Mudd Rover

☯Final Report☯
Version 1.0

Ron Coleman Josef Brks Schenker Akshay Kumar Athena Ledakis Justin Titi

Table of Contents

1. Overview

The basic idea behind Mudd Rover is to create an autonomous robot with an onboard camera that is capable of finding a line, orienting itself properly and then finally following it. The processing done behind the scenes will take place using the XiLinx Spartan FPGA that is mounted on XSB-300E. Part of the FPGA will be programmed to be our custom video processing hardware. The skeletal form of the robot will be a tank-like vehicle built with Legos™. The method of communication between the XSB-300E and the Mudd Rover will take advantage of the Lego RCX and its companion serial IR Tower. Primitive commands will be sent from the XSB-300E via the serial IR tower to the Lego RCX, which will be mounted on the Mudd Rover.

The seamless communication between the XSB-300E and the Mudd Rover will give the appearance of an intelligent robot with a vast amount of processing power onboard even though 100% of the processing will actually be taking place remotely. There are three distinct parts to this project: Video Processing, Serial / IR Communication, and the Brain of the Mudd Rover.

2. Video Processing

2.1. Overview

The X10 Video camera is connected to the SAA7114 Video Decoder located on the XSB-300E via a composite video cable. The X10 video camera will be sending a constant NTSC signal to the SAA7114 decoder. The SAA7114 decoder will process this stream of video and send it out in digital form to our custom video processor and its accompanying software. This package of hardware and software will be refered to as LightFinder from this point on. LightFinder will then process the digitized video and gather the information needed by the Brain of the Mudd Rover in order to direct Mudd Rover to achieve its task.

2.2. SAA7114 Video Decoder

The Philips SAA7114H chip is configured using the I2C protocol, which requires its own module on the OPB bus. A special thanks must be extended to both Marcio and Cristian who not only wrote most of the code for this part of the project but also made sure that we understood it.

2.3. LightFinder-Hardware

2.3.1. Overview

The LightFinder hardware's main goal is to read the digitized video line by line and find the longest line of pixels together that are above a certain threshold. This has the potential to be an extremely costly operation if done in software but can be achieved with little overhead in hardware. Therefore this part of the project must be implemented in hardware in order to meet the final goal of a robot that is relatively fast to move and respond. Had this part of the project not been done in hardware, it would have been very hard to achieve this goal.

2.3.2. Implementation

The custom video hardware built runs the pixels through a 3 pixel long mask on a line by line basis. By running it through this mask it can determine if any given pixel is 'on.' A pixel is determined to be on if it is below the threshold value, or if the two pixels surrounding it are below the threshold. The use of a 3 pixel mask eliminates the case where a single pixel above the threshold will ruin a line of pixels below the threshold. This is extremely important with regards to our project because we are making use of a relatively low quality video camera that can not be counted on to delivery reliable data.

For example, when our filter is run on the following stream of pixels using a threshold of 50, our filter would register the stream as follows.

Pixel	1	2	3	4	5	6	7	8	9	10	11	12	13
Value	52	51	50	48	50	47	48	51	49	48	50	53	54

Pixel	1	2	3	4	5	6	7	8	9	10	11	12	13
Value	OFF	OFF	ON	ON	ON	ON	ON	ON	ON	ON	ON	OFF	OFF

Notice how pixel 8 is still read by the hardware as ON even though it alone is above the threshold.

Once the hardware has determined the ON/OFF values of the pixels, it then counts to find the longest block of ON pixels. This is accomplished by using two temporary locations: one to hold the longest sequence(both start position and length) of ON pixels and one to hold the current sequence of pixels(both start position and length). These two are compared and when a longer sequence of ON pixels is found, it is stored. Once the line is determined finished by the active video flag, the start position of the longest block and its length are concatenated and written to a single register as a 32-bit number. The first 16 bits are the position and the second sixteen are the length of the line, making it simple for the corresponding LightFinder software to then poll the data with XIo_In32(0x018008FC).

2.4. LightFinder-Software

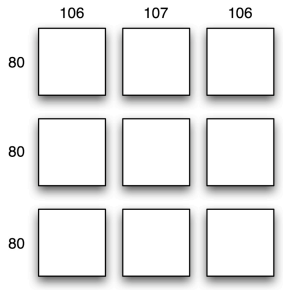
2.4.1. Overview

The output of the LightFinder hardware comes line by line, which is not the most ideal form for the Mudd Rover to base its decisions on. The accompanying LightFinder software transforms this line-by-line data into frame-by-frame data that is more useful for determining the movement of the Mudd Rover. In addition, it divides up the frame into sections and reports the number of pixels ON in each section. This is pivotal to helping determine where the Mudd Rover should move.

2.4.2. Implementation

Every time the value in the register is updated by the hardware, it is stored in an array called `line_array[]`. From the `NOT_VERT_SYNC` signal, we are able to determine when a full frame of information has been processed by the video hardware and stored in the array `line_array[]`.

As the starting position and length of the pixels per line are now stored in `line_array[],` in hexadecimal form, the `lightfinder` program takes this data and gives the `movement()` function a count of pixels in each section of the frame in decimal form. A frame is divided into 9 sections:



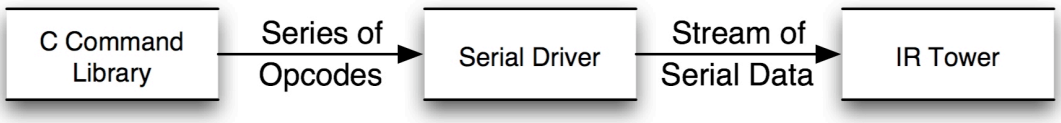
Each value stored in `line_array[]` is parsed and analyzed to find how many pixels fall into each column of the screen. A count is kept of how many pixels are in each section based on the start position of the line in `line_array[]`, the line's length and which index of `line_array[]` is being accessed.

This count of pixels for each section is stored in a two-dimensional array, section[SCREENSECTIONS][3], where SCREENSECTIONS represents the number of rows the frame is divided into and the second value represents the vertical screen sections, left (index 0), middle(index 1) and right (index 2). The section array stores the information that gets used by `movement()`, the function that operates as the Mudd Rover's brain.

3. Communication

3.1. Overview

At the core of the communication between the XSB-300E and the Mudd Rover are two components. The first of these components is the serial driver. The serial driver will take a hexadecimal opcode along with its arguments, make it into a full message, and transmit it over the serial cable to the IR tower. The second component is a small C library that will bunch these opcodes together to simplify and facilitate moving the robot.



3.2. Opcode Basics

An opcode is 1 byte in length. From this point on, these opcode bytes will be referenced using their hexadecimal value followed by a slash and their sister value. Each opcode has a sister opcode that has its 0x08 bit toggled on. This is used when sending the same opcode twice in a row and helps the RCX recognize the second opcode as a different message all together rather than a rebroadcast of the previous message. Below is a list of the opcodes used, which is only a small subset of the full library of opcodes supported by the RCX:

Lego RCX Opcode	Hex Value / Sister	Description
Play Sound	51/59	Play specified sound(for debugging)
Set Motor Direction	E1/E9	Set the direction of specified motors
Set Motor On / Off	21/29	Set the on/off state of the motors accordingly
Set Motor Speed	13/1B	Set the speed of the motors accordingly
Set Transmitter Range	31/39	Set the transmitter range accordingly

3.3. Opcode Format for Transmission

A 3-byte header (0x55 0xff 0x00) that is the same for all messages sent to the IR tower must precede the opcode in each message sent to the RCX. Depending upon the opcode being sent, it can also be accompanied by several bytes of data after the opcode. In addition to these requirements, error correction is done through checksums. After the header, each byte sent is promptly followed by its complement. At the end of the message, an overall checksum is calculated of the data bits (not including the header), which is then sent along with its complement to complete the message.

With this in mind a complete opcode will look like this:

Header	Opcode	~Opcode	Data 1	~Data 1	Data N	~Data N	Checksum	~Checksum

3.4. Opcode Details

3.4.1. Play Sound

Opcode:51/59
Arguments: byte *sound*

Sound	
Index	**Description**
0	Blip
1	Beep Beep
2	Downward Tones
3	Upward Tones
4	Low Buzz
5	Fast Upward Tones

3.4.2. Set Motor Direction

Opcode:E1/E9
Arguments: byte *code*

Code	
Bit	**Description**
0x01	Modify Direction of motor A
0x02	Modify Direction of motor B
0x04	Modify Direction of motor C
0x40	Flip the direction of the specified motors
0x80	Set the directions of the specified motors

3.4.3. Set Motor On / Off

Opcode: 21/29
Arguments: byte *code*

Code	
Bit	**Description**
0x01	Modify On / Off state of motor A
0x02	Modify On / Off state of motor B
0x04	Modify On / Off state of motor C
0x40	Turn off the specified motors
0x80	Turn on the specified motors

3.4.4. Set Motor Speed

Opcode: 13/1B
Arguments: byte *motors*, byte *source*, byte *argument*

Motors	
Bit	**Description**
0x01	Modify power level of motor A
0x02	Modify power level of motor B
0x04	Modify power level of motor C

Source specifies the source type for power level. It can only take on values of 0, 2, and 4.

Argument specifies a value from 0-7 for the power of the motor, with 7 being the fastest.

3.4.5. Set Transmitter Range

Opcode: 31/39
Arguments: byte *range*

Range sets the transmitter to short range when 0 and long range when 1

3.5. Serial Communication

3.5.1. Protocol

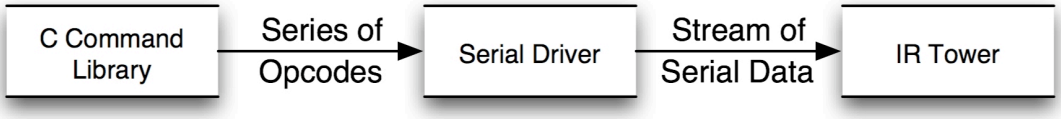
Each message, byte by byte, is sent via serial to the IR tower at with the following specification:

Baud Rate	2400
Non-Return to Zero	Yes
Stop Bit	1
Start Bit	1
Parity	Odd

3.5.2. Implementation

3.5.2.1.Overview

The serial communications package will be written in C and be comprised of two components, a serial driver and a library of C functions.



3.5.2.2.Library of C Functions

The functions are very simple but useful groupings of opcodes to achieve basic actions for the robot. Below are the planned library functions:

Function	Description
pt_turn_left	Turns Mudd Rover left
pt_turn_right	Turns Mudd Rover right
forward	Moves Mudd Rover forwards
reverse	Moves Mudd Rover backwards

Note: When any one of these commands is issued to the Mudd Rover, the Rover will continue performing this command until it is issued another. There is no such thing as turn for this amount or go forward for this amount.

3.5.2.3.Serial Driver

3.5.2.3.1. Overview

The serial driver takes an array of characters (opcode with arguments) with an arbitrary length, applies the message header, calculates the pertinent complements and checksums, and finally sends the completed message.

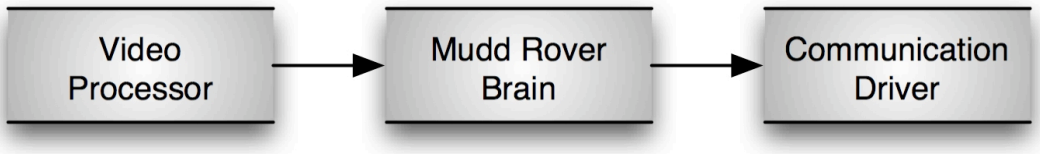
3.5.2.3.2. Implementation

The serial driver was implemented as a single function send_msg(char**). First, it steps through the 3-character header and writes each character out to serial using XUartLite_SendByte (XPAR_MYUART_BASEADDR, char). Once the header has been sent, in then steps through each character of the char array that was passed to it. Each step along the way, it sends the character in the array and then its complement, and adds to the checksum. Once the array has been exhausted, the checksum is then sent, followed by its complement.

4. Brains of Mudd Rover

4.1. Overview

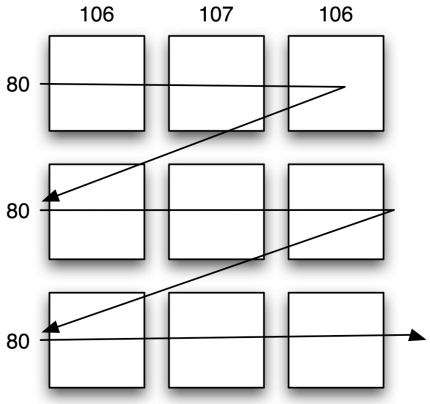
Now that the hardware and software groundwork for both the input and output of the project has been laid out it is now appropriate to talk about the Brains of Mudd Rover. The Brain component of Mudd Rover will act as a mediator between the input and the output of the project. It will take the information given to it from LightFinder, decide the appropriate actions to be taken, and then send out commands to the Mudd Rover.



4.2. Implementation

There are two cases that the robot will encounter; Never seen the line and seen the line. The first case is fairly trivial, if it has never seen the line then it will repeatedly go forward and then turn right until it finds something that it thinks is the line.

The second case is a little more involved. Each from is divided up into 9 regions(3x3) as shown below:



The algorithm runs through each row and find the section in each row with the highest number of pixels (if greater than some threshold and records it. Once done, it then looks at each row individually to figure out where to move. First it looks at the top row. If there are no sections with enough pixels, it moves on to the next row. If there is a section that has both enough pixels and the highest among the 3 columns a decision now must be made. If it's the left column, the robot will be commanded to turn left. If it's the right column, the robot will be commanded to turn right. If it's the middle column the robot will be commanded to go forward. After issuing this command the function returns and is not called again until the next frame.

There are two key issues to note. The first is the situation where no rows have a section with enough pixels to be considered for the comparison (not over the threshold). In this case, the robot is then told to consider where it moved last and once again move in that direction in hopes that it will find the line again. This brings us to the second issue

and that is the Rover remembering where it last saw the line. By remembering where it last saw the line, it can now keep moving in the right direction until it finds it again. For example, if it last saw the line in the bottom right hand corner of the frame, it will repeatedly turn right until it again sees the line and from there it will act accordingly.

5. Testing and Debugging

5.1. Video Hardware

For hardware debugging, we followed two basic methods. In order to ensure that the hardware was doing what was intended at any given moment, we bypassed the video stream from the Philips chip and we sent in specific streams for each line. This was done by disconnecting the Philips input in Marcio's video_decoder_intf.vhd and connecting the output to a switch statement based on the pixel counter. So, for example, we said if pixel counter is greater than 30, output 1, and when it reached 60, we had the decoder output 0. By this method we created a block of black pixels from 30 to 60 to test if the hardware was recognizing it properly.

Once we were sure that we were reading the input properly, we printed out the actual frame data through minicom to give ourselves a sense of the actual input the camera was seeing, allowing us to tailor the hardware towards actual luminance levels.

5.2. Video Software

We tested the video software using print statements in minicom. To do so we printed the pixel count for each of the 9 sections of the screen while using a simple input from the camera, such as a line entirely on the left, right, etc. We then roughly checked to see if the counts matched the number of pixels on the screen.

5.3. Communications

It was extremely hard to debug this part of the project due to the lack of minicom nor any real visual feedback from the Mudd Rover, at least until it actually started to move. The first step was to send a ping (is alive) command to the RCX and see if a little icon lights up on its display to show that it received it. The next step was to see if a full message could be sent with an opcode and arguments. The easiest way to test this was to use the RCX's ability to play music. Once that was achieved, a fully working serial driver was almost finished save the problems with checksums and complements that were eventually ironed out through trial and error along with brute force.

5.4. Mudd Rover Brain

The biggest obstacle to testing and debugging the line following algorithm was to try and see what the Mudd Rover was actually seeing. For this we had to display a good amount of debug data on the computer monitor. This data included the live feed straight from the camera, the converted video with just the longest lines of pixels and then finally the screen divided up into our grid showing us which sections were 'on'. Once we had this debug display up and running, it was much easier to diagnose the problems with the line following algorithm.

In terms of testing the algorithm, we set up a simple oval track at first and then slowly progressed to more complicated curves and lines at a larger scale. With each passing iteration, we built on what we learned from the past one. It was very much an iterative and evolutionary testing and implementation process.

6. Mudd Rover Environment

The Mudd Rover's overall environment consists of a large sheet of white paper with curves marked in thick black marker. The thickness of the line was determined by the position of the camera with respect to the floor and how much of the frame we deemed necessary to be filled in order to get satisfactory data.

While we are using the IR for wireless communication the range of the IR is not as spectacular as we had hoped. In reality the range of the IR tower is no greater than a circle with a 3-foot radius under the proper lighting conditions (dim lighting not consisting of fluorescent lamps or sunlight shinning directly on the area). These conditions, which are desired for optimal IR signaling, are the complete opposite of what is needed for good video quality. As a compromise we must hand hold the IR tower and point it in the proper direction to get the proper range. The final course for the demo is approximately 12-18 square feet as a result.

7. Lessons Learned

7.1. Ron Coleman

I learned to program in C. Also, I learned the importance of object-oriented programming. Our interface still worked, even with changes in the video and control. I learned during this project, as well as all group projects, that working well in a group is important. All members have to pull their own weight and learning to adapt and compensate for others is an important attribute. I now have a more extensive knowledge of the intricacies of the XiLinx FPGA and the XSB-300E board.

7.2. Akshay Kumar

Working on a group project for an extended period made me appreciate the importance of good planning, organization and teamwork. It became very clear early on that communication was going to be critical to the success of the project. The role of the project manager was also crucial to the successful completion of the project within the specified timeline. The project manger was able to delegate responsibility in a manner that divided the project into individual parts that could be worked on concurrently. The interfaces between the parts were cleanly defined which made it very easy to integrate all the pieces. When I got carried away with unrealistic goals in light of the allotted resources, the team was able to propose better and more realistic alternatives in a timely and productive fashion. The fact that I had very competent teammates that I could rely on was the backbone of this project. This helped keep the project moving forward.

7.3. Athena Ledakis

I learned how to read and make sense of DATA Sheets, and to actually implement the information I gained from them. Prior to this class and project I had only read them, for the purpose of trying to understand hardware, not actually doing anything with it. Getting familiar with data sheets and how to read pin layout diagrams is something I would suggest for future classes to do early. They can be dense and information can hard to get out of them if you don't know where to look.

And of course this course and project led to many lessons about working in teams: the importance of communication and the willingness to accept others ideas. Without this we could not have completed our project, but by proper division of our work load, and enough communication between the separate groups, we put together all the pieces seamlessly.

7.4. Josef Bryks Schenker

The most important thing I noticed was the importance of timing diagrams when working with hardware. Every single signal and flag is inherently tied to the other signals around it and raising or dropping a single wire at the wrong time could easily make the difference between the entire module working fine or doing everything completely wrong. Similarly, this project reinforced the lesson that has been drummed in our heads all semester that VHDL is NOT a programming language, but a tool to describe an actual

physical piece of hardware. It taught me to think of every process in terms of state machines and signals instead of algorithms and conditional statements. Ultimately, of course, this translates back to preparation. Before building any system we need to devote twice as much time to the design process. By carefully considering every case and possibility, by first drawing out the state diagrams and checking the appropriate timing constraints, we can cut down on 99% of the debugging issues we encountered with hardware.

7.5. Justin Titi

As team leader for this project, the two biggest lessons that I learned were the value of good communication and segmentation of the group to increase performance. With a group of 5 people, it was extremely hard to keep everyone together and informed. I learned that constant emails were the best route to take. In addition to that, weekly meetings also were extremely important to keeping the group working well. In addition I found that breaking the group into sub groups enhanced the overall productivity of the group and eliminated many of the scheduling conflicts that ensued with larger group meetings.

As a member of the team, the biggest lesson that I learned was to keep it simple. With a project of this magnitude consisting of so many variables that can not be controlled (See the environment section) it was extremely important to keep things simple in order to squash all of the bugs or even at times just to get something working.

8. Challenges

8.1. Overall

Taking on a project that made use of 2 peripherals and a FPGA board that we had very little experience with was a challenge in itself. Throw in all of the weird happenings with Legos breaking off, gears grinding, sunlight affecting IR, and a cheap video camera, this project provided a challenge to finish, not to mention to keep sane.

8.2. Video Hardware

There were several major issues we encountered when dealing with the video. First was the Philips chip itself. Lacking an appropriate knowledge of the registers and which values produce the proper results, members from several groups joined up to decipher the manual. This alone took several weeks and held up a significant portion of the project. Additionally, a mistake I made was waiting for the Philips chip to work before writing hardware to process it. Had I performed both actions somewhat simultaneously then I could have finished my part of the project earlier, and possibly spent more time improving rather than troubleshooting.

One issue with the video itself was the timing. Because the camera needs to ensure that no data gets lost, there are multiple layers of 'flags' letting the user know when the data is valid. This meant that we had to constantly ensure that we had all the correct flags activated when necessary. Also it made it more difficult to create registers to hold the longest block's length and position. In many early models the hardware would return the length and position of the first or last block only instead of the longest block. Additionally, creating a simple filter to ensure that a single 'light' (above-threshold pixel) wouldn't break our block proved to be quite a challenge. We ultimately solved it by delaying every pixel by one cycle and running a mask over the stream.

8.3. Video Software

The biggest obstacles encountered in this portion of the project dealt with integration of data from the video hardware and determining what type of information would be most useful for the `movement()` function to use. Originally we wanted to read the incoming information from a set of registers where each register held information from one line of video. This proved unsuccessful due to timing issues; the video would have been writing faster than what the software could read and store. It also proved to be inefficient for the video hardware to use so many memory locations. The next approach was to store information in two registers, one that held the starting position of the line and the other holding the length of the line with each register being read consecutively in a way similar to how we implemented reading data into the `line_array[]` as detailed above. Because reads and writes are expensive, this data was concatenated into one register so that only one read of memory was required. A second read would have been required if we stored data in two registers. This method was quickly replaced by quick use of bitwise operators and one register.

The next challenge was how we were going to analyze the data. We originally planned to find the average center point of the line per frame. This involved taking an

average pixel position of each line, then taking an average position of each screen section, and then finally the whole frame. This would give the `movement()` function one point from which to decide where to direct the rover, which we were quickly able to see was not sufficient information. This was overcome by keeping track of average positions for all sections of a frame, where a section is a horizontal row of the frame, and creating a vector to predict what direction the line is in. This algorithm required too much CPU processing time, which slowed down the amount of usable data the `count_pixel()` function was able to retrieve, and so we had to find an algorithm that was less intensive on the CPU. As we realized that the direction of the line was an important piece of information for the movement control portion of this project to use, we split the screen into the three vertical sections, left, center and right as mentioned above and utilized the algorithm that was specified.

8.4. Serial Communication

This was an extremely frustrating and tedious part of the project as the TA's were not well versed with regards to the operations of the Lego RCX and its accompanying tower, nor was there a completely thorough resource on the internet that explained all of this. Even when the serial driver was thought to be in working order, it in fact didn't work because the original IR tower given to us was not operational. After weathering the delay of the arrival of a new IR tower we found that the our serial driver did not work properly due to some small intricacies in the Lego hardware that needed to be ironed out: Proper firmare on the RCX to receive remote commands, checksums for only the data bits, timing issues for the RCX as it can only process messages so fast, and finally the repeat opcode situation when we finally figured out about toggling the 0x08 bit.

8.5. Mudd Rover Brains

As stated in our original design doc, this part was extremely evolutionary as the project moved forward. This was the last part of the project that was implemented because we had no idea how the RCX and IR would perform. Once we got an idea of the performance we quickly tried a simple algorithm to follow the line and it worked. However, with each successive attempt at trying to make it more robust and complicated, it failed. This was very frustrating. It was a great challenge to keep this part of the project as simple as possible while still getting the Mudd Rover to perform as we wanted.

9. Advice for Future Projects

Most project classes are taken within a certain major and do not combine with other majors. This is not the case with this class. As a result, teams must be formed as they would in the real world where positions are filled with people who are qualified for them. Essentially, this means if you have a heavily software based project, you shouldn't have a team full of Electrical Engineers and the same goes for its inverse. It is extremely important to pick the right mix of Computer Engineers, Computer Scientists, and Electrical Engineers so that no one in the group is stuck working on something they are not comfortable with.

After the team is formed, it is extremely important to meet early and often as a full team to completely discern the goal of the project. This can be extremely hard to achieve when dealing with 5 people with 5 different schedules. Once the overall project idea has been agreed on, the group should then break into subgroups to work on the project as this makes it easier to meet with each other. However, it is important not to forget about full team meetings as it keeps everyone up to date and the whole project in perspective. The final piece of advice we would like to share is to make use of your TA. We were extremely fortunate to have Cristian as the TA for the class and he was an amazing asset. Make sure you seek him out, he is always more than willing help!

10. Responsibilities

- Ron Coleman-Video Software / Testing
- Akshay Kumar-Serial Communication / Mudd Rover Brain / Testing
- Athena Ledakis-Video Software / Testing
- Josef Bryks Schenker-Video Hardware / Testing
- Justin Titi-Team Leader / Serial Communication / Mudd Rover Brain / Testing

11. Code

Opb_videodec.vhd – Marcio's module. Connects the video interfaces with the OPB bus.

Modifications: Added videoreader as a component.
Changed the multiplexer so read_data will take signal from VideoReader.vhd when appropriate.

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


entity opb_videodec is
  generic (
    C_OPB_AWIDTH : integer := 32;
    C_OPB_DWIDTH : integer := 32;
    C_BASEADDR   : std_logic_vector := X"0180_0000";   -- 512 positions of 32
    C_HIGHADDR   : std_logic_vector := X"0180_3FFF");  -- bits plus extra room.
                                                       -- Each 32 bits in the
                                                       -- block RAMs stores 4
                                                       -- pixels' luminance

  port (
    -- Global signals
    OPB_Clk : in std_logic;
    OPB_Rst : in std_logic;

    -- OPB signals
    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;

    -- Slave signals
    VIDEC_DBus    : out std_logic_vector (31 downto 0);
    VIDEC_errAck  : out std_logic;
    VIDEC_retry   : out std_logic;
    VIDEC_toutSup : out std_logic;
    VIDEC_xferAck : out std_logic;

    -- Coming from SAA7114H
    IPort       : in std_logic_vector (7 downto 0);
    HPort       : in std_logic_vector (7 downto 0);
    IDQ         : in std_logic;
    ICLK        : in std_logic;
    IPGV        : in std_logic;
    IPGH        : in std_logic;
    ITRI        : out std_logic;
    ITRDY       : out std_logic
  );
end opb_videodec;

architecture structural of opb_videodec is

-- Buffered version of the signals
-- with the same name in the entity
signal buf_iclk : std_logic;
signal buf_ipgh : std_logic;
signal buf_ipgv : std_logic;
signal buf_idq : std_logic;
signal buf_iport : std_logic_vector (7 downto 0);
signal buf_hport : std_logic_vector (7 downto 0);
signal buf_itri : std_logic;
signal buf_itrdy : std_logic;

-- Latched versions of the above buffered signals
signal latched_ipgh  : std_logic;
signal latched_ipgv  : std_logic;
```

```
signal latched_idq   : std_logic;
signal latched_iport : std_logic_vector (7 downto 0);
signal latched_hport : std_logic_vector (7 downto 0);

-- Signals used when reading from block
-- ram and filling status register
signal cs : std_logic;
signal ce : std_logic;
signal rnw : std_logic;
signal xfer : std_logic;

-- raddr(8 downto 0) is used to address the
-- block RAM. OPB_ABus(13) and OPB_ABus(12), which
-- correspond to raddr(11) and raddr(10), are
-- used to address the filling status register
signal raddr : std_logic_vector (11 downto 0);

-- Signals used by the filling level status
-- The video decoder interface sends a set
-- of signals indicating how much of the
-- current line it has already written into
-- the block RAMs (1/4, 1/2, 3/4 and 1)
-- Microblaze keeps polling this signal
signal filling_level : std_logic_vector(3 downto 0);

-- Count the number of lines being written by the video decoder
signal line_counter : std_logic_vector(15 downto 0);

-- Count the frame (Actually, it's the frame ID
signal frame_counter : std_logic_vector(1 downto 0);

-- Data coming from video decoder interface
signal data_from_decoder : std_logic_vector(15 downto 0);

-- Data bus and latched data bus
signal data_from_bram : std_logic_vector (31 downto 0);
signal data_bus_ce : std_logic_vector (31 downto 0);

-- Signals for the block ram state machine
signal q2, q1, q0 : std_logic;

-- Coming from video_decoder_intf, going to block_ram
signal intf_idq_out  : std_logic;
signal intf_iclk_out : std_logic;
signal waddr : std_logic_vector (10 downto 0);
signal luma_data : std_logic_vector (7 downto 0);

-- Count pixels
signal pix_count : std_logic_vector(10 downto 0);
signal active : std_logic;
signal position : std_logic_vector(15 downto 0);
signal length : std_logic_vector(15 downto 0);
signal strength : std_logic_vector(17 downto 0);
signal threshold : std_logic_vector(7 downto 0);

-- Dummy signals. Reserved for future enhancements
-- We currently not write from microblze (XIo_Out)
signal wdata : std_logic_vector (31 downto 0);
signal be    : std_logic_vector (3 downto 0);

component block_ram is
  port (
    waddr       : in std_logic_vector (10 downto 0);
    data_in     : in std_logic_vector (7 downto 0);
    raddr       : in std_logic_vector (8 downto 0);
    data_out    : out std_logic_vector (31 downto 0);
    idq         : in std_logic;
    iclk        : in std_logic;
    ipgh        : in std_logic;
    clock       : in std_logic;
    read_enable : in std_logic;
    reset       : in std_logic
  );
```

```
end component;

component video_decoder_intf is
  port (
    iport        : in std_logic_vector (7 downto 0);
    hport        : in std_logic_vector (7 downto 0);
    idq_in       : in std_logic;
    iclk_in      : in std_logic;
    ipgh         : in std_logic;
    ipgv         : in std_logic;
    data         : out std_logic_vector (15 downto 0);
    waddr        : out std_logic_vector (10 downto 0);
    idq_out      : out std_logic;
    iclk_out     : out std_logic;
    fil_level    : out std_logic_vector(3 downto 0);
    line_count   : out std_logic_vector(15 downto 0);
    frame_id     : out std_logic_vector(1 downto 0);
    pixel_counter: out std_logic_vector (10 downto 0);
    active_out   : out std_logic;
    reset        : in std_logic
  );
end component;

component videoreader is
 port
   (
     IPD           : in    std_logic_vector(7 downto 0);
     ICLK          : in    std_logic;
     IDQ           : in    std_logic;
     IGPH          : in    std_logic;
     IGPV          : in    std_logic;
     pix_count     : in  std_logic_vector(10 downto 0);
     active        : in  std_logic;
     frame_counter : in std_logic_vector(1 downto 0);
     position      : out   std_logic_vector(15 downto 0);
     length        : out   std_logic_vector(15 downto 0);
     strength      : out std_logic_vector(17 downto 0);
     threshold     : in std_logic_vector(7 downto 0)
     );
end component;

component IBUFG is
  port (
    I : in  std_logic;
    O : out std_logic);
end component;

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

component OBUF
  port(
    O:  out std_ulogic;
    I:  in  std_ulogic
    );
end component;

component FD
  port (
    C : 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 FD : component is "true";
```

```
begin

itrdy_buf : OBUF port map  (
  O => ITRDY,
  I => buf_itrdy
);

itri_buf : OBUF port map  (
  O => ITRI,
  I => buf_itri
);

vbuf : IBUFG port map (
  I => ICLK,
  O => buf_iclk
);

ipgh_pinbuf : IBUF port map (
  I => IPGH,
  O => buf_ipgh
);

ipgh_pinlatch : FD port map (
  C => buf_iclk,
  D => buf_ipgh,
  Q => latched_ipgh
);

ipgv_pinbuf : IBUF port map (
  I => IPGV,
  O => buf_ipgv
);

ipgv_pinlatch : FD port map (
  C => buf_iclk,
  D => buf_ipgv,
  Q => latched_ipgv
);

idq_pinbuf : IBUF port map (
  I => IDQ,
  O => buf_idq
);

idq_pinlatch : FD port map (
  C => buf_iclk,
  D => buf_idq,
  Q => latched_idq
);

databus : for i in 0 to 7 generate
  I_data_pad : IBUF port map (
      I => IPORT(i),
      O => buf_iport(i));

  I_data_ff : FD port map (
      C => buf_iclk,
      D => buf_iport(i),
      Q => latched_iport(i));

  H_data_pad : IBUF port map (
      I => HPORT(i),
      O => buf_hport (i));

  H_data_ff : FD port map (
      C => buf_iclk,
      D => buf_hport(i),
      Q => latched_hport(i));
end generate;

u1 : block_ram
port map
(
```

```
    waddr => waddr,
    data_in => luma_data,
    raddr => raddr(8 downto 0),
    data_out => data_from_bram,
    idq => intf_idq_out,
    iclk => intf_iclk_out,
    ipgh => latched_ipgh,
    clock => OPB_Clk,
    read_enable => '1',
    reset => OPB_Rst
);

u2 : video_decoder_intf
port map (
    iport => latched_iport,
    hport => latched_hport,
    idq_in => latched_idq,
    iclk_in => buf_iclk, -- For tests, use OPB_Clk
    ipgh => latched_ipgh,
    ipgv => latched_ipgv,
    data => data_from_decoder,
    waddr => waddr,
    idq_out  => intf_idq_out,
    iclk_out => intf_iclk_out,
    fil_level => filling_level,
    line_count => line_counter,
    frame_id => frame_counter,
    pixel_counter => pix_count,
    active_out => active,
    reset => OPB_Rst
);

u3: videoreader
    port map (
      IPD   => luma_data,
      ICLK  => buf_iclk,
      IDQ   => latched_idq,
      IGPH  => latched_ipgh,
      IGPV  => latched_ipgv,
      pix_count => pix_count,
      active => active,
      frame_counter => frame_counter,
      position => position,
      length => length,
      strength => strength,
      threshold => threshold
 );

-- Chip select for block RAM - port A of block RAMs is memory mapped
-- The binary number is X"0180" concatenated with binary "00"
cs <= OPB_select when OPB_ABus(31 downto 14) = "00000011000000000" else '0';


-- Latching read address. Used to address port A of block RAMs
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_RST = '1' then
      raddr <= "000000000000";
    else
      raddr <= OPB_ABus(13 downto 2);
    end if;
  end if;
end process;


-- Latching RNW signal
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;


-- Latching BE signal (byte enable). Dummy signal
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;


-- The following process is dummy. It is used to
-- create a mux between this entity and OPB_DBus
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk = '1' then
    if OPB_Rst = '1' then
      wdata <= X"0000_0000";
    else
      wdata <= OPB_DBus;
    end if;
  end if;
end process;

process(OPB_Clk)
  begin
     if OPB_Clk'event and OPB_Clk='1' then
       if q2='1' and q1='0' and rnw='0' and raddr(11)='1' and raddr(10)='0' then
         threshold <= wdata(7 downto 0);
       end if;
     end if;
  end process;

-- State machine for reading the block RAM
process (OPB_Clk)
begin
  if OPB_Clk'event and OPB_Clk='1' then
    q2 <= (not q2 and q1) or (q2 and not q1);
    q1 <= (cs and not q2 and not q1) or (q2 and not q1);
    q0 <= q2 and not q1;
  end if;
end process;


-- CE is data latch enable
ce <= q2 and not q1 and rnw;

-- Latch the data coming from the block RAM
-- or from the filling status register
-- at address 01803FFC
process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst='1' then
    data_bus_ce <= X"00000000";
  elsif OPB_Clk'event and OPB_Clk='1' then
    if ce='1' then
      if raddr(11)='1' and raddr(10)='1' then
        data_bus_ce <= "00000000000000000000000000" & latched_ipgv & frame_counter(0) &
filling_level;
      elsif raddr(11)='1' and raddr(10)='0' then
        data_bus_ce <= "01010000000000" & strength;
      elsif raddr(11)='0' and raddr(10)='1' then
        data_bus_ce <= X"0000" & line_counter;
      elsif raddr(11)='0' and raddr(10)='0' and raddr(9)='1' then
        data_bus_ce <= position & length;
```

```
      else
        data_bus_ce <= data_from_bram;
      end if;
    else
      data_bus_ce <= X"00000000";
    end if;
  end if;
end process;

-- Connect luma bits from video decoder interface to
-- block RAMs input data bus
luma_data <= data_from_decoder(15 downto 8);

-- XFER is transfer acknowledge
xfer <= q0;

-- Slave data bus
VIDEC_DBus(31 downto 0) <= data_bus_ce;

-- Tie unused signals to zero
VIDEC_errAck <= '0';
VIDEC_retry <= '0';
VIDEC_toutSup <= '0';

VIDEC_xferAck <= xfer;

buf_itri <= '1';
buf_itrdy <= '1';

end structural;
```

Videoreader.vhd – Original code. Used to find longest block of dark pixels; place as a component on opb_videodec.

Ports:

Inputs:	Threshold	inputs luminance threshold from control program
	IPD	video data from Philips chip
	ICLK	clock signal from Philips chip
	IGPH	horizontal sync signal from Philips chip
	IGPV	vertical sync signal from Philips chip
	Pix_count	pixel counter from video_decoder_intf
	Active	Active line flag from video_decoder_intf
	Frame_counter	frame counter from video_decoder_intf
Output:	Position	start position of longest block (16-bit)
	Length	length of longest block (16-bit)

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

-- entity
-------------------------------------------------------------------------------

entity videoreader is  --USER--

  port
  (
    IPD         : in    std_logic_vector(7 downto 0);
    ICLK        : in    std_logic;
    IDQ         : in    std_logic;
    IGPH        : in    std_logic;
    IGPV        : in    std_logic;

    pix_count   : in  std_logic_vector(10 downto 0);
    active       : in  std_logic;
    frame_counter : in std_logic_vector(1 downto 0);

    position      : out   std_logic_vector(15 downto 0);
    length        : out   std_logic_vector(15 downto 0);

    strength : out std_logic_vector(17 downto 0);
    threshold : in std_logic_vector(7 downto 0)

    );

end entity videoreader; --USER--

-------------------------------------------------------------------------------
-- architecture
-------------------------------------------------------------------------------

architecture imp of videoreader is --USER--

signal active_block_flag        : std_logic;  -- place pixels in registers
signal counter_valid            : std_logic_vector(1 downto 0);
signal temp_position            : std_logic_vector(15 downto 0) := "0000000000000000";
signal last_position            : std_logic_vector(15 downto 0) := "0000000000000000";
signal hold_position            : std_logic :='0';
signal temp_length              : std_logic_vector(15 downto 0) := "0000000000000000";
signal hold_length              : std_logic_vector(15 downto 0) := "0000000000000000";
signal last_length              : std_logic_vector(15 downto 0) := "0000000000000000";
signal reg_counter              : std_logic_vector(2 downto 0);
signal check_0                  : std_logic := '0';
signal check_1                  : std_logic := '0';
signal vid_reg0                 : std_logic_vector(7 downto 0) := "00000000";
signal vid_reg1                 : std_logic_vector(7 downto 0) := "00000000";
```

```
signal vid_reg2                : std_logic_vector(7 downto 0) := "00000000";
signal vid_reg3                : std_logic_vector(7 downto 0) := "00000000";
signal vid_reg4                : std_logic_vector(7 downto 0) := "00000000";
signal vid_reg5                : std_logic_vector(7 downto 0) := "00000000";
signal vid_reg6                : std_logic_vector(7 downto 0) := "00000000";
signal vid_reg7                : std_logic_vector(7 downto 0) := "00000000";
signal compare_reg             : std_logic;
signal compare_length          : std_logic;
signal current_state           : std_logic_vector(3 downto 0) := "1110";
signal next_state              : std_logic_vector(3 downto 0);
signal state_reset             : std_logic := '0';
signal dont_write              : std_logic := '0';
signal filter_safe             : std_logic := '0';
signal fstream                 : std_logic := '0';
signal region1_pix             : std_logic_vector(7 downto 0) := "00000000";
signal region2_pix             : std_logic_vector(7 downto 0) := "00000000";
signal region3_pix             : std_logic_vector(7 downto 0) := "00000000";

signal t_strength : std_logic_vector(17 downto 0);



--constant threshold             : std_logic_vector(7 downto 0) := "01010000";
--constant stateA                : std_logic_vector(3 downto 0) := "0000";
constant stateB                : std_logic_vector(3 downto 0) := "0001";
constant stateC                : std_logic_vector(3 downto 0) := "0010";
constant stateD                : std_logic_vector(3 downto 0) := "0011";
constant stateE                : std_logic_vector(3 downto 0) := "0100";
constant stateF                : std_logic_vector(3 downto 0) := "0101";
constant stateG                : std_logic_vector(3 downto 0) := "0110";
constant stateH                : std_logic_vector(3 downto 0) := "0111";
constant stateI                : std_logic_vector(3 downto 0) := "1000";
constant stateValid            : std_logic_vector(3 downto 0) := "1101";
constant stateInvalid          : std_logic_vector(3 downto 0) := "1110";


begin


-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- This is an attempt at a filter. This filter will examine the second bit in a
-- series of three and allow it to 'pass' if the two bits around it are 'on.'
-------------------------------------------------------------------------------
--first set up a minibuffer
  process (ICLK, IDQ)
  begin
    if ICLK'event and ICLK = '1' and IDQ = '1' then
      if active = '1' then
        vid_reg0 <= IPD;
        vid_reg1 <= vid_reg0;
        vid_reg2 <= vid_reg3;
      elsif active = '0' then
        vid_reg0 <= "00000000";
        vid_reg1 <= "00000000";
        vid_reg2 <= "00000000";
      end if;
    end if;
  end process;

 -- Continually running statement that turns on the middle pixel if surrounded
 -- by two on pixels
  fstream <= '1' and IDQ and active when (vid_reg1 < threshold or (vid_reg0 < threshold
and vid_reg2 < threshold))
             else '0';
-------------------------------------------------------------------------------
  compare_reg <= fstream;
--  compare_reg <= '1' and IDQ and active when IPD < threshold else '0';
  compare_length <= '1' when temp_length > last_length else '0';
-------------------------------------------------------------------------------

-- active_block_flag indicates whether we are in the middle of a region that is
-- entirely below the threshold
```

```
  process (ICLK)
  begin
    if ICLK'event and ICLK='1' and IDQ = '1' then
      if active_block_flag = '0' then
        if compare_reg = '1' then
          active_block_flag <= '1';
        end if;
      elsif active_block_flag = '1' then
        if compare_reg = '0' then
          active_block_flag <= '0';
        end if;
      end if;
      if active = '0' then
        active_block_flag <= '0';
      end if;
    end if;
  end process;

--These next few processes control the length output signal
  process(ICLK)
  begin
     if ICLK'event and ICLK='1' and IDQ = '1' then
       if active = '0' then
         t_strength <= X"0000" &"00";
       else
         t_strength <= t_strength + IPD;
       end if;
     end if;
  end process;

  process(ICLK)
    begin
      if ICLK'event and ICLK='1' and IDQ = '1' then
        if active = '1' then
          strength <= t_strength;
        end if;
      end if;
    end process;


-- increment last length appropriately

  process (ICLK)
  begin
    if ICLK'event and ICLK = '1' and IDQ='1' then
      if compare_reg = '1' then
        last_length <= last_length+1;
      elsif active_block_flag = '0' then
        last_length <= X"0000";
      end if;
      if active = '0' then
        last_length <= X"0000";
      end if;
    end if;
  end process;

-- assign last length to temp length if last length is longer
  process (ICLK)
  begin
    if ICLK'event and ICLK = '1' and IDQ = '1' then
      if compare_length = '0' then
        temp_length <= last_length;
      end if;
      if active = '0' then
        temp_length <= X"0000";
      end if;
    end if;
  end process;

-- output the length signal to opb_bus
  process (ICLK)
  begin
    if ICLK'event and ICLK = '1' and IDQ = '1' then
```

```
      if active = '1' then
        length <= '0' & temp_length(15 downto 1);
--        length <= temp_length;
      end if;
    end if;
  end process;

-- flags controlling the position signal

-- keep updating to last position before active block and then holds that value
-- once we start looking at active block



  process (ICLK)
  begin
    if ICLK'event and ICLK = '1' and IDQ = '1' then
      if active_block_flag = '0' and compare_reg = '0' then
        last_position <= "00000" & pix_count;
      end if;
    end if;
  end process;

-- if the block we are looking at now is longer than any previous block then
-- assign the starting position of the current block (last position) to temp
  process (ICLK)
  begin
    if ICLK'event and ICLK = '1' and IDQ = '1' then
      if compare_length = '0' then
        temp_position <= last_position;
      end if;
      if active = '0' then
        temp_position <= X"0000";
      end if;
    end if;
  end process;

-- output position to opb_bus
   process (ICLK)
   begin
     if ICLK'event and ICLK = '1' then
       if active='1' then
         position <= '0' & temp_position(15 downto 1);
--         position <= temp_position;
       end if;
     end if;
   end process;


end architecture imp;
```

Block_ram.vhd

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

-- Four RAMB4_S8_S8 components instantiated.
-- Each one stores 8 bits of information (luma)
-- on each memory cell. Block 0 stores pixels
-- 0,4,8, etc. Block 1 stores pixels 1, 5, 9, etc,
-- Block 2 stores pixels 2, 6, 10, etc. and
-- Block 3 stores pixels 3, 7, 11, etc.
-- and so on.
entity block_ram is
  port (
    -- Address generated by video decoder intf module
    -- (video_decoder_intf.vhd). All block-RAMs see
    -- the same 9 *upper* bits. The remaining 2 *lower*
    -- bits are used to choose which block to store.
    waddr : in std_logic_vector (10 downto 0);

    -- Luminance data coming from the video decoder
    -- The video decoder is actually being configured
    -- to transmit 16-bit data (upper bits are luma,
    -- lower bits are chroma). However, the chroma
    -- bits are just being disconsidered as of now.
    data_in : in std_logic_vector (7 downto 0);

    -- Read address. Generated by microblaze every
    -- time one executes XIO_In32. Microblaze reads
    -- four pixels at a time: pixel "i" from block
    -- 0, pixel "i+1" from block 1, pixel "i+2"
    -- from block 2 and pixel "i+3" from block 3.
    -- That's why the *lower* bits of addr are used.
    raddr : in std_logic_vector (8 downto 0);

    -- Data going to microblaze. The 32 bits read
    -- correspond to 4 pixels, each one coming
    -- from a specific block RAM.
    data_out : out std_logic_vector (31 downto 0);

    -- IDQ is '1' when valid data is
    -- coming from video decoder
    idq : in std_logic;

    -- clock for port B is ICLK
    -- from video decoder
    iclk : in std_logic;

    -- From the video decoder
    ipgh : in std_logic;

    -- clock for port A is
    -- clk from CPU
    clock : in std_logic;

    -- Read enable
    read_enable : in std_logic;

    -- Reset
    reset : in std_logic
  );
end block_ram;

architecture structural of block_ram is

-- Dual-port block RAM used for storing data coming from video decoder
-- Port B is written by the video decoder intf, Port A is read by CPU.
-- See  "http://www.xilinx.com/bvdocs/appnotes/xapp173.pdf"
component RAMB4_S8_S8
  generic (
    INIT_00, INIT_01, INIT_02, INIT_03, INIT_04, INIT_05,
    INIT_06, INIT_07, INIT_08, INIT_09, INIT_0a, INIT_0b,
```

```
    INIT_0c, INIT_0d, INIT_0e, INIT_0f: bit_vector(255 downto 0)
    :=X"0000000000000000000000000000000000000000000000000000000000000000"
  );
  port  (
    DIA,DIB    : in STD_LOGIC_VECTOR (7 downto 0);
    ENA,ENB    : in STD_logic;
    WEA,WEB    : in STD_logic;
    RSTA,RSTB  : in STD_logic;
    CLKA,CLKB  : in STD_logic;
    ADDRA,ADDRB : in STD_LOGIC_VECTOR (8 downto 0);
    DOA,DOB    : out STD_LOGIC_VECTOR (7 downto 0)
  );
end component;

-- i_clock is ICLK from video decoder
-- opb_clock is opb_clk from OPB bus
signal i_clock : std_logic;
signal opb_clock : std_logic;

-- Read enable
signal r_en : std_logic;

-- Reset
signal rst : std_logic;

-- Shared address bus for all 4 block RAMs
signal addr_a : std_logic_vector (8 downto 0);
signal addr_b : std_logic_vector (8 downto 0);

-- Enable signals for distinct blocks
signal enb0, enb1, enb2, enb3 : std_logic;

-- Data coming from video decoder interface to B ports
signal data_in_signal : std_logic_vector (7 downto 0);

-- Data going to OPB Bus from A ports
signal data_out_a0 : std_logic_vector (7 downto 0);
signal data_out_a1 : std_logic_vector (7 downto 0);
signal data_out_a2 : std_logic_vector (7 downto 0);
signal data_out_a3 : std_logic_vector (7 downto 0);


begin

block_0: RAMB4_S8_S8   -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb0,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a0, DOB => open
);


block_1: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb1,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a1, DOB => open
);


block_2: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
```

```
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb2,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a2, DOB => open
);


block_3: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb3,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a3, DOB => open
);

-- Enable signals for each block for writing
enb0 <= idq and ipgh and not waddr(2) and not waddr(1) and not waddr(0);  -- "000" -> Y0
enb1 <= idq and ipgh and not waddr(2) and     waddr(1) and not waddr(0);  -- "010" -> Y2
enb2 <= idq and ipgh and     waddr(2) and not waddr(1) and not waddr(0);  -- "100" -> Y4
enb3 <= idq and ipgh and     waddr(2) and     waddr(1) and not waddr(0);  -- "110" -> Y6

-- Data out merger
data_out(31 downto 24) <= data_out_a0;
data_out(23 downto 16) <= data_out_a1;
data_out(15 downto 8)  <= data_out_a2;
data_out(7 downto 0)   <= data_out_a3;

-- Data in
data_in_signal <= data_in;


-- Actual bits addressing block RAMs, port A
addr_a <= raddr;

-- Actual bits addressing block RAMs, port B
addr_b <= "0" & waddr(10 downto 3);

-- Connect clocks and reset
i_clock <= iclk;
opb_clock <= clock;
rst <= reset;

-- Read enable
r_en <= read_enable;

end structural;
```

opb_i2ccontroller.vhd

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

-- Four RAMB4_S8_S8 components instantiated.
-- Each one stores 8 bits of information (luma)
-- on each memory cell. Block 0 stores pixels
-- 0,4,8, etc. Block 1 stores pixels 1, 5, 9, etc,
-- Block 2 stores pixels 2, 6, 10, etc. and
-- Block 3 stores pixels 3, 7, 11, etc.
-- and so on.
entity block_ram is
  port (
    -- Address generated by video decoder intf module
    -- (video_decoder_intf.vhd). All block-RAMs see
    -- the same 9 *upper* bits. The remaining 2 *lower*
    -- bits are used to choose which block to store.
    waddr : in std_logic_vector (10 downto 0);

    -- Luminance data coming from the video decoder
    -- The video decoder is actually being configured
    -- to transmit 16-bit data (upper bits are luma,
    -- lower bits are chroma). However, the chroma
    -- bits are just being disconsidered as of now.
    data_in : in std_logic_vector (7 downto 0);

    -- Read address. Generated by microblaze every
    -- time one executes XIO_In32. Microblaze reads
    -- four pixels at a time: pixel "i" from block
    -- 0, pixel "i+1" from block 1, pixel "i+2"
    -- from block 2 and pixel "i+3" from block 3.
    -- That's why the *lower* bits of addr are used.
    raddr : in std_logic_vector (8 downto 0);

    -- Data going to microblaze. The 32 bits read
    -- correspond to 4 pixels, each one coming
    -- from a specific block RAM.
    data_out : out std_logic_vector (31 downto 0);

    -- IDQ is '1' when valid data is
    -- coming from video decoder
    idq : in std_logic;

    -- clock for port B is ICLK
    -- from video decoder
    iclk : in std_logic;

    -- From the video decoder
    ipgh : in std_logic;

    -- clock for port A is
    -- clk from CPU
    clock : in std_logic;

    -- Read enable
    read_enable : in std_logic;

    -- Reset
    reset : in std_logic
  );
end block_ram;

architecture structural of block_ram is

-- Dual-port block RAM used for storing data coming from video decoder
-- Port B is written by the video decoder intf, Port A is read by CPU.
-- See  "http://www.xilinx.com/bvdocs/appnotes/xapp173.pdf"
component RAMB4_S8_S8
  generic (
    INIT_00, INIT_01, INIT_02, INIT_03, INIT_04, INIT_05,
    INIT_06, INIT_07, INIT_08, INIT_09, INIT_0a, INIT_0b,
```

```
    INIT_0c, INIT_0d, INIT_0e, INIT_0f: bit_vector(255 downto 0)
    :=X"0000000000000000000000000000000000000000000000000000000000000000"
  );
  port  (
    DIA,DIB    : in STD_LOGIC_VECTOR (7 downto 0);
    ENA,ENB    : in STD_logic;
    WEA,WEB    : in STD_logic;
    RSTA,RSTB  : in STD_logic;
    CLKA,CLKB  : in STD_logic;
    ADDRA,ADDRB : in STD_LOGIC_VECTOR (8 downto 0);
    DOA,DOB    : out STD_LOGIC_VECTOR (7 downto 0)
  );
end component;

-- i_clock is ICLK from video decoder
-- opb_clock is opb_clk from OPB bus
signal i_clock : std_logic;
signal opb_clock : std_logic;

-- Read enable
signal r_en : std_logic;

-- Reset
signal rst : std_logic;

-- Shared address bus for all 4 block RAMs
signal addr_a : std_logic_vector (8 downto 0);
signal addr_b : std_logic_vector (8 downto 0);

-- Enable signals for distinct blocks
signal enb0, enb1, enb2, enb3 : std_logic;

-- Data coming from video decoder interface to B ports
signal data_in_signal : std_logic_vector (7 downto 0);

-- Data going to OPB Bus from A ports
signal data_out_a0 : std_logic_vector (7 downto 0);
signal data_out_a1 : std_logic_vector (7 downto 0);
signal data_out_a2 : std_logic_vector (7 downto 0);
signal data_out_a3 : std_logic_vector (7 downto 0);


begin

block_0: RAMB4_S8_S8   -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb0,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a0, DOB => open
);


block_1: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb1,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a1, DOB => open
);


block_2: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
```

```
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb2,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a2, DOB => open
);


block_3: RAMB4_S8_S8 -- 512 words of 8 bits
port map
(
  DIA => X"00", DIB => data_in_signal,
  ENA => r_en, ENB => '1',
  WEA => '0', WEB => enb3,
  RSTA => rst, RSTB => rst,
  CLKA => opb_clock, CLKB => i_clock,
  ADDRA => addr_a, ADDRB => addr_b,
  DOA => data_out_a3, DOB => open
);

-- Enable signals for each block for writing
enb0 <= idq and ipgh and not waddr(2) and not waddr(1) and not waddr(0);  -- "000" -> Y0
enb1 <= idq and ipgh and not waddr(2) and     waddr(1) and not waddr(0);  -- "010" -> Y2
enb2 <= idq and ipgh and     waddr(2) and not waddr(1) and not waddr(0);  -- "100" -> Y4
enb3 <= idq and ipgh and     waddr(2) and     waddr(1) and not waddr(0);  -- "110" -> Y6

-- Data out merger
data_out(31 downto 24) <= data_out_a0;
data_out(23 downto 16) <= data_out_a1;
data_out(15 downto 8)  <= data_out_a2;
data_out(7 downto 0)   <= data_out_a3;

-- Data in
data_in_signal <= data_in;


-- Actual bits addressing block RAMs, port A
addr_a <= raddr;

-- Actual bits addressing block RAMs, port B
addr_b <= "0" & waddr(10 downto 3);

-- Connect clocks and reset
i_clock <= iclk;
opb_clock <= clock;
rst <= reset;

-- Read enable
r_en <= read_enable;

end structural;
```

video_decoder_intf.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity video_decoder_intf is
  port (
    iport        : in std_logic_vector (7 downto 0);
    hport        : in std_logic_vector (7 downto 0);
    idq_in       : in std_logic;
    iclk_in      : in std_logic;
    ipgh         : in std_logic;
    ipgv         : in std_logic;
    data         : out std_logic_vector (15 downto 0);
    waddr        : out std_logic_vector (10 downto 0);
    idq_out      : out std_logic;
    iclk_out     : out std_logic;
    fil_level    : out std_logic_vector(3 downto 0);
    line_count   : out std_logic_vector(15 downto 0);
    frame_id     : out std_logic_vector(1 downto 0);
    pixel_counter: out std_logic_vector (10 downto 0);
    active_out   : out std_logic;
    reset        : in std_logic
  );
end video_decoder_intf;

architecture structural of video_decoder_intf is

signal active     : std_logic;
signal pix_count  : std_logic_vector (10 downto 0);
signal pixel_addr : std_logic_vector(10 downto 0);

signal line_counter : std_logic_vector(15 downto 0);
signal frame_counter : std_logic_vector(1 downto 0);

-- The following signals indicate how much of the
-- line was already written into  the  block  RAM
signal one_fourth     : std_logic;
signal half_line      : std_logic;
signal three_quarters : std_logic;
signal entire_line    : std_logic;
signal filling_level  : std_logic_vector(3 downto 0);

-- Comment the following line
signal pixel_data : std_logic_vector(15 downto 0);                      ----------------
-----------
signal line_start : std_logic_vector(10 downto 0) := "00000000000";     ----------------
-----------Test
                                                                        ----------------
-----------data
                                                                        ----------------
-----------remove
                                                                        ----------------
-----------from file

begin

-- pixel address - where to store valid pixels in the block RAMs
process (iclk_in, reset)
begin
 if reset='1' then
    pixel_addr <= "00000000000";
 elsif iclk_in'event and iclk_in='1' then
   if ipgh='0' then
     pixel_addr <= "00000000000";
   elsif idq_in = '1' and active = '1' then
     pixel_addr <= pixel_addr + 1;
   end if;
 end if;
end process;
```

```
-- count the actual data coming from iport and hport.
-- Some data is control (FF, 00 , 00 , SAV business)
-- Reset the counter whenever ipgh is zero
process (iclk_in, reset)
begin
  if reset='1' then
    pix_count <= "00000000000";
  elsif iclk_in'event and iclk_in='1' then
    if idq_in='1' then
      if ipgh='0' then
        pix_count <= "00000000000";
      else
        pix_count <= pix_count + 1;
      end if;
    end if;
  end if;
end process;

-- count the number of lines
process (iclk_in, reset)
begin
  if reset='1' then
    line_counter <= X"0000";
  elsif iclk_in'event and iclk_in='1' then
    if ipgv='0' then
      line_counter <= X"0000";
    elsif ipgh='1' and pix_count=719 then
      line_counter <= line_counter + 1;
    end if;
  end if;
end process;

-- give the frame ID
process (iclk_in, reset)
begin
  if reset='1' then
    frame_counter <= "00";
  elsif iclk_in'event and iclk_in='1' then
    if line_counter = 239 and pix_count=719 then
      frame_counter <= frame_counter+1;
    end if;
  end if;
end process;

-- Active means we are within
-- the horizontal line active video
process (iclk_in)
begin
 if iclk_in'event and iclk_in='1' then
   if ipgh='0' then
     active <= '0';
   elsif pix_count = 1 then
     active <= '1';
   elsif pix_count=720 then
     active <= '0';
   end if;
 end if;
end process;

-- Set output signals according to where
-- in the current line the video decoder
-- is writing the block RAM
process (iclk_in, reset)
begin
 if reset='1' then
   one_fourth <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     one_fourth <= '0';
   elsif pix_count=161 then
     one_fourth <= '1';
   end if;
```

```
 end if;
end process;

process (iclk_in, reset)
begin
 if reset='1' then
   half_line <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     half_line <= '0';
   elsif pix_count=321 then
     half_line <= '1';
   end if;
 end if;
end process;

process (iclk_in, reset)
begin
 if reset='1' then
   three_quarters <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     three_quarters <= '0';
   elsif pix_count=481 then
     three_quarters <= '1';
   end if;
 end if;
end process;

process (iclk_in, reset)
begin
 if reset='1' then
   entire_line <= '0';
 elsif iclk_in'event and iclk_in='1' then
   if pix_count=0 then
     entire_line <= '0';
   elsif pix_count=641 then
     entire_line <= '1';
   end if;
 end if;
end process;


filling_level(0) <= one_fourth;
filling_level(1) <= half_line;
filling_level(2) <= three_quarters;
filling_level(3) <= entire_line;


-- Output signals of this entity

fil_level <= filling_level;
line_count <= line_counter;
frame_id <= frame_counter;

  data(15 downto 8) <= iport;----------------------------------------these have to be
reinserted
  data(7 downto 0)  <= hport;----------------------------------------these have to be
reinserted
  idq_out <= idq_in;         ----------------------------------------these have to be
reinserted


waddr <= pixel_addr;
iclk_out <= iclk_in;
pixel_counter <= pix_count;
active_out <= active;



-- -- Test generator
-- data <= pixel_data;
```

```
-- idq_out <= '1';
-- line_start <= pix_count;  -- - line_counter - 4;
--  -- begin
--    --This is the test date for the line finder - we want to insert and remove
--    --this one as much as possible
--  -- pixel data
--  process (iclk_in, reset)
--  begin
--    if reset='1' then
--      pixel_data <= X"0000";
--    elsif iclk_in'event and iclk_in = '1' then
--      if line_start=12 then
--        pixel_data <= X"0000";
--      elsif line_start=45 then
--        pixel_data <= X"F000";
--      elsif line_start=100 then
--        pixel_data <= X"0000";
--      elsif line_start=200 then
--        pixel_data <= X"F000";
--      elsif line_start=225 then
--        pixel_data <= X"0000";
--      elsif line_start=275 then
--        pixel_data <= X"F000";
--      elsif line_start=306 then
--        pixel_data <= X"0000";
--       elsif line_start=382 then
--         pixel_data <= X"F000";
--       elsif line_start=383 then
--         pixel_data <= X"0000";
--      elsif line_start=456 then
--        pixel_data <= X"F000";
--      end if;
--    end if;
--  end process;

-- --ends here the test data


-- -- pixel address - where to store in the block RAMs
-- process (iclk_in, reset)
-- begin
--   if reset='1' then
--      pixel_addr <= "00000000000";
--   elsif iclk_in'event and iclk_in='1' then
--      pixel_addr <= pixel_addr + 1;
--   end if;
-- end process;


-- data <= pixel_data;
-- waddr <= pixel_addr;
-- iclk_out <= iclk_in;
-- idq_out <= '1';

 end structural;
```

System.mhs

```
# Parameters
PARAMETER VERSION = 2.0.0

# Global Ports

# Signals of opb_xsb300 module
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

#Signals for video decoder I2C Bus
PORT VID_I2C_SCL = VID_I2C_SCL, DIR = INOUT
PORT VID_I2C_SDA = VID_I2C_SDA, DIR = INOUT

# Signals of opb_videodec module
PORT IPort = IPort, DIR=IN, VEC=[7:0]
PORT HPort = HPort, DIR=IN, VEC=[7:0]
PORT IDQ = IDQ, DIR=IN
PORT ICLK = ICLK, DIR=IN
PORT IPGV = IPGV, DIR=IN
PORT IPGH = IPGH, DIR=IN
PORT ITRI = ITRI, DIR=OUT
PORT ITRDY = ITRDY, DIR=OUT

# 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 opb_videodec
 PARAMETER INSTANCE = videodec
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x01800000
 PARAMETER C_HIGHADDR = 0x01803FFF
 PORT IPort = IPort
 PORT HPort = HPort
 PORT IDQ = IDQ
 PORT ICLK = ICLK
 PORT IPGV = IPGV
 PORT IPGH = IPGH
 PORT ITRI = ITRI
 PORT ITRDY = ITRDY
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN opb_i2ccontroller
 PARAMETER INSTANCE = i2c
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFF0200
 PARAMETER C_HIGHADDR = 0xFEFF02ff
 PORT VID_I2C_SCL = VID_I2C_SCL
 PORT VID_I2C_SDA = VID_I2C_SDA
 PORT OPB_Clk = sys_clk
 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 = 0x000007FF
# 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_BAUDRATE   = 2400
 PARAMETER C_USE_PARITY = 1
 PARAMETER C_ODD_PARITY = 1

 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 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
```

System.mss

```
PARAMETER VERSION = 2.0.0
PARAMETER HW_SPEC_FILE = system.mhs

BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER EXECUTABLE = system.elf
 PARAMETER COMPILER = microblaze-gcc
 PARAMETER ARCHIVER = microblaze-ar
 PARAMETER DEFAULT_INIT = EXECUTABLE
 PARAMETER STDIN = myuart
 PARAMETER STDOUT = myuart
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = xsb300
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = videodec
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = i2c
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER LEVEL = 0
END

#BEGIN DRIVER
# PARAMETER HW_INSTANCE = intc
# PARAMETER DRIVER_NAME = intc
# PARAMETER DRIVER_VER = 1.00.b
# PARAMETER LEVEL = 0
#END
```

System.ucf

```
net ICLK period = 30.000;
net sys_clk period = 18.000;
net pixel_clock period = 36.000;


net FPGA_CLK1 loc="p77";

net PB_A<0> loc="p83"; #BAR1
net PB_A<1> loc="p84"; #BAR2
net PB_A<2> loc="p86"; #BAR3
net PB_A<3> loc="p87"; #BAR4
net PB_A<4> loc="p88"; #BAR5
net PB_A<5> loc="p89"; #BAR6
net PB_A<6> loc="p93"; #BAR7
net PB_A<7> loc="p94"; #BAR8
net PB_A<8> loc="p100";
net PB_A<9> loc="p101";
net PB_A<10> loc="p102";
net PB_A<11> loc="p109";
net PB_A<12> loc="p110";
net PB_A<13> loc="p111";
net PB_A<14> loc="p112";
net PB_A<15> loc="p113";
net PB_A<16> loc="p114";
net PB_A<17> loc="p115";
net PB_A<18> loc="p121";
net PB_A<19> loc="p122";

net PB_D<0> loc="p153"; #LEFT_A
net PB_D<1> loc="p145"; #LEFT_B
net PB_D<2> loc="p141"; #LEFT_C
net PB_D<3> loc="p135"; #LEFT_D
net PB_D<4> loc="p126"; #LEFT_E
net PB_D<5> loc="p120"; #LEFT_F
net PB_D<6> loc="p116"; #LEFT_G
net PB_D<7> loc="p108"; #LEFT_DP
net PB_D<8> loc="p127"; #RIGHT_A
net PB_D<9> loc="p129"; #RIGHT_B
net PB_D<10> loc="p132"; #RIGHT_C
net PB_D<11> loc="p133"; #RIGHT_D
net PB_D<12> loc="p134"; #RIGHT_E
net PB_D<13> loc="p136"; #RIGHT_F
net PB_D<14> loc="p138"; #RIGHT_G
net PB_D<15> loc="p139"; #RIGHT_DP


net PB_LB_N loc="p140";        #BAR9
net PB_UB_N loc="p146";  #BAR10
net PB_WE_N loc="p123";
net PB_OE_N loc="p125";


net RAM_CE_N loc="p147";



net VIDOUT_CLK loc="p23";
net VIDOUT_BLANK_N loc="p24";
net VIDOUT_HSYNC_N loc="p8";
net VIDOUT_VSYNC_N loc="p7";


net VIDOUT_RCR<0> loc="p41";
net VIDOUT_RCR<1> loc="p40";
net VIDOUT_RCR<2> loc="p36";
net VIDOUT_RCR<3> loc="p35";
net VIDOUT_RCR<4> loc="p34";
net VIDOUT_RCR<5> loc="p33";
net VIDOUT_RCR<6> loc="p31";
net VIDOUT_RCR<7> loc="p30";
```

```
net VIDOUT_RCR<8> loc="p29";
net VIDOUT_RCR<9> loc="p27";

net VIDOUT_GY<0> loc="p9" ;
net VIDOUT_GY<1> loc="p10";
net VIDOUT_GY<2> loc="p11";
net VIDOUT_GY<3> loc="p15";
net VIDOUT_GY<4> loc="p16";
net VIDOUT_GY<5> loc="p17";
net VIDOUT_GY<6> loc="p18";
net VIDOUT_GY<7> loc="p20";
net VIDOUT_GY<8> loc="p21";
net VIDOUT_GY<9> loc="p22";

net VIDOUT_BCB<0> loc="p42";
net VIDOUT_BCB<1> loc="p43";
net VIDOUT_BCB<2> loc="p44";
net VIDOUT_BCB<3> loc="p45";
net VIDOUT_BCB<4> loc="p46";
net VIDOUT_BCB<5> loc="p47";
net VIDOUT_BCB<6> loc="p48";
net VIDOUT_BCB<7> loc="p49";
net VIDOUT_BCB<8> loc="p55";
net VIDOUT_BCB<9> loc="p56";

net RS232_TD loc="p71";
net RS232_RD loc="p73";
#net RS232_CTS loc="p69";
#net RS232_RTS loc="p70";

net AU_CSN_N loc="p165";
net AU_BCLK loc="p166";
net AU_MCLK loc="p167";
net AU_LRCK loc="p168";
net AU_SDTI loc="p169";
net AU_SDTO0 loc="p173";

# Ports of opb_videodec
net IPort<0> loc="p188";
net IPort<1> loc="p189";
net IPort<2> loc="p191";
net IPort<3> loc="p192";
net IPort<4> loc="p193";
net IPort<5> loc="p194";
net IPort<6> loc="p198";
net IPort<7> loc="p199";

net HPort<0> loc="p174";
net HPort<1> loc="p175";
net HPort<2> loc="p176";
net HPort<3> loc="p178";
net HPort<4> loc="p179";
net HPort<5> loc="p180";
net HPort<6> loc="p181";
net HPort<7> loc="p187";

net IDQ loc="p205";
net ICLK loc="p185";
net IPGH loc="p200";
net IPGV loc="p201";
net ITRI loc="p204";
net ITRDY loc="p206";


# Video decoder I2C Bus
net VID_I2C_SCL loc="p6";
net VID_I2C_SDA loc="p5";
```

opb_xsb300_v2_0_0.pao

```
lib opb_xsb300_v1_00_a opb_xsb300
lib opb_xsb300_v1_00_a memoryctrl
lib opb_xsb300_v1_00_a vga
lib opb_xsb300_v1_00_a vga_timing
lib opb_xsb300_v1_00_a pad_io
```

opb_xsb300_v2_0_0.mpd

```
##############################################################################
##
## Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.
##
## opb_emc_v2_0_0.mpd
##
## Microprocessor Peripheral Definition
##
##############################################################################

PARAMETER VERSION = 2.0.0

BEGIN opb_xsb300, IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
#OPTION CORE_STATE = DEVELOPMENT

# Define bus interface
BUS_INTERFACE BUS=SOPB, BUS_STD=OPB, BUS_TYPE=SLAVE

# Generics for vhdl or parameters for verilog
PARAMETER C_OPB_AWIDTH = 32, DT=integer
PARAMETER C_OPB_DWIDTH = 32, DT=integer
PARAMETER C_BASEADDR = 0xFFFFFFFF, DT=std_logic_vector, MIN_SIZE=0x100, BUS=SOPB
PARAMETER C_HIGHADDR = 0x00000000, DT=std_logic_vector, BUS=SOPB


# Signals
PORT OPB_Clk = "", DIR=IN, BUS=SOPB, SIGIS=CLK
PORT OPB_Rst = OPB_Rst, DIR=IN, BUS=SOPB

# OPB slave signals
PORT OPB_ABus = OPB_ABus, DIR=IN, VEC=[0:C_OPB_AWIDTH-1], BUS=SOPB
PORT OPB_BE = OPB_BE, DIR=IN, VEC=[0:C_OPB_DWIDTH/8-1], BUS=SOPB
PORT OPB_DBus = OPB_DBus, DIR=IN, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR=IN, BUS=SOPB
PORT OPB_select = OPB_select, DIR=IN, BUS=SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR=IN, BUS=SOPB
PORT UIO_DBus = Sl_DBus, DIR=OUT, VEC=[0:C_OPB_DWIDTH-1], BUS=SOPB
PORT UIO_errAck = Sl_errAck, DIR=OUT, BUS=SOPB
PORT UIO_retry = Sl_retry, DIR=OUT, BUS=SOPB
PORT UIO_toutSup = Sl_toutSup, DIR=OUT, BUS=SOPB
PORT UIO_xferAck = Sl_xferAck, DIR=OUT, BUS=SOPB


PORT PB_A = "", DIR=OUT, VEC=[19:0], IOB_STATE=BUF
PORT PB_LB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_UB_N = "", DIR=OUT, IOB_STATE=BUF
PORT PB_D = "", DIR=INOUT, VEC=[15:0], 3STATE=FALSE, IOB_STATE=BUF
PORT PB_WE_N = "", DIR = OUT, IOB_STATE=BUF
PORT PB_OE_N = "", DIR = OUT, IOB_STATE=BUF
PORT RAM_CE_N = "", RAM_CE_N, DIR = OUT, IOB_STATE=BUF

PORT pixel_clock = "", DIR=IN

PORT VIDOUT_CLK = "", DIR=OUT, IOB_STATE=BUF
PORT VIDOUT_RCR = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_GY  = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_BCB = "", DIR=OUT, VEC=[9:0]
PORT VIDOUT_BLANK_N = "", DIR=OUT
PORT VIDOUT_HSYNC_N = "", DIR=OUT
PORT VIDOUT_VSYNC_N = "", DIR=OUT


END
```

isr.c

```
#include "xbasic_types.h"
#include "xio.h"
#include "xparameters.h"
#include "xuartlite_l.h"

int uart_interrupt_count = 0;
char uart_character;

/*
 * Interrupt service routine for the UART
 */
void uart_handler(void *callback)
{
  Xuint32 IsrStatus;

  Xuint8 incoming_character;

  /* Check the ISR status register so we can identify the interrupt source */
  IsrStatus = XIo_In32(XPAR_MYUART_BASEADDR + XUL_STATUS_REG_OFFSET);

  if ((IsrStatus & (XUL_SR_RX_FIFO_FULL | XUL_SR_RX_FIFO_VALID_DATA)) != 0) {
    /* The input FIFO contains data: read it */
    incoming_character =
      (Xuint8) XIo_In32( XPAR_MYUART_BASEADDR + XUL_RX_FIFO_OFFSET );

    uart_character = incoming_character;
    ++uart_interrupt_count;
  }

  if ((IsrStatus & XUL_SR_TX_FIFO_EMPTY) != 0) {
    /* The output FIFO is empty: we can send another character */
  }

}
```

Set_registers.c

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

#define W 640
#define H 480
#define VGA_START 0x00800000
#define RED 0xE0
#define GREEN 0x1C
#define BLUE 0x03

#define NOT_VERT_SYNC !XIo_In32(0x01802FFC)
#define FILLING_LEVEL XIo_In32(0x01803FFC)
#define FRAME_ID XIo_In32(0x01803FFC)
#define LINE_DATA XIo_In32(0x018008FC)
#define LINE_ARRAY_LENGTH 240

int i;


// Register addresses for SAA7114H configuration
unsigned char registers [] = {

  // Video decoder "generic" registers //
  0x01, 0x08, // Recommended setting
  0x02, 0xE4, // Analog input control 1 and input selection
  0x03, 0x10, // Analog input control 2
  0x04, 0x90, // Analog input control 3
  0x05, 0x90, // Analog input control 4
  0x06, 0xEB, // Horizontal Sync Start (delay)
  0x07, 0xE0, // Horizontal Sync Stop (delay)
  0x08, 0x59, //0x98, // 0xD8 // Sync control
  0x09, 0x40, // 0x45, // Luminance control
  0x0A, 0x80,
  0x0B, 0x44,
  0x0C, 0x40,
  0x0D, 0x00,
  0x0E, 0x89,
  0x0F, 0x2A, //0x24, // Chrominance gain
  0x10, 0x0E, //0x0C, // Chrominance control
  0x11, 0x00,
  0x12, 0x46, //0x00, // RT signal control
  0x13, 0x00,
  0x14, 0x00,
  0x15, 0x11,
  0x16, 0xFE,
  0x17, 0x40, //0x00, //0x40,

  0x18, 0x40,
  0x19, 0x80,
  0x1A, 0x00,
  0x1B, 0x00,
  0x1C, 0x00,
  0x1D, 0x00,
  0x1E, 0x00,
  0x30, 0x08, //0xBC, //0x08, // Audio clock stuff
  0x31, 0x08, //0xDF, //0x08, // Audio clock stuff
  0x32, 0x02,

  // Following 1 is added
  0x33, 0x00,

  0x34, 0xCD, //0x08, //0xCD, //0x08,
  0x35, 0xCC, //0x08, //0xCC, //0x08,
  0x36, 0x3A, //0x08, //0x3A, //0x08,

  // Following 1 is added
  0x37, 0x00,

  0x38, 0x03, //0x08, //0x03, //0x08,
  0x39, 0x10, //0x08, //0x10, //0x08,
```

```
0x3A, 0x00, //0x03, //0x00, //0x03,


// Following 5 are added
0x3B, 0x00,
0x3C, 0x00,
0x3D, 0x00,
0x3E, 0x00,
0x3F, 0x00,

0x40, 0x40, //0x00, //0x40, //0x00,
0x41, 0xFF,
0x42, 0xFF,
0x43, 0xFF,
0x44, 0xFF,
0x45, 0xFF,
0x46, 0xFF,
0x47, 0xFF,
0x48, 0xFF,
0x49, 0xFF,
0x4A, 0xFF,
0x4B, 0xFF,
0x4C, 0xFF,
0x4D, 0xFF,
0x4E, 0xFF,
0x4F, 0xFF,
0x50, 0xFF,
0x51, 0xFF,
0x52, 0xFF,
0x53, 0xFF,
0x54, 0xFF,
0x55, 0xFF,
0x56, 0xFF,
0x57, 0xFF,

0x58, 0x40, //0x00, //0x40,
0x59, 0x47,
0x5A, 0x06,
0x5B, 0x03, //0x83, //0x03,


// Following 1 is added
0x5C, 0x00,

0x5D, 0x3E, //0x00, //0x3E, //0x00,
0x5E, 0x00,

// Following 1 is added
0x5F, 0x00,


0x80, 0x30, //0x10, 0x10: only Task A 0x30 : both tasks.
0x83, 0x01,
0x84, 0xA0, //0xF0,
0x85, 0x10, //0x00,
0x86, 0x45,
0x87, 0x01,
0x88, 0xF0,

// Task A Registers //
0x90, 0x00, //0x00, //0x01,
0x91, 0x08, //0x00,
0x92, 0x10, //0x80,
0x93, 0xC0, //0xC0,
0x94, 0x10, //0x02,
0x95, 0x00,
0x96, 0xD0,
0x97, 0x02,
0x98, 0x0A,
0x99, 0x00,
0x9A, 0xF2, //0x00,
0x9B, 0x00,
0x9C, 0xD0, // Horizontal output window size upper bits \ 0xD002 = 720
```

```
0x9D, 0x02, // Horizontal output window size lower bits /          by
0x9E, 0xF0, // Vertical output window size upper bits  \  0xF000 = 240
0x9F, 0x00, // Vertical output window size lower bits  /
0xA0, 0x01,
0xA1, 0x00,
0xA2, 0x00,
0xA4, 0x80,
0xA5, 0x40,
0xA6, 0x40,
0xA8, 0x00,
0xA9, 0x04,
0xAA, 0x00,
0xAC, 0x00,
0xAD, 0x02,
0xAE, 0x00,
0xB0, 0x00,
0xB1, 0x04,
0xB2, 0x00,
0xB3, 0x04, // 0x02
0xB4, 0x00,
0xB8, 0x00,

// Following 3 were added
0xB9, 0x00,
0xBA, 0x00,
0xBB, 0x00,

0xBC, 0x00,

// Following 3 were added
0xBD, 0x00,
0xBE, 0x00,
0xBF, 0x00,


// Task B Registers //
0xC0, 0x00, //0x00,
0xC1, 0x08, //0x00,
0xC2, 0x10, //0x80,
0xC3, 0xC0,
0xC4, 0x10, //0x02,
0xC5, 0x00,
0xC6, 0xD0, //0x01,
0xC7, 0x02,
0xC8, 0x0A,
0xC9, 0x00,
0xCA, 0xF2,
0xCB, 0x00,
0xCC, 0xD0,
0xCD, 0x02,
0xCE, 0xF0,
0xCF, 0x00,
0xD0, 0x01,
0xD1, 0x00,
0xD2, 0x00,
0xD4, 0x80,
0xD5, 0x40,
0xD6, 0x40,
0xD8, 0x00,
0xD9, 0x04,
0xDA, 0x00,
0xDC, 0x00,
0xDD, 0x02,
0xDE, 0x00,
0xE0, 0x00,
0xE1, 0x04,
0xE2, 0x00,
0xE3, 0x04, //0x02
0xE4, 0x00,
0xE8, 0x00,
0xE9, 0x00,
0xEA, 0x00,
0xEB, 0x00,
```

```
  0xEC, 0x00,
  0xED, 0x00,
  0xEE, 0x00,
  0xEF, 0x00,

  // Reset sequence. Extremelly needed!!
  // Do not comment the following out! //
  0x88, 0xD8, //0xD0, //0xD8,
  0x88, 0xF8, //0xF0, //0xF8,
  0xFF, 0xFF,};

int w = 0xFF;

void i2c_delay()
{
    int i;
    for (i = 0; i < 1000; i++);
}

// Write "level" to SCL //
void SCLw(int level)
{
    if (level == 0)
      w &= 0xDF;
    else
      w |= 0x2F;

    // Assert the clock on SCL //
    // according to level      //
    XIo_Out8(0xFEFF0200, w);
    i2c_delay();
}

// Write "level" to SDA //
void SDAw(int level)
{
    if (level == 0)
      w &= 0x7F;
    else
      w |= 0x8F;

    // Assert the clock on SDA //
    // according to level      //
    XIo_Out8(0xFEFF0200, w);
    i2c_delay();
}

// Read from SDA //
int SDAr()
{
    int MSB = XIo_In8(0xFEFF0200);

    MSB = MSB >> 7;
    MSB &= 1;

    i2c_delay();
    return MSB;
}

// Tristate for SDA //
void SDAt(int rnw)
{
    if (rnw == 0)
      w &= 0xBF;
    else
      w |= 0x4F;

    // Assert the clock on SDA //
    // according to level      //
    XIo_Out8(0xFEFF0200, w);
    i2c_delay();
}
```

```
// Tristate for SCL //
void SCLt(int rnw)
{
    if (rnw == 0)
      w &= 0xEF;
    else
      w |= 0x1F;

    // Assert the clock on SDA //
    // according to level      //
    XIo_Out8(0xFEFF0200, w);
    i2c_delay();
}

// Send the start sequence
void send_start( void )
{
    SCLt(0);
    SDAt(0);
    SCLw(1);
    SDAw(0);
    SCLw(0);
}

// Send the restart sequence
// Needed for read register
void re_start( void )                /* This function must be entered with SDA High */
{
    SCLw(1);
    SDAw(0);
    SCLw(0);
}


// Send stop sequence
void send_stop( void )
{
    SCLw(0);
    SDAw(0);
    SCLw(1);
    SDAw(1);                         /* Should leave with both lines high to indicate
finish */
}

// Check acknowledge
int check_ack(void)
{
    int theresult;
    SDAt(1);
    SCLw(1);
    theresult=SDAr();
    SCLw(0);
    SDAw(1);                         /* Set the output before it becomes active to
eliminate spike */
    SDAt(0);
    return theresult;
}

// Send one bit
void send_bit(int x)
{
    x = x & 1;
    SDAw(x);
    SCLw(1);
    SCLw(0);
}

// Send an entire bit
void send_byte(int byte)
{
    int i;
    for (i = 7; i >= 0; i--)
    {
```

```
      send_bit(byte >> i);
    }
}

// Read a register from the video decoder
int read_register( int sub_address )
{
    int input = 0;
    int id;

    send_start();

    // Write slave address for SAA7114H is 43H //
    send_byte(0x42);

    check_ack();

    // Send the subaddress //
    send_byte(sub_address);

    check_ack();

    re_start();

    // Read address //
    send_byte(0x43);

    check_ack();

    SDAt(1);
    for( id=8 ; id>0 ; id=id-1 )
    {
        input=input<<1;
        SCLw(1);
        input=input|SDAr();
        SCLw(0);
    }
    SDAw(1);   /* Set the output prior to enable to eliminate spike and make compatible
with Restart */
    SDAt(0);
    SCLw(1);
    SCLw(0);
    send_stop();
    return input;
}

// Write a register into the video decoder
void write_register(int sub_address, int data)
{
    int i;

    // Start conditions //
    send_start();

    for (i = 0; i < 5; i++)
     i2c_delay();

    // Write slave address for SAA7114H is 42H //
    send_byte(0x42);

    check_ack();

    send_byte(sub_address);

    check_ack();

    send_byte(data);

    check_ack();

    send_stop();
}
```

```
void set_registers(){

 // Start the bus  protocol  by  sending //
  // a stop handshaking (SDA=1 and SCL=1) //
  send_stop();

  print("Configuring video decoder...");

  i = 0;

  // Configure the video decoder SAA7114H //
  while (registers[i] != 0xFF) {
    write_register (registers[i], registers[i+1]);
    i+=2;
  }

  print("Video decoder configured!\r\n");

/*   // Clear screen // */
/*   for (i = 0; i < H*W; i++) */
/*     XIo_Out8(VGA_START + i, 0); */

/*   for (i=0; i<100000000;i++); */

  // How the Philips chip responded to the configuration
  //  putnum(read_register(0x1F));

}
```

Write_video.c

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

#define W 640
#define VGA_START 0x00800000
#define BRAM_START 0x01800000

void write_video(int start, int end, int line)
{
  int nPixs;
  Xuint32 luma_4pixels;
  Xuint32 bram_addr;
  Xuint32 vga_addr;

  nPixs = (end - start);
  vga_addr = VGA_START + (start>>1) + W*line;
  bram_addr = BRAM_START + (start>>1);

  while (nPixs > 0)
  {
    luma_4pixels = XIo_In32(bram_addr+0);
    XIo_Out32(vga_addr+0, luma_4pixels);

    luma_4pixels = XIo_In32(bram_addr+4);
    XIo_Out32(vga_addr+4, luma_4pixels);

    luma_4pixels = XIo_In32(bram_addr+8);
    XIo_Out32(vga_addr+8, luma_4pixels);

    luma_4pixels = XIo_In32(bram_addr+12);
    XIo_Out32(vga_addr+12, luma_4pixels);

    bram_addr += 16;
    vga_addr += 16;
    nPixs -=32;

    if (!XIo_In32(0x01802FFC))
      break;
  }
}
```

Lighfinder.c

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

#include "xparameters.h"

#define N 9
#define ITER 2000

int send_msg (char *, int);
int send_msg_header (void);
void mywait (void);

char msg_header[3] = { 0x55, 0xff, 0x00 };
char sync[1] = { 0x10 };
char music_left[2] = { 0x51, 0x05 };
char music_right[2] = { 0x51, 0x01 };
char set_go[2] = { 0x21, 0x83 };
char set_stop[2] = { 0x21, 0x43 };
char set_reverse[2] = { 0xe1, 0x03 };
char set_reverse_b[2] = { 0xe1, 0x42 };
char set_reverse_a[2] = { 0xe1, 0x41 };
char set_forward[2] = { 0xe1, 0x83 };
char make_slow[4] = { 0x13, 0x03, 0x02, 0x02 };
char make_fast[4] = { 0x13, 0x03, 0x02, 0x04 };
char last_op = 0x00;
Xuint32 counter = 0;
int lastRegion;
#define ROWTOUSE 1
#define REGION_THRESH 6000
#define SCREENSECTIONS 3
#define FRAMEHEIGHT 240
#define FRAMEWIDTH 320
#define LINES_PER_SECTION (FRAMEHEIGHT/SCREENSECTIONS)

#define LEFT_BORDER 106
#define RIGHT_BORDER 213


int section[SCREENSECTIONS][3];


int start;
int length;
int endpoint;

int max_row;
int max_length;

int line_data[FRAMEHEIGHT];

int count;

int lcount, j, k, l, r;
int m, n;

/* section[SCREENSECTIONS][0]=left
   section[SCREENSECTIONS][1]=middle
   section[SCREENSECTIONS][2]=right
*/



extern Xuint32 line_array[FRAMEHEIGHT];

void
count_pixel ()
{
  lcount = 0;
  j = 0;
```

```
  for (n = 0; n < 3; n++)
    {
      for (m = 0; m < SCREENSECTIONS; m++)
        {
          section[m][n] = 0;
          section[m][n] = 0;
          section[m][n] = 0;
        }
    }



  for (j = 0; j < SCREENSECTIONS; j++)
    {                          //every time this one finishes we have calculated the
pixels in each section of the frame
      for (k = 0; k < LINES_PER_SECTION; k++)
        {                      //every time this loop finishes execution we have
calculated the amount of pixels per section.


          //  for(count=0; count<240; count++){
          //   print("do u come here?");



          length = line_array[lcount] & 0x0000ffff;
          start = ((line_array[lcount] & 0xffff0000) >> 16);
          endpoint = start + length;

          /*     putnum(line_array[count]);
             print(" ");
             putnum(start);
             print(" ");
             putnum(length);
             print("\n\r");
           */

          //Calculate # of pixels in left section
          if (start < 106 && endpoint < 106)
            {
              l = length;
              section[j][0] = length + section[j][0];
              //left[j] = length + left[j];
            }
          else if (start < 106 && endpoint > 106)
            {
              l = (106 - start);
              section[j][0] = (106 - start) + section[j][0];
              //left[j] = (106 - start) + left[j];
            }


          // Calculate # of pixels in right section
          if (213 < endpoint && 213 < start)
            {
              r = length;
              section[j][2] = length + section[j][2];
              //right[j] = length + right[j];
            }
          else if (213 < endpoint && 213 > start)
            {
              r = endpoint - 213;
              section[j][2] = endpoint - 213 + section[j][2];
              //right[j] = endpoint - 213 + right[j];
            }


          // Calculate the # of pixels in the middle section


          section[j][1] = section[j][1] + (length - (l + r));
```

```
          l = 0;
          r = 0;
          lcount = lcount + 1;
        }                       //end for k<lines per section

       /*
          putnum(j);
          print(" ");
          putnum(section[j][0]);
          print(" ");
          putnum( section[j][1]);
          print(" ");
          putnum(section[j][2]);
          print("\n\r");
        */


    }
}                               //end count_pixel

movement ()
{

  //go through the first row
  //whichever has the highest value
  //move accordingly

  int i, j;



  int highest_column[3] = { -1, -1, -1 };
  int highest_value[3] = { REGION_THRESH, REGION_THRESH, REGION_THRESH };
  send_msg (make_slow, 4);

  for (j = 0; j < 3; j++)
    {
      for (i = 0; i < 3; i++)
        {
          if (i == 2)
            section[j][i] *= 1.4;

          if (section[j][i] > highest_value[j])
            {
              highest_column[j] = i;
              highest_value[j] = section[j][i];
            }

        }

    }


//Error, return and do nothing

  for (j = 0; j < 3; j++)
    if (highest_column[j] == 0)
      {
        pt_turn_left ();
        return;
      }
    else if (highest_column[j] == 1)
      {
        forward ();
        return;
      }
    else if (highest_column[j] == 2)
      {
        pt_turn_right ();
        lastRegion = highest_column[j];
        return;
```

```
      }
  j = j - 1;
  if(lastRegion == -1){

    pt_turn_right();
    forward();
    return;
  }
  else{

  highest_column[2] = lastRegion;

  }
}


forward ()
{
  send_msg (set_forward, 2);
  send_msg (set_go, 2);
}

stop ()
{
  send_msg (set_stop, 2);
}

reverse ()
{
  send_msg (set_reverse, 2);
  send_msg (set_go, 2);
}

pt_turn_right ()
{
//      stop();
//      send_msg(music_left,2);
  send_msg (set_forward, 2);
  send_msg (set_reverse_b, 2);
  send_msg (set_go, 2);
}

pt_turn_left ()
{
//      stop();
//      send_msg(music_right,2);
  send_msg (set_forward, 2);
  send_msg (set_reverse_a, 2);
  send_msg (set_go, 2);

}

sync_rcx ()
{

  send_msg (sync, 2);
  send_msg (sync, 2);
  send_msg (music_right, 2);



}

wait ()
{

  int i;

  for (i = 0; i < 1550000; i++)
    i = i;

}
```

```
long_wait ()
{

  int i, j;

  for (i = 0; i < 1000000; i++)
    for (j = 0; j < 10; j++)
      j = j;
}

int
send_msg (char *message, int length)
{
  int i;
  char temp = message[0];
  char message_comp, checksum, checksum_comp;

  wait ();
  if (last_op == message[0])
    message[0] = message[0] + 0x08;


  checksum = 0;
  //Send the header
  send_msg_header ();

  //Now lets send the data, each data bit
  //is followed by its compliment and the
  //checksum is updated

  for (i = 0; i < length; i++)
    {
      XUartLite_SendByte (XPAR_MYUART_BASEADDR, message[i]);
      //wait();
      checksum += message[i];  /*update the checksum */
      message_comp = (~message[i]) & 0xff;    /*get the compliment */
      XUartLite_SendByte (XPAR_MYUART_BASEADDR, message_comp);
      //wait();
    }

  XUartLite_SendByte (XPAR_MYUART_BASEADDR, checksum);
  //wait();
  checksum_comp = (~checksum) & 0xff;
  XUartLite_SendByte (XPAR_MYUART_BASEADDR, checksum_comp);

  message[0] = temp;
  last_op = message[0];
  return 0;
}

int
send_msg_header (void)
{
  int i;
  for (i = 0; i < 2; i++)
    XUartLite_SendByte (XPAR_MYUART_BASEADDR, msg_header[i]);
  return 0;
}
```

Main.c

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

#define W 640
#define H 480
#define VGA_START 0x00800000
#define RED 0xE0
#define GREEN 0x1C
#define BLUE 0x03

#define LEFT_B 106
#define RIGHT_B 213

#define VERT_SYNC ((XIo_In32(0x01803FFC) & 0x20))
#define NOT_VERT_SYNC (! (XIo_In32(0x01803FFC) & 0x20))
#define FILLING_LEVEL (XIo_In32(0x01803FFC) & 0x0F)
//#define FRAME_ID XIo_In32(0x01803FFC)
#define LINE_DATA XIo_In32(0x018008FC)
#define STRENGTH XIo_In32(0x01802FFC)
#define LINE_ARRAY_LENGTH 240

#define THRESHOLD 0x53 //day threshold
//#define THRESHOLD 0x51 //night threshold

#define SCREENSECTIONS 3  //make sure this agrees with lightfinder.c



extern void write_video(int start, int end, int line);
extern void set_registers();

extern int section[SCREENSECTIONS][3];

extern void count_pixel();
extern void movement();
extern void sync_rcx();


extern int lastRegion;
extern int last_Block;
Xuint32 line_array[LINE_ARRAY_LENGTH];
Xuint32 strength_array[LINE_ARRAY_LENGTH];
Xuint32 last_strength[LINE_ARRAY_LENGTH];


int main()
{
  Xuint8 bw_1pixel;
  Xuint32 bw_4pixels;
  Xuint32 luma_4pixels;
  Xuint32 current_level;
  int frame_id;
  int line_number = 0;
  int line;
  int start, end;
  int i, j, x;
  int line_section;
  int testframe;
  int length;
  int position;
  int b;
  int framenum;
  int average;
// print("Hello World!\r\n");

  lastRegion = -1;

  microblaze_enable_icache();

  set_registers();
```

```
  sync_rcx();
  // Clear screen //
  for (i = 0; i < H*W; i++)
    XIo_Out8(VGA_START + i, 0);

  // THis is a delay to ensure that the chip is configured
  for (i=0; i<10000000;i++);

  line = 0;
  testframe = 0;
  framenum = 0;


  XIo_Out32(0x01802FFC, THRESHOLD);

  /*Top while loop returns us to grab another frame after we do some processing*/
  while (1){

    //This while loop grabs a whole screen/frame of video and then lets us out to process
it if we wish
    while (line<239){                          //  while (line<239 && framenum == 0) {

      while (VERT_SYNC);
      // Wait for the vertical synchronism
      while (NOT_VERT_SYNC);

      line = -1;


      while (1)
        {
          line = line + 1;

          // This variable indicates how much of //
          // the current line has  been  already //
          // written into the block RAMs         //
          current_level = 0x0001;

          for (line_section = 0; line_section < 4; line_section++)
            {
              // Wait for the current line to be  1/4,  1/2,  3/4 //
              // and full filled. The while then executes 4 times //
              while (!(FILLING_LEVEL & current_level))
                {
                  if (NOT_VERT_SYNC)
                    break;
                }

              /*  if (current_level == 0x01) { */
              /*               start = 0; */
              /*               end = 160; */
              /*             } */
              /*             else if (current_level == 0x02) { */
              /*               start = 160; */
              /*               end = 320; */
              /*             } */
              /*             else if (current_level == 0x04) { */
              /*               start = 320; */
              /*               end = 480; */
              /*             } */
              if (current_level == 0x08) {  //was originally an else if
                start = 480;
                end = 640;
                line_array[line] = LINE_DATA;
              }

              if (NOT_VERT_SYNC)
                break;

              //      write_video(start, end, line);

              current_level = current_level << 1;
```

```
            if (NOT_VERT_SYNC)
              break;
          }

        if (NOT_VERT_SYNC)
          break;
      }
    }


    for(i=241; i<480; i++){
      position = (line_array[i-240]>>16) & 0xFFFF;
      length = line_array[i-240] & 0xFFFF;
      for(j=0; j<position && j<320; j++){
        XIo_Out8 (VGA_START +i*W + j, 0xFF);
      }
      for(j=position;j<position+length && j<320;j++){
        XIo_Out8 (VGA_START +i*W + j, 0x00);
      }
      for(j=position+length;j<320;j++)
        XIo_Out8 (VGA_START +i*W + j, 0xFF);
      XIo_Out8(VGA_START +i*W+LEFT_B, GREEN);
      XIo_Out8(VGA_START +i*W+RIGHT_B, GREEN);
    }


    count_pixel();           //                                    PUT THIS BACK IN!!!!
    movement();


    line = -1;

  } //while (1)


  return 0;
}
```