

Key Design Features

- Synthesizable, technology independent VHDL Core
- Versatile RGB-video scaling engine capable of scaling up or down by any aspect ratio.
- Particularly suited to large scale-down operations
- Supports all video resolutions between 16x16 and 4096x4096
- Fully pipelined architecture with simple valid-ready interface protocol. No complex control required
- Variable-tap design allows scaling quality to be adjusted depending on the application
- Choice of 2x2, 4x4, 6x6 or 8x8 filter kernels
- No coefficient programming required
- Compact design with no requirement for a frame buffer
- Output 1 x 24-bit pixel per clock for scaling factors > 1
- Supports FPGA clock rates in excess of 200MHz¹

Applications

- RGB video up/down scaling with flexible image quality
- Higher quality downscaling (when scaling down by factors of 2 or more)
- Generation of 'thumbnail images'
- Digital TV set-top boxes and home media solutions
- Dynamic scaling of video in a window on a frame-by-frame basis
- Picture in picture applications

Generic Parameters

Generic name	Description	Type	Valid range
levels	Number of 2x2 scaler units to instantiate	integer	1: 2 x 2 filter 2: 4 x 4 filter 3: 6 x 6 filter 4: 8 x 8 filter
line_width	Width of linestores in pixels	integer	$2^4 < \text{pixels} < 2^{12}$
log2_line_width	Log2 of linestore width	integer	Log2 (line_width)

Block Diagram

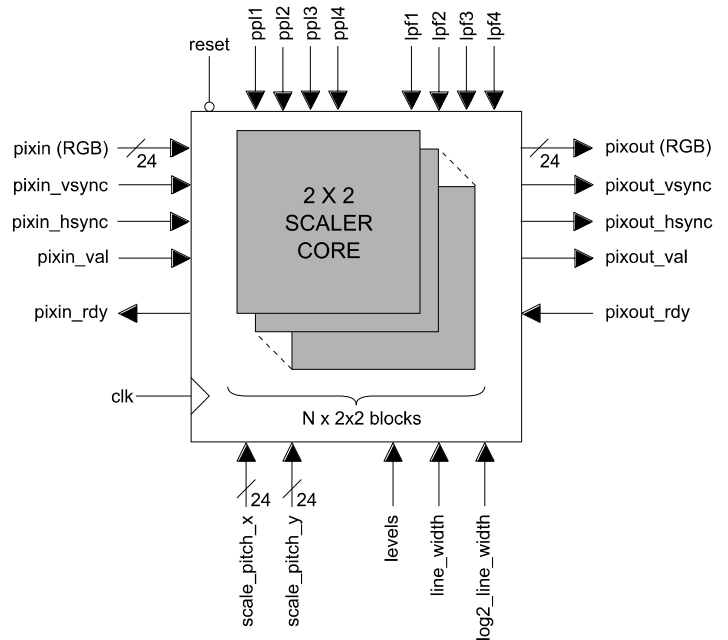


Figure 1: Variable-tap scaler architecture

Pin-out Description

Pin name	I/O	Description	Active state
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
scale_pitch_x [24:0]	in	X scale factor	data
scale_pitch_y [24:0]	in	Y scale factor	data
pp1, 2, 3, 4 [11:0]	in	Pixels per line - stages #1 to #4	data
lpf1, 2, 3, 4 [11:0]	in	Lines per frame - stages #1 to #4	data
pixin [23:0]	in	24-bit RGB pixel in	data
pixin_vsync		Vertical sync in (signifies first pixel in input frame)	
pixin_hsync	in	Horizontal sync in (signifies first pixel in input 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 RGB pixel out	data
pixout_vsync	out	Vertical sync out	high
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

¹ Xilinx Virtex 5 FPGA used as a benchmark

General Description

VT_SCALER is a variable-tap RGB video scaler capable of generating interpolated output video from 16x16 up to 4095x4095 pixels in resolution. The architecture permits seamless scaling (either up or down) depending on the chosen scale factor. VT_SCALER is particularly suited to large downscaling operations and the generation of high-quality thumbnail images.

The scaler itself is fully scalable - permitting the number of filter taps to be adjusted depending on the desired video quality. By setting the generic parameter *levels*, then the scaler may be modified from a basic 2x2 scaler to scaler with 8 taps in both the x and y dimensions.

Pixels flow in and out of the scaling engine in accordance with the valid-ready pipeline protocol. Pixels and syncs are transferred into the scaler on a rising clock-edge when *pixin_val* is high and *pixin_rdy* is high. Likewise, pixels and syncs 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.

Filter scaling parameters

The output resolution of the scaled output image is controlled by the generic parameters *scale_pitch_x*, *scale_pitch_y*, *ppl1,2,3,4* and *lpf1,2,3,4*. It is important to specify these parameters precisely in order to obtain the optimum scaling performance. The scaling factors in x and y are determined by the following formulae:

$$x_fact = (input_res_x/output_res_x)^{1/levels}$$

$$y_fact = (input_res_y/output_res_y)^{1/levels}$$

$$scale_pitch_x = x_fact * 2^{12}$$

$$scale_pitch_y = y_fact * 2^{12}$$

The parameters *ppl* and *lpf* control the resolution of the intermediate video generated internally by the scaler. These parameters should be set as follows:

$$ppl1 = input_res_x / x_fact$$

$$ppl2 = input_res_x / x_fact^2$$

$$ppl3 = input_res_x / x_fact^3$$

$$ppl4 = input_res_x / x_fact^4$$

$$lpf1 = input_res_y / y_fact$$

$$lpf2 = input_res_y / y_fact^2$$

$$lpf3 = input_res_y / y_fact^3$$

$$lpf4 = input_res_y / y_fact^4$$

In these formulae, 'input_res_x' and 'output_res_x' represent the input and output 'x' resolutions of the video signal. Likewise, 'input_res_y' and 'output_res_y' represent the input and output 'y' resolutions.

As an example, consider the scaling of VGA format video (640x480) to a small thumbnail image of dimensions (100x80) pixels. If we choose an 8x8 filter kernel, we would need to set the number of levels to '4'. Plugging the values into the scaling formulae, the parameters would be calculated as follows:

$$x_fact = (640/100)^{1/4} = 1.59054$$

$$y_fact = (480/80)^{1/4} = 1.56508$$

$$scale_pitch_x = 1.59054 * 2^{12} = 6515$$

$$scale_pitch_y = 1.56508 * 2^{12} = 6411$$

$$ppl1 = 640/1.59054 = 402$$

$$ppl2 = 640/1.59054^2 = 253$$

$$ppl3 = 640/1.59054^3 = 159$$

$$ppl4 = 640/1.59054^4 = 100$$

$$lpf1 = 480/1.56508 = 307$$

$$lpf2 = 480/1.56508^2 = 196$$

$$lpf3 = 480/1.56508^3 = 125$$

$$lpf4 = 480/1.56508^4 = 80$$

Note that the respective parameters *ppl* and *lpf* may be tied to zero for unused 2x2 filter stages. For example if we chose to use a 4x4 scaler with *levels* set to '2' then the parameters *ppl3*, *lpf3*, *ppl4*, *ppl4* are unused and may be tied to zero.

Choice of scaler architecture

For the the best image quality, it is recommended that the number of 2x2 stages or levels be determined by the relationship:

For downscaling:

$$levels = \lceil scale\ divisor / 2 \rceil$$

For upscaling:

$$levels \geq 1$$

For instance, when we wish to downscale by a factor of 3 then for best image quality, the number of levels should be the next largest integer value after the division (3/2) - i.e. 2.

Using too many levels will result in a 'softening' of the output image. Conversely, using too few levels may result in a decimated image with a 'blocky' appearance. Ultimately, the ideal number of filter stages is best determined by investigation. Figure 2 demonstrates the effect of using different numbers of filter stages.



(a)



(b)



(c)

Figure 2: Effect of : (a) too few 2x2 filter stages, (b) optimum number of stages, (c) too many causing 'softening'

Flow control

Pixels flow in and out of the scaler in accordance with the valid-ready pipeline protocol². The scaling operation occurs on a line-by-line basis with the signal *pixin_hsync* specifying the first pixel of a new line and *pixin_vsync* specifying the first pixel 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.

After reset, the scaler is held in an idle state. While in the idle state, the scaler will assert *pixin_rdy* high in until the first pixel of a frame is received. While in the idle state, no output pixels will be generated.

On receipt of the first valid *pixin_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 stalling of the input due to the change of data rate.

N.B. It is very important that the valid-ready protocol is observed correctly at the input and output interfaces (whether downscaling or upscaling) to avoid the loss of pixels and a corrupted output image.

² See ZIPcores application note: app_note_zc001.pdf for more examples of how to use the valid-ready pipeline protocol

Loading of scale parameters

All input scale parameters including *scale_pitch_x*, *scale_pitch_y*, *ppl*, and *lpf* 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.

Functional Timing

Figure 3 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.

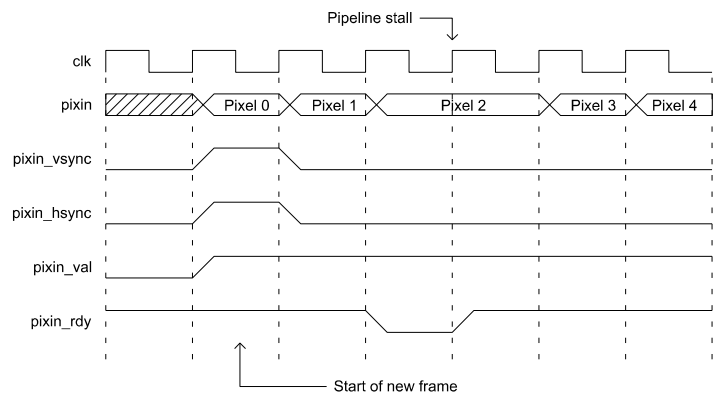


Figure 3: First pixel in a frame

Figure 4 shows the signalling at the output of the scaler.

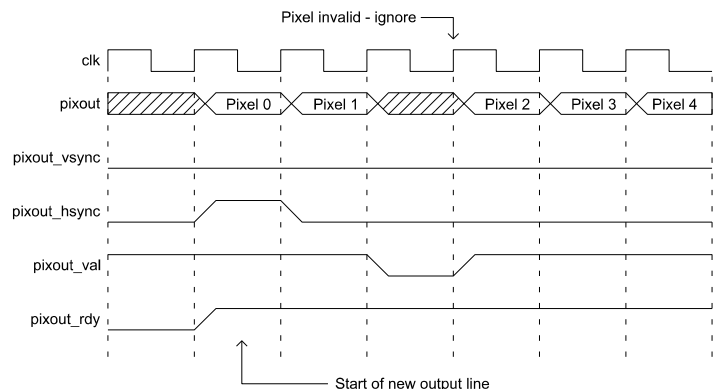


Figure 4: Scaler output showing invalid pixel

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.

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 element
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
scaler_2x2.vhd	Video scaler 2x2 core module
vt_scaler.vhd	Top-level component
vt_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. scaler_2x2.vhd
15. vt_scaler.vhd
16. vt_scaler_bench.vhd

The VHDL testbench instantiates the VT_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 0.2 in the x and y dimensions to give an output image of 128x96 pixels.

Synthesis

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

- vt_scaler
 - pipeline_reg.vhd
 - scaler_2x2.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. Increasing the number of filter stages (levels) will result in a greater number of filter taps - and hence a larger number of hardware multipliers.

Trial synthesis results are shown with the generic parameters set to: levels = 2, line_width = 2048 and log2_line_width = 11.

Resource usage is specified after Place and Route.

VIRTEX 5

Resource type	Quantity used
Slice register	1606
Slice LUT	1878
Block RAM	10
DSP48	24
Clock frequency (worst case)	200 MHz
Clock frequency (best case)	252 MHz

STRATIX III

Resource type	Quantity used
Register	1527
ALUT	1917
Block Memory bit	295344
DSP block 18	24
Clock frequency (worse case)	170 MHz
Clock frequency (best case)	220 MHz

Revision History

Revision	Change description	Date
1.0	Initial revision	08/01/2010
1.1	Fixed typos in scaling example Modified loading of scale parameters section	22/02/2010