RUN, STEPHEN, RUN:
Shoot First, Ask Questions Later
A laser-guided, USB missile launching system.

Andrew Bui*, Cathy Chen†, Johnny Chin*, Andrew Sabatino*, Michael Scott†
***Department of Electrical Engineering**
†Department of Computer Engineering

School of Engineering and Applied Science,
Columbia University in the City of New York

Friday, May 14, 2010

Abstract:

What follows is the design of a target acquisition and missile launching system. The main goal of this project is to create a platform that locates and shoots a non-lethal dart at a pre-determined target. This project utilizes both the hardware and software capabilities of the Altera DE2 Board, while also using principals of ballistics, computer vision, and triangulation. Each system and all necessary algorithms are described in detail, followed by all coding (in C) and hardware description (in VHDL) files necessary for full implementation. Please note that all files automatically generated by the development packages have been omitted for concision.

Introduction:

This project aims to create an automatic target location and missile launching robot on the FPGA. Through a camera interface, the device will be able to locate the position of a target, calculate its location in relation to the missile launcher, acquire a “lock” on the target and eliminate the threat. We have purchased a Dream Cheeky missile launcher that can rotate in both pitch and pan that we plan to use for this project, as well as a Digital Peripheral Solutions QS903C Mini Color Video Camera with Audio for vision.

LASER PATTERN DETECTION

Requirements:

- Up to 3 laser dots will be detected within the camera's field of view.
- The center coordinates of the dot(s) will be calculated and output in x,y pairs.
- Dots can be collinear.
- Speed is essential, and x,y pairs must be output before the completion of each frame.
- Laser dots must be distinguished from other light sources by their size. Ie. A light bulb present in the field of view will similarly saturate the pixel values, but will not register as a laser dot. See picture 3.

Assumptions:

- Most sources of light have radial symmetry. A laser pixel will be approximated as a square inscribed within the true bounding circle.
  This assumption is violated when shadows are cast by a source of light that is projecting into the field of vision (rather than a light source that is itself present in the field of vision). Such light, however, rarely thresholds as laser light.
- Laser dots appear as roughly the same size in multiple camera views. This is fairly consistent, with the minimum and maximum accepted size depending on the distance of the camera from the projected dot.
- Laser dots appear within a fairly tight range of RGB values against many different backgrounds and in different lighting. This has been verified experimentally. See pictures 1 and 2.
- Very little noise in a frame will have the same RBG values as a pixel of laser light. Therefore, there is minimal risk of noise appearing to be a laser dot.

PROTOTYPE: FINDING A SINGLE DOT IN MATLAB

```
function [] = laserpoint(point, picture)

counter = 0; %this will count 100 pixels in a 10x10 square

x = 0; %final x coordinate
y = 0; %final y coordinate
d = size(point); %dimensions of test point file
vert = d(1); %vertical length of test point file
horiz = d(2); %horizonal length of test point file
red = 0; %will be average red value of test point file
green = 0; %average green value in test point
blue = 0; %average blue value in test point
totalpix = vert * horiz; % the total number of pixels in test point
array

columns = sum (point,1);
total = sum (columns,2);

red = total(1)/totalpix;
green = total(2)/totalpix;
blue = total(3)/totalpix;

disp(['The target red value is ' num2str(red)]);
disp(['The target green value is ' num2str(green)]);
disp(['The target blue value is ' num2str(blue)]);

%red values appear to be 240 - 255
%green values appear to be 250 - 255
%blue values appear to be 230 - 255
%we are operating under the assumption that these values are precise
but
%not accurate, meaning that this range will be consistent, but will
center
%on different means with different cameras and in different lighting

%in other words, the laser point is a very white point in the camera
image
%with a very high degree of red, although the green and blue values
are
%occasoinally lower

e = size(picture); %dimensions of the picture
picvert = e(1); %vertical length of the picture
pichoriz = e(2); %horizontal length of the picture

disp(['vertical pixels: ' num2str(picvert)]);
disp(['horizontal pixels: ' num2str(pichoriz)]);

%when counter = 100, we have found a 100 pixel square matching a laser
%point in all three colors, and must break out of every loop
```

```
for i = 1:1:(picvert-10); %go down all vertical pictures looking for
square
    for j = 1:1:(pichoriz-10); %go across all horizontal pictures
        redvalue = picture(i,j,1);

        if (redvalue >= (red - 15) && redvalue <= 255)
            %we have found a possible laser pixel
            %now we make sure 10 pixels to the right and tex pixels
down in
            %a square also lie within this value
            %i and j will remain contant in the next two for loops,
which
            %will instantiate two more iterators to check the square
            for m = 0:1:9;
                for n = 0:1:9;
                %this covers every pixel in a 10X10
                %square starting with the triggered pixel

                sqvaluered = picture((i+m),(j+n),1);
                sqvaluegreen = picture((i+m),(j+n),2);
                sqvalueblue = picture((i+m),(j+n),3);
                    if (sqvaluered >= (red - 15) && sqvaluered <= 255)
                       if(sqvaluegreen >= (green - 15) &&sqvaluegreen
<= 255)
                          if (sqvalueblue >= (blue - 25)&&sqvalueblue
<= 255)
                            counter = counter + 1;
                                if(counter == 100)
                                break;
                                end
                            end
                        end
                    end
                end
                if(counter == 100)
                break;
                end
            end
          end
      if(counter == 100)
      break;
      end
    end
if(counter == 100)
break;
end
end

%if counter has become 100, i and j are the x and y coordinates of the
%upper left corner of a laser point
```

```
x = i + 5;
y = j + 5;

disp(['x is ' num2str(x)]);
disp(['y is ' num2str(y)]);
```

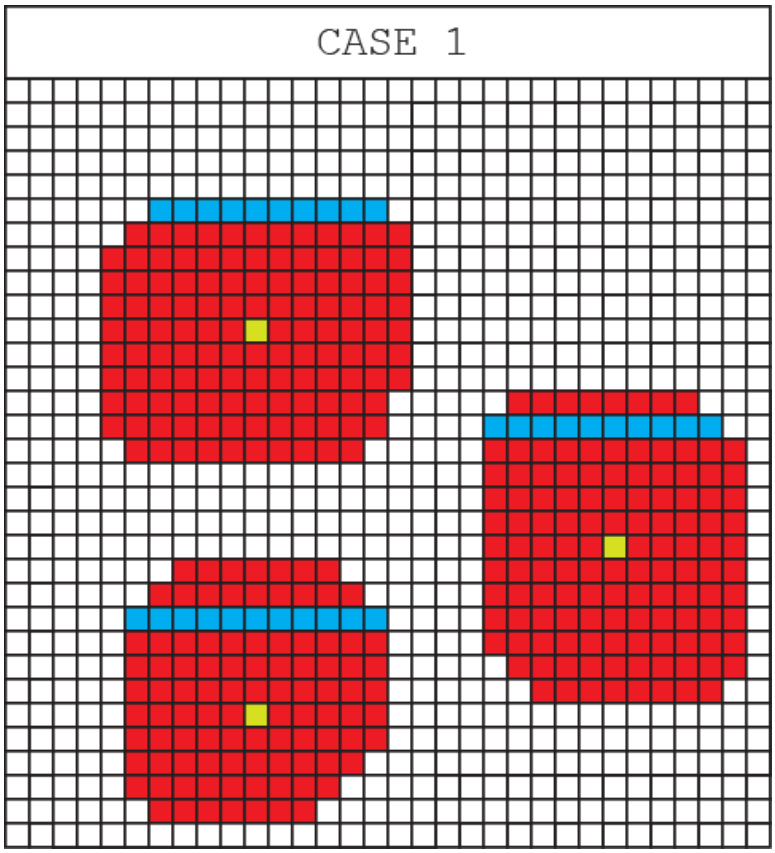
Algorithm in pseudocode:

Processing will be handled on a pixel by pixel basis, moving across subsequent rows.

```
//inputs:
vpos = vertical position of pixel
xpos = horizontal position of pixel
R = Red value
G = Green value
B = Blue value

//outputs:
x1,y1,x2,y2,x3,y3 //the x and y coordinates of up to three laser
points
//will be zero if no laser point is found

laser_pixel = boolean //true if the pixel is a laser pixel
flag1 = boolean //true if found 1st dot
flag2 = boolean // true if found 2nd dot
flag3 = boolean //true if found 3rd dot
Rmin = 236
Rmax = 255
Gmin = 221
Gmax = 255
Bmin = 220
Bmax = 255

if(Rmin <= R <=Rmax){
      if(Gmin <= G <= Gmax){
            if(Bmin <= B <= Bmax){
            laser_pixel = true;
            }
      }
}

if(laser_pixel and !flag1){
      dot1 = dot1++
}
```

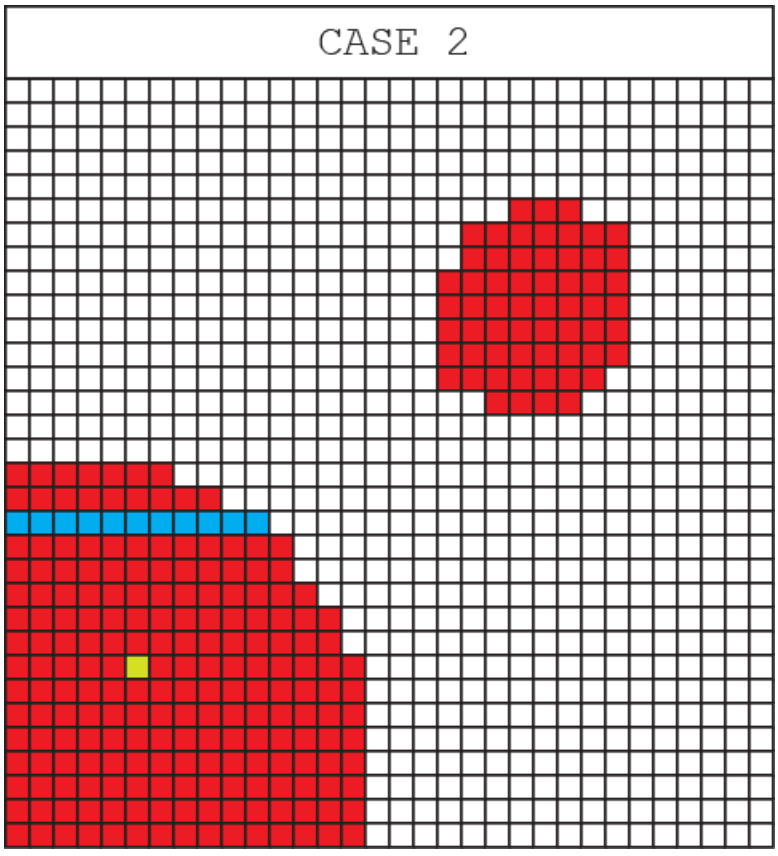
```
else(!laser_pixel && dot1 != 1 && dot1 < pix_min){
      dot1 = 0;
}
else(!laser_pixel and dot1 > pix_max){
      dot1 = 0;
}
else(!laser_pixel and dot1 <=pix_max and dot1 >= pix_min){
      flag1 = true;
      x1 = vpos - (dot1 / 2);
      y1 = hpos - (dot1 / 2);
}
//now continue in an analogous fashion to determine x2, y2, x3, and
y3.
```

FOUR DIFFERENT CASES

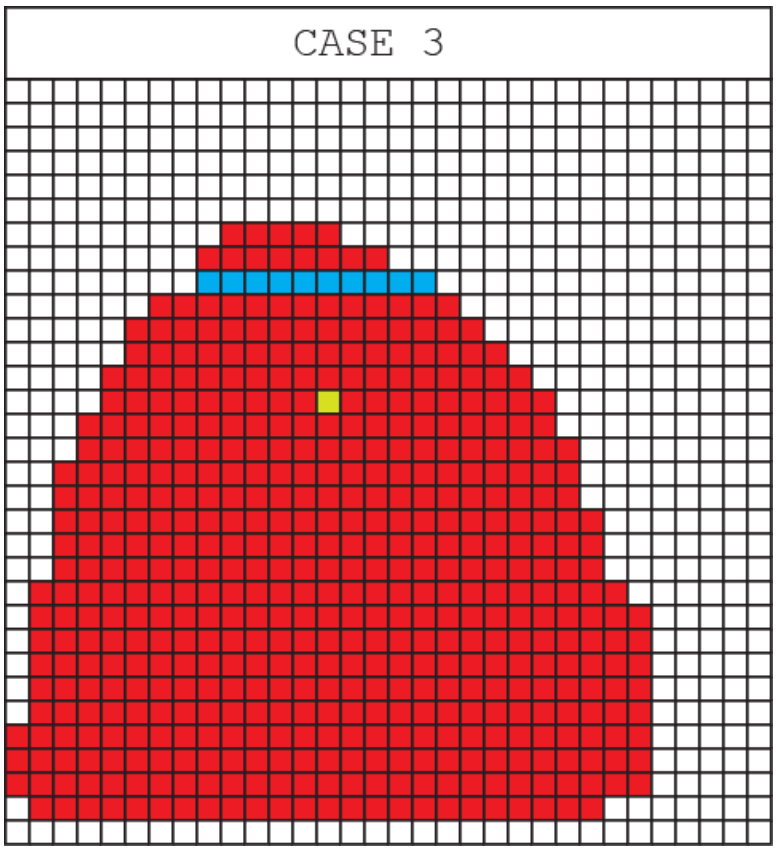


In the four following cases, laser pixels are represented as red dots against a white background. The blue pixels are laser pixels which, taken together, form a row which has been used to calculate the location of a laser dot. The light green pixel represents a calculated center.

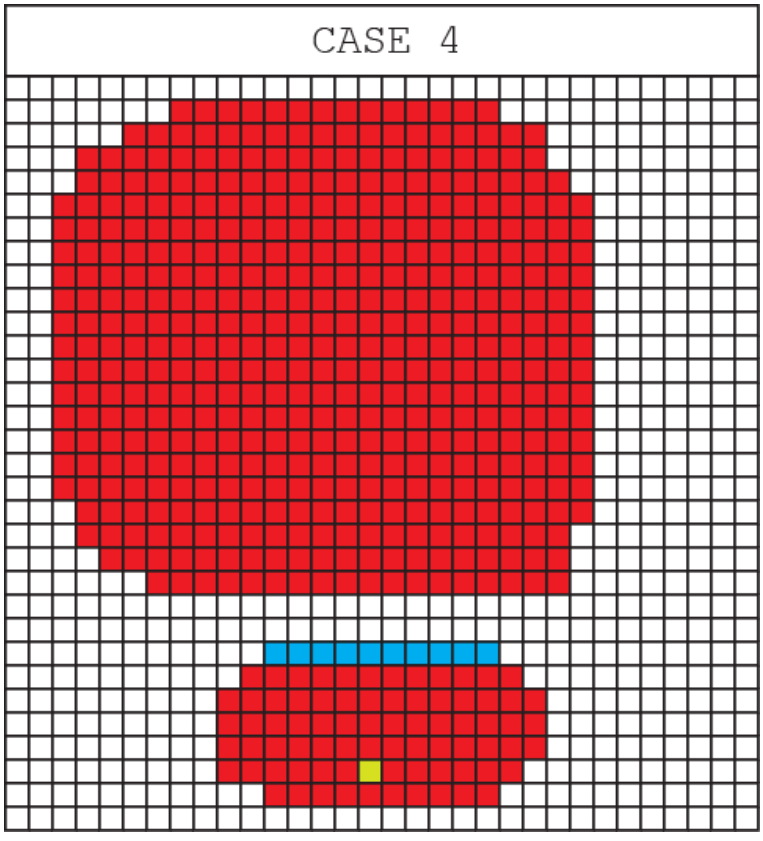
Case 1 is an ideal, in which 3 dots are fairly symmetric and lie entirely within the field of view. Note that all 3 of the dots coincide with another dot for a minimum of 3 lines.



Case 2 presents what is obviously a large light source that overlaps the field of view. The partial overlap is indistinguishable from a laser dot present at the edge of the field, and a center is produced as a result. Note also the small dot which is below the size threshold is and ignored. The observed incidence of noise in a photo lying within the laser color range is very low.



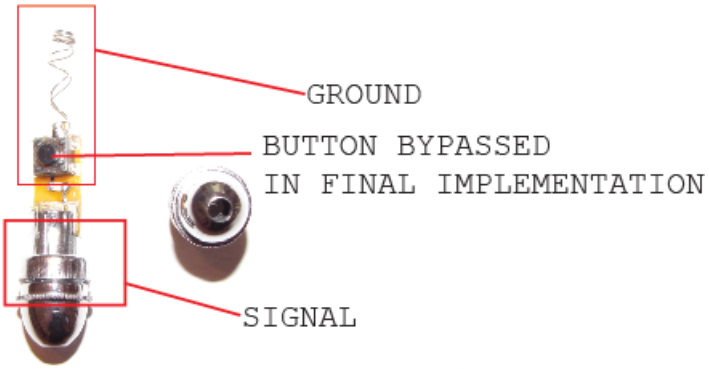
Case 3 violates the assumption of a symmetrical projection, resulting in failure to identify a light source far exceeding the size of a laser point. The top of the large, oblong projection registers as a dot, and the center point is calculated before the bottom is processed.

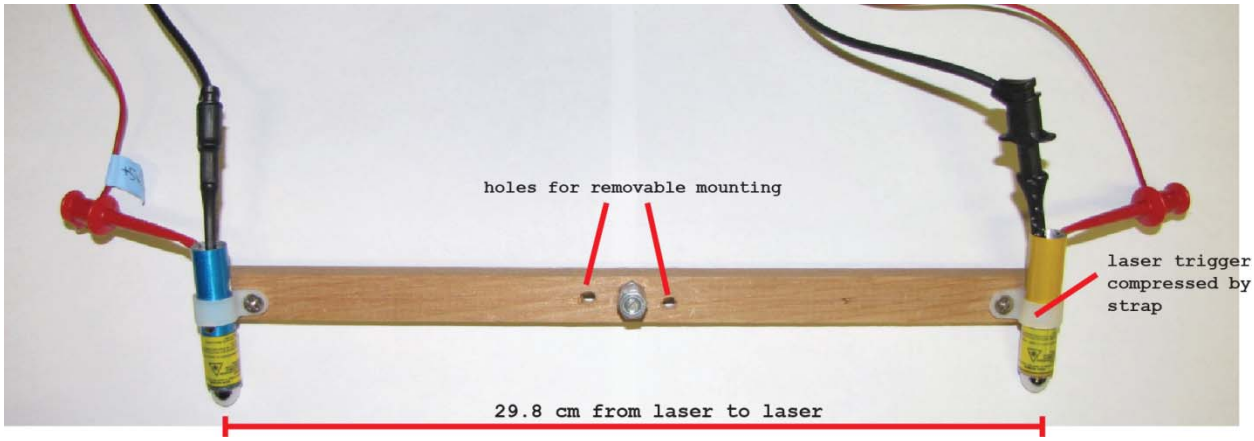


Case 4 demonstrates the successful rejection of a light source far exceeding a laser point in size. Note that the small dot on the bottom violated the assumption of symmetry, and its horizontal elongation results in an off-center point. Only in very extreme cases would this result in a point lying outside of the actual dot, and such a deviation is far outside the realm of accuracy of the USB missile launcher.

LASER SIGHTS

The laser targeting and calibration was accomplished with inexpensive, store-bought laser pointers simply wired to a shared button and power supply for convenience. The deconstructed laser pointer is shown below, with the case supplying the 4.5V signal and the center spring providing ground. The devices proved too delicate to use extracted when from their cases. Two lasers were mounted for sighting on a removable boom, screwed to the top of the USB missile launcher, powered from an external 3.5V – 5V supply. The centers of each laser are 29.8 cm (~30 cm) apart.





RANGE FINDING

One of the principal challenges in controlling the launcher is the notion of distance; without knowledge of the distance to the wall (and subsequently target), it is impossible to position and fire the launcher with any hope of accuracy. The main challenge lies obtaining three-dimensional position data, as it is captured on a two-dimensional CCD sensor. As a result, depth perception is lost, and must be measured externally.

The solution decided upon was to use laser dot rangefinding, where two lasers of a known distance apart are projected onto the surface whose distance is to be determined. This was decided upon because it is possible to finesse a great deal of the computation out of the software by loading the filtering techniques for threshold and RGB rejection/filtering onto the DE2. From the perspective of the camera, these dots will appear to be a shorter distance apart (as measured in pixels). This corresponds to the CCD sensor taking in a wider portion of the field when compared to the laser dots, implying that the physical distance spanned by the CCD's field of view (FOV) is greater. Quantifying this perspective allows the distance *from the CCD* to be determined. This relationship must be simplified with some approximations in order for the equations to have closed-form solutions.

The first assumption is that the camera is always focusing on the laser dots at S = infinity, so the lens focal length can be determined from the datasheet, which reads 6mm. However, because we are always assuming that objects are distant, the focal length doesn't actually factor into the calculations. We also posit that the effects of field curvature do not affect the perceived distance in pixels, though this would obviously lead to some measurement errors near the edges of the sensor. An imposed constraint is that the line of sight (LOS) of the CCD and the laser beams are nearly parallel. This simplifies the calculations and ensures that the pixel distance presented to the CCD is an accurate portrayal of the real-world distance.

This allows us to make the key assumption that there is a linear relationship between the the angle subtended by the laser dots and the angle subtended by the entire FOV. This relationship can be expressed as:

$$\frac{d_x}{W_{CCD}} = \frac{\beta}{FOV}$$

where d_x and W_CCD are the pixel distances of the laser dots and the CCD (the width of the sensor) and β and the FOV are the respective angles subtended by the laser dots and CCD. The FOV of the CCD was measured to be approximately 33 degrees, which is close enough to the datasheets value of 42 to be used in subsequent calculations.

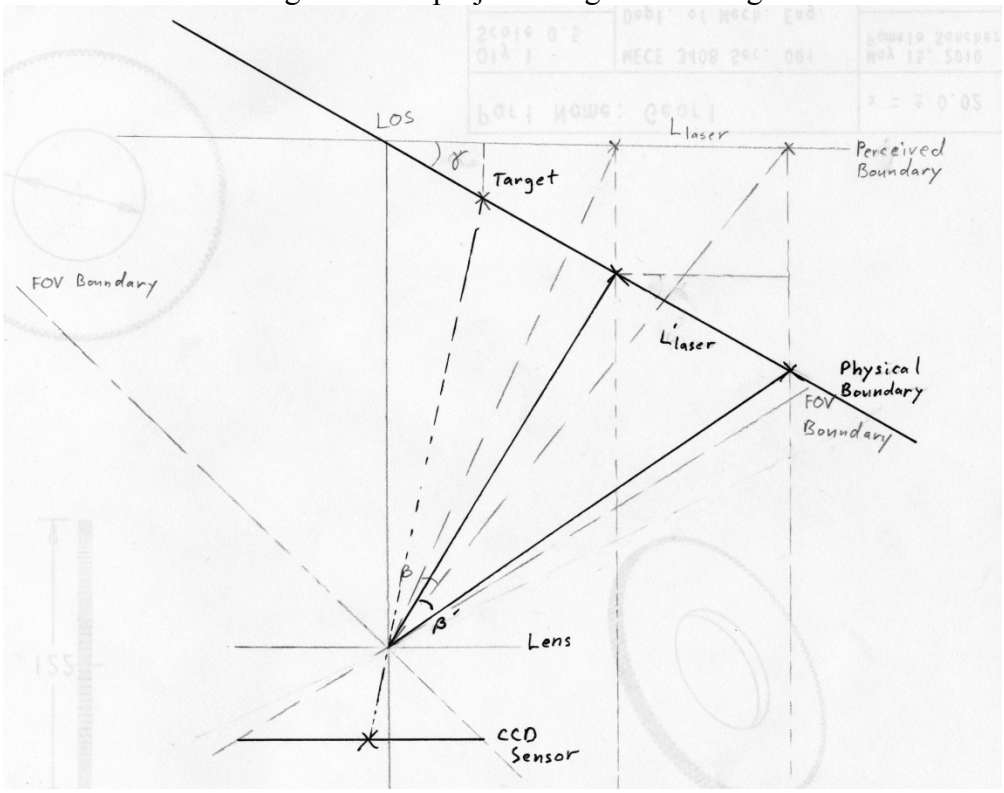
Knowing the angle subtended by the dots allows us to use the simple geometric expression

$$\frac{L_{laser}}{2D_{wall}} = \tan\frac{\beta}{2}$$

and the distance from the CCD is calculated.

Though these quantities are calculated assuming the LOS of both the launcher and CCD are coincidental, but because of parallelism constraint, this equation holds for any horizontal and vertical displacement of the laser dots. Of course, this introduces an offset error which would be handled on the ballistics side to correct for the non-conciding viewpoint. Because the perspective is always measured relative to the camera, it holds that two laser points are redundant, as they are indicative of the same distance from the wall.

Expanding upon the idea of CCD perspective, it is possible to quantify the angular displacement of the launcher and camera relative to the wall (keeping the camera and launcher LOS's parallel). We can model this as a rotation of the physical wall boundary through an angle γ (when looking at the CCD and camera from an overhead perspective). The perceived distance is seen to be the "straight-ahead" projection against the angled wall. This is illustrated below:



Observations find that lengths in the field, under the linear approximation, can be corrected by scaling by a factor of sec(γ). Similarly, the distance from the LOS and the depth for any given point (i.e. target) must be corrected for because we have imposed a rotation about the unrotated coordinate system, and the CCD still sees a 2D projection:

$$\begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix} \begin{pmatrix} W \\ D \end{pmatrix} = \begin{pmatrix} W_{corr} \\ D_{corr} \end{pmatrix}$$

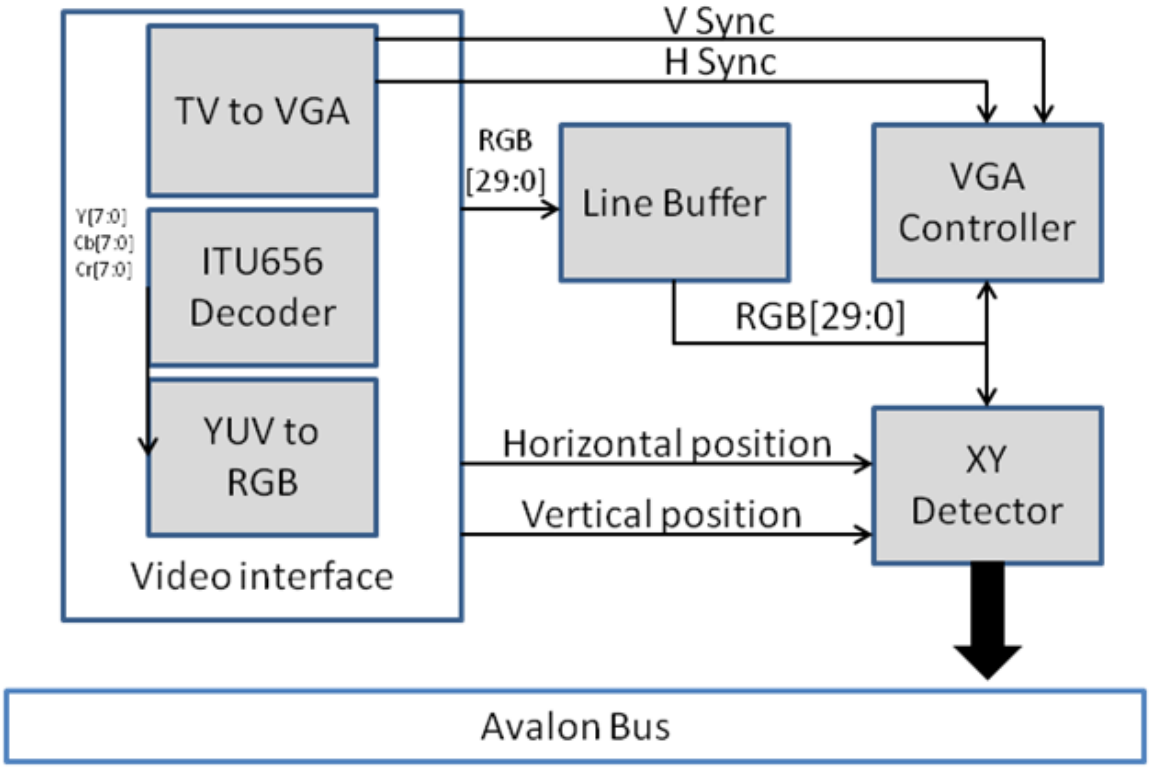
However, these equations are calculated based on the prior knowledge of the rotation angle γ, which cannot be ascertained from any single measurement in the CCD image without some "anchor" of a known physical distance. This is the original assumption upon which the system was based, and this posits that the problem becomes unconstrained under arbitrary initial conditions.

The code used to the implement the equations is included; it calculates the distance as well as the positive of negative x and y offsets, all in real-world distances. As this ultimately could not be tested against the actual CCD measurements, they are only implementations of the theory.

DATA ACQUISITION

VISUAL DATA ACQUISITION

To acquire the distance between the target and the launcher, visual information is needed for the computation of the positions. A camera with NTSC output is connected to the video input of the DE2 port. The data stream is processed through the ADV7181 video decoder chip on the board. The ADV7181 operates on a 27MHz clock and produces an 8-bit output encoded in ITU656 standard (YUV 4:2:2). Originally modules were to be designed to process this output and convert it to RGB color space. However, due to the difference in the refresh rate of the NTSC and VGA, synchronization of the components causes many problems. Three modules from the Altera DE2_TV demonstration files are ported for the purpose of data conversion and synchronization of the video input and the VGA display. The ITU656 decoder module converts this 8-bit input into three 8-bit signals for Y, Cb and Cr (YUV 4:4:4 standard), which provide the brightness (Y component) and chrominance (Cb and Cr components) information of the pixel. The 24-bit output is fed to the YUV2RGB module that translates the data from YUV to RGB color space. This data output is then supplied to a line buffer module that allows writing to one line and reading from the other for the VGA display. Module XY detector also takes the output from the line buffer and the video interface module that supplies the x and y positions of the pixel to determine the laser point coordinates.



The Rocket Launcher

Using the rocket_launcher.c test code found online, we were able to determine that the rocket launcher was the second version, with

Vendor ID	**0x0A81**
Product ID	0z0701
Manufacturer String	Rocket Baby
Product String	Rocket Baby
Version	2
Serial Number	/

and the following control endpoint:

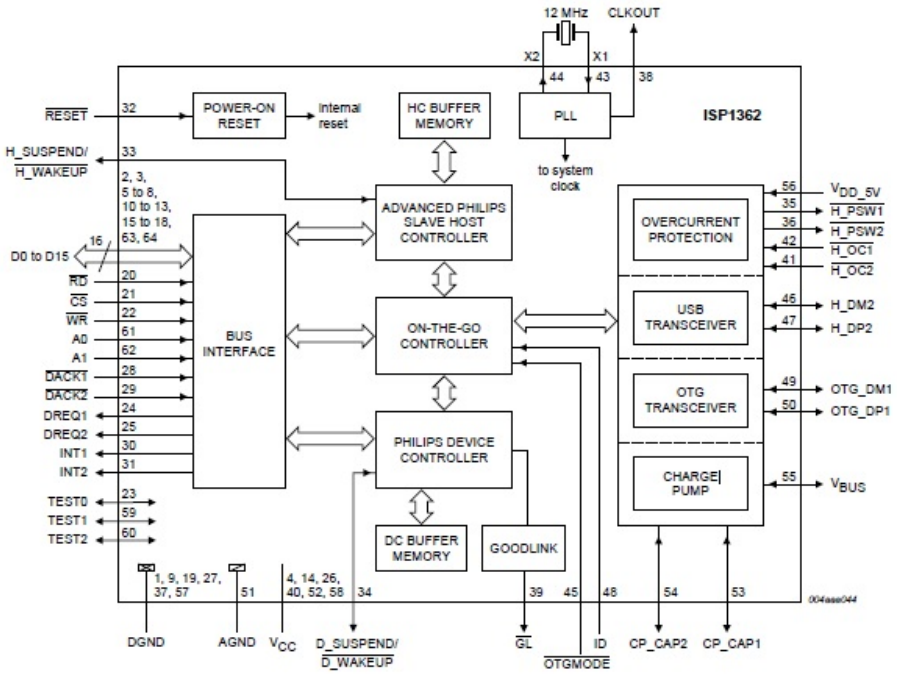
Byte 0	Description
0x01	Down
0x02	Up
0x04	Left
0x08	Right
0x10	Fire
0x40	Get Status
0x20	Stop

Using lsusb in Ubuntu, we were able to infer that the rocket launcher had 1 endpoint and that it's statistics were as follows:

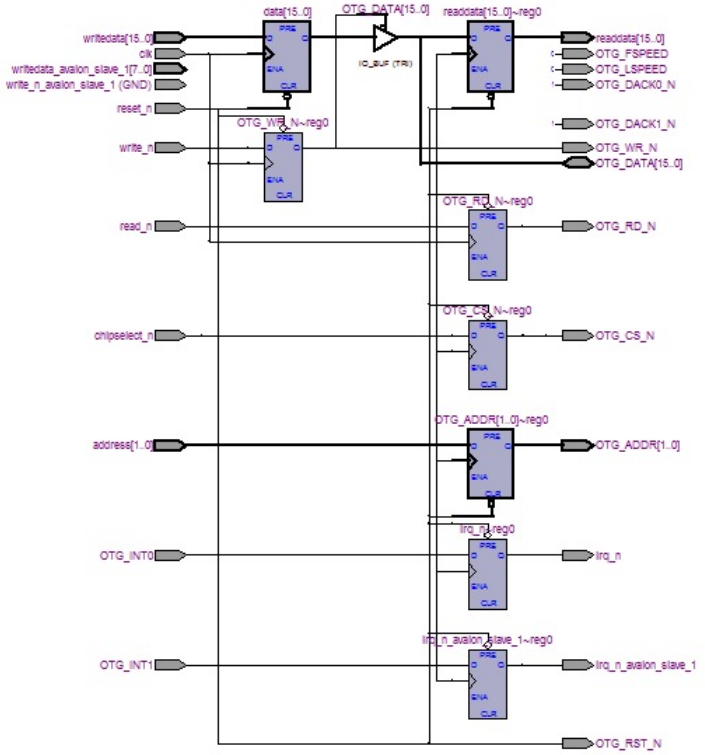
Endpoint Descriptor:
bLength 7
bDescriptorType 5
bEndpointAddress 0x81 EP 1 IN
bmAttributes 3
Transfer Type Interrupt
Synch Type None
Usage Type Data
wMaxPacketSize 0x0001 1x 1 bytes
bInterval 20
Device Status: 0x0000
(Bus Powered)

USB CODE

The Rocket Launcher was controlled through the on-board Philips ISP1362 chip on the DE2 Development board. The isp1362 chip can be set to be both a host controller and a device Controller. However in this case, it was only used as a host controller.



A chip controller (isp1362_cntrl.v) was built in the Quartus design tool and added to a NiosII chip with sram. This controller acted as an interface between the niosII software and the ISP1362 chip. The block diagram of the controller is as follows:



Additionally, the timing of the Avalon Bus had to be changed in order for the chip to function correctly. The time specification is as follows:

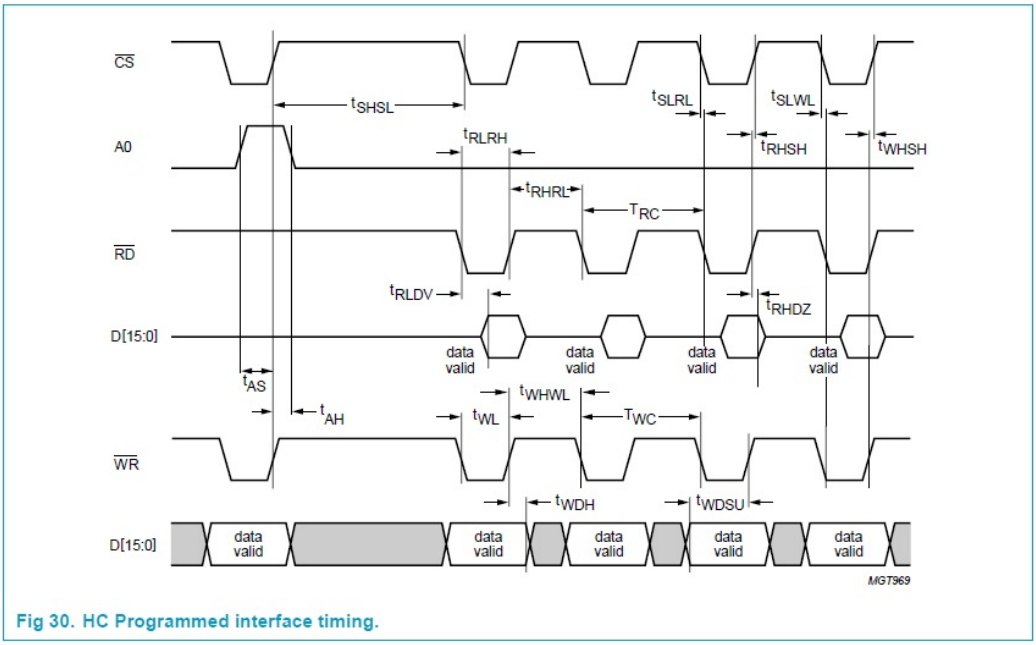


Fig 30. HC Programmed interface timing.

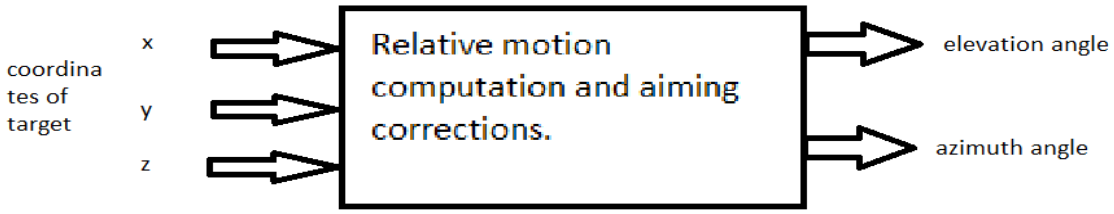
The chip is addressed through 2 address lines for the avalon_slave0 bus. These address lines write to a host controller (HC) data port, an HC command port, a DC data port and DC command port. For the purposes of this project, only the HC data ports were utilized due to the fact that the chip was only used as host controller. Following the embedded programming guide, we wrote functions that would set the chip into operational mode, and enable the HC registers on the chip to be written. Using these functions (hc_write and hc_read), the chip was configured to the HC mode. Two other important features are the make_ptd() and send_control() functions. The PTD is a buffer that is used to set up the packet to be sent to the chip. After the ptd is made, the send_control() function takes that PTD, sets up the correct buffers and takes the buffer contents to write to the device. This is used in the get_control() function that gets a device descriptor from the device. After receiving the device descriptor and analyzing it for correctness, the USB host then assigns it an address. Assuming all of this occurs without any errors, the device is now set up for transfer. Now, to control the rocket launcher, control endpoints need to be sent to it, using the make_int_ptd() and sent_int() functions, using the codes as seen in the table in the rocket launcher section.

BALLISTICS ALGORITHMS AND CALCULATIONS

External ballistics concentrates on the natural phenomena that cause projectile in flight to change direction and/or velocity. These natural phenomena include gravity, drag, wind, spin drift and coriolis effect. The main force acting on the darts used in the launcher, which affects its trajectory, is gravity. Other forces acting on the darts are due to their shape and air resistance (medium in which they travel) which also accounts for the deviation of the trajectory predicted by relative motion equations derived using the effects of gravity.

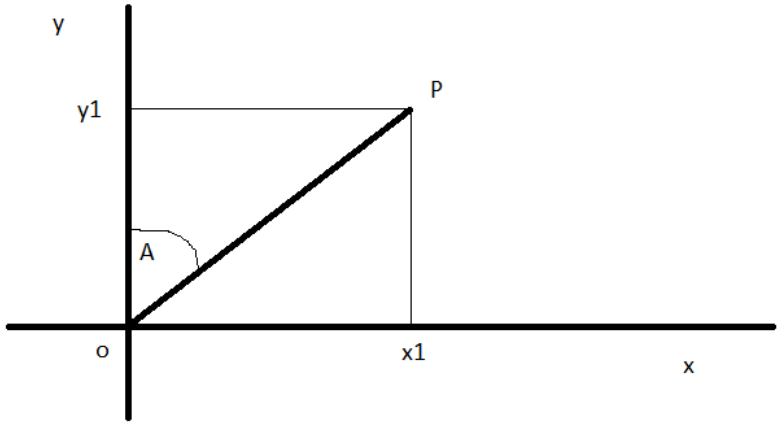
RELATIVE-MOTION CALCULATION

The figure below is a block diagram of the flow of acquiring a target, selecting the appropriate relative motion equations and compensations to predict a suitable trajectory to hit the target. These equations will produce the elevation and azimuth angles in which the launcher should be set to achieve the desired trajectory.



AZIMUTH CALCULATION

The following graph shows the calculations necessary to obtain the azimuth angle in which the launcher must be positioned given the x and y coordinates of the target.



tan a = x1/y1

⇨ $a = \tan^{-1} (x1/y1)$

where $-90^0 <= a <= 90^0$

MEASUREMENTS & CALCULATIONS FROM THE LAUNCHER

The following procedures were used to get the appropriate measurements and calculations necessary to predict the trajectory and hence the elevation and azimuth angles.

- ⇨ Measure step angles for elevation and azimuth.
- ⇨ Measure range at 21^0 elevation and 0^0 azimuth for each dart.
- ⇨ Measure azimuth deviation from 0^0 expected target.
- ⇨ Calculate u for each dart/position using the Range measurement.
- ⇨ Calculate H at 21^0 elevation & 0^0 azimuth.

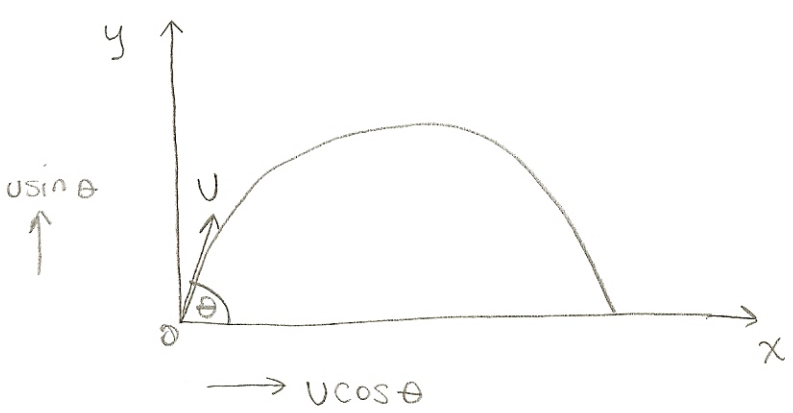
Target Test

- ⇨ Set (x,y) coordinates of the target at 0^0 azimuth .
- ⇨ Calculate launch elevation for each dart / position using
- ⇨ For each dart set its respective elevation angle and compensation for azimuth deviation
- ⇨ Fire at the target and measure the x & y deviation from the target.

NOTES

1. Reset does not always set launcher to the starting position. lt sometimes takes two tries to reset.
   [Run reset twice (program)]
2. Reset or hard reset (unplug) does not reset the firing sequence. Sequence is Right, Top,Left facing the launcher.
   [Set sequence before, use RTL order]
3. The same dart has to be used in the same location and orientation for optimal accuracy.
   [Label darts and load in same position]
4. Pressing the direction button once does not always move the launcher by the same distance (i.e. by one position at a time).
   [More testing]
5. Darts are very sensitive and produces different results if altered.
   [Handle with care]
6. When loading the darts, the position of the launcher can change (i.e. sensitive to the touch)
   [Handle with care and do a reset after loading darts]
7. First move left by 1 does not move as much as subsequent left moves.
   [More testing]

Effect of Gravity on Trajectory



$$y_{max} = H = (U^2 \sin^2\theta)/2g$$
$$x_{max} = R = (U^2 \sin 2\theta)/g \Rightarrow U = \sqrt{(Rg)/\sin 2\theta}$$

} H_{max} & R_{max} occurs at 45°

$$x = u\cos\theta \cdot t \quad \& \quad y = u\sin\theta \cdot t - \tfrac{1}{2}gt^2$$

solve simultaneously using t:

$$y = x\tan\theta - \frac{1}{2}gx^2\left(\frac{1}{U^2\cos^2\theta}\right)$$
$$= x\tan\theta - \frac{gx^2}{2U^2}(1+\tan^2\theta)$$

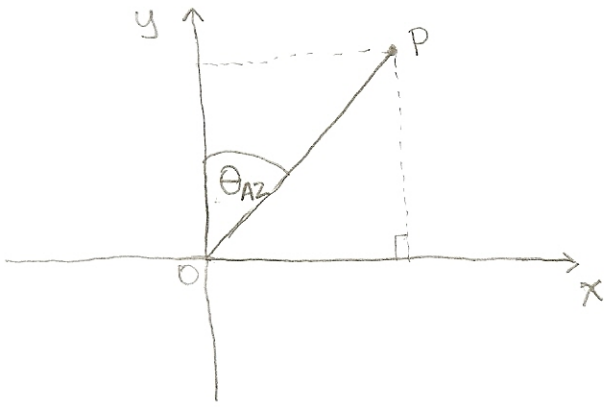
let $x = k_1$, $\frac{gx^2}{2U^2} = k_2$ & $t = \tan\theta$

$$y = -k_2t^2 + k_1t - k_2$$
$$\Rightarrow -k_2t^2 + k_1t - k_2 - y = 0$$

let $k_3 = k_2 + y$

$$\Rightarrow k_2t^2 - k_1t + k_3 = 0$$
$$t = \frac{k_1 \pm \sqrt{k_1^2 - 4k_2k_3}}{2k_2} = \tan\theta$$
$$\theta = \tan^{-1}\left(\frac{x \pm \sqrt{x^2 - \frac{2gx^2}{U^2}\left(\frac{gx^2}{2U^2} + y\right)}}{gx^2/U^2}\right) = \tan^{-1}\left(\frac{U^2 \pm \sqrt{U^4 - g(gx^2 + 2yU^2)}}{gx}\right)$$

Azimuth Calculation

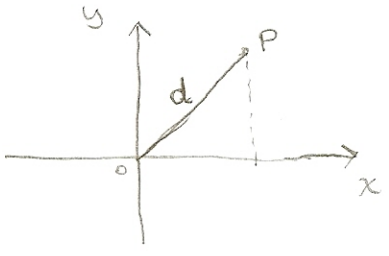


$$\tan\theta_{AZ} = \frac{x}{y}$$

$$\Rightarrow \theta_{AZ} = \tan^{-1}\left(\frac{x}{y}\right)$$

where $-90^\circ \leq \theta_{AZ} \leq 90^\circ$

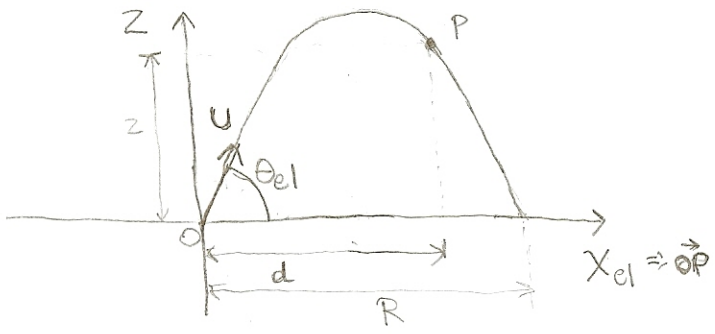
Elevation Calculation



$$d = \sqrt{x^2+y^2}$$

$$\vec{OP} \Rightarrow \vec{x_{el}}$$

$$\vec{oZ} \Rightarrow \vec{y_{el}}$$



$$\theta_{el} = \tan^{-1}\left(\frac{d \pm \sqrt{d^2 - \frac{2gd^2}{U^2}\left(\frac{gd^2}{2U^2}+z\right)}}{gd^2/U^2}\right)$$

where $-3^\circ \leq \theta_{el} \leq 21^\circ$

up elevation (in degrees)		11	13	15	17	19
down elevation (in degrees)		11	9	7	5	3
left and right azimuth		range -90 to 90 step 5				
Try		direct distance(cm)	deviated distance(cm)	angle of deviation (cos)	angle of deviation (sin)	Dart position
1		254	26	90.00001872	5.875214261	R
2		261.5	9.5	90.00001872	2.081949593	R
3		262	5	90.00001872	1.093497516	R
1		225.25	-12	90.00001872	-3.053829698	T
2		211.5	-18	90.00001872	-4.882143348	T
3		217.5	-10.5	90.00001872	-2.767079242	T
1		180.75	4.5	90.00001872	1.426598638	L
2		180.25	2	90.00001872	0.635749982	L
3		187.25	9.5	90.00001872	2.908111059	L
		direct distance (m)	deviated distance (m)		angle of deviation (sin)	
1		6.4516	0.6604		5.875214261	R
2		6.6421	0.2413		2.081949593	R
3		6.6548	0.127		1.093497516	R
1		5.72135	-0.3048		-3.053829698	T
2		5.3721	-0.4572		-4.882143348	T
3		5.5245	-0.2667		-2.767079242	T
1		4.59105	0.1143		1.426598638	L
2		4.57835	0.0508		0.635749982	L
3		4.75615	0.2413		2.908111059	L

21									
1	-1	-3							
elevation									
21			0.73303813						
21									
21									
21									
21									
21									
21									
21									
21									
elevation	u (m/s)	Height (m)	Height (in)	x (in)	y (in)	x (m)	y (m)	elevation	elevation (deg)
21	9.720562026	0.619134038	24.3753558	118.110236	11.8110236	3	0.3	0.260571	14.9296376
21	9.863030227	0.637415554	25.0951006	118.110236	11.8110236	3	0.3	0.255716	14.65146009
21	9.872455007	0.638634322	25.1430836	118.110236	11.8110236	3	0.3	0.255403	14.63352204
21	9.153915884	0.54905489	21.6163342	118.110236	11.8110236	3	0.3	0.282446	16.18295917
21	8.87012425	0.515538775	20.2968022	118.110236	11.8110236	3	0.3	0.295227	16.91527505
21	8.995061731	0.530163989	20.872598	118.110236	11.8110236	3	0.3	0.28943	16.58312978
21	8.199996566	0.440584556	17.3458487	118.110236	11.8110236	3	0.3	0.331836	19.01280947
21	8.188647084	0.439365789	17.2978657	118.110236	11.8110236	3	0.3	0.33255	19.05373779
21	8.34613549	0.456428538	17.9696275	118.110236	11.8110236	3	0.3	0.32295	18.50369793

step	actual elev	
2	15	
2	15	
2	15	
3	17	
3	17	
3	17	
4	19	
4	19	
4	19	

LESSONS LEARNED:

As with all projects, start earlier than you think you need to and stick to milestones. Integrate work with your partners sooner, and maybe schedule weekly meetings with required attendance. Splitting up the sections for individuals to work on was a good idea, we just didn't integrate work together as early as we should have. Also, don't get sick during the last week of the project. USB control transfer out protocol on the isp1362 has practically no documentation online, and the embedded programming guide was not as bad as I thought it would be. Start earlier. Everyone has interfaced a mouse with the chip, but no one has tried anything else. USB packet sniffers are very useful. When inconsistently manufactured projectiles are used, each must have a unique set of projectile constants and expectations. Lasers are an excellent distinguishing feature in a frame of otherwise ambient light. Code from outside projects is not a reliable source. Buy extras when structuring a project around plastic toys.

What follows is source code.

```
module TV_to_VGA (
            OSC_27,
            RESET,
            VGA_BLANK,
            VGA_SYNC,
            VGA_HS,
            VGA_VS,
            HSx2,
            TD_VS,
            TD_HS
);
input           OSC_27;
input           RESET;
input           HSx2;
input           TD_VS;
input           TD_HS;
output          VGA_BLANK;
output          VGA_SYNC;
output          VGA_HS;
output          VGA_VS;


wire mTD_HSx2;
reg     [10:0]L_COUNTER;//<<
reg     [10:0]RL_COUNTER;//<<
wire    sync_reset=(RL_COUNTER==9)?1:0;//<<
reg     sync_en;//<<
reg     [7:0]delay;//<<

reg     [9:0]   H_Cont;
reg     [9:0]   V_Cont;
reg             oVGA_H_SYNC;
reg             oVGA_V_SYNC;
reg             Pre_HS;
reg             Pre_VS;
reg             mACT_HS;
reg             mACT_VS;

always@(posedge OSC_27 or negedge sync_en)//<<
begin
    if(!sync_en)//<<
    begin
        Pre_HS      <=  0;
        mACT_HS     <=  0;
        H_Cont      <=  0;
        oVGA_H_SYNC <=  0;
    end
    else
    begin
        Pre_HS  <=  HSx2;
        if({Pre_HS, HSx2}==2'b10)
        mACT_HS <=  1;
        if(mACT_HS)
```

```
54          begin
55              //  H_Sync Counter
56              if( H_Cont < 852 )
57              H_Cont  <=  H_Cont+1;
58              else
59              begin
60                  H_Cont  <=  0;
61                  mACT_HS <=  0;
62              end
63              //  H_Sync Generator
64              if( H_Cont < 96 )
65              oVGA_H_SYNC <=  0;
66              else
67              oVGA_H_SYNC <=  1;
68          end
69          else
70          begin
71              oVGA_H_SYNC <=  0;
72              H_Cont      <=  0;
73          end
74      end
75  end
76
77  always@(posedge OSC_27 or negedge sync_en)//<<
78  begin
79      if(!sync_en)//<<
80      begin
81          Pre_VS      <=  1;
82          mACT_VS     <=  0;
83          V_Cont      <=  0;
84          oVGA_V_SYNC <=  0;
85      end
86      else
87      begin
88          Pre_VS  <=  TD_VS;
89          if({Pre_VS,TD_VS}==2'b01)
90          mACT_VS <=  1;
91          if( (H_Cont==1) && mACT_VS)
92          begin
93              //  V_Sync Counter
94              if( V_Cont < 524 )
95              V_Cont  <=  V_Cont+1;
96              else
97              V_Cont  <=  0;
98              //  V_Sync Generator
99              if( V_Cont < 2 )
100             oVGA_V_SYNC <=  0;
101             else
102             oVGA_V_SYNC <=  1;
103         end
104     end
105 end
106
```

```
107   assign  VGA_HS      =   oVGA_H_SYNC;
108   assign  VGA_VS      =   oVGA_V_SYNC;
109   assign  VGA_SYNC    =   1'b0;
110   assign  VGA_CLOCK   =   OSC_27;
111
112
113   //>>lock detector
114   always @(posedge TD_HS) begin
115   if (TD_VS) L_COUNTER=0;
116               else L_COUNTER=L_COUNTER+1;
117   end
118
119   always @(posedge TD_VS) begin
120           RL_COUNTER=L_COUNTER;//1714
121   end
122   always@(negedge sync_reset or posedge TD_VS) begin
123   if (!sync_reset)
124       delay=0;
125       else if (delay < 250)
126        delay=delay+1;
127   end
128
129   always@(negedge sync_reset or negedge TD_VS) begin
130   if (!sync_reset)
131           sync_en=0;
132   else if (delay < 100)
133       sync_en=0;
134       else
135       sync_en=1;
136   end
137   //<<
138
139   endmodule
```

```
 1
    //*****************************************************************
    **********************
 2      //*  COMPANY    :  TERASIC.  www.terasic.com  (c) 2005 all
    rights reserved.                *
 3      //*  NAME       :  ITU-R BT.656 YCrCb 4:2:2
    DECODER
      *
 4      //*  Created    :
    7/5/2005
       *
 5      //*  Author     :  Joe
    Yang                                                            *
 6
    //*****************************************************************
    **********************
 7
 8  `define sync 8'd101
 9  module itu_r656_decoder
10  (
11      CLOCK,    //system clock
12      TD_D,     //4:2:2 video data stream
13      TD_HS,    //Decoder_hs
14      TD_VS,    //Decoder_vs
15
16      Y,        //4:4:4 Y
17      Cb,       //4:4:4 Cb
18      Cr,       //4:4:4 Cr
19
20      Ypix_clock, //Y pixel clock
21      HSx2,
22      VSx1,
23      Y_check,
24      START,
25      COUNTER,
26      R,G,B,
27      h_tr,
28      SW0,SW1,
29      blank
30  );
31      input CLOCK;
32      input [7:0]TD_D;
33      input TD_HS;
34      input TD_VS;
35      input SW0;
36      input SW1;
37
38      output [7:0]Y;
39      output [7:0]Cb;
40      output [7:0]Cr;
41
42      output HSx2;
43      output VSx1;
```

```
44        output Ypix_clock;
45
46
47    //test
48        output Y_check;
49        output START;
50        output [1:0]COUNTER;
51
52
53        output [9:0]R;
54        output [9:0]G;
55        output [9:0]B;
56        output h_tr;
57        output blank;
58
59        reg [7:0]YY;
60        reg [7:0]CCb,Cbb;
61        reg [7:0]CCr,Crr;
62
63        reg  HSx2 ;//TD_HS;
64        wire VSx1=TD_VS;
65
66        reg[7:0]R1,R2,R3;
67        reg[7:0]RR1,RR2,RR3;
68
69        wire [7:0]cr={2'b0,Cr[6:1]};
70        wire [7:0]cb={3'b0,Cb[6:2]};
71
72        wire [7:0]Rr=(Y-16)-{1'b0,Cr[7:0]};
73        wire [7:0]Gg=(Y-16)-cr -cb;
74        wire [7:0]Bb=(Y-16)-{1'b0,Cb[7:0]};
75
76        wire [9:0]R={Rr,2'b00};
77        wire [9:0]G={Gg,2'b00};
78        wire [9:0]B={Bb,2'b00};
79
80    //
81        wire  Y_check=( (R3==8'hff) && (R2==8'h00) && (R1==8'h00) )?1:0
      ;
82        always @(posedge CLOCK) begin
83           RR1=TD_D;
84           RR2=R1;
85           RR3=R2;
86        end
87
88        always @(negedge CLOCK) begin
89            R1=RR1;
90            R2=RR2;
91            R3=RR3;
92        end
93
94        reg START,Field;
95        always @(posedge CLOCK) begin
```

```
 96       if (Y_check==1)
 97       begin
 98           START=~TD_D[4];
 99           Field= TD_D[6];
100       end
101       end
102
103       reg [1:0]COUNTER;
104       always @(posedge CLOCK) begin
105           if (!START)
106              COUNTER=0;
107               else COUNTER=COUNTER+1;
108       end
109
110       reg Ypix_clock;
111       always @(posedge CLOCK) begin
112           case (COUNTER)
113               0:begin Cbb=TD_D;                   Ypix_clock =0;end
114               1:begin YY  =TD_D;CCr=Crr;CCb=Cbb; Ypix_clock =1;end
115               2:begin Crr=TD_D;                   Ypix_clock =0;end
116               3:begin YY  =TD_D;CCr=Crr;CCb=Cbb; Ypix_clock =1;end

117           endcase
118       end
119
120
121
122   reg [10:0]H_COUNTER;
123   reg [10:0]RH_COUNTER;
124       always @(posedge CLOCK) begin
125           if (TD_HS) H_COUNTER=0;
126               else H_COUNTER=H_COUNTER+1;
127       end
128
129       always @(posedge TD_HS) begin
130           RH_COUNTER=H_COUNTER;
131       end
132
133       always @(posedge CLOCK) begin
134           if (
135               ((H_COUNTER >= 0) && (H_COUNTER < `sync)) ||
136               ((H_COUNTER >= RH_COUNTER[10:1]) && (H_COUNTER < (
      RH_COUNTER[10:1]+`sync+1)))
137               )
138           HSx2=0;
139           else
140           HSx2=1;
141       end
142
143       reg [10:0]h;
144       reg h_tr;
145       reg h_tr_h;
146
```

```
147    always @(posedge CLOCK) begin
148       if(!HSx2) h=0;
149           else
150           h=h+1;
151       end
152
153      always @(posedge CLOCK) begin
154        if ((h< 51) || (h > 771))
155           h_tr=0;
156           else
157           h_tr=1;
158      end
159      always @(posedge CLOCK) begin
160        if ((h< 41) || (h > 781))
161           h_tr_h=0;
162           else
163           h_tr_h=1;
164      end
165
166
167   wire  [7:0]Yw,YYw;
168   wire  [7:0]Cwr,CCwr;
169   wire  [7:0]Cwb,CCwb;
170
171   reg [10:0]pcounter_h;
172   reg [10:0]pcounter_v;
173
174   always @(posedge CLOCK) begin
175   if (!h_tr) pcounter_h=0;
176   else pcounter_h=pcounter_h+1;
177   end
178
179   always @(posedge h_tr) begin
180   if (!TD_VS) pcounter_v=0;
181   else pcounter_v=pcounter_v+1;
182   end
183
184   wire t=(
185   //      ((pcounter_h >=0) && (pcounter_h < 45)  &&
      (pcounter_v[9:5]==5'b00000)) ||
186           ((pcounter_h >=0) && (pcounter_h < 90)  && (pcounter_v[9:5
      ]==5'b00001)) ||
187           ((pcounter_h >=0) && (pcounter_h < 135) && (pcounter_v[9:5
      ]==5'b00010)) ||
188           ((pcounter_h >=0) && (pcounter_h < 180) && (pcounter_v[9:5
      ]==5'b00011)) ||
189           ((pcounter_h >=0) && (pcounter_h < 225) && (pcounter_v[9:5
      ]==5'b00100)) ||
190           ((pcounter_h >=0) && (pcounter_h < 270) && (pcounter_v[9:5
      ]==5'b00101)) ||
191           ((pcounter_h >=0) && (pcounter_h < 315) && (pcounter_v[9:5
      ]==5'b00110)) ||
192           //((pcounter_h >=0) && (pcounter_h < 360) &&
```

```
      (pcounter_v[9:5]==5'b00111)) ||
193           ((pcounter_h >=0) && (pcounter_h < 405) && (pcounter_v[9:5
      ]==5'b01000)) ||
194           ((pcounter_h >=0) && (pcounter_h < 450) && (pcounter_v[9:5
      ]==5'b01001)) ||
195           ((pcounter_h >=0) && (pcounter_h < 495) && (pcounter_v[9:5
      ]==5'b01010)) ||
196           ((pcounter_h >=0) && (pcounter_h < 540) && (pcounter_v[9:5
      ]==5'b01011)) ||
197           ((pcounter_h >=0) && (pcounter_h < 585) && (pcounter_v[9:5
      ]==5'b01100)) ||
198   //      ((pcounter_h >=0) && (pcounter_h < 630) &&
      (pcounter_v[9:5]==5'b01101)) ||
199           ((pcounter_h >=0) && (pcounter_h < 675) && (pcounter_v[9:5
      ]==5'b01110)) ||
200           ((pcounter_h >=0) && (pcounter_h < 720) && (pcounter_v[9:5
      ]==5'b01111))
201           )?1:0;
202   wire [7:0]ym= t? 8'hff:8'h80;
203   wire [7:0]crm=t? 8'h80:8'h80;
204   wire [7:0]cbm=t? 8'h80:8'h80;
205
206   assign  YYw  =((h_tr==1) && (VSx1==1))?Yw : 8'h10;//current set
207   assign  CCwr =((h_tr==1) && (VSx1==1))?Cwr: 8'h80;
208   assign  CCwb =((h_tr==1) && (VSx1==1))?Cwb: 8'h80;
209
210   wire  [7:0]Ymm  =SW1? ym :Yw ;
211   wire  [7:0]Crmm =SW1? crm:Cwr;
212   wire  [7:0]Cbmm =SW1? cbm:Cwb;
213
214   reg[10:0]Hde_counter;
215   reg[10:0]Vde_counter;
216
217   always @(posedge CLOCK)begin
218       if(HSx2==0)
219         Hde_counter=0;
220           else
221         Hde_counter=Hde_counter+1;
222   end
223
224   always@(posedge  HSx2)begin
225   if (TD_VS==0)
226         Vde_counter=0;
227           else
228         Vde_counter=Vde_counter+1;
229   end
230
231   wire hde=((Hde_counter > 51) && (Hde_counter < 691)) ? 1:0;//720
232   wire vde=((Vde_counter > 31) && (Vde_counter < 511)) ? 1:0;//480
233   wire blank_h  = h_tr   & vde;
234   wire blank  =   h_tr_h & vde;
235
236   wire  [7:0]Y  = (blank_h)?Ymm   :8'h10;
```

```
237   wire  [7:0]Cr = (blank_h)?Crmm  :8'h80;
238   wire  [7:0]Cb = (blank_h)?Cbmm  :8'h80;
239
240   dul_port_c1024 YYYR(
241       .iDATA(YY[7:0]),
242       .iHSYNC(TD_HS),
243   //  .iHSYNCx2(HSx2),
244       .iHSYNCx2(blank),
245       .Y_CLOCK(Ypix_clock),
246       .Y_CLOCKx2(CLOCK),
247       .oDATA(Yw[7:0]),
248       .field(Field),
249       .VS(TD_VS)
250   );
251
252
253   dul_port_c1024 CBB(
254       .iDATA(CCb[7:0]),
255       .iHSYNC(TD_HS),
256   //  .iHSYNCx2(HSx2),
257       .iHSYNCx2(blank),
258       .Y_CLOCK(Ypix_clock),
259       .Y_CLOCKx2(CLOCK),
260       .oDATA(Cwb[7:0]),
261       .field(Field),
262       .VS(TD_VS)
263   );
264
265   dul_port_c1024 CRR(
266       .iDATA(CCr[7:0]),
267       .iHSYNC(TD_HS),
268   //  .iHSYNCx2(HSx2),
269       .iHSYNCx2(blank),
270       .Y_CLOCK(Ypix_clock),
271       .Y_CLOCKx2(CLOCK),
272       .oDATA(Cwr[7:0]),
273       .field(Field),
274       .VS(TD_VS)
275   );
276
277   endmodule
278
```

```
1   //
    ---------------------------------------------------------------------
2   // Copyright (c) 2005 by Terasic Technologies Inc.
3   //
    ---------------------------------------------------------------------
4   //
5   // Permission:
6   //
7   //   Terasic grants permission to use and modify this code for use
8   //   in synthesis for all Terasic Development Boards and Altera
    Development
9   //   Kits made by Terasic.  Other use of this code, including the
    selling
10  //   ,duplication, or modification of any portion is strictly
    prohibited.
11  //
12  // Disclaimer:
13  //
14  //   This VHDL/Verilog or C/C++ source code is intended as a design
    reference
15  //   which illustrates how these types of functions can be
    implemented.
16  //   It is the user's responsibility to verify their design for
17  //   consistency and functionality through the use of formal
18  //   verification methods.  Terasic provides no warranty regarding
    the use
19  //   or functionality of this code.
20  //
21  //
    ---------------------------------------------------------------------
22  //
23  //                     Terasic Technologies Inc
24  //                     356 Fu-Shin E. Rd Sec. 1. JhuBei City,
25  //                     HsinChu County, Taiwan
26  //                     302
27  //
28  //                     web: http://www.terasic.com/
29  //                     email: support@terasic.com
30  //
31  //
    ---------------------------------------------------------------------
32  //
33  // Major Functions: YCbCr to RGB Color Doamin Converter.
34  //                  ( 10 Bits Resolution )
35  //
36  //
    ---------------------------------------------------------------------
37  //
38  // Revision History :
39  //
    ---------------------------------------------------------------------
40  //   Ver  :| Author            :| Mod. Date :| Changes Made:
41  //   V1.0 :| Johnny Chen       :| 05/09/05  :|      Initial Revision
```

```
42  //
    ---------------------------------------------------------------------
43
44  module YCbCr2RGB (  Red,Green,Blue,oDVAL,
45                      iY,iCb,iCr,iDVAL,
46                      iRESET,iCLK);
47  //  Input
48  input [7:0] iY,iCb,iCr;
49  input iDVAL,iRESET,iCLK;
50  //  Output
51  output [9:0] Red,Green,Blue;
52  output reg  oDVAL;
53  //  Internal Registers/Wires
54  reg [9:0] oRed,oGreen,oBlue;
55  reg [3:0] oDVAL_d;
56  reg [11:0] X_OUT,Y_OUT,Z_OUT;
57  wire [27:0] X,Y,Z;
58
59  assign  Red  =  oRed;
60  assign  Green=  oGreen;
61  assign  Blue =  oBlue;
62
63  always@(posedge iCLK)
64  begin
65      if(iRESET)
66      begin
67          oDVAL<=0;
68          oDVAL_d<=0;
69          oRed<=0;
70          oGreen<=0;
71          oBlue<=0;
72      end
73      else
74      begin
75          // Red
76          if(X_OUT[11])
77          oRed<=0;
78          else if(X_OUT[10:0]>1023)
79          oRed<=1023;
80          else
81          oRed<=X_OUT[9:0];
82          // Green
83          if(Y_OUT[11])
84          oGreen<=0;
85          else if(Y_OUT[10:0]>1023)
86          oGreen<=1023;
87          else
88          oGreen<=Y_OUT[9:0];
89          // Blue
90          if(Z_OUT[11])
91          oBlue<=0;
92          else if(Z_OUT[10:0]>1023)
93          oBlue<=1023;
```

```
 94          else
 95          oBlue<=Z_OUT[9:0];
 96          // Control
 97          {oDVAL,oDVAL_d}<={oDVAL_d,iDVAL};
 98      end
 99  end
100
101  always@(posedge iCLK)
102  begin
103      if(iRESET)
104      begin
105          X_OUT<=0;
106          Y_OUT<=0;
107          Z_OUT<=0;
108      end
109      else
110      begin
111          X_OUT<=( ( X - 105555 ) >>7 ) + 1;
112          Y_OUT<=( ( Y + 64218  ) >>7 ) + 1;
113          Z_OUT<=( ( Z - 131072 ) >>7 ) + 1;
114      end
115  end
116
117  //  Y       551,        0,          756
118  MAC_3 u0(   iY,         iCb,        iCr,
119              18'h00227,  18'h00000,  18'h002F4,
120              X,      iRESET,     iCLK);
121  //  Cb      551,        -186,       -385
122  MAC_3 u1(   iY,         iCb,        iCr,
123              18'h00227,  18'h3FF46,  18'h3FE7F,
124              Y,      iRESET,     iCLK);
125  //  Cr      551,        955,        0
126  MAC_3 u2(   iY,         iCb,        iCr,
127              18'h00227,  18'h003BB,  18'h00000,
128              Z,      iRESET,     iCLK);
129
130  endmodule
```

```
1   ----------------------------------------------------------------------
    ----------
2   --
3   -- Simple VGA raster display
4   --
5   -- Stephen A. Edwards
6   -- sedwards@cs.columbia.edu
7   --
8   ----------------------------------------------------------------------
    ----------
9   library ieee;
10  use ieee.std_logic_1164.all;
11  use ieee.numeric_std.all;
12
13  entity vga_controller is
14
15    port (
16
17      reset : in std_logic;
18      clk   : in std_logic;                    -- Should be 25.125
    MHz
19      rgb : in std_logic_vector(29 downto 0);
20      buffer_read_addr : out unsigned(15 downto 0);
21      hs_in : in std_logic;
22      vs_in : in std_logic;
23      sync_in : in std_logic;
24      blank_in : in std_logic;
25      line_sel : out std_logic;
26
27      VGA_CLK,                          -- Clock
28      VGA_HS,                           -- H_SYNC
29      VGA_VS,                           -- V_SYNC
30      VGA_BLANK,                        -- BLANK
31      VGA_SYNC : out std_logic;         -- SYNC
32      VGA_R,                            -- Red[9:0]
33      VGA_G,                            -- Green[9:0]
34      VGA_B : out unsigned(9 downto 0) -- Blue[9:0]
35      );
36
37  end vga_controller;
38
39  architecture rtl of vga_controller is
40
41    -- Video parameters
42
43    constant HTOTAL       : integer := 800;
44    constant HSYNC        : integer := 96;
45    constant HBACK_PORCH  : integer := 48;
46    constant HACTIVE      : integer := 640;
47    constant HFRONT_PORCH : integer := 16;
48
49    constant VTOTAL       : integer := 525;
50    constant VSYNC        : integer := 2;
```

```
51      constant VBACK_PORCH  : integer := 33;
52      constant VACTIVE      : integer := 480;
53      constant VFRONT_PORCH : integer := 10;
54
55      -- Signals for the video controller
56      signal Hcount : unsigned(15 downto 0);  -- Horizontal position
      (0-800)
57      signal Vcount : unsigned(15 downto 0);  -- Vertical position
      (0-524)
58      signal EndOfLine, EndOfField : std_logic;
59      signal buff_read_addr : unsigned(15 downto 0) :=
      "0000000000000000";
60      signal line_s : std_logic := '0';
61
62      signal vga_hblank, vga_hsync,
63        vga_vblank, vga_vsync : std_logic;  -- Sync. signals
64
65      signal pre_hs : std_logic;
66      signal hs_2 : std_logic_vector(1 downto 0);
67
68      signal pre_vs : std_logic;
69      signal vs_2 : std_logic_vector(1 downto 0);
70
71      signal mACT_HS, mACT_VS : std_logic;
72      signal hc, vc : unsigned(15 downto 0) := x"0000";
73
74      signal L_COUNTER, RL_COUNTER : unsigned(10 downto 0);
75      signal delay : unsigned(7 downto 0);
76      signal sync_en : std_logic;
77      signal sync_reset : std_logic;
78
79    begin
80
81      LSEL : process(clk)
82      begin
83        if Vcount(0) = '1' then
84            line_s <= not line_s;
85        end if;
86      end process LSEL;
87
88      -- Horizontal and vertical counters
89
90      HCounter : process (clk)
91      begin
92        if rising_edge(clk) then
93          if reset = '1' then
94            Hcount <= (others => '0');
95            buff_read_addr <= (others => '0');
96          elsif EndOfLine = '1' then
97            Hcount <= (others => '0');
98            buff_read_addr <= (others => '0');
99          else
100           buff_read_addr <= buff_read_addr + 1;
```

```
101           Hcount <= Hcount + 1;
102         end if;
103       end if;
104     end process HCounter;
105
106   --  Haddr : process (clk)
107   --  begin
108   --  if rising_edge(clk) then
109   --    if reset = '1' or EndOfLine <= '1' then
110   --      buff_read_addr <= (others => '0');
111   --    else
112   --      buff_read_addr <= buff_read_addr + 1;
113   --    end if;
114   --  end if;
115   --  end process Haddr;
116
117     buffer_read_addr <= buff_read_addr;
118   --  EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';
119     EndOfLine <= '1' when Hcount = 852 else '0';
120
121     VCounter: process (clk)
122     begin
123       if rising_edge(clk) then
124         if reset = '1' then
125           Vcount <= (others => '0');
126         elsif EndOfLine = '1' then
127           if EndOfField = '1' then
128             Vcount <= (others => '0');
129           else
130             Vcount <= Vcount + 1;
131           end if;
132         end if;
133       end if;
134     end process VCounter;
135
136   --  EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';
137     EndOfField <= '1' when Vcount = 525 else '0';
138     -- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK
139
140     HSyncGen : process (clk, sync_en)
141     begin
142       if clk'event and clk = '1' then
143         if sync_en = '0' then
144           Pre_HS <= '0';
145           mACT_HS <= '0';
146           hc <= x"0000";
147           vga_hsync <= '0';
148         else
149           pre_hs <= hs_in;
150           hs_2 <= pre_hs & hs_in;
151           if reset = '1' then
152             vga_hsync <= '0';
153           elsif hs_2 = "10" then
```

```
154             mACT_HS <= '1';
155             if mACT_HS = '1' then
156               if hc < 852 then
157                 hc <= hc + 1;
158               else
159                 hc <= x"0000";
160                 mACT_HS <= '0';
161               end if;
162               if hc < HSYNC then
163                 vga_hsync <= '0';
164               else
165                 vga_hsync <= '1';
166               end if;
167             end if;
168           end if;
169         end if;
170       end if;
171     end process HSyncGen;
172
173     HBlankGen : process (clk)
174     begin
175       if rising_edge(clk) then
176         if reset = '1' then
177           vga_hblank <= '1';
178         elsif Hcount = HSYNC + HBACK_PORCH then
179           vga_hblank <= '0';
180         elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
181           vga_hblank <= '1';
182         end if;
183       end if;
184     end process HBlankGen;
185
186     VSyncGen : process (clk)
187     begin
188       if rising_edge(clk) then
189         if reset = '1' then
190           vga_vsync <= '1';
191         elsif EndOfLine ='1' then
192           if EndOfField = '1' then
193             vga_vsync <= '0';
194           elsif Vcount = VSYNC then
195             vga_vsync <= '1';
196           end if;
197         end if;
198       end if;
199     end process VSyncGen;
200
201   --  VSyncGen : process (clk, sync_en)
202   --  begin
203   --    if clk'event and clk = '1' then
204   --      if sync_en = '0' then
205   --      Pre_VS <= '1';
206   --      mACT_VS <= '0';
```

```
207   --      vc <= x"0000";
208   --      vga_vsync <= '0';
209   --    else
210   --        pre_vs <= vs_in;
211   --        vs_2 <= pre_vs & vs_in;
212   --      if reset = '1' then
213   --        vga_vsync <= '0';
214   --      elsif vs_2 = "01" then
215   --        mACT_VS <= '1';
216   --        if hc = "0000000000000001" and mACT_VS = '1' then
217   --          if vc < 524 then
218   --            vc <= vc + 1;
219   --          else
220   --            vc <= x"0000";
221   --          end if;
222   --          if vc < VSYNC then
223   --            vga_vsync <= '0';
224   --          else
225   --            vga_vsync <= '1';
226   --          end if;
227   --        end if;
228   --      end if;
229   --    end if;
230   --    end if;
231   --  end process VSyncGen;
232
233     LCOUNT : process(hs_in)
234     begin
235       if hs_in = '1' then
236         L_COUNTER <= "00000000000";
237       else
238         L_COUNTER <= L_COUNTER + 1;
239       end if;
240     end process LCOUNT;
241
242     RLCOUNT : process(vs_in)
243     begin
244       if vs_in = '1' then
245         RL_COUNTER <= L_COUNTER;
246       end if;
247     end process RLCOUNT;
248
249     DELAYGen : process(sync_reset, vs_in)
250     begin
251       if sync_reset = '0' then
252         delay <= x"00";
253       elsif vs_in = '1' and delay < 250 then
254         delay <= delay + 1;
255       end if;
256     end process DELAYGen;
257
258     SYNCENGen : process(sync_reset, vs_in)
259     begin
```

```
260       if sync_reset = '0' then
261         sync_en <= '0';
262       elsif vs_in = '0' and delay < 100 then
263         sync_en <= '0';
264       else
265         sync_en <= '1';
266       end if;
267     end process SYNCENGen;
268
269     SYNCRESETGen: process(clk)
270     begin
271       if RL_COUNTER = 9 then
272         sync_reset <= '1';
273       else
274         sync_reset <= '0';
275       end if;
276     end process SYNCRESETGen;
277
278
279     VBlankGen : process (clk)
280     begin
281       if rising_edge(clk) then
282         if reset = '1' then
283           vga_vblank <= '1';
284         elsif EndOfLine = '1' then
285           if Vcount = VSYNC + VBACK_PORCH - 1 then
286             vga_vblank <= '0';
287           elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
288             vga_vblank <= '1';
289           end if;
290         end if;
291       end if;
292     end process VBlankGen;
293
294     -- Registered video signals going to the video DAC
295
296     VideoOut: process (clk, reset)
297     begin
298       if reset = '1' then
299         VGA_R <= "0000000000";
300         VGA_G <= "0000000000";
301         VGA_B <= "0000000000";
302       elsif clk'event and clk = '1' then
303         if Hcount >= 0 then
304               VGA_R <= unsigned(rgb(29 downto 20));
305               VGA_G <= unsigned(rgb(19 downto 10));
306               VGA_B <= unsigned(rgb(9 downto 0));
307         elsif vga_hblank = '0' and vga_vblank ='0' then
308           VGA_R <= "0000000000";
309           VGA_G <= "1111111111";
310           VGA_B <= "0000000000";
311         else
312           VGA_R <= "1111111111";
```

```
313           VGA_G <= "0000000000";
314           VGA_B <= "0000000000";
315         end if;
316       end if;
317     end process VideoOut;
318 
319 
320     line_sel <= line_s;
321     VGA_CLK <= clk;
322     VGA_HS <= hs_in;
323     VGA_VS <= vs_in;
324     VGA_SYNC <= '0';
325     VGA_BLANK <= not (vga_vsync or vga_hsync);
326 
327   end rtl;
328 
```

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

entity line_buffer is
  port (
    reset     : in std_logic;
    wclk      : in std_logic;
    rclk      : in std_logic;
    write_enable  : in std_logic;
    rgb : in std_logic_vector(29 downto 0);
    write_address : in unsigned(15 downto 0);
    read_data     : out std_logic_vector(29 downto 0);
    read_address  : in unsigned(15 downto 0);
    -- page = 0: write to line1, read from line2
    -- page = 1: write to line2, read from line1
    page          : in std_logic
  );
end line_buffer;

architecture RTL of line_buffer is

  type ram_type is array(0 to 800) of std_logic_vector(29 downto 0);

  signal line1: ram_type;
  signal line2: ram_type;

begin

  WriteLine : process (wclk)
  begin
    if rising_edge(wclk) then
      if write_enable = '1' then
        if page = '0' then
          line1(to_integer(write_address)) <= rgb;
        else
          line2(to_integer(write_address)) <= rgb;
        end if;
      end if;
    end if;
  end process;

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

```

```
54    end RTL;
55
```

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.numeric_std.all;
 4
 5  entity xy_detector is
 6
 7  port (
 8      clk50:  in std_logic;
 9
10      -- Avalon Bus Signals
11      signal address : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
12      signal chipselect : IN STD_LOGIC;
13      signal read : IN STD_LOGIC;
14      signal reset_n : IN STD_LOGIC;
15      signal write : IN STD_LOGIC;
16      signal writedata : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
17      signal readdata : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
18
19      -- Input from the line buffer
20      read_data : IN std_logic_vector(29 downto 0);
21      v_pos   :   in std_logic_vector(15 downto 0); --unsigned
22      h_pos   :   in std_logic_vector(15 downto 0) --unsigned
23  );
24  end xy_detector;
25
26  architecture RTL of xy_detector is
27
28              --first we will determine whether or not a given pixel
    is a laser pixel
29              signal Red : unsigned(9 downto 0) := "0000000000";
30              signal Green : unsigned(9 downto 0) := "0000000000";
31              signal Blue : unsigned(9 downto 0) := "0000000000";
32              constant Rmin : integer := 236;
33              constant Rmax : integer := 255;
34              constant Gmin : integer := 221;
35              constant Gmax : integer := 255;
36              constant Bmin : integer := 220;
37              constant Bmax : integer := 255;
38              signal pix_min : integer := 2;--the minimum number of
    across a dot
39              signal pix_max : integer := 30;--the maximum number of
    across a dot
40              signal laser_pixel : boolean := false; --decides if the
    pixel is a laser pixel
41              signal dot1 : unsigned(9 downto 0) := "0000000000";
42              signal dot2 : unsigned(9 downto 0) := "0000000000";
43              signal dot3 : unsigned(9 downto 0) := "0000000000";
44              signal flag1 : boolean := false;
45              signal flag2 : boolean := false;
46              signal flag3 : boolean := false;
47
48              signal x1_coord : std_logic_vector(15 downto 0); --the
    x values
```

```
49              signal y1_coord : std_logic_vector(15 downto 0);
50              signal x2_coord : std_logic_vector(15 downto 0); --the
    x values
51              signal y2_coord : std_logic_vector(15 downto 0);
52              signal x3_coord : std_logic_vector(15 downto 0); --the
    x values
53              signal y3_coord : std_logic_vector(15 downto 0);
54
55          begin
56              --unsigned can compare to integer
57              SetLaser    :   process(clk50)
58              begin
59                  if rising_edge(clk50) then
60                      Red <= unsigned(read_data(29 downto 20));
61                      Green <= unsigned(read_data(19 downto 10));
62                      Blue <= unsigned(read_data(9 downto 0));
63                          if (Rmin <= Red and Red <= Rmax) then
64                              if (Gmin <= Green and Green <= Gmax) then
65                                  if (Bmin <= Blue and Blue <= Bmax)
    then
66                                      laser_pixel <= true;
67
68                                  end if;
69                              end if;
70                          end if;
71                  --will only set flag1 once dot1 has been confirmed,
    otherwise will reset
72                  --same is true of dot2 and flag2
73
74                      if(not laser_pixel and dot1 = 0) then
75                      --do nothing
76                      elsif(laser_pixel and not flag1) then --find
    the first laser pixel
77                          dot1 <= dot1 + 1;
78                      elsif(not laser_pixel and dot1 /= 1 and dot1 <
    pix_min) then-- if found a null dot before minimum dot size met
79                          dot1 <= "0000000000";
80                      elsif(not laser_pixel and dot1 > pix_max) then
    -- first null dot after maximum dot size
81                          dot1 <= "0000000000";
82                      elsif(not laser_pixel and dot1 <= pix_max and
    dot1 >= pix_min) then -- first dot complete
83                          flag1 <= true;
84                          x1_coord <= std_logic_vector(unsigned(v_pos
    )-(dot1/2)); --the x value
85                          y1_coord <= std_logic_vector(unsigned(h_pos
    )+(dot1/2)); --the y value
86
87                      elsif(flag1 and not(laser_pixel) and dot2 = 0)
    then
88                      --do nothing
89                      elsif(flag1 and laser_pixel and not flag2) then
90                          dot2 <= dot2 + 1;
```

```
91                      elsif(flag1 and not laser_pixel and dot2 <
    pix_min) then
92                          dot2 <= "0000000000";
93                      elsif(flag1 and not laser_pixel and dot2 >
    pix_max) then
94                          dot2 <= "0000000000";
95                      elsif(flag1 and not laser_pixel and dot2 <=
    pix_max and dot2 >= pix_min) then
96                          flag2 <= true;
97                          x2_coord <= std_logic_vector(unsigned(v_pos
    )-(dot2/2)); --the x value
98                          y2_coord <= std_logic_vector(unsigned(h_pos
    )+(dot2/2)); --the y value
99                      end if;
100                     if(flag2 and not laser_pixel and dot3 = 0) then
101                     --do nothing
102                     end if;
103                     if(flag2 and laser_pixel and not flag3) then
104                         dot3 <= dot3 + 1;
105                     end if;
106                     if(flag2 and not laser_pixel and dot3 < pix_min)
     then
107                         dot3 <= "0000000000";
108                     end if;
109                     if(flag2 and not laser_pixel and dot3 > pix_max)
     then
110                         dot3 <= "0000000000";
111                     end if;
112                     if(flag2 and not laser_pixel and dot3 <= pix_max
     and dot3 >= pix_min) then
113                         flag3 <= true;
114                         x3_coord <= std_logic_vector(unsigned(v_pos
    )-(dot3/2)); --the x value
115                         y3_coord <= std_logic_vector(unsigned(h_pos
    )+(dot3/2)); --the y value
116                     end if;
117                     if(unsigned(v_pos) = 0 and unsigned(h_pos) = 0)
    then
118                         dot1 <= "0000000000";
119                         dot2 <= "0000000000";
120                         dot3 <= "0000000000";
121                         flag1 <= false;
122                         flag2 <= false;
123                         flag3 <= false;
124                     end if;
125                     if chipselect = '1' then
126                         if address(2 downto 0) = "001" then
127                             readdata <= h_pos;
128                         elsif address(2 downto 0) = "010" then
129                             readdata <= v_pos;
130                         elsif address(2 downto 0) = "011" then
131                             readdata <= x2_coord;
132                         elsif address(2 downto 0) = "100" then
```

```
133                             readdata <= y2_coord;
134                         elsif address(2 downto 0) = "101" then
135                             readdata <= x3_coord;
136                         elsif address(2 downto 0) = "110" then
137                             readdata <= y3_coord;
138                         end if;
139                     end if;
140                 end if;
141
142             end process SetLaser;
143
144 end RTL;
```

```c
//find the length to two points of a known, fixed distance apart
//calculate the pixels as a portion of the horizontal distance
//inputs are ordered pairs
//angular offset in radians
//this pseudo code outlines the coincidental case
//assuming the camera line of sight is perpendicular to the wall
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

double findrange(int *point1, int *point2, double angoffset){

const double pi = 3.1415926535;

//the following two quantities are measured
const double CCD_FOV = 33.397*pi/180; //33.397 degrees converted to radians
const float LASER_DIST_CM = 29.8;

const double CCD_WIDTH = 640;
const double CCD_HEIGHT = 480;

int del_x = (abs(point1[0]-abs(point2[0])))*cos(angoffset);
double y_avg = (abs(point1[1]) - abs(point2[1]))/2; //average these points, approximating
the dot finder as horizontal
double horiz_portion = (double)(del_x)/CCD_WIDTH;
double horiz_angle_subtended = horiz_portion*CCD_FOV; //linear angle approximation

double dist_away = (LASER_DIST_CM/2)/tan(horiz_angle_subtended/2);
double field_width = LASER_DIST_CM/horiz_portion;
double field_height = CCD_HEIGHT/CCD_WIDTH*field_width;

double xoffset, yoffset;

if((point1[0] > 0 && point2[0] >0)||(point1[0] < point2[0])){ //right-half
    xoffset = (del_x/2-320)*LASER_DIST_CM;
}else if((point1[0] < 0 && point2[0] < 0)||point1[0] > point2[0]){ //left half
    xoffset = (-del_x+320)*LASER_DIST_CM;
}

if(y_avg>0) yoffset = (y_avg-240)/480*field_height;
else yoffset = -(y_avg-240)/480*field_height;

printf("The field dimensions are %f cm x %f cm\n",field_width ,field_height);
printf("The offset from the origin is %f cm,%f cm\n", xoffset, yoffset);

return dist_away;
}

//testbench
int main(int argc, char *argv []){
    if(argc != 6){
        printf("Specify x1 y1 x2 y2\n");
        return 1;
    }

    int x1,y1,x2,y2;
    x1 = atoi(argv[1]);
    y1 = atoi(argv[2]);
```

```
    x2 = atoi(argv[3]);
    y2 = atoi(argv[4]);
    double gamma =  atof(argv[5]);
    int point1 [] = {x1, y1};
    int point2 [] = {x2, y2};

    double test_output = findrange(point1,point2, gamma);
    printf("The calculated distance from the launcher is %f cm\n", test_output);
    return 0;
}
```

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.numeric_std.all;
 4
 5  entity usb is
 6
 7  port (
 8      CLOCK_27 : in std_logic;
    -- 27 MHz
 9      signal CLOCK_50 : in std_logic; --50 MHz
10      signal LEDR : out std_logic_vector(17 downto 0); --LEDs
11
12
13      SRAM_DQ : inout std_logic_vector(15 downto 0);
14      SRAM_ADDR : out std_logic_vector(17 downto 0);
15      SRAM_UB_N, --Highbyte Data Mask
16      SRAM_LB_N, --Lowbyte Data Mask
17      SRAM_WE_N, --Write Enable
18      SRAM_CE_N, --Chip Enable
19      SRAM_OE_N : out std_logic; --Output Enable
20
21      KEY : in std_logic_vector(3 downto 0);          -- Push buttons
22
23      OTG_ADDR: OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
24          OTG_CS_N: OUT STD_LOGIC;
25          OTG_DACK0_N: OUT STD_LOGIC;
26          OTG_DACK1_N: OUT STD_LOGIC;
27          OTG_DATA: INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
28          OTG_FSPEED: OUT STD_LOGIC;
29          OTG_INT0: IN STD_LOGIC;
30          OTG_INT1: IN STD_LOGIC;
31          OTG_LSPEED: OUT STD_LOGIC;
32          OTG_RD_N: OUT STD_LOGIC;
33          OTG_RST_N: OUT STD_LOGIC;
34          OTG_WR_N: OUT STD_LOGIC;
35
36
37      -- I2C bus
38
39      I2C_SDAT : inout std_logic; -- I2C Data
40      I2C_SCLK : out std_logic;   -- I2C Clock
41
42      -- Video Decoder
43
44      TD_DATA : in std_logic_vector(7 downto 0);  -- Data bus 8 bits
45      TD_HS,                                      -- H_SYNC
46      TD_VS : in std_logic;                       -- V_SYNC
47      TD_RESET : out std_logic;                   -- Reset
48
49      -- VGA output
50
51      VGA_CLK,                                            -- Clock
52      VGA_HS,                                             -- H_SYNC
```

```
53        VGA_VS,                                                 -- V_SYNC
54        VGA_BLANK,                                              -- BLANK
55        VGA_SYNC : out std_logic;                               -- SYNC
56        VGA_R,                                                  -- Red[9:0]
57        VGA_G,                                                  -- Green[9:0]
58        VGA_B : out unsigned(9 downto 0)                        -- Blue[9:0]
59    );
60    end usb;
61    architecture rtl of usb is
62
63
64        component I2C_AV_Config is
65            port (
66                iCLK : in std_logic;
67                iRST_N : in std_logic;
68                I2C_SCLK : out std_logic;
69                I2C_SDAT : inout std_logic
70            );
71        end component;
72
73        signal r_sig, g_sig, b_sig : unsigned(9 downto 0);
74        signal counter : unsigned(15 downto 0);
75        signal reset_n : std_logic := '0';
76        signal clk25: std_logic := '0';
77        signal hc_sig: unsigned(15 downto 0);
78        signal vc_sig: unsigned(15 downto 0);
79
80    begin
81        --LEDR(17) <= OTG_ADDR(1);
82        --LEDR(16) <= OTG_ADDR(0);
83
84      TD_RESET <= KEY(0);
85
86      V1: entity work.ntsc2vga port map (
87        reset => '0',
88        clk25 => clk25,
89        clk27 => CLOCK_27,
90        clk50 => CLOCK_50,
91        td_data => TD_DATA,
92        td_hs => TD_HS,
93        td_vs => TD_VS,
94        VGA_CLK => VGA_CLK,
95        VGA_HS => VGA_HS,
96        VGA_VS => VGA_VS,
97        VGA_BLANK => VGA_BLANK,
98        VGA_SYNC => VGA_SYNC,
99        VGA_R => r_sig,
100       VGA_G => g_sig,
101       VGA_B => b_sig,
102       hcount => hc_sig,
103       vcount => vc_sig
104     );
105
```

```
106     i2c: I2C_AV_Config port map(
107       iCLK => CLOCK_50,
108       iRST_N => KEY(2),
109       I2C_SCLK => I2C_SCLK,
110       I2C_SDAT => I2C_SDAT
111       );
112
113   process (CLOCK_50)
114   begin
115       if rising_edge(CLOCK_50) then
116         if counter = x"ffff" then
117           reset_n <= '1';
118         else
119           reset_n <= '0';
120           counter <= counter + 1;
121         end if;
122       end if;
123   end process;
124
125   nios : entity work.nios_system port map (
126               clk => CLOCK_50,
127               reset_n => reset_n,
128               --leds_from_the_leds => LEDR(15 downto 0),
129               SRAM_ADDR_from_the_sram => SRAM_ADDR,
130               SRAM_CE_N_from_the_sram => SRAM_CE_N,
131               SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
132               SRAM_LB_N_from_the_sram => SRAM_LB_N,
133               SRAM_OE_N_from_the_sram => SRAM_OE_N,
134               SRAM_UB_N_from_the_sram => SRAM_UB_N,
135               SRAM_WE_N_from_the_sram => SRAM_WE_N,
136
137                       -- the_isp1362_ctrl_inst
138                           OTG_ADDR_from_the_isp1362 => OTG_ADDR,

139                           OTG_CS_N_from_the_isp1362=> OTG_CS_N,
140                           OTG_DACK0_N_from_the_isp1362 => OTG_DACK0_N,
141                           OTG_DACK1_N_from_the_isp1362 => OTG_DACK1_N,
142                           OTG_DATA_to_and_from_the_isp1362 => OTG_DATA,
143                           OTG_FSPEED_from_the_isp1362 => OTG_FSPEED,
144                           OTG_INT0_to_the_isp1362 => OTG_INT0,
145                           OTG_INT1_to_the_isp1362 => OTG_INT1,
146                           OTG_LSPEED_from_the_isp1362 => OTG_LSPEED,
147                           OTG_RD_N_from_the_isp1362 => OTG_RD_N,
148                           OTG_RST_N_from_the_isp1362 => OTG_RST_N,
149                           OTG_WR_N_from_the_isp1362 => OTG_WR_N,
150
151                           h_pos_to_the_xy_pos => std_logic_vector(
      hc_sig),
152                           read_data_to_the_xy_pos => std_logic_vector(
      r_sig & g_sig & b_sig),
153                           v_pos_to_the_xy_pos => std_logic_vector(
      vc_sig)
154
```

```
155   );
156
157   VGA_R <= r_sig;
158   VGA_G <= g_sig;
159   VGA_B <= b_sig;
160
161   end rtl;
162
```

```
 1  //ISP1362 controler
 2  module isp1362_ctrl (
 3      //Avalon Interface
 4      clk,
 5      address,
 6      readdata,
 7      writedata,
 8      writedata_avalon_slave_1,
 9      chipselect_n,
10      read_n,
11      write_n,
12      reset_n,
13      write_n_avalon_slave_1,
14      irq_n,
15      irq_n_avalon_slave_1,
16
17      //Phillips USB controller
18      OTG_ADDR,
19      OTG_DATA,
20      OTG_CS_N,
21      OTG_RD_N,
22      OTG_WR_N,
23      OTG_RST_N,
24      OTG_INT0,
25      OTG_INT1,
26      OTG_FSPEED,
27      OTG_LSPEED,
28      OTG_DACK0_N,
29      OTG_DACK1_N
30  );
31
32  //Avalon Interface
33  input clk, chipselect_n, read_n, write_n, reset_n,
    write_n_avalon_slave_1;
34  input [1:0] address;
35  input [15:0] writedata;
36  input [7:0] writedata_avalon_slave_1;
37  output [15:0] readdata;
38  output irq_n, irq_n_avalon_slave_1;
39
40  //Phillips USB controller
41  output [1:0] OTG_ADDR;
42  inout [15:0] OTG_DATA;
43  output OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N;
44  input OTG_INT0, OTG_INT1;
45  output OTG_FSPEED, OTG_LSPEED, OTG_DACK0_N, OTG_DACK1_N;
46
47  //Registers
48  reg [15:0] data, readdata;
49  reg [1:0] OTG_ADDR;
50  reg OTG_CS_N, OTG_RD_N, OTG_WR_N;
51  reg irq_n, irq_n_avalon_slave_1;
52
```

```
53   //Assignments
54   assign OTG_RST_N = reset_n;
55   assign OTG_DATA = OTG_WR_N ? 16'hzzzz : data;
56   assign OTG_DACK0_N = 1'b1, OTG_DACK1_N = 1'b1;
57   assign OTG_FSPEED = 0, OTG_LSPEED = 0;
58
59
60       always @ (posedge clk or negedge reset_n)begin
61           //Reset condition
62           if (reset_n==0)begin
63               data        <=  0;
64               readdata    <=  0;
65               OTG_ADDR    <=  0;
66               OTG_CS_N    <=  1;
67               OTG_RD_N    <=  1;
68               OTG_WR_N    <=  1;
69               irq_n       <=  1;
70               irq_n_avalon_slave_1    <=  1;
71           end
72           //normal operation
73           else begin
74               data        <=  writedata;
75               readdata    <=  OTG_DATA;
76               OTG_ADDR    <=  address;
77               OTG_CS_N    <=  chipselect_n;
78               OTG_RD_N    <=  read_n;
79               OTG_WR_N    <=  write_n;
80               irq_n                   <=  OTG_INT0;
81               irq_n_avalon_slave_1    <=  OTG_INT1;
82           end
83       end
84   endmodule
85
```

```
#ifndef ISP1362_H_
#define ISP1362_H_

#endif /*ISP1362_H_*/
#define device_name "ISP1362"
#define ram_size 4096

#define SETUP 0
#define OUT   1
#define IN    2
#define TRUE  1
#define FALSE 0

#define HcControl           0x01
#define HcCmdStatus         0x02
#define HcIntStatus         0x03
#define HcIntEnable         0x04
#define HcIntDisable        0x05
#define HcFmItv             0x0D
#define HcFmNo              0x0F
#define HcRhA               0x12
#define HcRhB               0x13
#define HcRhStatus          0x14
#define HcRhP1              0x15
#define HcRhP2              0x16
#define HcIntDone           0x17
#define HcIntSkip           0x18
#define HcIntLast           0x19
#define HcIntActive         0x1A
#define HcATLDone           0x1B
#define HcATLSkip           0x1C
#define HcATLLast           0x1D
#define HcATLActive         0x1E
#define HcHWCfg             0x20
#define HcDMACfg            0x21
#define HcTransferCnt       0x22
#define HcUpInt             0x24
#define HcUpIntEnable       0x25
#define HcChipID            0x27
#define HcScratch           0x28
#define HcBufStatus         0x2C
#define HcISTLBufferSize    0x30
#define HcDirAddrLen        0x32
#define HcINTLBufferSize    0x33
#define HcATLBufferSize     0x34
#define HcISTL0BufferPort   0x40
#define HcISTL1BufferPort   0x42
#define HcINTLBufferPort    0x43
#define HcATLBufferPort     0x44
#define HcDirAddrData       0x45
#define HcISTLTogRate       0x47
#define HcATLPTDDoneThrsCnt 0x51
#define HcATLPRDDoneTimeOut 0x52

#define HcIntBlkSize        0x53
#define HcATLBlkSize        0x54
```

```
#define OTGControl          0x62
#define OTGStatus           0x67
#define OTGInterrupt        0x68
#define OTGInterruptEnable  0x69
#define OTGTimer            0x6A

#define HcReset             0xA9
```

```c
#include <stdio.h>
#include <system.h>
#include <io.h>
#include "isp1362.h"

#define IOWR_ISP_DATA(base, offset, data) \
IOWR_16DIRECT(base, (offset), data) // CHANGED FROM *2
#define IORD_ISP_DATA(base, offset) \
IORD_16DIRECT(base, (offset))
#define IOWR_ISP_DATA32(base,offset,data) \
IOWR_32DIRECT(base, offset,data)
#define IORD_ISP_DATA32(base,offset) \
IORD_32DIRECT(base, offset)

#define HC_DATA 0
//#define HC_TEST 1
#define HC_COM  4

#define DC_DATA     8
#define DC_COM  12

#define CLEAR_BUFF = 0x70

unsigned int port1speed = 0;
unsigned int port2speed = 0;
unsigned int port_speed = 0;
unsigned int retry = 60000; //50000

struct ptd_struct
{
 unsigned int c_code;
 unsigned int active_bit;
 unsigned int toggle;
 unsigned int actual_size;
 unsigned int endpoint;
 unsigned int last_ptd;
 unsigned int speed;
 unsigned int max_size;
 unsigned int pid;
 unsigned int total_size;
 unsigned int format;
 unsigned int func_addr;
 unsigned char fm;
}
ptd2send;


void OneuSDelay(void)
 {
  unsigned j;
  unsigned i;

  for (i=0;i<0xd0;i++)
   {
    j++;
   };
 }
```

```
void uSDelay(unsigned wDlyTime)
 {
  unsigned j;
  unsigned i;

  for (i=0;i<wDlyTime;i++)
   {
    OneuSDelay();
    j++;
   };
 }


void hc_write(unsigned int reg_index, unsigned int data){
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,reg_index|0x80); // reg index to
command port

        uSDelay(10);
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA,data);
        //unsigned int reg_value = IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM);
        //printf("writing %x to reg %x\n", data, reg_value);
}

unsigned int hc_read(unsigned int reg_index){
        unsigned int data_return; //, data_test;
        //data_test= IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);
        //printf("data_test: %x\n", data_test);
        //IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_TEST,reg_index);
        //uSDelay(500000);
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,reg_index);
        uSDelay(10);
        data_return= IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);
        //printf("data_return: %x\n", data_return);
        return(data_return);

}
void dc_write(unsigned int reg_index, unsigned int data){
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,DC_COM,reg_index|0x80); // reg index to
command port

        uSDelay(10);
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,DC_DATA,data);
        //unsigned int reg_value = IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM);
        //printf("writing %x to reg %x\n", data, reg_value);
}

unsigned int dc_read(unsigned int reg_index){
        unsigned int data_return; //, data_test;
        //data_test= IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);
        //printf("data_test: %x\n", data_test);
        //IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_TEST,reg_index);
        //uSDelay(500000);
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,DC_COM,reg_index);
        uSDelay(10);
        data_return= IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,DC_DATA);

        //printf("data_return: %x\n", data_return);
```

```
        return(data_return);

}


void poll(unsigned int poll_len)
{
 unsigned int temp=0;

 do
  {
   temp++;
  }
 while(temp<poll_len);
}


void hc_write32(unsigned int reg_index, unsigned long data){
        unsigned int data_low, data_high;
        data_low = data & 0x0000ffff;
        data_high = (data & 0xffff0000) >> 16;
        //printf("data_low:%4x\n", data_low);
        //printf("data_high:%4x\n", data_high);
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,reg_index|0x80); // reg index to
command port
        uSDelay(10);
        //IOWR_ISP_DATA32(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA,data);


        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA,data_low);
        uSDelay(10);
        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA,data_high); // data phase


}

unsigned long hc_read32(unsigned int reg_index)
{
        unsigned int data_low, data_high;
        unsigned long data_return;

        IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,reg_index);
        uSDelay(10);
        //data_return = IORD_ISP_DATA32(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);

        data_low= IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);
        uSDelay(10);
        data_high= IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);

        data_return = data_high;
        data_return = data_return << 16;
        data_return = data_return + data_low;


        return(data_return);
        //printf("data_return: %x\n", data_return);
}
```

```
void new_make_ptd(int *rptr,char token,char ep,int max,char tog,char addr,char port,
unsigned int total)
{
 ptd2send.c_code=0x0F;
 ptd2send.active_bit=1;
 ptd2send.toggle=tog;
 ptd2send.actual_size=0;
 ptd2send.endpoint=ep;
 ptd2send.last_ptd=0;

 ptd2send.speed=port_speed;
 if(port==1) {ptd2send.speed=port1speed;}
 if(port==2) {ptd2send.speed=port2speed;}

 ptd2send.max_size=max;
 ptd2send.total_size=total;
 ptd2send.pid= token;
 ptd2send.format=0;
 ptd2send.fm=0;
 ptd2send.func_addr=addr;

 *(rptr+0)= (ptd2send.c_code        &0x0000)<<12
           |(ptd2send.active_bit    &0x0001)<<11
           |(ptd2send.toggle        &0x0001)<<10
           |(ptd2send.actual_size   &0x03FF);

 *(rptr+1)=  (ptd2send.endpoint     &0x000F)<<12
           |(ptd2send.last_ptd      &0x0001)<<11
           |(ptd2send.speed         &0x0001)<<10
           |(ptd2send.max_size      &0x03FF);

 *(rptr+2)=  (0x0000                &0x000F)<<12
           |(ptd2send.pid           &0x0003)<<10
           |(ptd2send.total_size    &0x03FF);

 *(rptr+3)=  (ptd2send.fm            &0x00FF)<<8
           |(ptd2send.format        &0x0001)<<7
           |(ptd2send.func_addr     &0x007F);
}

unsigned int error_recorder(char mode,int ccode1, int ccode2)
{
 static int cnt=0;
 static int c1[2048];
 static int c2[2048];

 int data2return;

 if(mode=='W')
 {
  c1[cnt]=ccode1;
  c2[cnt]=ccode2;
  cnt++;

  if(cnt>2047) {cnt=2047;}
 }

 if(mode=='R')
```

```
 {
  data2return=c1[ccode1];
  data2return=data2return<<8;
  data2return|=c2[ccode2];
 }

 if(mode=='C')
 {
  data2return=cnt;
 }

 if(mode=='Z')
 {
  cnt=0;
 }

 return(data2return);
}


void write_atl(unsigned int *a_ptr, unsigned int data_size)
{
 int cnt;

 hc_write(HcTransferCnt,data_size*2);
 IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,HcATLBufferPort |0x80);

 uSDelay(10);   //waste some time

 cnt=0;
 do
  {
   IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA,*(a_ptr+cnt));
   cnt++;
  }
 while(cnt<(data_size));
}

void read_atl(unsigned int *a_ptr, unsigned int data_size)
{
 int cnt;

 hc_write(HcTransferCnt,data_size*2);
 IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,HcATLBufferPort );

 uSDelay(10);   //wait a little

 cnt=0;
 do
  {
   *(a_ptr+cnt)=IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);
   cnt++;
  }
 while(cnt<(data_size));
}

void make_ptd(int *rptr,char token,char ep,int max,char tog,char addr,char port)
{
```

```
 ptd2send.c_code=0x0F;
 ptd2send.active_bit=1;
 ptd2send.toggle=tog;
 ptd2send.actual_size=0;
 ptd2send.endpoint=ep;
 ptd2send.last_ptd=0;

 ptd2send.speed=port_speed;
 if(port==1) {ptd2send.speed=port1speed;}
 if(port==2) {ptd2send.speed=port2speed;}

 ptd2send.max_size=max;
 ptd2send.total_size=max;
 ptd2send.pid= token;
 ptd2send.format=0;
 ptd2send.fm=0;
 ptd2send.func_addr=addr;

 *(rptr+0)= (ptd2send.c_code        &0x0000)<<12
           |(ptd2send.active_bit    &0x0001)<<11
           |(ptd2send.toggle        &0x0001)<<10
           |(ptd2send.actual_size   &0x03FF);

 *(rptr+1)=  (ptd2send.endpoint     &0x000F)<<12
           |(ptd2send.last_ptd      &0x0001)<<11
           |(ptd2send.speed         &0x0001)<<10
           |(ptd2send.max_size      &0x03FF);

 *(rptr+2)=  (0x0000                &0x000F)<<12
           |(ptd2send.pid           &0x0003)<<10
           |(ptd2send.total_size    &0x03FF);

 *(rptr+3)=  (ptd2send.fm            &0x00FF)<<8
           |(ptd2send.format        &0x0001)<<7
           |(ptd2send.func_addr     &0x007F);
}

void array_app(int *cbuf,int *tbuf,int no_of_words)
{
 int cnt=0;

 do
  {
   *(cbuf + cnt) = *(tbuf + cnt);
   cnt++;
  }
 while(cnt<no_of_words);
}

//addr - from ptd.c
unsigned int addr_info(int addr,int mode,int dtype,int para)
{
 static unsigned int speed[8];
 static unsigned int maxpac[8];
 static unsigned int iman[8];
 static unsigned int ipro[8];

 unsigned int data2return;
```

```
 //Displaying data
 if(mode=='D')
 {
  printf("\nPort 1 : Speed[%2d] MPS[%4d] iMan[%2d]
iPro[%2d]",speed[1],maxpac[1],iman[1],ipro[1]);
  printf("\nPort 2 : Speed[%2d] MPS[%4d] iMan[%2d]
iPro[%2d]",speed[2],maxpac[2],iman[2],ipro[2]);
 }

 //Writing data
 if(mode=='W')
 {
  if(dtype=='S')
  {
   speed[addr]=para;
  }

  if(dtype=='M')
  {
   maxpac[addr]=para;
  }

  if(dtype=='O') //Manufaturer
  {
   iman[addr]=para;
  }

  if(dtype=='P') //Product
  {
   ipro[addr]=para;
  }
 }

 //Reading data
 if(mode=='R')
 {
  if(dtype=='S')
  {
   data2return=speed[addr];
  }

  if(dtype=='M')
  {
   data2return=maxpac[addr];
  }

  if(dtype=='O')
  {
   data2return=iman[addr];
  }

  if(dtype=='P')
  {
   data2return=ipro[addr];
  }
 }
```

```
 return(data2return);
}

long make_DirAddrLen(unsigned int count,unsigned char flow,unsigned int addr)
{
 unsigned long addr2return;

 addr2return =(long)(addr&0x7FFF);
 addr2return|=((long)flow)<<15;
 addr2return|=(((long)count)<<16);

 return(addr2return);
}

unsigned int get_port_speed(unsigned int port)
{
 if(port==1)
 {
    return(port1speed);
 }
 if(port==2)
 {
    return(port2speed);
 }
}



//set_port_speed - used in enable_port
void set_port_speed(unsigned int port, unsigned int speed)
{
 if(port==1)
 {
  port1speed=speed;
  addr_info(1,'W','S',speed);
 }

 if(port==2)
 {
  port2speed=speed;
  addr_info(2,'W','S',speed);
 }

 port_speed=speed;
}

unsigned int send_control(unsigned int *a_ptr,unsigned int *r_ptr)
{
 unsigned int cnt=retry;
 unsigned int active_bit;
 unsigned int abuf[128];
 unsigned int UpInt;
 unsigned int ccode;
 unsigned int timeout=9;
 unsigned int ccode1=0,ccode2=0;

 do
 {
```

```
  cnt=retry;

  write_atl(a_ptr,8);  //write 16 bytes

  hc_write(HcUpInt,0x100);
  hc_read32(HcATLDone); //read and clear done map, enables ATL interrupt
  hc_read32(HcATLDone); //read and clear done map, enables ATL interrupt
  hc_write(HcBufStatus,0x08);
  do
   {
    UpInt=hc_read(HcUpInt);
    if( (UpInt&0x100)!=0) {active_bit=0;}
    else {active_bit=1;}

    poll(50);
    cnt--;
   }
  while((cnt!=0)   &&   (active_bit!=0));
  hc_write(HcBufStatus,0x00);
  read_atl(r_ptr,72);

  //This part does 1 retry if ccode!=0, and records the errors
  ccode=((*r_ptr)&(0xF000))>>12;
  if(timeout==9) {ccode1=ccode;}
  if(timeout==8) {ccode2=ccode;}

  timeout--;
 }
 while(  (ccode!=0) && (timeout!=0)  );

 if(ccode1!=0 && ccode2!=0)
 {
  error_recorder('W',ccode1,ccode2);
 }
 return(cnt);
}

//sets the Host Controller to operational mode - the bits in HcBufferStatus
//register can now request the Host Controller to start processing the data in buffers
void set_operational(void)
{
    hc_write(HcHWCfg , 0x002D);
    hc_write32(HcFmItv , 0x25002EDF);
    hc_write32(HcControl , 0x00000680);
}

void enable_port(void)
{
    unsigned long dat32;
    hc_write32(HcRhP1,0x00000102);
    hc_write32(HcRhP2,0x00000102);
    hc_write32(HcRhA,0x05000B01);
    hc_write32(HcRhB,0x00000000);
    hc_write32(HcRhP1,0x00000102);
    hc_write32(HcRhP2,0x00000102);
    dat32=hc_read32(HcRhP2);

    if((dat32&0x00000001)==1)
```

```
    {
        set_port_speed(2,0);
        if(((dat32)&(0x00000200))!=0)
        {
            set_port_speed(2,1);
        }
    }

    dat32=hc_read32(HcRhP1);

    if((dat32&0x00000001)==1)
    {
        set_port_speed(1,0);
        if(((dat32)&(0x00000200))!=0)
        {
            set_port_speed(1,1);
        }
    }
}


void make_int_ptd(int *rptr,char token,char ep,int max,char tog,char addr,char port,int
freq)
{
 ptd2send.c_code=0x0F;
 ptd2send.active_bit=1;
 ptd2send.toggle=tog;
 ptd2send.actual_size=8;
 ptd2send.endpoint=ep;
 ptd2send.last_ptd=0;

 ptd2send.speed=port_speed;
 if(port==1) {ptd2send.speed=port1speed;}
 if(port==2) {ptd2send.speed=port2speed;}

 ptd2send.max_size=max;
 ptd2send.total_size=max;
 ptd2send.pid= token;
 ptd2send.format=0;
 ptd2send.fm=freq;
 ptd2send.func_addr=addr;

 *(rptr+0)= (ptd2send.c_code        &0x0000)<<12
           |(ptd2send.active_bit    &0x0001)<<11
           |(ptd2send.toggle        &0x0001)<<10
           |(ptd2send.actual_size   &0x03FF);

 *(rptr+1)=  (ptd2send.endpoint     &0x000F)<<12
           |(ptd2send.last_ptd      &0x0001)<<11
           |(ptd2send.speed         &0x0001)<<10
           |(ptd2send.max_size      &0x03FF);

 *(rptr+2)=  (0x0000                &0x000F)<<12
           |(ptd2send.pid           &0x0003)<<10
           |(ptd2send.total_size    &0x03FF);

 *(rptr+3)=  (ptd2send.fm            &0x00FF)<<8
           |(ptd2send.format        &0x0001)<<7
```

```
            |(ptd2send.func_addr     &0x007F);
}

void read_int(unsigned int *a_ptr, unsigned int data_size)
{
 int cnt;

 hc_write(HcTransferCnt,data_size*2);
 IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,HcINTLBufferPort);

 uSDelay(10);   //waste some time
 cnt=0;
 do
  {
   *(a_ptr+cnt)=IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);
   cnt++;
  }
 while(cnt<(data_size));
}

void write_int(unsigned int *a_ptr, unsigned int data_size)
{
 int cnt;

 hc_write(HcTransferCnt,data_size*2);
 IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,HcINTLBufferPort|0x80);

 uSDelay(10);   //waste some time
 cnt=0;
 do
  {
   IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA,*(a_ptr+cnt));
   cnt++;
  }
 while(cnt<(data_size));
}

unsigned int send_int(unsigned int *a_ptr,unsigned int *r_ptr)
{
 unsigned int cnt=retry;
 unsigned int active_bit;
 unsigned int abuf[128];
 unsigned int UpInt;

// unsigned int far *dptr;
 unsigned int  *dptr;

 dptr=a_ptr;

 write_int(a_ptr,8);  //write 16 bytes

 hc_write(HcUpInt,0x0100); // 80
 hc_write(HcBufStatus,0x08); // 04

 hc_read32(HcIntDone); //read and clear done map, enables ATL interrupt

 do
  {
```

```
   UpInt=hc_read(HcUpInt);
   if( (UpInt&0x80)!=0) {active_bit=0;}
   else {active_bit=1;}

   poll(50);
   cnt--;
  }
 while((cnt>0)   &&   (active_bit!=0));

 hc_write(HcBufStatus,0x00);

 read_int(r_ptr,72);

 return(cnt);
}



unsigned int set_address(int old_addr, int new_addr, int port)
{
 unsigned int cbuf[128];
 unsigned int rbuf[128];
 unsigned int uni_req[4]={0x0500,0x0000,0x0000,0x0000};
 unsigned int mycode=0;
 unsigned int tcnt;

 uni_req[1]=new_addr;

 hc_write(HcUpInt,0x100);

 hc_read32(HcATLDone);
 hc_read32(HcATLDone);

 make_ptd(cbuf,SETUP,0,8,0,old_addr,port);
 array_app(cbuf+4,uni_req,4);
 tcnt=send_control(cbuf,rbuf);
 mycode=(*rbuf&0xF000)>>12;

 if(tcnt==0)
 {
    mycode|=0xF000;
 }

 if(mycode==0)
 {
      //send out DATA IN packet
      make_ptd(cbuf,IN,0,0,1,old_addr,port);
      tcnt=send_control(cbuf,rbuf);
      mycode=(*rbuf&0xF000)>>12;
      if(tcnt==0)
    {
     mycode|=0xF000;
    }
 }

 hc_read32(HcATLDone);

 return(mycode);
```

```
}

void erase_all(void)
{
 unsigned long word_cnt,test_target;
 unsigned int test_size=4096;
 unsigned long DirAddrLen_data;
 unsigned long cnt;
 unsigned long rb;
 unsigned long *data_ptr;

 DirAddrLen_data=make_DirAddrLen(test_size,0,0);
 hc_write32(HcDirAddrLen,DirAddrLen_data);

 IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,HcDirAddrData |0x80);
 cnt=0;
  do
   {
    IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA,0);
    cnt++;
   }
  while(cnt<2048);
}

void reset_usb(void)
{
//if (g_is_PCI == TRUE)
 hc_write(HcHWCfg        , 0x302d    );  // set INT1 to Active LOW, Level Triggered
//else                       w16(HcHWCfg     , 0x342D    );
 hc_write32(HcFmItv        , 0x25002EDF);
 hc_write32(HcControl      , 0x00000600);
}



unsigned int assign_address(unsigned int addr1, unsigned int addr2, int mode)
{
 printf("assigning address\n");
 unsigned long rhp1,rhp2;
 unsigned int status;
 unsigned int ccode=0;

 rhp1=hc_read32(HcRhP1);
 rhp2=hc_read32(HcRhP2);

 printf("rhp1:%lx\n", rhp1);
 printf("rhp2:%lx\n", rhp2);
 enable_port();

// gotoxy(1,5);

 if(mode==1)
 {
  printf("\nRhP1 = %8lX  RhP2 = %8lX",rhp1,rhp2);
  printf("\nPort1Speed = %2d",get_port_speed(1));
  printf("\nPort2Speed = %2d",get_port_speed(2));
 }
```

```
 if(  ((rhp1&0x01)==1)  &&  ((rhp2&0x01)==1)  )
 {
  printf("in second if\n");
  if(mode==1)printf("\nBoth ports has USB device connected.");

  hc_write32(HcRhP1,0x00000010); //Resets port 1
//  delay(500);
  hc_write32(HcRhP2,0x00000010); //Resets port 2

  ccode=set_address(0,1,1);
  status=0x0100|(ccode<<12);

  enable_port();

//  delay(200);
  ccode=set_address(0,2,2);

  status=0x0001|(ccode<<4);
 }

 else if(  ((rhp1&0x01)==1)  &&  ((rhp2&0x01)==0)  )
 {
  if(mode==1)printf("\nPort 1 has USB device connected, assigning address 1...");
   ccode=set_address(0,1,1);

  status=0x0100|(ccode<<12);
 }

 else if(  ((rhp1&0x01)==0)  &&  ((rhp2&0x01)==1)  )
 {
  if(mode==1)printf("\nPort 2 has USB device connected, assigning address 2...");
   ccode=set_address(0,2,2);

  status=0x0001|(ccode<<4);
 }

 else
 {
  if(mode==1)printf("\nNo device connected to ISP1362, aborting enumeration...");

  status=0x0000;
 }

 return(status);
}

unsigned int get_control(unsigned int *rptr,unsigned int addr,char control_type,unsigned
 int extra,int port)
{
 unsigned int cbuf[128];
 unsigned int rbuf[128];
 unsigned int cnt=0,lcnt=0;
 unsigned int toggle_cnt=0;
 unsigned int word_size;
 unsigned int DesSize,MaxSize;

 unsigned int dev_req[4]={0x0680,0x0100      ,0x0000,0x8};
 unsigned int cfg_req[4]={0x0680,0x0200      ,0x0000,0x8};
```

```
 unsigned int str_req[4]={0x0680,0x0300      ,0x0000,0x8};
 unsigned int int_req[4]={0x0680,0x0400      ,0x0000,0x8};
 unsigned int end_req[4]={0x0680,0x0500      ,0x0000,0x8};
 unsigned int hid_req[4]={0x0681,0x2100      ,0x0000,0x8};

 unsigned int ccode=0;
 unsigned int stage=1;
 unsigned int tout; //timeout indicator

 //STAGE 1 : Send out first setup packet
 make_ptd(cbuf,SETUP,0,8,0,addr,port);
 if(control_type=='D') {array_app(cbuf+4,dev_req,4);}
 if(control_type=='C') {array_app(cbuf+4,cfg_req,4);}
 if(control_type=='S') {array_app(cbuf+4,str_req,4);}
 if(control_type=='I') {array_app(cbuf+4,int_req,4);}
 if(control_type=='E') {array_app(cbuf+4,end_req,4);}
 if(control_type=='H') {array_app(cbuf+4,hid_req,4);}

 if(control_type=='S')
 {
  cbuf[5]=cbuf[5]|extra;   //This is for string processing
 }

 tout=send_control(cbuf,rbuf);
 if(tout==0) {ccode|=0xF000;}   //Indicates Timeout in transaction
 printf("\nccode0 = %x" , ccode);
 if(ccode==0)
 {
  toggle_cnt++;
  make_ptd(cbuf,IN,0,8,toggle_cnt%2,addr,port);
  tout=send_control(cbuf,rbuf);
  ccode|=(rbuf[0]&0xF000)>>12;
  //printf("\nccode_control = %x" , ccode);
  if(ccode==0x09)  //Descriptor Size is less than 8
  {
   ccode=0;
  }

  if(tout==0) {ccode|=0xF000; printf("Timeout Stage1");}   //Indicates Timeout in
transaction

  if(control_type!='C')
  {
   DesSize=((rbuf[4]&0x00FF));
  }

  if(control_type=='C')
  {
   DesSize=rbuf[5];
  }

  if(control_type!='D')
  {
   MaxSize=addr_info(addr,'R','M',MaxSize);
  }

  if(control_type=='D')
  {
```

```
   MaxSize=(rbuf[7]&0xFF00)>>8;
   printf("\nMaxSize =%2d",MaxSize);
   if(MaxSize<8) {MaxSize==8;}
   if(MaxSize == 0) {MaxSize = 8;} // added by me!
   addr_info(addr,'W','M',MaxSize);
  }

  if(control_type=='H')
  {
   DesSize=(rbuf[7]&0xFF00)>>8;
   if(DesSize<=8) {DesSize==8;}
  }
  //printf("\nDesSize = %2d MaxSize =%2d",DesSize,MaxSize);
  //printf("\nrbuf[7] = %2d MaxSize =%2d",rbuf[7],MaxSize);
 }
 //printf("\nccode = %x" , ccode);
 if(ccode==0)
 {
  //send out DATA OUT packet
  make_ptd(cbuf,OUT,0,0,toggle_cnt%2,addr,port);
  tout=send_control(cbuf,rbuf);
  if(tout==0) {ccode|=0xF000; printf("timeout");}   //Indicates Timeout in transaction

  ccode|=(rbuf[0]&0xF000)>>12;
  printf("\nrbuf[0]: %x", rbuf[0]);
 }
 //STAGE 1: END


 if(ccode==0)
 {
  stage=2;

  hid_req[1]=0x2200; //Change HID req into HID report descriptor

  //STAGE 2
  make_ptd(cbuf,SETUP,0,8,0,addr,port);
  if(control_type=='D') {array_app(cbuf+4,dev_req,4);}
  if(control_type=='C') {array_app(cbuf+4,cfg_req,4);}
  if(control_type=='S') {array_app(cbuf+4,str_req,4);}
  if(control_type=='I') {array_app(cbuf+4,int_req,4);}
  if(control_type=='E') {array_app(cbuf+4,end_req,4);}
  if(control_type=='H') {array_app(cbuf+4,hid_req,4);}

  if(control_type=='S')
  {
   cbuf[5]=cbuf[5]|extra;
  }

  cbuf[7]=DesSize;
  tout=send_control(cbuf,rbuf);
  if(tout==0) {ccode|=0xF000; printf("\nTimeout stage 2");}   //Indicates Timeout in
transaction

  word_size=(DesSize+1)>>1;

  cnt=0;
```

```
  //send out DATA IN packet
   new_make_ptd(cbuf,IN,0,MaxSize,1,addr,port,DesSize);

   tout=send_control(cbuf,rbuf);
   if(tout==0) {ccode|=0xF000; printf("Data In Timeout!");}   //Indicates Timeout in
transaction

   ccode|=(rbuf[0]&0xF000)>>12;
   if(ccode==0)//Data In is sucessful
   {
    //printf("\ndata in successful!");
    lcnt=0;
    do
    {
     //Copy the data located right after the 8 bytes PTD
     *(rptr+cnt)=rbuf[4+lcnt];

     cnt++;
     lcnt++;
    }
    while(lcnt<DesSize);
   }
 //STAGE 2: END
 }

 if(ccode==0)
 {
  stage=3;
  //printf("stage 3");
  //STAGE 3 :Send out DATA OUT packet
  make_ptd(cbuf,OUT,0,0,toggle_cnt%2,addr,port);
  send_control(cbuf,rbuf);

  ccode=(rbuf[0]&0xF000)>>12;
  //STAGE 3: END
 }

 return( (ccode)|(stage<<8) );
// byte 0 indicates the error code
// byte 2 indicates at which stage the error was encountered
// byte 3 is F if time-out, else 0
}


unsigned int  set_config(int addr, int config)
{
 unsigned int cbuf[128];
 unsigned int rbuf[128];
 unsigned int uni_req[4]={0x0900,0x0000,0x0000,0x0000};
 unsigned int tcnt;
 unsigned int mycode=0;

 uni_req[1]=config;

 hc_write(HcUpInt,0x100);

 hc_read32(HcATLDone);
 hc_read32(HcATLDone);
```

```
 make_ptd(cbuf,SETUP,0,8,0,addr,addr);

 array_app(cbuf+4,uni_req,4);
 tcnt=send_control(cbuf,rbuf);
 if(tcnt==0) { mycode|=0xF000;}
 mycode=mycode | (*rbuf&0xF000)>>12;

 if(mycode==0)
 {
 //send out DATA IN packet
  make_ptd(cbuf,IN,0,0,1,addr,addr);
  tcnt=send_control(cbuf,rbuf);
  if(tcnt==0) { mycode|=0xF000;}
  mycode=mycode | (*rbuf&0xF000)>>12;
 }

 hc_read32(HcATLDone);
 hc_read32(HcATLDone);

 return(mycode);
}

unsigned short read_endpoint (unsigned char EPIndex , unsigned short* ptr , unsigned short
 LENGTH)
{
unsigned short j,i;
/* Select endpoint */
//dc_write( READ_EP+EPIndex); /* READ_EP = 0x10 */
j = IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE, DC_DATA); /* Read the length in bytes
inside the OUT buffer */
if( j > LENGTH)
j = LENGTH;
for(i=0 ; i<j ; i++)
{ /* Read buffer */
*(ptr+i) = IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,DC_DATA);
}
/* Clear buffer */
//IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE, DC_COM , CLEAR_BUFF + EPIndex);
return j;
}

int try(unsigned int addr, unsigned int command){
unsigned int cbuf[128];
unsigned int rbuf[128];
unsigned int uni_req = command;
//int parameters
 unsigned long int_skip=0xFFFFFFFE;
// unsigned long int_done=0;
 unsigned long int_last=0x00000001;
 unsigned int  int_blk_size=64;
 unsigned int mycode=0;

 unsigned int atllen,ptllen,intllen;
 unsigned int int_start;

  unsigned int freq;
  //unsigned int tog = 0;
```

```
 //freq=0x00;

 erase_all();

 hc_write(HcBufStatus,0x00);

 //Setup Int Parameters
 hc_write32(HcIntSkip,int_skip);
 hc_write32(HcIntLast,int_last);
 hc_write(HcIntBlkSize,int_blk_size);

 //Setup Buffer
 atllen  =hc_read(HcATLBufferSize);
 ptllen  =hc_read(HcISTLBufferSize);
 intllen =hc_read(HcINTLBufferSize);

 //int_start=ptllen+ptllen;
//hc_write(HcUpInt,0x100);

 //hc_read32(HcATLDone);
 //hc_read32(HcATLDone);
// e_box(320,192,580,266);
// e_box(bx    ,by,40,30);
// e_box(bx+50 ,by,40,30);
// e_box(bx+100,by,40,30);

// advlogo();

// gotoxy(5,1);
 printf("\n@Port %X",addr);


    //send out DATA IN packet

   make_int_ptd(cbuf,SETUP,0,1,0,addr,addr,freq);
   //*((char*)cbuf + 8) = 0x40;
   unsigned int tcnt = send_int(cbuf,rbuf);
   if(tcnt==0) { mycode|=0xF000;}
    mycode=mycode | (*rbuf&0xF000)>>12;
   if(mycode ==0){
   //send out DATA O packet
   make_int_ptd(cbuf,OUT,0,1,0,addr,addr,freq);
   *((char*)cbuf + 8) = command;
   unsigned int tcnt = send_int(cbuf,rbuf);
   if(tcnt==0) { mycode|=0xF000;}
    mycode=mycode | (*rbuf&0xF000)>>12;
   }
   //printf("\n HcRhP2 %x", (hc_read(HcRhP2) & 0x01));
//   if(mycode ==0){
//    make_int_ptd(cbuf,OUT,0,1,0,addr,addr,freq);
//   *((char*)cbuf + 8) = 0x08;
//   unsigned int tcnt = send_int(cbuf,rbuf);
//   if(tcnt==0) { mycode|=0xF000;}
//    mycode=mycode | (*rbuf&0xF000)>>12;
//   if(mycode ==0){
//   //send out DATA O packet
//   make_int_ptd(cbuf,IN,0,1,0,addr,addr,freq);
```

```
//   //*((char*)cbuf + 8) = command;
//   unsigned int tcnt = send_int(cbuf,rbuf);
//   if(tcnt==0) { mycode|=0xF000;}
//    mycode=mycode | (*rbuf&0xF000)>>12;
//   }
//   }
//   if(mycode == 0){
//    make_int_ptd(cbuf,OUT,0,1,0,addr,addr,freq);
//   *((char*)cbuf + 8) = 0x20;
//   unsigned int tcnt = send_int(cbuf,rbuf);
//   if(tcnt==0) { mycode|=0xF000;}
//    mycode=mycode | (*rbuf&0xF000)>>12;
//   if(mycode ==0){
//   //send out DATA O packet
//   make_int_ptd(cbuf,IN,0,1,0,addr,addr,freq);
//   //*((char*)cbuf + 8) = command;
//   unsigned int tcnt = send_int(cbuf,rbuf);
//   if(tcnt==0) { mycode|=0xF000;}
//    mycode=mycode | (*rbuf&0xF000)>>12;
//   }
//
//   }
   //if(mycode == 0){
   //make_int_ptd(cbuf,IN, 0x00, 8, 0, addr, addr, freq);
   //send_int(cbuf, rbuf);
   //}
   //make_ptd(cbuf,SETUP,0,8,0,addr,addr);
   //send_control(cbuf,rbuf);
   //make_int_ptd(cbuf,SETUP,1,8,0,addr,addr, freq);
   //send_int(cbuf,rbuf);

   //make_int_ptd(cbuf,OUT,1,8,0,addr,addr, freq);
   //send_int(cbuf,rbuf);
   //printf("timcnt: %x\n",timecnt);
   //make_int_ptd(cbuf,IN,0,0,0,0,addr,freq); // Makes the PTD
   //send_int(cbuf,rbuf);

  /*make_ptd(cbuf,SETUP,0,1,0,addr,addr);

  unsigned int tcnt=send_control(cbuf,rbuf);
 if(tcnt==0) { mycode|=0xF000;}
 mycode=mycode | (*rbuf&0xF000)>>12;

 if(mycode==0)
 {



   make_ptd(cbuf,OUT,0,1,1,addr,addr);
   *((char*)cbuf + 8) = command;
   //array_app(cbuf+2,uni_req,1);
   tcnt=send_control(cbuf,rbuf);

   if(tcnt==0) { mycode|=0xF000;}
   mycode=mycode | (*rbuf&0xF000)>>12;
   printf("\nmycode = %x",mycode);
 }
   if(mycode==0)
```

```
   {
 //send out DATA IN packet
  make_ptd(cbuf,IN,0,0,1,addr,addr);
  *((char*)cbuf + 8) = command;
  tcnt=send_control(cbuf,rbuf);
  if(tcnt==0) { mycode|=0xF000;}
  mycode=mycode | (*rbuf&0xF000)>>12;
 }

 hc_read32(HcATLDone);
 hc_read32(HcATLDone);
*/
 return(mycode);


}




int main()
{
  //unsigned int i;

  //test_0 = IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,2);

  //printf("test_0 is %4x\n", test_0);
  //IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,0, 0x08);
  unsigned int chipID = hc_read(0x27);//(HcChipID);

  //unsigned int hc_com = IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM);
  //printf("hc_com is %4x\n", hc_com);
  printf("chip ID0 is %4x\n", chipID);

  unsigned int mycode = 0;
  unsigned int status = 1;
  unsigned int rocket02 = 0;
  unsigned int iManufacturer,iProduct;

  unsigned int rbuf[128];
  unsigned int fire = 0x10;
  //configuring the buffersizes
  //original buffersize 0x3630

  //chipID = hc_read(0x27);

  unsigned long scratch = hc_read(HcScratch);
  printf("scratch_before is %lx\n", scratch);
  hc_write32(HcScratch, 0xdeadbeef);



  set_operational();
  enable_port();

  reset_usb();
  erase_all();
```

```
  set_operational();
  enable_port();


  //uSDelay(20);

  //write(HcScratch, 0x09);
  chipID = hc_read(0x27);
  hc_write32(HcFmItv , 0x25002EDF);
  printf("scratch_after is %lx\n", scratch);
  printf("chipID is %4x\n", chipID);



  unsigned int buff = hc_read32(HcFmItv);
  printf("buff is %x\n", buff);

  hc_write(HcATLBufferSize, 1536);
  hc_write(HcINTLBufferSize,1024);
  hc_write(HcISTLBufferSize, 512);

  //Interrupts disabled for now
  hc_write(HcUpIntEnable, 0);
  hc_write32(HcIntDisable, 0xFFFFFFFF);

  //controls of the ATL buffer
  hc_write32(HcATLLast, 1 ); // First PTD is the last
  hc_write(HcATLBlkSize, 64 ); // Block size of 64 bytes
  hc_write(HcATLPTDDoneThrsCnt , 1 ); // Generates interrupt for every PTD Done
  hc_write(HcATLPRDDoneTimeOut, 5 ); // 5 ms before giving up on NAKs

  //Disableing processing of all buffers
  hc_write(HcBufStatus, 0);

  //status = assign_address(1,2,0);
  status = assign_address(1,2,1);
  mycode=get_control(rbuf,1,'D',0,1);

  printf("\nmycode: %4x\n", mycode);
  printf("status: %4x\n", status);


  hc_write(HcHWCfg , 0x002D);
  hc_write32(HcFmItv , 0x25002EDF);

  hc_write32(HcControl , 0x00000680);


  if( (status&0x0001)!=0) //port 2 active
 {
//  Check port 2 for launcher
  mycode=get_control(rbuf,2,'D',0,2);
  mycode=get_control(rbuf,2,'D',0,2);
  printf("\nGetting device descriptor for device at port 2... ");
  printf("%04X",mycode);
  if(mycode==0x0300)
  {
   iManufacturer = rbuf[7]&0xFF;
```

```
  iProduct = (rbuf[7]&0xFF00)>>8;

  addr_info(2,'W','O',iManufacturer);
  addr_info(2,'W','P',iProduct);

  mycode=get_control(rbuf,2,'H',addr_info(2,'R','P',0),2);
  printf("\n mycode0 = %x\n", mycode);
  if( *(rbuf+1)==0x0209  )
  {
   printf("\nMouse Detected @Port2!!! ");

   //rocket02=1;
  }
  if( *(rbuf+1)==0x0609  )
  {
   printf("\nKeyboard Detected @Port2!!! ");

  // rocket02=1;
  }
   if( *(rbuf+1)==0x09ff  )
  {
   printf("\nLauncher Detected @Port2!!! ");

   rocket02=1;
  }
 }
}
 if(rocket02 ==1)
{
 mycode=set_config(2,2);
 //printf("\nSetting config of device 2 to config 2... %04X",mycode);

 //if(mycode==0)
 //{
  //printf("\nset_config successful!");
  unsigned int try_succ = try(2,0x01);
  printf("\ntry_succ = %x", try_succ);
  //void make_ptd(int *rptr,char token,char ep,int max,char tog,char addr,char port)
  //IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE, DC_DATA , 0x10);
 //}
}
 //unsigned long control = hc_read32(HcControl);
 //printf("Hello from Nios II!\n");
 //buff = hc_read(HcATLBufferSize);
 //printf("\n*rbuff +1 is %4x\n", *(rbuf+1) );
 //printf("\niMan = %4x\n", iManufacturer);
 //printf("iprod = %4x\n", iProduct);
 /*for(i = 0x00; i < 0x50; i++)
 {
   //IOWR_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_COM,i);
   //uSDelay(100);
   //unsigned int j = IORD_ISP_DATA(ISP1362_AVALON_SLAVE_0_BASE,HC_DATA);
   //uSDelay(100);

   unsigned int j = hc_read(i);
   printf("HC_reg %x is %x\n", i, j);

 }  */
```

```
  //printf("chip ID is %4x\n", chipID);
  //printf("HcControl %lx\n",control);
  //printf("ATL buffer is %4x\n",HcALTbufferSize );
  return 0;
}
```

```c
#include <io.h>
#include <stdio.h>

// define read and write to memory
/*
#define IORD_CORD_DATA(base, offset) \
    IORD_16DIRECT(base, (offset) * 2)

#define IOWR_STEP_DATA(base, offset, data) \
    IOWR_16DIRECT(base, (offset) * 2, data)

*/

// functions used to calculate launch position
int launch_right(float x, float y, float z, int elev, int azim);
int launch_top(float x, float y, float z, int elev, int azim);
int launch_left(float x, float y, float z, int elev, int azim);

const float g = 9.8;    // accel due to gravity
const float u_r = 9.863;    // initial speed of right dart
const float u_t = 8.87;     // initial speed of top dart
const float u_l = 8.189;    // initial speed of left dart

int main()
{
    // Variable declarations
    float x, y, z;  // input coordinates of target
    int elev_step, azim_step;  // output elevation and azimuth step

    char curr_dart;  // current dart to be launched
    int curr_elev;  // current elevation of launcher
    int curr_azim;  // current azimuth of launcher




    //int i = 123;
    //printf("%d",i);

    return 0;


}

int launch_right(float x, float y, float z, int elev, int azim)
{
}

int launch_top(float x, float y, float z, int elev, int azim)
{
}

int launch_left(float x, float y, float z, int elev, int azim)
{
}
```

```
1   --Legal Notice: (C)2007 Altera Corporation. All rights reserved.
    Your
2   --use of Altera Corporation's design tools, logic functions and other
3   --software and tools, and its AMPP partner logic functions, and any
4   --output files any of the foregoing (including device programming or
5   --simulation files), and any associated documentation or
    information are
6   --expressly subject to the terms and conditions of the Altera Program
7   --License Subscription Agreement or other applicable license
    agreement,
8   --including, without limitation, that your use is for the sole
    purpose
9   --of programming logic devices manufactured by Altera and sold by
    Altera
10  --or its authorized distributors.  Please refer to the applicable
11  --agreement for further details.
12
13
14  -- turn off superfluous VHDL processor warnings
15  -- altera message_level Level1
16  -- altera message_off 10034 10035 10036 10037 10230 10240 10030
17
18  library altera;
19  use altera.altera_europa_support_lib.all;
20
21  library ieee;
22  use ieee.std_logic_1164.all;
23  use ieee.std_logic_arith.all;
24  use ieee.std_logic_unsigned.all;
25
26  entity xy_pos is
27          port (
28                -- inputs:
29                   signal address : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
30                   signal chipselect : IN STD_LOGIC;
31                   signal clk50 : IN STD_LOGIC;
32                   signal h_pos : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
33                   signal read : IN STD_LOGIC;
34                   signal read_data : IN STD_LOGIC_VECTOR (29 DOWNTO 0
    );
35                   signal reset_n : IN STD_LOGIC;
36                   signal v_pos : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
37                   signal write : IN STD_LOGIC;
38                   signal writedata : IN STD_LOGIC_VECTOR (15 DOWNTO 0
    );
39
40                -- outputs:
41                   signal readdata : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
42                );
43  end entity xy_pos;
44
45
46  architecture europa of xy_pos is
```

```
47   component xy_detector is
48              port (
49                    -- inputs:
50                       signal address : IN STD_LOGIC_VECTOR (4 DOWNTO 0
     );
51                       signal chipselect : IN STD_LOGIC;
52                       signal clk50 : IN STD_LOGIC;
53                       signal h_pos : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
54                       signal read : IN STD_LOGIC;
55                       signal read_data : IN STD_LOGIC_VECTOR (29
     DOWNTO 0);
56                       signal reset_n : IN STD_LOGIC;
57                       signal v_pos : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
58                       signal write : IN STD_LOGIC;
59                       signal writedata : IN STD_LOGIC_VECTOR (15
     DOWNTO 0);
60
61                    -- outputs:
62                       signal readdata : OUT STD_LOGIC_VECTOR (15
     DOWNTO 0)
63                    );
64   end component xy_detector;
65
66                  signal internal_readdata :  STD_LOGIC_VECTOR (15
     DOWNTO 0);
67
68   begin
69
70     --the_xy_detector, which is an e_instance
71     the_xy_detector : xy_detector
72       port map(
73         readdata => internal_readdata,
74         address => address,
75         chipselect => chipselect,
76         clk50 => clk50,
77         h_pos => h_pos,
78         read => read,
79         read_data => read_data,
80         reset_n => reset_n,
81         v_pos => v_pos,
82         write => write,
83         writedata => writedata
84       );
85
86
87     --vhdl renameroo for output signals
88     readdata <= internal_readdata;
89
90   end europa;
91
92
```