Thing-A-Ma-Flipper (TAMF)

Rhonda Jordan
Eveliza Herrera
Amon Wilkes
Essa Farhat

March 31, 2004

Embedded Systems and Design
W4840 – Prof. Edwards

Table of Contents

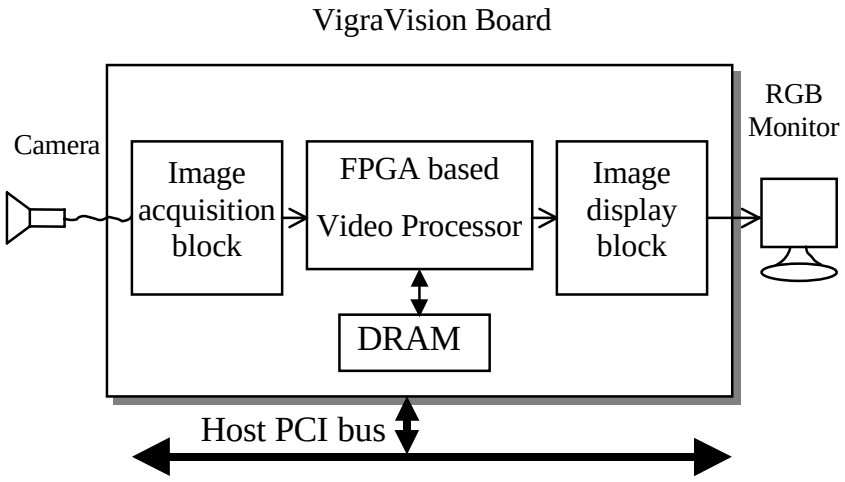
Thing-A-Ma-Flipper

The Spartan™-IIE Field-Programmable Gate Array (FPGA) is the main repository of programmable logic on the XSB board. The board is provided to students in the Embedded Systems CSEE W4840 course, which includes many features such as a 4 Mbit Flash RAM, a 256K x 16 SRAM, 16M x 16 SDRAM, Video DAC, Compact flash interface, a video decoder and many other features. The video decoder is a Philips SAA7114H chip that can accept up to six signals through dual RCA jacks and dual S-Video connectors. Applications of this include capturing and scaling video images to be provided as digital video stream through the image port of a VGA controller, for display via the frame buffer, or for capture to system memory[1]. The Flash RAM, SRAM, and SDRAM are used to store general–purpose data. The Video DAC generates the analog red, green, and blue signals for the VGA display while the FPGA generates the horizontal and vertical sync pulses directly. In our project we will use some of the features on the XSB board to create a video effects generator.

We will use the Philips SAA7114H chip to capture and scale video images. This process involves an image acquisition through a digital camera and laptop. The S-Video ports are directly connected to the video decoder and the digitized video signals are then sent to the FPGA and placed into a frame buffer. We will read the image and store it in the RAM of the XSB board, we will then retrieve this data from the RAM and format it into lines of pixels and send the lines to the monitor with appropriate pulses. We will use the video scaling feature on the video decoder along with clever compression algorithms to display an image on the VGA monitor and to compress and expand an image. While compression is taking place, it must be taken into consideration that there will be distortion, as we are not evenly scaling the picture. We will also use the Texas Instrument video DAC (THS8133B) to generate the video signals for a VGA display. Ultimately, our goal is to display an image onto the screen, horizontally compress the image about the center of the screen until the image is a width of one pixel (one vertical line), and finally expand it to its original size as a mirror image of the original input.

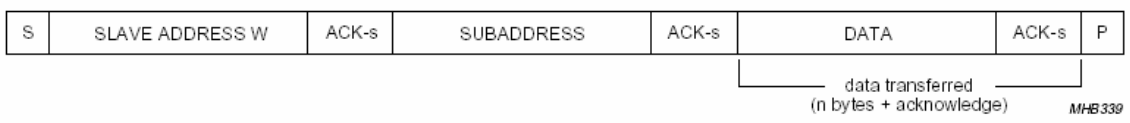
1 Philips Semiconductor SAA7114H. Data Sheet. March 14, 2000 (pg. 4)

II. Implementation

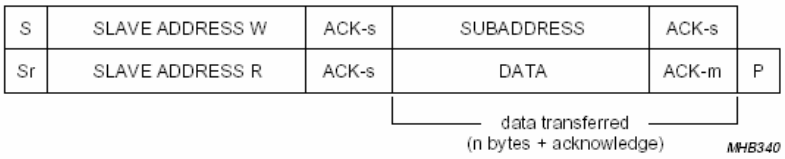


We will start by simply taking a picture with a digital camera (Canon S320 Powershot), and connecting the digital camera to a laptop through a USB connection. We will download the JPG image onto the laptop and convert it to a TIFF file. We will transfer image data from the laptop to the XSB Board via the S-Video ports.

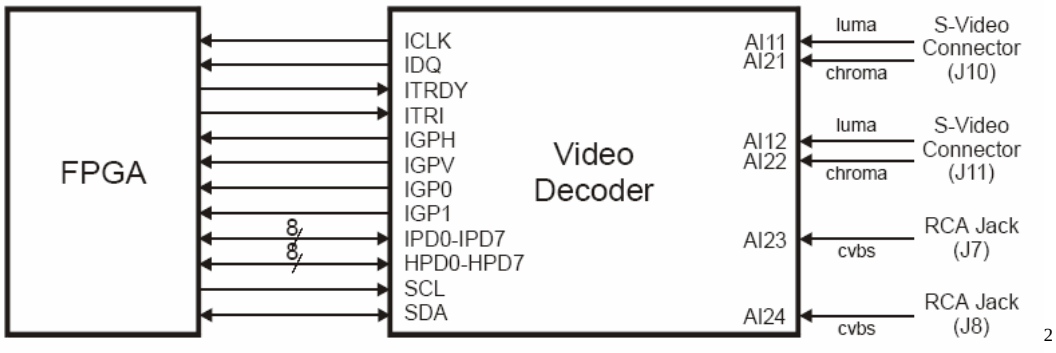
The XSB board can digitize NTSC, SECAM, and PAL video signals using the Philips SAA7114H Video Decoder chip. The digitized video that enters arrives at the FPGA over the IPD and HPD buses when IDQ is active. The arrival of the video data is synchronized with the ICLK driven from the FPGA. The FPGA programs the video decoder options using the I^2C bus. After the signal is digitized it is sent to the FPGA and stored into RAM. You will find below the format of the I^2C bus and the connections between the FPGA and the Video Decoder.



a. Write procedure.

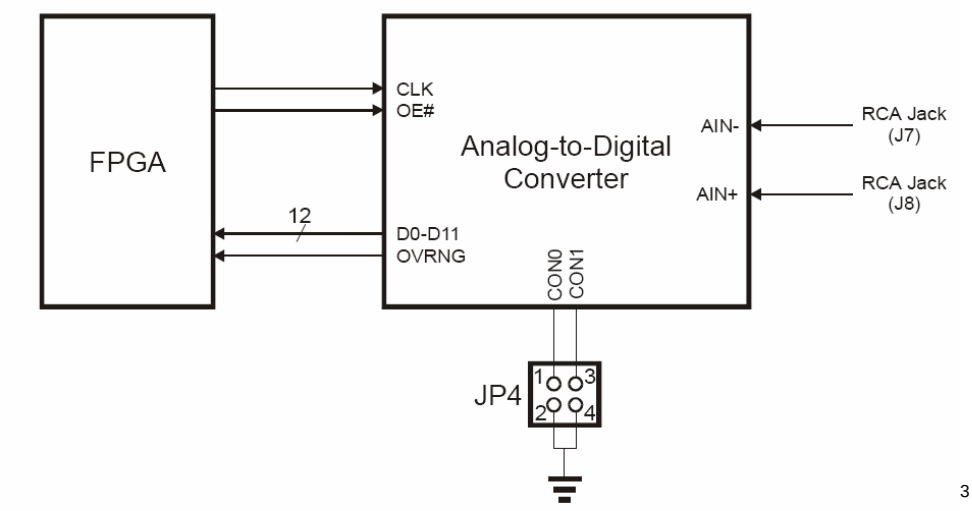


b. Read procedure (combined).



The FPGA also uses a Texas Instruments video DAC (THS8133B) to generate the video signals for a VGA display. The FPGA passes 30-bit pixel data (10 bits for the red, green and blue color components) to the video DAC on each clock edge. The DAC generates the analog red, green and blue signals for the VGA display while the FPGA generates the horizontal and vertical sync pulses directly. Below is an illustration of the pinouts of the Video-DAC on the XSB Board.

2 XSB Board V1.0 Manual. XESS Corporation August 20, 2003 (pg. 27)



3

Once the image is stored on the RAM, the digital-to-analog converter (DAC) is used to convert it to analog data for the display scanning mechanism. Once the display information is in analog form, it is sent to the monitor through a VGA cable. See the diagram below:

4

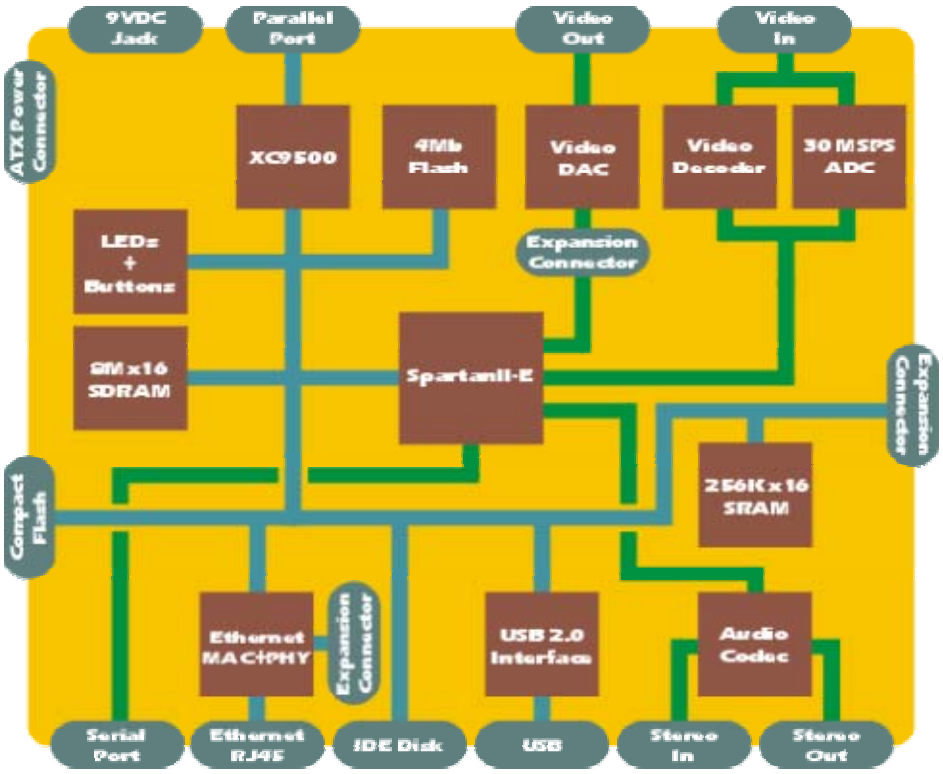
1: Red out	**6**: Red return (ground)	**11**: Monitor ID 0 in
2: Green out	**7**: Green return (ground)	**12**: Monitor ID 1 in or data from display
3: Blue out	**8**: Blue return (ground)	**13**: Horizontal Sync out
4: Unused	**9**:	**14**: Vertical Sync **5**: Ground
10: Sync return (ground)	**15**: Monitor ID 3 in or data clock	

3 XSB Board V1.0 Manual. XESS Corporation August 20, 2003 (pg. 30)
4 http://computer.howstuffworks.com/monitor4.htm

Why S-Video?

In our design we had several ideas as to how we can transfer an image onto Philips chip. The XSB-300E board has a variety of interfaces for communicating with the outside world: parallel and serial ports, USB 2.0 port, and S-Video port. One possible, most costly, design would be to connect a digital camera directly to the on-board S-Video interface. Through research we discovered that this would be a costly process, since it will require more memory. Images coming from a digital camera are normally JPEG or BMP. A JPEG image file format is a compressed raster image. We then have the option of decompressing the image and then sending it through the video decoder or transferring the image onto the Philips chip and then performing the decompression. Either way, the JPG image format is a 24-bit RGB image, which uses lossy compression methods. Without getting into too much detail in this section about image formats, later we will discuss the difference between jpg, bmp, tiff, and gif formats.

One other possible solution of placing an image onto the Xilinx board/memory would be to use a Compact Flash memory card. The Compact Flash card is a form of nonvolatile data storage and retrieval in which the XSB board can access. The FPGA stores and retrieves data from the card by reading and writing registers on the card through the Compact Flash interface. The major discrepancy with the Compact Flash is that it is not directly connected on the same I/O as the video decoder. The diagram below shows the schematic of the connections.



The Universal Serial Bus provides an expandable, hot-pluggable interface that ensures a standard, low-cost connection for peripheral devices; such as the digital camera or a Compact Flash memory device. The XSB board acts as a USB 2.0 peripheral through a USB chip. The FPGA controls the USB chip by reading and writing registers and FIFO buffers on the chip through a microprocessor bus interface. Similarly, if we connect a USB 2.0 device directly to the Xilinx board, we will have to overcome the same design issues associated with the interlacing I/O buses located on the board.

Finally, through research and discussions with both Josh Mackler and Cristian Soviani, we concluded that our best solution would be to connect a device to the S-Video port on the Xilinx board. We decided to use Super-Video (S-Video) because it's a technology for transmitting video signals over a cable by dividing the video information into two separate signals: one for brightness (Luminance Y) and one for color (Chrominance C). Our computer monitors are designed for RGB signals. Most digital video devices, such as digital cameras, produce video in RGB format. The images look best, therefore, when output on a computer monitor. To use the S-Video, the device sending the signals must support S-Video output and the device receiving the signals must have an S-Video input

jack. In our case, we are going to connect an IBM Thinkpad T40's S-Video port to the XSB board's S-Video to make the image transfer. Alternatively we could connect a digital camera to the laptop using the S-Video to S-Video connection to make the transfer.

Why TIFF?

Along with deciding how we are going to capture the image, we had to decide which type of image we want to work with. There are as many as 100 different types of image formats and we decided to investigate the four main types which are JPEG, GIF, GIF, and TIFF.

Short for *Joint Photographic Experts Group,* JPEG is a data compression technique for color images where some amount of data is lost. An advantage on using JPEG is that its variable compression allows file size control. It can support high levels of compression; however, some details are lost in the compression. JPEG does not provide superior results for simpler pictures that contain few colors, broad areas of similar color, or stark differences in brightness. Like JPEG, Graphics Image File (GIF) is able to handle high compression, however it is able to handle compression by 40% or more without losing any data.5 GIF are good for multi-image files (animated images) and is supported on numerous platforms. The disadvantage of using GIF is that it has a 256-color palette and detailed pictures and photo-realistic images lose color information and look paletted.

Unlike JPEG, BMP images do not support compression. Although BMP is uncompressed and well-supported under MS Windows, it is poorly supported everywhere else. Since there is no compression it results in a very large file, which can be an inconvenience in certain situations. Tagged image file formats (TIFF) like BMP is also uncompressed and is rated one of the best choices for data exchange between media.6 TIFF is arguably the most widely supported graphic file format in the printing industry

5 http://animalscience.ucdavis.edu/Intranet/Graphics/PictureFormat/default.htm
6 http://www.csc.fi/visualization/viikki/formats.html

and supports optional compression. One of the few disadvantages is that TIFF is not suitable for viewing in Web browsers.

Considering we want an uncompressed image and the fact that we are using a Linux machine, we decided that using a TIFF format for our image is the best way to go.

III. Image Capture

We will read the stored image information from RAM and use a C program to convert each pixel into ASCII values which will then be used to write to the screen. The following C-code is similar to what will be implemented and it allows the user to read in an image and print out the ASCII values to a file.

Specifically this program reads 1-byte images and output them as ASCII text one byte at a time. The data has been stored in 2500 byte blocks, NOT 512 byte blocks. Since the binary imagery is character data, it cannot be browsed. However, the character data can easily be converted into integer values.

Code:

```
static FILE *fp1, *fp2;

int main(int argc, char *argv[])
{
 char image[2260000];
 int xsize=2500, ysize=904, i=0;
 register int I;
 short data;

       //checks number of args
   if (argc !=3) {
      printf("usage image_read inputfile outputfile\n");
      exit(1);}
       //opens a binary file for reading

    if (!(fp1=fopen(argv[1],"rb"))){
     printf("cannot open file %s to read\n", argv[1]);
     exit(1);}
       //creates an outputfile to store ASCII int values

    if (!(fp2=fopen(argv[2],"w"))){
       printf("Cannot open file %s to write\n",argv[2]);
       exit(1); }

       //converts binary pixel val to ASCII int value
      fread(&image[0],1,xsize*ysize,fp1);

      for(I=0;i<2260000;i++) {
         data=(short) image[i];
         fprintf(fp2,"%d\n",data);
       }
/*    Write output to user-defined file   */
} //end main()
```

Here is a synopsis of the standard format for an image W pixels wide by H pixels high.

Example: 6 bytes of data. Three two-byte integers (high order byte first - "big endian"):

Bytes 1 -> 2: The number of Columns in the image (W). Valid values are 1 - 65535
Bytes 3 -> 4: The number of Rows in the image (H). Valid values are 1 - 65535
Bytes 5 -> 6: The number of colors. Valid values are 1 or 3.

The number of colors is 1 if the image is monochrome (gray scale) or 3 if the image is color. In that case there are three color planes stored in this order: Red (first), Green, Blue (last).

Bytes 7 →(WxH + 6): Monochrome color data if the number of colors is 1.
Bytes 7 → (WxH + 6): Red color data if the number of colors is 3
Bytes (WxH + 6) → (2xWxH + 6): Green color data
Bytes (2xWxH + 6) → (3xWxH + 6) Blue color data

The data are bytes read out in rows running from the top left. The printer's convention is used: 0x00 is white (or maximum amount of a color) and 0xFF is black.

This program will take an image and convert each pixel character data into sequential integer ASCII values. From there, we take the integer value, completely ignoring RAM, and store the ASCII values in a ROM, similar to the task executed in Lab 3 with the font_8x8 ROM file. We will then take the integer value, corresponding to the pixel value, and enable that pixel on the screen. This process will be done using VHDL. This is one hypothesis of describing how to display an image on the VGA. Later we will describe several other methods which actually use the 2x16 SRAM on board the Xilinx.

IV. Horizontal Image Compression[7]

The image has now been captured and horizontal image compression must now be executed. Horizontal image compression is the act of reducing the size of the horizontal axis while maintaining the vertical dimension of the original image.

There are two ways to approach this task. The first approach involves the creation of a C program that instructs the video decoder to continuously downscale the captured image to a pre-determined horizontal frame size. The program will generate the desired horizontal scaling values, which are then utilized by the horizontal scaling feature of the Philips SAA7114H Video Decoder. The horizontal scaling block of the decoder is comprised of a horizontal prescaler and a fine scaler. The prescaler consists of an FIR anti-alias filter and an integer prescaler, which builds an adaptive prescale dependent low-pass filter, to balance sharpness and aliasing effects.

The functionality of the prescaler is defined by the following parameters:

XPSC[5:0]A0H[5:0]: (= 1 to 53) covers the integer downscale range 1 to 1/63
XACL[5:0]A1H[5:0]: (= 0 to 63) averaging sequence length, range 1 to 64
XDCG[2:0]A2H[2:0]: DC gain renormalization, 1 down to 1/128
XC2_1[A2H[3]]: (= 0 or 1) defines the weighting of the incoming pixels

- Equation for XPSC calculation:
  XPSC[5:0] = lower integer of Npix_in/Npix_out
  Where Npix_in = number of input pixels
  Npix_out = number of desired output pixels over the complete horizontal scaler.

7 Information presented in this section has been obtained from the Philips Semiconductor SAA7114H. Data Sheet. March 14, 2000 (pgs. 42-47)

- XACL[5:0] can be used to vary low-pass filter characteristics for a given prescale of 1/XPSC[5:0], which determines the compromise between bandwidth (=sharpness) and alias effects.
- DC gain = ((XACL – XC2_1) +1) x (XC2_1 +1)

The horizontal fine scaler implements variable phase delay VPD circuitry. This scaler calculates horizontally interpolated new samples with a 6-bit phase accuracy. VPD acts equivalent to a polyphase filter with 64 possible phases. The functionality of the fine scaler is defined by the luminance and chrominance scaling parameters, XSCY[12:0]A9H[4:0]A8H[7:0] and XSCC[12:0]ADH[4:0]ACH[7:0], respectively, which are defined by the following equations:

XSCY[12:0] = 1024 x (N_pix_in/XPSC) x (1/Npix_out)

XSCC[12:0] = XSCY[12:0] / 2

The fine scaler in combination with the prescaler makes it possible to get very accurate samples from a highly anti-aliased integer down-scaled input image.
The functionality of the overall horizontal scaling block is characterized by a primary scaling factor, the H_scale ratio.

H_scale ratio = output pixel/input pixel

As the horizontal scaling block is comprised of two main scalers, the H-scale ratio is accordingly split into a binary and rational value defined by the following:

H_scale ratio = (1/XPSC[5:0]) x (1024/XSCY[12:0])

Various H_scale ratio values will be generated by our C program, instructing the decoder to horizontally compress the captured image.

The second approach to horizontally compressing the captured image involves reading the image via the S-video port found on the XSB board, storing image information in RAM, and implementing the scaling operation using hardware, i.e. using VHDL. This hardware implementation will lie to the frame buffer about the VGA frame size,

producing an image with horizontal dimensions smaller than that of the actual VGA. Functionality is characterized by the following ratio:

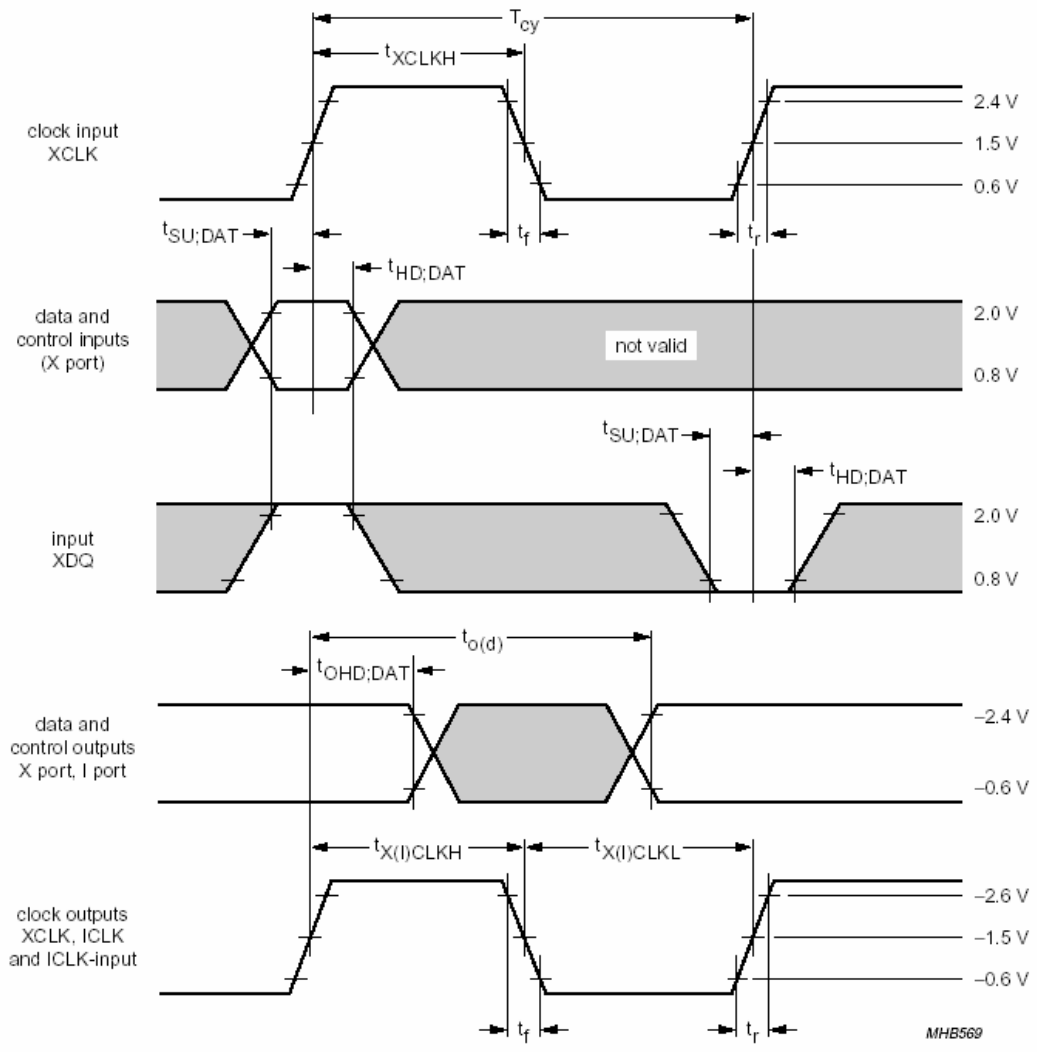
Scale ratio = # of desired horizontal pixels / actual horizontal pixel size of VGA monitor

Video Decoder Input/Output Interfaces and Ports

Digital image information is transferred to and from the Philips SAA7114H Video Decoder via the digital image port (i-port) of the chip. The I-port consists of the pins and signals as listed in the following table:

SYMBOL	PIN	I/O	DESCRIPTION	BIT
IPD7 to IPD0	54 to 57 and 59 to 62	I/O	I-port data	ICODE[93H[7]], ISWP[1:0]85H[7:6] and IPE[1:0]87[1:0]
ICLK	45	I/O	continuous reference clock at image port, can be input or output, as output decoder LLC or XCLK from X-port	ICKS[1:0]80H[1:0] and IPE[1:0]87H[1:0]
IDQ	46	O	data valid flag at image port, qualifier, with programmable polarity; secondary function: gated clock	ICKS2[80H[2]], IDQP[85H[0]] and IPE[1:0]87H[1:0]
IGPH	53	O	horizontal reference output signal, copy of the H-gate signal of the scaler, with programmable polarity; alternative functions: HRESET pulse	IDH[1:0]84H[1:0], IRHP[85H[1]] and IPE[1:0]87H[1:0]
IGPV	52	O	vertical reference output signal, copy of the V-gate signal of the scaler, with programmable polarity; alternative functions: VRESET pulse	IDV[1:0]84H[3:2], IRVP[85H[2]] and IPE[1:0]87H[1:0]
IGP1	49	O	general purpose output signal for I-port	IDG12[86H[4]], IDG1[1:0]84H[5:4], IG1P[85H[3]] and IPE[1:0]87H[1:0]
IGP0	48	O	general purpose output signal for I-port	IDG02[86H[5]], IDG0[1:0]84H[7:6], IG0P[85H[4]] and IPE[1:0]87H[1:0]
ITRDY	42	I	target ready input signals	–
ITRI	47	I	port control, switches I-port into 3-state	IPE[1:0]87H[1:0]

The I-port transfers data from the scaler. The data formats at the I-port are defined in Dwords of 32 bits (4 bytes), but the physical data stream at the image port is only 16-bit or 8-bit wide. In 16-bit mode, the pins HPD7 to HPD0 are used for chrominance data. For handshake with the VGA controller, F, H, and V reference signals and programmable FIFO flags are provided on pins IGP0, IGP1, IGPH, and IGPV. You will find below the Data input/output timing diagram for the Image Port (I-port).


T_{cy}
t_{XCLKH}
clock input
XCLK
2.4 V
1.5 V
0.6 V
$t_{SU;DAT}$
t_f
t_r
$t_{HD;DAT}$
data and
control inputs
(X port)
2.0 V
not valid
0.8 V
$t_{SU;DAT}$
$t_{HD;DAT}$
input
XDQ
2.0 V
0.8 V
$t_{o(d)}$
$t_{OHD;DAT}$
data and
control outputs
X port, I port
−2.4 V
−0.6 V
$t_{X(I)CLKH}$
$t_{X(I)CLKL}$
clock outputs
XCLK, ICLK
and ICLK-input
−2.6 V
−1.5 V
−0.6 V
t_f
t_r
MHB569

V. Image Display

There are three signals -- red, green, and blue -- that send color information to a VGA monitor. These three signals each drive an electron gun that emits electrons which paint one primary color at a point on the monitor screen. Each analog color input can be set to one of four levels by two digital outputs using a simple two-bit digital-to-analog converter.

We will use a VHDL program to send pixels to the monitor with the correct timing and framing, similar to ideas presented in Lab 5. We will store a picture in the RAM of the XSB Board, and then we can retrieve the data from the RAM, format it into lines of pixels, and send the lines to the monitor with the appropriate pulses on the horizontal and vertical sync pulses. The VHDL code will have statements to get the next byte from the RAM. Each byte contains four two bit pixels. A small loop iteratively extracts each pixel to be displayed from the lower two bits of the byte. Then the byte is shifted by two bits so the next pixel will be in the right position during the next iteration of the loop. Since it has only two bits, each pixel can store one of four colors.8

We have to make sure that we account for delays in accessing data from the RAM. The figure below has three stages:

Stage 1: The circuit uses the horizontal and vertical counters to compute the address where the next pixel is found in RAM. The counters are also used to determine the firing of the sync pulses and whether the video should be blanked. The pixel data from the RAM, blanking signal, and sync pulses are latched at the end of this stage so they can be used in the next stage.

Stage 2: The circuit uses the pixel data and the blanking signal to determine the binary color outputs. These outputs are latched at the end of this stage.

8 http://www.xess.com/appnotes/vga.pdf

Stage 3: The binary color outputs are applied to the DAC which sets the intensity levels for the monitor's color guns. The actual pixel is painted on the screen during this stage.

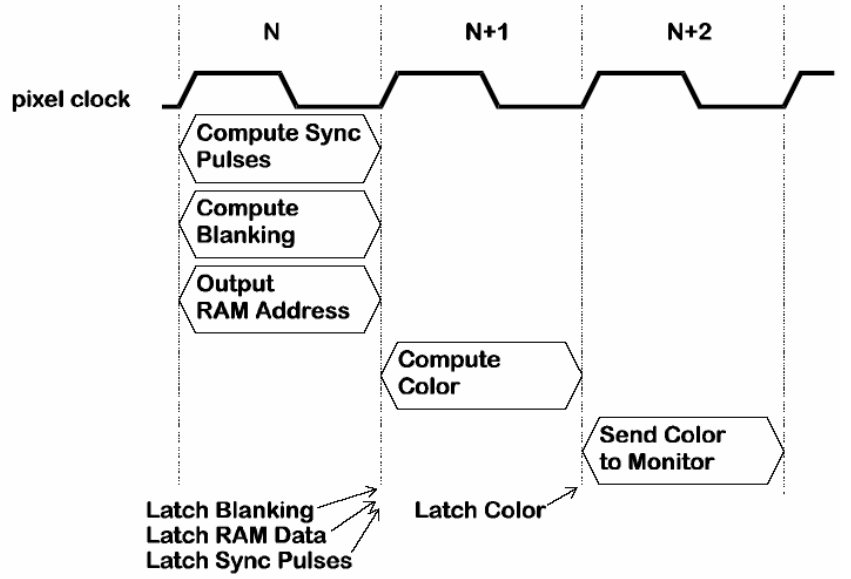


Figure 3: Pipelining of VGA signal generation tasks. [9]

Once we have displayed the image on the screen we want to start compressing the image horizontally. We will do this by using the horizontal scaling feature in the Video Decoder. We will compress the image until we have a vertical line on the screen. Once we have done this we will have to deal with inverting the image (creating a mirror image) and expanding it back to the original size.

Inverted Expansion

The expansion will work in a fashion similar to that of the horizontal compression using the video decoder. For the first method of horizontal compression, in which a C program is used to send various scaling factors to the horizontal scaling feature of the video decoder, the image can be expanded simply by making an amendment to the C file. We

9 http://www.xess.com/appnotes/vga.pdf

will add a function to the program that reverses the order in which the scaling factors are sent into the horizontal scaling block of the Philips chip. The output will be an interpolation of the input, which will expand the image horizontally. As for inverting the image, the pixel value can be sent to the screen starting from the right instead of the left, resulting in the display of a mirror image. If we use the second method of horizontal compression, expanding the image and inverting it will be comparable to the implementation used in the first method. However instruction will be given to the VGA in VHDL.

Bibliography

1. Philips Semiconductor SAA7114H. Data Sheet. March 14, 2000 (pg. 4)
2. XSB Board V1.0 Manual. XESS Corporation August 20, 2003 (pg. 27)
3. XSB Board V1.0 Manual. XESS Corporation August 20, 2003 (pg. 30)
4. http://computer.howstuffworks.com/monitor4.htm
5. http://animalscience.ucdavis.edu/Intranet/Graphics/PictureFormat/default.htm
6. http://www.csc.fi/visualization/viikki/formats.html
7. Information presented in this section has been obtained from the Philips Semiconductor SAA7114H. Data Sheet. March 14, 2000 (pgs. 42-47)
8. http://www.xess.com/appnotes/vga.pdf
9. http://www.xess.com/appnotes/vga.pdf