

Key Design Features

- Synthesizable, technology independent VHDL Core
- Versatile RGB-video scaling engine capable of scaling up or down by any aspect ratio
- 24-bit accumulator with 24-bit scale-pitch in [24 12] format
- Supports all video resolutions between 16x16 and 4096x4096
- Fully pipelined architecture with simple valid-ready interface protocol. No complex control required
- Features a 2-tap polyphase filter in the x-dimension and a 2-tap polyphase filter in the y-dimension. Each filter has 16 unique phases or interpolation points
- Fully programmable filter coefficients to suit the desired application. Example coefficients shipped with the design
- Output 1 x 24-bit pixel per clock for scaling factors > 1

Applications

- High quality RGB video scaling
- Conversion of popular video formats to any aspect ratio such as VGA to XGA or SVGA to HD1080p.
- Digital TV set-top boxes and home media
- Notebook PC external video ports - e.g. to drive an external monitor or flat panel display
- Dynamic scaling of video in a window on a frame-by-frame basis
- Picture in Picture applications

Block Diagram

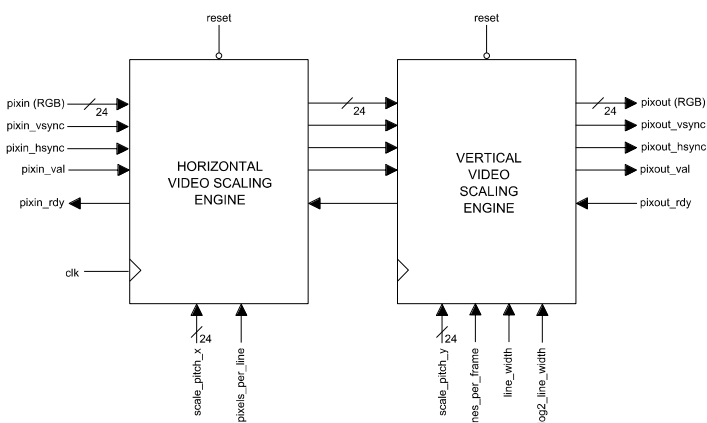


Figure 1: Bilinear Video Scaler Architecture

Generic Parameters

Generic name	Description	Type	Valid range
line_width	Width of linestores in pixels	integer	$2^4 < \text{pixels} < 2^{12}$
log2_line_width	Log2 of linestore width	integer	$\log_2(\text{line_width})$

Pin-out Description

Pin name	I/O	Description	Active state
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
pixin [23:0]	in	24-bit RGB pixel in	data
scale_pitch_x	in	1 / (x scale factor) Specified as an unsigned number in [24 12] format Range: $0 < \text{pitch} < 2^{24}$	data
scale_pitch_y	in	1 / (y scale factor) Specified as an unsigned number in [24 12] format Range: $0 < \text{pitch} < 2^{24}$	data
pixels_per_line	in	Desired number of pixels per line in the scaled output image Range: $2^4 < \text{lines} < 2^{12}$	data
lines_per_frame	in	Desired number of lines per frame in the scaled output image Range: $2^4 < \text{lines} < 2^{12}$	data
pixin_hsync	in	Horizontal sync in (signifies start of line)	high
pixin_val	in	Input pixel valid	high
pixin_rdy	out	Ready to accept input pixel (handshake signal)	high
pixout [23:0]	out	24-bit pixel out	data
pixout_hsync	out	Horizontal sync out	high
pixout_val	out	Output pixel valid	high
pixout_rdy	in	Ready to accept output pixel (handshake signal)	high

General Description

XY2_SCALER is a high quality RGB-video scaler capable of generating interpolated output images from 16x16 up to 4096x4096 pixels in resolution. The architecture permits seamless scaling (either up or down) depending on the chosen scale factor. Internally, the scaler uses a 24-bit accumulator and a bank of polyphase FIR filters with 16 phases or interpolation points. All filter coefficients are programmable, allowing the user to define a wide range of filter characteristics.

Pixels flow in and out of the scaling engine in accordance with the valid-ready pipeline protocol. Pixels are transferred into the scaler on a rising clock-edge when *pixin_val* is high and *pixin_rdy* is high. Likewise, pixels are transferred out of the scaler on a rising clock-edge when *pixout_val* is high and *pixout_rdy* is high. As such, the pipeline protocol allows both input and output interfaces to be stalled independently.

The scaler is partitioned into a horizontal scaling section in series with a vertical scaling section (Figure 1). Both components are available as separate cores. For a detailed explanation of the horizontal and vertical scaler sections, please refer to the relevant ZIPcores documentation.

Scale pitch, pixels per line and lines per frame

The output resolution of the scaled output image is controlled by the generic parameters *scale_pitch_x*, *scale_pitch_y*, *pixels_per_line* and *lines_per_frame*. The scale pitch may be calculated using the following formula:

$$pitch = (Input\ resolution / Output\ resolution) * 2^{12}$$

As an example, consider the scaling of VGA format video (640x480) to XGA format video (1024x768). In this case the scale pitch in the x and y dimensions would be 0.625. As the value must be specified as a 12.12-bit number the actual scale pitch must be multiplied by 2^{12} giving the generic value '2560'.

Ultimately, as the scale pitch is a quantized number, the number of generated output pixels and output lines for a given scale factor may differ slightly by the desired amount. For this reason, the parameter *pixels_per_line* and *lines_per_frame* allow the exact resolution of the output video to be specified. The following tables give a list of generic parameters required for the conversion of some example video formats.

SCALE UP

Video in	Video out	Scale pitch x	Scale pitch y	Pixels per line	Lines per frame
VGA (640x480)	SVGA (800x600)	3277	3277	800	600
SVGA (800x600)	XGA (1024x768)	3200	3200	1024	768
XGA (1024x768)	HD1080 (1920x1080)	2184	2913	1920	1080
SXGA (1280x1024)	2K (2048x1080)	2560	3884	2048	1080

SCALE DOWN

Video in	Video out	Scale pitch x	Scale pitch y	Pixels per line	Lines per frame
SVGA (800x600)	VGA (640x480)	5120	5120	640	480
XGA (1024x768)	SVGA (800x600)	5243	5243	800	600
HD1080 (1920x1080)	XGA (1024x768)	7680	5760	1024	768
2K (2048x1080)	SXGA (1280x1024)	6554	4320	1280	1024

Flow control

Pixels flow in and out of the scaling engine in accordance with the valid-ready pipeline protocol¹. The scaling operation occurs on a line-by-line basis with the signal *pixin_hsync* specifying the start of a new line and *pixin_vsync* specifying the start of a new frame. All pixels into the scaler (including *vsync* and *hsync*) must be qualified by the *pixin_val* signal asserted high, otherwise changes to the input signals will be ignored. Note that the first pixel of a new frame is accompanied by a valid *vsync* and *hsync*. The first pixel in a new line is accompanied by *hsync* only.

On receipt of the first *vsync*, the scaling operation begins and output pixels are generated in accordance with the chosen scale parameters. Generally, for scale-down (decimation) operations, the input interface will not stall. Conversely, for scale-up (interpolation) the number of output pixels will be greater than the number of input pixels. This will result in the occasional stalling of the input due to the change of data rate.

Loading of scale parameters

The input parameters *scale_pitch* and *lines_per_frame* are loaded into the scaler at the start of each new frame. The parameters are considered valid when *pixin_val* and *pixin_vsync* are both high. In all other cases, the state of the scale parameters is ignored. The fully programmable scale parameters allow the input video to be scaled differently on a frame-by-frame basis. Alternatively, different video sources can be multiplexed into the same scaler with different scaling parameters.

Scaling algorithm

The scaler uses a 2-tap polyphase filter in the x-dimension and a 2-tap polyphase filter in the y-dimension. By default, the x and y filters use bi-linear interpolation (Figure 2). Of course, the user is permitted to use any type of function to derive the filter coefficients depending on the application².

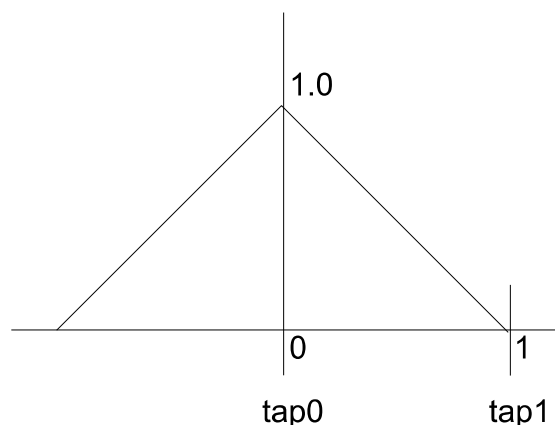


Figure 2: Bi-linear function – x and y filter tap positioning

- 1 See ZIPcores application note: [app_note_zc001.pdf](#) for more examples of how to use the valid-ready pipeline protocol
- 2 See ZIPcores application note: [app_note_zc003.pdf](#) for examples of how to generate coefficient sets

Functional Timing

Figure 4 shows the signalling at the input to the scaler at the start of a new frame. The first line of a new frame begins with *pixin_vsync* and *pixin_hsync* asserted high together with the first pixel. Note that the signals *pixin*, *pixin_vsync* and *pixin_hsync* are only valid if *pixin_val* is also asserted high. In addition, the diagram shows what happens when *pixin_rdy* is de-asserted. In this case, the pipeline is stalled and the upstream interface must hold-off before further pixels are processed.

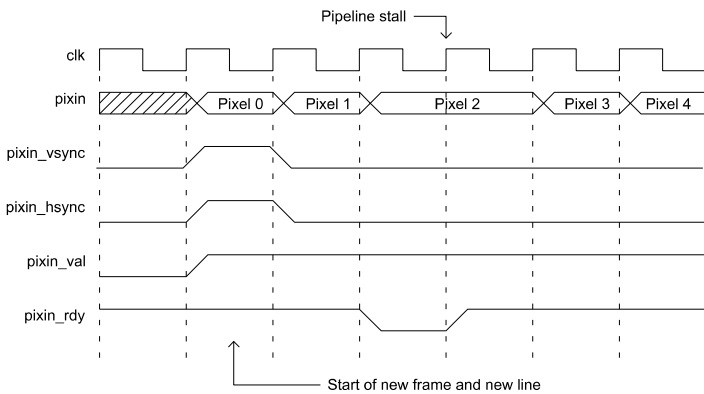


Figure 3: First line of a new frame

Figure 5 shows the signalling at the output of the scaler. The output uses exactly the same protocol as the input. Each new output line begins with *pixout_hsync* and *pixout_val* asserted high. In this particular example, it shows *pixout_val* de-asserted for 1 clock-cycle, in which case, the output pixel should be ignored. Remember that transfers at a valid-ready interface are only permitted when valid and ready are both simultaneously high.

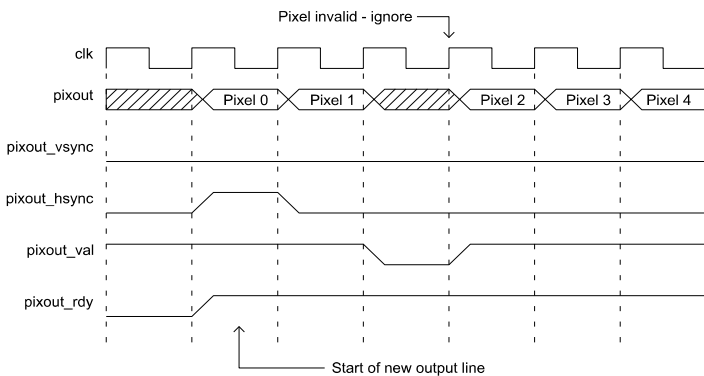


Figure 4: Scaler output showing invalid pixel

Source File Description

All source files are provided as text files coded in VHDL. The following table gives a brief description of each file.

Source file	Description
video_in.txt	Text-based source video file
video_file_reader.vhd	Reads text-based source video file
pipeline_reg.vhd	Pipelined register element
pipeline_shovel.vhd	Pipelined 'shovel' register
ram_dp_w_r.vhd	Dual port RAM component
fifo_sync.vhd	Synchronous FIFO
x2_buffer.vhd	Pixel input buffer/shift register
x2_filter_pack.vhd	Package containing x-filter coefficients
x2_filter_polyphase.vhd	Horizontal scaler output pixel filter
x2_scaler.vhd	Horizontal scaler component
y2_buffer.vhd	Line buffer
y2_filter_pack.vhd	Package containing y-filter coefficients
y2_filter_polyphase.vhd	Vertical scaler output pixel filter
y2_scaler.vhd	Vertical scaler component
xy2_scaler.vhd	Video scaler top-level component
xy2_scaler_bench.vhd	Top-level test bench

Functional Testing

An example VHDL testbench is provided for use in a suitable VHDL simulator. The compilation order of the source code is as follows:

1. video_file_reader.vhd
2. pipeline_reg.vhd
3. pipeline_shovel.vhd
4. ram_dp_w_r.vhd
5. fifo_sync.vhd
6. x2_buffer.vhd
7. x2_filter_pack.vhd
8. x2_filter_polyphase.vhd
9. x2_scaler.vhd
10. y2_buffer.vhd
11. y2_filter_pack.vhd
12. y2_filter_polyphase.vhd
13. y2_scaler.vhd
14. xy2_scaler.vhd
15. xy2_scaler_bench.vhd

The VHDL testbench instantiates the XY2_SCALER component and the user may modify the generic parameters in order to generate the desired scaled output image.

The source video for the simulation is generated by the video file-reader component. This component reads a text-based file which contains the RGB pixel data. The text file is called *video_in.txt* and should be placed in the top-level simulation directory.

The file *video_in.txt* follows a simple format which defines the state of signals: *pixin_val*, *pixin_vsync*, *pixin_hsync* and *pixin* on a clock-by-clock basis. An example file might be the following:

```
1 1 1 00 11 22 # pixel 0 line 0 (start of frame)
1 0 0 33 44 55 # pixel 1
0 0 0 00 00 00 # don't care!
1 0 0 66 77 88 # pixel 2
.
1 0 1 00 11 22 # pixel 0 line 1 etc..
```

In this example, the first line of of the *video_in.txt* file asserts the input signals *pixin_val* = 1, *pixin_vsync* = 1, *pixin_hsync* = 1 and *pixin* = 0x001122.

The simulation must be run for at least 10 ms during which time an output text file called *video_out.txt* will be generated. This file contains a sequential list of 24-bit output pixels in the same format as *video_in.txt*. The example provided scales a 640x480 (VGA) source image by a factor of 1.6 in the x and y dimensions to give an output image of 1024x768 pixels.

Performance

The Digital Video Scaler was tested with a large number of scale factors to verify correct operation and to observe the quality of the output image. The true definition and quality is difficult to show within the limitations of this document, but the following figures may give an idea of what is achievable.

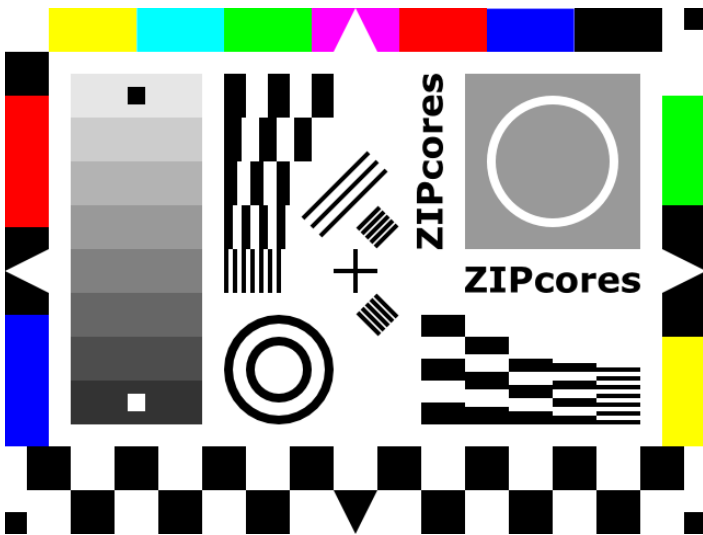


Figure 5: Original Testcard VGA Source image (640x480)

Figure 5 shows the original testcard source image at 640x480 (VGA) resolution. Figure 6 is a scale-down of this image to 512x384 pixels. For instance, this is equivalent to scaling down SVGA to VGA format.

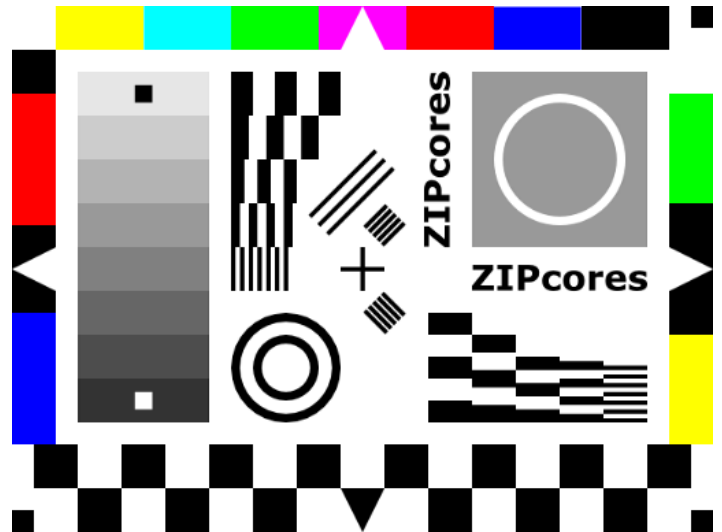


Figure 6: Scale-down by 0.8 - e.g. SVGA to VGA format

Figure 7 demonstrates a scale-up of the original source image by a factor of 1.28 in x and y. The video scaler is completely versatile and permits seamless scaling either up or down.

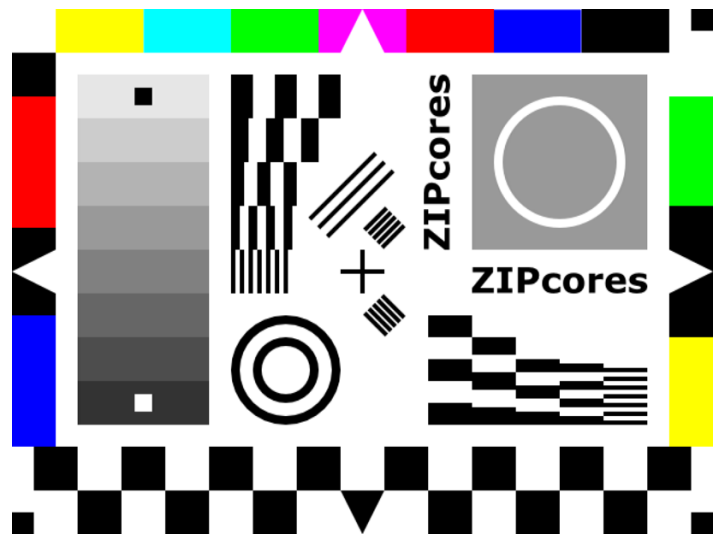


Figure 7: Scale-up by 1.28 - e.g. SVGA to XGA format

Finally, Figure 8 shows the scaling of a image by a factor of 1.875 in the x dimension and 1.406 in the y dimension. As an example, this could represent a scaling from XGA to HD1080p formats.

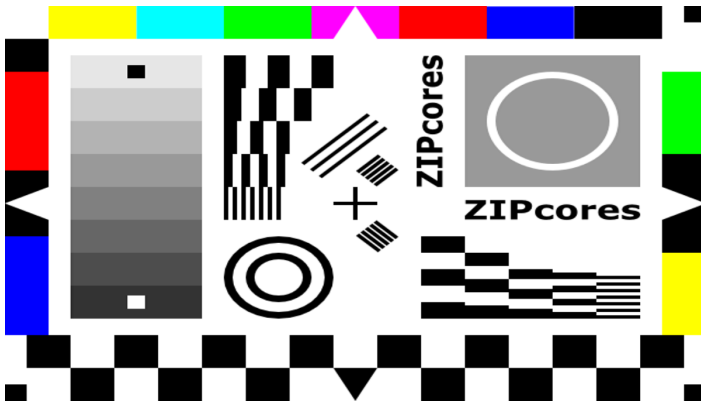


Figure 8: Scale-up by 1.875 in x and 1.406 in y - e.g. XGA to HD1080

If you would like to see the original bitmap source images, or if you would like to see further examples, please contact ZIPcores and we would be happy to send them to you for further inspection.

Synthesis

The files required for synthesis and the design hierarchy is shown below:

- xy2_scaler.vhd
 - pipeline_reg.vhd
 - x2_scaler.vhd
 - pipeline_shovel.vhd
 - x2_buffer.vhd
 - x2_filter_polyphase.vhd
 - pipeline_reg.vhd
 - y2_scaler.vhd
 - pipeline_shovel.vhd
 - y2_buffer.vhd
 - ram_dp_w_r.vhd
 - fifo_sync.vhd
 - pipeline_reg.vhd
 - y2_filter_polyphase.vhd
 - pipeline_reg.vhd

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx Virtex 5 and the Altera Stratix III series of FPGA devices. The lowest and highest speed grade devices have been chosen in both cases for comparison.

Fixing the scale parameters at the scaler input will result in the most optimum scaler design. In addition, the speed of the design may be improved by tying the signal *pixout_rdy* low. This may be possible if the designer knows that the pipeline downstream of the scaler will always be able to accept output pixels. Careful attention must be made to the width of the line stores as this will effect the amount of RAM resource used in the design.

Trial synthesis results are shown with the generic parameters set to: *line_width* = 1024 and *log2_line_width* = 10. Resource usage is specified after Place and Route.

VIRTEX 5

Resource type	Quantity used
Slice register	743
Slice LUT	891
Block RAM	5
DSP48	12
Clock frequency (worst case)	200 MHz
Clock frequency (best case)	250 MHz

STRATIX III

Resource type	Quantity used
Register	1147
ALUT	1230
Block Memory bit	73944
DSP block 18	12
Clock frequency (worse case)	170 MHz
Clock frequency (best case)	220 MHz

Revision History

Revision	Change description	Date
1.0	Initial revision	05/02/2009
1.1	Added extra items to key features	12/06/2009
1.2	Updated synthesis results	15/12/2009
1.4	Added scaling formula Updated source file descriptions to include shovels	18/02/2010