1) How do the pixin_vsync and pixin_hsync relate to the VSYNC and HSYNC video timing signals at the DAC?

The signals *pixin_vsync* and *pixin_hsync* should not be confused with true video timing signals; for instance, the VSYNC and HSYNC at a video DAC or HDMI transmitter IC. Their only purpose is to identify the first pixel in a frame and first pixel in a line.

The signal *pixin_vsync* is coincident with the first pixel of an input frame. The signal *pixin_hsync* is coincident with the first pixel of an input line. Likewise, the signals *pixout_vsync* and *pixout_hsync* are coincident with the first pixels of an output frame or line. Figure 1. shows the simplified timing waveforms.

If it helps, the *pixin_vsync* and *pixin_hsync* signals can be considered as two extra bits of sideband information attached to the pixel. Also remember that, like the pixel, the sync signals are qualified by the valid.

When valid is low then the syncs are ignored.

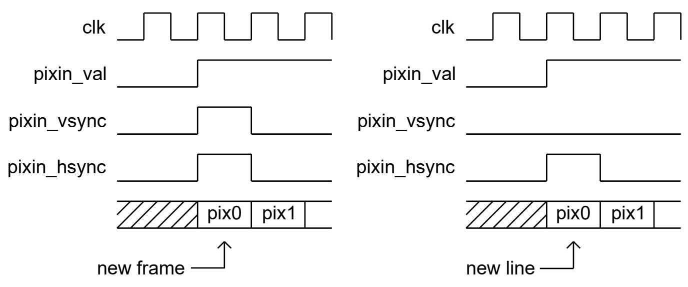

Figure 1: pixin_vsync and pixin_hsync timing

Essentially, the video scaler uses a high-speed data-streaming interface. It only processes active pixels and does not waste bandwidth by processing 'redundant' periods of horizontal or vertical blanking.

2) Why doesn't the video scaler generate output pixels immediately?

After reset, the scaler will discard input pixels until the first input frame is found. This will be when *pixin_vsync* and *pixin_val* are both high. From this point onwards, the scaling operation begins.

Depending on the scaler type, it may take up to 2 or 3 lines of input video before any output video is generated. This is due to the internal line buffers being pre-filled with pixels.

3) How do the video scalers generate pixels and lines at the start and at the end of each line or frame?

At the start of a new frame, the video scaler filter taps in x and y are empty (or at least contain invalid pixels). For this reason the pixels/lines are treated differently at the frame boundaries.

At the beginning of a new line or frame then the first couple of pixels or lines are replicated in the filter taps. Likewise, at the end of a line or frame then the last pixel or line is also replicated. For this reason, the pixels at the frame boundaries may show some slight discrepancies with the original image. Please take this into consideration when designing with the video scalers.

4) The output video looks corrupted. I think I'm doing something wrong ...

The most common way that the output video can be corrupted is if pixels are lost or repeated at the scaler interfaces. This can happen when the valid-ready protocol is used incorrectly. Pixels and syncs are sampled at the input (or output) of the scaler on a rising clock-edge when valid and ready are both high. It's important to wire up valid and ready correctly on both scaler interfaces (irrespective of whether down-scaling or up-scaling is used). Failure to do so will result in corrupted video. It may be easier to understand valid-ready signalling in terms of FIFO nomenclature. See FAQ 6. for an example wiring diagram.

One other way the output video may be corrupted is if the length of line buffers is not correct. The line buffers must be set large enough to accommodate a fully *scaled* output line. For instance, if scaling up from 640x480 to 1920x1080 then the line buffers should be set to 2048 deep which is the nearest power of 2 for the scaled output line.

5) Is the video scaler compatible with Xilinx AXI or Altera/Intel Avalon streaming interfaces?

Yes. The valid/ready flow control is directly compatible with the AMBA®4 AXI4-stream protocol as specified by ARM and used extensively on Xilinx FPGAs. In general, the valid/ready signalling and pixel/sideband data may be interchanged with the equivalent *tvalid*, *tready* and *tdata* signals of AXI.

Likewise, the valid/ready interface is also compatible with Avlon-ST used by Altera/Intel FPGAs. One exception is that the 'readyLatency' must be set to zero for full compatibility with the Zipcores scaler.

6) How can I interface the video scaler to an input FIFO or output FIFO? Which Xilinx or Altera /Intel FIFOs should I use?

It's very simple to connect a FIFO to the input or output of our video scalers. Figure 2. shows an example wiring diagram.

On the input side of the scaler, the *pixin_val* input should be wired to the 'not empty' flag of the FIFO. The *pixin_rdy* output should be wired to the read-enable port of the FIFO. On the output side, The *pixout_val* signal should be wired to the write-enable port of the FIFO. The *pixout_rdy* signal should be wired to the 'not full' flag. The sync signals should be bundled together with the pixel.

It is important to note that in order for the flow-control to work correctly, the 'empty' and read flags of the FIFO should have zero latency. That is, the flags should immediately reflect the state of the FIFO. For instance, if the empty flag is updated one cycle late, then an input pixel may be incorrectly read. Likewise, if the full flag is updated one cycle late, then an output pixel could be lost.

Xilinx® and Altera® offer FIFO solutions in their IP libraries. For compatibility with the video scalers, the Xilinx FIFO should be generated with the FIRST-WORD-FALL-THROUGH option (FWFT). In the Altera case, the SHOW-AHEAD-SYNC-FIFO-MODE should be selected.

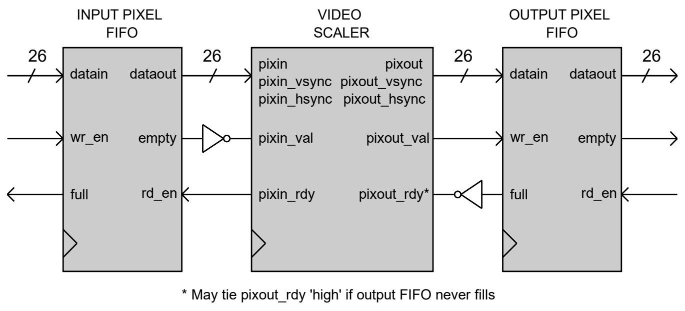

Figure 2: Video scaler FIFO wiring

7) I've sent one complete frame of video to the scaler but I don't get one complete frame out. Am I doing something wrong?

For every complete input video frame the scaler will produce one complete *scaled* output video frame. If this behaviour is not observed, then there are a couple of common things to check. One possibility is that the valid-ready protocol at the interfaces is not being observed correctly. This will result in pixels being lost at the interfaces. The other is incorrect scaler parameters. Always ensure that the number of pixels per line and lines per frame have been specified correctly for the chosen scale factor.

8) Can I multiplex different video sources into the same scaler?

Yes, this is easily done, as long as the video input sources are multiplexed cleanly between frames. The circuit should be designed so that *pixin_val*, *pixin_vsync* and *pixin_data* are swapped between each source at the end of each complete input frame. This is often easiest to do during vertical blanking periods. Also make sure that the scaling parameters are correct for each respective source. Figure 3. shows a possible arrangement for scaling multiple video sources with the same video scaler.

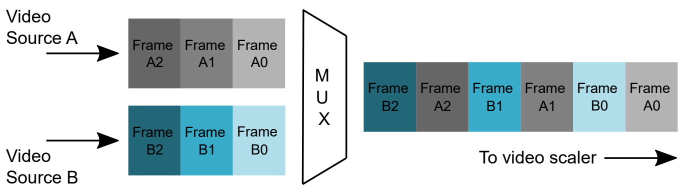

Figure 3: Multiplexing dual video streams into the video scaler

9) There are long periods when pixin_rdy goes low. Why is this?

There are a number of situations where this might happen. The first instance is at the start of a new frame. In this case, there is a delay as the first pixel taps are filled both horizontally and vertically (see FAQ 2). When up-scaling, *pixin_rdy* will go low due to the change in ratio.

For optimum performance, it is essential that the input frame size is correct for the given scaling parameters. In addition, the user must ensure that the input size frame is at least 16x16 pixels otherwise the scaler will fail.

10) How do I recover from a corrupted input video frame?

The safest way to recover is to assert a system reset (active low) for at least 1 system clock cycle. As the system reset is asynchronous, make sure that the signal is clocked though a couple registers to avoid possible removal time issues. Once reset, the video scaler will then re-align to the next start of frame and continue operation as normal.

11) Can I use the video scaler without a frame buffer?

Yes, it's possible to use the video scaler without a frame buffer. However, the design must ensure that the frame rate in is *exactly* the same as the frame rate out. If this is not the case, then the input and output video will become out of sync and eventually the system will fail.

In order to maintain the correct relationship between input and output frame rates, then the input pixel clock and the output pixel clock must have a fixed relationship. Most FPGAs have PLL resources that easily allow multiple clocks to be generated from a single source.

For instance, consider the case where a VGA input source at 60Hz frame rate is scaled-up to XGA at 60Hz. The VGA pixel clock is 25MHz and the output XGA pixel clock is 65MHz. A PLL may be used with the ratio 13/5 to generate the 65MHz pixel clock from the 25MHz clock source. In this way, both input and output pixel clocks are related and the input and output video will not become out of sync.

In addition, when using the scaler without a frame buffer, make sure that there is enough input buffering for a few lines of video (see FAQ 2). Input/output buffering is normally implemented using a FIFO arrangement similar to that shown in FAQ 6.

12) Can I change the scale parameters on-the-fly?

The scale parameters can be changed on a frame-by-frame basis if needed. The only requirement is that the user must assert the system reset after the parameters are changed. This allows the scaler to re-synchronize to the new parameters and lock to the next start of frame.

Generally it's convenient to change the parameters and toggle the reset during vertical blanking when there are no active pixels. This will ensure uninterrupted operation.

13) Is the scaler IP Core compatible with all FPGA and ASIC vendors? Can you give me timing and area figures for a specific technology?

The video scalers are provided as generic VHDL (or Verilog) RTL source code that is compatible with all major FPGA vendors and technologies. Specific timing and area figures can be provided on request.

14) Is it possible to use the scaler for simple greyscale operation? What about support for different pixel widths for interfacing with different cameras and sensors?

Our video scalers were designed for full 3-channel RGB operation such as RGB444 or RGB888 for example. However, single-channel operation is easily implemented by using only one of the available RGB channels; for example when implementing greyscale video applications.

In addition, the user should hard-wire the unused input channels to zero and also leave the unused output channels open. In this way, the unused channels will be optimized away during synthesis.

15) Is there an optimum choice of scaling factor for best image quality?

As per the datasheet, the scale factors should be specified as *pitch_x* and *pitch_y* where:

$$pitch=\left(\frac{input\ resolution}{output\ resolution}\right)*2^{12}$$

The calculated pitch for x and y should be rounded to the nearest whole number. However, there is one caveat. If the calculated pitch results in a number that is an exact power of 2, then the value should be adjusted up or down by a small margin in order to prevent the filter from 'ringing' at exact harmonics of 2^N.

For example, if the pitch is calculated as 1024 (scale down by exactly 4) then choose a number either side of 1024 instead. Prime numbers are good choices, so a value of 1021 or 1031 would be preferred values in this case for the best possible image quality.

16) Can I use two video scalers in parallel to accommodate higher pixel clock rates on lower-cost FPGAs; for example, in 4K UHD applications?

For video standards such as 4K (UHD) then pixel clock frequencies of 500MHz+ are generally required. Although such pixel rates are attainable using higher-end devices, it does become a limitation on lower-cost, lower-performance FPGAs.

One solution to this problem is to use two (or more) video scalers in parallel with each scaler processing pixels at half the pixel clock rate. The way to do this is to split each incoming video line into two and scale each half-line independently. After scaling, the separate halves of the image are then joined together again using a multiplexer at full pixel-clock rate.

One issue that arises from this approach is that there can be a discontinuity between the left and right parts of the scaled images when the two halves are joined back together. This is especially the case when scaling up by large factors; for instance when scaling from SD to UHD.

In order to alleviate this problem, the best approach is to scale a bit more than half the number of pixels in the left and right sections of the source image. The resulting left and right sections are then clipped and joined to form a clean transition down the middle of the scaled image.

In this example, an arbitrary image frame of 300 x 300 pixels is used as a source image. A scale factor of x 5.333 was chosen in the horizontal and vertical dimensions resulting in an output image of 1600 x 1600 pixels. Figure 4 describes the various steps involved in order to achieve a clean output image.

(If help is needed with the general circuit architecture, then please contact Zipcores for more information and we would be pleased to offer design assistance).

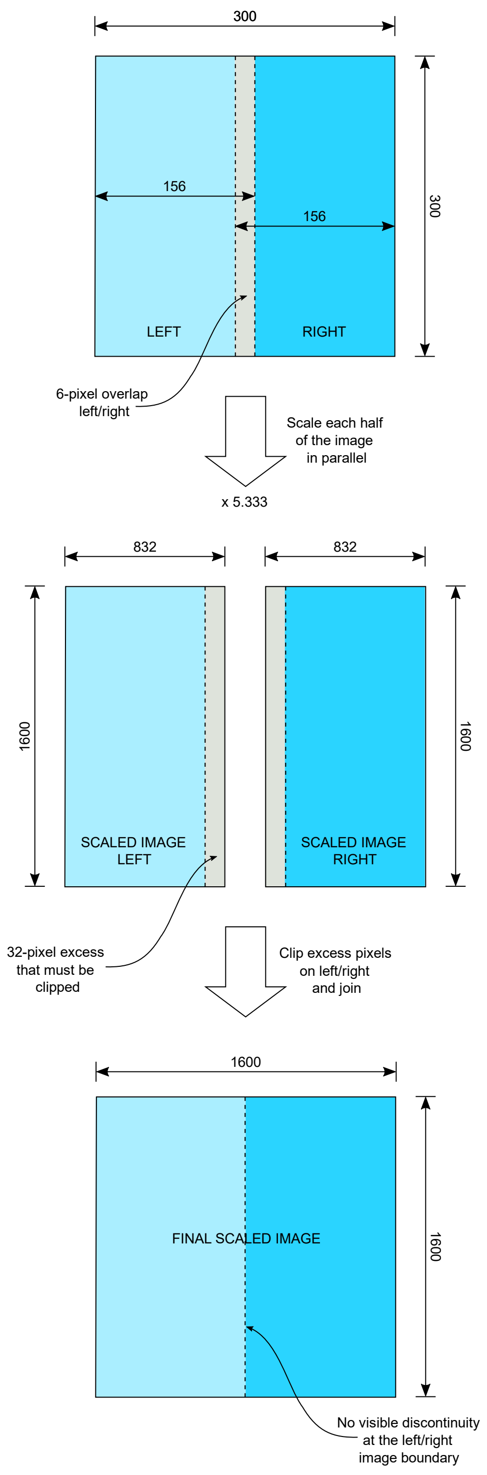
300
156
156
300
LEFT
RIGHT
6-pixel overlap left/right
Scale each half of the image in parallel
x 5.333
832
832
1600
1600
SCALED IMAGE LEFT
SCALED IMAGE RIGHT
32-pixel excess that must be clipped
Clip excess pixels on left/right and join
1600
FINAL SCALED IMAGE
1600
No visible discontinuity at the left/right image boundary

Figure 4: Split-image scaling example using 2 x scalers in parallel

Revision History

Revision	*Change description*	*Date*
1.0	Initial revision.	17/08/2009
1.1	Added FAQ 6,7.	22/01/2010
1.2	Added FAQ 8,9.	09/03/2010
1.3	Added FAQ 10. Updated logos and fixed typos.	23/08/2011
1.4	Added FAQ 11. Some minor corrections and additions to text.	21/01/2013
1.5	Added FAQ 12. Modified 11 in keeping with new code changes.	20/05/2013
1.6	Added more FAQs about AXI and Avalon-ST interface compatibility. Added note on greyscale operation.	13/10/2017
1.7	Updated FAQs 3 & 4. Added FAQ 15 regarding scale factors.	14/01/2019
1.8	Added FAQ 16.	05/03/2019