

iCE65 as a XGA to WVGA Image Scaler using Nearest Neighbor Algorithm

Overview

This design document illustrates the implementation of Nearest Neighbor XGA to WVGA scaling algorithm using low power iCE65 FPGAs. 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.

Features supported

- Nearest neighbor XGA (1024x768) to WVGA (800x480) downscaling
- Uses iCE65 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

Resource Utilization

Table 1: Resource Utilization

LUTs	Registers	Memory	I/Os	GBs
688	325	16	0	0

Note: Resource Utilization is based on iCEcube 2010.12.14671 release.

System Block Diagram

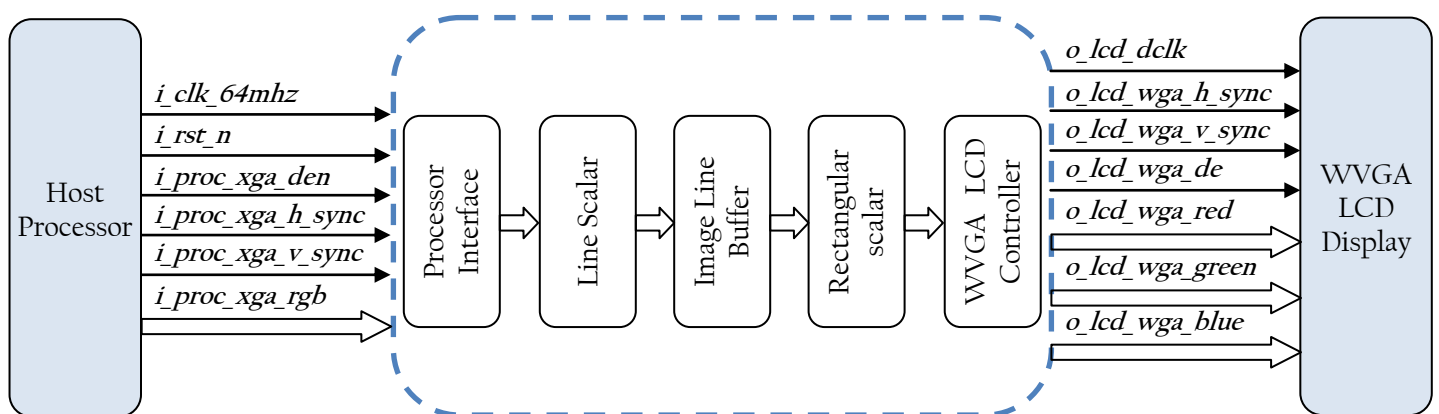


Figure 1: System Block Diagram

Functional Block Diagram

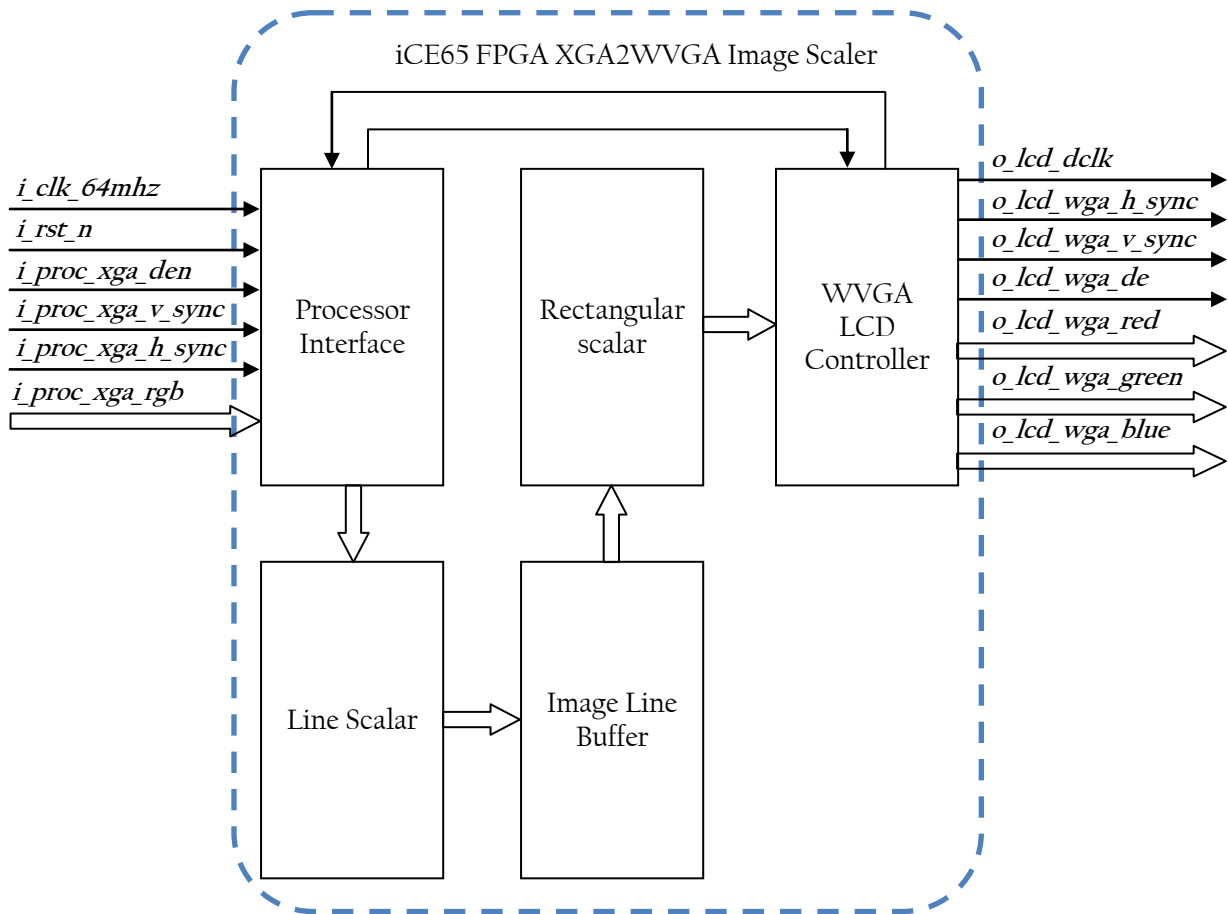


Figure 2: Functional Block Diagram

Design Interface

Table 2 lists the pin interface details for host CPU. Signal direction references are with respect to the FPGA device.

Table 2: Processor Interface Signals(Total 21 pins)

Signal Name	Pin Type	Signal Description
i_clk_64mhz	Input	Processor Clock
i_rst_n	Input	System reset (active low)
i_proc_xga_h_sync	Input	XGA horizontal synchronization signal
i_proc_xga_v_sync	Input	XGA vertical synchronization signal
o_proc_xga_de	Input	XGA de signal
i_proc_xga_rgb[0]	Input	XGA RGB data(0)
i_proc_xga_rgb[1]	Input	XGA RGB data(1)
i_proc_xga_rgb[2]	Input	XGA RGB data(2)
i_proc_xga_rgb[3]	Input	XGA RGB data(3)
i_proc_xga_rgb[4]	Input	XGA RGB data(4)
i_proc_xga_rgb[5]	Input	XGA RGB data(5)
i_proc_xga_rgb[6]	Input	XGA RGB data(6)
i_proc_xga_rgb[7]	Input	XGA RGB data(7)
i_proc_xga_rgb[8]	Input	XGA RGB data(8)
i_proc_xga_rgb[9]	Input	XGA RGB data(9)
i_proc_xga_rgb[10]	Input	XGA RGB data(10)
i_proc_xga_rgb[11]	Input	XGA RGB data(11)
i_proc_xga_rgb[12]	Input	XGA RGB data(12)
i_proc_xga_rgb[13]	Input	XGA RGB data(13)
i_proc_xga_rgb[14]	Input	XGA RGB data(14)
i_proc_xga_rgb[15]	Input	XGA RGB data(15)

Table 3 lists the WVGA LCD interface pins for the FPGA. Signal direction references are with respect to the FPGA device.

Table 3: WVGA LCD interface signals(Total 20 pins)

Signal Name	Pin Type	Signal Description
o_lcd_dclk	Output	WVGA LCD clock
o_lcd_wvga_h_sync	Output	WVGA horizontal synchronization signal
o_lcd_wvga_v_sync	Output	WVGA vertical synchronization signal
o_lcd_wvga_de	Output	WVGA de signal
o_lcd_wvga_red[0]	Output	WVGA LCD red data(0)
o_lcd_wvga_red[1]	Output	WVGA LCD red data(1)
o_lcd_wvga_red[2]	Output	WVGA LCD red data(2)
o_lcd_wvga_red[3]	Output	WVGA LCD red data(3)
o_lcd_wvga_red[4]	Output	WVGA LCD red data(4)
o_lcd_wvga_green[0]	Output	WVGA LCD green data(0)
o_lcd_wvga_green[1]	Output	WVGA LCD green data(1)
o_lcd_wvga_green[2]	Output	WVGA LCD green data(2)
o_lcd_wvga_green[3]	Output	WVGA LCD green data(3)
o_lcd_wvga_green[4]	Output	WVGA LCD green data(4)
o_lcd_wvga_green[5]	Output	WVGA LCD green data(5)
o_lcd_wvga_blue[0]	Output	WVGA LCD blue data(0)
o_lcd_wvga_blue[1]	Output	WVGA LCD blue data(1)
o_lcd_wvga_blue[2]	Output	WVGA LCD blue data(2)
o_lcd_wvga_blue[3]	Output	WVGA LCD blue data(3)
o_lcd_wvga_blue[4]	Output	WVGA LCD blue data(4)

Configurable Parameters

None

Register Map

This design does not have any user accessible registers or memory..

Design Details

Processor Interface

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 line buffers.

The algorithm of the line scaler is explained as below :

The general equation of a line is $y=ax+b$. If (x_0,y_0) is fixed at some arbitrary location and such that $x_{i+1}=x_i+1$, in other words, the algorithm dictates that it steps one pixel horizontally at each computation iteration then $y_{i+1}=y_i+a$. 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:

$$y_{i+1}=y_i+\text{int}(a)$$

$$D_{i+1}=D_i+\text{frac}(a)$$

The use of floating point arithmetic can be avoided by using a scaled integer for $\text{frac}(a)$. For example, if $a= D_y/ D_x$, where D_x and D_y are integers (and assuming $D_x>D_y$), you could replace D by $E=D_x\cdot D$ and state $E_{i+1}=E_i+D_y$. 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 threshold)

```
{
    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

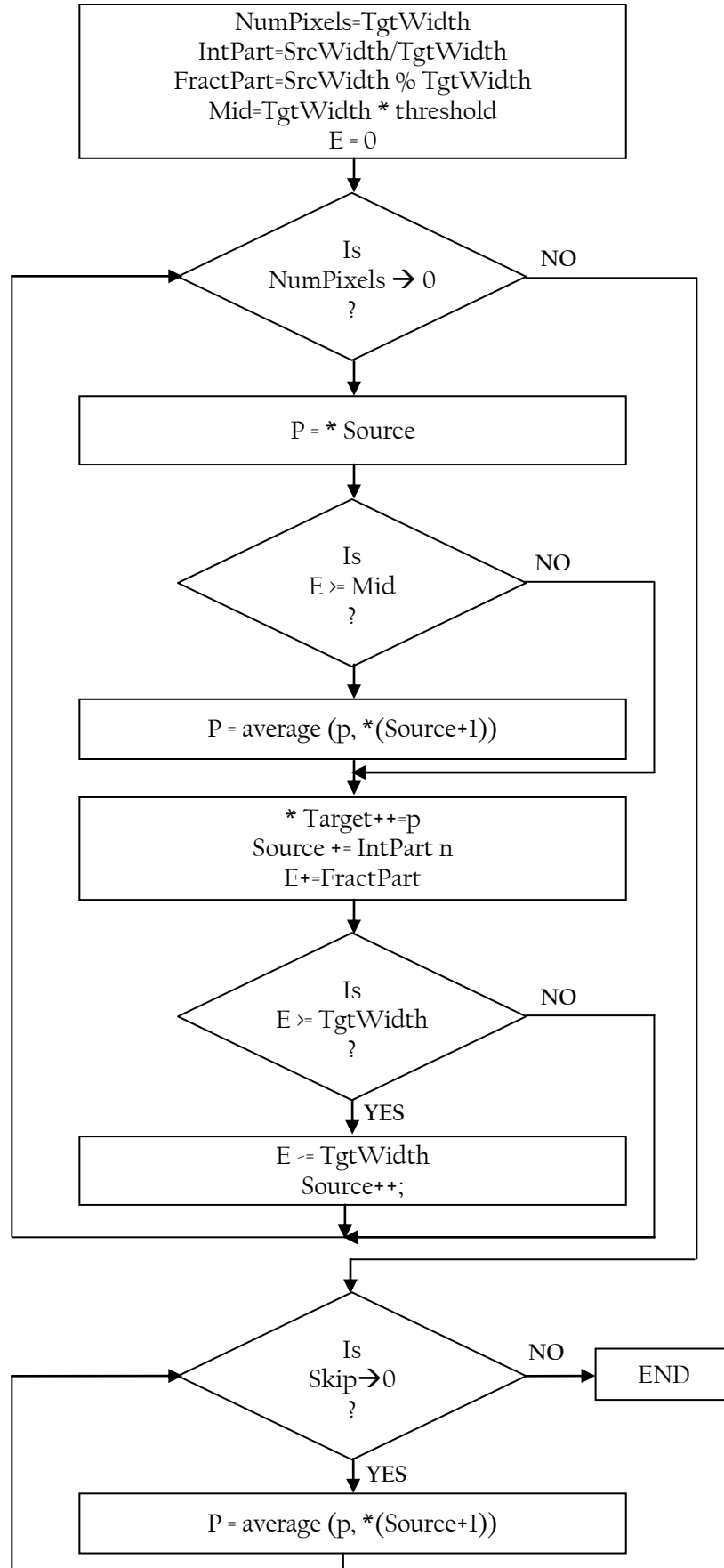


Figure 3: Line scaling flow chart

Image line buffer module

The image buffers are the buffers used to temporarily store the downscaled pixel line data for the 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.

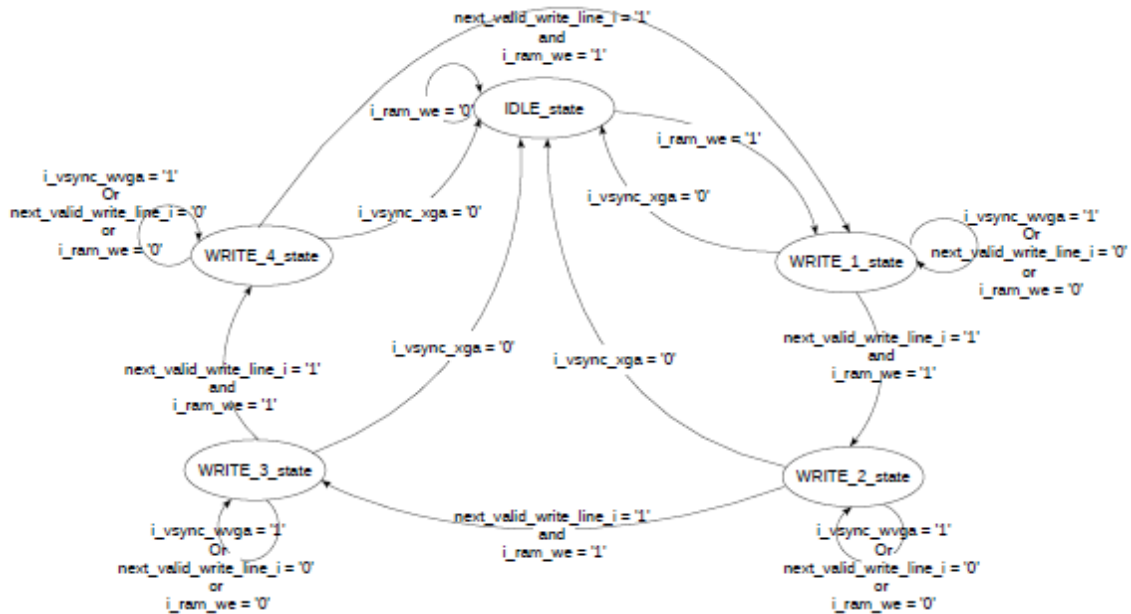


Figure 4: WRITE to Image line buffers FSM

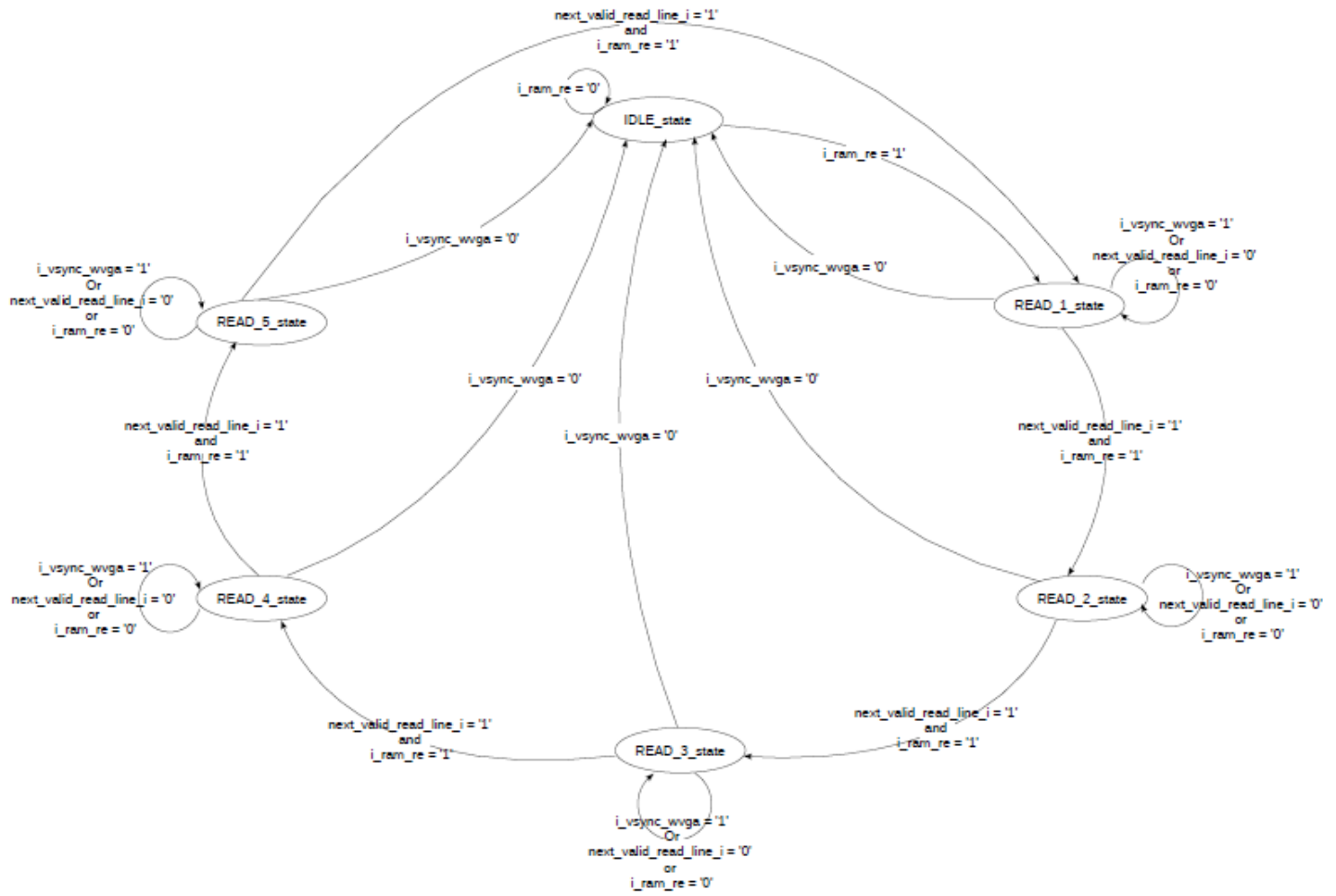


Figure 5: READ from image line buffers FSM

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 TgtWidth, 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

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.

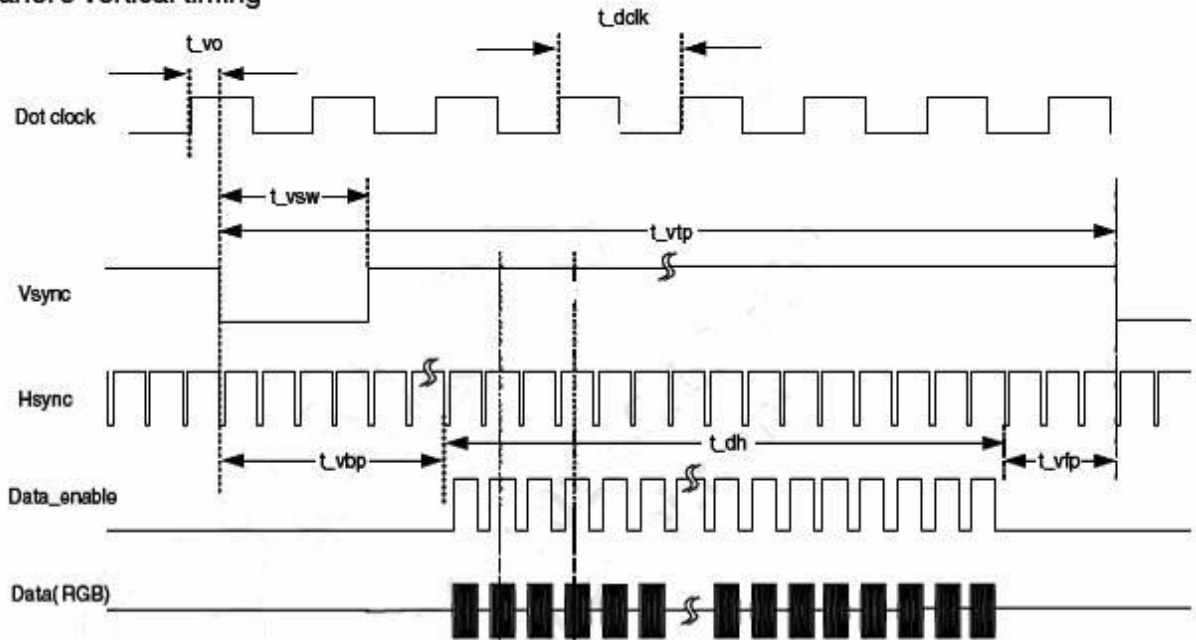
The timing and data control signals and data with respect to the pixel clock are shown in the Figure 6

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

TFT panel's vertical timing



TFT panel's horizontal timing

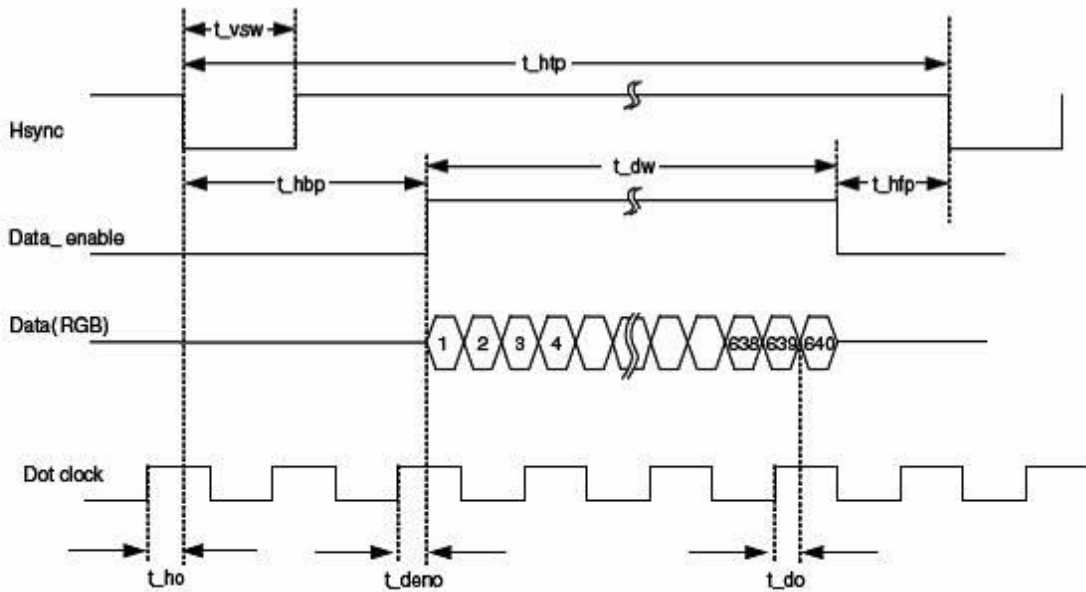


Figure 6: WVGA LCD Timing diagram

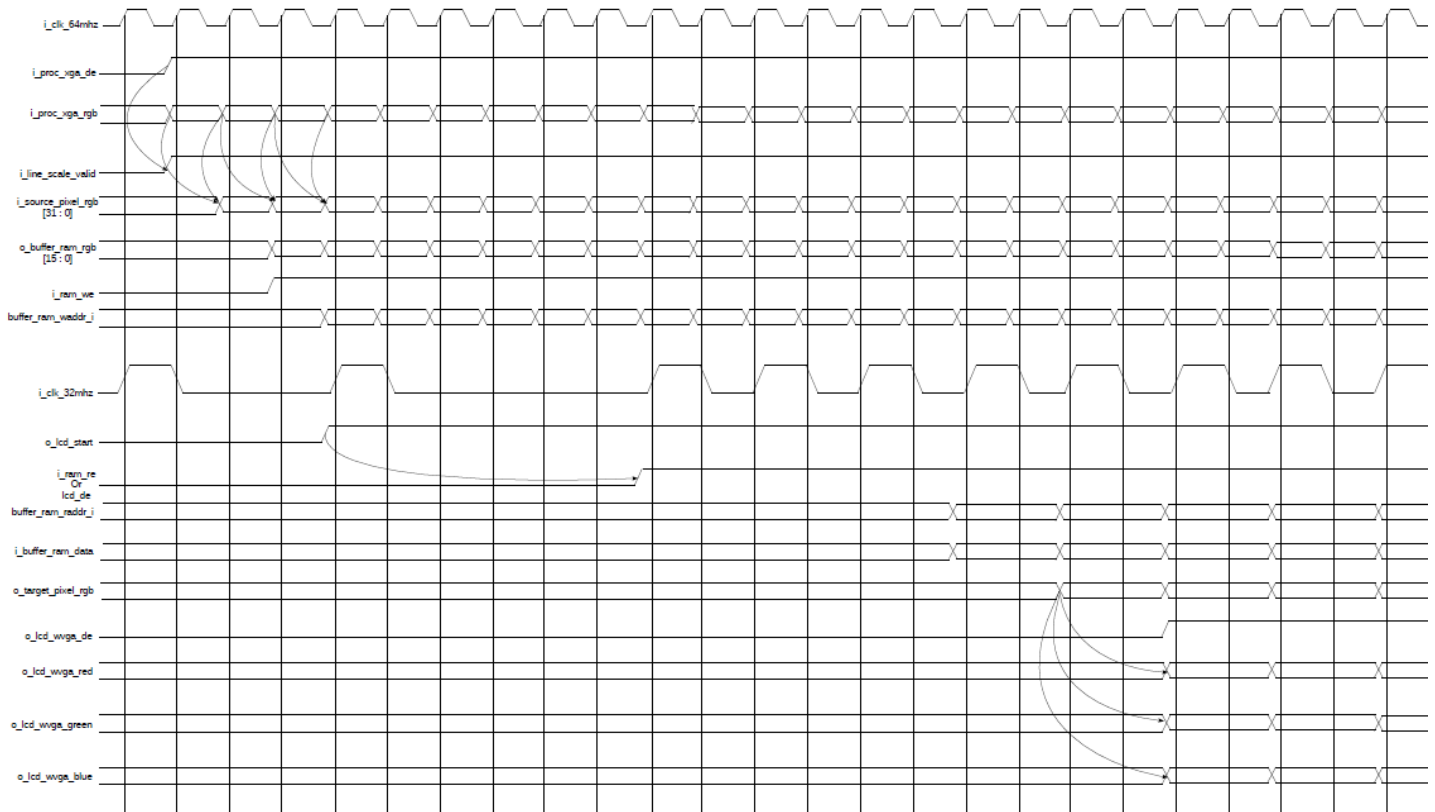


Figure 7: Timing diagram of communication between different modules in the design

System Designer Flow

XGA to WVGA Image Scaler using Nearest Neighbor algorithm is compatible with System Designer/IP-XACT 1.2. The System Designer flow is as follows,

1. Launch the System Designer from Synplify Pro using menu 'Import -> Launch System Designer'.
2. Create a new project(open an existing old project, as necessary) and import the IP-XACT XML file
3. Drag and place the component from the 'Library' pane to the 'Design' pane
4. Click on the “Generate Files” button, which generates the necessary files required for synthesis and simulation.
5. Go to Synplify Pro and click on the “Run” button to synthesize the System Designer generated files. Synplify Pro generates all the necessary files for P&R in iCECube.

References

The following references were used in the creation of this design:

- SiliconBlue Technologies, Inc. “[iCE65 Ultra Low-Power mobileFPGA Family](#)” datasheet (26-MAY-2010).
- Image Scaling: http://en.wikipedia.org/wiki/Image_scaling
- Bilinear Interpolation: http://en.wikipedia.org/wiki/Bilinear_interpolation
- "Image Scaling With Bresenham" <http://www.ddj.com/architect/184405045>

Revision History

Version	Date	Description
1.0	09-SEP-2010	Initial Draft Document
1.1	08-DEC-2010	IP-XACT Format Update

Disclaimer

Copyright © 2007–2009 by SiliconBlue Technologies LTD. All rights reserved. SiliconBlue is a registered trademark of SiliconBlue Technologies LTD in the United States. Specific device designations, and all other words and logos that are identified as trademarks are, unless noted otherwise, the trademarks of SiliconBlue Technologies LTD. All other product or service names are the property of their respective holders. SiliconBlue products are protected under numerous United States and foreign patents and pending applications, maskwork rights, and copyrights. SiliconBlue warrants performance of its semiconductor products to current specifications in accordance with SiliconBlue's standard warranty, but reserves the right to make changes to any products and services at any time without notice. SiliconBlue assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by SiliconBlue Technologies LTD. SiliconBlue customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



SiliconBlue Technologies Corporation

3255 Scott Blvd.,
Building 7, Suite 101
Santa Clara, CA 95054

Tel: 408-727-6101
Fax: 408-727-6085

www.SiliconBlueTech.com