Key Design Features

- Technology independent soft IP Core for FPGA, SoC and ASIC devices
- Supplied as human readable VHDL source code (or Verilog on request)
- Versatile RGB video scaler capable of scaling up or down by any factor
- Fully programmable scale parameters and scaler bypass function
- Optimized for 4K video (UHD) but will support any video resolution up to 2^{16} x 2^{16} pixels
- Fully pipelined architecture with simple flow control and AXI4-stream compatible data streaming interfaces[1]
- Features a 5x5-tap polyphase filter core with optional decimation and interpolation input and output filters
- Supports 2 x 24-bit RGB pixels per clock
- Generates one scaled output frame for every input frame
- No external frame buffer needed (i.e. no external memory requirement)
- Supports 300 MHz+ operation on mid-range FPGA devices[2]. (i.e. equivalent to 600 MHz+ @ 2 x pixels / clock)

Applications

- Studio quality dynamic real-time video scaling
- Conversion between any standard or custom video resolution
- Support for the latest generation video formats with resolutions of 4K (UHD) and above
- Video scaling for flat panel displays, portable devices, image sensors, digital cameras, video consoles, video format converters, set-top boxes, smart TVs etc.
- Scaling for Picture-in-Picture (PiP), Picture-by-Picture (PbP), Multi-window formats and dynamic zoom applications

Generic Parameters

Generic name	*Description*	*Type*	*Valid range*
line_width	Width of linestores in pixels (Must accommodate max line length *before* scaling)	integer	2^4 < pixels < 2^{16}
log2_line_width	Log_2 of linestore width	integer	Log_2 (line_width)

1 Please ask Zipcores for details re: AXI4-Stream compatible interfaces
2 AMD / Xilinx® 7-series devices used as a benchmark

Block Diagram

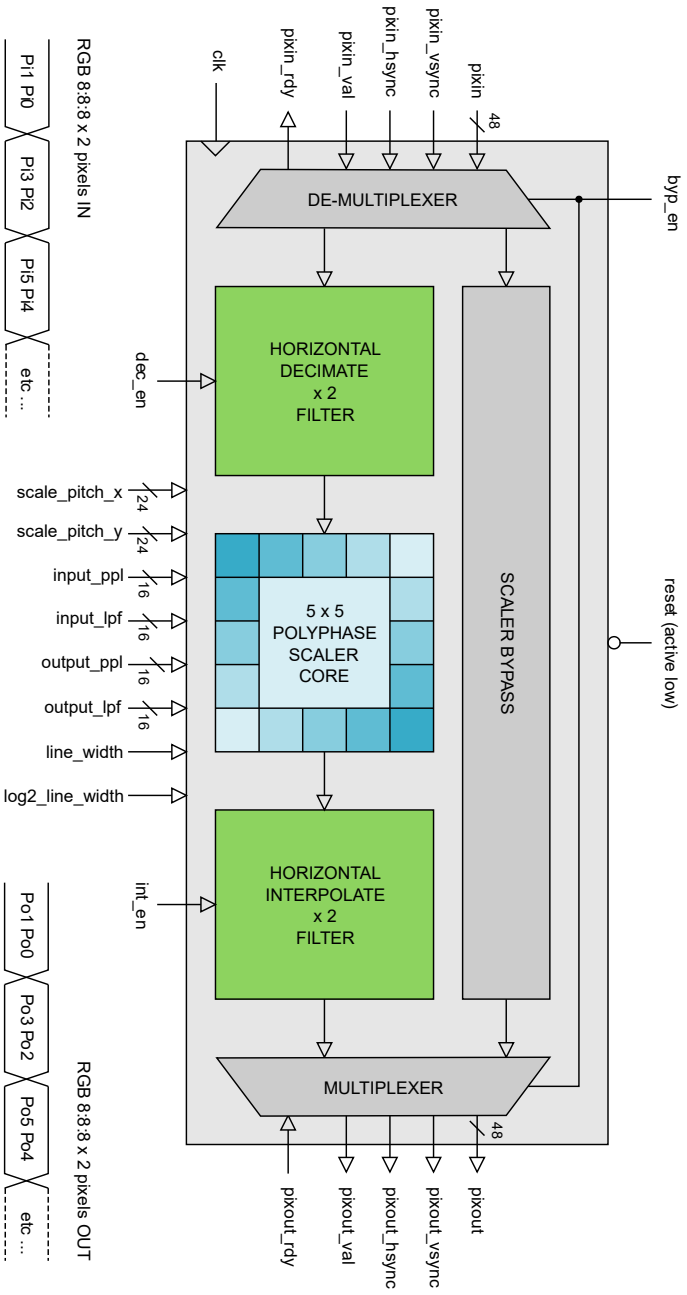

Figure 1: 4K Video Scaler IP Core general architecture

Pin-out Description

Pin name	I/O	Description	Active state
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low

dec_en	in	Enable decimate by 2 in x	0: no decimate 1: decimate
int_en	in	Enable interpolate by 2 in x	0: no interp. 1: interpolate
byp_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

pixin [47:0]	in	2 x RGB pixels (RGB 888)	data
pixin_vsync	in	Vertical sync flag in (coincident with first pixels of input frame)	high
pixin_hsync	in	Horizontal sync flag in (coincident with first pixels of input line)	high
pixin_val	in	Input pixels valid	high
pixin_rdy	out	Ready to accept input pixels (handshake signal)	high

pixout [47:0]	out	2 x RGB pixels (RGB 888)	data
pixout_vsync	out	Vertical sync flag out (coincident with first pixels of output frame)	high
pixout_hsync	out	Horizontal sync flag out (coincident with first pixels of output line)	high
pixout_val	out	Output pixels valid	high
pixout_rdy	in	Ready to accept output pixels (handshake signal)	high

General Description

The VID_SCALER_4K IP Core is a studio quality video scaler capable of generating scaled output images 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 and the decimation or interpolation parameters.

Internally, the video scaler is comprised of an input decimation section, a polyphase filter core section and an output interpolation section. In addition, the scaler is capable of processing two RGB pixels per clock and, as such, is able to support 4K+ pixel rates on most mid-range FPGA and SoC devices. Figure 1 shows the general architecture in more detail.

The polyphase filter core 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 if required. By default, the polyphase filter kernel uses a Lanczos2 windowed-sinc function that gives excellent all round performance.

Pixels flow into and out of the video scaler in accordance with a simple valid-ready streaming protocol[3]. Pixels and syncs are transferred into the scaler on a rising clock-edge when *pixin_val* and *pixin_rdy* are both active high. Likewise, pixels and syncs are transferred out of the scaler on a rising clock-edge when *pixout_val* and *pixout_rdy* are both active. In this way, the pipeline protocol allows both input and output interfaces to be controlled independently. The scaler may also be easily adapted to use standard AXI4-stream interfaces which are popular with many vendors of FPGA, SoC and ASIC devices.

Input decimation filter

The input stage of the scaler pipeline is a horizontal decimate-by-2 filter with 4 filter taps. The decimate function is enabled by setting the *dec_en* signal active high.

Use of the decimate function is recommended for *all downscale* operations where the input video resolution is 4K (UHD) or above. This is necessary in order to optimize the video bandwidth into the scaler.

The decimate-by-2 function operates in the x-dimension only and *halves* the number of input pixels per line. So for example, if the input video resolution is 3840 x 2160 pixels then the video resolution into the 5x5 polyphase scaler core will be 1920 x 2160 pixels *after* decimation.

Output interpolation filter

The output stage of the scaler pipeline is a horizontal interpolate-by-2 filter which performs the reciprocal operation to the decimation filter. The interpolate function is enabled by setting the *int_en* signal active high.

Use of the interpolate function is recommended for *all upscale* operations where the output video resolution is 4K (UHD) or above. This is necessary in order to optimize the video bandwidth out of the scaler.

As with the decimation function, the interpolate-by-2 function only operates in the x-dimension but *doubles* the number of output pixels per line. So for example, if the required output video resolution is 3840 x 2160 pixels then the video resolution out of the 5x5 polyphase scaler core should be 1920 x 2160 pixels *before* the interpolation stage.

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

Polyphase scaler core

The main scaling function is provided by the 5x5 polyphase scaler core which operates in both the x and y dimensions. The setup of the polyphase scaler section requires various parameters to be programmed correctly while also taking into account whether the input decimation or output interpolation functions are enabled.

The format of the scaled output image is controlled by the parameters: *dec_en, int_en, byp_en, scale_pitch_x, scale_pitch_y, input_ppl, input_lpf, output_ppl and output_lpf.*

The scale pitch of the main polyphase scaler may be calculated using the formula:

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

Example 1: Scale down of 3840x2160 → 1280x720

In this case we set:

dec_en = 1, int_en = 0, byp_en = 0

(Remembering that decimation in x is applied first so that the scaling params into the polyphase scaler are for 1920x2160 → 1280x720)

scale_pitch_x = (1920/1280 * 2^{12}) = 6144
scale_pitch_y = (2160/720 * 2^{12}) = 12288
input_ppl = 1920
input_lpf = 2160
output_ppl = 1280
output_lpf = 720

Example 2: Scale up of 1280x720 → 3840x2160

In this case we set:

dec_en = 0, int_en = 1, byp_en = 0

(Remembering that interpolation in x is applied last so that the scaling params into the polyphase scaler are for 1280x720 → 1920x2160)

scale_pitch_x = (1280/1920 * 2^{12}) = 2731
sclae_pitch_y = (720/2160 * 2^{12}) = 1365
input_ppl = 1280
input_lpf = 720
output_ppl = 1920
output_lpf = 2160

Example 3: Scale up of 640x480 (VGA) → 1920x1080 (HD)

In this case we may disable the decimation and interpolation stages (as recommended when not dealing with 4K video).

dec_en = 0, int_en = 0, byp_en = 0

scale_pitch_x = (640/1920 * 2^{12}) = 1365
sclae_pitch_y = (480/1080 * 2^{12}) = 1820
input_ppl = 640
input_lpf = 480
output_ppl = 1920
output_lpf = 1080

Loading of scale parameters

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 *byp_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 possible corruption of the output video.

Polyphase scaling algorithm

The polyphase 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).

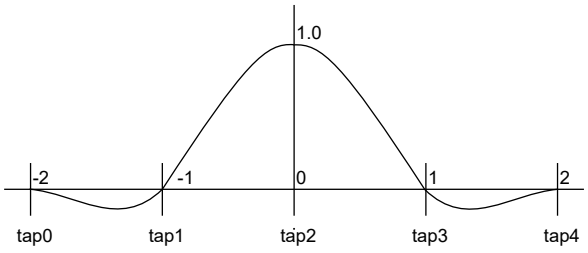

Figure 2: Lanczos2 windowed-sinc function - filter tap positioning

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.

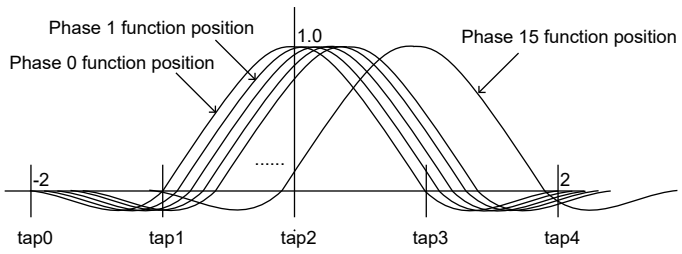

Figure 3: The 16-phases of the 5-tap filter

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

Functional Timing

The functional timing of the input and output interfaces follows a simple flow-control protocol with a *valid* and *ready* signal. These signals are compatible with the *t_valid* and *t_ready* signals used in the AMBA / AXI4-stream adopted by ARM and many other IP Core vendors like AMD / Xilinx®.

Input interface timing

Figure 4 shows the signalling at the start of a new frame. Pixels are processed in pairs with the least recent pixel in the lower 24-bits and the most recent pixel in the upper 24-bits of pixel data.

The first line of a new frame begins with *pixin_vsync* and *pixin_hsync* asserted high together with the first pixel data. 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.

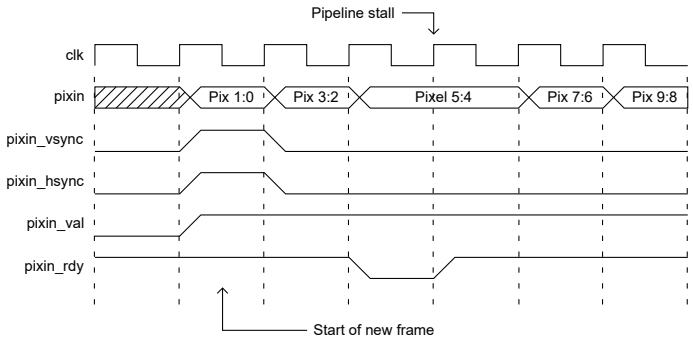

Figure 4: Start of a new input frame (also showing a pipeline stall)

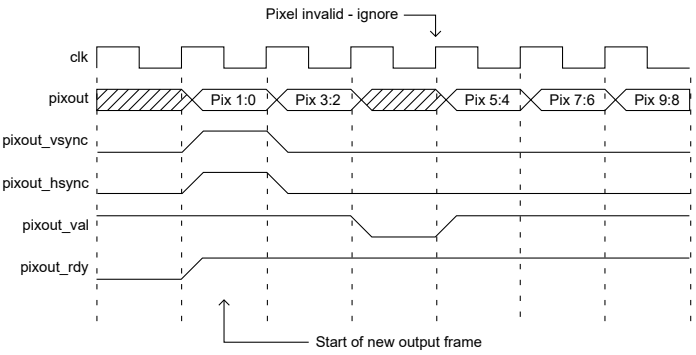

Figure 5: Start of a new output frame (also showing invalid output pixel)

Output interface timing

Figure 5 shows the signalling at the output of the scaler for a new frame. The output uses exactly the same protocol as the input. In this particular example, it shows *pixout_val* de-asserted for 1 clock-cycle, in which case, the output pixel data should be ignored.

Transfers at a valid-ready interface are only permitted when valid and ready are both active high on the same clock edge.

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***
video2_in.txt	Text-based source image file
video2_file_reader.vhd	Reads image file into test bench

pipeline_reg.vhd	Pipeline register component
pipeline_shovel.vhd	Pipeline 'shovel' register
ram_dp_w_r.vhd	Dual-port RAM component
fifo_sync.vhd	Synchronous FIFO

dec2_x.vhd	Decimate-by-2 filter component
dec2_x_mux.vhd	Decimate-by-2 multiplexer
int2_x.vhd	Interpolate-by-2 filter component
int2_x_mux.vhd	Interpolate-by-2 multiplexer
pix_24_to_48.vhd	Pixel packer (24-bit to 48-bit)
pix_48_to_24.vhd	Pixel unpacker (48-bit to 24-bit)

x_filter_pack.vhd	Package containing x-filter coefficients
x_buffer.vhd	Input pixel buffer
x_filter_polyphase.vhd	Horizontal scaler pixel filter kernel
x_scaler.vhd	Horizontal scaler component

y_filter_pack.vhd	Package containing y-filter coefficients
y_buffer.vhd	Input line buffer
y_filter_polyphase.vhd	Vertical scaler pixel filter kernel
y_scaler.vhd	Vertical scaler component

xy_reg.vhd	Polyphase scaler input register stage
xy_scaler.vhd	Polyphase scaler top-level component

vid_scaler_4k.vhd	Top-level 4K video scaler component
vid_scaler_4k_dec_bench.vhd	Top-level test bench (downscale)
vid_scaler_4k_int_bench.vhd	Top-level test bench (upscale)

Functional Testing

An example test bench is provided for use in a suitable hardware simulator. The compilation order of the source code is the same order as the source code file description (above).

The test bench instantiates the *vid_scaler_4k.vhd* component and the user may modify the parameters in order to generate the desired scaled output image.

The input source image for the simulation is generated by the *video2_file_reader.vhd* component. This component reads a text file which contains the RGB pixel data and video sync flags. The text file is called *video2_in.txt* and should be placed in the top-level simulation directory.

The text file 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 50 70 03 45 65 00  # pixel 1,0 line0 (SOF & SOL)
1 0 0 51 71 06 55 74 0c  # pixel 3,2 line0
1 0 0 53 73 06 58 78 0b  # pixel 5,4 line0

.
.
.

1 0 1 4b 6b 00 4f 6f 02  # pixel 1,0 line1 (SOL only)
1 0 0 52 72 07 46 65 00  # pixel 3,2 line1
1 0 0 5d 7d 10 48 68 00  # pixel 5,4 line1

.
.
.

etc.
```

For instance, in this example, the first line of of the *video2_in.txt* file asserts the input signals pixin_val = 1, pixin_vsync = 1, pixin_hsync = 1 and pixin = 0x507003, 0x456500. This represents the first 2 pixels in the first line of the first frame.

The simulation must be run for around 50 ms during which time an output text file called *video2_out.txt* will be generated[4]. This file contains a sequential list of output pixels in the same format as *video2_in.txt*.

There are two example test benches provided. The test bench called *vid_scaler_4k_dec_bench.vhd* demonstrates a downscale operation from 3840x2160 to 1280x720 pixels. Conversely, the test bench called *vid_scaler_4k_int_bench.vhd* demonstrates an upscale operation from 1280x720 to 3840x2160 pixels.

Note that for each test then the correct *video2_in.txt* file must be used for the correct size input frame. If not, then the output image will be corrupted.

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

Figure 6: Example scaled image from the hardware simulation

Synthesis and Implementation

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

- vid_scaler_4k.vhd
  - dec2_x_mux.vhd
    - dec2_x.vhd
      - pipeline_reg.vhd
    - pix_48_to_24.vhd
  - int2_x_mux.vhd
    - int2_x.vhd
      - pipeline_reg.vhd
    - pix_24_to_48.vhd
  - xy_scaler.vhd
    - xy_reg.vhd
      - pipeline_reg.vhd
    - x_scaler.vhd
      - pipeline_shovel.vhd
      - x_buffer.vhd
      - x_filter_polyphase.vhd
        - pipeline_reg.vhd
    - y_scaler.vhd
      - pipeline_shovel.vhd
      - y_buffer.vhd
        - ram_dp_w_r.vhd
      - fifo_sync.vhd
        - pipeline_reg.vhd
      - y_filter_polyphase.vhd
        - pipeline_reg.vhd

The IP Core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the AMD / Xilinx® 7-series FPGA devices. Synthesis results for different FPGAs, SoCs and other technologies can be provided on request.

Fixing the scale parameters at the scaler inputs 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. For single channel (greyscale) operation then the user may use only one of the RGB pixel channels and tie the other channel inputs to zero. This will result in further resource savings with the other 2 channels optimized away during synthesis.

Trial synthesis results are shown with the generic parameters set to: *line_width* = 4096 and *log2_line_width* = 12.

Resource usage is specified after place and route of the design.

AMD / XILINX® 7-SERIES FPGAS

Resource type	***A-7***	***K-7***	***V-7***	***US+***
Registers	2100	2100	2100	2100
LUTs	3253	3198	3202	3271
Block RAM	18	18	18	18
DSPs	0	0	0	0
Occupied Slices	1224	1085	1145	647 (CLB)
Clk freq. (approx)	200 MHz	300 MHz	325 MHz	350 MHz

Revision History

Revision	***Change description***	***Date***
1.0	Initial revision (internal library)	03/01/2024
1.1	Added support for 2 x pixels/clock	22/11/2024
1.2	Added bypass function	20/12/2024
1.3	First official release	02/01/2025