

Light saber generator-*Return of the Jedi*

[CSEE 4840 Project Report – May 2009]

Anusha Dachepally
(ad2657)

Roopa Kakarlapudi
(rk2489)

Devesh Dedhia
(ddd2121)

Raghu Binnamangalam
(rsb2145)



Contents

1	INTRODUCTION.....	4
2	TOP LEVEL SYSTEM DESIGN.....	6
3	HARDWARE DESIGN.....	7
3.1	I2C configuration:.....	7
3.2	ITU DECODER.....	7
3.2.1	Understanding the ITU Standard.....	7
3.2.2	YCrCb Stream.....	7
3.2.3	SAV and EAV timing codes.....	9
3.2.4	VERTICAL BLANKING INTERVALS.....	9
3.2.5	VERILOG MODULE.....	10
3.2.6	SLIDING WINDOW.....	10
3.2.7	DOWN SAMPLE 720 to 640.....	11
3.2.8	ACTIVE VIDEO.....	11
3.2.9	XY DETECTION.....	11
3.2.10	YCrCb TO RGB CONVERTOR.....	12
3.2.11	HANDLING SPECIAL CASES.....	13
3.2.12	AVALON COMMUNICATOR.....	13
3.2.13	WRITE TRANSFERS.....	14
3.2.14	READ TRANSFERS.....	14
3.2.15	SYNCHRONIZATION.....	14
3.3	VGA Unit.....	15
3.3.1	BASIC DESIGN.....	15
3.3.2	DESIGN DECISION.....	15
3.3.3	PROBLEMS FACED AND SOLUTIONS.....	15
	MODIFIED DISPLAY FOR LIGHT SABER.....	16
4	SOFTWARE SYSTEM.....	16
4.1	INTRODUCTION.....	16
4.2	IDENTIFICATION OF CENTER OF MASS.....	16
4.2.1	Routine.....	17

4.2.2	Timing Requirement:	18
4.2.3	LIGHT SABER CALCULATION	18
4.2.4	OPTIMIZATION	20
4.2.5	FINAL OUTPUT.....	20
5	RESOURCE UTILIZATION	21
6	DESIGN EXPERIENCES.....	22
6.1	CHALLENGES FACED IN HARDWARE	22
6.1.1	Synchronization:.....	22
6.1.2	Compilation Delay:.....	22
6.1.3	Multiple Clock Domains:	22
6.1.4	SOFTWARE DESIGN CHALLENGES	22
6.1.5	Floating point computation:	22
6.1.6	Sampling signals(from hardware) at higher frequency:	22
7	PITFALLS AND SUGGESTIONS	23
7.1.1	Color Detection:	23
7.1.2	Computational delay in software:.....	23
8	LESSONS LEARNT	23
9	PICTURES of the JEDI	24
10	TASK DIVISION	24
11	ACKNOWLEDGEMENT	24
12	REFERENCES	25
13	APPENDIX	25

ABSTRACT

The goal of this project was to create special effects on the incoming video and display it in real-time at 60fps. Specifically, we aimed to recognize a sword in the input video and replace it with a light saber (of the Star Wars fame). The Light saber generator system designed was robust and could successfully emulate a real light saber with a halo around the sword even from a significant distance from the camera.

1 INTRODUCTION

The Light Saber Generator (henceforth referred to as LSG) is a fairly complex video processing application which can be used in the entertainment industry (for sequels of Star Wars!!). The motivation in choosing this project was our passion for Video processing. A plus point of this choice was the ease of debugging with the help of output display.

The project is a Hardware/Software co-design. We developed a customized video accelerator (LSG) in which the hardware components provide for real-time processing and the software adds flexibility to the special effects that can be created. In our design, the decisions to implement sub-blocks either in hardware/software were made to obtain maximum efficiency. The block that recognizes blue and green pixels (ends of sword) has been implemented in hardware. It sends this information to the software which draws the light saber in place of the sword.

In this project, we implemented the Light Saber Generator on the Cyclone II FPGA embedded in the Altera DE2 board. The video input to the system comes from a camera connected to the S-video input on the board, and is an analog signal that carries the video data as two separate signals, *lumen* (luminance) and *chroma* (color).

The ADV7181B integrated video decoder converts this signal compatible with NTSC standard into 4:2:2 component video data-compatible with 16-bit/8-bit ITU-R BT.656/601. This data stream goes to the LSG implemented on the FPGA. The LSG uses straightforward image processing techniques to identify the sword (i.e., green and blue markers at the ends) and processes it to create the desired special effects on the output video. This data stream is then sent to ADV7123 chip on board which produces the output stream for the VGA display.

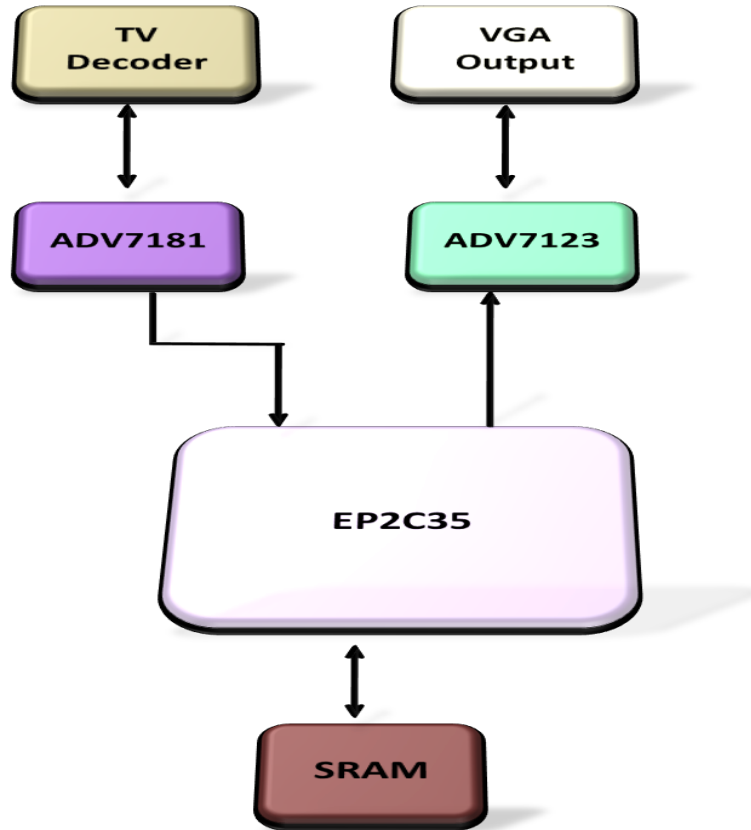


Figure 1 System Level Diagram

A kick start for the project was the DE2 Terasic TV Box Demonstration[1] example which displays real-time video out. We tried to comprehend this code which was useful in implementing those blocks in LSG responsible for providing video output.

2 TOP LEVEL SYSTEM DESIGN

The following figure shows the complete LSG system followed by a description of the data flow from the Camera input right up to the display.

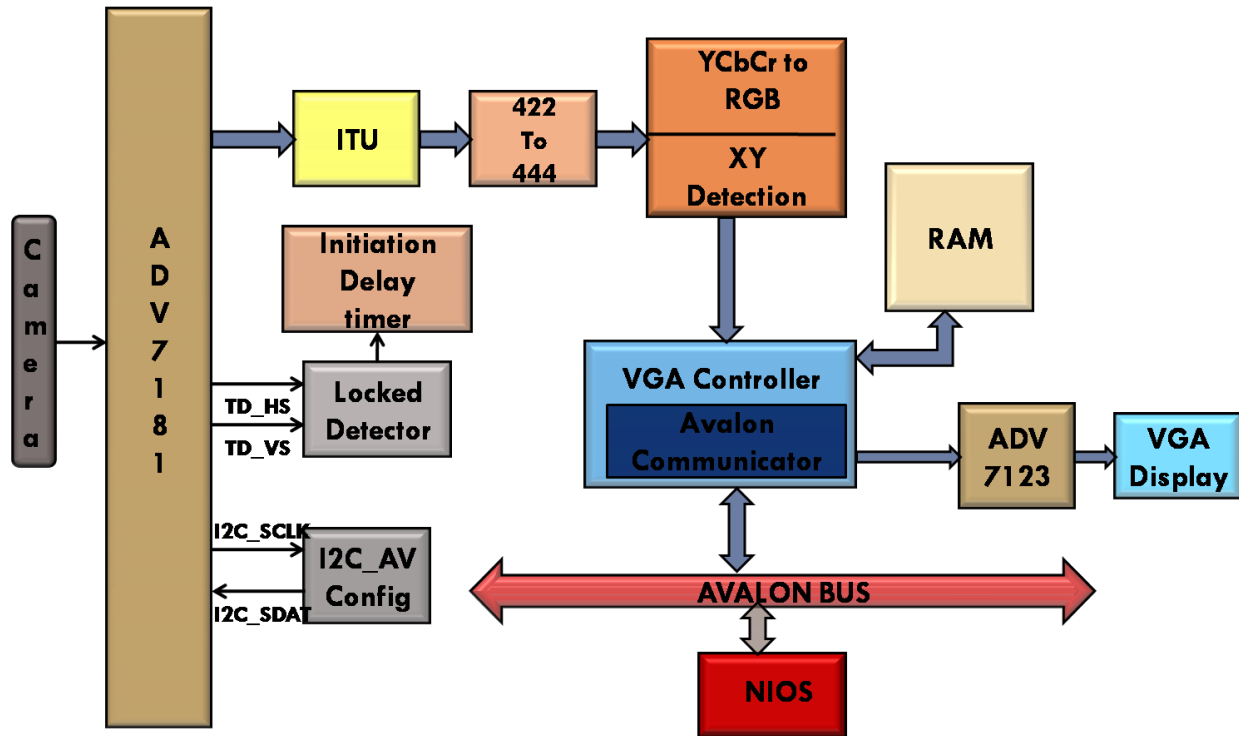


Figure 2 Top level Block Diagram

The data stream that comes out of the ADV7181 is a digital output 8 bits wide encoded in the ITU R 656 format. The ITU protocol builds upon the 4:2:2 digital video encoding parameters defined in ITU-R Recommendation BT.601, which provides interlaced video data, streaming each field separately, and uses the YCrCb color space and operates at 13.5 MHz sampling frequency for pixels.

This data is analyzed by the ITU decoder block shown in the figure which down-samples the incoming pixels and also extracts the required YCrCb color information. This color information is in 4:2:2 format and is converted to 4:4:4 format by the converter block shown. A color space conversion from YCrCb to RGB is necessary because the ADV7123 chip reads data in RGB color space only. The chip also needs horizontal sync signal for every line and vertical sync signal for each frame which are generated by the VGA block.

The YCrCb color information is also used by the XY detection module which recognizes the blue and green pixels in every line. This information is sent to the NIOS II processor through the Avalon bus and this communication is facilitated by the Avalon Communicator in VGA block.

The processor acts upon this information by computing the Centers of Mass of both green and blue markers on the sword. It uses this information to compute a lookup table for the coordinates of the edges of the sword for every line. This lookup table is then stored in a dual-ported RAM block in hardware through the Avalon bus. The VGA component refers to the lookup table in the RAM to modify the output to make a saber in each frame.

3 HARDWARE DESIGN

This section gives details about the various blocks involved in the LSG shown above zeroing on the functionality and operation of each module with timing diagrams wherever required.

3.1 I2C configuration:

The ADV7181 must be set up via the I₂C interface. Through the I2C 40 configuration registers are set which control the sync width, width of the back porch and front porch, whether the signals are active high /low. These registers are completely configured in hardware.

I2C Interface: the I2C interface comprises two lines -a clock, and a serial data line. Each write to a register in the ADV7181 happens in the following steps

- Send a START bit; this is done by pulling the data line low and then pulling the clock line low.
- Send the WRITE mode slave address with the SDATA being clocked by the SCLK line
- Receive a single bit ACK
- Send the register address (8 bits) on the SDATA line, again accompanied by the SCLK
- Receive a single bit ACK

3.2 ITU DECODER

3.2.1 Understanding the ITU Standard

The ITU-R BT 656 defines the parallel and serial interfaces for transmitting 4:2:2 YCrCb digital video between equipment. The active video resolutions are either 720X486 (525/60 video systems) or 720X576 (625/50 video systems). For the LSG we have used the NTSC format which is 525 lines with a frame rate of 60 frames/sec.

The BT 656 parallel interface uses 8 or 10 bits of multiplexed YCrCb data and 27MHz clock. Instead of conventional video timing signals (HSYNC, VSYNC and VBLANK) also being transmitted, BT 656 uses unique timing codes embedded within video stream.

3.2.2 YCrCb Stream

Each line of video is sampled at 13.5MHz generating 720 active samples of 24-bit 4:4:4 YCrCb data as shown in the figure below which is converted to 16-bit 4:2:2 YCrCb data resulting in 720 active samples of Y per line, and 360 active samples each of Cb and Cr per line. After each SAV code the stream of active

data words always begins with a Cb sample. The Y data and the CbCr data are multiplexed, and the 13.5MHz sample clock rate is increased by two times to 27MHz.

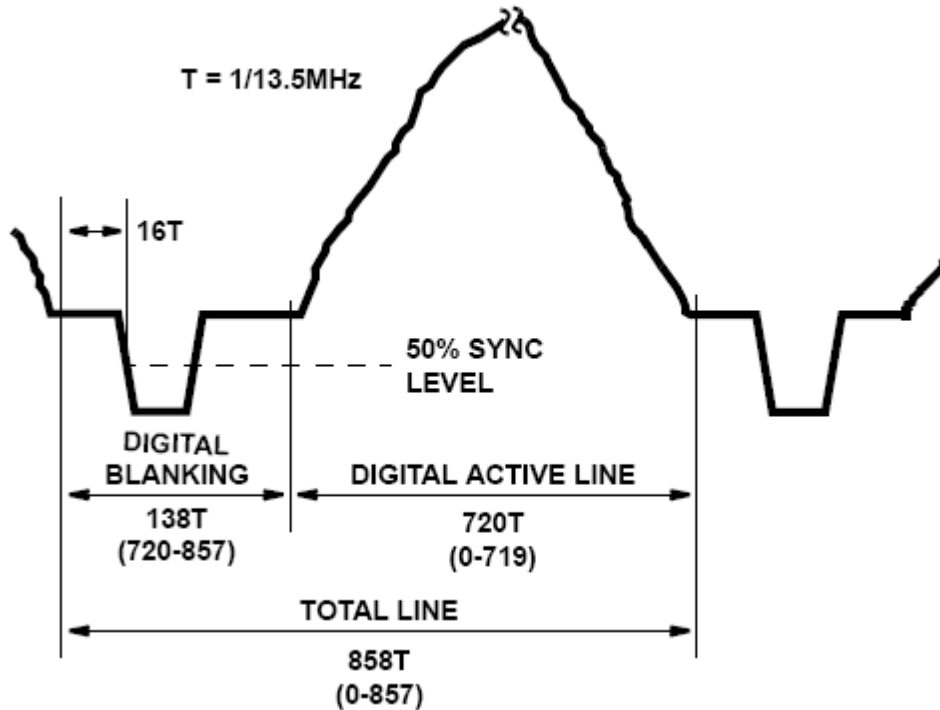


Figure 3 BT 656 Horizontal timing for 525/60 video system

The 4:2:2 YCrCb data is multiplexed into an 8-bit or 10-bit stream: Cb0Y0Cr0Y1Cb2Y2Cr2... etc. The following figure illustrates the format:

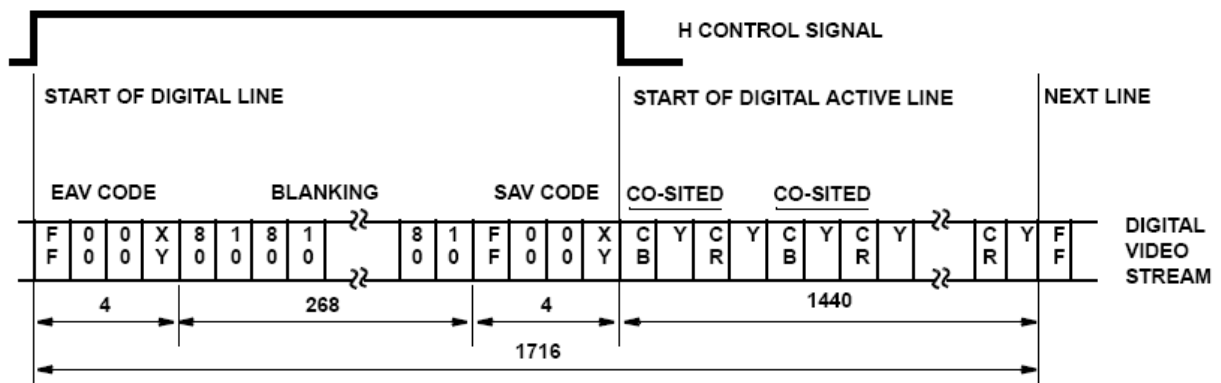


Figure 4 BT 656 8 bit parallel interface data format for 525/60 video system

3.2.3 SAV and EAV timing codes

SAV (start of active video) and EAV (end of active video) codes are embedded within the YCrCb video stream. The XY status word which also indicates whether it is an SAV or EAV sequence, is defined as:

F=0 for field 1; F=1 for field2

V=1 during vertical blanking

H_₀=0 at SAV, H=1 at EAV

P3-P0 =protection bits

	8-BIT DATA								10-BIT DATA	
	D9 (MSB)	D8	D7	D6	D5	D4	D3	D2	D1	D0
Preamble	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
Status Word	1	F	V	H	P3	P2	P1	P0	0	0

3.2.4 VERTICAL BLANKING INTERVALS

BT.656 uses the BT.601-defined vertical blanking intervals, as shown in following figure. Note that the active resolutions other than 720X486 and 720X576 may be supported (effectively cropping the image) by adjusting where the EAV and SAV codes and vertical blanking internals occur.

Note that in every field (even/odd), the active video is followed by a Blanking period.

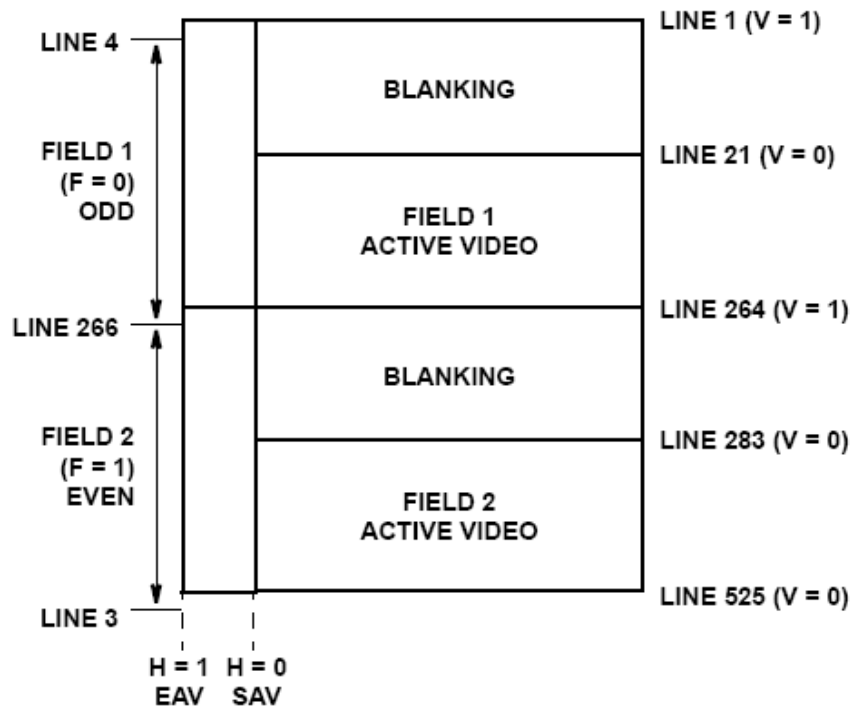


Figure 5 Bt 656 Vertical blanking interval for 525/60 video

3.2.5 VERILOG MODULE

The ITU decoder Verilog module that we have implemented does the following two things functionally:

1. Down sample the number of incoming pixels from 720 to 640 per each line.
2. Extract the YCrCb information from the incoming encoded video stream ONLY in Active region and output 16 bit color information for each pixel.
3. Also generate a "data valid" signal for every active data that we receive.

3.2.6 SLIDING WINDOW

To analyze the continuous data stream we utilize the concept of sliding window, where we define a window of 24 bits, i.e. 3 bytes. A 24-bit window is chosen as it aids us for two things:

- a. The SAV and EAV codes are both 4 bytes as shown in figure (). Of the 4 bytes the MSB 3 bytes are FF, 00, and 00. This facilitates in distinguishing between the control code and the active data.
- b. Also since the complete pixel information will be 24 bits, i.e., 1 byte of each Y, Cr, and Cb.

We append each incoming byte with the MSB 16 bits of the previous window for analysis and this helps us to hold each byte for 3 clock cycles which is sufficient time for the computation.

3.2.7 DOWN SAMPLE 720 to 640

For the VGA Display that we use, the active resolution is 640X480. Therefore, we need to down sample the number of incoming pixels from 720 to 640.

To do this we have used an algorithm, which eliminates the excess 80 pixels, by skipping every 9th pixel of the active video. By doing so we do not lose any color information since we only skip Y component each time.

3.2.8 ACTIVE VIDEO

To distinguish the control code (SAV/EAV) we first check if sliding window has a 24'hFF0000 and analyze the control word XY for its 5th and 6th bits that indicate the H and V values.

In order to check for active video, a series of conditions must be met:

- Valid frame?
- Start of frame?
- check SAV?
- check skip pixel?

By a logical AND check the validity of all these conditions and we only assert a data valid as 1 when all of them are met and we accept the current pixel as an active pixel.

3.2.9 XY DETECTION

The two markers at the ends of the swords must be recognized in order to determine the exact position of the sword in every frame. Green and blue colors were chosen for the two markers as they are least like the skin tone. This choice also imposes an implicit restriction on the usage of clothes or objects of the same shades of green and blue in the video.

Color detection is typically done by specifying a base color and accepting all colors within a small Euclidean distance to the base color. However, this method is computationally intensive and involves square and square root calculations. To avoid complexity of the hardware design, we chose to simply specify ranges for the blue and green shades.

The color detection can be done in YCrCb or the RGB color space. Initial experiments of color detection in the RGB color space were not too successful. This is because a single color can have a very wide range of values of R, G and B when exposed to different intensities of light. When the hue of a color is increased the R, G, B values do not change linearly. These problems made us shift to the YCrCb domain for color detection.

In the YCrCb color space, only the Y component changes with varying intensities of light on the color and the variations in Cb or Cr are minimal. By allowing a wide range for Y, we can detect the blue and green markers with varying light exposure (This happens very often while brandishing the sword). After a lot of experiments, these are the final ranges for the green and blue markers.

Green - $iY > 100 \ \&\& \ iCb < 120 \ \&\& \ iCr < 110$

Blue - $iY > 85 \ \&\& \ iCb > 140 \ \&\& \ iCr < 120$

Our original plan was to compute the Centers of mass of all the green pixels and blue pixels in hardware to pin-point the ends of the sword. But implementing even a simple algorithm to eliminate noise (stray blue/green pixels in the video) would increase the hardware complexity tremendously. Hence this task was left to be performed by the software. The XY detection unit only sends information about the number and position of blue/green pixels in every line to the software for it to do the rest of the computations.

There were a couple of ideas for communication of blue/green pixel info from XY Detector to the NIOS processor. One option is to send the blue/green information after every line using the Avalon communicator. The NIOS processor would use it in the center of mass computation as and when it reads this information. We had doubts regarding the synchronization aspect of this design as the hardware blocks function at 27MHz while the processor operates at 50MHz. The second option was to use a dual ported RAM where the XY detector would fill up the RAM after processing each line. The processor would read the information about the entire frame after the TS_VS signal goes high and then computes the centers of masses. Eventually we chose the first design idea because of the following limitation faced at the software end. The processor has the following tasks to complete between two frames - centre of mass calculation, light saber table computation, writing the table in a dual ported RAM through the Avalon bus. These tasks already take long enough that they do not complete during the time and TD_VS is high. Adding the task of reading the RAM for blue/green information during TD_VS would only worsen the problem and hence it was avoided.

3.2.10 YCrCb TO RGB CONVERTOR

The pixel information given by the ADV7181 chip is in 8 bit YCrCb format. In the YCrCb color space, Y is the luma component and Cb and Cr are the blue-difference and red-difference chroma components. The ADV7123 chip reads data only in 10 bit RGB format. Thus, it's essential to convert from 8 bit YCrCb format to 10 bit RGB format.

Following are the equations to convert from 8 bit YCrCb to 8 bit RGB.

$$B = 1.164(Y - 16) + 2.018(Cb - 128)$$

$$G = 1.164(Y - 16) - 0.813(Cr - 128) - 0.391(Cb - 128)$$

$$R = 1.164(Y - 16) + 1.596(Cr - 128)$$

The equations if implemented as they appear above clearly require floating point computations which must be avoided if possible. The equations are simplified as follows to avoid floating point arithmetic.

$$B = 1.164Y + 2.018Cb - 276.928$$

$$G = 1.164Y - 0.813Cr - 0.391Cb + 135.488$$

$$R = 1.164Y + 1.596Cr - 222.912$$

Multiplying and dividing the equations by 512 and approximating, we get

$$B = (596Y + 1033Cb) / 512 - 141787/512$$

$$G = (596Y - 416Cr - 200Cb) / 512 + 69370/512$$

$$R = (596Y + 817Cr) / 512 - 114131/512$$

The above equations have no floating point arithmetic. The multiplications are performed by the MAC unit. The division by 512 can be done right-shifting the value by 9. Note that the values obtained are in 8 bit RGB format. We need the data in 10 bit RGB format. This is done by a scaling factor of 4, R, G, B values are multiplied by 4 (left-shifted twice) to obtain the corresponding 10 bit RGB data.

3.2.11 HANDLING SPECIAL CASES

With the above scheme of YCrCb to RGB conversion, the output video was pretty clear. However, saturated colors in the input video had a very different shade in the output video. After we confirmed that the problem was with only saturated colors, we inferred that the problem may be related to overflow problems. Note that there may be an overflow in some of the multiplications or bits shifts described above. These problems were solved by clipping method. The final results of the computations are first stored in temporary registers which are wider than 10 bits. If any of the values in these registers are greater than 1023, they are clipped at 1023 before they are stored in the final 10 bit registers.

3.2.12 AVALON COMMUNICATOR

This block is embedded in the VGA block and is mainly used for the purpose of transferring data between hardware and software (NIOS II).

The Avalon Memory-Mapped (Avalon-MM) interface specification is designed to accommodate peripheral development for the system-on-a programmable-chip (SOPC) environment. The specification provides peripheral designers with a basis for describing the address-based read/write interface found on master and slave peripherals, such as microprocessors, memory, UART, timer, etc.

The Avalon-MM interface defines:

- A set of signal types
- The behavior of these signals
- The types of transfers supported by these signals

The communication process with NIOS II takes place at 50MHz as the NIOS processor functions typically at that frequency.

In our system, since all the hardware modules run at 27MHz whereas the NIOS processor runs at 50MHz, we have used a dual-ported RAMs into which we store the values before we transmit/after receiving.

3.2.13 WRITE TRANSFERS

The data from software is communicated to the hardware using the “writedata” signal of the Avalon-MM interface. In order to transfer data we must also assert the “chipselect” signal high and the hardware asserts the “write” signal high by initiating a write call.

The Software computes the X1 and X2 co-ordinates (outer ends of sword - halo) and inner_X1 and inner_X2 co-ordinates (inner ends of sword – solid white saber)of the saber for each line of the previous and sends it to the VGA for display in the TD_VS time, i.e. the during the vertical sync blanking period. Every time we receive these values, we store them in the RAM to avoid loss of information between transfers.

3.2.14 READ TRANSFERS

The data from the hardware is communicated to the software using the “readdata” signal of the Avalon-MM interface. In order to transfer data we must also assert the “chipselect” signal high and the software asserts the “read” signal high by initiating a read call.

3.2.15 SYNCHRONIZATION

The synchronization of various transfers between software and hardware is very important in our design, mainly because we operate at 2 different frequencies. Another important factor that demands synchronization is the process of generating the video as it needs horizontal sync for every line and a vertical sync for every frame.

If the data does not arrive/is not transmitted at a requested time instant then it might lead to the following problems:

- The data may be old or may interrupt the current frame and lead to junk being displayed
- The data may overflow into the next frame information and this may lead into a cycle, if this were to happen it would lead to the chain problem in every frame being displayed

Therefore, in order to for the synchronization of data between hardware and software we also send the TD_HS,TD_VS, VGA_HS, and VGA_VS (described in later part of the report).

We also transmit the number of blue/green pixels per line and the co-ordinates X1 and X2 for each line and also the line count information.

3.3 VGA Unit

3.3.1 BASIC DESIGN

This unit is the master controller of Video Generation. It is responsible for sending the Blanking and the Sync signals to the ADV7123 chip along with a clock and R,G,B data.

3.3.2 DESIGN DECISION

The ITU Decoder sends interlaced video by sending the odd field and even field alternately. A TD_VS signal is asserted between two fields. There are a number of methods to deinterlace the video. In field combination deinterlacing, weaving is done by adding consecutive fields together. Weaving requires a frame buffer. Using the SDRAM as a frame buffer may slow down the LSG due to its slow access time. Using an SRAM too has its disadvantages, since it is not dual ported. Writing the frame into the SRAM and reading from it has to be sequential which can act as a bottleneck. For these reasons, field combination interlacing was not used.

The other method of deinterlacing is field extension deinterlacing. In this method, only one field is displayed at a time. Hence the VGA_VS signal must have the same frequency as TD_VS. Single field display is done either by reducing the vertical resolution to half or by displaying each line twice. We used the latter of the two options to main the vertical resolution. In order to avoid any loss of information, we need to be able to display a line twice in the same amount of time that it takes to receive one line. Hence the VGA_HS signal must have twice the frequency of TD_HS. This method of display necessitates two line buffers of length 640 and data width. The relevant timing diagram is shown below. When Line buffer1 was being filled the VGA would read from the Line buffer2, they would switch on every TD_HS going high.

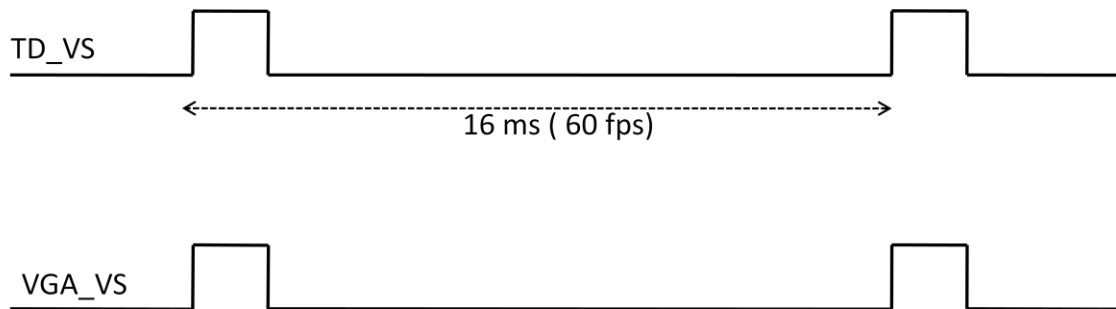
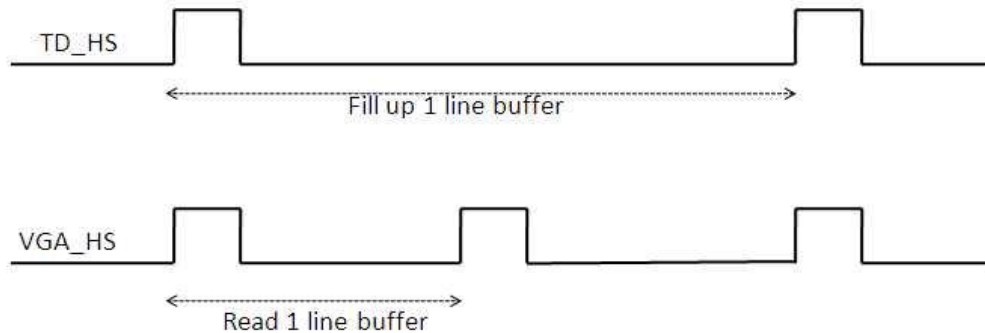


Figure 6 Timing diagram of TD_VS and VGA_VS

3.3.3 PROBLEMS FACED AND SOLUTIONS

It was quite a challenge to get generate the horizontal and vertical sync signals. Apart from achieving the desired frequencies for these signals, it was also important that these signals get aligned with the incoming TD_HS and TD_VS signals. It was found using a cathode ray oscilloscope that the width of the vertical sync received (TD_VS) was 60 lines instead of 2 lines. Hence edge detection circuit was used and the sync of the required width was created using counters.

Another problem was that the TD_HS has the frequency of 2 lines. Hence accurate counters were required to place an extra sync signal of the required width exactly in the center of 2 TD_HS signals



MODIFIED DISPLAY FOR LIGHT SABER

The RAM block is the lookup table for drawing the light saber. For every line, if edge co-ordinates of sword are present in the corresponding address of the RAM, the VGA draws a light saber between the coordinates. To emulate the light saber shown in the Star Wars series, we wanted our saber to have a bright white core and a transparent green glow around it. The white core was pretty straightforward ($R=G=B=1023$). The green halo around it was achieved by adding a significant component of green to the existing color pixel data. This gives it a translucent effect.

4 SOFTWARE SYSTEM

4.1 INTRODUCTION

The NIOS II processor family uses a 32-bit RISC architecture. The instance that it is used in this project is the Nios II/f processor, clocked at 50 MHz and attached to an instruction cache of 4 KB and a data Cache of 2 KB. Also, the processor is built with hardware multiplication and hardware division units along with a dynamic Branch Prediction and barrel Shifter logic. The entire detection of the centre of mass and creation of the light saber is performed in software. We rely on the fact that the human eye cannot distinguish the one frame delay between the detection of the center of mass and the display of the light saber. That clearly makes it evident that the time available to the software for detection of the markers and creation of the light saber is 1 frame time (i.e. 16 ms). The Design is very time critical and at every stage of the software an effort has been made to optimize the design to satisfy the timing requirements. Another important point to note would be that the hardware provides only 263 lines between 2 vertical syncs (lines per field) these have been duplicated to provide a 525 lines in the VGA unit. Hence the Y coordinate of the center of mass should be doubled.

4.2 IDENTIFICATION OF CENTER OF MASS

The marker position module determines position of the marker by taking the average of the position of all the marker pixels. For each line the last marker pixel seen with the no. of marker pixels is recorded.

The ideal method to detect the center of mass would be

$$X_{center} = \sum X/N$$

$$Y_{center} = \sum Y/N$$

Where, N is the no of marker pixels.

But this method is computationally intensive and doesn't satisfy the timing requirement. Hence for each line we record the mean of the X coordinates and average the means for all the lines at the end of the frame. For Y coordinates we implemented a less reliable but computationally less intensive method. We record the line count of the first and the last line of the marker and average it out at the end of the frame.

4.2.1 Routine

```
LineCount= ReadLightSaber(CORE_NIOS_BASE,26);
```

```
//record the marker pixels when the count changes
```

```
if(LineCount!=prevlc)
```

```
{
```

```
    //blue Detection
```

```
    b1.count= ReadLightSaber(CORE_NIOS_BASE,8);//get the no. of blue pixel for line
```

```
    X_blue = ReadLightSaber(CORE_NIOS_BASE,10);// read the X coordinate for last  
                                                Blue pixel on the line
```

```
    b1.x1=X_blue-b1.count;
```

```
    b1.x2=X_blue;
```

```
    //Green Detection
```

```
    g1.count= ReadLightSaber(CORE_NIOS_BASE,12);
```

```
    X_green = ReadLightSaber(CORE_NIOS_BASE,14);
```

```
    g1.x1=X_green-g1.count;
```

```
    g1.x2=X_green;
```

```
}
```

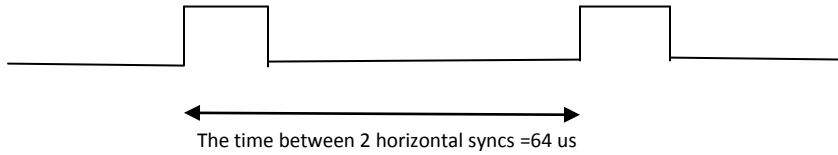
For every change in the line count we record the no of marker pixels with the last marker pixel seen. The x1 and x2 coordinates of the marker pixels are calculated as

$$x2 = X_{\text{last marker pixel}}$$

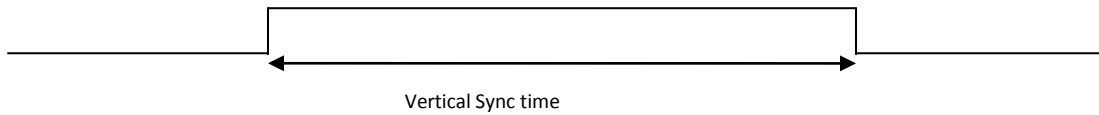
$$x1 = X_{\text{last marker pixel}} - \text{no of marker pixels.}$$

The mean for each line is calculated as $(x1+x2) / 2$.

4.2.2 Timing Requirement:



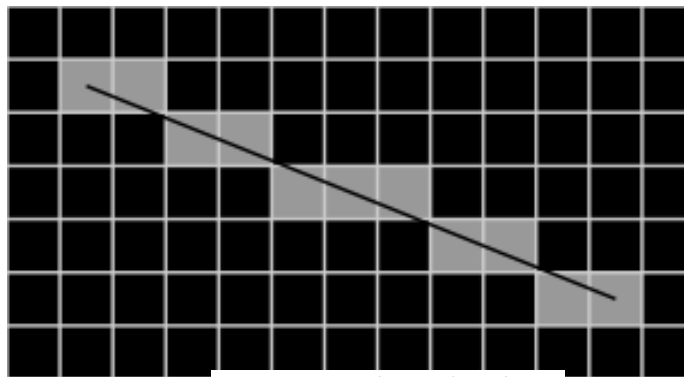
In the above time we record the means for the X coordinate and sum it to means of the previous line.



In the vertical sync time we do the averaging to find the center of mass of the markers. The centers of mass of the 2 markers are given to the light saber calculation routine which returns an array of with light saber coordinates. These values are written into ram in hardware.

4.2.3 LIGHT SABER CALCULATION

We first used the normal line equations to make the light saber. The computation involved floating point arithmetic in a loop. Floating point arithmetic proved to be costly in the time domain. It took software 50 seconds to draw the light saber on the screen! Therefore we explored line drawing algorithms in the graphics. The algorithm most widely used is the Bresenham's line algorithm.



4.2.3.1

4.2.3.2

Figure 7 Breesenhams Algorithm

4.2.3.3 Algorithm: To draw a line given 2 points.

```
function line(x0, x1, y0, y1)
  boolean steep := abs (y1 - y0) > abs(x1 - x0)
  if steep then
    swap(x0, y0)
    swap(x1, y1)
  if x0 > x1 then
    swap(x0, x1)
    swap(y0, y1)
```

```
int deltax := x1 - x0
int deltax := abs(y1 - y0)
int error := deltax / 2
int ystep
int y := y0
if y0 < y1 then ystep := 1 else ystep := -1
for x from x0 to x1
  if step then plot(y,x) else plot(x,y)
  error := error - deltax
  if error < 0 then
    y := y + ystep
    error := error + deltax
```

The algorithm can track, instead of possibly large y values, a small *error value* between -0.5 and 0.5 : the vertical distance between the rounded and the exact y values for the current x . Each time x is increased, the error is increased by the slope; if it exceeds 0.5 , the rasterization y is increased by 1 (the line continues on the next lower row of the raster) and the error is decremented by 1.0

We use the algorithm to join the 4 points of the light saber.

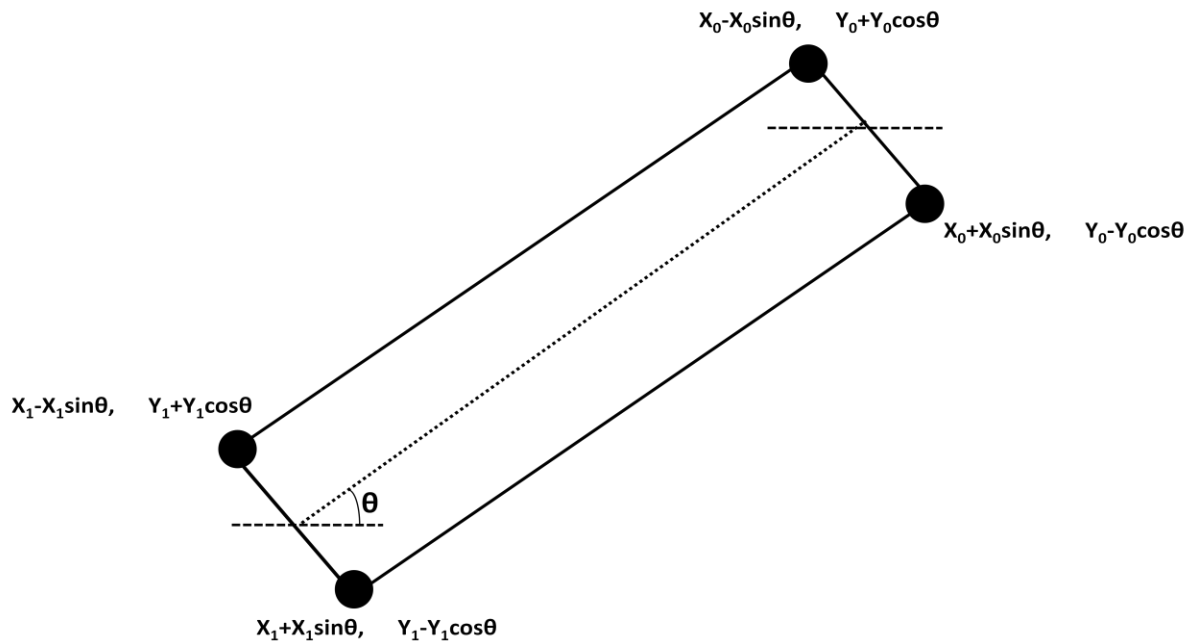


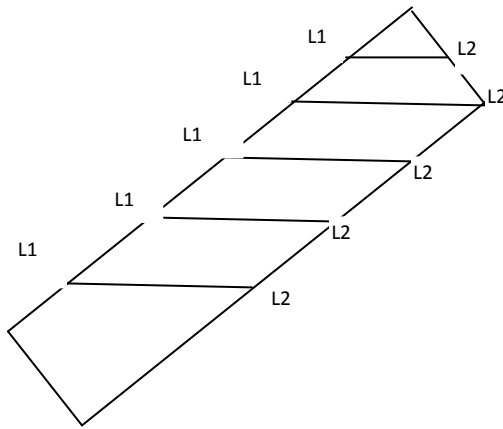
Figure 8 tracking coordinates for light saber

4.2.4 OPTIMIZATION

For the calculation of the θ we use fixed point calculation instead of floating point. For finding the square root we use bit shifting.

4.2.5 FINAL OUTPUT

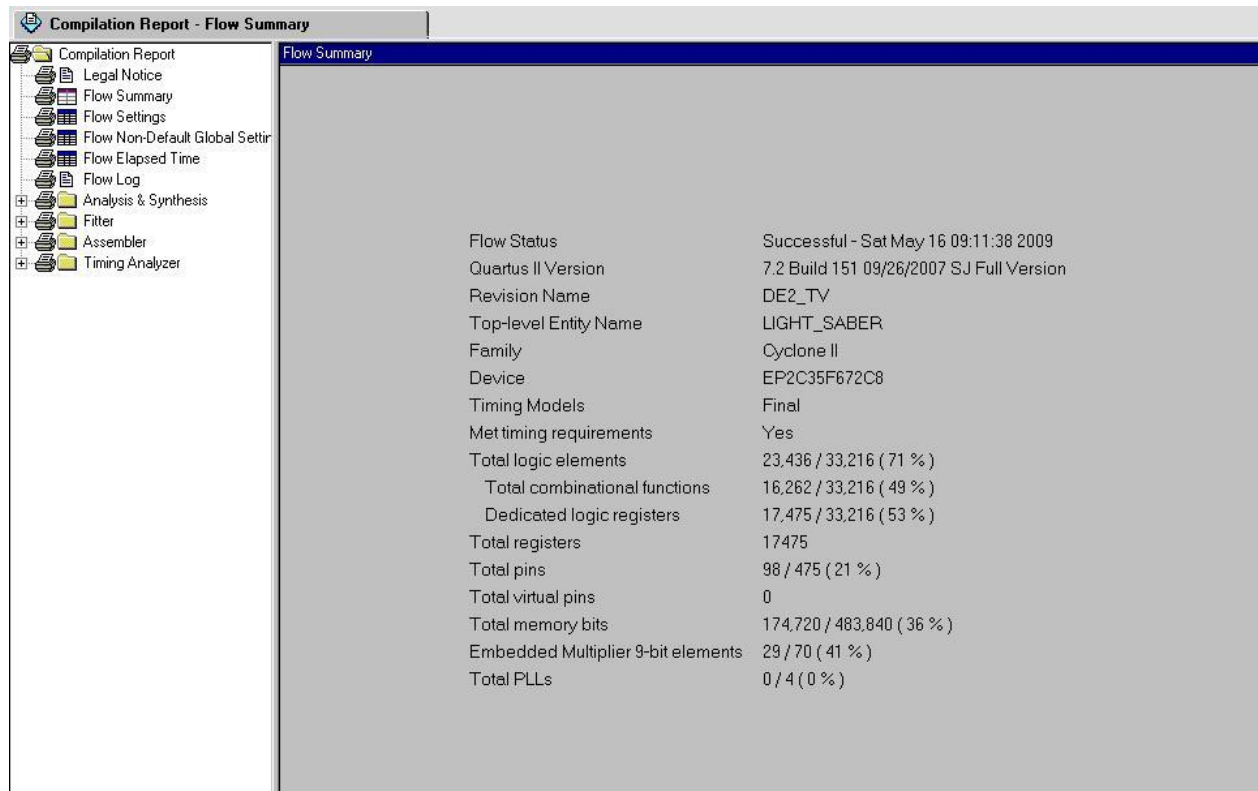
The software must provide the hardware with the boundaries between which the light saber must be drawn per line. A table (an array) is created to store L1 and L2 values for the line.



Line No (Table Index)	L1	L2
1		
2		
.....		
480		

5 RESOURCE UTILIZATION

Shown below is the compilation summary for the project which gives a description on the resource utilization by the LSG



Flow Status	Successful - Sat May 16 09:11:38 2009
Quartus II Version	7.2 Build 151 09/26/2007 SJ Full Version
Revision Name	DE2_TV
Top-level Entity Name	LIGHT_SABER
Family	Cyclone II
Device	EP2C35F672C8
Timing Models	Final
Met timing requirements	Yes
Total logic elements	23,436 / 33,216 (71 %)
Total combinational functions	16,262 / 33,216 (49 %)
Dedicated logic registers	17,475 / 33,216 (53 %)
Total registers	17475
Total pins	98 / 475 (21 %)
Total virtual pins	0
Total memory bits	174,720 / 483,840 (36 %)
Embedded Multiplier 9-bit elements	29 / 70 (41 %)
Total PLLs	0 / 4 (0 %)

6 DESIGN EXPERIENCES

6.1 CHALLENGES FACED IN HARDWARE

6.1.1 Synchronization:

The ADV7123 needs accurate frequencies for the sync signals. Achieving these frequencies was difficult and proved to be a road-block, as no video processing could be done without getting the real-time video working. This was solved by generating local HS and VS signals in the VGA controller (described in VGA)

6.1.2 Compilation Delay:

The ROM type memory which was used for line buffers in VGA controller lead to a very high compilation time and this occasionally discouraging as we could not proceed without analyzing the results of the current compilation. During the later stages we figured that using a RAM in place of the ROM type memory would serve the same purpose but with a lesser compilation time.

6.1.3 Multiple Clock Domains:

Our design demanded the use of two different clock frequencies – 27 MHz and 50MHz, as all our hardware components were functional at 27MHz while we wanted an operating frequency of 50MHz for the NIOS processor. Our understanding of the Avalon bus signals in detail helped us successfully set up two different clocks in LSG

6.1.4 SOFTWARE DESIGN CHALLENGES

6.1.5 Floating point computation:

In the software design, calculations involving the slope and width of the saber required floating point arithmetic. But this had to be avoided at any cost to generate the saber at real-time. Hence, we converted all the computations into fixed-point with the help of Bresenham's Line Drawing Algorithm.

6.1.6 Sampling signals(from hardware) at higher frequency:

Since the hardware functions at a frequency lower than the NIOS processor operating frequency, we faced the problem of software sampling hardware signals multiple times. In order to avoid this, we set up flags which would be set in hardware but reset in software

7 PITFALLS AND SUGGESTIONS

7.1.1 Color Detection:

Our color detection is slightly flawed as it can pick up a few shades of blue that are unwanted. Use of Euclidean distance calculations would have made the color detection more robust.

7.1.2 Computational delay in software:

The software has to perform final centre of mass calculation, line drawing algorithm and writing into the RAM – all in the vertical blank time. Failure to do so would lead to some of these computations overflowing into active display time and hence affect the quality of output video with saber. One optimization that can be used to save time is sending X coordinate information only to those lines where the saber is present. This would effectively save some time spent in unnecessary write operations.

8 LESSONS LEARNT

1. Gain a thorough understanding of all the peripheral ICs involved in the design. In our project, an accurate understanding of the ADV7123 chip at an earlier stage would have helped us save time. Ignorant of the fact that the output data stream would be sent to a DAC before displaying, we constrained the VGA unit's clock frequency to 25MHz which gave us many problems integrating it with the rest of the system.
2. RAMs help in isolating processes functional at different clock frequencies and set up a reliable communication system.
3. Watch out for carries and overflow bits in all computations. If ignored, the error may propagate and manifest itself in a way which is not suggestive of the root of the bug.
4. Oscilloscope – can prove to be a better debugging tool than simulators in some cases. In our project, while generating the synchronization signals, their frequencies observed on the CRO were the last resort as the video was not comprehensible.
5. When multiple clocks used in the design are declared as 'clock' in the SOPC builder, it maps all of them to a single signal in the generated file. In order to avoid this, one of the clocks must be an 'export' signal.

9 PICTURES of the JEDI



10 TASK DIVISION

Our project was more of group work than individual effort. Brain storming in the group on various design decisions to be made helped us get rid of some of the pitfalls that we may have run into individually.

However, once we overcame the challenges of displaying video in real-time we split into two teams. Anusha and Roopa worked on hardware blocks while Devesh and Raghu worked on software routines.

11 ACKNOWLEDGEMENT

Firstly we would like to thank Prof. Edwards for his valuable suggestions and giving us the right pointers at every major hurdle in our design. We are grateful to the Teaching assistants Nalini and Sung jun for their support.

We would like to thank a few of our seniors whose reports and project files were extremely helpful.

12 REFERENCES

- [1] BT. 656 Video interface for ICs – Intersil corporation
<http://www.intersil.com/data/an/an9728.pdf>
- [2] DE2 Development and Educational Board User Manual – Altera Corporation
- [3] Altera documentation (Found on course 4840 webpage)
<http://www1.cs.columbia.edu/~sedwards/classes/2009/4840/index.html>
- [4] Previous Semester Projects (Found on Course 4840 webpage)
<http://www1.cs.columbia.edu/~sedwards/classes/2008/4840/index.html>

13 APPENDIX

HARDWARE BLOCKS CODE

```
--  
-- DE2 top-level module that includes the simple VGA raster generator  
--  
-- Stephen A. Edwards, Columbia University, sedwards@cs.columbia.edu  
--  
-- From an original by Terasic Technology, Inc.  
-- (DE2_TOP.v, part of the DE2 system board CD supplied by Altera)  
-- Modified by : Anusha Dachepally  
--           Devesh Dedhia  
--           Raghu Binnamangalam  
--           Roopa Karkarlapudi  
--  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.numeric_std.all;  
  
entity LIGHT_SABER is  
  
    port (  
        -- Clocks  
        OSC_50,  
            OSC_27,                -- 27 MHz  
        EXT_CLOCK : in std_logic;  -- External Clock
```

```
KEY: in std_logic_vector(3 downto 0);
-- VGA output

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

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

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

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

end LIGHT_SABER;

architecture datapath of LIGHT_SABER is

signal I2C_SDAT1 : std_logic := I2C_SDAT;
signal iY_core, iCb_core, iCr_core : std_logic_vector(7 downto 0);
signal tempaddr : std_logic_vector(9 downto 0);
signal X_blue,blue_count: std_logic_vector(9 downto 0);
```

```

component I2C_AV_Config port(      --      Host Side
    iCLK,
    iRST_N : in std_logic;
    --      I2C Side
    I2C_SCLK : out std_logic;
    I2C_SDAT : inout std_logic);

end component;

begin

V0: I2C_AV_Config port map(  --      Host Side
    iCLK => OSC_27,
    iRST_N => '1',
    --      I2C Side
    I2C_SCLK => I2C_SCLK,
    I2C_SDAT => I2C_SDAT1);

V4: entity work.nios_top port map (
    clk => OSC_50, --
    reset_n => '1', --
    KEY_core_to_the_CORE_nios => KEY,
    clk27_to_the_CORE_nios => OSC_27,

    -- VGA output
    VGA_CLK_core_from_the_CORE_nios => VGA_CLK,
-- Clock
    VGA_HS_core_from_the_CORE_nios => VGA_HS,
-- H_SYNC
    VGA_VS_core_from_the_CORE_nios => VGA_VS,
-- V_SYNC
    VGA_BLANK_core_from_the_CORE_nios =>
VGA_BLANK,      -- BLANK
    VGA_SYNC_core_from_the_CORE_nios => VGA_SYNC,
-- SYNC
    VGA_R_core_from_the_CORE_nios => VGA_R,
-- Red[9:0]
    VGA_G_core_from_the_CORE_nios => VGA_G,
-- Green[9:0]
    VGA_B_core_from_the_CORE_nios => VGA_B,
-- Blue[9:0]
    -- TD ports
    TD_DATA_core_to_the_CORE_nios => TD_DATA,
--TV Decoder Data bus 8 bits

```

```
TD_RESET_core_from_the_CORE_nios => TD_RESET,
TD_HS_core_to_the_CORE_nios => TD_HS,
    --      TV Decoder H_SYNC
TD_VS_core_to_the_CORE_nios => TD_VS,
Linecount_core_from_the_CORE_nios => tempaddr,
bluecount_from_the_CORE_nios => blue_count,
x_blue_from_the_CORE_nios => X_blue,
--      oCb_from_the_CORE_nios => iCb_core,
--      oCr_from_the_CORE_nios => iCr_core,
--      oY_from_the_CORE_nios => iY_core,

--      clk27_to_the_XY_GEN_nios => OSC_27,
--      KEY_XY_to_the_XY_GEN_nios => KEY,
--      TD_HS_to_the_XY_GEN_nios => TD_HS,
--      TD_VS_to_the_XY_GEN_nios => TD_VS,
--      iY_to_the_XY_GEN_nios => iY_core,
--      iCb_to_the_XY_GEN_nios => iCb_core,
--      iCr_to_the_XY_GEN_nios => iCr_core,

clk27_to_the_Blue_inst => OSC_27,
TD_HS_to_the_Blue_inst => TD_HS,
tempaddr_to_the_Blue_inst => tempaddr,
X_blue_to_the_Blue_inst => X_blue,
blue_count_to_the_Blue_inst => blue_count,

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

);

--v12: entity work.XY_GEN port map (
--  reset_n => '1',
--  clk27_to_the_XY_GEN_nios => clk27,
--  clk => CLK50,
```

```
-- KEY_XY_to_the_XY_GEN_nios => KEY,  
-- TD_HS_to_the_XY_GEN_nios => TD_HS,  
-- TD_VS_to_the_XY_GEN_nios => TD_VS,  
-- iY_to_the_XY_GEN_nios => iY_core,  
-- iCb_to_the_XY_GEN_nios => iCb_core,  
-- iCr_to_the_XY_GEN_nios => iCr_core  
--     -- Avalon_signals  
--);
```

end datapath;

```
-- Modified by : Anusha Dacheppally  
--     Devesh Dedhia  
--     Raghu Binnamangalam  
--     Roopa Karkarlapudi  
--
```

```
module I2C_AV_Config (      //      Host Side  
                        iCLK,  
                        iRST_N,  
                        //      I2C Side  
                        I2C_SCLK,  
                        I2C_SDAT      );  
  
//      Host Side  
input      iCLK;  
input      iRST_N;  
//      I2C Side  
output     I2C_SCLK;  
inout      I2C_SDAT;  
//      Internal Registers/Wires  
reg [15:0] mI2C_CLK_DIV;  
reg [23:0] mI2C_DATA;  
reg      mI2C_CTRL_CLK;  
reg      mI2C_GO;  
wire     mI2C_END;  
wire     mI2C_ACK;  
reg [15:0] LUT_DATA;  
reg [5:0]  LUT_INDEX;  
reg [3:0]  mSetup_ST;
```

```
//      Clock Setting
parameter  CLK_Freq      =      50000000;    //      50      MHz
parameter  I2C_Freq      =      20000;      //      20      KHz
//      LUT Data Number
parameter  LUT_SIZE      =      51;
//      Audio Data Index
parameter  Dummy_DATA    =      0;
parameter  SET_LIN_L     =      1;
parameter  SET_LIN_R     =      2;
parameter  SET_HEAD_L    =      3;
parameter  SET_HEAD_R    =      4;
parameter  A_PATH_CTRL   =      5;
parameter  D_PATH_CTRL   =      6;
parameter  POWER_ON      =      7;
parameter  SET_FORMAT    =      8;
parameter  SAMPLE_CTRL   =      9;
parameter  SET_ACTIVE    =      10;
//      Video Data Index
parameter  SET_VIDEO     =      11;
```

```
//////////////////////////////////// I2C Control Clock //////////////////////////////////////
always@(posedge iCLK or negedge iRST_N)
begin
```

```
    if(!iRST_N)
    begin
        mI2C_CTRL_CLK<= 0;
        mI2C_CLK_DIV <= 0;
    end
    else
    begin
        if( mI2C_CLK_DIV< (CLK_Freq/I2C_Freq) )
        mI2C_CLK_DIV <= mI2C_CLK_DIV+1;
        else
        begin
            mI2C_CLK_DIV <= 0;
            mI2C_CTRL_CLK<= ~mI2C_CTRL_CLK;
        end
    end
end
```

```
end
////////////////////////////////////
I2C_Controller u0 ( .CLOCK(mI2C_CTRL_CLK), // Controller Work Clock
```

```

.I2C_SCLK(I2C_SCLK), // I2C CLOCK
.I2C_SDAT(I2C_SDAT), // I2C DATA
.I2C_DATA(mI2C_DATA), //
DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
.GO(mI2C_GO), // GO transfor
.END(mI2C_END), // END

transfor

.ACK(mI2C_ACK), // ACK
.RESET(iRST_N );

////////////////////////////////////
//////////////////////////////////// Config Control //////////////////////////////////////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
begin
    if(!iRST_N)
    begin
        LUT_INDEX    <=    0;
        mSetup_ST    <=    0;
        mI2C_GO      <=    0;
    end
    else
    begin
        if(LUT_INDEX<LUT_SIZE)
        begin
            case(mSetup_ST)
            0:    begin
                    if(LUT_INDEX<SET_VIDEO)
                    mI2C_DATA    <=    {8'h34,LUT_DATA};
                    else
                    mI2C_DATA    <=    {8'h40,LUT_DATA};
                    mI2C_GO      <=    1;
                    mSetup_ST    <=    1;
                end
            1:    begin
                    if(mI2C_END)
                    begin
                        if(!mI2C_ACK)
                        mSetup_ST    <=    2;
                        else
                        mSetup_ST    <=0;

                        mI2C_GO      <=    0;
                    end
                end
            end
        end
    end
end

```

```
                end
            2:    begin
                    LUT_INDEX    <=    LUT_INDEX+1;
                    mSetup_ST    <=    0;
                end
            endcase
        end
    end
end

////////////////////////////////////
////////////////////////////////////    Config Data LUT    //////////////////////////////////////
always
begin
    case(LUT_INDEX)
        //    Audio Config Data
        Dummy_DATA    :    LUT_DATA    <=    16'h0000;
        SET_LIN_L    :    LUT_DATA    <=    16'h001A;
        SET_LIN_R    :    LUT_DATA    <=    16'h021A;
        SET_HEAD_L    :    LUT_DATA    <=    16'h047B;
        SET_HEAD_R    :    LUT_DATA    <=    16'h067B;
        A_PATH_CTRL    :    LUT_DATA    <=    16'h08F8;
        D_PATH_CTRL    :    LUT_DATA    <=    16'h0A06;
        POWER_ON    :    LUT_DATA    <=    16'h0C00;
        SET_FORMAT    :    LUT_DATA    <=    16'h0E01;
        SAMPLE_CTRL    :    LUT_DATA    <=    16'h1002;
        SET_ACTIVE    :    LUT_DATA    <=    16'h1201;
        //    Video Config Data
        SET_VIDEO+0    :    LUT_DATA    <=    16'h1500;
        SET_VIDEO+1    :    LUT_DATA    <=    16'h1741;
        SET_VIDEO+2    :    LUT_DATA    <=    16'h3a16;
        SET_VIDEO+3    :    LUT_DATA    <=    16'h5004;
        SET_VIDEO+4    :    LUT_DATA    <=    16'hc305;
        SET_VIDEO+5    :    LUT_DATA    <=    16'hc480;
        SET_VIDEO+6    :    LUT_DATA    <=    16'h0e80;
        SET_VIDEO+7    :    LUT_DATA    <=    16'h5020;
        SET_VIDEO+8    :    LUT_DATA    <=    16'h5218;
        SET_VIDEO+9    :    LUT_DATA    <=    16'h58ed;
        SET_VIDEO+10    :    LUT_DATA    <=    16'h77c5;
        SET_VIDEO+11    :    LUT_DATA    <=    16'h7c93;
        SET_VIDEO+12    :    LUT_DATA    <=    16'h7d00;
        SET_VIDEO+13    :    LUT_DATA    <=    16'hd048;
        SET_VIDEO+14    :    LUT_DATA    <=    16'hd5a0;
```



```
SET_VIDEO+15 : LUT_DATA <= 16'hd7ea;
SET_VIDEO+16 : LUT_DATA <= 16'he43e;
SET_VIDEO+17 : LUT_DATA <= 16'hea0f;
SET_VIDEO+18 : LUT_DATA <= 16'h3112;
SET_VIDEO+19 : LUT_DATA <= 16'h3281;
SET_VIDEO+20 : LUT_DATA <= 16'h3384;
SET_VIDEO+21 : LUT_DATA <= 16'h37A0;
SET_VIDEO+22 : LUT_DATA <= 16'he580;
SET_VIDEO+23 : LUT_DATA <= 16'he603;
SET_VIDEO+24 : LUT_DATA <= 16'he785;
SET_VIDEO+25 : LUT_DATA <= 16'h5000;
SET_VIDEO+26 : LUT_DATA <= 16'h5100;
SET_VIDEO+27 : LUT_DATA <= 16'h0050;
SET_VIDEO+28 : LUT_DATA <= 16'h1000;
SET_VIDEO+29 : LUT_DATA <= 16'h0402;
SET_VIDEO+30 : LUT_DATA <= 16'h0860;
SET_VIDEO+31 : LUT_DATA <= 16'h0a18;
SET_VIDEO+32 : LUT_DATA <= 16'h1100;
SET_VIDEO+33 : LUT_DATA <= 16'h2b00;
SET_VIDEO+34 : LUT_DATA <= 16'h2c8c;
SET_VIDEO+35 : LUT_DATA <= 16'h2df8;
SET_VIDEO+36 : LUT_DATA <= 16'h2eee;
SET_VIDEO+37 : LUT_DATA <= 16'h2ff4;
SET_VIDEO+38 : LUT_DATA <= 16'h30d2;
SET_VIDEO+39 : LUT_DATA <= 16'h0e05;
default:LUT_DATA <= 16'h0000;
endcase
end
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
endmodule
```

```
--
-- Authors : Anusha Dacheppally
--           Devesh Dedhia
--           Raghu Binnamangalam
--           Roopa Karkarlapudi
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity CORE is
```

```

port (

    clk27 : in std_logic;
    reset_n : in std_logic;
    CLK50 : in std_logic;
    KEY_core : in std_logic_vector (3 downto 0);

-- VGA output
    VGA_CLK_core,           -- Clock
    VGA_HS_core,           -- H_SYNC
    VGA_VS_core,           -- V_SYNC
    VGA_BLANK_core,       -- BLANK
    VGA_SYNC_core : out std_logic;      -- SYNC
    VGA_R_core,           -- Red[9:0]
    VGA_G_core,           -- Green[9:0]
    VGA_B_core : out std_logic_vector(9 downto 0);      -- Blue[9:0]

    TD_DATA_core : in std_logic_vector(7 downto 0);      --TV
Decoder Data bus 8 bits
    TD_RESET_core: out std_logic;
    TD_HS_core,           --      TV      Decoder
H_SYNC
    TD_VS_core:   in std_logic;
--      TV Decoder V_SYNC
--TV Decoder Reset
--oY, oCb, oCr : out std_logic_vector(7 downto 0);      --XY_Gen
    Linecount_core : out std_logic_vector(9 downto 0);
    x_blue : out std_logic_vector(9 downto 0);
    bluecount : out std_logic_vector(9 downto 0);
-- Avalon_signals

    signal address : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    signal chipselect : IN STD_LOGIC;
    signal read : IN STD_LOGIC;
    signal write : IN STD_LOGIC;
    signal writedata : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    signal readdata : OUT STD_LOGIC_VECTOR (15 DOWNT0 0)

);

end CORE;
```

architecture datapath of CORE is

```
signal globalvalid: std_logic;
signal remain_sig: std_logic;
signal YCbCr : std_logic_vector (23 downto 0);
signal TV_X : std_logic_vector (9 downto 0);
signal TV_DVAL : std_logic;
signal DLY1 : std_logic;
signal DLY2,TD_Stable : std_logic;
signal quotient : std_logic_vector(9 downto 0);
signal DLY0 : std_logic;
signal remain : std_logic_vector ( 3 downto 0);
signal vga_vsync_edge, TD_HS_edge: std_logic;
signal Y_sig, Cb_sig, Cr_sig: std_logic_vector(7 downto 0);
signal RGB : std_logic_vector(14 downto 0);
signal b_c, g_c, x_b, x_g : std_logic_vector(9 downto 0);

signal Linecount_core_sig : std_logic_vector(9 downto 0);
```

--@@

component VGA port (

```
  reset_n : in std_logic;
  clk27   : in std_logic;
  clk50   : in std_logic;
  VGA_CLK,          -- Clock
  VGA_HS,           -- H_SYNC
  VGA_VS,           -- V_SYNC
  VGA_BLANK,        -- BLANK
  VGA_SYNC : out std_logic;  -- SYNC
  VGA_R,           -- Red[9:0]
  VGA_G,           -- Green[9:0]
  VGA_B : out std_logic_vector(9 downto 0); -- Blue[9:0]
  readin:in std_logic_vector(14 downto 0 );
  TD_HS,TD_HS_edge,
  TD_VS: in std_logic;
  vga_vsync_edge: in std_logic;
  data_valid: in std_logic;

  bluecount, greencount : in std_logic_vector(9 downto 0);
  x_blue, x_green : in std_logic_vector(9 downto 0);

  Linecount_vga : in std_logic_vector(9 downto 0);
```

```
    address : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    chipselect : IN STD_LOGIC;
    read : IN STD_LOGIC;
    write : IN STD_LOGIC;
    writedata : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    readdata : OUT STD_LOGIC_VECTOR (15 DOWNT0 0)
);

end component;

component ITU_656_Decoder port(    --    TV Decoder Input
    iTD_DATA : in std_logic_vector(7 downto 0);
    --    Position Output
    oTV_X,
    oTV_Y : out std_logic_vector(9 downto 0);
    oTV_Cont : out std_logic_vector(31 downto 0);
    --    YUV 4:2:2 Output
    oYCbCr :out std_logic_vector(23 downto 0);
    oDVAL : out std_logic;
    --    Control Signals
    iSwap_CbCr,
    iSkip,
    iRST_N,
    iCLK_27 : in std_logic    );

end component;

component DIV port(
    aclr,
    clock :in std_logic;
    denom :in std_logic_vector(3 downto 0);
    numer:in std_logic_vector(9 downto 0);
    quotient :out std_logic_vector(9 downto 0);
    remain:out std_logic_vector(3 downto 0);
    remain_0: out std_logic);
end component;

component Reset_Delay port (iCLK,iRST : in std_logic;
    oRST_0,oRST_1,oRST_2 : out std_logic);

end component;

component TD_Detect port(oTD_Stable : out std_logic;
```

```
        iTD_VS,  
        iTD_HS,  
        iRST_N : in std_logic);  
  
end component;  
  
component YUV422_to_444 port(          --      YUV 4:2:2 Input  
                                iYCbCr:in   std_logic_vector(23  
downnto 0);  
                                iFlag: in std_logic;  
                                --valid: in std_logic;  
                                --YUV  4:4:4 Output  
                                --RGB : out std_logic_vector(15  
downnto 0);  
                                oY,  
                                oCb,  
                                oCr:   out  std_logic_vector(7  
downnto 0);  
                                --Control Signals  
                                --iFlag,  
                                iCLK,  
                                iRST_N: in std_logic      );  
  
end component;  
  
component YCbCr2RGB port(   oRGB : out std_logic_vector(14 downto 0);  
                            -- oDVAL: out std_logic;  
                            x_blue,  x_green,  bluecount,  greencount  :  out  
std_logic_vector(9 downto 0);  
                            datavalid, TD_HS, TD_VS: in std_logic;  
                            Linecount : out std_logic_vector(9 downto 0);  
                            iY,iCb,iCr: in std_logic_vector(7 downto 0);  
                            iRESET,iCLK : in std_logic --, iDVAL  
                            );  
  
end component;  
  
component edge_detector port(  
                                clk:in std_logic;  
                                TD_VS: in std_logic;  
                                vga_vsync_edge: out std_logic  
);  
end component;  
  
component h_edge_detector is
```

```
port(
clk:in std_logic;
TD_HS: in std_logic;
hsync_edge: out std_logic
);
end component;

begin

TD_RESET_core <=KEY_core(0);

Linecount_core <= Linecount_core_sig;           --@@
bluecount <= b_c;
x_blue <= x_b;
--oY <= y_sig;
--oCb <= Cb_sig;
--oCr <= Cr_sig;           --Xy_GEn

v5: Reset_Delayport map(      iCLK =>clk27,
                             iRST =>TD_Stable,
                             oRST_0 =>DLY0,
                             oRST_1 =>DLY1,
                             oRST_2 =>DLY2);

v6: TD_Detect port map(oTD_Stable =>TD_Stable,
                      iTD_VS =>TD_VS_core,
                      iTD_HS =>TD_HS_core,
                      iRST_N =>KEY_core(0));

v1:   ITU_656_Decoder port map(iTD_DATA =>TD_DATA_core,
                              oTV_X =>TV_X,
                              oYCbCr =>YCbCr,
                              oDVAL=>globalvalid,
                              iSwap_CbCr =>Quotient(0),
                              iSkip=>remain_sig,
                              iRST_N =>DLY1,
                              iCLK_27=>clk27 );

v8:YUV422_to_444 port map(
                                iYCbCr=>YCbCr,
                                iFlag => globalvalid,
```

```
oY => y_sig,  
oCr => Cr_sig,  
oCb => Cb_sig,  
--      Control Signals  
iCLK=>clk27,  
iRST_N=> KEY_core(0) );
```

```
v10: YCbCr2RGB port map (      oRGB => RGB,  
                                iY => y_sig,  
                                iCb =>Cb_sig,  
                                iCr =>Cr_sig,  
                                iRESET => '0',  
                                TD_HS => TD_HS_core,  
                                TD_VS => TD_VS_core,  
                                Linecount => Linecount_core_sig,  
                                datavalid => globalvalid,  
                                bluecount => b_c,  
                                greencount => g_c,  
                                x_blue => x_b,  
                                x_green => x_g,  
                                iCLK => clk27);
```

```
v3: DIV port map(      aclr => not (DLY0),  
                                clock =>clk27,  
                                denom =>"1001",  
                                numer =>TV_X,  
                                quotient =>Quotient,  
                                remain =>Remain,  
                                remain_0 => remain_sig);
```

```
V4: VGA port map (  
  reset_n => '1',  
  clk27 =>clk27,  
    clk50 => CLK50,  
  VGA_CLK => VGA_CLK_core,  
  VGA_HS => VGA_HS_core,  
  VGA_VS => VGA_VS_core,  
  VGA_BLANK => VGA_BLANK_core,  
  VGA_SYNC => VGA_SYNC_core,  
  VGA_R => VGA_R_core,  
  VGA_G => VGA_G_core,  
  VGA_B => VGA_B_core,
```

```
    TD_HS => TD_HS_core,
    TD_VS => not(TD_VS_core),
    vga_vsync_edge => vga_vsync_edge,
    readin => RGB,
    data_valid => globalvalid,
    TD_HS_edge => TD_HS_edge,
    bluecount => b_c,
    greencount => g_c,
    x_blue => x_b,
    x_green => x_g,

    Linecount_vga => Linecount_core_sig,

    address => address,
    chipselect => chipselect,
    read => read,
    write => write,
    writedata => writedata,
    readdata => readdata
);

V9: edge_detector port map(
    clk => clk27,
    TD_VS => TD_VS_core,
    vga_vsync_edge => vga_vsync_edge
);

v12:h_edge_detector port map(
    clk => clk27,
    TD_HS => TD_HS_core,
    hsync_edge => TD_HS_edge
);

end datapath;
```

```
--
-- Modified by : Anusha Dacheppally
--             Devesh Dedhia
--             Raghu Binnamangalam
--             Roopa Karkarlapudi
```



```
module Reset_Delay(iCLK,iRST,oRST_0,oRST_1,oRST_2);
input      iCLK;
input      iRST;
output reg oRST_0;
output reg oRST_1;
output reg oRST_2;

reg  [21:0] Cont;

always@(posedge iCLK or negedge iRST)
begin
    if(!iRST)
    begin
        Cont  <=  0;
        oRST_0 <=  0;
        oRST_1 <=  0;
        oRST_2 <=  0;
    end
    else
    begin

        if(Cont!=22'h228F5B) //228F5B.9F4CC1086
        Cont  <=  Cont+1;
        if(Cont>=22'h1147AD) //1147AD.8A87A8312
        oRST_0 <=  1;
        if

        (Cont>=22'h19EB84) //19EB84.94EA349CC
        oRST_1 <=  1;
        if(Cont>=22'h228F5B) //228F5B.9F4CC1086
        oRST_2 <=  1;

        //      if(Cont!=22'h3FFFFFF)
        //      Cont  <=  Cont+1;
        //      if(Cont>=22'h1FFFFFF)
        //      oRST_0 <=  1;
        //      if(Cont>=22'h2FFFFFF)
        //      oRST_1 <=  1;
        //      if(Cont>=22'h3FFFFFF)
        //      oRST_2 <=  1;

    end
end
```

```
endmodule
```

```
--  
-- Modified by : Anusha Dacheppally  
--           Devesh Dedhia  
--           Raghu Binnamangalam  
--           Roopa Karkarlapudi  
  
module TD_Detect(    oTD_Stable,  
                   iTD_VS,  
                   iTD_HS,  
                   iRST_N );  
  
input      iTD_VS;  
input      iTD_HS;  
input      iRST_N;  
output     oTD_Stable;  
reg        TD_Stable;  
reg        Pre_VS;  
reg [7:0]  Stable_Cont;  
  
assign oTD_Stable = TD_Stable;  
  
always@(posedge iTD_HS or negedge iRST_N)  
begin  
    if(!iRST_N)  
    begin  
        TD_Stable    <=    1'b0;  
        Stable_Cont  <=    4'h0;  
        Pre_VS       <=    1'b0;  
    end  
    else  
    begin  
        Pre_VS <= iTD_VS;  
        if(!iTd_VS)  
        Stable_Cont <= Stable_Cont+1'b1;  
        else  
        Stable_Cont <= 0;  
  
        if({Pre_VS,iTd_VS}==2'b01)  
        begin
```

```
        if(Stable_Cont==9)
            TD_Stable    <=    1'b1;
        else
            TD_Stable    <=    1'b0;
        end
    end
end
end

endmodule
```

```
--
-- Modified by : Anusha Dacheppally
--             Devesh Dedhia
--             Raghu Binnamangalam
--             Roopa Karkarlapudi

module ITU_656_Decoder(    //    TV Decoder Input
                        iTD_DATA,
                        //    Position Output
                        oTV_X,
                        oTV_Y,
                        oTV_Cont,
                        //    YUV 4:2:2 Output
                        oYCbCr,
                        oDVAL,
                        //    Control Signals
                        iSwap_CbCr,
                        iSkip,
                        iRST_N,
                        iCLK_27);

input  [7:0]  iTD_DATA;
input          iSwap_CbCr;
input          iSkip;
input          iRST_N;
input          iCLK_27;
output [23:0]  oYCbCr;
output [9:0]   oTV_X;
output [9:0]   oTV_Y;
output [31:0]  oTV_Cont;
output          oDVAL;
```

```
// For detection
reg      [23:0] Window;          // Sliding window register
reg      [17:0] Cont;           // Counter
reg      Active_Video;
reg      Start;
reg      Data_Valid;
reg      Pre_Field;
reg      Field;
wire     SAV;
reg      FVAL;
reg      [9:0] TV_Y;
reg      [31:0] Data_Cont;

// For ITU-R 656 to ITU-R 601
reg      [7:0] Cb;
reg      [7:0] Cr;
reg      [23:0] YCbCr;

assign oTV_X = Cont>>1;
assign oTV_Y = TV_Y;
assign oYCbCr = YCbCr;
assign oDVAL = Data_Valid;
assign SAV = (Window==24'hFF0000)&(iTD_DATA[4]==1'b0);
assign oTV_Cont= Data_Cont;

always@(posedge iCLK_27 or negedge iRST_N)
begin
    if(!iRST_N)
    begin
        // Register initial
        Active_Video<= 1'b0;
        Start <= 1'b0;
        Data_Valid <= 1'b0;
        Pre_Field <= 1'b0;
        Field <= 1'b0;
        Window <= 24'h0;
        Cont <= 18'h0;
        Cb <= 8'h0;
        Cr <= 8'h0;
        YCbCr <= 23'h0;
        FVAL <= 1'b0;
    end
end
```

```
TV_Y      <= 10'h0;
Data_Cont <= 32'h0;

end
else
begin
    // Sliding window
    Window <= {Window[15:0],iTD_DATA};
    // Active data counter
    if(SAV)
        Cont <= 18'h0;
    else if(Cont<1440)
        Cont <= Cont+1'b1;
    // Check the video data is active?
    if(SAV)
        Active_Video<= 1'b1;
    else if(Cont==1440)
        Active_Video<= 1'b0;
    // Is frame start?
    Pre_Field <= Field;
    if({Pre_Field,Field}==2'b10)
        Start <= 1'b1;
    // Field and frame valid check
    if(Window==24'hFF0000)
        begin
            FVAL <= !iTD_DATA[5];
            Field <= iTD_DATA[6];
        end
    // ITU-R 656 to ITU-R 601
    if(iSwap_CbCr)
        begin
            case(Cont[1:0]) // Swap
            0: Cb <= iTD_DATA;
            1: YCbCr <= {iTD_DATA,Cb,Cr};
            2: Cr <= iTD_DATA;
            3: YCbCr <= {iTD_DATA,Cb,Cr};
            endcase
        end
    else
        begin
            case(Cont[1:0]) // Normal
            0: Cb <= iTD_DATA;
            1: YCbCr <= {iTD_DATA,Cb,Cr};
```

```
                2:    Cr          <=    iTD_DATA;
                3:    YCbCr <=    {iTD_DATA,Cb,Cr};
                endcase
            end
            //    Check data valid
            if(        Start                //    Frame Start?
                &&    FVAL                //    Frame valid?
                &&    Active_Video        //    Active video?
                &&    Cont[0]                //    Complete ITU-R 601?
                &&    !iSkip )            //    Is non-skip pixel?
            Data_Valid <=    1'b1;

            else
            Data_Valid <=    1'b0;

            //    TV decoder line counter for one field
            //if(FVAL && SAV)
            //TV_Y<=    TV_Y+1;
            //if(!FVAL)
            //TV_Y<=    0;
            //    Data counter for one field
            //if(!FVAL)
            //Data_Cont <=    0;
            //if(Data_Valid)
            //Data_Cont <=    Data_Cont+1'b1;
        end
    end

endmodule

-----
--
-- Modified by : Anusha Dacheppally
--             Devesh Dedhia
--             Raghu Binnamangalam
--             Roopa Karkarlapudi

module YUV422_to_444    (    //    YUV 4:2:2 Input
                        iYCbCr,

                        iFlag,
```

```

//valid,
//    YUV    4:4:4 Output
oY,
oCb,
oCr,
//    Control Signals
iX,
iCLK,
iRST_N );

//    YUV 4:2:2 Input
input  [23:0]  iYCbCr;
//    YUV    4:4:4 Output
output [7:0]   oY;
output [7:0]   oCb;
output [7:0]   oCr;
//    Control Signals
input  [9:0]   iX;
input                iCLK;
input                iFlag;
//input            valid;
input                iRST_N;
//    Internal Registers
reg      [7:0]  mY;
reg      [7:0]  mCb;
reg      [7:0]  mCr;
reg flag;

assign oY    =    mY;
assign oCb   =    mCb;
assign oCr   =    mCr;
initial begin
flag=0;
end

always@(posedge iCLK or negedge iRST_N)
begin
    if(!iRST_N)
    begin
        mY    <=    0;
        mCb   <=    0;
        mCr   <=    0;
    end
end
```

```
        else
        begin
//          if(iFlag)
//            flag <=~flag;
//          if(flag)
//            {mY,mCr}    <=    iYCbCr;
//          else
//            {mY,mCb}    <=    iYCbCr;
//            {mY,mCb,mCr} <= iYCbCr;
        end
    end

//always@(valid)
//begin
//    flag <=~flag;
//end
Endmodule
```

```
--
-- Modified by : Anusha Dacheppally
--             Devesh Dedhia
--             Raghu Binnamangalam
--             Roopa Karkarlapudi
```

```
module YCbCr2RGB ( //Red,Green,Blue,oDVAL,
                  oRGB, x_blue, x_green, bluecount, greencount,
                  datavalid, TD_HS,
                  Linecount, TD_VS,
                  iY,iCb,iCr,//iDVAL,
                  iRESET,iCLK);

//    Input
input [7:0] iY,iCb,iCr;
//input //iDVAL,
input iRESET,iCLK;
input datavalid, TD_HS, TD_VS;
//    Output
output [14:0]oRGB;
output [9:0] Linecount;
output [9:0] x_blue, x_green, bluecount, greencount;
//output [9:0] Red,Green,Blue;
```



```
//output reg  oDVAL;
//      Internal Registers/Wires
reg [14:0] oRGB_sig;
reg [9:0] Linecount;
reg [9:0] oRed,oGreen,oBlue;
reg [9:0] x_blue, x_green,Temp_x_blue,Temp_x_green;
reg [10:0] Temp_bluecount,Temp_greencount,bluecount,greencount;
//reg  [3:0] oDVAL_d;
reg [19:0] X_OUT,Y_OUT,Z_OUT;
wire [26:0] X,Y,Z;
reg [12:0] Xcount,Reset_count;
assign oRGB = oRGB_sig;
//assignRed = oRed;
//assignGreen= oGreen;
//assignBlue = oBlue;

// manages the count values
always@(posedge iCLK)
begin
    if(iRESET || TD_HS)
        begin
            Xcount<= 0;
        end
    else if (datavalid)
        Xcount <= Xcount + 1;
end

//line count
always@(posedge iCLK)
begin
    if(datavalid)
    begin
        if(Xcount == 638)
            Linecount<= Linecount + 1;
        else
            Linecount<= Linecount + 0;
    end
    if(!TD_VS)
        Linecount <= 0;
end
```

```
//temp values assigned at TD_HS
always@(posedge iCLK)
begin
    if( TD_HS)
    begin
        bluecount<=Temp_bluecount;
        greencount<=Temp_greencount;
        x_blue<= Temp_x_blue;
        x_green<=Temp_x_green;
    end
end

// main part
always@(posedge iCLK)
begin
    Reset_count<=Reset_count+1;

    if(iRESET || TD_HS)
    begin
        oRed<=0;
        oGreen<=0;
        oBlue<=0;
        Reset_count<=0;
    end

    //The Temp values reset 10 clockcycles after TD_HS
    else if(Reset_count==5)
    begin
        Temp_x_blue<= 0; Temp_x_green <= 0;
        Temp_bluecount<=0; Temp_greencount<= 0;
    end

    else if (datavalid)
    begin
        if(iY > 85 && iCb>140 && iCr<120)           //iY>130           // blue detection
        //if (1)
        begin
            Temp_bluecount<= Temp_bluecount + 1;
            oRed<=1023;
```

```
oGreen<=1023;
oBlue<=1023;
Temp_x_blue <= Xcount;
end

else if(iY>100 && iCb<120 && iCr<110)           // green detection
begin
Temp_greencount <= Temp_greencount + 1;
oRed<=1023;
oGreen<=1023;
oBlue<=1023;
Temp_x_green <= Xcount;
end

else //if(iDVAL)
begin
    // Red
    if(X_OUT[19])
        oRed<=0;
    else if(X_OUT[18:0]>1023)
        oRed<=1023;
    else
        oRed<=X_OUT[9:0];

    // Green
    if(Y_OUT[19])
        oGreen<=0;
    else if(Y_OUT[18:0]>1023)
        oGreen<=1023;
    else
        oGreen<=Y_OUT[9:0];

    // Blue
    if(Z_OUT[19])
        oBlue<=0;
    else if(Z_OUT[18:0]>1023)
        oBlue<=1023;
    else
        oBlue<=Z_OUT[9:0];
    // Control
    //{oDVAL,oDVAL_d}<={oDVAL_d,iDVAL};
    //oDVAL <= 1;
```

```
        end
        end
        oRGB_sig <= {oRed[9:5], oGreen[9:5], oBlue[9:5]};
end

always@(posedge iCLK)
begin
    if(iRESET)
        begin
            X_OUT<=0;
            Y_OUT<=0;
            Z_OUT<=0;
        end
    else //if(iDVAL)
        begin
            X_OUT<=( X - 114131 ) >>7;
            Y_OUT<=( Y + 69370 ) >>7;
            Z_OUT<=( Z - 141787 ) >>7;
        end
    end
end

//      Y          596,          0,          817
MAC_3 u0(  iY,          iCb,          iCr,
          17'h00254,  17'h00000,  17'h00331,
          X,          iRESET,        iCLK);

//      Cb          596,          -200,         -416
MAC_3 u1(  iY,          iCb,          iCr,
          17'h00254,  17'h3FF38,  17'h3FE60,
          Y,          iRESET,        iCLK);

//      Cr          596,          1033,          0
MAC_3 u2(  iY,          iCb,          iCr,
          17'h00254,  17'h00409,  17'h00000,
          Z,          iRESET,        iCLK);

Endmodule

-----
-
-- Authors: Anusha Dacheppally
```

```
--          Devesh Dedhia
--          Raghu Binnamangalam
--          Roopa Karkarlapudi--
-----
-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity VGA is

generic
    (
        DATA_WIDTH : natural := 64;
        ADDR_WIDTH  : natural := 10
    );

port (
    reset_n : in std_logic;
    clk27   : in std_logic;
    clk50   : in std_logic;
    VGA_CLK,                -- Clock
    VGA_HS,                 -- H_SYNC
    VGA_VS,                 -- V_SYNC
    VGA_BLANK,              -- BLANK
    VGA_SYNC : out std_logic; -- SYNC
    VGA_R,                 -- Red[9:0]
    VGA_G,                 -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0); -- Blue[9:0]
    readin:in std_logic_vector(14 downto 0 );
    --iG,iR,iB: in std_logic_vector(7 downto 0);
    TD_HS,TD_HS_edge,
    TD_VS: in std_logic;
    vga_vsync_edge: in std_logic;
    data_valid: in std_logic;

    bluecount, greencount : in std_logic_vector(9 downto 0);
    x_blue, x_green : in std_logic_vector(9 downto 0);
    Linecount_vga : in std_logic_vector(9 downto 0);    --@@
                -- Avalon_signals

    signal address : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    signal chipselect : IN STD_LOGIC;
    signal read : IN STD_LOGIC;
    signal write : IN STD_LOGIC;
    signal writedata : IN STD_LOGIC_VECTOR (15 DOWNT0 0);
    signal readdata : OUT STD_LOGIC_VECTOR (15 DOWNT0 0)
);

end VGA;

architecture rtl of VGA is

component ram_dual
```

```
port (  
    rclk : in std_logic;  
    wclk : in std_logic;  
    raddr : in natural range 0 to 2**ADDR_WIDTH - 1;  
    waddr : in natural range 0 to 2**ADDR_WIDTH - 1;  
    data : in std_logic_vector((DATA_WIDTH-1) downto 0);  
    we    : in std_logic := '1';  
    q     : out std_logic_vector((DATA_WIDTH -1) downto 0)  
);  
  
end component;
```

```
constant HTOTAL      : integer := 858;  
constant HSYNC       : integer := 103;  
constant HBACK_PORCH : integer := 76;  
constant HACTIVE     : integer := 640;  
constant HFRONT_PORCH : integer := 39;  
  
constant VTOTAL      : integer := 524;  
constant VSYNC       : integer := 2;  
constant VBACK_PORCH : integer := 17;           --34;  
constant VACTIVE     : integer := 500;--478;  
constant VFRONT_PORCH : integer := 5;--10;  
  
-- Signals for the video controller  
signal Hcount : unsigned(10 downto 0); -- Horizontal position (0-800)  
signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)  
signal EndOfLine, EndOfField : std_logic;  
--signal vga_x : std_logic := '0';  
  
signal vga_hblank, vga_hsync,  
       vga_vblank, vga_vsync,  
flag_TD_HS_edge, flag_TD_VS_edge, flag_TD_HS_edge_reset : std_logic; --  
Sync. signals  
  
type ROMType is array(0 to 640) of std_logic_vector(14 downto 0);  
signal ROM1, ROM2:ROMType := (others => "0000000000000000");  
signal Read_HS_flag : std_logic := '0';  
signal a,b,c: std_logic;  
signal INDEX : integer range 0 to 800;  
signal regx1, regx2, inner_regx1, inner_regx2 : integer := 0;  
    --range 0 to 640;  
signal x_info_from_nios, x_info_to_vga : std_logic_vector(63 downto 0);  
signal ram_index : integer range 0 to 500:=0;  
  
begin  
  
VGA_RAM : ram_dual port map(  
    rclk => clk27,  
    wclk => clk50,  
    raddr => (to_integer(Vcount)-VSYNC - VBACK_PORCH),
```

```
        waddr => ram_index,
        data  => x_info_from_nios,
        q    => x_info_to_vga
    );

ReadfromRAM : process (clk27)
begin
if rising_edge (clk27) then
regx1 <= to_integer(unsigned(x_info_to_vga(15 downto 0)));
regx2 <= to_integer(unsigned(x_info_to_vga(31 downto 16)));
inner_regx1<=to_integer(unsigned(x_info_to_vga(47 downto 32)));
inner_regx2<=to_integer(unsigned(x_info_to_vga(63 downto 48)));
end if;
end process ReadfromRAM;

HCounter : process (clk27)
begin
if rising_edge (clk27) then
if reset_n = '0' then-- or
Hcount <= (others => '0');
elsif TD_HS='1'then
Hcount <=(others =>'0');
else
Hcount <= Hcount+1;
end if;
end if;
end process HCounter;

VCounter: process (clk27)
begin
if rising_edge (clk27) then
if reset_n = '0'then
Vcount <= (others => '0');
elsif vga_vsync_edge='1' then
Vcount <= (others => '0');
elsif TD_HS='1' then
Vcount<=Vcount+1;
end if;
end if;
end process VCounter;

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

HSyncGen : process (clk27)
begin
if rising_edge (clk27) then
if reset_n = '0' then
vga_hsync <= '1';
elsif TD_HS='1' then
vga_hsync <= '1';
elsif Hcount=Htotal-1 then
vga_hsync <= '1';
elsif Hcount = HSYNC - 1 or Hcount=HTOTAL+HSYNC-1 then
vga_hsync <= '0';
```

```
        end if;
    end if;
end process HSyncGen;

HBlankGen : process (clk27)
begin
    if rising_edge(clk27) then
        if reset_n = '0' then
            vga_hblank <= '1';
        elsif Hcount = HSYNC + HBACK_PORCH-1 or Hcount = HTOTAL+HSYNC +
HBACK_PORCH-1 then
            vga_hblank <= '0';
        elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE-1 or
            Hcount = HTOTAL + HSYNC + HBACK_PORCH + HACTIVE-1 then
            vga_hblank <= '1';
        end if;
    end if;
end process HBlankGen;

VSyncGen : process (clk27)
begin
    if rising_edge(clk27) then
        if reset_n = '0' then
            vga_vsync <= '1';
        elsif vga_vsync_edge='1' then
            vga_vsync<='1';
        elsif Vcount = VSYNC - 1 then
            vga_vsync <= '0';
        end if;
    end if;
end process VSyncGen;

VBlankGen : process (clk27)
begin
    if rising_edge(clk27) then
        if reset_n = '0' then
            vga_vblank <= '1';
        if Vcount = VSYNC+VBACK_PORCH - 1 then
            vga_vblank <= '0';
        elsif Vcount =VSYNC+VBACK_PORCH + VACTIVE - 1 then
            vga_vblank <= '1';
        end if;
    end if;
end process VBlankGen;

Read_HS: process(TD_HS)
begin
    if TD_HS'event and TD_HS = '1' then
        Read_HS_flag <= not Read_HS_flag;
    end if;
end process;
```



```
end process Read_HS;

fill_buffer: process(clk27,data_valid)
variable i,j : integer range 0 to 720 := 0;
begin
    if rising_edge(clk27) then
        if Read_HS_flag = '1' and data_valid='1' then
            ROM1 ( i ) <= readin ;
            i := i+ 1;
            if i = 639 then
                i := 0;
            elsif TD_HS='1' then
                i:=0;
            end if;
        elsif Read_HS_flag='0' and data_valid='1' then
            ROM2 ( j ) <= readin ;
            j := j+1;
            if j = 639 then
                j := 0;
            elsif TD_HS='1' then
                j:=0;
            end if;
        end if;
    end if;
end process fill_buffer;

-----
-----
-----NIOS Communication-----
-----
--bus_comm: process (clk50)
-- begin
--     if rising_edge(clk50) then
--         if (chipselect = '1')then
--             if address(4 downto 0) = "00000" then
--                 if write = '1' then
--                     regx1 <= to_integer(unsigned(writedata));--x1
--                 end if;
--             elsif address(4 downto 0) = "00010" then
--                 if write = '1' then
--                     regx2 <= to_integer(unsigned(writedata));--x2
--                 end if;
--             end if;
--         end if;
--     end if;
-- end process bus_comm;

-----
bus_comm: process (clk50)
begin
    if rising_edge(clk50) then
        if (chipselect = '1')then
            if address(4 downto 0) = "00000" then
```

```

        if write = '1' then
            x_info_from_nios(15 downto 0) <= writedata;--x1
        end if;
        elsif address(4 downto 0) = "00010" then
            if write = '1' then
                x_info_from_nios(31 downto 16) <= writedata;--x2
            end if;
            elsif address(4 downto 0) = "11110" then --- using
address 30 inner x1
            if write = '1' then
                x_info_from_nios(47 downto 32) <= writedata;
            end if;
            elsif address(4 downto 0) = "10010" then --- using
address 18 inner x2
            if write = '1' then
                x_info_from_nios(63 downto 48) <= writedata;
            end if;

            elsif address(4 downto 0) = "11000" then -- unused
address 24
            if write = '1' then
                ram_index <= to_integer(unsigned(writedata));
            end if;

            end if;
    end if;
end process bus_comm;

```

```

-----
-----
Read_process: process(clk50)
begin

```

```

    if rising_edge(clk50) then
        if vga_vsync_edge = '1' then
            flag_TD_VS_edge <= '1';
        end if;
        if TD_HS='1' then
            flag_TD_HS_edge<='1';
        end if;

```

```

        if (chipselct = '1')then

```

```

-----VSFlagwrite(ADDR 20)-----
-----

```

```

            if address(4 downto 0) = "10100" then
                if write = '1' then
                    flag_TD_VS_edge <= '0';
                end if;
            end if;

```

```

-----HS FlagWrite(ADDR 30)-----
-----

```

```

--            if address(4 downto 0) = "11110" then

```

```

--          if write = '1' then
--              flag_TD_HS_edge <= '0';
--              end if;
--          end if;

-----VGA_HS (ADDR 4)-----
-----
          if address(4 downto 0) = "00100" then
          if read = '1' then
              if vga_hsync='1' then
                  readdata<="00000000000000001";
-- We send VGA_HS in form of 1 or 0
                  else
                      readdata<=(others=>'0');
                  end if;
              end if;
          end if;
          end if;

-----ROW_NO (ADDR 6)-----
-----
          if address(4 downto 0) = "00110" then
          if read = '1' then
              if vga_hsync='0' then
                  if (to_integer(Vcount)-VSYNC -
VBACK_PORCH)>0 and (to_integer(Vcount)-VSYNC - VBACK_PORCH) < 478 then
-- line number is sent to NIOS
                      readdata <=
std_logic_vector(to_unsigned((to_integer(Vcount)-VSYNC - VBACK_PORCH),
16));
--
                  else
                      readdata<=(others=>'0');
                  end if;
              end if;
          end if;
          end if;
          end if;

-----Blue count(ADDR 8)-----
-----
          if address(4 downto 0) = "01000" then
          if read = '1' then
              --if TD_HS = '0' then
              readdata <= ("000000" & bluecount);
              end if;
              end if;
              --end if;

-----Xblue(ADDR 10)-----
-----
          if address(4 downto 0) = "01010" then
          if read = '1' then
              -- if TD_HS = '0' then
              readdata <= ("000000" & x_blue);
-- software must find x_blue - bluecount

```

```

        end if;
        end if;
        --end if;
-----Green count(ADDR 12)-----
-----
        if address(4 downto 0) = "01100" then
            if read = '1' then
                --if TD_HS = '0' then
                    readdata <= ("000000" & greencount);
                end if;
            end if;
-----Xgreen(ADDR 14)-----
-----
        if address(4 downto 0) = "01110" then
            if read = '1' then
                readdata <= ("000000" & x_green);
            -- software must find x_green - greencount
            end if;
        end if;

-----VGA_VS(ADDR 16)-----
-----
        if address(4 downto 0) = "10000" then
            if read='1' then
                if vga_vsync='1' then
                    readdata<="000000000000000001";
            -- We send VGA_VS in form of 1 or 0
                else
                    readdata<=(others=>'0');
                end if;
            end if;
        end if;
-----TD_HS(ADDR 18)-----
-----
        -- if address(4 downto 0) ="10010" then
        -- if read='1' then
        -- if TD_HS ='1'then
        -- readdata<="000000000000000001";           -- We send
        TD_HS in form of 1 or 0
        -- else
        -- readdata<=(others=>'0');
        -- end if;
        -- end if;
        -- end if;

-----TD_HS_edge(ADDR 22)-----
-----
        -- if address(4 downto 0) ="10110" then
        -- if read='1' then
        -- if flag_TD_HS_edge = '1' then
        -- readdata<= "000000000000000001";           -- We send
        TD_HS in form of 1 or 0

```

```

--          else
--          readdata<=(others=>'0');
--          end if;
--          end if;
--          end if;

-----Linecount_vga (ADDR 26)-----
-----
        if address(4 downto 0) ="11010" then
            if read='1' then
                --if TD_HS = '1' then
                readdata<= "000000" & Linecount_vga;          --
send line count
                end if;
                end if;

-----VS_edge (ADDR 28)-----
-----
        if address(4 downto 0) ="11100" then
            if read='1' then
                if flag_TD_VS_edge = '1' then
                    readdata<= "00000000000000001";          -- We send
TD_VS_edge
                else
                    readdata<=(others=>'0');
                    end if;
                    end if;
                    end if;

                end if;
            end if;
        end process Read_process;

VideoOut: process (clk27, reset_n)
begin
if rising_edge(clk27) then
if (Hcount<HTOTAL) then
INDEX<=to_integer(HCOUNT-HSYNC-HBACK_PORCH);          --
else
INDEX<=to_integer(HCOUNT-HSYNC-HBACK_PORCH-HTOTAL);          --
end if;

        if reset_n = '0' then
            VGA_R <= "0000000000";
            VGA_G <= "0000000000";
            VGA_B <= "0000000000";
        else
            if vga_hblank = '0' and vga_vblank ='0' then
                if Read_HS_flag = '1' then
                    if INDEX > regx1 and INDEX < regx2 then
                        if ( INDEX > inner_regx1 and INDEX <
inner_regx2) then
                            VGA_R <= "1111111111";
                            VGA_G <= "1111111111";          -- making a plain
green light saber

```

```

                                VGA_B <= "1111111111";
                                else
                                VGA_R <= ROM2(INDEX)(14 downto 10) & "11111";
                                VGA_G <= ROM2(INDEX)(9) & "1111111111";  --
transparent light saber
                                VGA_B <= ROM2(INDEX)(4 downto 0)& "11111";
                                end if;
                                else
                                VGA_R <= ROM2(INDEX)(14 downto 10) & "00000";
                                VGA_G <= ROM2(INDEX)(9 downto 5)& "00000";
                                VGA_B <= ROM2(INDEX)(4 downto 0)& "00000";
                                end if;
                                elsif Read_HS_flag = '0' then
                                if INDEX > regx1 and INDEX < regx2 then
                                if( INDEX > inner_regx1 and INDEX <
inner_regx2) then
                                VGA_R <= "1111111111";
                                VGA_G <= "1111111111";  -- making a plain
green light saber
                                VGA_B <= "1111111111";
                                else
                                VGA_R <= ROM1(INDEX)(14 downto 10) & "11111";
                                VGA_G <= ROM1(INDEX)(9) & "1111111111";  --
transparent light saber
                                VGA_B <= ROM1(INDEX)(4 downto 0)& "11111";
                                end if;
                                else
                                VGA_R <= ROM1(INDEX)(14 downto 10) & "00000";
                                VGA_G <= ROM1(INDEX)(9 downto 5)& "00000";
                                VGA_B <= ROM1(INDEX)(4 downto 0)& "00000";
                                end if;
                                else
                                VGA_R <= "0000000000";
                                VGA_G <= "0000000000";
                                VGA_B <= "0000000000";
                                end if;
                                end if;
                                end if;
                                end if;
                                end process VideoOut;

VGA_CLK <= clk27;
VGA_HS <= not vga_hsync;
VGA_VS <= not vga_vsync;
VGA_SYNC <= '0';
VGA_BLANK <= not (vga_hsync or vga_vsync);

end rtl;
```

```
--
-- Authors   : Anusha Dachepally
--            : Devesh Dedhia
--            : Raghu Binnamangalam
```

```
--          Roopa Karkarlapudi
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity edge_detector is
  port(
    clk:in std_logic;
    TD_VS: in std_logic;
    vga_vsync_edge: out std_logic
  );
end edge_detector;

architecture rtl of edge_detector is
  signal q: std_logic;
begin

  process(clk)
  begin
    if rising_edge(clk) then
      q<=TD_VS;
    end if;
  end process;

  vga_vsync_edge<=q and not (TD_VS);
end rtl;
```

```
--
-- Authors   : Anusha Dacheppally
--            Devesh Dedhia
--            Raghu Binnamangalam
--            Roopa Karkarlapudi
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity h_edge_detector is
  port(
    clk:in std_logic;
    TD_HS: in std_logic;
    hsync_edge: out std_logic
  );
end h_edge_detector;

architecture rtl of h_edge_detector is
  signal q: std_logic;
begin
  process(clk)
  begin
    if rising_edge(clk) then
      q<=TD_HS;
    end if;
  end process;
```

```
hsync_edge<=q and not (TD_HS);  
end rtl;
```

SOFTWARE CODE

```
--  
-- Authors : Anusha Dacheppally  
--           Devesh Dedhia  
--           Raghu Binnamangalam  
--           Roopa Karkarlapudi  
  
#include <io.h>  
#include <system.h>  
#include <unistd.h>  
#include <stdio.h>  
#include "light_saber.h"  
  
#define WriteLightSaber(base, address, data) IOWR_16DIRECT(base, address*2, data)  
#define ReadLightSaber(base,address) IORD_16DIRECT(base, address*2)  
#define ReadXY(base,address) IORD_32DIRECT(base, address*4)  
  
struct color{  
    int x1;  
    int x2;  
    int count;  
    int first;  
    long average;  
    int last;  
    int final_x;  
    int final_y;  
    int mycount_x;  
    int mean;  
};  
  
int main(){  
    int mycount_blue=0,mycount_green=0;
```



```
int i,X_Blue,X_green;
int LineCount;
struct color b1;
struct color g1;
int prevlc=0,TD_VS_edge;
int VGA_HS;
struct lookuptable table[506];
LineCount= ReadLightSaber(CORE_NIOS_BASE,26);
int b=0;
int a=0;
while(1){
    prevlc=LineCount;
    LineCount= ReadLightSaber(CORE_NIOS_BASE,26);

//-----X-Y calculation-----

    if(LineCount!=prevlc){
        //blue calc
        int blue_data = ReadXY(BLUE_INST_BASE,LineCount);
        int temp_bluedata = blue_data;
        b1.count = 0x0FFFF & temp_bluedata;
        X_Blue =temp_bluedata >> 16;
        b1.x1=(X_Blue+1)-b1.count;
        if(b1.x1==1)
            b1.x1=0;
        b1.x2=X_Blue;

        //green calc
        g1.count=ReadLightSaber(CORE_NIOS_BASE,12);
        X_green=ReadLightSaber(CORE_NIOS_BASE,14);
        g1.x1=(X_green+1)-g1.count;
        if(g1.x1==1){
            g1.x1=0;
        }
        g1.x2=X_green;

        if (b1.count> 0) {
            if(b1.first ==0){
                b1.first=LineCount;
            }
            b1.last=LineCount;
        }
    }
}
```

```
        mycount_blue ++;
        b1.mean=(b1.x1+b1.x2)>>1;
        b1.average=(b1.average+b1.mean);
    }

    if (g1.count> 0) {
        if(g1.first ==0){
            g1.first=LineCount;
        }
        g1.last=LineCount;
        mycount_green ++;
        g1.mean=(g1.x1+g1.x2)>>1;
        g1.average=(g1.average+g1.mean);
    }
}

//-----LightSaber Calculation-----//
    TD_VS_edge=ReadLightSaber(CORE_NIOS_BASE,28);

    if(TD_VS_edge==1){

        WriteLightSaber(CORE_NIOS_BASE,20,0); //rest the TD_VS_flag
        b1.final_y=(b1.last+b1.first);
        b1.final_y = b1.final_y >> 1;
        g1.final_y=(g1.last+g1.first)>>1;

        if(mycount_blue>0)
            b1.final_x=b1.average/mycount_blue;
        else
            b1.final_x=0;

        if(mycount_green>0)
            g1.final_x=g1.average/mycount_green;
        else
            g1.final_x=0;

        g1.average=0;
        b1.average=0;
        mycount_green = 0;
        mycount_blue=0;
    }
}
```

```
b1.first=0;
b1.last=0;
g1.first=0;
g1.last=0;
g1.count = 0;
b1.count = 0;
b=0;

for(i=0;i<506;i++){
    table[i].x1=0;
    table[i].x2=0;
    table[i].in_x1=0;
    table[i].in_x2=0;
}

if((b1.final_x>0 && (b1.final_y<<1)>0) && (g1.final_x>0 && (g1.final_y<<1)>0))
    DrawLightSaber(b1.final_x,(b1.final_y<<1),g1.final_x,(g1.final_y<<1),table);

for (i = 0; i<506; i++)
{
    WriteLightSaber(CORE_NIOS_BASE, 0,table[i].in_x1);
    WriteLightSaber(CORE_NIOS_BASE, 2,table[i].in_x2);
    WriteLightSaber(CORE_NIOS_BASE, 30,table[i].x1);
    WriteLightSaber(CORE_NIOS_BASE, 18,table[i].x2);
    WriteLightSaber(CORE_NIOS_BASE, 24, i);

}

}

}

return 0;

}
```

```
--  
-- Modified by : Anusha Dacheppally  
--           Devesh Dedhia  
--           Raghu Binnamangalam  
--           Roopa Karkarlapudi  
  
#define SHIFT 7  
#include <stdio.h>  
#include <io.h>  
#include <system.h>  
#include <unistd.h>  
#include "light_saber.h"  
#define SWAP(x0,y0) {(x0 ^= y0); (y0 ^= x0); (x0 ^= y0);}  
  
int sqrt(int num);  
void line1(int x0, int y0, int x1, int y1,struct lookuptable *table);  
  
//-----Draw Light Saber-----//  
  
void DrawLightSaber(int x0,int y0,int x1,int y1,struct lookuptable* table)  
{  
    //struct lookuptable table[480];  
  
    int dx = x0 - x1;  
    int dy = y0 - y1;  
    int hyp = sqrt((x0 - x1) * (x0 - x1) + (y0 - y1) * (y0 - y1));  
    int costheta = dy << SHIFT / hyp;  
    int sintheta = dx << SHIFT / hyp;  
  
    int deltay = (swordwidth * sintheta) >> SHIFT;  
    int deltax = (swordwidth * costheta) >> SHIFT;  
  
    line1(x0 + deltax + 1, y0 - deltay - 1, x1 + deltax +1, y1 - deltay - 1 ,table);  
    line1(x0 - deltax - 1, y0 + deltay + 1, x1 - deltax - 1, y1 + deltay + 1,table);  
    line1(x0 - deltax - 1, y0 + deltay + 1 , x0 + deltax + 1, y0 - deltay - 1 ,table);  
    line1(x1 + deltax + 1, y1 - deltay - 1 , x1 - deltax - 1, y1 + deltay + 1 ,table);  
  
}  
  
void line1(int x0, int y0, int x1, int y1,struct lookuptable *table) {  
    int Dx = x1 - x0;
```

```
int Dy = y1 - y0;
int steep = (abs(Dy) >= abs(Dx));
if (steep) {
    SWAP(x0, y0);
    SWAP(x1, y1);
    // recompute Dx, Dy after swap
    Dx = x1 - x0;
    Dy = y1 - y0;
}
int xstep = 1;
if (Dx < 0) {
    xstep = -1;
    Dx = -Dx;
}
int ystep = 1;
if (Dy < 0) {
    ystep = -1;
    Dy = -Dy;
}
int TwoDy = Dy<<1;
int TwoDyTwoDx = TwoDy - (Dx<<1); // 2*Dy - 2*Dx
int E = TwoDy - Dx; //2*Dy - Dx
int y = y0;
int xDraw, yDraw;
int x;
int prev_YDraw;
prev_YDraw=yDraw;
for (x = x0; x != x1; x += xstep) {
    if (steep) {
        xDraw = y;
        yDraw = x;
    } else {
        xDraw = x;
        yDraw = y;
    }
    // plot
    if (table[yDraw].x1 == 0){
        table[yDraw].x1 = xDraw;
        table[yDraw].in_x1=xDraw-Halowidth;
    }
    if (xDraw < table[yDraw].x1){
        table[yDraw].x1 = xDraw;
    }
}
```

```
        table[yDraw].in_x1=xDraw-Halowidth;
    }
    if (table[yDraw].x1<0){
        table[yDraw].x1 = 1;
        table[yDraw].in_x1=1;
    }

    if(xDraw>0){
    if (table[yDraw].x2 == 0){
        table[yDraw].x2 = xDraw;
        table[yDraw].in_x2=xDraw+Halowidth;
    }
    if (xDraw > table[yDraw].x2){
        table[yDraw].x2 = xDraw;
        table[yDraw].in_x2=xDraw+Halowidth;
    }
    if (table[yDraw].in_x2>639){
        table[yDraw].x2 = 639;
        table[yDraw].in_x2=639;
    }
    }

    // next
    if (E > 0) {
        E += TwoDyTwoDx; //E += 2*Dy - 2*Dx;
        y = y + ystep;
    } else {
        E += TwoDy; //E += 2*Dy;
    }
}

int sqrt(int num) {
    int op = num;
    int res = 0;
    int one = 1 << 14; // The second-to-top bit is set: 1L<<30 for long

    // "one" starts at the highest power of four <= the argument.
    while (one > op)
        one >>= 2;
```

```
while (one != 0) {  
    if (op >= res + one) {  
        op -= res + one;  
        res += one << 1;  
    }  
    res >>= 1;  
    one >>= 2;  
}  
return res;  
}
```

END OF CODE