")) == NULL) {
    printf("Failed to open receive channel.\n");
    exit(1);
  }

  ret = alt_dma_txchan_ioctl(txchan, ALT_DMA_SET_MODE_16, NULL);
  if (ret < 0) {
    printf("IOCTL failed, could not set mode 16.\n");
    exit(1);
  }

  ret = alt_dma_rxchan_ioctl(rxchan, ALT_DMA_SET_MODE_16, NULL);
  if (ret < 0) {
    printf("IOCTL failed, could not set mode 16.\n");
    exit(1);
  }

  ret = alt_dma_rxchan_ioctl(rxchan, ALT_DMA_RX_ONLY_OFF, NULL);

  if (ret < 0) {
    printf("IOCTL failed, could not set default mode.\n");
    exit(1);
  }
  ret = alt_dma_txchan_ioctl(txchan, ALT_DMA_TX_ONLY_OFF, NULL);
  if (ret < 0) {
    printf("IOCTL failed, could not set default mode.\n");
    exit(1);
  }

  // main program loop
  while(1) {
  rx_done = 0;

  // transfer one dummy line so that the next transfer doesn't
  // result in a delay
  alt_dma_txchan_send(txchan, (void *)(TV_IN_BASE + 176),
      HRES*2, snd_done, NULL);
  alt_dma_rxchan_prepare(rxchan, buf, HRES*2, rcv_done, NULL);
  while (rx_done == 0);
  rx_done = 0;
  // wait for vsync
  while (IORD_TV_IN_HCOUNT(TV_IN_BASE) > 100 ||
 (IORD_TV_IN_FIELD(TV_IN_BASE) & 0x0004) == 1 ||
        IORD_TV_IN_VCOUNT(TV_IN_BASE) != 0);
  for(i = 0; i < VRES; i++)
    {
      // wait for hsync
       do {
        hc = IORD_TV_IN_HCOUNT(TV_IN_BASE);
       }
       while (hc < 200 || hc > 300);

       alt_dma_txchan_send(txchan, (void *)(TV_IN_BASE + 176),
   HRES*2, snd_done, NULL);
       alt_dma_rxchan_prepare(rxchan, buf+HRES*i, HRES*2, rcv_done, NULL);



       while (rx_done == 0);
       rx_done = 0;
    }
    rx_done = 0;


    track_obj4(buf, row, col, &row, &col);
    draw_crosshair(buf, col, row);

    // set the speed of the robot based on how far the center of the
    // object is from the center of the image
    speed = col - 160;
    if (speed < 0)
        speed = -speed;
    speed += 40;
    if (speed > 80) speed = 80;

    // turn the robot
    if (col < 145) {
        turnLeft(speed);
    }
    else if (col > 175) {
        turnRight(speed);
    }
    else {
        stopRobot();
    }

    // transfer from buffer to video buffer in SRAM
    alt_dma_txchan_send(txchan, buf, HRES*VRES*2, snd_done, NULL);
    alt_dma_rxchan_prepare(rxchan, (void*)(VGA_BASE), HRES*VRES*2,
   rcv_done, NULL);
    while (!tx_done);
    tx_done = 0;
  }

  free(buf);
  return 0;
}
```

6.8 img_tracking.c

```
/******************* img_tracking.c *****************************************
/* Author: Baolin Shao
 * Date: 2009 04 10
 * Description: This file has a very simple object tracking algorithm.
 *              The algoirthm "tracks" an object based on the object's
 *              position. Given the object's previuos centroid, this
 *              algorithm calcutes the object's current centroid.
 *              This algorithm works based on the following assumption:
 *              1. The object's initial position is in the center of image
 *              2. The object does not move dramatically between two
 *                  succesive images
 *              3. The object's color is reasonably different from its
 *                  background
 ****************************************************************************/
#include "stdlib.h"
#include "stdio.h"
#include "img_tracking.h"


inline void comp(int *pt_r,int *pt_c)
{
    if(*pt_r<0) *pt_r = 0;
    if(*pt_r>ROW) *pt_r = ROW-1;
    if(*pt_c<0) *pt_r = 0;
    if(*pt_c>COL) *pt_r = COL-1;

}

void get_obj_def(short *buf,
        int center_row,int center_col,
        unsigned char *val_r,
        unsigned char *val_g, unsigned char *val_b)
{
    int c_offset;
    int i;
    c_offset = OFFSET(center_row,center_col);
    short v = IORD_SDRAM(buf+c_offset);
    *val_r = GET_R(v);
    *val_g = GET_G(v);
    *val_b = GET_B(v);

}
void get_obj_val_rgb(short *buf,
int center_row,int center_col,
unsigned char *val_r,
unsigned char *val_g, unsigned char *val_b)
{
unsigned short val =0;
int i;
    int sumr,sumg,sumb;
    sumr=sumg=sumb=0;
    *val_r=0;
    *val_g=0;
```

```
*val_b=0;
int c_offset;
/***************************************/
/*        b b 1 b b                    */
/*        b 2 x 3 b                    */
/*        4 x c x 5                    */
/*        b 6 x 7 b                    */
/*        b b 8 b b                    */
/***************************************/
int p_r[8];
int p_c[8];
p_r[0] = center_row-2;
p_c[0] = center_col;

p_r[1] = center_row-1;
p_c[1] = center_col-1;

p_r[2] = center_row-1;
p_c[2] = center_col+1;

p_r[3] = center_row;
p_c[3] = center_col-2;

p_r[4] = center_row;
p_c[4] = center_col+2;

p_r[5] = center_row+1;
p_c[5] = center_col-1;

p_r[6] = center_row+1;
p_c[6] = center_col+1;

p_r[7] = center_row+2;
p_c[7] = center_col;

for(i=0;i<8;i++)
  {
    comp(p_r+i,p_c+i);
    c_offset = OFFSET(p_r[i],p_c[i]);
    val=IORD_SDRAM(buf+c_offset);
    sumr+=GET_R(val);
    sumg+=GET_G(val);
    sumb+=GET_B(val);
  }

*val_r = (unsigned char)(sumr>>3);
*val_g = (unsigned char)(sumg>>3);
*val_b = (unsigned char)(sumb>>3);
int of = OFFSET(center_row,center_col);
unsigned short cv = IORD_SDRAM(buf+of);
unsigned char cvr = GET_R(cv);
unsigned char cvg = GET_G(cv);
unsigned char cvb = GET_B(cv);
```

```
}

void get_obj_val(short *buf,
 int center_row,int center_col,
 unsigned char *cval)
{
  unsigned short val;
  unsigned char v_r,v_g,v_b;
  val=0;
  v_r=v_g=v_b=0;
  int i,j;
  int c_offset;
  int left,right,top,bottom;
  left = center_col - 2;
  right = center_col + 1;
  top = center_row - 2;
  bottom = center_row + 1;
  if(left < 0) left =0;
  if(right > COL) right = COL;
  if(top < 0) top =0;
  if(bottom > ROW) bottom = ROW;
  for(i=top;i<=bottom;i++)
    for (j=left;j<=right;j++)
      {
c_offset = OFFSET(i,j);
val=IORD_SDRAM(buf+c_offset);
v_r=GET_R(val);
v_g=GET_G(val);
v_b=GET_B(val);
val = CONVERT(v_r,v_g,v_b);
*cval = *cval + val;
      }
  *cval= (*cval)>>4;
}

static int closeto(int val, int target) {
  if (val < target + PIC_THRESHOLD && val > target - PIC_THRESHOLD)
    return 1;
  return 0;
}


/****************************** track_obj *********************************
   Function: track_obj
   Parameter:
   input:
     short * buf:image data, every pixel constitutes two bytes (1 short).
     int center_row, center_col: previous centroid of object.
   output:
     int *new_center_row, +*new_center_col:newly calculated centroid
**************************************************************************/

void track_obj ( short *buf, int center_row, int center_col,
 int *new_center_row, int *new_center_col)
```

```
{
  int left=COL-1;
  int right=0;
  int top=ROW-1;
  int bottom=16;
  unsigned short center_val=0;
  unsigned char center_val_r=0;
  unsigned char center_val_g=0;
  unsigned char center_val_b=0;
  if(center_row==-1 && center_col==-1)
    {
      center_row=ROW>>1;
      center_col=COL>>1;
    }
  center_val_r=220;
  center_val_g=0;
  center_val_b=220;

  int i,j,k1,k2;
  i=j=k1=k2=0;
  for(i=0;i<ROW;i=i+BLOCK_SIZE)
    {
      for(j=0;j<COL;j=j+BLOCK_SIZE)
{
  unsigned short sum_r,sum_g,sum_b;
  sum_r=sum_g=sum_b=0;
  for(k1=i;k1<i+BLOCK_SIZE;k1++)
    {
      for(k2=j;k2<j+BLOCK_SIZE;k2++)
{
  int offset=OFFSET(k1,k2);
  unsigned short val=IORD_SDRAM(buf+offset);
  unsigned char val_r=0;
  unsigned char val_g=0;
  unsigned char val_b=0;
  val_r=GET_R(val);
  val_g=GET_G(val);
  val_b=GET_B(val);
  if(val_r>center_val_r-PIC_THRESHOLD &&
     val_r<center_val_r+PIC_THRESHOLD)
    sum_r++;

}
    }
  if( closeto(sum_r, center_val_r) && closeto(sum_b, center_val_b))
            {
      if(i<top)
                {
  top=i;
                }
      if(i>bottom)
                {
  if(i+BLOCK_SIZE>ROW)
    bottom = ROW-1;
  else
                    bottom=i+BLOCK_SIZE-1;
              }
      if(j<left)
              {
  left = j;
              }
      if(j>right)
              {
  if(j+BLOCK_SIZE>COL)
    right = COL-1;
  else
    right = j+BLOCK_SIZE-1;
              }
    }
       }
   }
*new_center_row = (top+bottom)>>1;
*new_center_col = (left+right)>>1;

}

void track_obj2 ( short *buf, int center_row, int center_col,
         int *new_center_row, int *new_center_col)
{
    int left=COL-1;
    int right=0;
    int top=ROW-1;
    int bottom=16;

    unsigned char center_val_r=0;
    unsigned char center_val_g=0;
    unsigned char center_val_b=0;
    unsigned char cval=0;
    if(center_row==-1 && center_col==-1)
    {
        center_row=ROW>>1;
        center_col=COL>>1;
    }
    get_obj_def(buf,center_row,center_col,
&center_val_r,&center_val_g,&center_val_b);
    cval = CONVERT(center_val_r,center_val_g,center_val_b);
    int i,j,k1,k2;
    i=j=k1=k2=0;
    int cnt=0;
    for(i=16;i<ROW;i=i+BLOCK_SIZE)
    {
        for(j=0;j<COL;j=j+BLOCK_SIZE)
        {

            unsigned short sum=0;

            for(k1=i;k1<i+BLOCK_SIZE;k1++)
            {
                for(k2=j;k2<j+BLOCK_SIZE;k2++)
                {
                    int offset=OFFSET(k1,k2);
                    unsigned short val=IORD_SDRAM(buf+offset);
                    unsigned char val_r=0;
                    unsigned char val_g=0;
                    unsigned char val_b=0;
                    unsigned char v=0;
                    val_r=GET_R(val);
                    val_g=GET_G(val);
                    val_b=GET_B(val);
                    v = CONVERT(val_r,val_g,val_b);
                    if(v>200 &&
                       v<255)
                        sum++;

                }
            }
            if( sum>BLOCK_THRESHOLD )
            {
                 cnt++;
                if(i<top)
                {
                    top=i;
                }
                if(i>bottom)
                {
                    if(i+BLOCK_SIZE>ROW)
                      bottom = ROW-1;
                    else
                      bottom=i+BLOCK_SIZE-1;
                }
                if(j<left)
                {
                    left = j;
                }
                if(j>right)
                {
                    if(j+BLOCK_SIZE>COL)
                      right = COL-1;
                    else
                      right = j+BLOCK_SIZE-1;
                }
            }
        }
    }
    *new_center_row = (top+bottom)>>1;
    *new_center_col = (left+right)>>1;

}
void track_obj3 ( short *buf, int center_row, int center_col,
         int *new_center_row, int *new_center_col, short *color)
{
    int left=COL-1;
    int right=0;
    int top=ROW-1;
    int bottom=16;
    int c_offset=0;
    unsigned char center_val_r=0;
    unsigned char center_val_g=0;
    unsigned char center_val_b=0;
    unsigned char cval=0;
    if(center_row==-1 && center_col==-1)
    {

        center_row=ROW>>1;
        center_col=COL>>1;
        c_offset = OFFSET(center_row,center_col);
        *color = IORD_SDRAM(buf+c_offset);
    }
    c_offset = OFFSET(center_row,center_col);
    *color = IORD_SDRAM(buf+c_offset);
    center_val_r = GET_R(*color);
    center_val_g = GET_G(*color);
    center_val_b = GET_B(*color);
    cval = CONVERT(center_val_r,center_val_g,center_val_b);
    int i,j,k1,k2;
    i=j=k1=k2=0;
    int cnt=0;
    for(i=16;i<ROW;i=i+BLOCK_SIZE)
    {
        for(j=0;j<COL;j=j+BLOCK_SIZE)
        {

            unsigned short sum=0;

            for(k1=i;k1<i+BLOCK_SIZE;k1++)
            {
                for(k2=j;k2<j+BLOCK_SIZE;k2++)
                {
                    int offset=OFFSET(k1,k2);
                    unsigned short val=IORD_SDRAM(buf+offset);
                    unsigned char val_r=0;
                    unsigned char val_g=0;
                    unsigned char val_b=0;
                    unsigned char v=0;
                    val_r=GET_R(val);
                    val_g=GET_G(val);
                    val_b=GET_B(val);
                    v = CONVERT(val_r,val_g,val_b);
                    if(v>cval-PIC_THRESHOLD &&
                       v<cval+PIC_THRESHOLD)
                        sum++;

                }
            }
            if( sum>BLOCK_THRESHOLD )
            {
```

```
                 cnt++;
                if(i<top)
                {
                    top=i;
                }
                if((i+BLOCK_SIZE)>bottom)
                {
                    if(i+BLOCK_SIZE>ROW)
                      bottom = ROW-1;
                    else
                      bottom=i+BLOCK_SIZE-1;
                }
                if(j<left)
                {
                    left = j;
                }
                if((j+BLOCK_SIZE)>right)
                {
                    if(j+BLOCK_SIZE>COL)
                      right = COL-1;
                    else
                      right = j+BLOCK_SIZE-1;
                }
            }

        }

    }
    *new_center_row = (top+bottom)>>1;
    *new_center_col = (left+right)>>1;
    *color= IORD_SDRAM(buf+ OFFSET(*new_center_row,*new_center_col) );

}


void track_obj4 ( short *buf, int center_row, int center_col,
         int *new_center_row, int *new_center_col)
{
    int left=COL-1;
    int right=0;
    int top=ROW-1;
    int bottom=16;
    unsigned short center_val=0;
    unsigned char center_val_r=0;
    unsigned char center_val_g=0;
    unsigned char center_val_b=0;
    if(center_row==-1 && center_col==-1)
    {
        center_row=ROW>>1;
        center_col=COL>>1;
    }

    int i,j,k1,k2;
```

```
i=j=k1=k2=0;
for(i=0;i<ROW;i=i+BLOCK_SIZE)
{
    for(j=0;j<COL;j=j+BLOCK_SIZE)
    {
        unsigned short sum_r,sum_g,sum_b;
        sum_r=sum_g=sum_b=0;
        for(k1=i;k1<i+BLOCK_SIZE;k1++)
        {
            for(k2=j;k2<j+BLOCK_SIZE;k2++)
            {
                int offset=OFFSET(k1,k2);
                unsigned short val=IORD_SDRAM(buf+offset);
                unsigned char val_r=0;
                unsigned char val_g=0;
                unsigned char val_b=0;
                val_r=GET_R(val);
                val_g=GET_G(val);
                val_b=GET_B(val);
                if(closeto(val_r, TARGET_R))
                    sum_r++;

              if(closeto(val_g, TARGET_G))
                  sum_g++;

              if(closeto(val_b, TARGET_B))
                  sum_b++;
            }
        }

       if( sum_r >= BLOCK_THRESHOLD && sum_g >= BLOCK_THRESHOLD &&
   sum_b >=BLOCK_THRESHOLD )
        {
            if(i<top)
            {
                top=i;
            }
            if(i>bottom)
            {
                if(i+BLOCK_SIZE>ROW)
                  bottom = ROW-1;
                else
                  bottom=i+BLOCK_SIZE-1;
            }
            if(j<left)
            {
                left = j;
            }
            if(j>right)
            {
                if(j+BLOCK_SIZE>COL)
                  right = COL-1;
                else
                  right = j+BLOCK_SIZE-1;
```

```
                    }
                }
            }
        }

        *new_center_row = (top+bottom)>>1;
        *new_center_col = (left+right)>>1;

    }
```

6.9 img_tracking.h

```
#include <stdio.h>
#include <stdlib.h>
#include <system.h>
#include <sys/alt_dma.h>
#include <io.h>
#define ROW 200
#define COL 320


#define TARGET_R 200
#define TARGET_G 100
#define TARGET_B 70


#define BLOCK_SIZE 16
#define PIC_THRESHOLD 30
#define BLOCK_THRESHOLD 180

#define OFFSET(i,j) ((i<<8)+(i<<6))+j // i*COL+j

//get most significant 5 bits, and return a byte
#define GET_R(color) (unsigned char)((color&0xF800)>>8)
//get least significant 5 bits, and return a byte
#define GET_B(color) (unsigned char)((color&0x001F)<<3)
//get the middle 6 bits,5-6-5,from color, and return a byte
#define GET_G(color) (unsigned char)((color&0x07E0)>>3)
//X / 5 = X/(4+1) = (X/4) /(1+1/4)= (X/4) * (1 - 1/4 + 1/16 - 1/64 + 1/256 ...)
//      = X/4 - X/16 + X/64 - X/256 + X/1024 - X/4096 ...
#define DIV(x) (unsigned char)((x>>2)-(x>>4)+(x>>6)-(x>>8))
#define CONVERT(r,g,b) (unsigned char)(DIV(((r>>2)+(g<<4))-(b)))
#define IOWR_SDRAM(base, data) IOWR_16DIRECT(base, 0, data)
#define IORD_SDRAM(base) IORD_16DIRECT(base, 0)

inline void comp(int *pt_r,int *pt_c);

void get_obj_def(short *buf,
        int center_row,int center_col,
        unsigned char *val_r,
        unsigned char *val_g, unsigned char *val_b);

void get_obj_val_rgb(short *buf,
int center_row,int center_col,
unsigned char *val_r,
unsigned char *val_g, unsigned char *val_b);

void get_obj_val(short *buf,
int center_row,int center_col,
unsigned char *cval);
void track_obj ( short *buf, int center_row, int center_col,
         int *new_center_row, int *new_center_col);

void track_obj2 ( short *buf, int center_row, int center_col,
 int *new_center_row, int *new_center_col);
```

```
void track_obj3 ( short *buf, int center_row, int center_col,
         int *new_center_row, int *new_center_col, short *color);
void track_obj4 ( short *buf, int center_row, int center_col,
         int *new_center_row, int *new_center_col);
```