XGA to WVGA Nearest Neighbor Image Scaler

April 2013 Reference Design RD1160

Introduction

Image scaling is the process of resizing a digital image. Scaling is a non-trivial process that involves a trade-off between efficiency, smoothness and sharpness. As the size of the image is increased, the pixels which comprise the image become increasingly visible, making the image appear “soft”. Conversely, reducing an image will tend to enhance its smoothness and sharpness.

This design document illustrates the implementation of a video image downscaler. The downscaling algorithm used is the nearest neighbor algorithm.

In computing the value of the downscaled or destination pixel, the nearest neighbor algorithm applied in this design either adopts the value of the nearest point or the average of two points in the source pixels grid for its value.

The design is implemented in VHDL. The Lattice iCEcube2™ Place and Route tool integrated with the Synopsys Synplify Pro® synthesis tool is used for the implementation of the design. The design can be targeted to other iCE40™ FPGA product family devices.

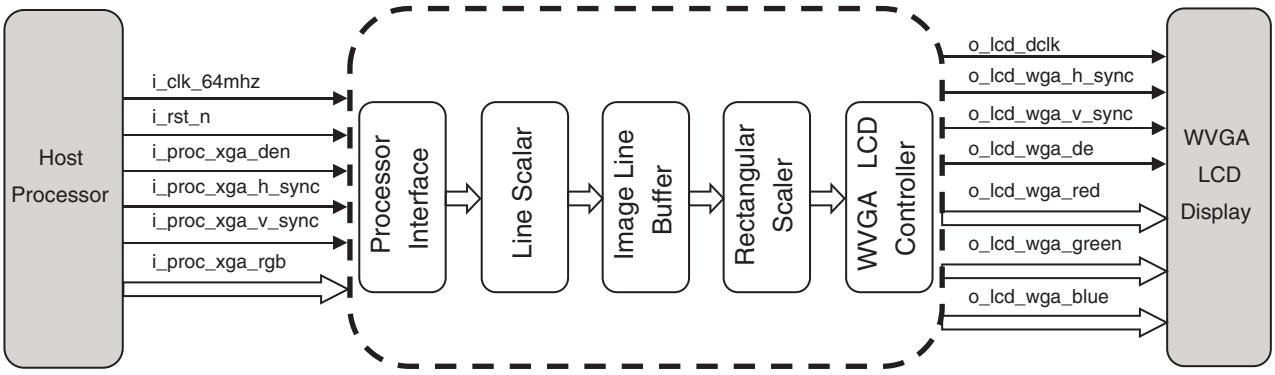
Figure 1 shows the System Block Diagram for the Image Scaler.

Features

- Nearest neighbor XGA (1024x768) to WVGA (800x480) downscaling
- Uses iCE40 RAM Blocks as image line buffers
- No external frame buffers required
- Supports color component average input instead of RGB565 pixel average input
- Supports RGB565 output
- IP-XACT version 1.2 compliant

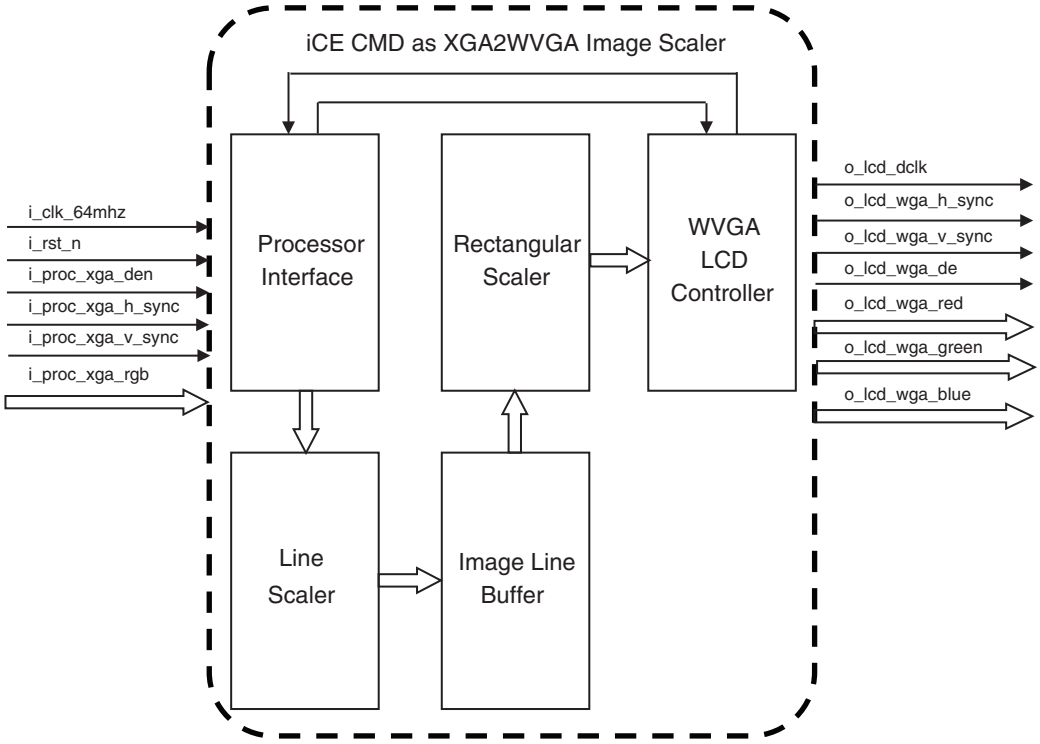
System Block Diagram

Figure 1. System Block Diagram



Functional Block Diagram

Figure 2. Functional Block Diagram



Processor Interface Module

A parallel Processor Interface is implemented between the processor and the Image Scaler. This module bridges the host device and the Image Scaler module. The host device provides the XGA video/image data to the Processor Interface. This interface also passes on the three data and timing control signals such as Data Enable (de), Horizontal Sync (h_sync), and Vertical Sync (v_sync) signals to the scaler. The h_sync signal determines the start and the end of a horizontal pixel line and the v_sync signal determines the end of a frame. The Processor Interface extracts incoming active pixels from the RGB data channels and transfers them to the Image Scaler.

Line Scaler Module

The valid XGA image data received by the processor interface is unconditionally passed through the Line Scaler module. The output of the Line Scaler module is subsequently loaded into the image buffers. The function of the Line Scaler is to downscale the pixel line horizontally to the targeted WVGA resolution before storing it in the Image Line Buffer.

The algorithm of the Line Scaler is explained below:

The general equation of a line is y=a•x+b. If (x0,y0) is fixed at some arbitrary location and such that xi +1=xi+1, in other words, the algorithm dictates that it steps one pixel horizontally at each computation iteration then yi+1=yi+a. The value of ‘a’ typically has a fractional part, so another symbol is introduced to keep the accumulation of these fractional parts as illustrated below:

```
yi +1=yi+int(a)

Di +1=Di+frac(a)
```

The use of floating point arithmetic can be avoided by using a scaled integer for frac(a). For example, if a= Dy/ Dx, where Dx and Dy are integers (and assuming Dx>Dy), you can replace D by E=Dx•D and state Ei +1=Ei+Dy. The "overflow rule" must be scaled accordingly.

These reformulations of polynomials are widely used to draw lines, circles and ellipses. Here, the same algorithm is used in resampling (scaling an image down). The algorithm sets each destination pixel to either adopt the value of the closest pixel or the unweighted average of the two neighboring pixels. The criterion to either select the replicated or the average pixel value is based on the position of the destination pixel relative to the grid of the source pixels. If the destination pixel is on top of a source pixel, or close to one, that source pixel is replicated. If, on the other hand, the destination pixel is closer to the midpoint between two source pixels, those two source pixels are averaged.

Below is the code snippet of the line scaling algorithm. There are two criteria for the accumulated error: when it exceeds the midpoint, it must start calculating the average between two neighboring pixels; when it exceeds unity, the source position must be adjusted.

```
ScaleLineAvg(PIXEL *Target, PIXEL *Source, int SrcWidth, int TgtWidth, float thresh-
old)
{
    int NumPixels = TgtWidth;
    int IntPart = SrcWidth / TgtWidth;
    int FractPart = SrcWidth % TgtWidth;
    int Mid = TgtWidth * threshold;
    int E = 0;
    int skip;
    PIXEL p;
    skip = (TgtWidth < SrcWidth) ? 0 : TgtWidth / (2*SrcWidth) + 1;
    NumPixels -= skip;
    //go through the entire line
    while (NumPixels-- > 0) {
    p = *Source;
    if (E >= Mid)
    p = average(p, *(Source+1));
    *Target++ = p;
    Source += IntPart;
    E += FractPart;
    if (E >= TgtWidth) {
    E -= TgtWidth;
    Source++;
      }
    }
    while (skip-- > 0)
    *Target++ = *Source;
    }
```

The flow chart of the line scaling algorithm is shown in the Figure 3.

Figure 3. Line Scaling Flow Chart

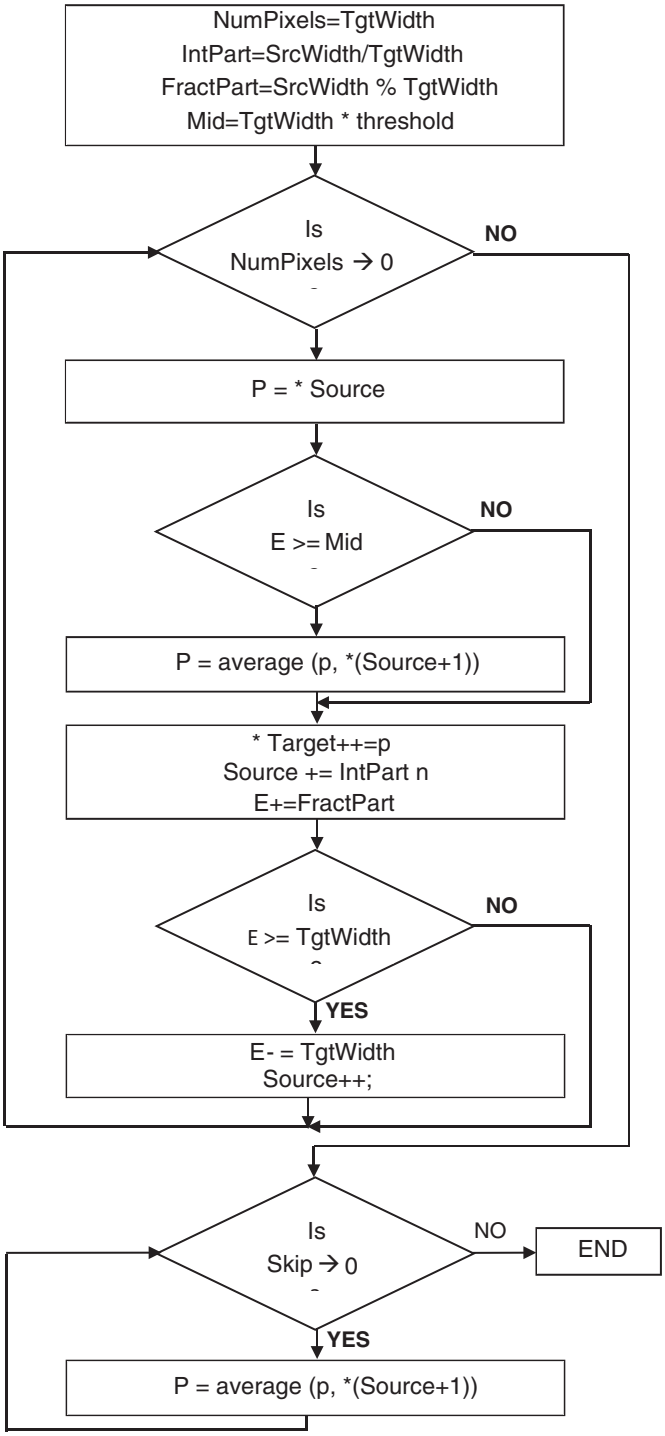


Image Line Buffer Module

The image buffers are used to temporarily store the downscaled pixel line data for the rectangular scaling later. When the three line buffers are completely filled with three downscaled lines, the rectangular scaler starts accessing the data from these line buffers for rectangular scaling. At any time, at least two lines are available in these buffers. Figure 4 and Figure 5 illustrate the read and write state diagram of the image line buffers.

Figure 4. WRITE to Image Line Buffers FSM

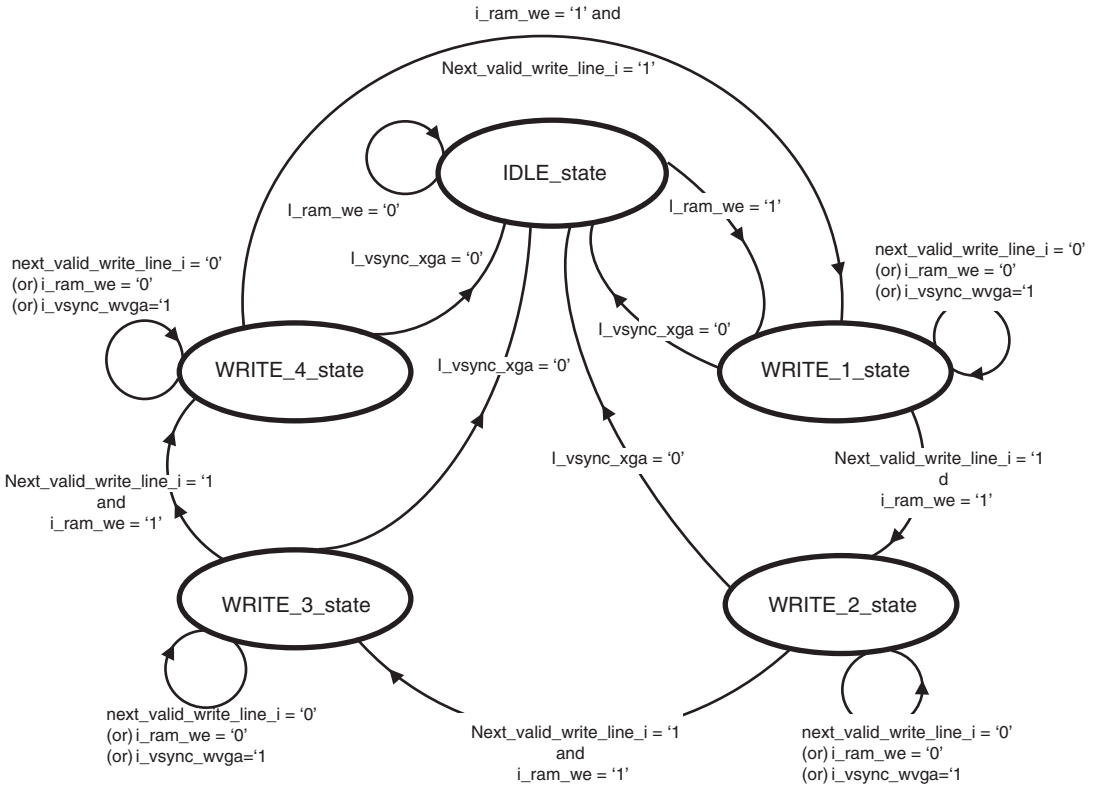
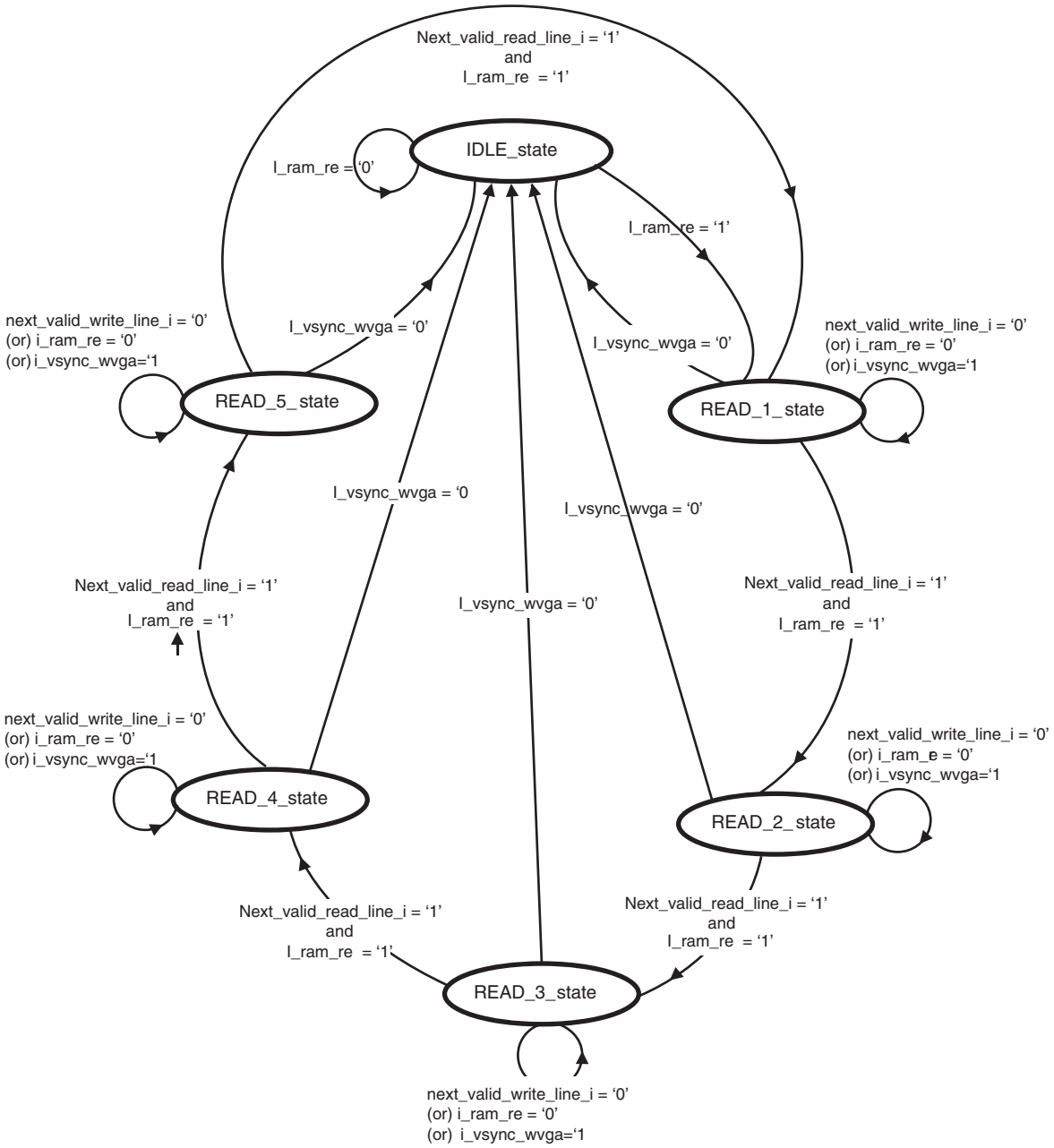


Figure 5. READ to Image Line Buffers FSM



Rectangular Scaler Module

The Rectangular Scaler is a combination of horizontal and vertical scaling in one single pass. The downscaled pixel lines accessed from the image line buffers are vertically scaled to the targeted height. Below is the code snippet of rectangular scaling algorithm.

```
ScaleRectAvg(PIXEL *Target, PIXEL *Source, int SrcWidth, int SrcHeight, int Tgt-
Width, int TgtHeight, float threshold)
{
    int NumPixels = TgtHeight;
    int IntPart = (SrcHeight / TgtHeight) * SrcWidth;
    int FractPart = SrcHeight % TgtHeight;
    int Mid = TgtHeight * threshold;
    int E = 0;
    int skip;
    PIXEL *ScanLine, *ScanLineAhead;
    PIXEL *PrevSource = NULL;
    PIXEL *PrevSourceAhead = NULL;
    skip = (TgtHeight < SrcHeight) ? 0 : TgtHeight / (2*SrcHeight) + 1;
    NumPixels -= skip;
    ScanLine = (PIXEL *)malloc(TgtWidth*sizeof(PIXEL));
    ScanLineAhead = (PIXEL *)malloc(TgtWidth*sizeof(PIXEL));
    while (NumPixels-- > 0) {
    if (Source != PrevSource) {
    if (Source == PrevSourceAhead) {
    PIXEL *tmp = ScanLine;
    ScanLine = ScanLineAhead;
    ScanLineAhead = tmp;
    } else {
    ScaleLineAvg(ScanLine, Source, SrcWidth, TgtWidth, threshold);
    }
    PrevSource = Source;
    }
    if (E >= Mid && PrevSourceAhead != Source+SrcWidth) {
    int x;
    ScaleLineAvg(ScanLineAhead, Source+SrcWidth, SrcWidth, TgtWidth, threshold);
    for (x = 0; x < TgtWidth; x++)
    ScanLine[x] = average(ScanLine[x], ScanLineAhead[x]);
    PrevSourceAhead = Source + SrcWidth;
    }
    memcpy(Target, ScanLine, TgtWidth*sizeof(PIXEL));
    Target += TgtWidth;
    Source += IntPart;
    E += FractPart;
    if (E >= TgtHeight) {
    E -= TgtHeight;
    Source += SrcWidth;
    }
    }
    if (skip > 0 && Source != PrevSource)
    ScaleLineAvg(ScanLine, Source, SrcWidth, TgtWidth, threshold);
    while (skip-- > 0) {
    memcpy(Target, ScanLine, TgtWidth*sizeof(PIXEL));
    Target += TgtWidth;
    }
    free(ScanLine);
    free(ScanLineAhead);
}
```

LCD Controller Module

The LCD Controller module generates the necessary timing and data control signals to drive a WVGA LCD display. During operation, the Processor Interface module provides a synchronizing signal to the LCD Controller module to lock the start of frame. Essentially, this module uses the required back porch and front porch timing of WVGA frame to generate the h_sync, v_sync and de signals to the WVGA LCD display.

The timing and data control signals and data with respect to the pixel clock are shown in Figure 6.

Signal Description

Table 1. Signal Description

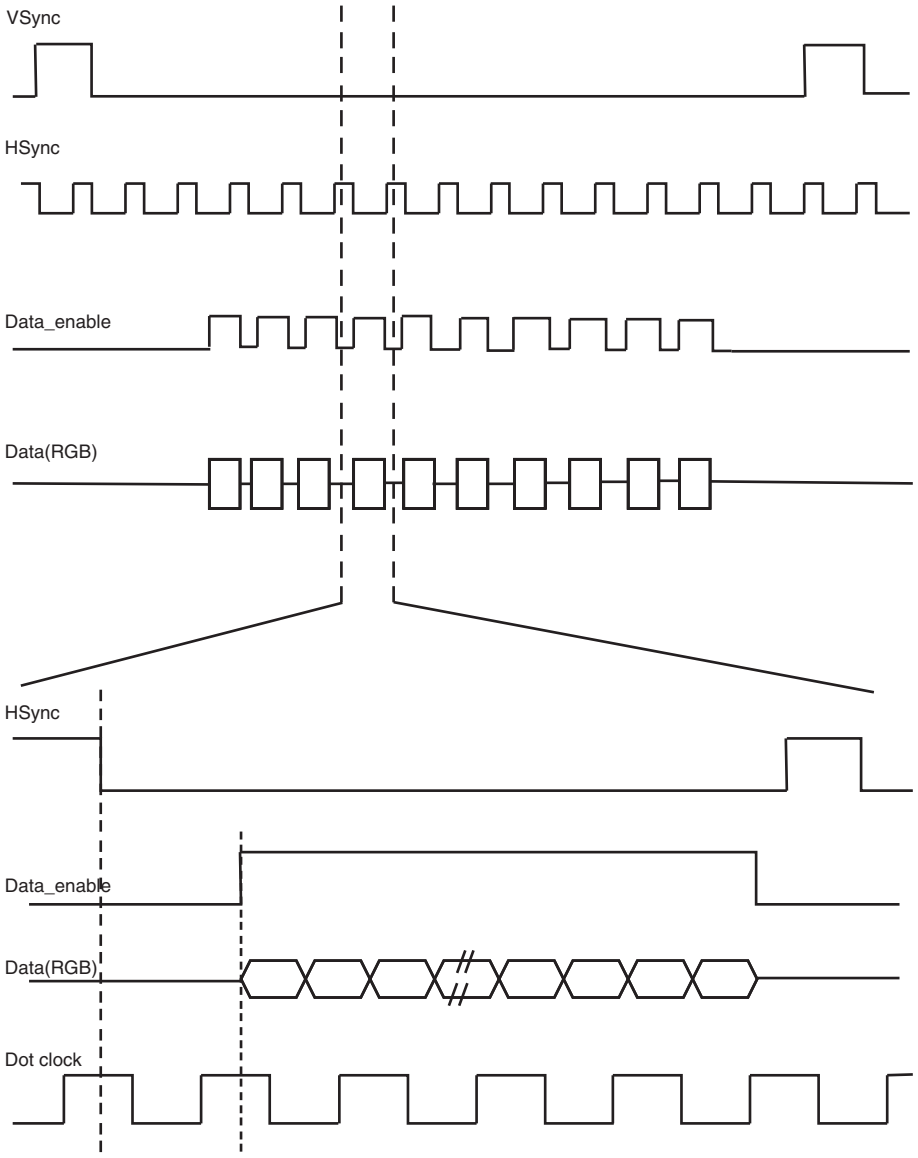
Signal	Width	Type	Description
i_clk_64mhz	1	Input	Processor Clock
i_rst_n	1	Input	System reset (active low)
i_proc_xga_h_sync	1	Input	XGA horizontal synchronization signal
i_proc_xga_v_sync	1	Input	XGA vertical synchronization signal
i_proc_xga_de	1	Input	XGA de signal
i_proc_xga_rgb	16	Input	XGA RGB data
o_lcd_dclk	1	Output	WVGA LCD clock
o_lcd_wvga_h_sync	1	Output	WVGA horizontal synchronization signal
o_lcd_wvga_v_sync	1	Output	WVGA vertical synchronization signal
o_lcd_wvga_de	1	Output	WVGA de signal
o_lcd_wvga_red	5	Output	WVGA LCD red data
o_lcd_wvga_green	6	Output	WVGA LCD green data
o_lcd_wvga_blue	5	Output	WVGA LCD blue data

Initialization Conditions

An active low reset signal assertion is required to initialize the scalers and LCD controller state machines to a known operating state. No register configuration is necessary.

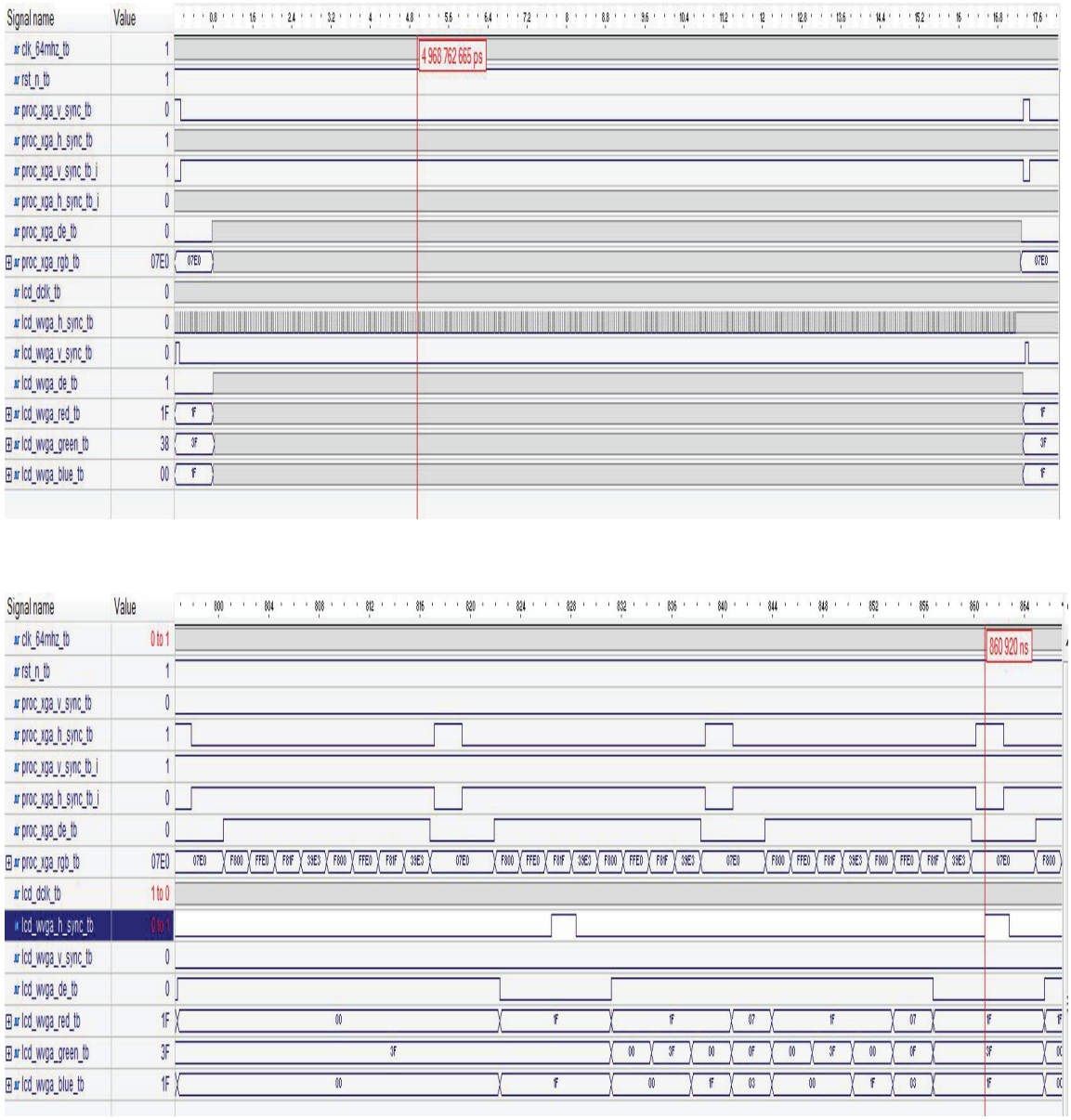
Timing Diagram

Figure 6. Timing Diagram



Simulation Waveforms

Figure 7. Simulation Waveforms



Operation Sequence

1. Testbench will have a wrapper instantiated where the wrapper will in turn instantiate the design top level.
2. It will generate clock and reset signals to the design.
3. It has counters for counting pixels, lines, and pulse width period of HSync and VSync.
4. It generates XGA control signals like HSync, VSync and DE.
5. It feeds XGA data to the image scaler design during the active region of the XGA image.
6. The output red, green and blue data are fed to a .tiff image writer.
7. A module tiff.v will take the WVGA pixel data in and add a header to make it a .tiff image.
8. Current simulation directory provides the resulting WVGA image. This can be viewed by using any image viewing software.

Implementation

This design is implemented in VHDL. When using this design in a different device, density, speed or grade, performance and utilization may vary.

Table 2. Performance and Resource Utilization

Family	Language	Utilization (LUTs)	f_{MAX} (MHz)	I/Os
iCE40[1]	VHDL	740	>50	40

1. Performance and utilization characteristics are generated using iCE40HX8K-CM225 with iCEcube2 design software.

References

- iCE40 Family Handbook

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
April 2013	01.0	Initial release.