Key Design Features

- Synthesizable, technology independent soft IP Core for FPGA, ASIC and SoC devices
- Supplied as human readable VHDL (or Verilog) source code
- Versatile RGB (or YCbCr 444) video scaler capable of scaling up or down by any factor
- Fully programmable scale parameters and scaler bypass function
- Fully programmable RGB channel widths allow support for any RGB format (or greyscale if only one channel is used)
- Supports all video resolutions up to 2^{16} x 2^{16} pixels
- Fully pipelined architecture with simple data-streaming flow control
- Features a 5x5-tap polyphase filter in the x and y dimensions with 16 unique phases
- Example general purpose 'Lanczos2' filter coefficients shipped with the design. Different coefficient sets available on request
- Output rate is 1 x pixel per clock for scaling factors > 1
- Generates one scaled output frame for every input frame
- No frame buffer required
- Supports 350MHz+ operation on basic FPGA devices[1]

Applications

- Studio quality dynamic real-time video scaling
- Conversion of all standard and custom video resolutions such as HD720P to HD1080P, XGA to VGA etc.
- Support for the latest generation video formats with resolutions of 4K and above
- Video scaling for flat panel displays, portable devices, video image sensors, consoles, video format converters, set-top boxes, digital TV etc.
- Picture-in-Picture (PiP) and dynamic zoom applications

Generic Parameters

Generic name	*Description*	*Type*	*Valid range*
dw	RGB channel width	integer	≥ 2
line_width	Width of linestores in pixels	integer	2^4 < pixels < 2^{16}
log2_line_width	Log_2 of linestore width	integer	Log_2 (line_width)

Block Diagram

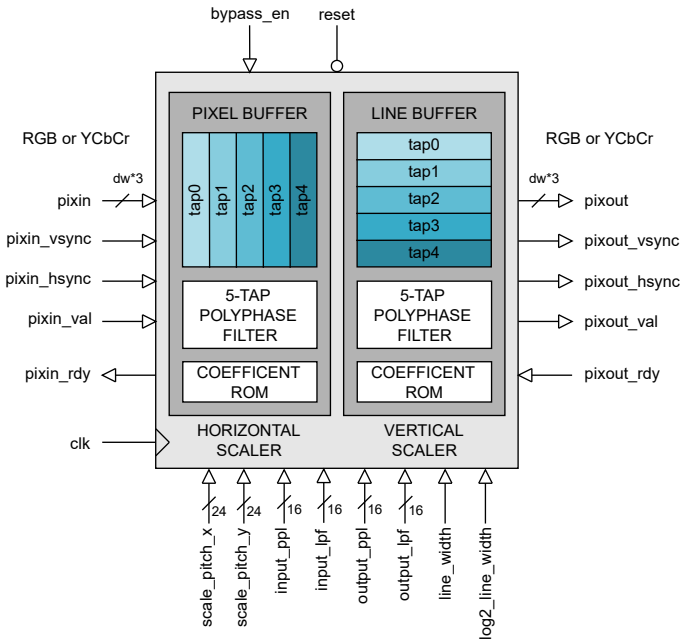

Figure 1: Digital video scaler architecture

Pin-out Description

Pin name	*I/O*	*Description*	*Active state*
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
bypass_en	in	Bypass the video scaling function	0: scaled video 1: bypass video
scale_pitch_x [23:0]	in	1 / (x scale factor) (Unsigned number in [24 12] format)	data
scale_pitch_y [23:0]	in	1 / (y scale factor) (Unsigned number in [24 12] format)	data
input_ppl [15:0]	in	Number of pixels per line in the source video (Unsigned 16-bit number)	data
input_lpf [15:0]	in	Number of lines per frame in the source video (Unsigned 16-bit number)	data
output_ppl [15:0]	in	Number of pixels per line in the scaled output video (Unsigned 16-bit number)	data
output_lpf [15:0]	in	Number of lines per frame in the scaled output video (Unsigned 16-bit number)	data

1 Xilinx® 7-series used as a benchmark

Pin-out Description cont ...

Pin name	I/O	Description	Active state
pixin [dw*3 - 1:0]	in	RGB pixel in	data
pixin_vsync	in	Vertical sync in (Coincident with first pixel of input frame)	high
pixin_hsync	in	Horizontal sync in (Coincident with first pixel of input line)	high
pixin_val	in	Input pixel valid	high
pixin_rdy	out	Ready to accept input pixel (Handshake signal)	high
pixout [dw*3 - 1:0]	out	RGB pixel out	data
pixout_vsync	out	Vertical sync out (Coincident with first pixel of output frame)	high
pixout_hsync	out	Horizontal sync out (Coincident with first pixel of output line)	high
pixout_val	out	Output pixel valid	high
pixout_rdy	in	Ready to accept output pixel (Handshake signal)	high

General Description

The XY_SCALER IP Core is a studio quality video scaler capable of generating interpolated output images from 16 x 16 up to 2^{16} x 2^{16} 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 into and out of the video scaler in accordance with a simple valid-ready streaming protocol. Pixels are transferred into the scaler on a rising clock-edge when *pixin_val* and *pixin_rdy* are both active high. Likewise, pixels are transferred out of the scaler on a rising clock-edge when *pixout_val* and *pixout_rdy* are both active high. As such, the pipeline protocol allows both input and output interfaces to be stalled independently.

The scaler is partitioned into a horizontal scaling module in series with a vertical scaling module as shown by Figure 1.

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*, *input_ppl, input_lpf, output_ppl* and *output_lpf*. The scale pitch may be calculated using the following formula:

$$\text{pitch} = \left(\frac{\text{Input resolution}}{\text{Output resolution}}\right) * 2^{12}$$

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 value '2560'.

In addition the user must also specify the exact resolution of the source input frame and the scaled output frame using the parameters: *input_ppl, input_lpf, output_ppl* and *output_lpf*. The following tables give a list of generic parameters required for the conversion of some example video formats.

SCALE UP

Video IN	Video OUT	Pitch X	Pitch Y	I/P PPL	I/P LPF	O/P PPL	O/P LPF
VGA 640x480	SVGA 800x600	3277	3277	640	480	800	600
SVGA 800x600	XGA 1024x768	3200	3200	800	600	1024	768
XGA 1024x768	HD1080 1920x1080	2184	2913	1024	768	1920	1080
SXGA 1280x1024	2K 2048x1080	2560	3884	1280	1024	2048	1080

SCALE DOWN

Video IN	Video OUT	Pitch X	Pitch Y	I/P PPL	I/P LPF	O/P PPL	O/P LPF
SVGA 800x600	VGA 640x480	5120	5120	800	600	640	480
XGA 1024x768	SVGA 800x600	5243	5243	1024	768	800	600
HD1080 1920x1080	XGA 1024x768	7680	5760	1920	1080	1024	768
2K 2048x1080	SXGA 1280x1024	6554	4320	2048	1080	1280	1024

Flow control

Pixels flow in and out of the video scaler in accordance with the valid-ready pipeline protocol[2]. 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 *pixin_vsync* and *pixin_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 in ratio.

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

Loading of scale parameters and bypass mode

The scale parameters are fully programmable and allow the input video to be scaled differently on a frame-by-frame basis. With careful design, the architecture also permits different video sources to be multiplexed into the same scaler with different scaling parameters.

Parameters are updated continuously on every rising clock edge and must remain stable during the scaling operation. When programming new scale parameters (e.g. due to a change of video mode) it is necessary to assert the system reset signal for at least one clock cycle to avoid any possible corruption in the output video. This is often convenient to do during the vertical blanking period of an input video frame when there are no active pixels. After reset the scaler will lock to the next clean input frame before the scaling operation continues.

The video scaling function may be bypassed completely by asserting the *bypass_en* signal high. In bypass mode then the video input is passed directly to the video output to give exact 1:1 video in/out. Switching in and out of bypass mode must be done in the same manner as switching scaling parameters. That is, a system reset must be performed when there are no active pixels being processed by the scaler to avoid corruption of the output video.

Scaling algorithm

The scaler uses a 5-tap polyphase filter with 16 phases in both the x and y dimensions. By default, both the x and y filter kernels use a coefficient set sampled from the Lanczos2 function (Figure 2).

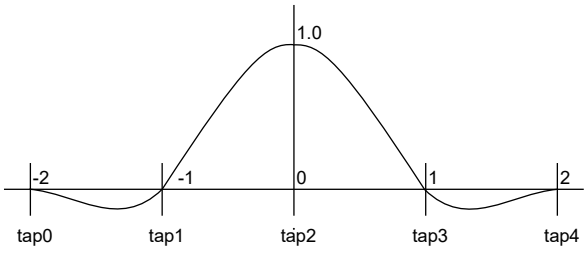

Figure 2: Lanczos2 windowed-sinc function - filter tap positioning

Figure 3, below shows how the phase changes relative to the pixel taps during the scaling operation. Depending on the fractional part of the accumulator, different weights are given to the pixel taps when generating the interpolated output pixels.

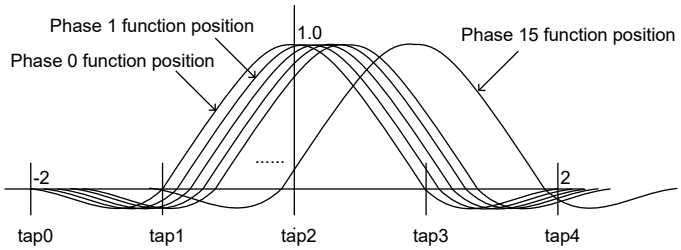

Figure 3: The 16-phases of the 5-tap filter

Different filter kernels can generate slightly different results. Example scripts are provided to generate: Lanczos2, Lanczos3, Hamming and Kaiser coefficient sets. Alternatively, the user may choose to generate their own coefficient sets[3].

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.

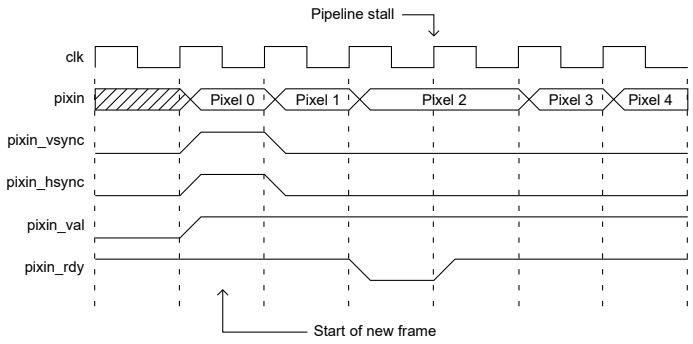

Figure 4: First line of a new input frame - also showing pipeline stall

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.

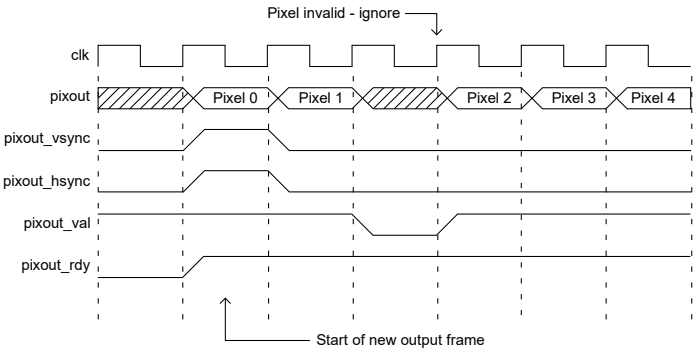

Figure 5: First line of a new output frame – also showing invalid output pixel

3 See Zipcores application note: app_note_zc003.pdf for examples of how to generate different coefficient sets

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
x_buffer.vhd	Pixel input buffer/shift register
x_filter_pack.vhd	Package containing x-filter coefficients
x_filter_polyphase.vhd	Horizontal scaler output pixel filter
x_scaler.vhd	Horizontal scaler component
y_buffer.vhd	Line buffer
y_filter_pack.vhd	Package containing y-filter coefficients
y_filter_polyphase.vhd	Vertical scaler output pixel filter
y_scaler.vhd	Vertical scaler component
xy_reg.vhd	Video scaler input registers
xy_scaler.vhd	Video scaler top-level component
xy_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. x_buffer.vhd
7. x_filter_pack.vhd
8. x_filter_polyphase.vhd
9. x_scaler.vhd
10. y_buffer.vhd
11. y_filter_pack.vhd
12. y_filter_polyphase.vhd
13. y_scaler.vhd
14. xy_reg.vhd
15. xy_scaler.vhd
16. xy_scaler_bench.vhd

The VHDL testbench instantiates the XY_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[4]. This file contains a sequential list of 24-bit output pixels in the same format as *video_in.txt*. The example provided scales a 768x576 source test pattern by a factor of 0.833 in the x and y dimensions to give a VGA output image of 640x480 pixels. Figure 6 shows the resulting image from the test.

Figure 6: Output frame from the hardware simulation example (Scale-down of 768x576 to 640x480)

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 video. The true definition and quality is difficult to show within the limitations of this document, however, example images can be provided on request.

The video scaler was also verified using the Zipcores ZIP-HDV-001 development board featuring a Xilinx Spartan6 FPGA. The photo in Figure 7 demonstrates the scale down of a PAL source image to a small custom video window of 500x400 pixels on an SXGA (1280x1024) background.

4 Simple PERL scripts for generating and processing input and output text files are provided with the IP Core package